

CUADERNO DE TRABAJO PARA LA ASIGNATURA DE CIBERNÉTICA Y COMPUTACIÓN II



UNIDAD 3

Polimorfismo, constructores,
colaboración y herencia de
clases

UNIDAD 4

Interfaz gráfica de usuario.

AUTORES

Juan Gutiérrez Quiroz
Coordinador

Rogelio Argonza Moreno

María Del Socorro Ávila Nicolás

José Luis Hermoso Sandoval

Gabriela López Vargas

Guillermina Luna Santiago

Carmen Yadira Martínez Valdés

Alejandro Vela Bustamante

ÍNDICE

| | |
|--|----|
| ¿Cómo utilizar el Cuaderno de trabajo? | 6 |
| 3. Polimorfismo, constructores, colaboración y herencia de clases..... | 8 |
| 3.1 Polimorfismo y Constructores | 10 |
| 3.1.1 Introducción | 10 |
| Actividad 1..... | 11 |
| 3.1.2 Polimorfismo..... | 11 |
| Actividad 2..... | 12 |
| Actividad 3..... | 13 |
| 3.1.3 Constructor | 13 |
| Actividad 4..... | 16 |
| Actividad 5..... | 17 |
| Actividad 6..... | 17 |
| 3.1.4 Implementación de constructores con polimorfismo | 17 |
| Desafío 1 | 17 |
| 3.2 Colaboración de clases | 19 |
| 3.2.1 Introducción | 19 |
| Actividad 7..... | 20 |
| 3.2.2 Colaboración de clases..... | 20 |
| Actividad 8..... | 22 |
| Desafío 2 | 23 |
| Actividad 9..... | 24 |
| 3.3 Herencia..... | 25 |
| 3.3.1 Introducción | 25 |
| 3.3.2 Superclases y subclases..... | 26 |
| Actividad 10 | 27 |
| Actividad 11 | 28 |
| Actividad 12 | 29 |
| 3.3.3 Sintaxis y palabras reservadas super y this | 30 |
| Actividad 13 | 32 |
| Actividad 14 | 32 |
| Actividad 15 | 33 |
| Actividad 16 | 35 |
| Actividad 17 | 36 |

| | |
|--|-----|
| Actividad 18 | 37 |
| 3.3.4 Ventajas de la herencia..... | 38 |
| Actividad 19 | 39 |
| Desafío 3 | 39 |
| 3.4 Soluciones | 40 |
| Actividades | 40 |
| Desafíos..... | 48 |
| 3.5 Referencias | 57 |
| 4 Interfaz gráfica de usuario..... | 59 |
| 4.1 Clase Swing | 61 |
| 4.1.1 Introducción | 62 |
| Actividad 1..... | 63 |
| 4.1.2 Clase Swing con NetBeans..... | 63 |
| Actividad 2..... | 63 |
| 4.1.3 Entorno de trabajo de NetBeans..... | 67 |
| Actividad 3..... | 69 |
| Actividad 4..... | 70 |
| 4.1.4 JFrame – Formulario | 71 |
| Actividad 5..... | 71 |
| 4.1.5 JLabel - Etiqueta | 76 |
| Actividad 6..... | 76 |
| 4.1.6 JButton - Botón | 80 |
| Actividad 7..... | 81 |
| Desafío 1 | 84 |
| 4.1.7 Iniciando un proyecto..... | 86 |
| Actividad 8..... | 86 |
| 4.1.8 Elaborando la plantilla en NetBeans..... | 88 |
| Actividad 9..... | 88 |
| Actividad 10 | 96 |
| Actividad 11 | 97 |
| 4.1.9 JTextArea – Área de texto..... | 98 |
| Actividad 12 | 98 |
| 4.1.10 Campo de Texto | 101 |
| Actividad 13 | 102 |

| | |
|---|-----|
| Actividad 14 | 107 |
| 4.1.11 Casilla de activación | 108 |
| Actividad 15 | 108 |
| Actividad 16 | 114 |
| 4.1.12 Botón de Opción..... | 114 |
| Actividad 17 | 114 |
| 4.1.13 Combo Box..... | 119 |
| Actividad 18 | 119 |
| Actividad 19 | 122 |
| 4.1.14 Barra de Menú..... | 123 |
| Actividad 20 | 123 |
| 4.1.15 Menú..... | 127 |
| Actividad 21 | 127 |
| Actividad 22 | 129 |
| 4.1.16 JMenuItem - Elemento de Menú | 130 |
| Actividad 23 | 131 |
| Actividad 24 | 134 |
| Actividad 25 | 142 |
| 4.2 Clase Graphics..... | 144 |
| 4.2.1 Introducción | 144 |
| 4.2.2 Clase Graphics..... | 145 |
| Actividad 26 | 146 |
| 4.2.3 Método setColor | 148 |
| 4.2.4 Método drawLine..... | 148 |
| 4.2.5 Método drawRect | 149 |
| 4.2.6 Método drawRoundRect..... | 150 |
| 4.2.7 Método drawOval | 151 |
| 4.2.8 Método drawPolygon..... | 152 |
| Actividad 27 | 153 |
| Desafío 2 | 156 |
| 4.2.9 Método fillRect..... | 158 |
| 4.2.10 Método fillRoundRect..... | 158 |
| 4.2.11 Método fillOval | 159 |
| 4.2.12 Método fillPolygon..... | 160 |

Actividad 28161

Actividad 29162

Desafío 3163

4.3 Soluciones164

Actividades164

Desafíos.....179

4.4 Referencias185

¿Cómo utilizar el Cuaderno de trabajo?

El contenido de este Cuaderno de trabajo está dirigido a alumnos que cursan la asignatura de Cibernética y Computación II, con el propósito de ofrecer un material de apoyo para reforzar los conceptos vistos y lograr los aprendizajes planteados en el programa de la asignatura.

El cuaderno de trabajo es un conjunto estructurado de técnicas y procedimientos para realizar actividades tanto teóricas como prácticas sobre una o varias unidades del programa, necesarios para la enseñanza aprendizaje de la asignatura de Cibernética y Computación II, los contenidos están adecuados al nivel y profundidad de temática especificada en el programa de estudios de esta asignatura.

Este material se puede utilizar en la modalidad de Curso taller, ya que en cada unidad temática contiene conceptos, actividades que constan de ejemplos desarrollados, ejercicios y desafíos para los diferentes temas. Para su desarrollo se contemplaron los propósitos de cada una de las unidades abordadas, los aprendizajes a alcanzar por el alumno y la temática que conforma dicho programa.

Consta de dos unidades y está organizado de la siguiente forma:

En cada una de las unidades:

- Se explican los conceptos correspondientes a la temática.
- Se proponen actividades para visualizar y reflexionar los conceptos abordados.
- Se proponen actividades de evaluación para que el alumno estime el aprendizaje obtenido.
- Se proponen desafíos para que el alumno aplique los conocimientos adquiridos.

Al final de cada una de las unidades se incluyen las respuestas de los ejercicios y desafíos y al final del cuaderno de trabajo se anexa la bibliografía y fuentes de información utilizadas para su desarrollo.

Se puede utilizar en clase después de revisar los conceptos y ejemplos en el desarrollo de la solución a los ejercicios, esto le permitirá observar al profesor las dificultades o dudas de los alumnos proporcionando un asesoramiento adecuado con sugerencias y orientaciones para obtener el resultado pertinente. También es recomendable que el profesor explique alguna de las soluciones obtenidas para una mejor comprensión del tema.

Para lograr un mejor proceso de enseñanza - aprendizaje, se recomienda que:

El alumno:

- Revise los conceptos y ejemplos previamente a la clase
- Desarrolle las actividades de evaluación
- Compare con sus compañeros los resultados obtenidos
- Plantee dudas en clase.

El profesor:

- Exponga el tema por revisar.
- Solucione las dudas de los alumnos.
- Oriente en la solución de los ejercicios.

Con lo anterior se pretende lograr los aprendizajes indicados.

Este cuaderno de trabajo aborda las unidades 3 y 4 de Cibernética y Computación II y tiene como objetivos primordiales:

Que los profesores:

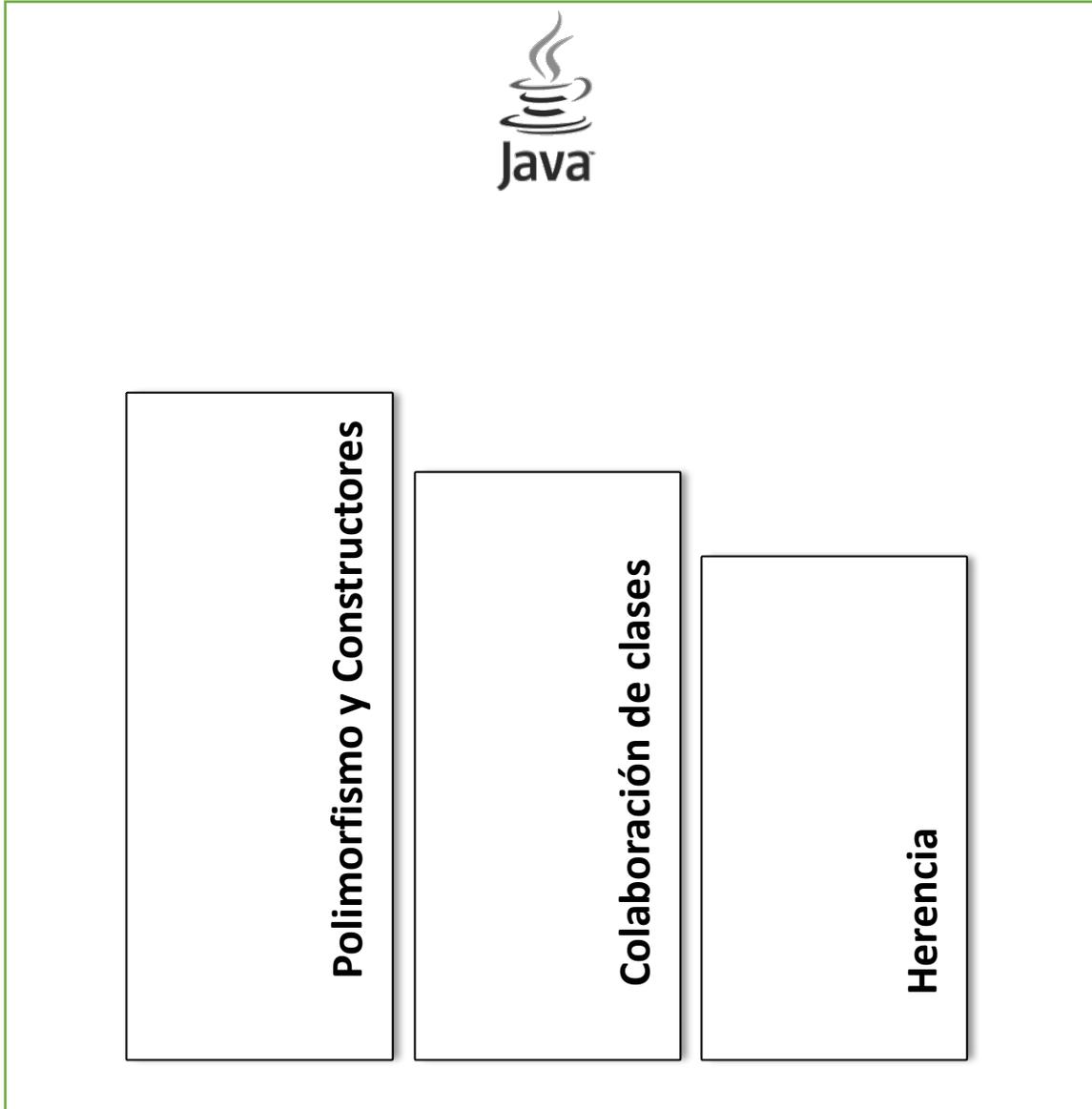
- Cuenten con un material adecuado para preparar su clase y reforzar los conceptos vistos.

Que los alumnos:

- Comprendan y apliquen los conocimientos adquiridos en la asignatura.
- Comprendan la influencia de la programación en el desarrollo de la ciencia.
- Adquieran una metodología para resolver problemas utilizando la computadora.
- Sean capaces de desarrollar programas utilizando la interfaz gráfica del lenguaje de programación Java.

UNIDAD 3

3. Polimorfismo, constructores, colaboración y herencia de clases



Unidad 3 Polimorfismo, constructores, colaboración y herencia de clases

TIEMPO: 10 HORAS

Propósito de la unidad:

Al finalizar la unidad el alumno: Implementará programas en Java utilizando polimorfismo, constructores, colaboración y herencia de Clases para aprovechar las bondades de la programación orientada a objetos.

Aprendizajes.

El alumno:

- ✓ Conoce el concepto de polimorfismo y constructor.
- ✓ Desarrolla programas que involucren polimorfismo y constructores.
- ✓ Comprende la colaboración de Clases para la resolución de problemas.
- ✓ Desarrolla programas que involucren la colaboración de Clases.
- ✓ Comprende el concepto de herencia en la resolución de un problema.
- ✓ Desarrolla programas que involucren la herencia de Clases.

3.1 Polimorfismo y Constructores

Aprendizajes esperados.

El alumno:

- ✓ Conoce el concepto de polimorfismo y constructor.
- ✓ Desarrolla programas que involucren polimorfismo y constructores.

Contenido temático.

- Concepto de polimorfismo.
- Concepto de constructor.
- Implementación de constructores con polimorfismo.

Objetivo de la actividad:

- Que el alumno elabore programas utilizando el polimorfismo y constructores.

3.1.1 Introducción

La palabra polimorfismo viene del griego «poly» que significa mucho, abundancia y pluralidad, y del sufijo «morfo» que quiere decir forma. Esto es que tiene o puede tener varias formas.

En la figura 3.1.1 es fácil darse cuenta de que los tres autos son de la misma marca; sin embargo, sus colores y modelo varía lo cual los hace diferentes uno de otro, eso es polimorfismo.



fig. 3.1.1. Ejemplo de polimorfismo.

Un constructor es un método que se ejecuta cada vez que se crea un objeto. En la figura 3.1.2. el sello es la clase y el cojín de pintura es el constructor porque crea diferentes objetos.

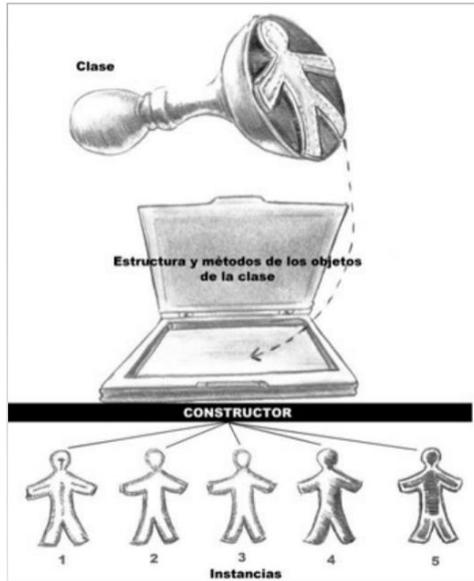


fig. 3.1.2. Clase, constructor e instancias.

Actividad 1

Conceptos de polimorfismo y constructores

Responde las siguientes preguntas.

1.1. ¿Qué significa polimorfismo?

1.2. Menciona un ejemplo de polimorfismo.

1.3. ¿Para qué se utiliza el método constructor en Java?

3.1.2 Polimorfismo

Consiste en utilizar una misma expresión para invocar a diferentes versiones de un mismo método. Dicho de otra manera, es hacer que un mismo método haga diferentes cosas dependiendo del objeto donde se ejecute.

Ejemplo:

Realizar un programa que calcule la suma de números enteros y números reales, aplicando el polimorfismo.

Para ello se va a utilizar una clase llamada **Calculadora** donde se declaran dos métodos, ambos llamados suma, cuya diferencia entre ellos son los parámetros. Estos métodos son polimórficos porque se llaman igual, pero tienen diferentes parámetros. A continuación, se muestra el código correspondiente:

```
public class Calculadora
{
    //Método suma con parámetros enteros
    public int suma(int a, int b)
    {
        return(a+b);
    }
    //Método suma con parámetros reales
    public double suma(double a, double b)
    {
        return(a+b);
    }
}
```

En el método principal se instancia un objeto de la clase y se llama a los métodos distinguiendo los argumentos, enteros o reales y se imprimen en pantalla. A continuación, se muestra el código del método principal:

```
public static void main(String[] args)
{
    //Se instancia al objeto calc
    Calculadora calc=new Calculadora();
    //Se llama al método suma con argumentos enteros
    int suma1=calc.suma(9,6);
    //Se llama al método suma con argumentos reales
    double suma2=calc.suma(9.32,6.98);
    //Se imprimen las variables con los resultados
    System.out.println("SUMA CON MÉTODOS POLIMÓRFICOS");
    System.out.println("Suma de números enteros:"+suma1);
    System.out.println("Suma de números reales:"+suma2);
}
```

Actividad 2

Codificando el programa de polimorfismo

Utilizando el *IDE BlueJ*, (si tienes dudas sobre la instalación o uso de BlueJ consulta el

Paquete Didáctico¹), escribe el programa del ejemplo anterior y ejecútalo, deberás obtener la salida de la figura 3.1.3.

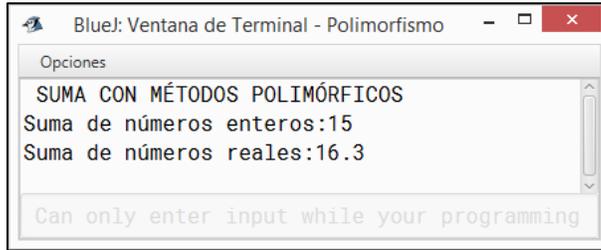


fig. 3.1.3. Ejecución del programa de polimorfismo.

Actividad 3

Codificando métodos polimórficos

Escribe el código del método para realizar una resta de enteros y otra de reales utilizando polimorfismo para que se llamen igual, pero tengan diferentes parámetros.

| Método para restar números enteros | Método para restar números reales |
|--|--|
| <pre>public ____ ____ (____) { ____; }</pre> | <pre>public ____ ____ (____) { ____; }</pre> |

3.1.3 Constructor

Es un tipo especial de método que se invoca al instanciar un objeto, el cual crea y a veces inicializa al objeto creado. Los constructores se llaman igual que la Clase, pueden tener o no parámetros, no regresan resultados, se deben declarar como *public* y su función consiste en reservar la memoria necesaria además de inicializar las variables que forman parte de los atributos de la clase con valores válidos.

Existen tres tipos de constructores en Java:

- Constructor por default. En este caso, si no se especifica ningún constructor, Java genera automáticamente uno por defecto.
- Constructor sin parámetros. Este tipo de constructor se define al escribirse en el código del programa con el mismo nombre de la clase y los atributos se pueden inicializar.
- Constructor con parámetros. Es aquel en el cual se envían determinados valores para asignarlos a los atributos de la clase mediante el uso de parámetros.

¹ Ávila, S. et. al. (2019). Paquete didáctico para la asignatura de Cibernética y Computación I. México: ENCCH Plantel Oriente, UNAM. p. 195 - 212.

A continuación, se muestra la sintaxis de los métodos constructores:

Constructor con parámetros

```
public NombreClase(tipoDato parametro)
{
    this.atributo = parametro;
}
```

Constructor sin parámetros

```
public NombreClase()
{
    atributo = valor;
}
```

Un constructor se emplea cada que se instancia un objeto de la clase, ya sea con el que no pide parámetros o con el que si los pide:

```
NombreClase objeto1=new NombreClase();
NombreClase objeto2=new NombreClase(tipoDato parametro);
```

Los métodos constructores son polimórficos porque tienen el mismo nombre y lo que define a qué constructor se ingresa, son los argumentos que se envían como parámetros.

Ejemplo:

Realiza un programa que calcule sumas de:

- 1) Dos números enteros.
- 2) Dos números reales.
- 3) Dos números enteros utilizando el método constructor.

Solución.

1. En la misma clase *Calculadora*, del ejemplo anterior, se declaran dos atributos de tipo entero que serán los parámetros del constructor para hacer la operación suma.

```
public class Calculadora
{
    private int a;
    private int b;
```

2. Se agrega el constructor que no pide parámetros e inicializa los atributos.

```
public Calculadora()
{
    a=0;
    b=0;
}
```

3. Se escribe el constructor que pide parámetros y modifica los atributos.

```
public Calculadora(int a, int b)
{
    this.a=a;
    this.b=b;
}
```

4. Se declara un método que no pida parámetros y haga la suma del valor de los atributos.

```
public int suma()
{
    return(this.a+this.b);
}
```

Consideremos el método principal que se utilizó en el ejemplo anterior en los métodos polimórficos.

```
public static void main(String[] args)
{
    //Se instancia al objeto calc
    Calculadora calc=new Calculadora();
    //Se llama al método suma con argumentos enteros
    int suma1=calc.suma(9,6);
    //Se llama al método suma con argumentos reales
    double suma2=calc.suma(9.32,6.98);
    //Se imprimen las variables con los resultados
    System.out.println("SUMA CON MÉTODOS POLIMÓRFICOS");
    System.out.println("Suma de números enteros:"+suma1);
    System.out.println("Suma de números reales:"+suma2);
}
```

5. Se agrega al final, antes de la llave que cierra el método principal anterior, una instancia de la clase con el constructor, después se llama al método suma, ambos no piden parámetros, y se imprime un mensaje con el resultado.

```
//Constructor sin parámetros
Calculadora calc1=new Calculadora();
//Llamada al método suma
int suma3=calc1.suma();
System.out.println("\n\nCONSTRUCTOR SIN PARÁMETROS");
System.out.println("Suma de números enteros:"+suma3);
```

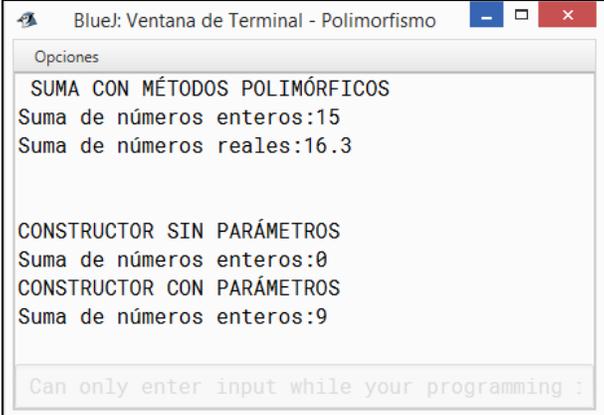
6. A continuación se instancia un objeto con el constructor que pide parámetros y se llama al método suma mediante el siguiente código.

```
//Constructor con parámetros
Calculadora calc2=new Calculadora(5,4);
//Llamada al método suma
int suma4=calc2.suma();
//Se imprime la variable que contiene el resultado
System.out.println("CONSTRUCTOR CON PARÁMETROS");
System.out.println("Suma de números enteros:"+suma4);
```

Actividad 4

Codificando constructores

Utilizando el *IDE BlueJ*, escribe el programa completo del ejemplo anterior y ejecútalo, deberás obtener la salida de la figura 3.1.4.



```
BlueJ: Ventana de Terminal - Polimorfismo
Opciones
SUMA CON MÉTODOS POLIMÓRFICOS
Suma de números enteros:15
Suma de números reales:16.3

CONSTRUCTOR SIN PARÁMETROS
Suma de números enteros:0
CONSTRUCTOR CON PARÁMETROS
Suma de números enteros:9

Can only enter input while your programming :
```

fig. 3.1.4. Ejecución del programa de constructor.

Actividad 5

Reflexión del programa de constructores

Responde las siguientes preguntas.

5.1. ¿Por qué la suma que utiliza el constructor sin parámetros es igual a cero?

5.2. ¿Por qué la suma que utiliza el constructor con parámetros es igual a nueve?

Actividad 6

Codificar métodos constructores

Completa el código para desarrollar un método constructor de la clase **Calculadora** que pida tres parámetros y el método suma que calcule la operación de los tres atributos. Después ejecuta el programa.

| Constructor con tres parámetros | Método para sumar los tres atributos |
|---|---|
| <pre>public Calculadora(int a, int b, _____) { this.a=a; this.b=b; _____; }</pre> | <pre>public int suma() { return(_____); }</pre> |

3.1.4 Implementación de constructores con polimorfismo



Con la resolución de un problema se aplicarán los conceptos vistos anteriormente.

Desafío 1

Alfombras

Resuelve el siguiente problema diseñando un programa que conste de una clase llamada **Rectangulo** con los atributos, métodos y objetos necesarios.

Se tiene un salón en forma rectangular de 7 metros de largo por 6.5 metros de ancho. Se

cuenta con dos alfombras también rectangulares, que se colocarán sobre el piso de dicho salón. La alfombra 1 mide 3.8 metros X 4.6 metros y la alfombra 2, 4.5 metros X 2.3 metros. Se desea saber qué parte del piso quedará cubierta y qué parte no, para ayudar a decidir si se compran o no otras alfombras.

Planteamiento.

Para resolver el problema realiza las siguientes actividades.

- a. Dibuja el rectángulo del salón y en su interior las dos alfombras:
- b. Calcula el área de las tres figuras:

Alfombra1= _____

Alfombra2= _____

Salón = _____

- c. Realiza las operaciones necesarias para resolver el problema:

Área cubierta= _____

Área descubierta= _____

- d. Describe el resultado: _____

- e. Escribe el código del programa para lograr la salida que se muestra en la figura 3.1.5. Compara tus resultados con los obtenidos en la pantalla.

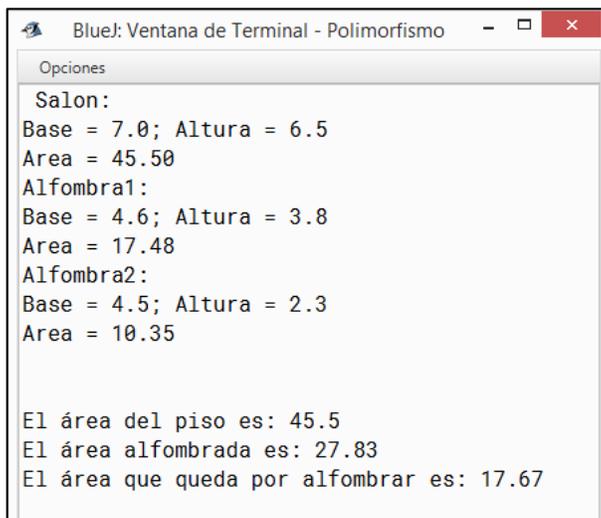


fig. 3.1.5. Ejecución del programa de las alfombras.

3.2 Colaboración de clases

Aprendizajes esperados

El alumno:

- ✓ Comprende la colaboración de Clases para la resolución de problemas.
- ✓ Desarrolla programas que involucren la colaboración de Clases.

Contenido temático

- Interacción y comunicación entre Clases.

Objetivo de la actividad:

- Que el alumno elabore programas utilizando la colaboración de clases.

3.2.1 Introducción

Una clase es una plantilla que define la forma de un objeto. Especifica los datos y el código que operará en estos. Java usa una especificación de clase para construir objetos. Los objetos son instancias de una clase. Por lo tanto, una clase es esencialmente un conjunto de elementos que especifican cómo construir un objeto.

Los atributos, también llamados datos o variables son porciones de información que un objeto posee o conoce de sí mismo. Una clase puede tener cualquier número de atributos o no tener ninguno. Se declaran con un identificador y el tipo de dato correspondiente. Además, los atributos tienen asociado un modificador que define su visibilidad según se muestra en la siguiente tabla.

| Modificador | Visibilidad |
|-------------|-----------------------------|
| public | Pública (+) |
| protectec | Protegida / en herencia (#) |
| private | Privada (-) |
| package | De paquete (~) |

Tabla 3.2.1.

Normalmente la Programación Orientada a Objetos no utiliza una sola clase, sino que hay muchas clases que interactúan y se comunican, por ejemplo, las clases Scanner y Math. Una de ellas es llamada principal pues contiene el método principal y es la que va a instanciar objetos de las otras clases. La colaboración de clases es la interacción entre ellas a través de la creación de objetos y llamadas a los métodos.

Actividad 7

Conceptos de colaboración de clases

Responde con tus propias palabras las siguientes preguntas:

7.1. ¿Qué es una clase en Java?

7.2. ¿Qué es un atributo?

7.3. ¿Qué es la colaboración de clases?

3.2.2 Colaboración de clases

Cuando se da la acción de instanciar un objeto de una clase desde otra clase, se genera la colaboración de clases. Esta operación también permite que una clase se comunique con otra con el propósito de utilizar los métodos de esta última para mejorar algún servicio o mantener una relación lógica de dependencia.

Ejemplo:

Elabora un programa que calcule el promedio de dos números, utilizando la colaboración de clases.

Planteamiento.

Se desarrolla en primer lugar una clase llamada “*Promedio*”, que contenga un método “promedio”, cuya función será calcular el promedio de dos números, se implementará otra clase “*Principal*” que contenga el método principal y que invoque al método *promedio* instanciando un objeto *calc*.

A continuación, se muestra el código de la clase *Promedio*, el método recibe los parámetros *a*, *b* y regresa el valor del promedio en la variable *resultado*.

```
public class Promedio
{
    //Método promedio para calcular el promedio
    public double promedio(double a, double b)
    {
        double resultado=(a+b)/2;
        return resultado;
    }
}
```

Ahora se muestra el código de la Clase *Principal*, esta llama al método *promedio* contenido en la clase *Promedio*.

```
import java.util.Scanner;
//Clase principal
public class Principal
{
    //Método principal
    public static void main(String[] args){
        //Instanciación de objeto
        Scanner teclado=new Scanner(System.in);
        //Se instancia al objeto calc con la clase Promedio
        // En esta línea de código es donde se implementa la colaboración de clases
        Promedio calc=new Promedio();
        //Limpia la pantalla
        System.out.println("\u000c");
        //Declaración de variables
        double a=0;
        double b=0;
        //Lee los valores de a y b
        System.out.print("Dame el valor a: ");
        a=teclado.nextDouble();
        System.out.print("Dame el valor b: ");
        b=teclado.nextDouble();
        //Trae el resultado del promedio llamando al método promedio
        System.out.println("El promedio es: "+ String.format("%.2f",calc.promedio(a,b)));
    }
}
```

Ejecución del programa.

Primero se compila el programa y BlueJ muestra una pantalla (Figura 3.2.1.) en la que se visualiza la colaboración de clases; por ejemplo, se ve que la clase *Principal* hace uso de la clase *Promedio* por una flecha que sale de ella.

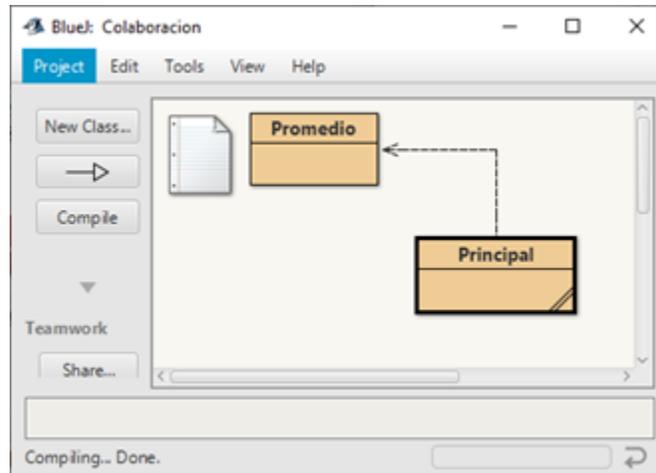


fig. 3.2.1. Colaboración de clases.

Actividad 8

Codificando un programa de colaboración de clases.

Utilizando el IDE BlueJ escribe un programa que calcule el promedio de tres valores reales y obtén una salida similar a la de la figura 3.2.2.

```

Opciones

Dame el valor a:8.5
Dame el valor b:9.1
Dame el valor c:7.4
El promedio es: 7.95

Can only enter input while your programming is
    
```

fig. 3.2.2. Ejecución del programa de colaboración de clases.

Diagrama de clases.

La colaboración entre clases se puede trabajar de forma gráfica mediante el uso de diagramas de clase. Para elaborarlos, existe un lenguaje gráfico, conocido como Lenguaje Unificado de Modelado (UML por sus siglas en inglés, Unified Modeling Language), es un esquema que emplea diagramas donde cada clase se modela en forma de rectángulo que contiene tres partes: el nombre de la clase se ubica al inicio, los atributos en la parte de en medio y las operaciones o métodos al final.

El diagrama de clases del ejemplo anterior se muestra en la figura 3.2.3.:

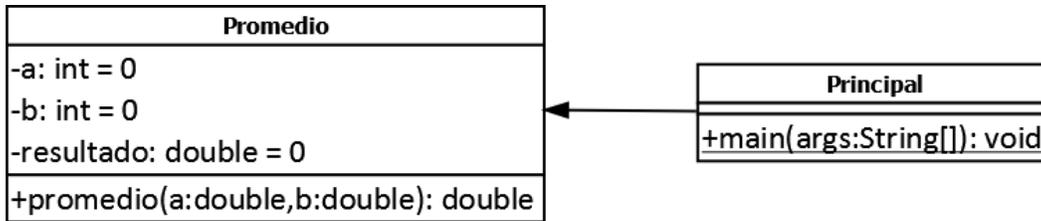


Fig. 3.2.3. Diagrama de clases con colaboración.

En seguida se presentan las características del esquema anterior:

- El nombre de la clase inicia con mayúscula.
- El tipo de variable aparece a la derecha del atributo.
- Se emplea el signo – para indicar que es de tipo private.
- Se emplea el signo + para indicar que es de tipo public.
- Se especifica el tipo de valor que devuelve el método después de dos puntos.
- Se subraya el método principal porque es estático.

Desafío 2

Áreas

Elaborar un programa que calcule el área de un rectángulo, un triángulo y un círculo, utilizando la colaboración de clases. El programa debe pedir los datos de cada figura, calcular el área y mostrar el resultado.

Planteamiento.

Se desarrolla primero una clase para cada figura, con sus atributos y sus métodos *pedirDatos* y *calcularArea*, se escribe otra clase llamada *Areas* que contenga el método principal y que instancie un objeto de cada clase e invoque los métodos. Observa el siguiente diagrama de clases de la fig 3.2.4. para escribir el programa:

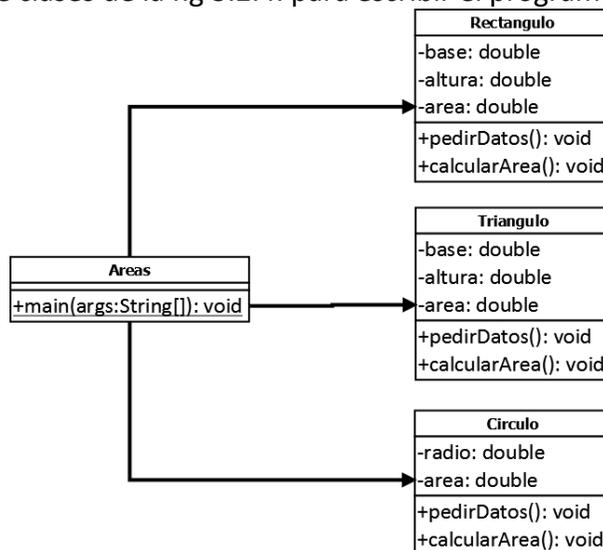


fig. 3.2.4. Diagrama de clases de la clase *Areas*.

Escribe el código del programa para lograr una salida similar a la que se muestra a continuación (figura 3.2.5.).

```

BlueJ: Ventana de Terminal - Colaboracion
Opciones

CÁLCULO DE ÁREAS
CON COLABORACIÓN DE CLASES

DATOS DEL RECTÁNGULO

Valor de la base:5
Valor de la altura:6
El área del rectángulo es: 30.00

DATOS DEL TRIÁNGULO

Valor de la base:5
Valor de la altura:6
El área del triángulo es: 15.00

DATOS DEL CÍRCULO

Valor del radio:5
El área del círculo es: 78.54

Can only enter input while your programming
    
```

Fig. 3.2.5. Ejecución del programa de áreas.

Actividad 9

Reflexión del diagrama de clases

Responde las siguientes preguntas con base en el diagrama mostrado en la figura 3.9.

9.1. ¿Qué modificador de acceso deben tener los atributos de la clase *Rectangulo*?

9.2. ¿Qué modificador de acceso deben tener los métodos de la clase *Triangulo*?

9.3. ¿Cuál es el tipo de dato que devuelven los métodos de la clase *Circulo* y por qué?

3.3 Herencia

Aprendizajes esperados

El alumno:

- ✓ Comprende el concepto de herencia en la resolución de un problema.
- ✓ Desarrolla programas que involucren la herencia de Clases.

Contenido temático

- ✓ Herencia
 - Superclase.
 - Subclase.
 - Ventajas.
- ✓ Implementación de la herencia de Clases.

Objetivo de la actividad:

- ✓ Que el alumno comprenda la herencia entre clases y sus ventajas.

3.3.1 Introducción

La herencia es una característica de la Programación Orientada a Objeto que permite crear nuevas clases a partir de clases ya existentes, logrando reutilizar código y jerarquización de clases. Asimismo, ayuda al mantenimiento y favorece realizar extensiones de las aplicaciones.

Por ejemplo, tenemos la clase Persona.

```
public class Persona
{
}
}
```

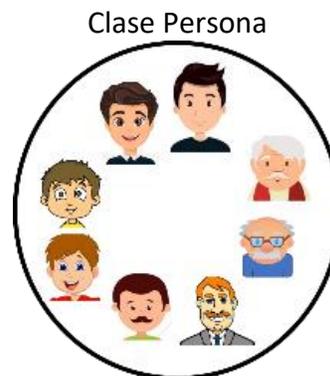


fig. 3.3.1. Personas

Como podemos observar en la figura dentro de esta clase existen diferentes tipos de personas con características generales comunes, asimismo, cada uno de ellos cuenta con sus propias características, en esta clase se observa que, entre el conjunto de personas

encontramos niños, jóvenes, adultos y Ancianos.

Por lo anterior, se pueden crear las clases Niño, Joven, Adulto y Anciano que se derivan de la clase Persona.

Esto es:

El niño *es una* persona

El joven *es una* persona

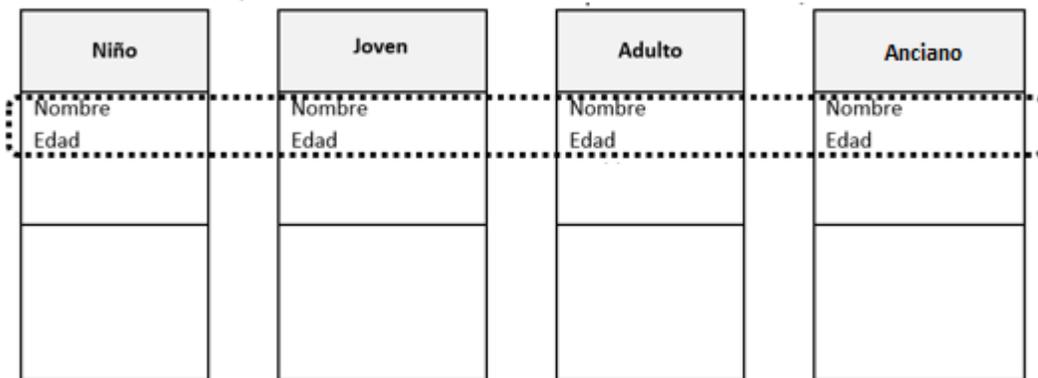
El adulto *es una* persona

El Anciano *es una* persona

Asimismo, la herencia se ve reflejada mediante el proceso en el cual una clase adquiere las propiedades (atributos) y comportamiento (métodos) de otra, haciendo también posible añadir nuevos elementos (atributos o métodos) o redefinir los elementos existentes.

Ejemplo:

Revisaremos los atributos que puede tener cada clase del ejemplo anterior.



Como se observa en la figura anterior una Persona en las diferentes etapas de su vida, tiene características comunes. Los atributos que son comunes en las etapas de vida serán los atributos de la clase Persona.

3.3.2 Superclases y subclases

La herencia conduce a una estructura jerárquica de clases o estructura de árbol, lo cual significa que en la Programación Orientada a Objetos todas las relaciones entre clases deben ajustarse a dicha estructura, para identificar la superclase y las subclases.

La siguiente imagen muestra el orden jerárquico de las clases:

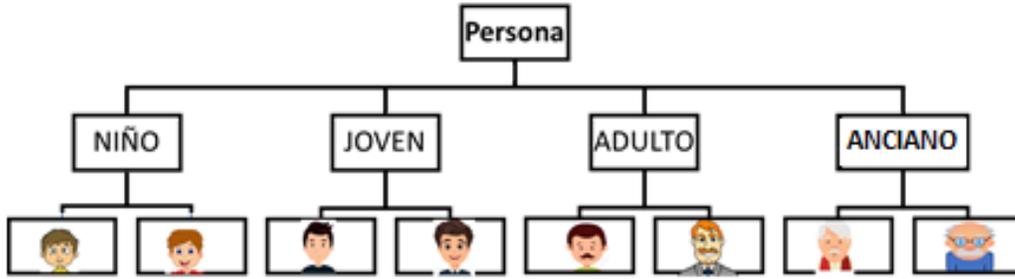


fig. 3.3.2. Organigrama de Personas

Se puede apreciar la superclase, también llamada clase padre o clase base *Persona* y las subclases, clases hijas o clases derivadas como *Niño*, *Joven*, *Adulto* y *Anciano*.

Una superclase puede tener cualquier número de subclases. Cuentan con sus propios métodos y atributos que pueden servir de base para la creación de otras subclases. Por ejemplo, la clase *Persona* servirá de base para crear a las subclases *Niño*, *Joven*, *Adulto* y *Anciano*.

Una subclase tiene solo una superclase, teniendo acceso a todos los atributos y métodos *public* y *protected* de la superclase, no a los *private*. Asimismo, puede contar con atributos o métodos propios.

Para heredar los atributos y métodos se utiliza el identificador de acceso *protected*.

Actividad 10

Conceptos

Contesta las siguientes preguntas:

10.1 ¿Qué es herencia?

10.2 ¿Qué es una superclase?

10.3 ¿Cómo se le puede llamar también a una superclase?

10.4 ¿Qué es una clase hija?

Actividad 11

Desarrollando el diagrama de clases

Llena las tablas con sus características y acciones que tiene una persona en las diferentes etapas de su vida (niño, joven, adulto y Anciano).

| Niño | Joven | Adulto | Anciano |
|------------------------|------------------------|------------------------|------------------------|
| características | características | características | características |
| <u>nombre</u> | <u>nombre</u> | <u>nombre</u> | <u>nombre</u> |
| <u>edad</u> | <u>edad</u> | <u>edad</u> | <u>edad</u> |
| _____ | _____ | _____ | _____ |
| _____ | _____ | _____ | _____ |
| _____ | _____ | _____ | _____ |
| acciones | acciones | acciones | acciones |
| <u>caminar</u> | <u>caminar</u> | <u>caminar</u> | <u>caminar</u> |
| <u>comer</u> | <u>comer</u> | <u>comer</u> | <u>comer</u> |
| _____ | _____ | _____ | _____ |
| _____ | _____ | _____ | _____ |

Ahora, vamos a obtener las características y acciones comunes de las diferentes etapas para obtener los atributos y métodos de la superclase.

Por ejemplo:

| Niño | Joven | Adulto | Anciano |
|---------|--------------------|------------|-----------------------|
| nombre | nombre | nombre | nombre |
| edad | edad | edad | edad |
| escuela | carrera | sueldo | monto de pensión |
| grado | promedio | hijos | grupo de tercera edad |
| | | | núm. de nietos |
| caminar | caminar | caminar | caminar |
| comer | comer | comer | comer |
| dormir | dormir | dormir | dormir |
| bañarse | bañarse | bañarse | bañarse |
| jugar | ir al cine con su | trabajar | asistir a su grupo |
| saltar | novi@ | pagar | ir al médico |
| correr | terminar carrera | proteger | visitar a sus nietos |
| | divertirse con sus | vacacionar | |
| | amigos | | |

De lo anterior se deduce que la superclase es *Persona* porque:

- El niño *es una* persona
- El joven *es una* persona
- El adulto *es una* persona
- El Anciano *es una* persona

Sus atributos comunes son *nombre* y *edad*, sus métodos comunes son *caminar*, *comer*, *dormir*, y *bañarse*, por lo tanto, estos serán los correspondientes a la superclase, para que después los pueda heredar.

Actividad 12

Identificando la superclase

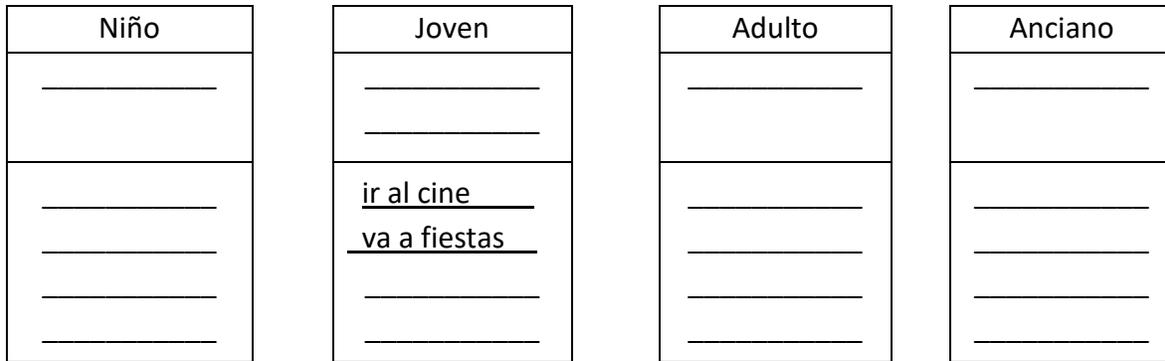
De acuerdo con tu actividad No. 11, responde las siguientes actividades.

12.1. Coloca en la tabla las características y acciones comunes en cada etapa de vida de una Persona.

| Persona |
|------------------------|
| características |
| <u>Nombre</u> |
| <u>Edad</u> |
| _____ |
| _____ |
| _____ |
| acciones |
| <u>Caminar</u> |
| <u>comer</u> |
| _____ |
| _____ |
| _____ |

12.2. Agrega en la columna correspondiente las características y acciones de cada etapa de vida.

| Niño | Joven | Adulto | Anciano |
|-------|-----------------|--------|---------|
| _____ | <u>carrera</u> | _____ | _____ |
| _____ | <u>promedio</u> | _____ | _____ |
| _____ | _____ | _____ | _____ |



12.3. Construye el diagrama de clases. Recuerda agregar el tipo de variable y el modificador de acceso en la superclase.

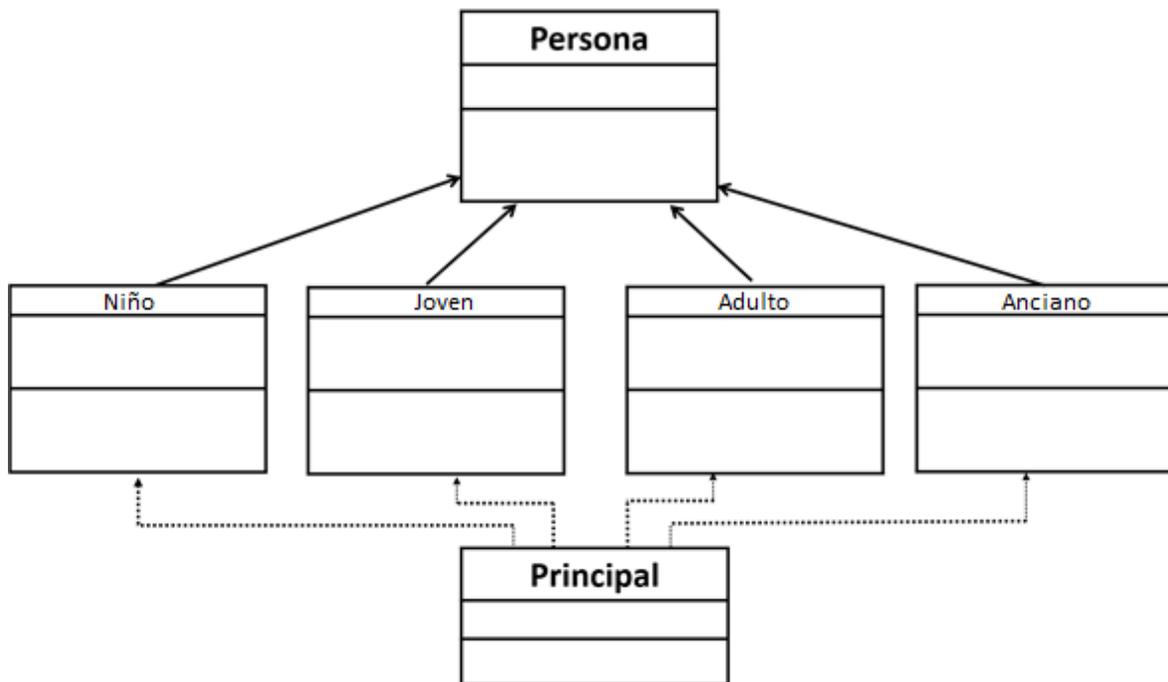


fig. 3.3.3. Diagrama de Personas

3.3.3 Sintaxis y palabras reservadas super y this

Sintaxis

La herencia se expresa mediante la palabra `extends`. Por ejemplo, para declarar la clase B que hereda de una clase A, se expresa:

```
public class B extends A
{
....
}
```

Como se puede ver en la sintaxis anterior a las clases derivadas o subclases se les agrega la palabra reservada *extends* y se indica el nombre de la superclase.

Ejemplo.

Si tenemos a la superclase Persona y se crean las subclases Joven y Anciano con sus respectivos atributos como se puede visualizar a continuación:

```
public class Persona{
    protected String nombre;
    protected int edad;
}
```

```
public class Joven extends Persona
{
    private String carrera;
    private double promedio;
}
```

```
public class Anciano extends Persona
{
    private double montoPension;
    private String gpoTerceraEdad;
    private int nietos;
}
```

Palabras reservadas *super* y *this*

Como se ha mencionado la herencia también permite que las subclases adquieran los métodos de la superclase.

En el constructor de la subclase la palabra reservada *super* se utiliza para hacer referencia al constructor de la clase padre. Se debe tener en cuenta que los constructores no se heredan.

Por ejemplo:

Para crear el constructor de la clase Joven, vamos a hacer referencia a los atributos nombre y edad, los cuales estamos heredando de la clase Persona, para ello, hacemos uso de la palabra reservada *super*.

```
public Joven(String nombre, int edad) {
    super(nombre, edad);
}
```

La palabra *this* se utiliza para hacer referencia a los atributos de la misma clase.

Actividad 13

Reforzamiento

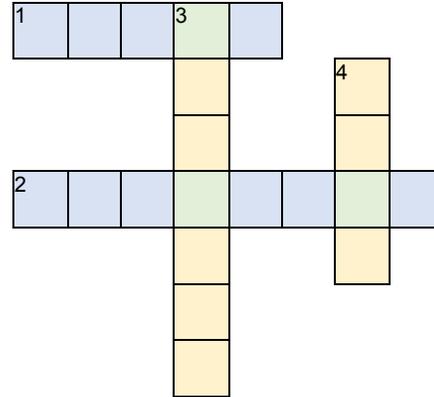
Contesta el siguiente crucigrama.

Horizontal

- 1.- Palabra reservada que se usa para acceder a atributos de la clase padre.
- 2.- Característica de la POO que permite que las subclases adquieran los atributos y métodos de la superclase.

Vertical

- 3.- La Herencia en una subclase se expresa mediante la palabra reservada.
- 4.- Palabra reservada que permite hacer referencia a atributos de la misma clase.



Actividad 14

Codificando la superclase Persona

Completa los espacios con la palabra correspondiente.

Sintaxis de la superclase Persona

```
public ____ Persona {
```

Se definen los atributos

```
protected ____ nombre;
____ int edad;
```

Se define el constructor con los parámetros nombre y edad.

```
public Persona(String nombre, int ____) {
```

Se utiliza **this** para hacer referencia a los atributos de esta clase.

```
this.nombre = nombre;
____.edad = ____;
}
```

Ahora se crean los métodos de la superclase, esto es, las acciones que una persona puede realizar. En cada uno de los métodos colocaremos un letrero alusivo a la acción.

Ejemplo:

Para el método caminar se tiene:

```
public void camina() {
    System.out.print("camina diariamente, ");
}
}
```

Para el método comer coloca un letrero referente a la acción:

```
public void come() {
    System.out.print("_____, ");
}
}
```

Completa el método para dormir:

```
public _____ duerme() {
    System.out.print("duerme para reponer energias, ");
}
}
```

Para el método bañarse:

```
_____ void seBaña() {
    System.out.print("_____ ");
}
}
```

Actividad 15

Codificando la subclase Joven

Completa los espacios con la sintaxis correspondiente.

La clase llamada Joven se deriva de la clase Persona, por ello se escribe:

```
public class Joven _____ Persona {
```

Indica que *Joven* es una subclase que **extiende** la funcionalidad de la superclase *Persona*.

Ahora, se declaran los atributos específicos de esta subclase:

```
private _____ carrera;
_____ double promedio;
```

Los atributos de la clase Joven se agregan a los de la clase *Persona*. Seguiremos con el constructor con parámetros, indicando también los atributos de la superclase, en este caso es nombre y edad, además los propios de la clase

```
public _____ (String nombre, int edad, String _____, double _____) {
```

Ahora, para hacer referencia al constructor de la clase padre *Persona* se utiliza ***super()***, la cual se coloca al principio del constructor de las subclases. El orden es importante para evitar un error de sintaxis, quedando:

```
_____(nombre, edad);
```

Como regla de sintaxis los atributos exclusivos de la clase hija *Joven* se asignan después del constructor de la clase padre referenciada con ***super()***

La palabra ***this*** hace referencia a cada atributo de esta clase, se utiliza cuando el parámetro tiene el mismo nombre, ambos deben ser del mismo tipo.

```
this._____ = carrera;
_____.promedio = promedio;
}
```

Empezaremos a construir los métodos.

El método ***mostrarDatos()*** se utilizará para visualizar la información de cada clase.

```
public void _____() {
```

Haremos uso de la palabra ***super*** para mostrar los atributos de la clase padre, atributos que se heredan de la clase *Persona*.

```
System.out.println("Nombre: " + super. _____);
System.out.println("Edad: " + _____. edad)
```

Mostramos los atributos exclusivos de la clase *Joven*.

```
System.out.println("Datos como Joven:");
System.out.println("\tEstudia: " + _____);
System.out.println("\tPromedio: " + _____);
}
```

Seguiremos con los métodos para las actividades que un joven puede realizar, en los cuales solo se mostrará un mensaje alusivo al método.

```

public void carrera( ) {
    System.out.print("_____, ");
}
public void titulo( ) {
    System.out.print("se gradua y titula de " + carrera + ", ");
}

public void cine( ) {
    _____.out._____("va al cine con su novi@, ");
}

public void fiesta( ) {
    System.out.print("va a fiestas con sus amig@s ");
}
}
    
```

Actividad 16

Codificando la clase Principal

Completa los espacios con la palabra correspondiente

Se crea la clase **Principal** y el método **main()**

```

public class _____ {
    public static void _____ (String args[ ]) {
    
```

A continuación, colocaremos un letrero para después instanciar *Joven*.

```

System.out.println("Instanciando Joven");
    
```

Se crea una **nueva** instancia de la clase *Joven*, con el constructor con parámetros: *nombre*, *edad*, *carrera*, *promedio*.

```

Joven joven1 = _____ Joven("Luis Miguel", 20, "Arquitectura", 8.5);
    
```

Ahora se hará uso del método *mostrarDatos()* para poder visualizar la información.

```

joven1.mostrarDatos( );
    
```

En seguida, se llamarán a los métodos, primero se visualizarán las actividades como persona, para ello colocaremos un letrero.

```

System.out.print("Actividades como persona\n\t");
    
```

Se llamarán los métodos que hereda de la superclase.

```
joven1.camina( );
_____.come( );
joven1._____( );
joven1.seBaña( );
System.out.println( );
```

A continuación, se llamarán a los métodos de la misma clase, esto es, las actividades que realiza específicamente la clase Joven.

```
System.out.print("Actividades como Joven:\n\t");
joven1.carrera( );
joven1.titulo( );
joven1._____( );
joven1._____( );
```

Terminaremos cerrando el método *main* y la clase *Principal*.

```
} //Se cierra el método main
} //Se cierra la clase Principal
```

Actividad 17

Primera prueba

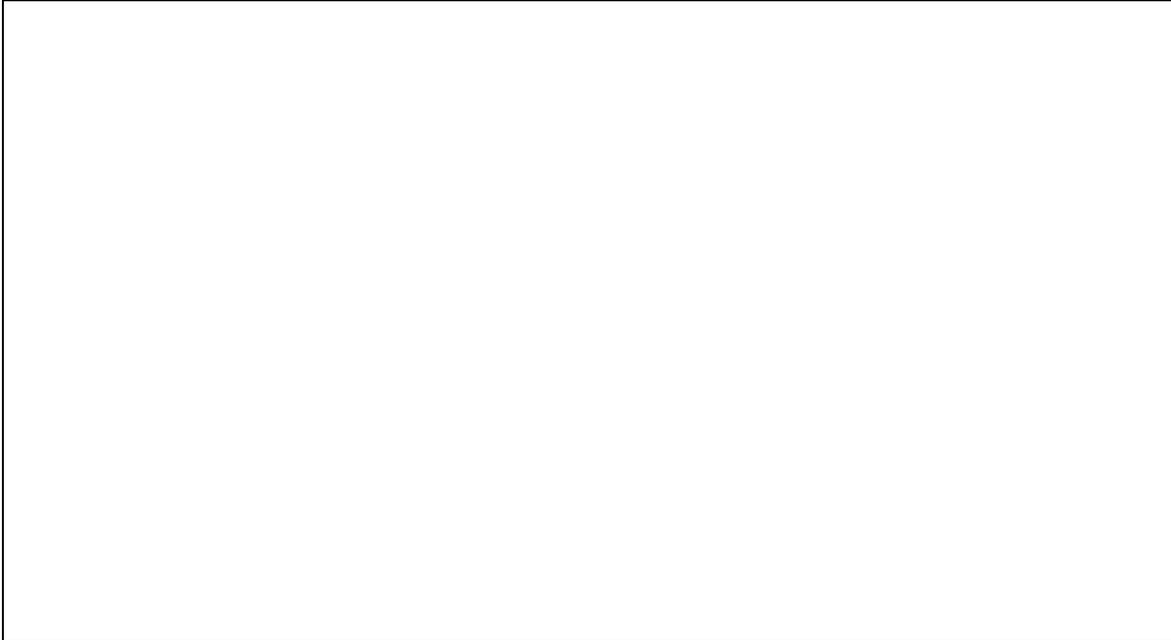
Edita y ejecuta en BlueJ las clases creadas en las actividades anteriores (Superclase, subclase joven y clase principal).

Escribe en el siguiente espacio lo que se muestra en la pantalla de ejecución.

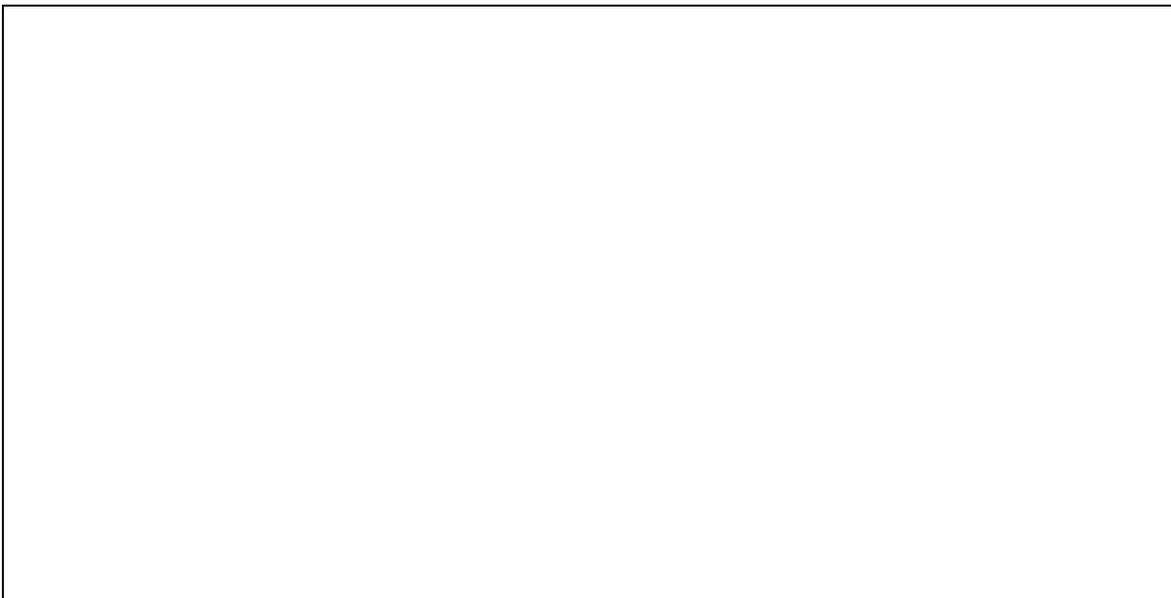
Actividad 18

Subclase Anciano

Tomando como referencia la clase Joven, codifica y edita en BlueJ la subclase Anciano. En el siguiente espacio escribe el código.



Agrega en la clase principal las sentencias para visualizar los datos y acciones del Anciano que puede tener como Persona y los datos y acciones que tiene como Anciano.



Ejemplo de pantalla de cómo puede quedar:

```

BlueJ: Terminal Window - HerenciaTransporteBlueJ
Options
Instanciando Joven
Nombre: Luis Miguel
Edad: 20
Datos como Joven:
Estudia: Arquitectura
Promedio: 8.5
Actividades como persona
camina diariamente, come 3 veces al día, duerme para reponer energías,
se baña para estar limpio y presentable
Actividades como Joven:
estudia una carrera, se gradúa y titula de Arquitectura,
va al cine con su novi@, va a fiestas con sus amig@s

Instanciando Anciano
Nombre: Don Teofilito
Edad: 75
Datos como Anciano:
Monto de Pensión: 15200.0
Grupo Tercera edad: Cabecitas blancas
Tiene: 5 nietos
Actividades como persona
camina diariamente, come 3 veces al día, duerme para reponer energías,
se baña para estar limpio y presentable
Actividades como Anciano:
va al medico con mayor frecuencia, juega con sus 5 nietos,
va con su grupo de la tercera edad Cabecitas blancas
    
```

3.3.4 Ventajas de la herencia

Entre las ventajas de la herencia se tiene:

- Reutilización del código: La herencia evita escribir el mismo código varias veces.
- Mantenimiento de aplicaciones existentes: Utilizando la herencia, si tenemos una clase con una determinada funcionalidad y tenemos la necesidad de ampliar dicha funcionalidad, no necesitamos modificar la clase existente (la cual se puede seguir utilizando para el tipo de programa para la que fue diseñada) sino que podemos

crear una clase que herede a la primera, adquiriendo toda su funcionalidad y añadiendo la suya propia.

Actividad 19

Reflexiona

1. ¿Cuál de las ventajas observaste en el programa?

Desafío 3

Diferentes tipos de transportes

Se examina la comprensión de conceptos relativos al tema de herencia. Analiza y resuelve el siguiente ejercicio:

Existen diferentes medios de transporte como pueden ser terrestre, marítimo y aéreo entre otros. Analiza el diagrama jerárquico que se muestra a continuación para poder realizar las actividades que se solicitan.

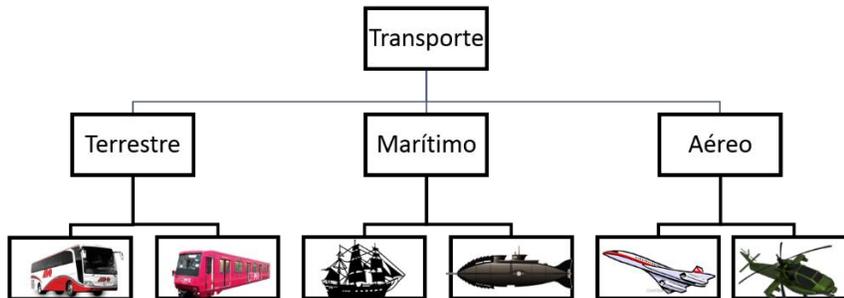


fig. 3.3.4. Organigrama de Transporte

Realiza su diagrama de clases.

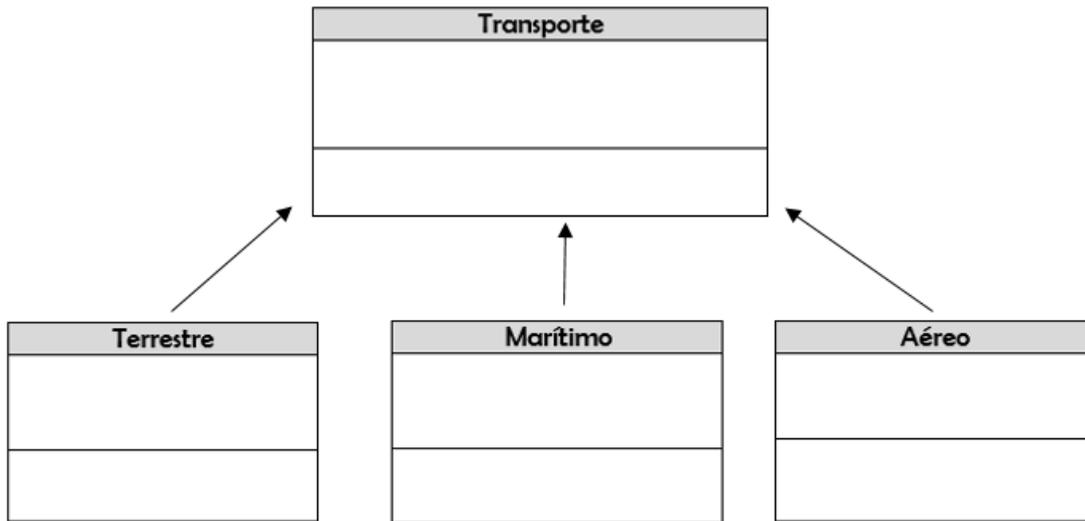


fig. 3.3.5. Diagrama de Transporte

Realiza el programa.

3.4 Soluciones

Actividades

| Actividad 1 | | |
|--|---|---------------------------------|
| 1.1. Polimorfismo significa que tiene o puede tener varias formas. | 1.2. Tres autos son de la misma marca; sin embargo, sus colores y modelo varía lo cual los hace diferentes uno de otro. | 1.3. Para crear un objeto nuevo |

| Actividad 3 | |
|--|---|
| Método para restar números enteros | Método para restar números reales |
| <pre>public int resta(int a, int b) { return(a-b); }</pre> | <pre>public double resta(double a, double b) { return(a-b); }</pre> |

| Actividad 5 | |
|---|--|
| 5.1. Porque se instancia el objeto calc1 con atributos a=0 y b=0, y la suma de estos valores es cero. | 5.2. Porque se instancia el objeto calc2 con atributos a=5 y b=4, y la suma de estos valores es igual a nueve. |

| Actividad 6 | |
|--|--|
| Constructor con tres parámetros | Método para sumar los tres atributos |
| <pre>public Calculadora(int a, int b, int c) { this.a=a; this.b=b; this.c=c; }</pre> | <pre>public int suma() { return(this.a+this.b+this.c); }</pre> |

| Actividad 7 | | |
|---|--|--|
| 7.1. Es una plantilla que define la forma de un objeto | 7.2. Son las características que un objeto tiene. | 7.3. Es la comunicación entre clases a través de la creación de objetos y llamadas a los métodos. |

| Actividad 8 |
|---|
| <pre>//Clase donde se calcula el promedio public class PromedioTres { //Método para calcular el promedio public double promedio(double a, double b, double c) { double resultado=(a+b+c)/2; return resultado; } } //Clase principal que colabora con la clase anterior import java.util.Scanner; public class PrincipalPromedioTres { //Método principal public static void main(String[] args) { //Instanciación de objeto Scanner teclado = new Scanner(System.in); //Se instancia al objeto calc con la clase PromedioTres PromedioTres calc=new PromedioTres(); //Limpia la pantalla System.out.println("\u000C"); //Declaración de variables double a=0; double b=0; double c=0; //Lee los valores de a, b y c System.out.print("Dame el valor a:"); a=teclado.nextDouble(); System.out.print("Dame el valor b:");</pre> |

```

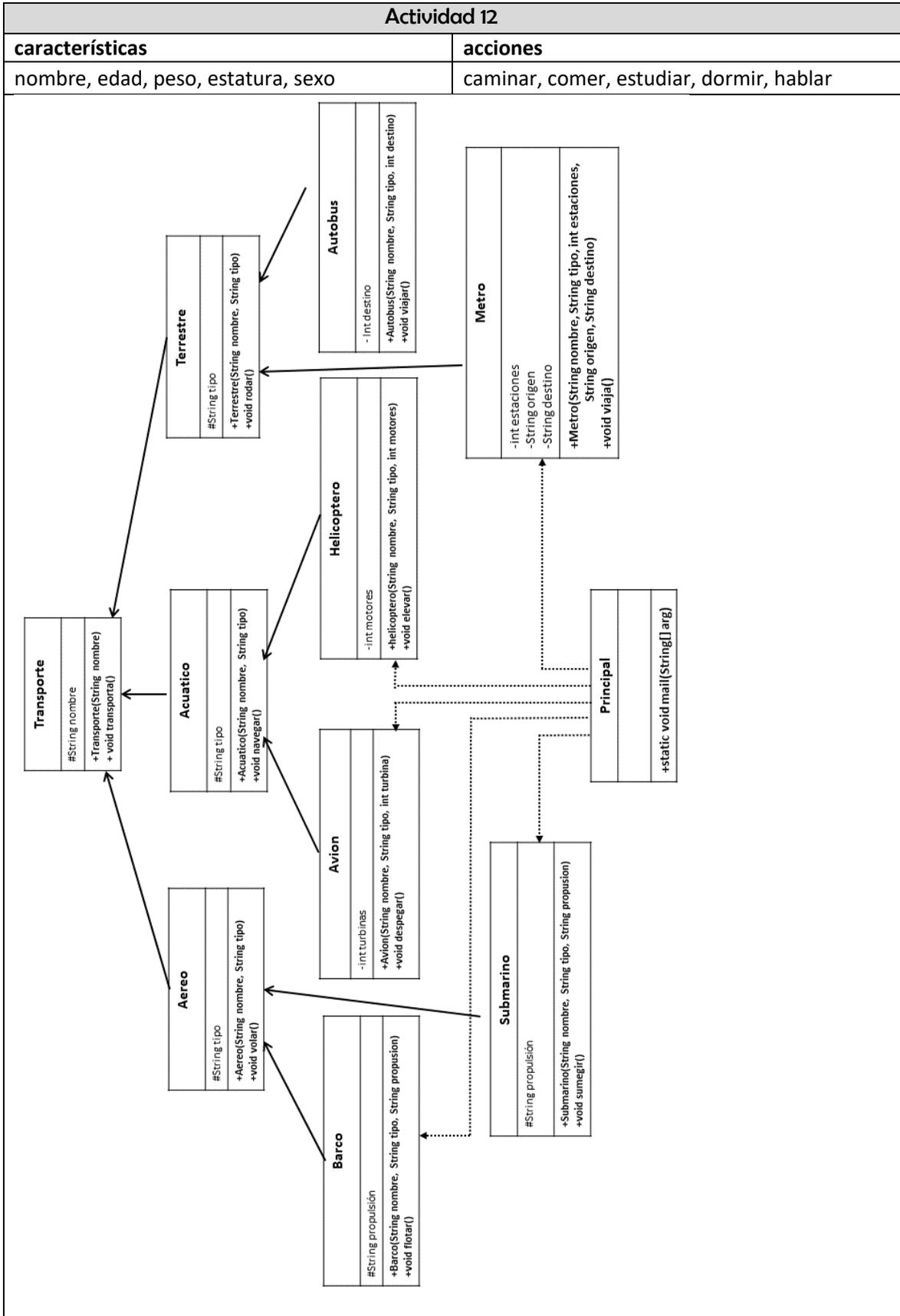
b=teclado.nextDouble();
System.out.print("Dame el valor c:");
b=teclado.nextDouble();
//Trae el resultado promedio llamando al método promedio
System.out.println("El promedio es: "+String.format("%.2f",calc.promedio(a,b,c)));
}
}
    
```

| Actividad 9 | | |
|--------------|-------------|---|
| 9.1. private | 9.2. public | 9.3. void, porque no regresa ningún valor sin embargo imprime mensajes de salida y realiza operaciones. |

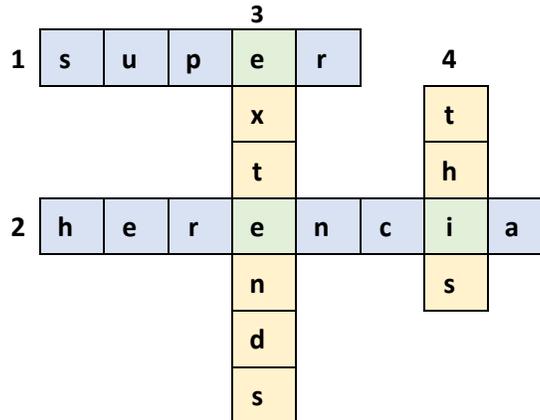
| Actividad 10 | |
|--|---|
| 10.1 La herencia es una característica de la Programación Orientada a Objeto que permite crear nuevas clases a partir de clases ya existentes, logrando reutilizar código y jerarquización de clases | 10.2 Una estructura jerárquica de clases o estructura de árbol, lo cual significa que en la Programación Orientada a Objetos todas las relaciones entre clases deben ajustarse a dicha estructura |
| 10.3 Clase padre o clase base | 10.4 Son las subclasses |

Actividad 11

| Niño | Joven | Adulto | Anciano |
|--|--|--|--|
| características | características | características | características |
| nombre edad Peso estatura sexo | nombre edad peso estatura sexo | nombre edad Peso Estatura Sexo | Nombre edad Peso estatura sexo |
| acciones | acciones | acciones | acciones |
| caminar comer estudiar dormir hablar | caminar comer estudiar dormir hablar | caminar comer estudiar dormir hablar | caminar comer estudiar dormir hablar |



Actividad 13



Actividad 14

```

public class Persona {
    protected String nombre;
    protected int edad;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
    public void camina() {
        System.out.print("camina diariamente, ");
    }
    public void come() {
        System.out.print("come 3 veces al día, ");
    }
    public void duerme() {
        System.out.print("duerme para reponer energias, ");
    }

    public void seBaña() {
        System.out.print("se baña para estar limpio y presentable");
    }
}
    
```

Actividad 15

```

public class Joven extends Persona {

    private String carrera;
    private double promedio;

    public Joven(String nombre, int edad, String carrera, double promedio) {
    
```

```

super(nombre, edad);
this.carrera = carrera;
this.promedio = promedio;
}

public void mostrarDatos( ) {
System.out.println("\tNombre: "+super.nombre);
System.out.println("\tEdad: "+super.edad);
System.out.println("Datos como Joven:");
System.out.println("\tEstudia: " + carrera);
System.out.println("\tPromedio: " + promedio);
}

public void carrera( ) {
System.out.print("estudia una carrera, ");
}

public void titulo( ) {
System.out.print("se gradua y titula de " + carrera + ", ");
}

public void cine( ) {
System.out.print("va al cine con su novi@, ");
}

public void fiesta( ) {
System.out.print("va a fiestas con sus amig@s ");
}
}
    
```

Actividad 16

```

public class Principal {

public static void main(String arg[]) {
System.out.println("Instanciando Joven");
Joven joven1 = new Joven("Luis Miguel", 20, "Arquitectura", 8.5);
joven1.mostrarDatos( );
System.out.print("Actividades como persona\n\t");
joven1.camina( );
joven1.come( );
joven1.duerme( );
System.out.print("\n\t"); //Salto de línea y sangría de un tabulador
joven1.seBaña( );
System.out.println( );
System.out.print("Actividades como Joven:\n\t");
//Métodos que se realizan en esta clase
joven1.carrera( );
joven1.titulo( );
}
}
    
```

```

System.out.print("\n\t");
joven1.cine( );
joven1.fiesta( );
}
}

```

Actividad 17

```

Options
Instanciando Joven
Nombre: Luis Miguel
Edad: 20
Datos como Joven:
Estudia: Arquitectura
Promedio: 8.5
Actividades como persona
camina diariamente, come 3 veces al día, duerme para
reponer energías,
se baña para estar limpio y presentable
Actividades como Joven:
estudia una carrera, se gradúa y titula de Arquitectura,
va al cine con su novi@, va a fiestas con sus amig@s

```

Actividad 18

```

public class Anciano extends Persona {
    private double montoPension;
    private String gpoTerceraEdad;
    private int nietos;
    //Constructor
    public Anciano(String nombre, int edad, double montoPension, String gpoTerceraEdad, int
nietos) {
        super(nombre, edad);
        this.montoPension = montoPension;
        this.gpoTerceraEdad = gpoTerceraEdad;
        this.nietos = nietos;
    }
    //Se crea el método muestraDatos() para esta clase
    public void mostrarDatos() {

```

```

        System.out.println("\tNombre: "+super.nombre);
        System.out.println("\tEdad: "+super.edad);
        //Mostramos los atributos exclusivos de esta clase
        System.out.println("Datos como Anciano:");
        System.out.println("\tMonto de Pensión: " + montoPension);
        System.out.println("\tGrupo Tercera edad: " + gpoTerceraEdad);
        System.out.println("\tTiene: " + nietos + " nietos ");
    }
    //Métodos para las actividades que un Anciano puede realizar
    public void medico() {
        System.out.print("va al medico con mayor frecuencia, ");
    }
    public void nietos() {
        System.out.print("juega con sus " + nietos + " nietos, ");
    }
    public void terceraEdad() {
        System.out.print("va con su grupo de la tercera edad " + gpoTerceraEdad + " ");
    }
}

```

```

System.out.println("\n\nInstanciando Anciano");
//Argumentos de la clase Anciano: nombre, edad, montoPension, gpoTerceraEdad
Anciano Anciano1 = new Anciano("Don Teofilito", 75, 15200.0, "Cabecitas blancas", 5);
Anciano1.mostrarDatos();
System.out.print("Actividades como persona\n\t");
//Métodos que hereda de la superclase
Anciano1.camina();
Anciano1.come();
Anciano1.duerme();
System.out.print("\n\t");
Anciano1.seBaña();
System.out.println();
System.out.print("Actividades como Anciano:\n\t");
//Métodos que se realizan en esta clase
Anciano1.medico();
Anciano1.nietos();
System.out.print("\n\t");
Anciano1.terceraEdad();

```

Actividad 19

Reutilización del código: La herencia evita escribir el mismo código varias veces.

Desafíos

| Desafío 1 | |
|--|--|
| <p>a.</p> <p>The diagram shows a large yellow rectangle with a total width of 7m and a height of 6.5m. On top of this yellow rectangle, two smaller rectangles are placed side-by-side. The left one is purple, with a width of 3.8m and a height of 4.6m. The right one is green, with a width of 2.3m and a height of 4.5m. The total width of the top two rectangles is 3.8m + 2.3m = 6.1m, leaving a gap of 0.9m between the right edge of the purple rectangle and the right edge of the yellow rectangle.</p> | <p>b.</p> <p>Alfombra1= $(3.8 * 4.6) = 17.48$ Alfombra2= $(2.3 * 4.5) = 10.35$ Salón = $(7 * 6.5) = 45.5$</p> <p>c.</p> <p>Área cubierta=$17.48 + 10.35 = 27.83$ Área descubierta=$45.5 - 27.83 = 17.67$</p> <p>d.</p> <p>Es necesario comprar más alfombras porque hay un área de 17.67 metros cuadrados descubierta.</p> |
| <p>e. Solución propuesta del problema de las alfombras. <i>//Declaración de la clase rectángulo porque es la figura del problema.</i> public class Rectangulo { <i>//Se declaran los atributos</i> private double base; private double altura; <i>//Se declaran los métodos Getter de los atributos</i> public double getBase() { return base; } public double getAltura() { return altura; } <i>/*Método constructor que no pide parámetros, pero inicializa los atributos. Son las medidas fijas del salón. */</i> public Rectangulo() { base=7; altura=6.5; } <i>/*Método constructor que pide parámetros y guarda los valores en los atributos. Sirve para indicar las medidas de las alfombras. */</i> public Rectangulo(double b, double a) { this.base=b; this.altura=a; } <i>/*Método para calcular el área del rectángulo. */</i> public double calcularArea() {</p> | |

```

double area = base*altura;
return area;
}
//Método principal
public static void main(String[] args)
{
    //Se limpia la pantalla
    System.out.print("\u000C");
    //Se instancia un objeto de la clase con el constructor que no pide parámetros.
    Rectangulo salon=new Rectangulo();
    //Se instancian dos objetos de la clase con el constructor que pide parámetros
    Rectangulo alfombra1=new Rectangulo(4.6, 3.8);
    Rectangulo alfombra2=new Rectangulo(4.5, 2.3);
    //Se llama al método calcularArea con cada objeto y se guardan los valores en variables
    double area1=salon.calcularArea();
    double area2=alfombra1.calcularArea();
    double area3=alfombra2.calcularArea();
    //Se escriben los datos de cada rectángulo: salón, alfombra1 y alfombra2
    System.out.println("Salon:\nBase = "+salon.getBase()+"; Altura =
"+salon.getAltura()+"\nArea = "+String.format("%.2f",area1));
    System.out.println("Alfombra1:\nBase = "+alfombra1.getBase()+"; Altura =
"+alfombra1.getAltura()+"\nArea = "+String.format("%.2f",area2));
    System.out.println("Alfombra2:\nBase = "+alfombra2.getBase()+"; Altura =
"+alfombra2.getAltura()+"\nArea = "+String.format("%.2f",area3));
    //Se realizan las operaciones de suma y resta de áreas para conocer el piso alfombrado y el
    piso descubierto.
    double alfombra=area2+area3;
    double piso=area1-alfombra;
    //Se escriben en pantalla los resultados obtenidos
    System.out.println("\n\nEl área del piso es: "+area1+"\nEl área alfombrada es: "+alfombra);
    System.out.println("El área que queda por alfombrar es: "+piso);
}
}

```

Desafío 2

Solución propuesta del problema de las Áreas.

Código de la clase *Rectangulo*

```

//Librería Scanner para pedir datos por el teclado
import java.util.Scanner;
//Declaración de la clase Rectangulo
public class Rectangulo
{
    //Atributos de la clase
    private double base;
    private double altura;
    private double area;
    //Método para pedir datos por teclado

```

```

public void pedirDatos()
{
    Scanner teclado = new Scanner(System.in);
    System.out.println("DATOS DEL RECTÁNGULO \n");
    System.out.print("Valor de la base:");
    base=teclado.nextDouble();
    System.out.print("Valor de la altura:");
    altura=teclado.nextDouble();
}
//Método para calcular el área del rectángulo
public void calcularArea()
{
    area=base*altura;
    System.out.println("El área del rectángulo es: "+String.format("%.2f",area));
}
}

```

Código de la clase *Triangulo*

```

//Librería Scanner para pedir datos por el teclado
import java.util.Scanner;
//Declaración de la clase Triangulo
public class Triangulo
{
    //Atributos de la clase
    private double base;
    private double altura;
    private double area;
    //Método para pedir datos por teclado
    public void pedirDatos()
    {
        Scanner teclado = new Scanner(System.in);
        System.out.println("\nDATOS DEL TRIÁNGULO \n");
        System.out.print("Valor de la base:");
        base=teclado.nextDouble();
        System.out.print("Valor de la altura:");
        altura=teclado.nextDouble();
    }
    //Método para calcular el área del triángulo
    public void calcularArea()
    {
        area=base*altura/2;
        System.out.println("El área del triángulo es: "+String.format("%.2f",area));
    }
}

```

Código de la clase *Circulo*

```

//Librería Scanner para pedir datos por el teclado
import java.util.Scanner;

```

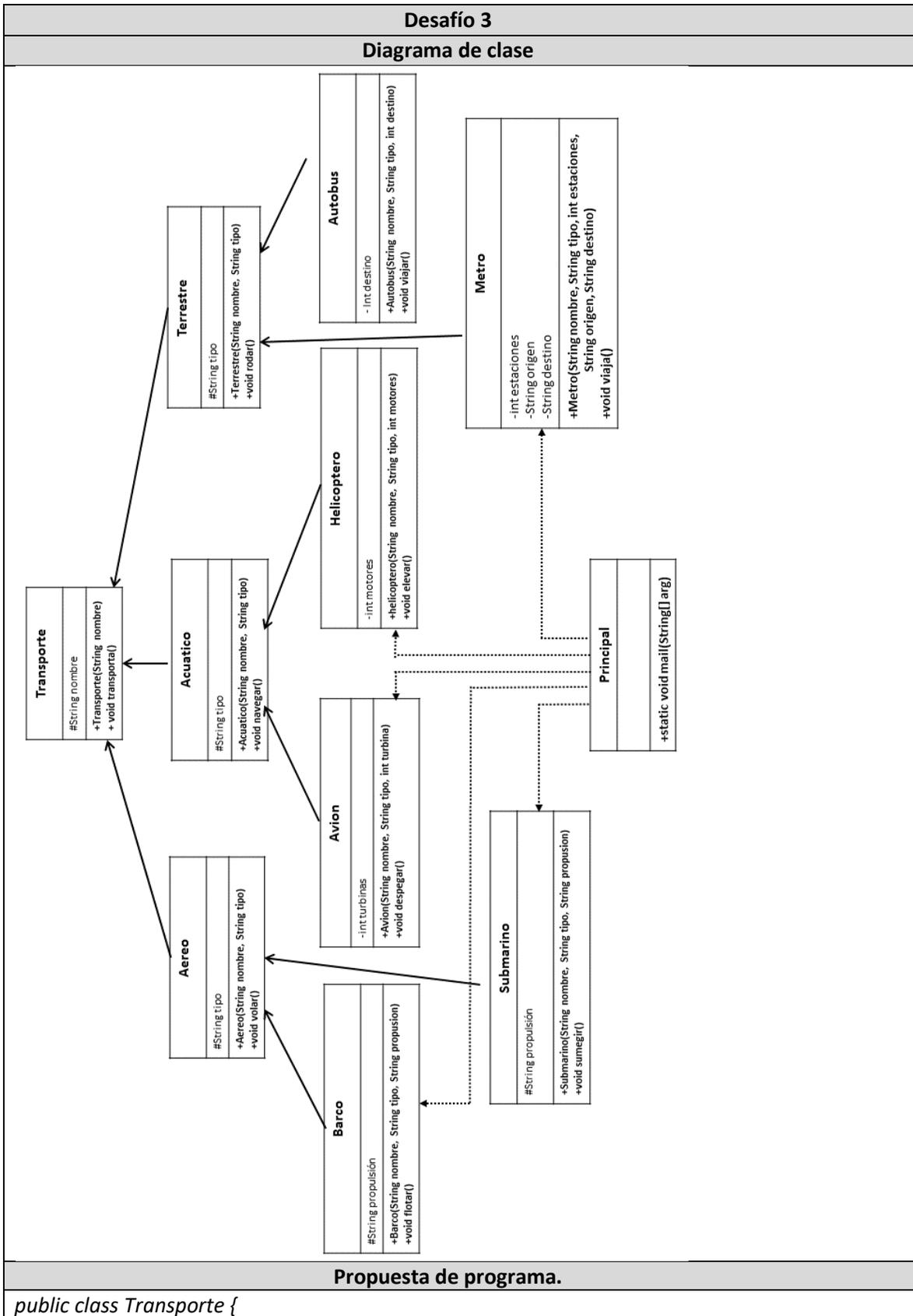
```
//Declaración de la clase Circulo
public class Circulo
{
    //Atributos de la clase
    private double radio;
    private double area;
    //Método para pedir datos por teclado
    public void pedirDatos()
    {
        Scanner teclado = new Scanner(System.in);
        System.out.println("\nDATOS DEL CÍRCULO \n");
        System.out.print("Valor del radio:");
        radio=teclado.nextDouble();
    }
    //Método para calcular el área del círculo
    public void calcularArea()
    {
        area=3.1416*radio*radio;
        System.out.println("El área del círculo es: "+String.format("%.2f",area));
    }
}

```

Código de la clase principal

```
public class Areas
{
    //Método principal
    public static void main(String[] args)
    {
        //Limpia la pantalla
        System.out.println("\u000C");
        System.out.println("CÁLCULO DE ÁREAS \nCON COLABORACIÓN DE CLASES \n");
        //Instanciación de objeto de la clase rectángulo
        Rectangulo rectangulo=new Rectangulo();
        //Llamada a los métodos con el objeto rectangulo
        rectangulo.pedirDatos();
        rectangulo.calcularArea();
        //Instanciación de objeto de la clase triángulo
        Triangulo triangulo=new Triangulo();
        //Llamada a los métodos con el objeto triangulo
        triangulo.pedirDatos();
        triangulo.calcularArea();
        //Instanciación de objeto de la clase circulo
        Circulo circulo=new Circulo();
        //Llamada a los métodos con el objeto circulo
        circulo.pedirDatos();
        circulo.calcularArea();
    }
}

```



```

protected String nombre;

public Transporte(String nombre) {
    this.nombre = nombre;
}

public void transporta() {
    System.out.println("El " + nombre + " es un transporte");
}
}

```

```

public class Aereo extends Transporte {

    protected String tipo;

    public Aereo(String nombre, String tipo) {
        super(nombre);
        this.tipo = tipo;
    }

    public void volar() {
        System.out.println("El " + super.nombre + " se desplaza en el aire ");
    }
}

```

```

public class Barco extends Acuatico {

    private String propulsion;

    public Barco(String nombre, String tipo, String propulsion) {
        super(nombre, tipo);
        this.propulsion = propulsion;
    }

    public void flotar() {
        System.out.println("El " + super.nombre + " es un " + super.tipo + ", flota y se impulsa con " +
propulsion);
    }
}

```

```

public class Submarino extends Acuatico {

    private String propulsion;

    public Submarino(String nombre, String tipo, String propulsion) {
        super(nombre, tipo);
        this.propulsion = propulsion;
    }
}

```

```

    public void sumergir() {
        System.out.println("El " + super.nombre + " es un " + super.tipo + ", se sumerge y se impulsa
con " + propulsion);
    }
}

```

```

public class Acuatico extends Transporte {

    protected String tipo;

    public Acuatico(String nombre, String tipo) {
        super(nombre);
        this.tipo = tipo;
    }

    public void navegar() {
        System.out.println("El " + super.nombre + " se desplaza en el agua");
    }
}

```

```

public class Avion extends Aereo {

    private int turbinas;

    public Avion(String nombre, String tipo, int turbinas) {
        super(nombre, tipo);
        this.turbinas = turbinas;
    }

    public void despegar() {
        System.out.println("El " + super.nombre + " es un avion " + super.tipo + ", despegamos con sus "
+ turbinas + " turbinas");
    }
}

```

```

public class Helicoptero extends Aereo {

    private int motores;

    public Helicoptero(String nombre, String tipo, int motores) {
        super(nombre, tipo);
        this.motores = motores;
    }

    public void elevar() {
        System.out.println("El " + super.nombre + " es un helicóptero de " + super.tipo + ", se eleva
con sus " + motores + " hélices");
    }
}

```

```

}
public class Terrestre extends Transporte {

    protected String tipo;

    public Terrestre(String nombre, String tipo) {
        super(nombre);
        this.tipo = tipo;
    }

    public void rodar() {
        System.out.println("El " + super.nombre + " rueda por tierra");
    }
}

public class Metro extends Terrestre {

    private int estaciones;
    private String origen;
    private String destino;

    public Metro(String nombre, String tipo, int estaciones, String origen, String destino) {
        super(nombre, tipo);
        this.estaciones = estaciones;
        this.origen = origen;
        this.destino = destino;
    }

    public void viajar() {
        System.out.println("El " + super.nombre + " es un transporte " + super.tipo + ", que tiene \n"
            + estaciones + " estaciones, algunas son subterráneas, \n"
            + "va de " + origen + " a " + destino
            + " dentro del área Metropolitana");
    }
}

public class Autobus extends Terrestre {

    private int destinos;

    public Autobus(String nombre, String tipo, int destinos) {
        super(nombre, tipo);
        this.destinos = destinos;
    }

    public void viajar() {
        System.out.println("Los " + super.nombre + " son un transporte " + super.tipo

```

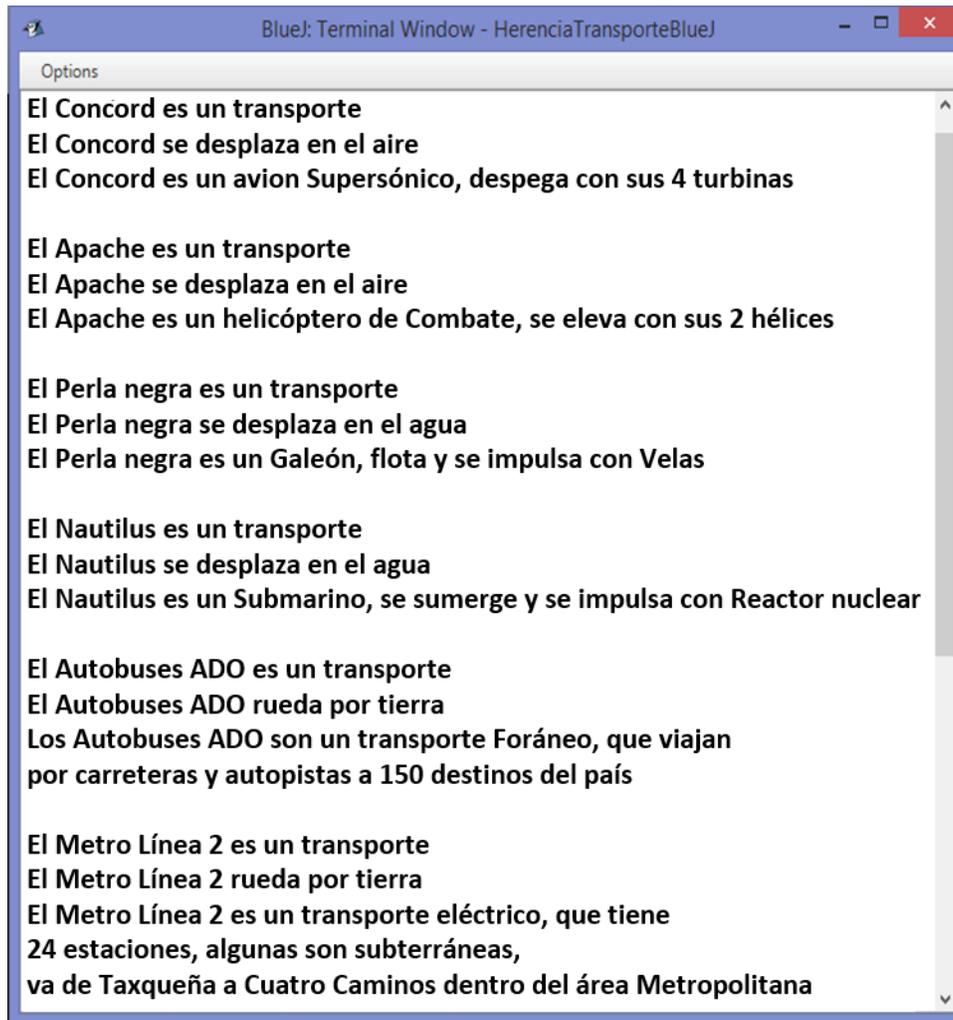
```

        + ", que viajan \npor carreteras y autopistas a " + destinos + " destinos del país");
    }
}

public class Principal {

    public static void main(String[] args) {
        //Parametros de la clase Avion: nombre, tipo, turbinas
        Avion avion1 = new Avion("Concord", "Supersónico", 4);
        //Como Transporte, transporta
        avion1.transporta();
        //Como transporte aereo, vuela
        avion1.volar();
        //Como avión, despegar
        avion1.despegar();
        System.out.println();
        //Parametros de la clase Helicoptero: nombre, tipo, motores
        Helicoptero heli1 = new Helicoptero("Apache", "Combate", 2);
        heli1.transporta();
        heli1.volar();
        heli1.elevar();
        System.out.println();
        //Parametros de la clase Barco: nombre, tipo, propulsión
        Barco barco1 = new Barco("Perla negra", "Galeón", "Velas");
        barco1.transporta();
        barco1.navegar();
        barco1.flotar();
        System.out.println();
        //Parametros de la clase Submarino: nombre, tipo, propulsión
        Submarino sub1 = new Submarino("Nautilus", "Submarino", "Reactor nuclear");
        sub1.transporta();
        sub1.navegar();
        sub1.sumergir();
        System.out.println();
        //Parametros de la clase Autobus: nombre, tipo, destinos
        Autobus aut1 = new Autobus("Autobuses ADO", "Foráneo", 150);
        aut1.transporta();
        aut1.rodar();
        aut1.viajar();
        System.out.println();
        //Parametros de la clase Metro: nombre, estaciones, origen, destino
        Metro metro1 = new Metro("Metro Línea 2", "eléctrico", 24, "Taxqueña", "Cuatro Caminos");
        metro1.transporta();
        metro1.rodar();
        metro1.viajar();
        System.out.println();
    }
}

```

Resultado.


```

Options
El Concord es un transporte
El Concord se desplaza en el aire
El Concord es un avion Supersónico, despegas con sus 4 turbinas

El Apache es un transporte
El Apache se desplaza en el aire
El Apache es un helicóptero de Combate, se eleva con sus 2 hélices

El Perla negra es un transporte
El Perla negra se desplaza en el agua
El Perla negra es un Galeón, flota y se impulsa con Velas

El Nautilus es un transporte
El Nautilus se desplaza en el agua
El Nautilus es un Submarino, se sumerge y se impulsa con Reactor nuclear

El Autobuses ADO es un transporte
El Autobuses ADO rueda por tierra
Los Autobuses ADO son un transporte Foráneo, que viajan
por carreteras y autopistas a 150 destinos del país

El Metro Línea 2 es un transporte
El Metro Línea 2 rueda por tierra
El Metro Línea 2 es un transporte eléctrico, que tiene
24 estaciones, algunas son subterráneas,
va de Taxqueña a Cuatro Caminos dentro del área Metropolitana

```

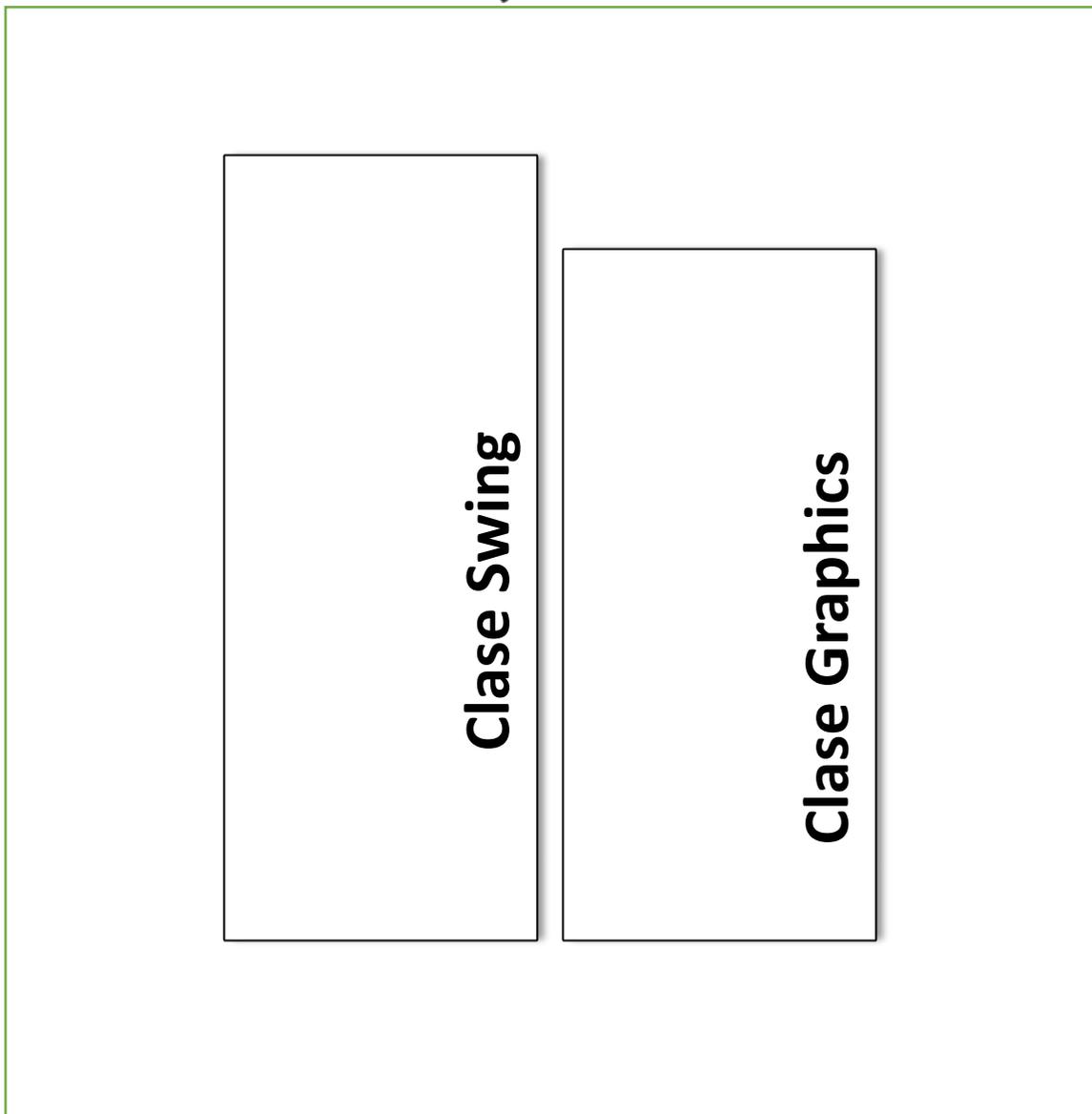
3.5 Referencias

- Ávila, S. et. al. (2019). *Paquete didáctico para la asignatura de Cibernética y Computación II*. México: ENCCH Plantel Oriente, UNAM.
- Dean, J. (2009). *Introducción a la programación con Java*. Mc. Graw-Hill. México.
- Guardati, Silvia. (2015). *Estructuras de datos básicas - Programación Orientada a Objetos con Java*. México. Alfaomega.
- Medina, N. (2016). *Programación Orientada a Objetos con Java "La Novela"*. México. Alfaomega.
- Ricardo, M. (2014). *Herencia en Java, con ejemplos*. [en línea] Disponible en: <https://jarroba.com/herencia-en-la-programacion-orientada-a-objetos-ejemplo-en-java/> [Consultado el 1 de marzo de 2019].

- Rodriguez, A. (2019). *Aprende a programar*. [en línea]. Recuperado de https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=660:sobreescribir-metodos-en-java-tipo-estatico-y-dinamico-ligadura-metodos-polimorficos-ejercicio-cu00690b&catid=68&Itemid=188 el 23 de agosto de 2019.
- Rulas. (2009). *Vida de programador*. [en línea]. Recuperado de: <http://rauldice.blogspot.com/2009/10/polymorphism-for-dummies.html> el 22 de agosto de 2019.
- Zárate, U. (s.f.). *Programación con Java*. [en línea] Disponible en: <http://profesores.fi-b.unam.mx/carlos/java/> [Consultado el 1 de septiembre de 2019].

UNIDAD 4

Interfaz gráfica de usuario.



Unidad 4. Interfaz gráfica de usuario.

TIEMPO: 22 HORAS

Propósito de la unidad:

Al finalizar la unidad el alumno: Desarrollará programas en Java utilizando interfaces gráficas de usuario para aplicar y ampliar sus conocimientos de la programación orientada a objetos.

Aprendizajes.

El alumno:

- ✓ Conoce las características de la Clase Swing.
- ✓ Elabora programas con una interfaz gráfica de usuario, aplicando las Clases: JFrame, JLabel, JButton
- ✓ Propone un proyecto que utilice las Clases: JFrame, JLabel y JButton.
- ✓ Elabora programas con una interfaz gráfica de usuario, aplicando las Clases: JTextField, JTextArea, JComboBox.
- ✓ Propone un proyecto que utilice las Clases: JTextField, JTextArea y JComboBox.
- ✓ Elabora programas con una interfaz gráfica de usuario, aplicando las Clases: JMenuBar, JMenu, JMenuItem, JCheckBox, JRadioButton.
- ✓ Elabora programas con una interfaz gráfica de usuario, aplicando las Clases: JCheckBox, JRadioButton.
- ✓ Elabora programas con interfaz gráfica de usuario aplicando las Clases: setColor, drawLine, drawRect, drawRoundRect, drawOval, drawPolygon.
- ✓ Elabora programas con interfaz gráfica de usuario aplicando las Clases: fillRect, fillRoundRect, fillOval, fillPolygon.
- ✓ Propone un proyecto que utilice las Clases: setColor, drawLine, drawRect, drawRoundRect, drawOval, drawPolygon, fillRect, fillRoundRect, fillOval, fillPolygon.
- ✓ Desarrolla un proyecto que integre las Clases estudiadas en esta unidad.

4.1 Clase Swing

Aprendizajes esperados.

El alumno:

- ✓ Conoce las características de la Clase Swing.
- ✓ Elabora programas con una interfaz gráfica de usuario, aplicando las Clases: JFrame, JLabel y JButton.
- ✓ Elabora programas con interfaz gráfica de usuario aplicando las Clases: JTextField, JTextArea y JComboBox.
- ✓ Elabora programas con interfaz gráfica de usuario aplicando las Clases: JMenuBar, JMenu, JMenuItem.
- ✓ Elabora programas con interfaz gráfica de usuario aplicando las Clases: JCheckBox y JRadioButton.

Contenido temático.

- Concepto de Interfaz gráfica de usuario (GUI)
- La Clase Swing.
- Componentes javax.swing.
- La Clase AWT como antecedente de la Clase Swing.
- Relación de la Clase Swing con la librería AWT: java.awt.* y java.awt.event.*
- JFrame.
- JLabel.
- JButton.
- JTextField.
- JTextArea.
- JComboBox.
- JCheckBox.
- JRadioButton
- JMenuBar.
- JMenu.
- JMenuItem.

Objetivo:

El alumno:

- ✓ Conozca las características de la interfaz gráfica de usuario, la Clase Swing y emplee los componentes JFrame, JLabel JButton, JTextField, JTextArea, JComboBox, JCheckBox, JRadioButton, JMenuBar, JMenu, JMenuItem.
- ✓ Comprenda el uso de algunos componentes de la clase Swing para la construcción de una interfaz gráfica.

4.1.1 Introducción

La Interfaz Gráfica de Usuario, conocida como GUI, (Graphic User Interface por sus siglas en inglés), se compone de un conjunto de formas gráficas y métodos que permiten a los usuarios la interacción con un sistema, para lo cual se emplean un conjunto de formas gráficas e imágenes, estas se componen de elementos como botones, íconos, ventanas, fuentes, entre otros, estos representan funciones, acciones e información correspondiente al contexto de ese sistema. Asimismo, permite visualizar de una manera más amigable la entrada y salida de datos, con la ayuda de diversos componentes, por ejemplo: etiquetas, botones, cuadros de texto, casillas de verificación, listas despegables, botones de selección, menús, elementos de menús, barra de menús y áreas de texto.

Java proporciona dos bibliotecas de clases para crear interfaces gráficas de usuarios: AWT y Swing, esta última abarca componentes como botones, tablas, marcos, etcétera. Estos se identifican porque pertenecen al paquete javax.swing, permiten brindar una interacción con el usuario del sistema y cada uno corresponde a una clase en Java. Dentro de los más importantes de la clase Swing tenemos los siguientes:

| | |
|--|---|
| <p style="text-align: center;">Contenedores.</p> <p>Estos elementos permiten agrupar y contener a los elementos gráficos.</p> | <ul style="list-style-type: none"> • JFrame • JDialog • JPanel |
| <p style="text-align: center;">Componentes atómicos.</p> <p>Son aquellos elementos que no pueden almacenar a otros elementos gráficos.</p> | <ul style="list-style-type: none"> • JLabel • JCheckBox • JComboBox • JButton • JRadioButton |
| <p style="text-align: center;">Componentes de texto.</p> <p>Son aquellos que permiten presentar texto tanto para la entrada como para la salida de información.</p> | <ul style="list-style-type: none"> • JTextField • JTextArea • JTextPane |
| <p style="text-align: center;">Componentes de menú.</p> <p>Permiten la creación y vinculación de opciones de menú en una ventana de aplicación.</p> | <ul style="list-style-type: none"> • JMenuBar • JMenu • JMenuItem |

El antecedente de la clase Swing para la elaboración de interfaces gráficas fue AWT (Abstract Window Toolkit), de ella se heredan los elementos para el manejo de eventos y proporciona la librería estándar para el desarrollo de una interfaz gráfica de usuario.

Actividad 1

Conceptos de interfaz gráfica de usuario.

Responde las siguientes preguntas.

1.1. ¿Qué significan las siglas GUI?

1.2. ¿Qué es la clase Swing?

1.3. ¿Cuáles son los componentes de la clase Swing?

1.4. ¿Cuál es la relación entre la clase Swing y la clase AWT?

4.1.2 Clase Swing con NetBeans

NetBeans es un entorno de desarrollo (IDE), esto es, una herramienta para escribir, compilar, depurar y ejecutar programas escritos en Java o en otros lenguajes de programación. Es un producto libre y gratuito sin restricciones de uso que simplifica el desarrollo de aplicaciones para Java Swing, la cual es una biblioteca gráfica para Java, que incluye widgets (pequeñas aplicaciones), para interfaz gráfica de usuario tales como caja de texto, botones, listas desplegables, entre otros.

Actividad 2

Creación de un proyecto en NetBeans

Realiza los siguientes pasos para empezar a trabajar con NetBeans y la clase Swing:

- Abrir el programa NetBeans  y crear un nuevo proyecto desde el menú **Archivo** y la opción **Proyecto Nuevo**.

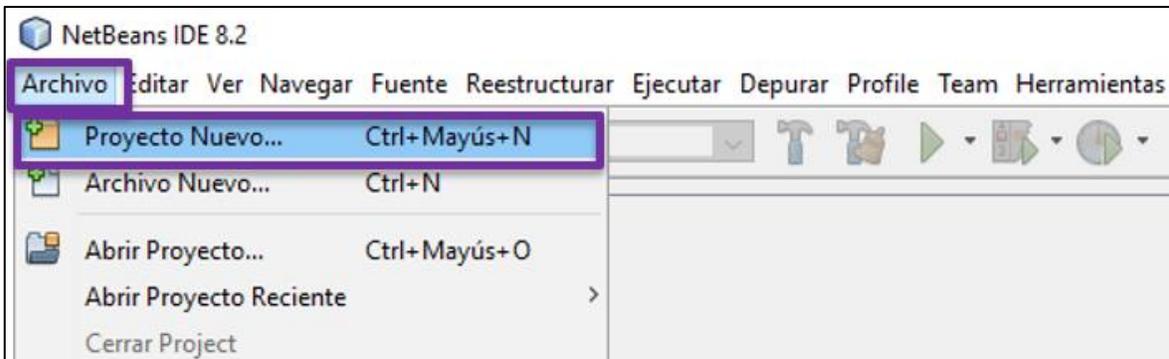


fig. 4.1.1. Nuevo proyecto.

- b. En la ventana que se muestra, elegir Categorías **Java** y Proyectos **Java Application**. Oprimir **Siguiente**.

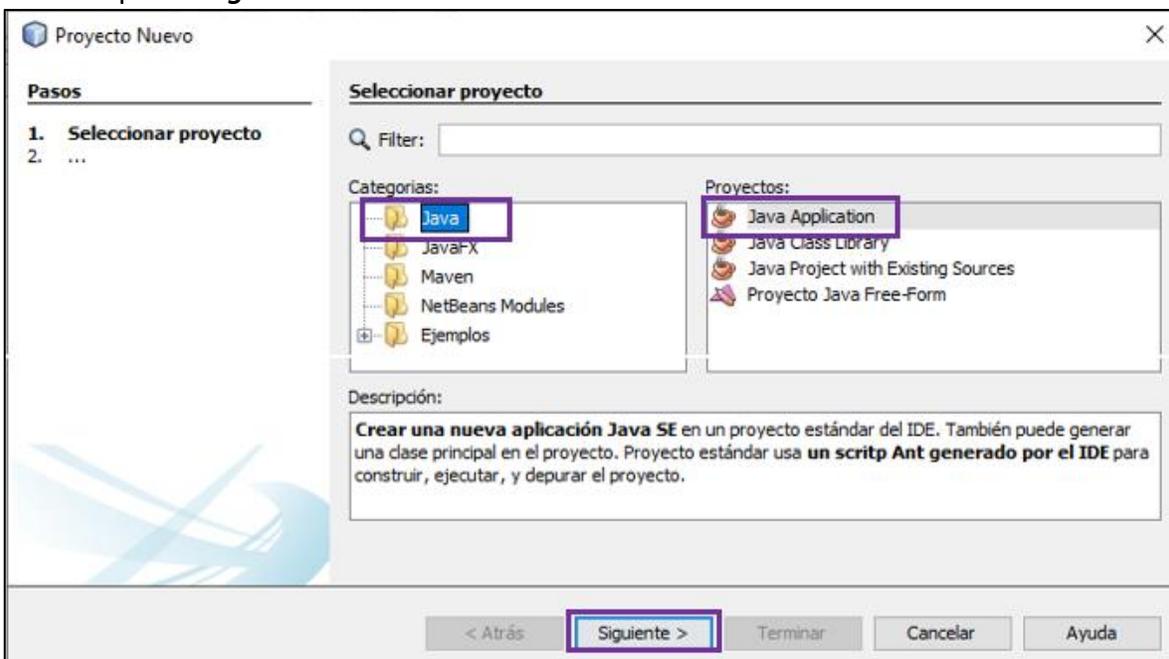


fig. 4.1.2. Aplicación Java.

- c. Escribir el **nombre del proyecto**, en este caso **Cibernetica**, elegir la ubicación donde se va a guardar con el botón **Examinar**, desactivar la casilla **Crear clase principal** y presiona **Terminar**.

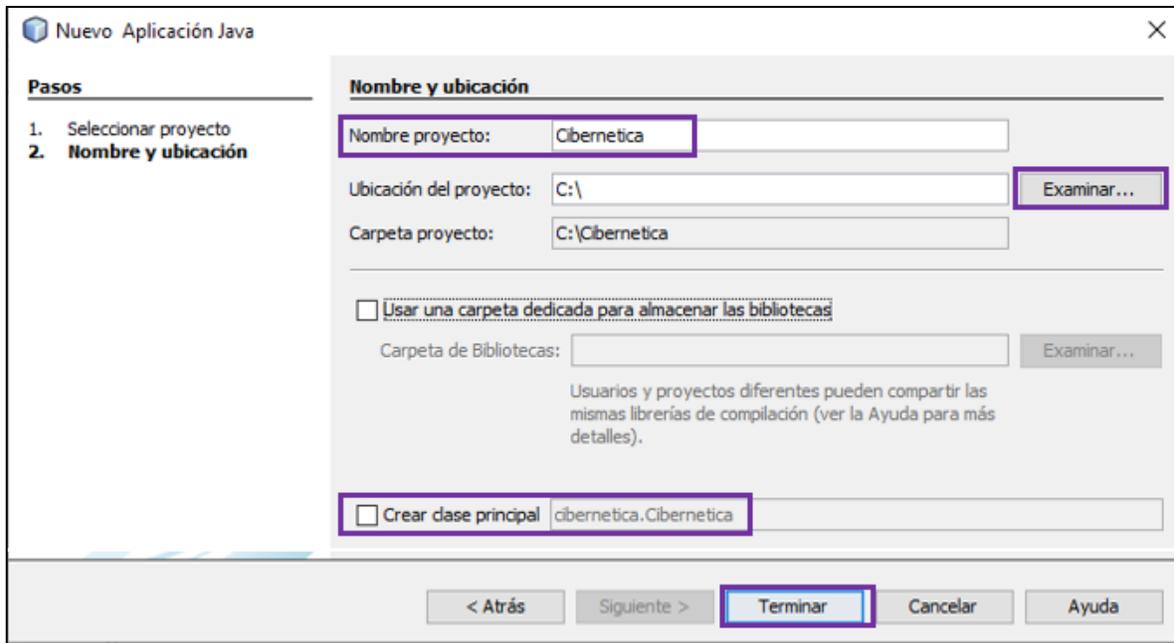


fig. 4.1.3. Nombre y ubicación.

- d. Después de esto, aparecerá el programa abierto y del lado izquierdo el proyecto como se nombró. Desplegar las carpetas que lo conforman dando clic en el símbolo [+]. Dar clic derecho en la carpeta **Paquete de fuentes**, seleccionar **Nuevo** y la opción **Java Package**.

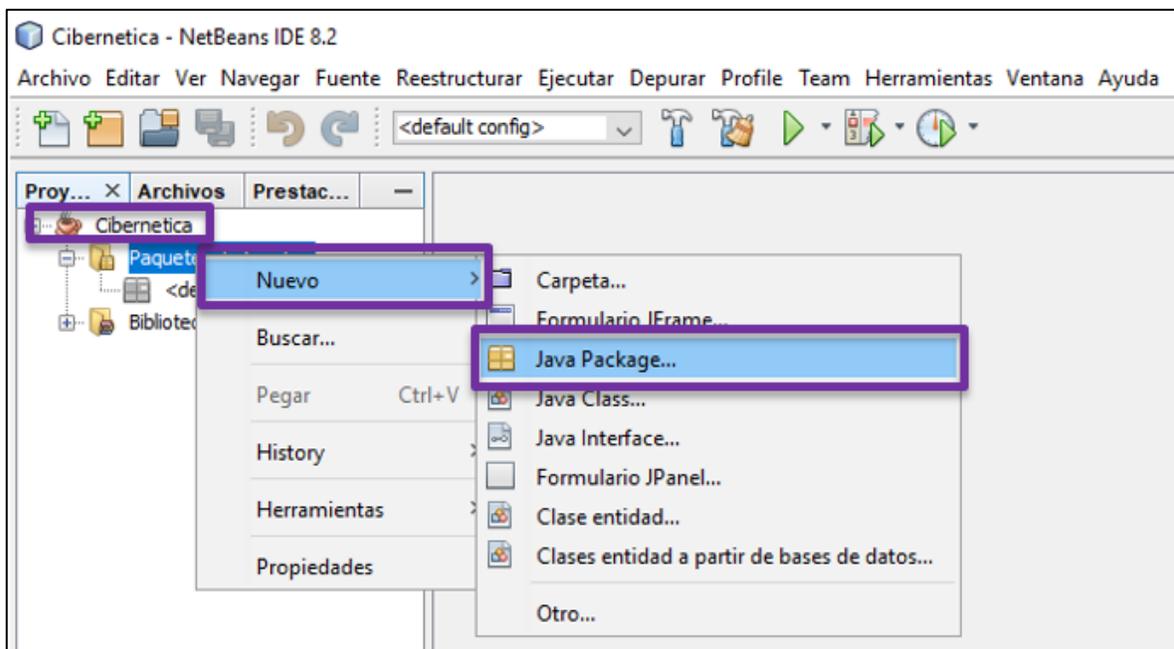


fig. 4.1.4. Nuevo paquete.

- e. Escribir el nombre al paquete **CiberII** y dar clic en **Terminar**.

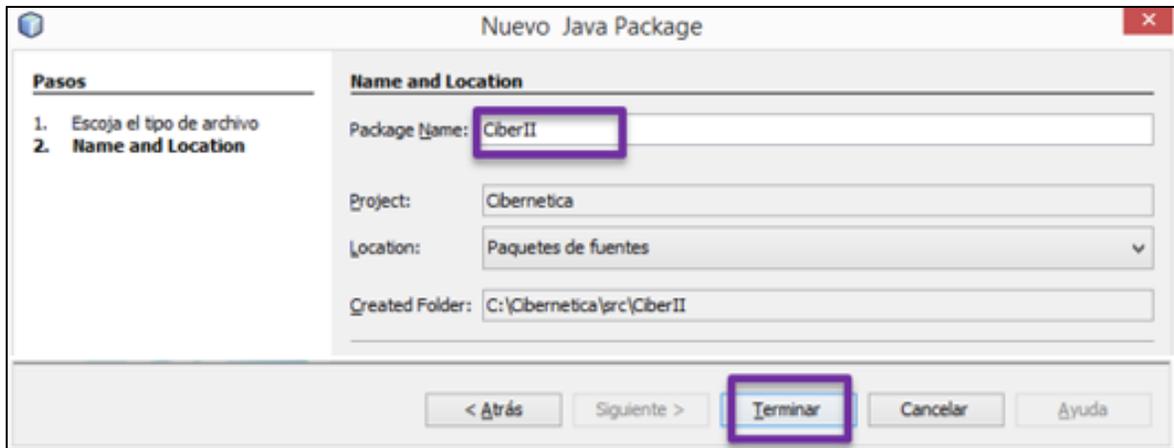


fig. 4.1.5. Nombre del paquete.

- f. Dar clic con el botón derecho del mouse sobre el paquete creado, elegir **Nuevo** y la opción **Formulario JFrame**.

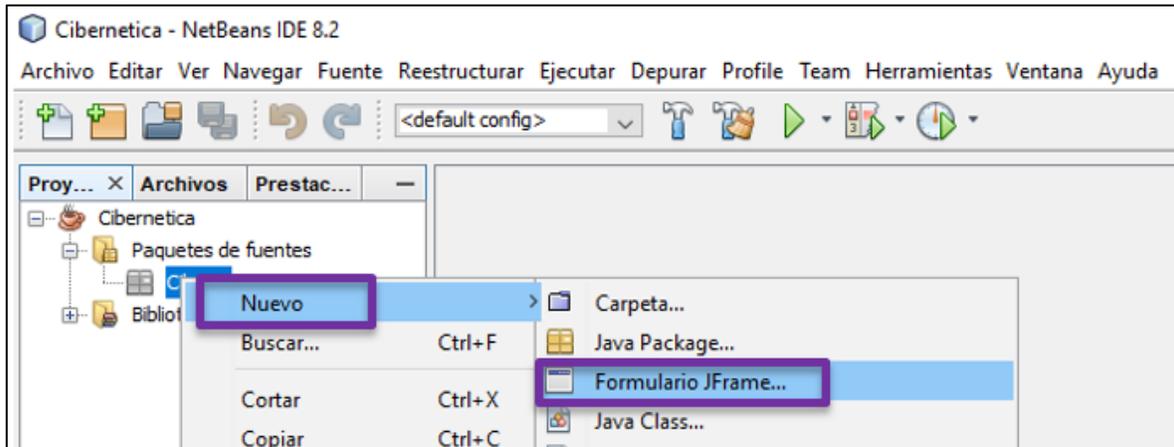


fig. 4.1.6. Nuevo formulario.

- g. Escribir el nombre a la clase **Mensaje** y dar clic en **Terminar**.

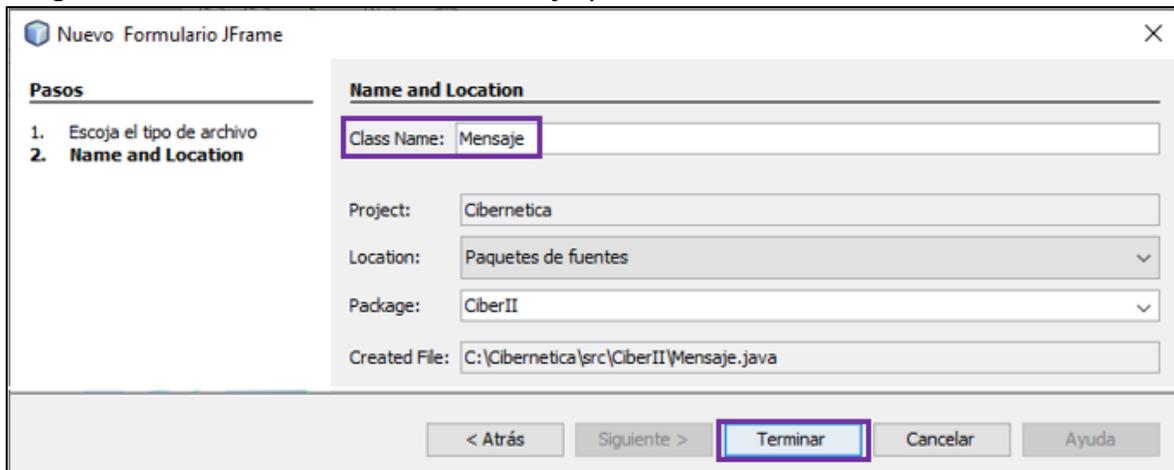


fig. 4.1.7. Nombre de la clase.

- h. Imprime y pega la pantalla resultante que muestre el Proyecto **Cibernetica**, el Paquete **CiberII** y la Clase **Mensaje**.



4.1.3 Entorno de trabajo de NetBeans

A continuación, explicaremos algunos elementos del entorno de trabajo de NetBeans:

- a. **Barra de título:** presenta el nombre del proyecto y el nombre del programa.
- b. **Barra de menús:** contiene los diferentes menús a los que se puede tener acceso.
- c. **Barra de herramientas:** esta área tiene botones para crear un nuevo archivo, crear un nuevo proyecto, abrir proyecto, guardar todo, deshacer, rehacer y ejecutar proyecto, entre otras.
- d. **Botón Cerrar:** Para cerrar el programa oprime el botón que se encuentra del lado derecho en la barra de título.

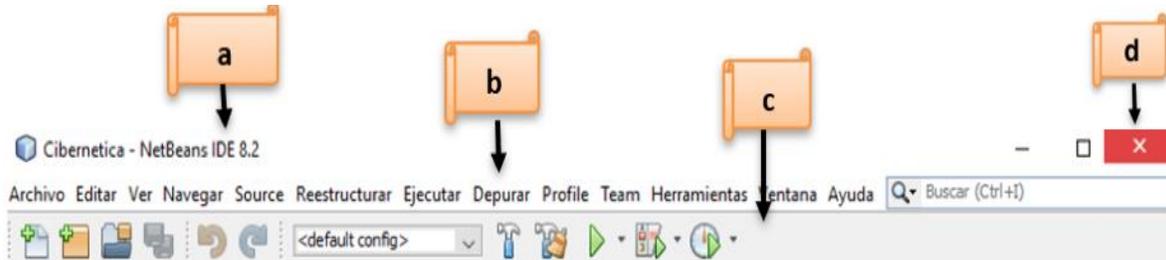


fig. 4.1.8. Barras de título, menús y herramientas.

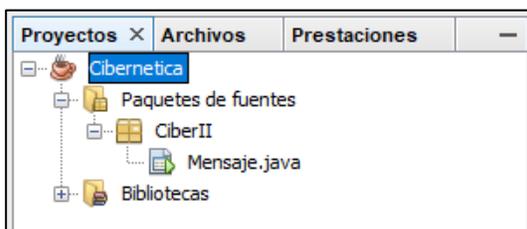


fig. 4.1.9. Ventana de lista de proyectos.

Lista de proyectos: muestra los paquetes y clases que contiene el o los proyectos.

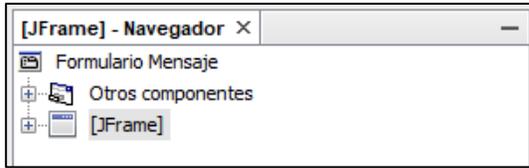


fig. 4.1.10. Ventana de navegador de componentes.

Ventana de diseñador visual: está formada por la pestaña de archivos abiertos, los botones para intercambiar entre la vista código (Source) y vista diseño (Design) así como modo selección, modo conexión, diseño previo, tipos de alineaciones y el área del formulario.

Navegador de componentes: enlista todos los componentes que contiene el JFrame o formulario.

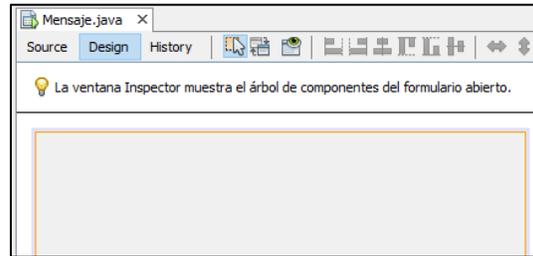


fig. 4.1.11. Ventana de diseñador visual.

Paleta de componentes: contiene todos los elementos que se pueden agregar al formulario.

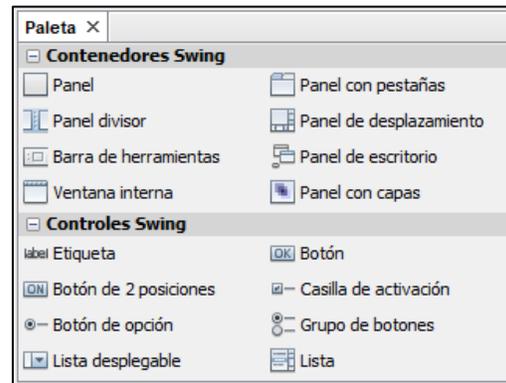


fig. 4.1.12. Ventana de paleta de componentes.

Contenedores Swing: panel, panel divisor, barra de herramientas, etcétera.

Controles Swing: etiqueta, botón, botón de opción, casilla de activación, grupo de botones, lista desplegable, etcétera.

Propiedades de los elementos: dependiendo del componente seleccionado muestra las características que pueden ser modificadas de ese elemento.

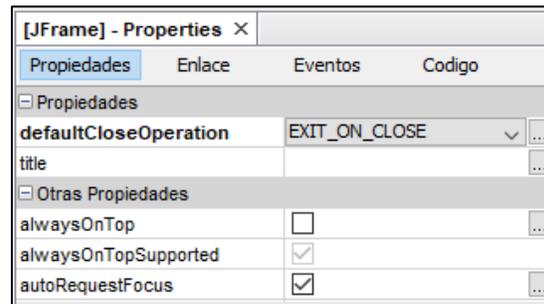


fig. 4.1.13. Ventana de propiedades.

Panel de salida: después de ejecutar el proyecto, este panel muestra los mensajes de error o de éxito de la compilación del programa.

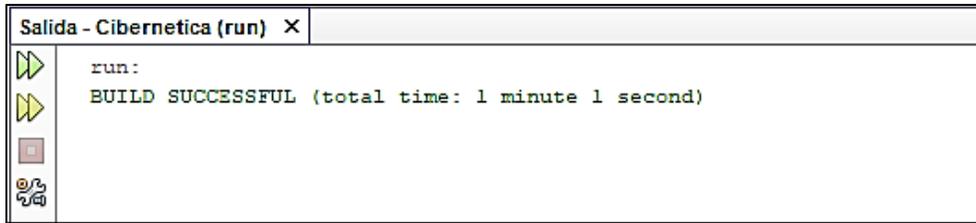


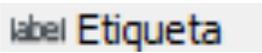
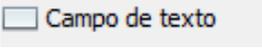
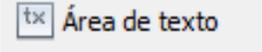
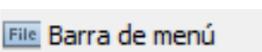
fig. 4.1.14. Ventana de salida.

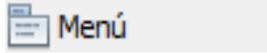
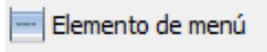
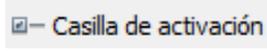
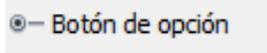
Actividad 3

Descripción de componentes.

Retomando el Proyecto **Cibernetica**, desarrollado en la Actividad 2, abre el Paquete **CiberII** y la Clase **Mensaje**. Después realiza lo siguiente:

1. Ir a la pestaña **Design**.
2. Ir a la **Paleta de componentes**.
3. Buscar el componente, encontrar la descripción correspondiente y relacionar las siguientes columnas.

| | Componentes | Descripción |
|--------|---|---|
| 1. () | JLabel  | a. Es un elemento individual de un menú. |
| 2. () | JButton  | b. Un contenedor para menús y elementos de menú. |
| 3. () | JTextField  | c. Un componente que combina un botón o campo editable y una lista desplegable. |
| 4. () | JTextArea  | d. Un área de visualización para una cadena de texto o una imagen, o ambas. |
| 5. () | JComboBox  | e. Un área de varias líneas que muestra texto sin adornos. |
| 6. () | JMenuBar  | f. Un elemento puede ser seleccionado o deseleccionado. Usado con un objeto ButtonGroup para crear un |

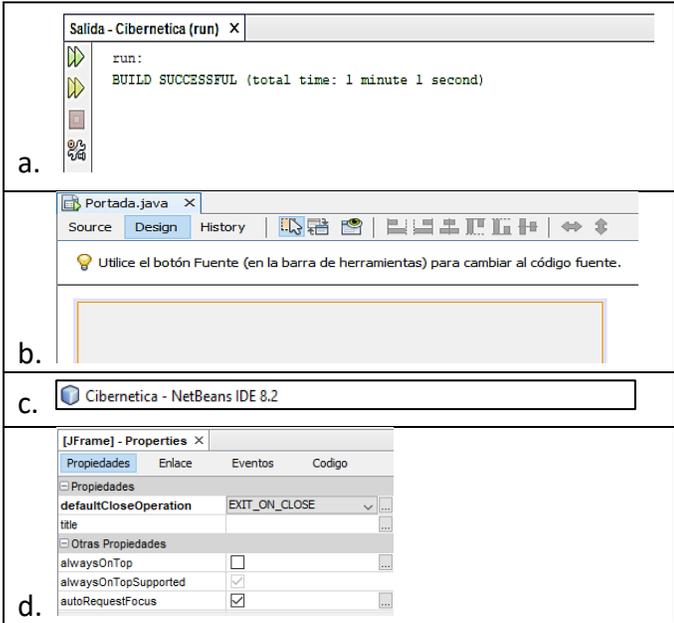
| | Componentes | Descripción |
|---------|---|--|
| | | grupo de botones en el cual un botón puede ser seleccionado a la vez. |
| 7. () | JMenu  | g. Un botón “oprimir” (para desencadenar una acción). |
| 8. () | JMenuItem  | h. Un elemento puede ser seleccionado o deseleccionado. Por convención, cualquier checkbox en un grupo puede ser seleccionado. |
| 9. () | JCheckBox  | i. Un menú para elementos de menú y submenús. |
| 10. () | JRadioButton  | j. Un componente ligero que permite editar una línea individual de texto. |

Actividad 4

Entorno de trabajo de NetBeans.

Relaciona las siguientes columnas.

1. Ventana de diseñador ()
2. Barra de título ()
3. Panel de salida ()
4. Barra de menús ()
5. Controles Swing ()
6. Propiedades de los elementos ()



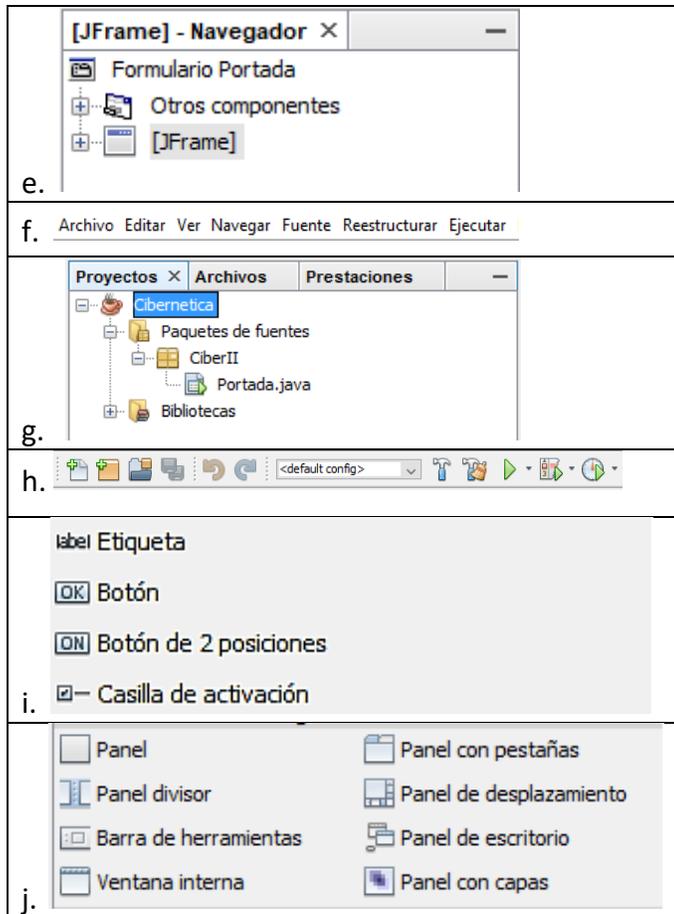
a.

b.

c.

d.

- 7. Lista de proyectos ()
- 8. Navegador de componentes ()
- 9. Barra de herramientas ()
- 10. Contenedores Swing ()



4.1.4 JFrame – Formulario

Hoy en día, casi todo el software está basado en ventanas que contienen una barra de título, límites, un botón para minimizar, otro para cerrar, la función de modificar el tamaño, así como mensajes y botones para oprimir y realizar una acción. Estas y otras características pueden implementarse con la clase JFrame.

Actividad 5

Diseño de un formulario.

Realiza los siguientes pasos para diseñar una ventana de 600 X 300, que no puede modificar su tamaño, con fondo verde y título Mensaje, como la que se muestra en la figura 4.1.15



fig. 4.1.15. Salida del formulario.

- a. Abrir NetBeans, seleccionar el menú **Archivo** y la opción **Abrir proyecto**.

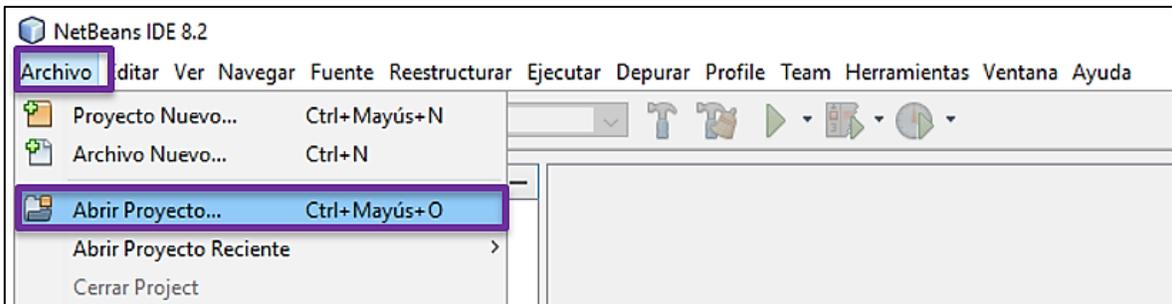


fig. 4.1.16. Abrir proyecto.

- b. En la ventana seleccionar la unidad donde se guardó el proyecto y dar clic en **Abrir proyecto**.

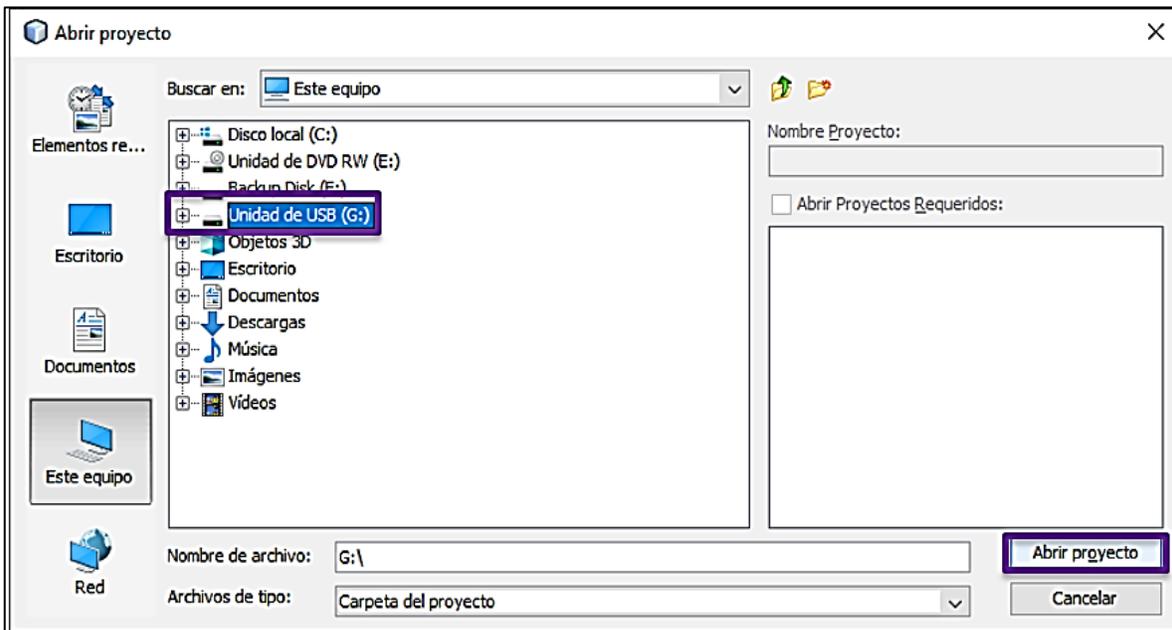


fig. 4.1.17. Ubicación de proyecto.

- c. Seleccionar el proyecto **Cibernetica** y dar clic en **Abrir proyecto**.

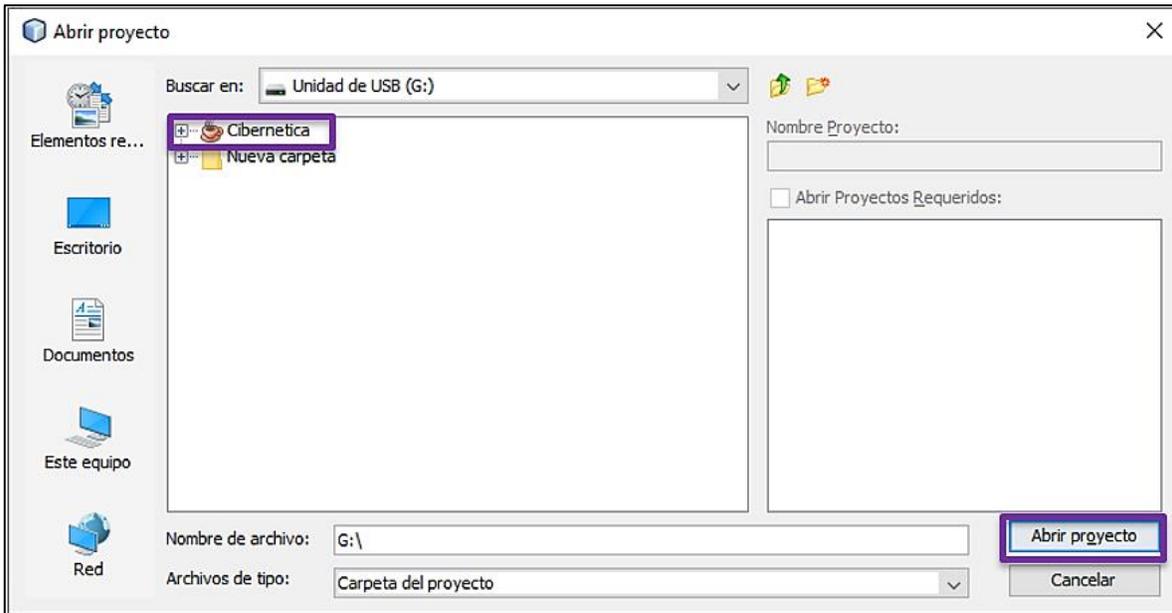


fig. 4.1.18. Selección de proyecto.

- d. Se abre la siguiente ventana y dar clic en la pestaña **Design** para diseñar el formulario con las características solicitadas.

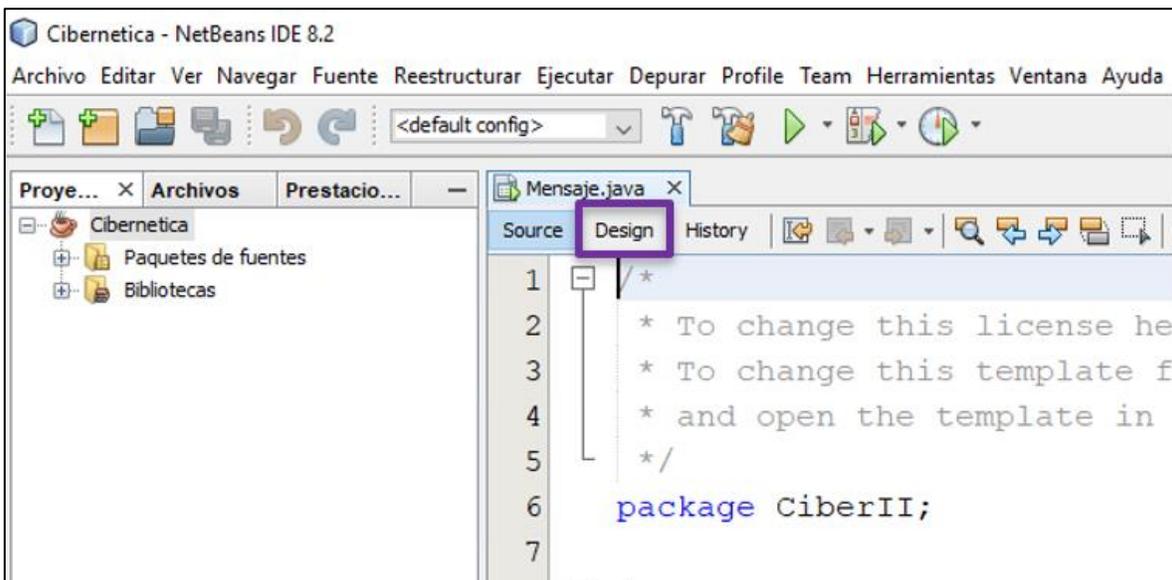


fig. 4.1.19. Proyecto abierto.

- e. En la propiedad **defaultCloseOperation** verificar la opción EXIT_ON_CLOSE para que la ventana se cierre y se termine el programa.

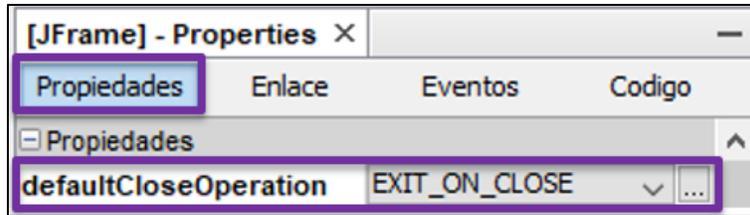


fig. 4.1.20. Propiedades de formulario.

- f. En la opción **title** escribir el título **Mensaje** que va a tener la ventana.

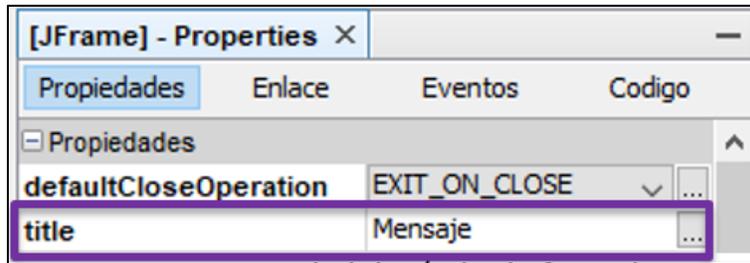


fig. 4.1.21. Propiedad de título de formulario.

- g. Buscar la opción **preferredSize** para cambiar el tamaño de la ventana indicando la anchura y la altura de **600 X 300**.

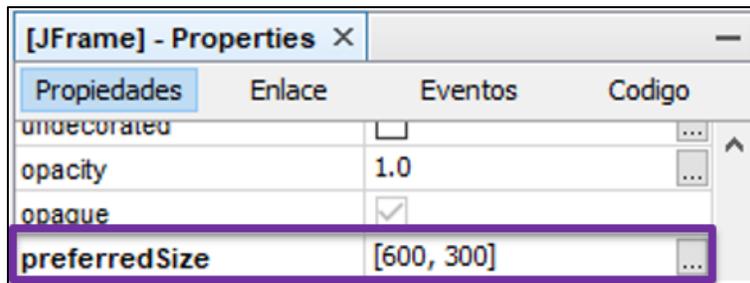


fig. 4.1.22. Propiedad de tamaño de formulario.

- h. Desactivar la casilla de la propiedad **resizable** para que no se pueda modificar las dimensiones de la ventana.

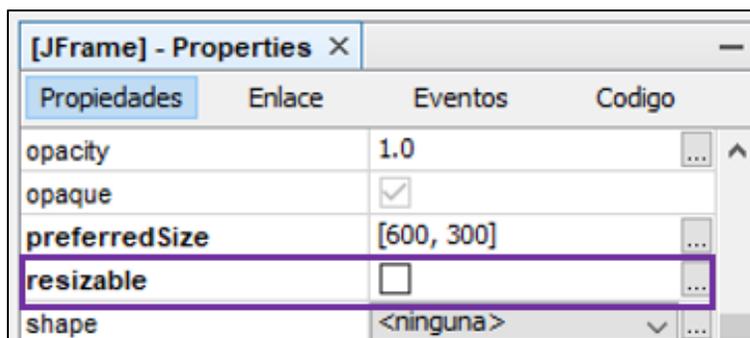


fig. 4.1.23. Propiedad de redimensionar formulario.

- i. Para cambiar el color de fondo del JFrame se va a emplear la librería `java.awt.Color` la cual contiene la clase `Color` con los valores que se pueden utilizar. Para ello dar clic en la pestaña **Source**, como se indica en la figura 4.1.24 y agregar la siguiente línea de código:

```
import java.awt.Color;
```

Después escribir la siguiente sentencia dentro del método constructor:

```
this.getContentPane().setBackground(Color.green);
```

Con esta sentencia se hace referencia al método para obtener el contenedor (`getContentPane`) y modificar su atributo de fondo (`setBackground`) con un atributo de la clase `Color` (`Color.green`) que puede tomar los valores: `black`, `blue`, `cyan`, `dark_gray`, `gray`, `green`, `lightGray`, `magenta`, `orange`, `pink`, `red`, `white` y `yellow`.

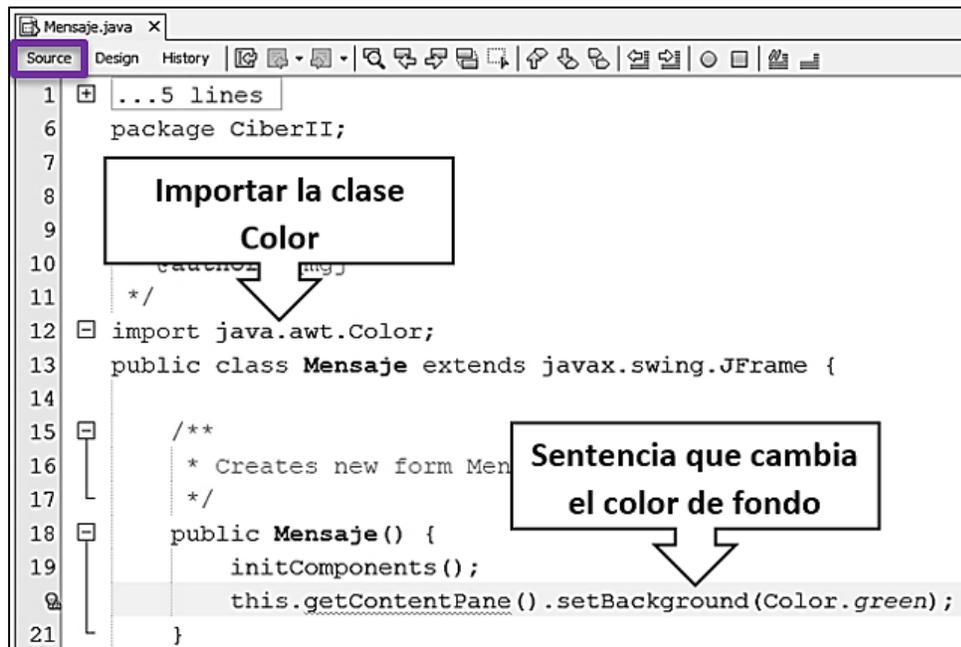


fig. 4.1.24. Sentencias para cambiar el color de fondo del formulario.

- j. Ejecutar el programa dando clic con el botón derecho del mouse sobre el archivo y seleccionar **Ejecutar archivo**.

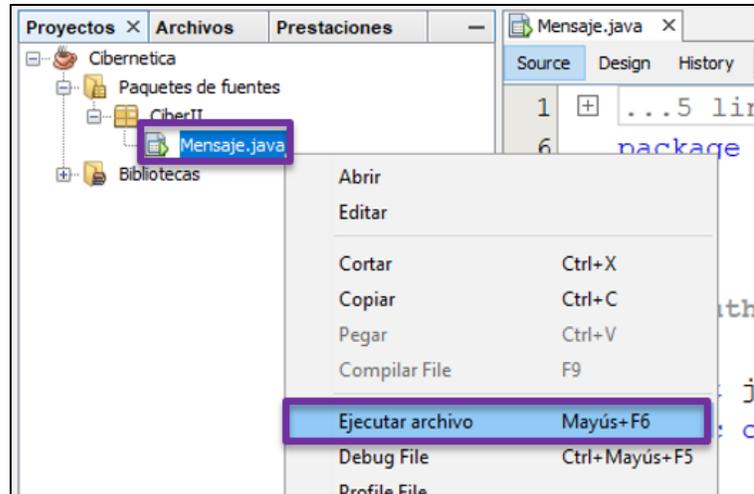


fig. 4.1.25. Ejecución del programa.

4.1.5 JLabel - Etiqueta



fig. 4.1.26 Etiqueta.

El componente **JLabel** es una etiqueta en la que podemos escribir texto que puede servir como título o descripción, el usuario no lo puede modificar. Adicionalmente puede utilizarse para mostrar una imagen.

Actividad 6

Insertar una etiqueta

Utilizando el formulario de la actividad 5 realiza los siguientes pasos para insertar una etiqueta con la frase **Hola Mundo** como se muestra en la figura 4.1.27.



fig. 4.1.27. Salida del mensaje Hola Mundo.

- a. Seleccionar la pestaña **Design** y en la paleta elegir el componente **etiqueta** de los **Controles Swing** y colocarlo en un lugar del formulario.

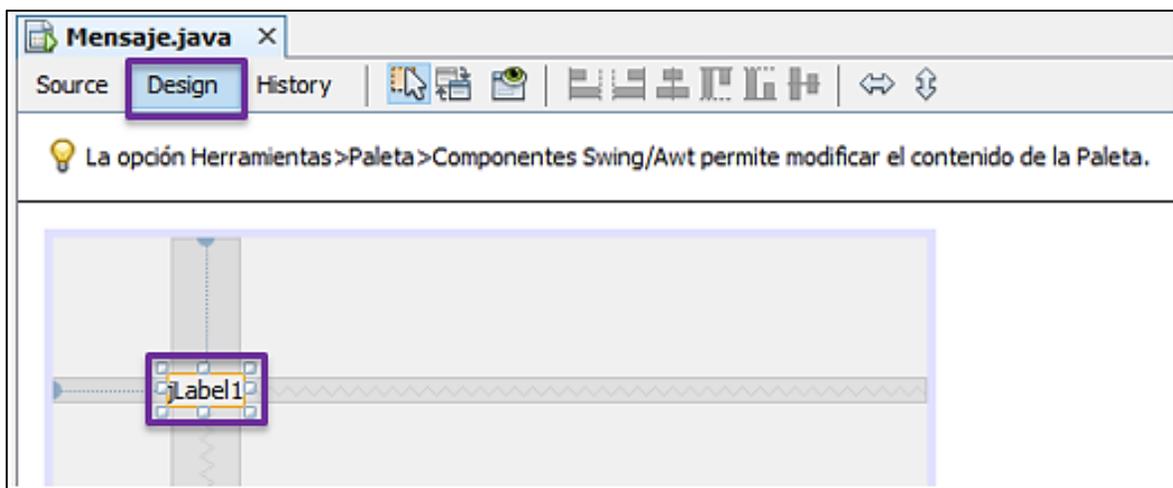


fig. 4.1.28. Insertar etiqueta.

- b. Dar clic con el botón derecho del mouse sobre la etiqueta insertada y seleccionar **Propiedades** para ver y editar sus características, o también en la sección de Propiedades de la pantalla de trabajo. Las propiedades de la etiqueta se muestran en la figura 4.1.29.

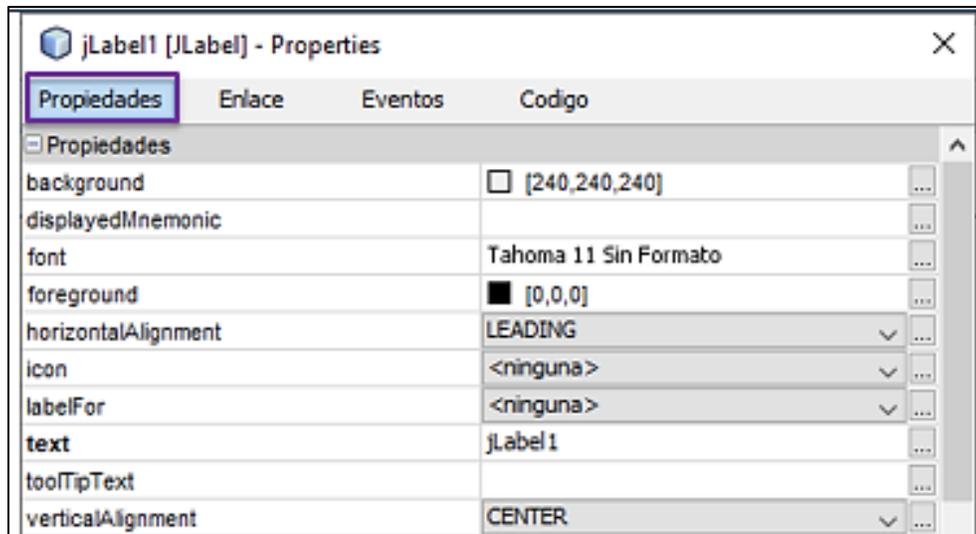


fig. 4.1.29. Propiedades de la etiqueta.

- c. Elegir la opción **font** para cambiar el tipo, estilo y tamaño de la letra.

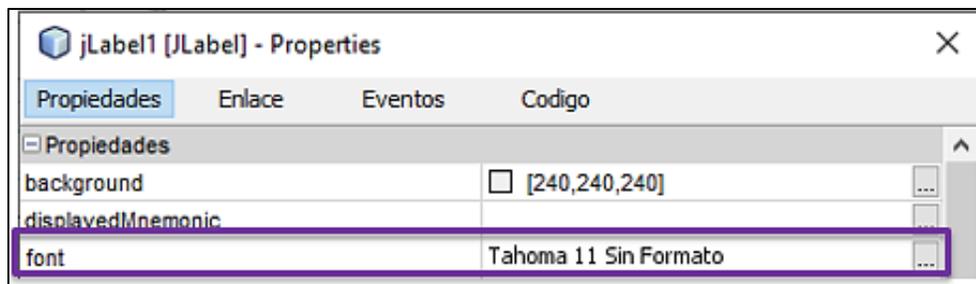


fig. 4.1.30. Propiedades de tipo, estilo y tamaño de la etiqueta.

- d. Dar clic sobre el botón que tiene los tres puntos [...] y se muestra la ventana con los tipos, estilos y tamaños de fuente; elegir Tahoma, Negrita, 18 y dar clic en **Aceptar**.

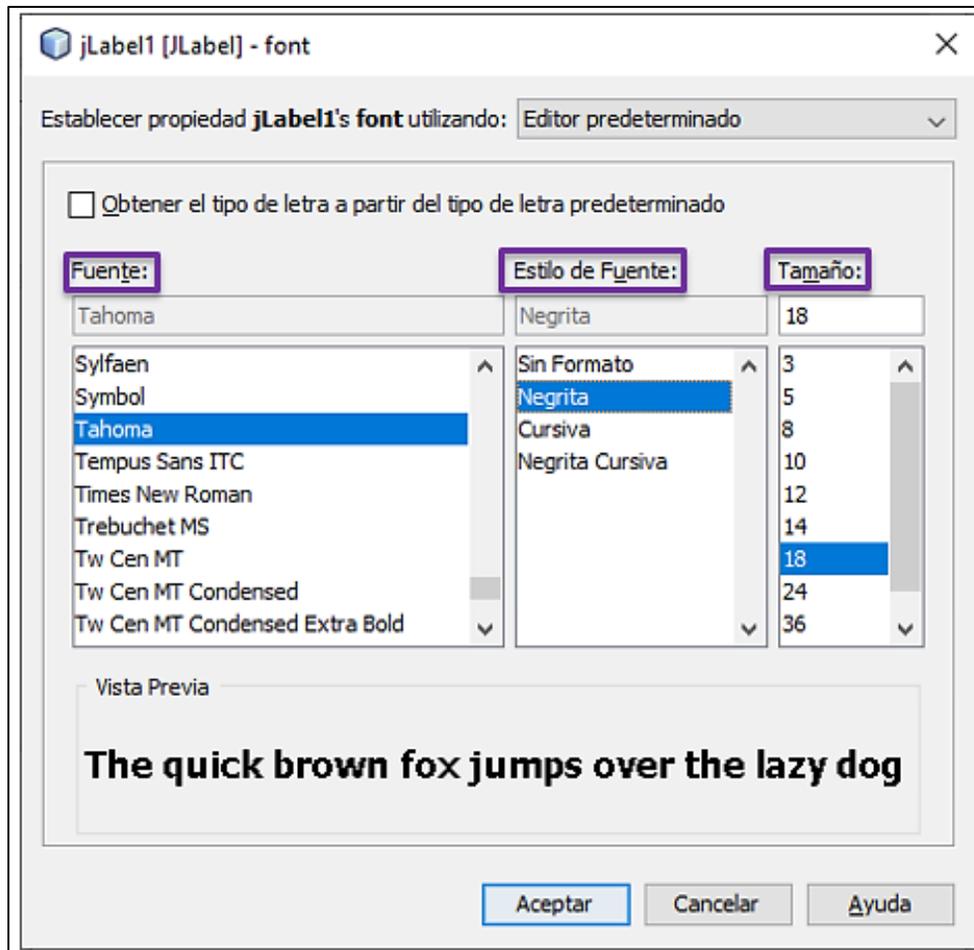


fig. 4.1.31. Selección de fuente, estilo y tamaño.

- e. Elegir la opción **foreground** para cambiar el color de la letra.

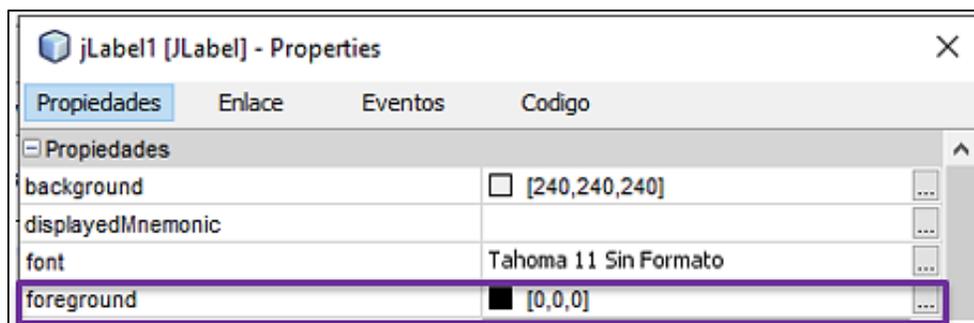


fig. 4.1.32. Propiedad de color de la etiqueta.

- f. Dar clic sobre el botón que tiene los tres puntos [...] se muestra la ventana con la paleta de colores. Elegir el color azul y dar clic en **Aceptar**.

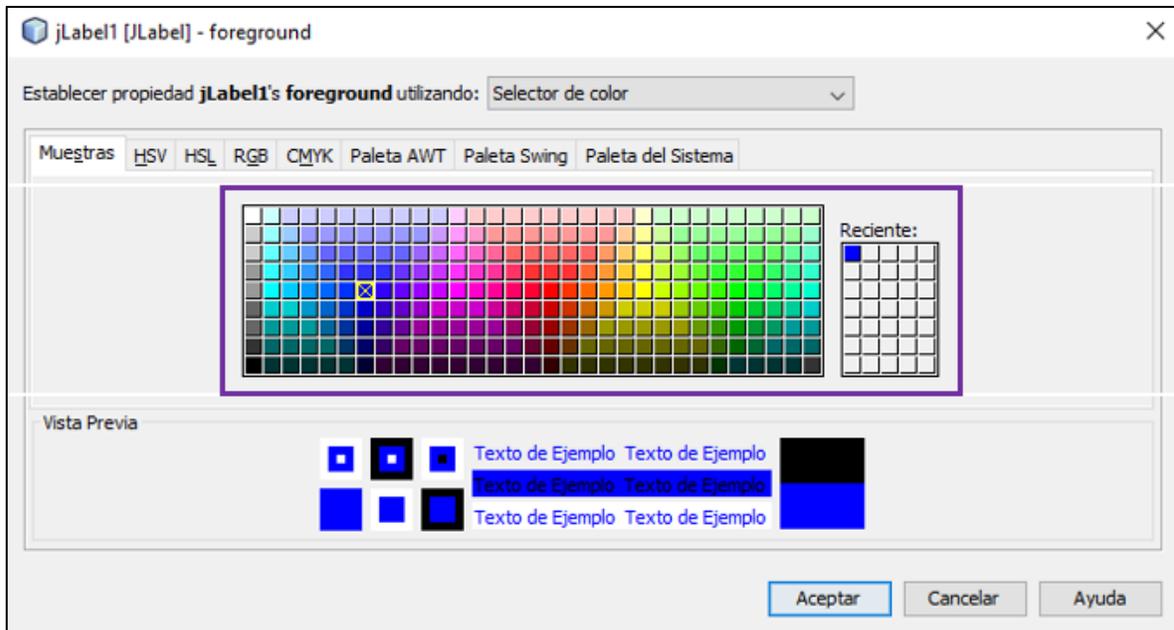


fig. 4.1.33. Selección de color de la etiqueta.

- g. Elegir la opción **text** para cambiar el texto que aparece en la etiqueta. Escribir el mensaje **Hola Mundo**.



fig. 4.1.34. Propiedad de texto de la etiqueta.

- h. Ejecutar el programa con la combinación de las teclas Mayús + F6.

4.1.6 JButton - Botón



fig. 4.1.35. Botón

JButton es un componente que provee un botón en el que el usuario hace clic para ejecutar una acción. Se pueden utilizar varios tipos de botones dentro de un programa, según la acción específica que se requiera.

Un botón genera un evento (ActionEvent) cuando el usuario hace clic en él y se debe programar para que realice la acción deseada.

Actividad 7

Insertar botones.

Utilizando el formulario de la actividad 5 realiza los siguientes pasos para insertar los botones **Limpiar** que debe borrar el texto Hola Mundo, y **Salir** para cerrar el programa, como se muestra en la figura 4.1.36.



fig. 4.1.36. Salida del programa con botones.

- a. Seleccionar el componente **botón** de la paleta, y colocar dos botones en un lugar del formulario.

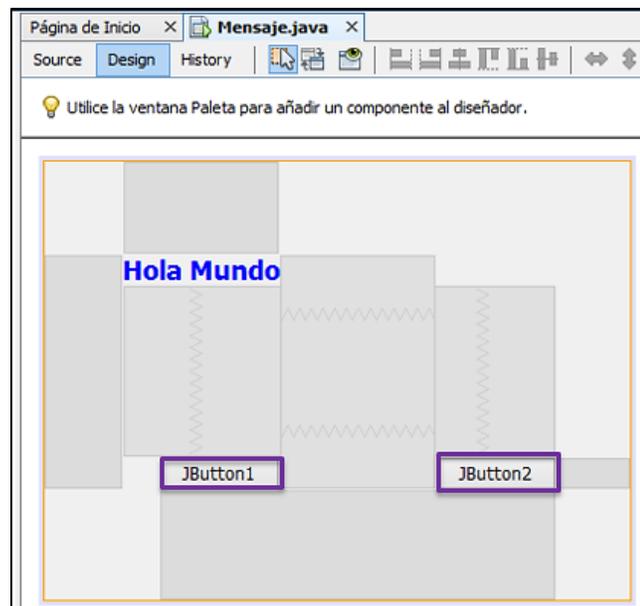


fig. 4.1.37. Insertar botones.

- b. Cambiar las propiedades de texto, letra y color de la letra, con los valores Limpiar, Tahoma 18, Negrita, azul, desde las **Propiedades** dando clic con el botón derecho del mouse sobre el JButton1. La figura 4.1.38 muestra todas las propiedades para el botón **Limpiar**.

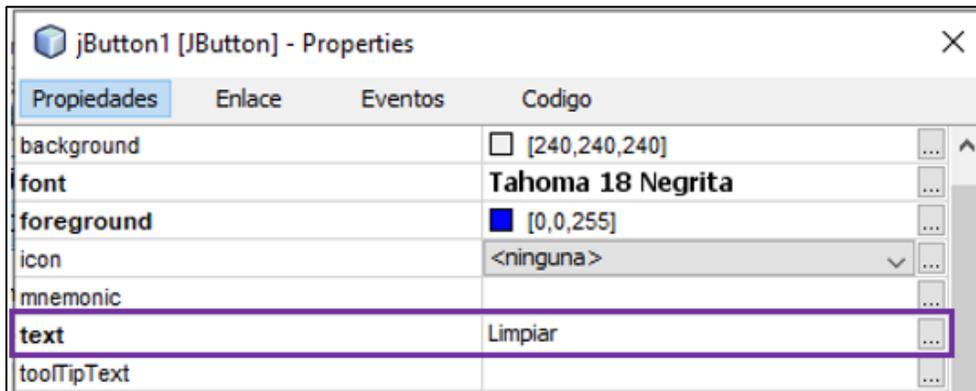


fig. 4.1.38. Propiedades del botón Limpiar.

- c. Seleccionar la pestaña **Código** y cambiar el nombre a la variable por **limpiar**.



fig. 4.1.39. Propiedad de nombre de variable del botón Limpiar.

- d. Dar doble clic sobre el botón **Limpiar**, a continuación, se abre la pantalla de código mostrándonos el método para programar la acción que va a realizar el botón. Para la acción de limpiar, la línea de código que se debe agregar es la siguiente:

```
this.jLabel1.setText("");
```

Lo que hace esta sentencia es seleccionar la etiqueta jLabel1 y cambiar el texto por una cadena vacía (""). El código final es el siguiente:

```
private void limpiarActionPerformed(java.awt.event.ActionEvent evt) {
    this.jLabel1.setText("");
}
```

- e. Regresar a la pestaña **Design** y cambiar las propiedades de letra, color y texto del botón **Salir**, por Tahoma 18 Negrita como se muestra en la figura 4.1.40.

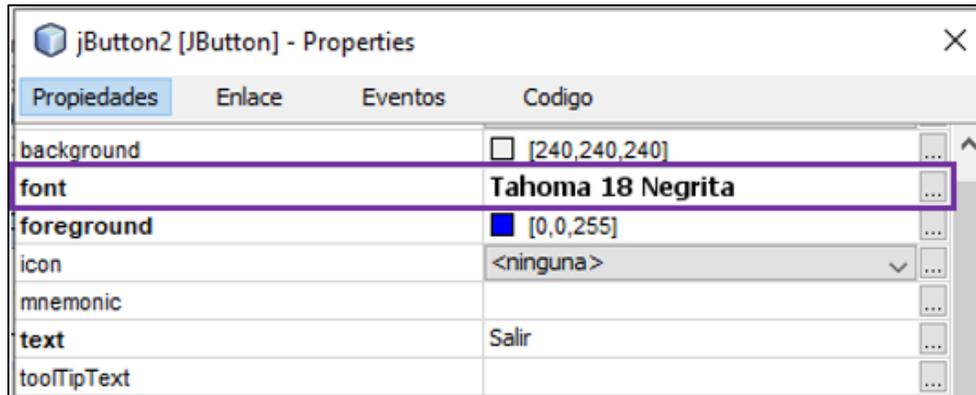


fig. 4.1.40. Propiedades del botón Salir.

- f. En la pestaña de **Código** cambiar el **Nombre de variable** por **salir**.

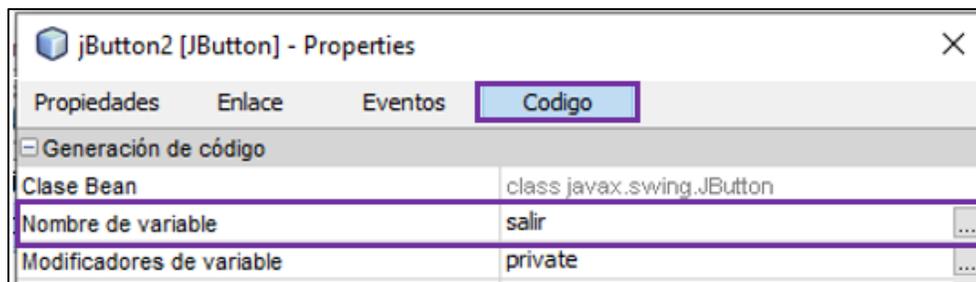


fig. 4.1.41. Propiedad de nombre de variable del botón Salir.

- g. Dar doble clic sobre el botón **Salir** y en el método que muestra escribir la siguiente línea de código.

System.exit(0);

Lo que hace esta sentencia es cerrar la ventana. El código final es el siguiente:

```
private void salirActionPerformed(java.awt.event.ActionEvent evt) {
    System.exit(0);
}
```

- h. Ejecutar el programa con la combinación de las teclas Mayús + F6.

Desafío 1

Diseño de formulario.

Diseñar un nuevo formulario que se llame **Bienvenida** con las siguientes características:

- La ventana debe tener el título de **Bienvenida**.
- El color de fondo debe ser **cyan**.
- Las dimensiones son de **500 X 200**.
- No se puede cambiar de tamaño del formulario.
- Contener una etiqueta que diga **Hola...** con letra Monotype Cursiva número 24 y color azul.
- Debe tener cinco botones con los mensajes de **Limpiar**, **Restablecer**, **Saludar**, **Cambiar** y **Salir** con la misma letra de la etiqueta y color rojo.
- Al pulsar el botón saludar la etiqueta que debe decir hola y su nombre, por ejemplo: **Hola José**.
- Al pulsar el botón restablecer debe mostrar el mensaje inicial **Hola...** y el color inicial **cyan** de fondo.
- Al pulsar el botón limpiar debe borrar todo el mensaje.
- Al pulsar el botón cambiar debe cambiar el fondo a color **rosa**.
- Al pulsar el botón salir debe cerrar la ventana.

El resultado esperado se muestra en la figura 4.1.42.



fig. 4.1.42. Salida del programa Bienvenida.

Al pulsar el botón **Saludar** aparece el mensaje Hola José, como muestra la figura 4.1.43.



fig. 4.1.43. Ejecución del botón Saludar.

Al pulsar el botón **Limpiar** se borra el mensaje, mostrado en la figura 4.1.44.



fig. 4.1.44. Ejecución del botón Limpiar.

Al pulsar el botón **Cambiar** el fondo pasa a color rosa.

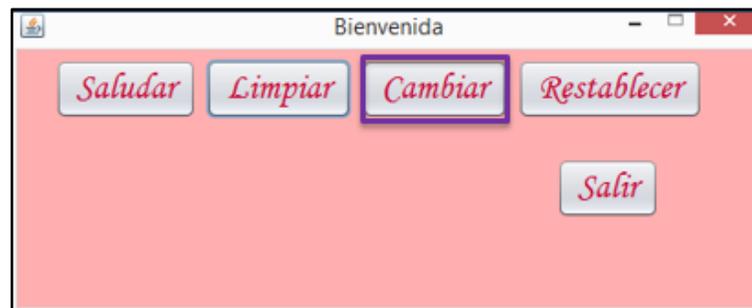


fig. 4.1.45. Ejecución del botón Cambiar.

Al pulsar el botón **Restablecer** aparece el mensaje y el color iniciales.



fig. 4.1.46. Ejecución del botón Restablecer.

Al pulsar el botón **Salir** se cierra la ventana.

4.1.7 Iniciando un proyecto

Se desarrollará un proyecto para explicar el funcionamiento de cada uno de los componentes, elaborando un mapa mental como el que se muestra a continuación.

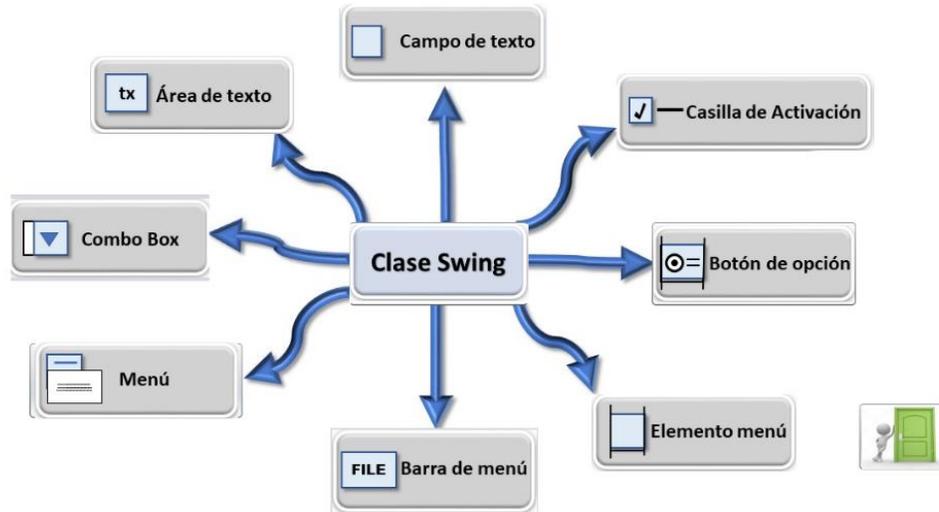


fig. 4.1.47. Mapa mental

La finalidad es que cada imagen de los componentes tenga la funcionalidad de un botón que, al dar clic, aparezca una ventana con su descripción y un ejemplo de uso. Para ello se realizarán las siguientes actividades.

Actividad 8

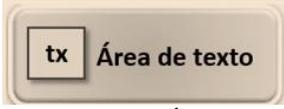
Imágenes

Creación de imágenes para los botones

1. Crea una carpeta con el nombre **botones** en tu memoria USB.
2. Diseña las imágenes en un programa de edición de imagen (Paint, Gimp o equivalente) los botones con tu propio estilo para los componentes que se visualizan en el mapa mental que se muestra en la figura 4.1.44
 - a. El tamaño de la imagen debe ser 235 x 90 píxeles.
 - b. El formato de la imagen debe ser png
 - c. El nombre de la imagen debe corresponder con el nombre del componente, por ejemplo:
 area_texto.png; campo_texto.png; casilla_activacion.png;
 boton_opcion.png; combo_box.png; barra_menu.png; menu.png;
 elemento_menu.png y clase_swing.png

Duplica las imágenes para los botones, modifica el color y guárdalos con el mismo nombre agregando el número 2 para diferenciar cada grupo de imágenes.

Ejemplo:

| Imagen | Nombre |
|--|-----------------|
|  fig. 4.1.48. Cuadro Área de texto | area_texto.png |
|  fig. 4.1.49. Cuadro Área de texto 2 | area_texto2.png |

3. Diseña la plantilla utilizando el editor de imagen que elegiste anteriormente para el mapa mental, como se muestra a continuación:
 - a. El tamaño de la imagen debe ser 1280 x 720 pixeles y cada rectángulo punteado 235 x 90.

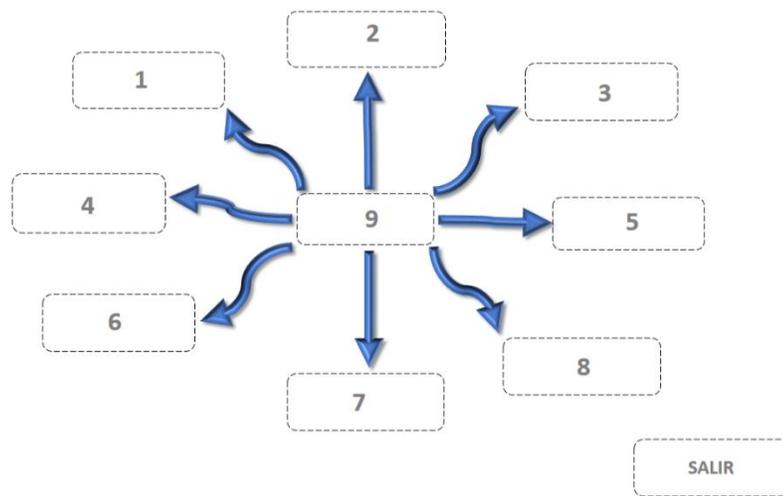


fig. 4.1.50. Plantilla de mapa mental.

4. Busca en internet o crea los botones para salir de la aplicación.
 - a. El tamaño de la imagen debe ser 235 x 90 pixeles.
 - b. El formato de la imagen debe ser png

| Imagen | Nombre Imagen |
|---|---------------|
|  | salir.png |

fig. 4.1.51. Botón salir



salir2.png

fig. 4.1.52. Botón salir 2

4.1.8 Elaborando la plantilla en NetBeans

En la siguiente actividad, se revisarán algunas propiedades de los componentes que ya conoces como el JFrame, JLabel y JButton, las cuales servirán como base para la creación de la plantilla del Mapa Mental.

Actividad 9

Creación del Mapa mental

1. Crea un **Proyecto Nuevo** en NetBeans con el nombre **InterfazGrafica** y guárdalo en tu memoria USB.
2. Mueve tu carpeta de **botones** en la subcarpeta **src** del proyecto generado.

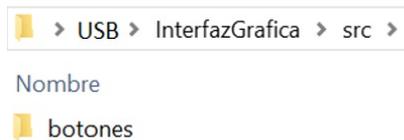


fig. 4.1.53. Carpeta Botones

3. En la carpeta *Paquete de fuentes* y en el paquete *interfazgrafica* coloca un nuevo formulario JFrame.



fig. 4.1.54. Carpeta Interfazgrafica.

Con el siguiente formato (propiedades):

Tamaño: 1300 x 800

Nombre del formulario: **MapaMental**

4. Para cambiar el formulario a **diseño nulo**, realiza lo siguiente:
 - a. Sobre el formulario dar clic con el botón derecho para visualizar el siguiente menú, en donde se elige *Activar gestor de distribución* y seleccionar *Diseño Nulo*.

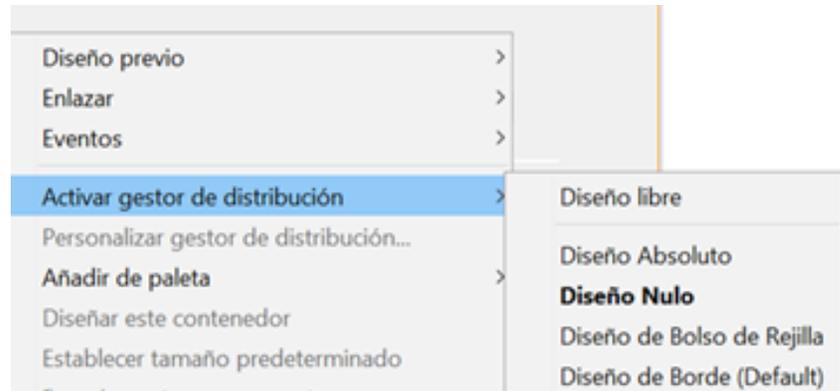


fig. 4.1.55. Menú de gestor de distribución.

- b. En la ventana Navegador se podrá visualizar la propiedad agregada al JFrame como se muestra en la siguiente figura.

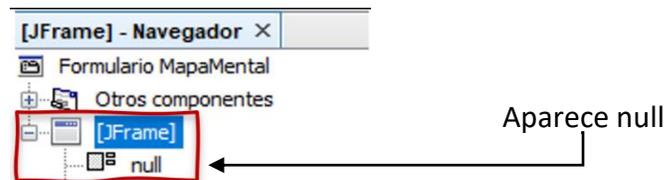


fig. 4.1.56. Propiedad diseño nulo

Esto es para poder sobreponer componentes, en este caso servirá para colocar los botones en los lugares correspondientes sobre la plantilla.

- 5. Coloca una etiqueta en el extremo superior derecho del formulario

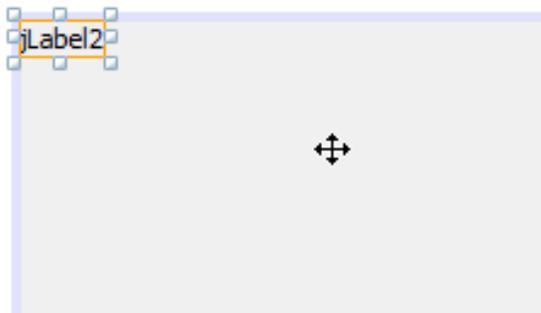


fig. 4.1.57. Insertar etiqueta

En esta etiqueta se insertará la Plantilla, para ello, en la propiedad *icon* se selecciona la imagen.

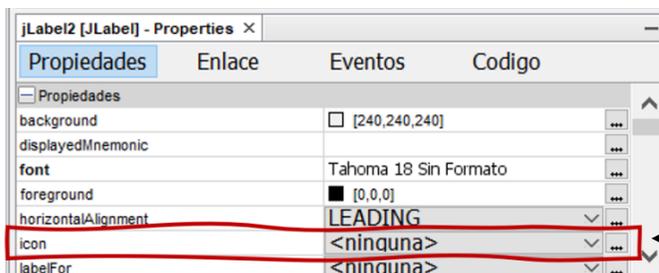


fig. 4.1.58. Propiedad *icon* de una etiqueta

Se mostrará la siguiente ventana:

- b. En el menú desplegable *Paquete* se selecciona *interfazgrafica* y después la carpeta **botones** que contiene las imágenes de los botones.

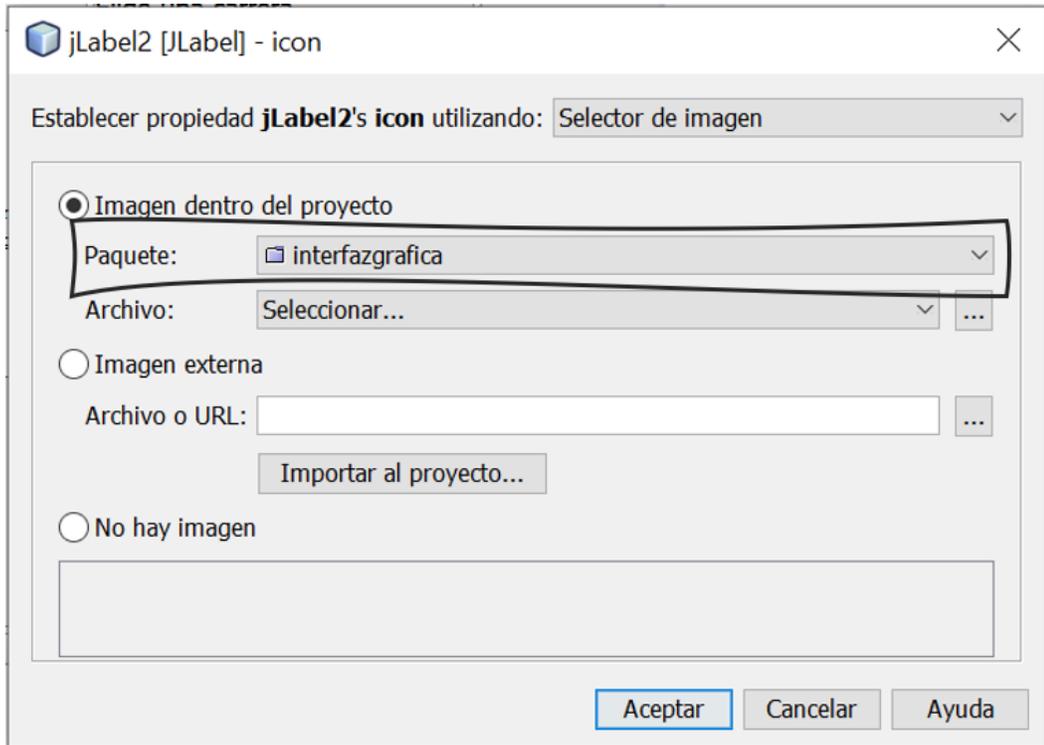


fig. 4.1.59. Selección interfazgrafica.

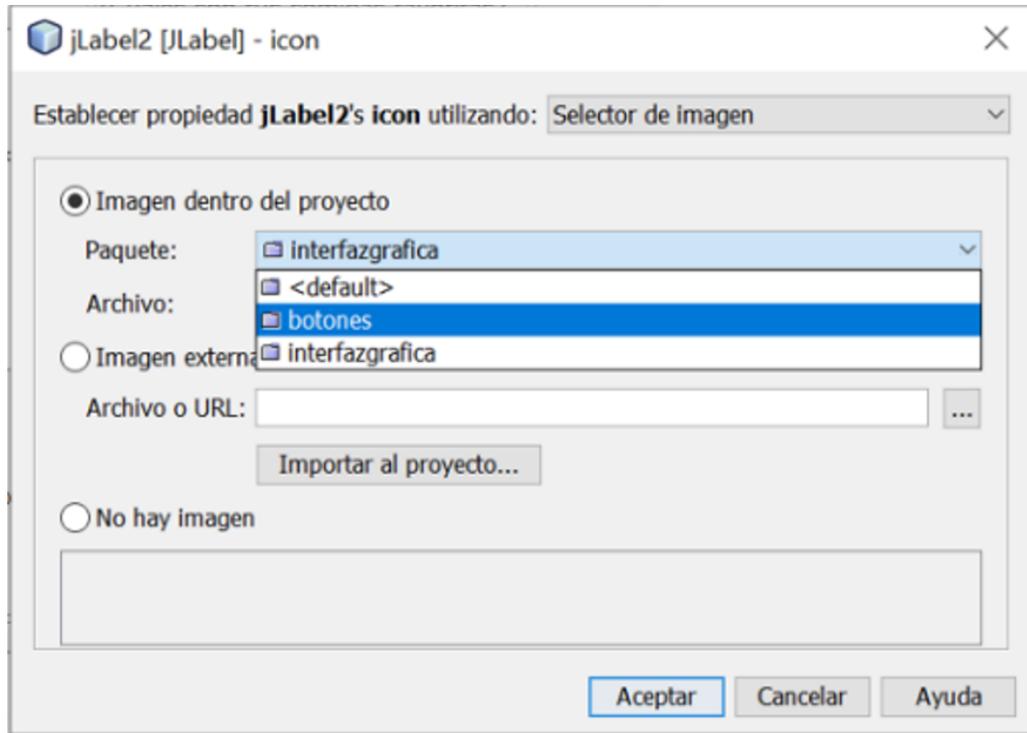


fig. 4.1.60. Selección botones.

- c. En el espacio de Archivo se da clic en *Seleccionar* para que se desplieguen los diferentes archivos que contiene la carpeta *Imágenes*.

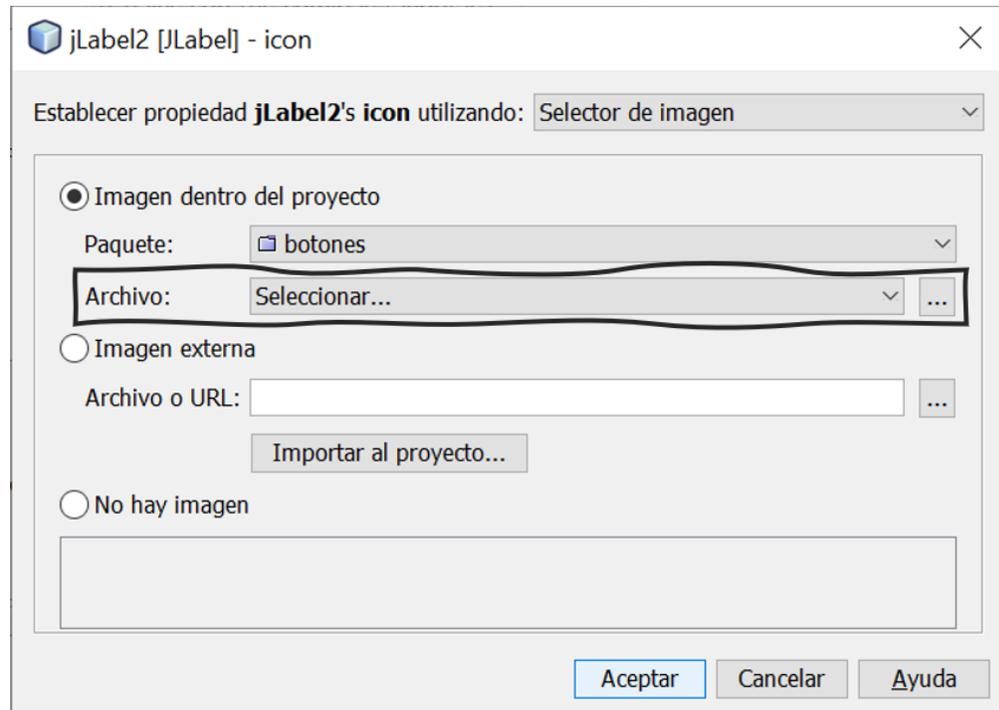


fig. 4.1.61. Propiedades para insertar imagen

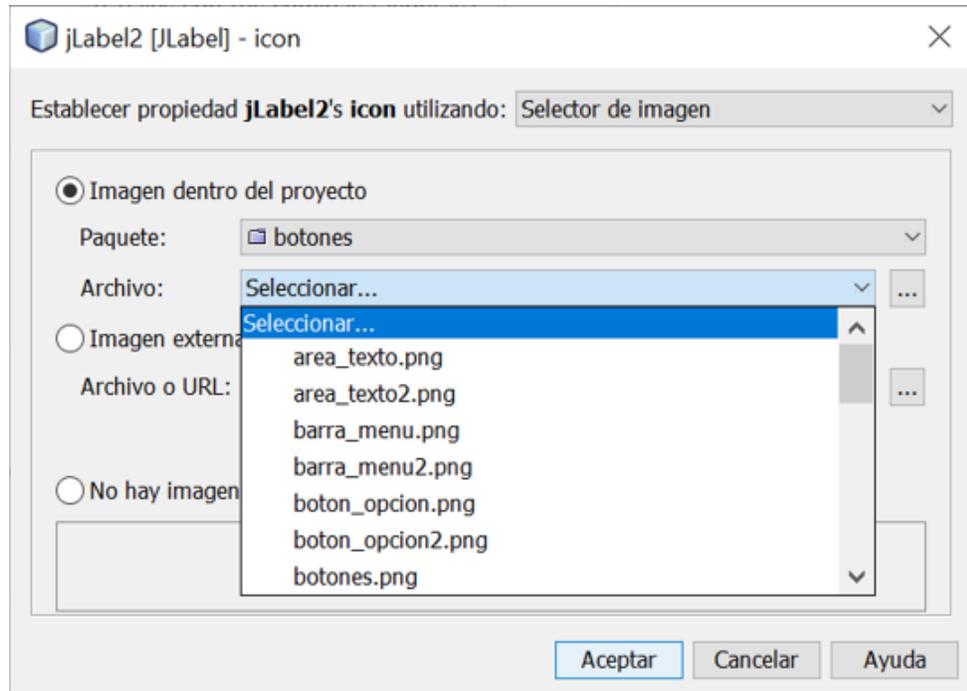


fig. 4.1.62. Selección de la imagen.

- d. Selecciona plantilla.png y da clic en Aceptar.

Aparece la siguiente imagen.

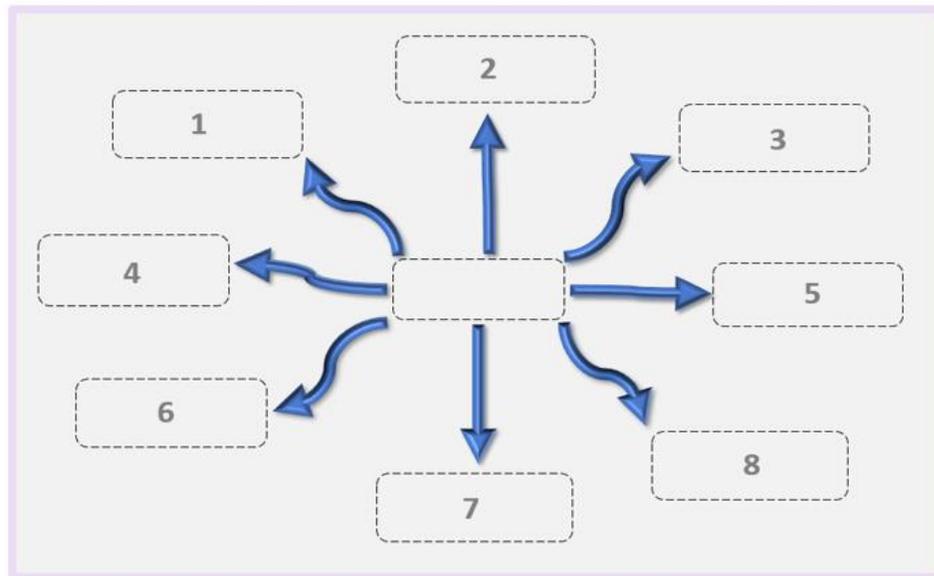


fig. 4.1.63. Imagen de la Platilla de mapa mental

- 6. Coloca un JButton en el espacio  de la plantilla.

- a. En la ventana del *Navegador* de componentes se visualiza

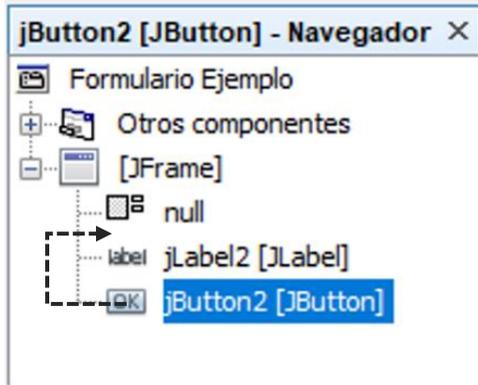


fig. 4.1.64. Ventana Componentes 1

Para que el botón se presente sobre la imagen se debe subir de nivel el componente hasta presentarse en un nivel superior de la etiqueta. Para ello, selecciona el jButton2 y arrastra hacia arriba, para visualizarlo.

Nota. El número del componente puede cambiar.

A continuación, se muestra cómo debe quedar:



fig. 4.1.65. Ventana Componentes 2

- b. Quitar el texto del botón, para ello se da clic derecho sobre él, y se selecciona editar texto para borrarlo.

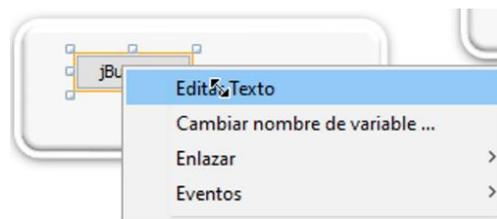


fig. 4.1.66. Editor de texto

- c. Ahora se redimensiona el botón al tamaño marcado de la plantilla, esto se puede realizar estirando uno de los extremos.

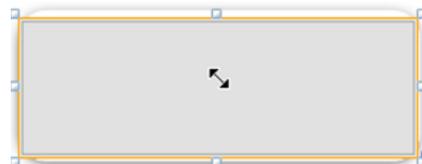


fig. 4.1.67. Cambio de tamaño plantilla

- d. Se continuará agregando la imagen indicada para el botón.

Selecciona el jButton2, en la ventana de Propiedades, buscar la opción

icon y selecciona

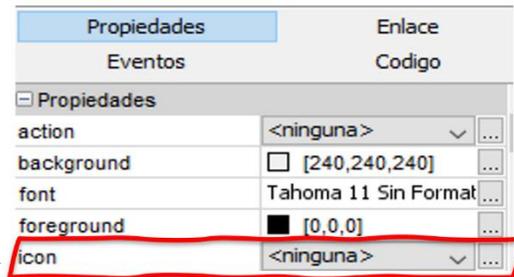


fig. 4.1.68. Propiedades botón

- e. En la siguiente ventana selecciona el paquete generado por la carpeta de imágenes y en archivo seleccionar la imagen del **Area_de_texto.png** y dar Aceptar.

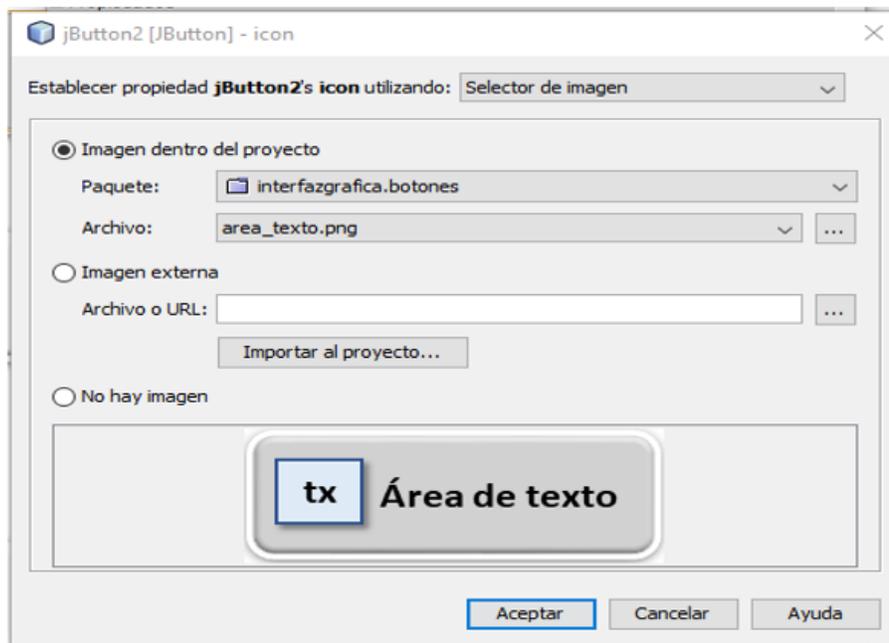


fig. 4.1.69. Selección Botón de área de texto.

A continuación, se muestra el resultado del procedimiento anterior:

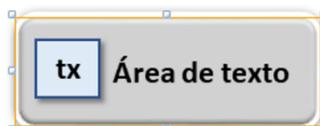


fig. 4.1.70. Botón área de texto.

- f. Ahora, para dar un efecto al botón se cambiará la propiedad **rollovericon** para cambiar la imagen cuando el mouse pase sobre el botón. Se selecciona la imagen del componente versión 2, en este caso Area_de_texto2.png

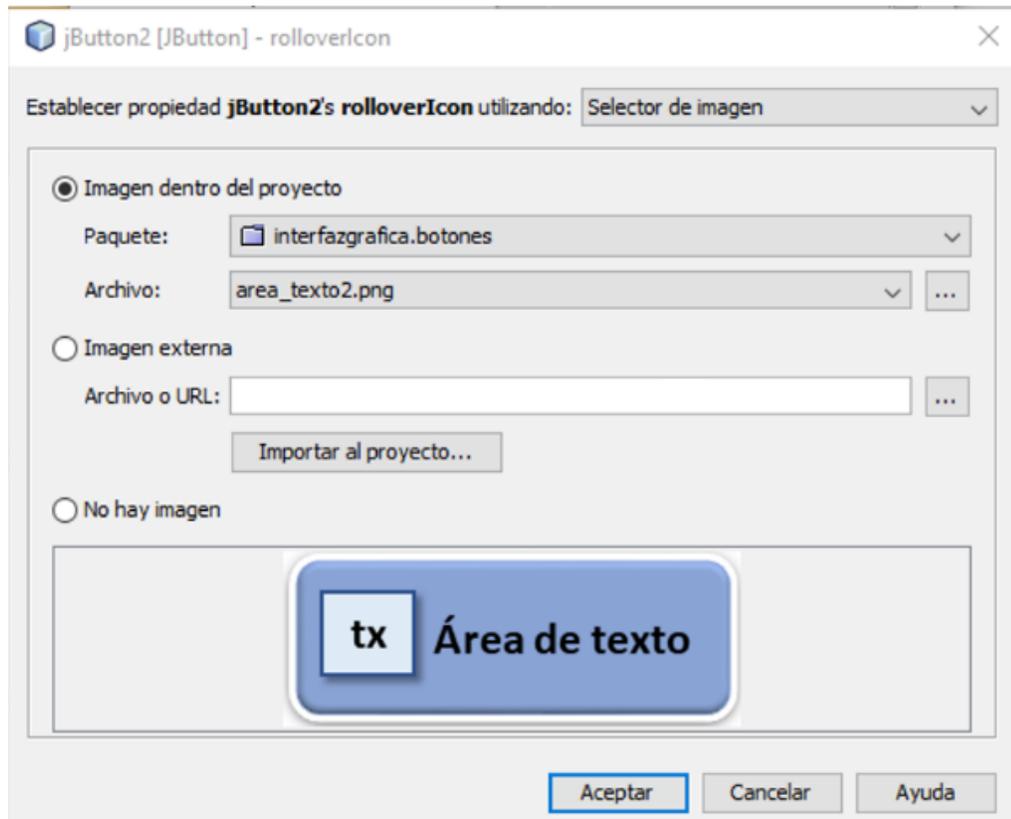


fig. 4.1.71. Selección Botón área de texto 2.

- 7. Agrega un botón en la parte inferior derecha para colocar en la propiedad **icon** sólo la imagen salir.png y en la propiedad **rollovericon** la imagen salir2.png

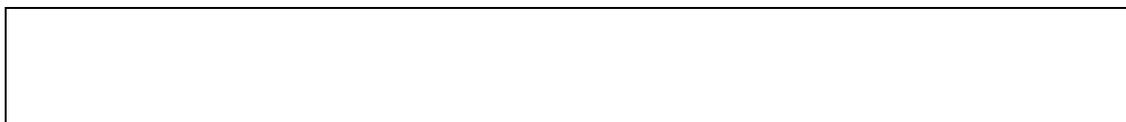


fig. 4.1.72. Botón salir



fig. 4.1.73. Botón Salir 2

- 8. Guarda el proyecto, imprime y pega la pantalla que aparece al ejecutar el programa.





9. Prueba el efecto del primer botón.

Pasa el mouse sobre el botón que se generó, ¿Qué se visualiza?

Actividad 10

Plantilla

1. Crea los botones faltantes de la plantilla en NetBeans según corresponda en la siguiente lista de componentes, haciendo el cambio de las propiedades para tener efecto sobre cada botón.

area_texto.png; campo_texto.png; casilla_activacion.png; boton_opcion.png;
 combo_box.png; barra_menu.png; menu.png; elemento_menu.png y clase_swing.png

2. Imprime y pega la pantalla que aparece al ejecutar el programa.



| |
|--|
| |
|--|

Actividad 11

Descripción.

Investiga la descripción de cada componente de la clase Swing.

| | |
|--------------------------------------|--|
| JTextField – Campo de Texto | |
| | |
| | |
| | |
| JTextArea – Área de texto | |
| | |
| | |
| | |
| JMenuBar – Barra de menú | |
| | |
| | |
| | |
| JMenuItem – Elemento de Menú | |
| | |
| | |
| | |
| JComboBox – Combo Box | |
| | |
| | |
| | |
| JCheckBox - Casilla de Activación | |
| | |
| | |
| | |
| JRadioBotton – Botón de opción | |
| | |
| | |
| | |

4.1.9 JTextArea – Área de texto



JTextArea – Área de texto

fig. 4.1.74. Área de texto

Este control nos permite ingresar o mostrar varias líneas de texto, su limitación es que sólo permite mostrar texto sencillo en el que se utilice un tipo de letra. Por ejemplo, un comentario, una sugerencia o cualquier otra cosa que lleve varias líneas, puede ser un texto amplio.

Este objeto se encuentra en la paleta de Controles Swing y la propiedad para ingresar o modificar el texto es **text**.

Actividad 12

Insertando y programando el botón del componente *Área de Texto*

Se hará uso del componente **Área de texto** para describir cada elemento, ya que su funcionalidad es contener texto. En este caso, contendrá su descripción.

Continuaremos con la construcción del Proyecto InterfazGrafica

1. Abrir el proyecto InterfazGrafica
 - a. Crear un formulario con el nombre del componente que se va a trabajar con la terminación Form (Formulario)
En este caso, será: *AreaTextoForm*

- b. En el formulario:
Se coloca una etiqueta y un área de texto, preferentemente distribuidos como se muestra.

La etiqueta se utilizará para el título, da doble clic en ella y escribe como texto el nombre del Componente.



fig. 4.1.75. Insertar etiquetas

- c. Se colocará la explicación o definición del componente dentro de un Área de Texto.

Da doble clic en el componente y escribe la descripción que anotaste de éste en la actividad 11

Se activa la propiedad *lineWrap* para que el texto se ajuste al área definida.

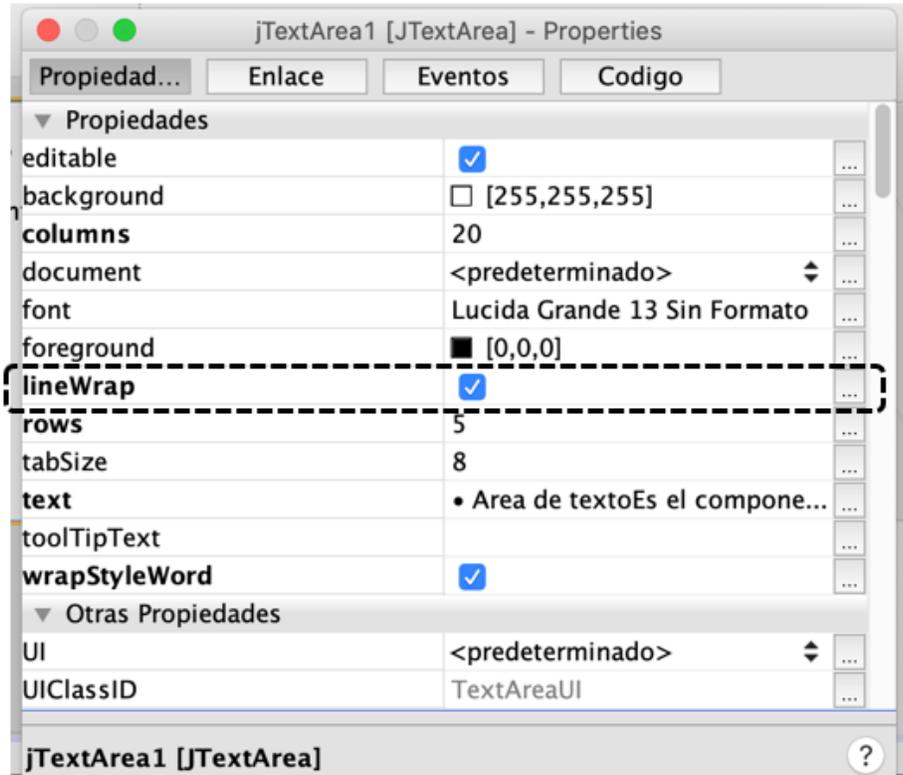


fig. 4.1.76. Propiedades para ajustar texto

- d. Coloca un botón que servirá para regresar al MapaMental. En este botón se modifica la propiedad *icon* para agregar la imagen clase_swing.png y en la propiedad *rollOverIcon*, coloca la imagen clase_swing2.png

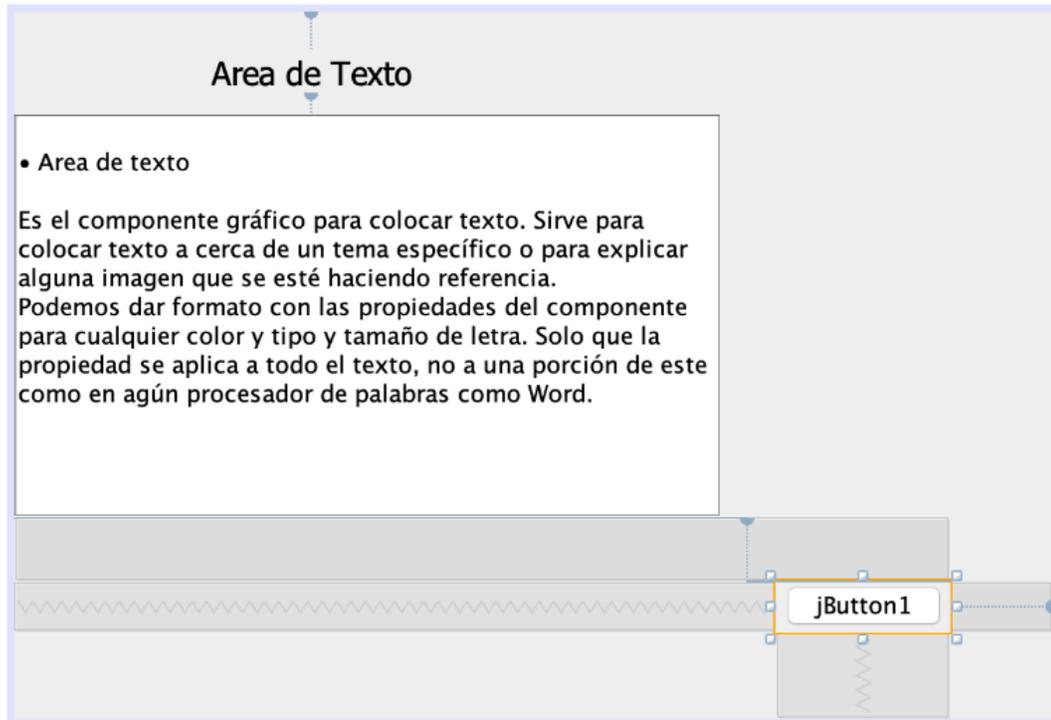


fig. 4.1.77. Agregar botón clase_swing

- e. Código para el botón de regreso.

Da doble clic en el botón para dirigirnos al código y agregar lo que aparece entre llaves { }. El nombre del componente *jframe1* puede cambiar.

```
private void btnClaseSwingActionPerformed(java.awt.event.ActionEvent evt) {
    MapaMental jframe1 = new MapaMental();
    jframe1.setVisible(true);
    this.setVisible(false);
}
```

- f. Para centrar el formulario en la pantalla basta con agregar la línea de código que aparece debajo de *initComponentes()*;

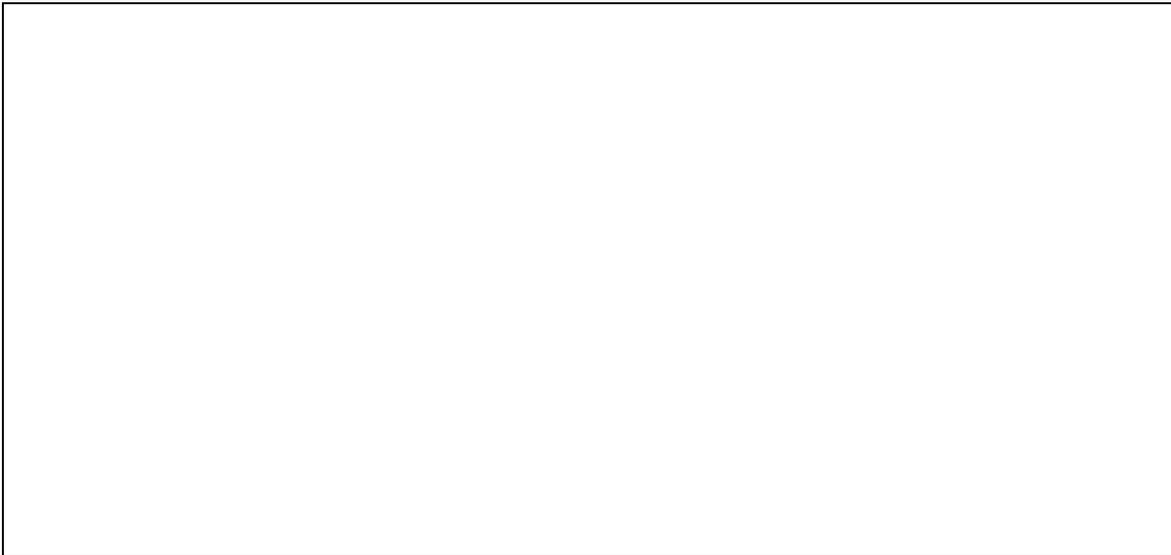
```
public AreaTextoForm() {
    initComponents();
    this.setLocationRelativeTo(null);
}
```

- g. Ahora, se agregará el funcionamiento del botón Área de texto del Mapa mental para dirigirnos al formulario correspondiente.

En el formulario MapaMental, da doble clic en el botón *Área de texto* y agrega el código que aparece entre llaves { }.

```
private void btnAreaTextoActionPerformed(java.awt.event.ActionEvent evt) {
    AreaTextoForm jframe1 = new AreaTextoForm();
    jframe1.setVisible(true);
    this.setVisible(false);
}
```

h. Imprime y pega la pantalla que aparece al ejecutar el programa.



4.1.10 Campo de Texto



fig. 4.1.78. Campo de texto

JTextField – Campo de texto

Este componente nos permite introducir datos para poderlos manipular. Cuando se introducen datos con este componente, estos ingresan como una cadena de caracteres *String* aunque representen valores numéricos, por ello es importante hacer la conversión pertinente cuando sea necesario.

Para que los datos introducidos sean tomados como números, se requiere hacer uso de las siguientes funciones para convertirlos a tipo numérico.

```
int num = Int.parseInt(jTextField1.getText( ));
float num = Float.parseFloat(jTextField1.getText( ));
double num = Double.parseDouble(jTextField1.getText( ));
```

En donde:

| | |
|--|--|
| num | Representa la variable que va a tomar el valor numérico. |
| textField1 | Representa el campo de texto en donde el usuario escribe el valor. |
| getText() | Método para acceder al valor escrito en el campo. |
| parseInt, parseFloat parseDouble | Métodos para convertir los valores del tipo <i>String</i> a los tipos numéricos correspondientes que generalmente son utilizados para realizar cálculos o para algún otro propósito. |

Asimismo, al obtener los resultados deseados y se tengan que mostrar se tiene que realizar el proceso inverso, convertir los cálculos numéricos a tipo texto *String* para mostrar su valor en una etiqueta *JLabel*, para ello se usa la siguiente función:

```
jLabelN.setText(Double.toString(resultado));
```

En donde:

| | |
|-----------|--|
| jLabelN | Representa el nombre de la etiqueta en donde se va a mostrar |
| setText | Función que asigna un valor del tipo String |
| Double | Clase para usar los métodos del tipo de dato double |
| toString | Función para convertir un número a formato texto. |
| resultado | Variable que contiene el resultado a mostrar |

Actividad 13

Insertando y programando el botón del componente *Campo de Texto*

Se desarrollará un programa en donde se utilice este componente para introducir dos números y realizar su suma a través de un botón.

1. Crear un formulario con el nombre: **CampoTextoForm** y agregar al formulario los siguientes componentes seis etiquetas, un área de texto, dos campos de texto y dos botones, distribuidos como se muestra a continuación:



fig. 4.1.79. Componentes de formulario

Las propiedades de texto correspondientes a cada componente son:

| Componente | Propiedad - Texto | Componente | Propiedad - Texto |
|-------------|-------------------|-------------|----------------------------|
| jLabel1 | Campo de Texto | jLabel4 | Número 2 |
| jLabel2 | Ejemplo | jLabel5 | Resultado |
| jLabel3 | Número 1 | jLabel6 | Aquí aparece el resultado |
| jTextField1 | | jTextField2 | |
| jButton1 | + | jTextArea | Descripción del componente |
| jButton2 | | | |

A continuación, se muestra la ventana que resulta del procedimiento anterior.

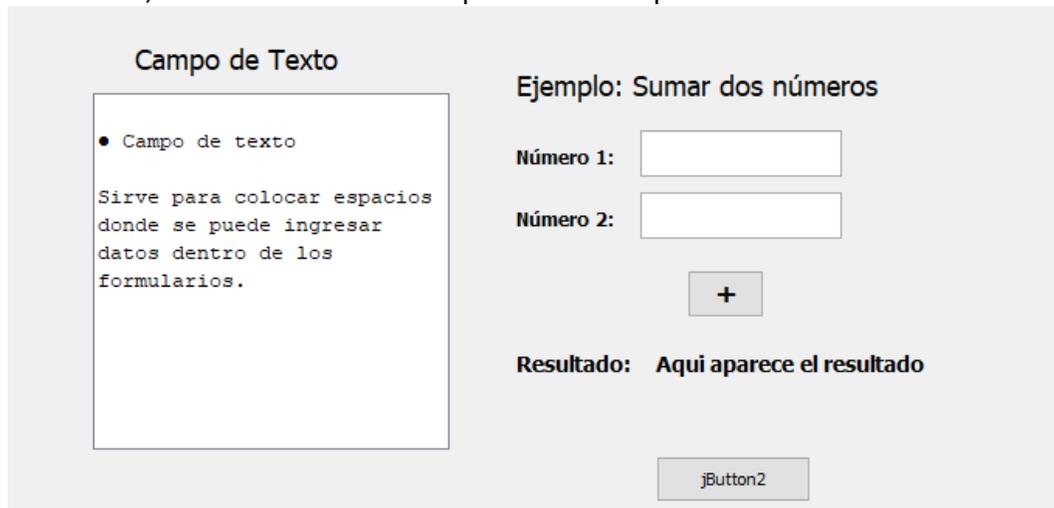


fig. 4.1.80. Formulario de diseño del componente campo de texto

2. Ahora cambiamos el nombre del campo de texto *(jTextField1* a *txtNumero1*.

- a. Para cambiar su nombre da clic derecho sobre el elemento *jTextField1* y en el menú contextual selecciona **Cambiar nombre de variable ...**

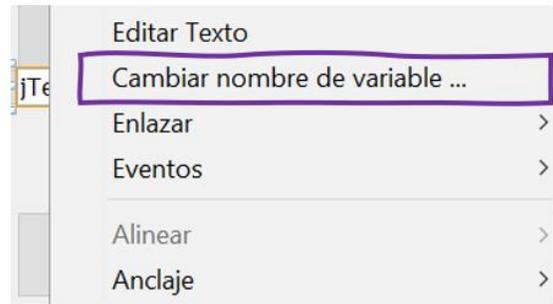


fig. 4.1.81. Cuadro de cambio de variable

Se mostrará la siguiente ventana, la cual muestra el nombre que tiene el elemento seleccionado y solicita el Nombre Nuevo, en este caso se escribe *txtNumero1* y se oprime *Aceptar*.

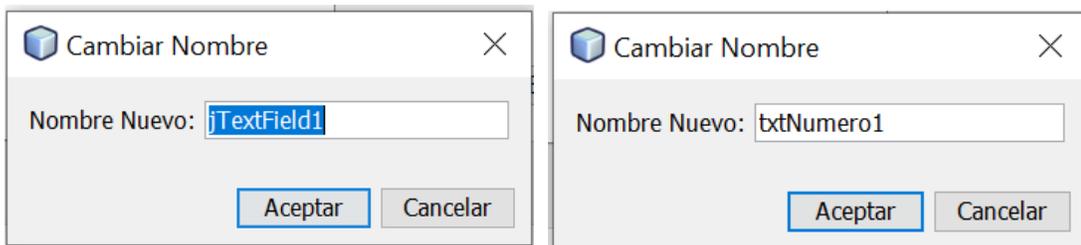


fig. 4.1.82. Cambio de variable

Este mismo procedimiento se realizará para el *jTextField2*, *jButton1* y *jLabel6*, para cambiar su nombre respectivo *txtNumero2*, *btnSuma* y *lblResultado*

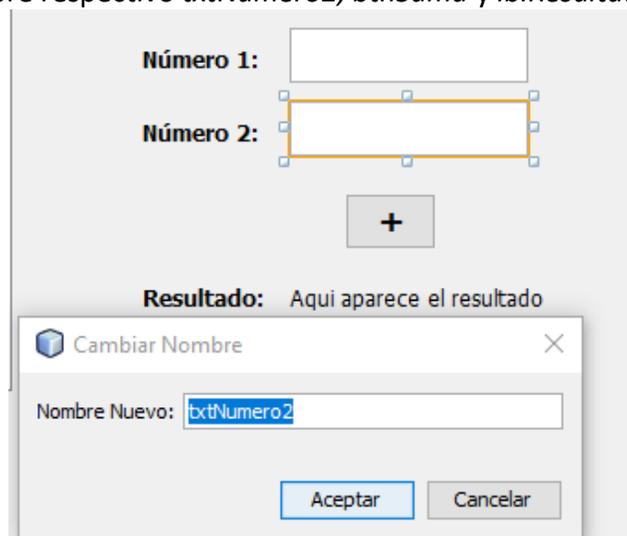


fig. 4.1.83. Cambio texto *txtNumero2*

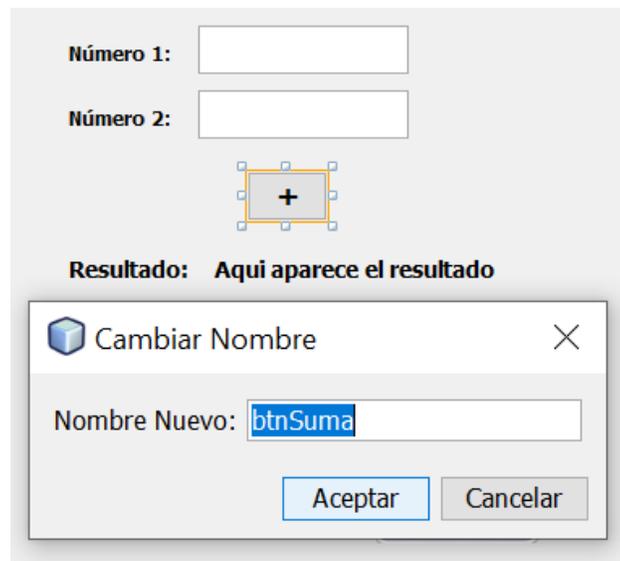


fig. 4.1.84. Cambio botón a btnSuma

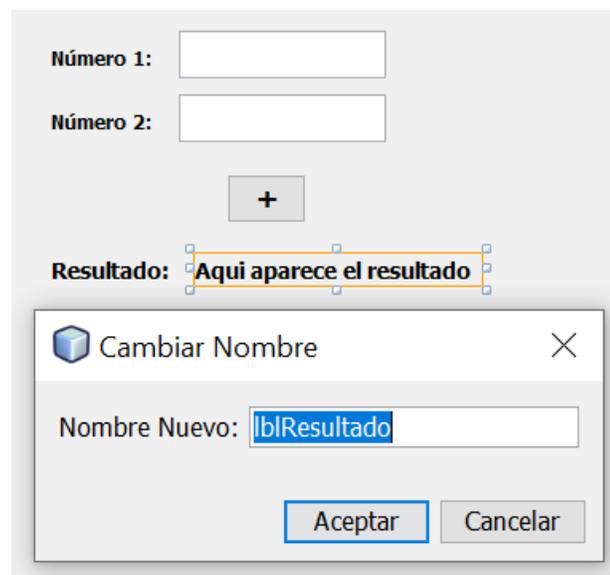


fig. 4.1.85. Renombrar etiqueta

- b. En el componente campo de texto se colocará un texto de ayuda, para ello en la propiedad *toolTipText* se coloca el texto *ingresa un número*

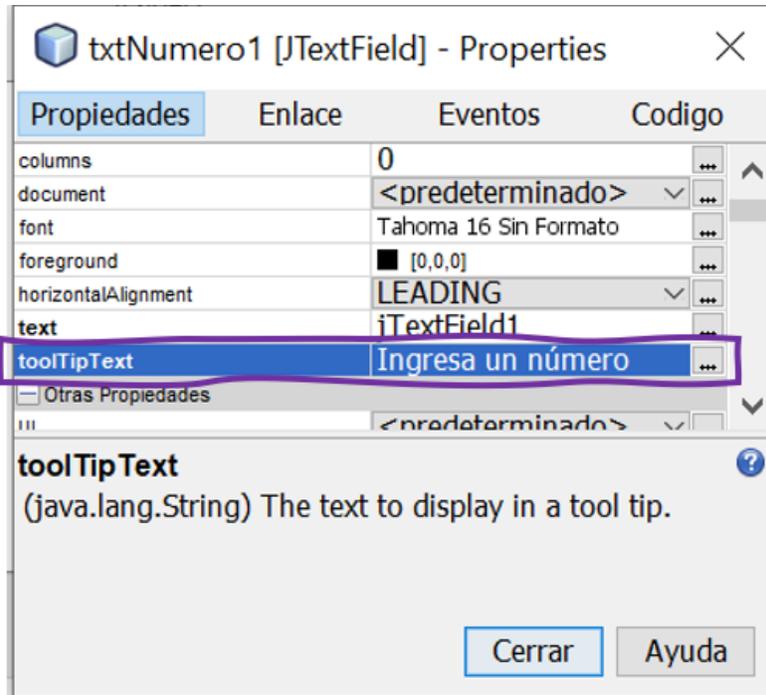


fig. 4.1.86. Propiedades campo texto

3. Se da doble clic al botón “+” para agregar el siguiente código

```
private void btnSumaActionPerformed(java.awt.event.ActionEvent evt) {
    //Variables para captar los números a sumar haciendo la conversión al tipo
    Double
    double numero1 = Double.parseDouble(this.txtNumero1.getText());
    double numero2 = Double.parseDouble(this.txtNumero2.getText());
    //Variable para guardar el resultado de la suma
    double resultado = numero1 + numero2;
    //Conversión del resultado y asignación a la etiqueta en el formulario
    this.lblResultado.setText(Double.toString(resultado));
}
```

4. Modifica las propiedades **icon** y **rollOverIcon** del **jButton2** para que aparezca la imagen **clase_swing** y **clase_swing2** respectivamente. Agrega en el mismo el botón su respectivo código para regresar al Mapa mental. (Este procedimiento lo puedes revisar en el punto 1 incisos d y e de la Actividad 12).
6. Agrega el código para centrar el formulario en la pantalla. (Actividad 12, inciso f)
7. En el formulario MapaMental agrega el código correspondiente al botón *Campo de Texto*. (Actividad 12, inciso f).

8. Imprime y pega la pantalla que aparece al ejecutar el programa.



Actividad 14

Conversión de datos de tipo *String* a numérico y viceversa.

De acuerdo con su función, selecciona el número y colócalo en el paréntesis correspondiente en la tabla.

```

    1      2      3      4
    ────┬───┬───┬───┬───
    int num = Int.parseInt(jTextField1.getText( ));

    5      6      7      8
    ────┬───┬───┬───┬───
    jLabelN.setText(Double.toString(resultado));
  
```

fig. 4.1.87. Función.

| | |
|-----|--|
| () | Clase para usar los métodos del tipo de dato double. |
| () | Método para convertir los valores del tipo String a tipo numérico. |
| () | Variable que contiene el resultado del cálculo a mostrar. |
| () | Representa el nombre de la etiqueta en donde se va a mostrar. |
| () | Representa el campo de texto en donde el usuario escribe el valor. |
| () | Método para acceder al valor escrito en el campo. |

| | |
|-----|--|
| () | Función que asigna un valor del tipo String. |
| () | Representa la variable que va a tomar el valor numérico. |

4.1.11 Casilla de activación



JCheckBox - Casilla de Activación

fig. 4.1.88. Casilla de Activación.

Este componente nos permite seleccionar una o varias opciones de una lista. Se utilizan generalmente como botones de estado.

Actividad 15

Insertando y programando el botón del componente Casilla de Activación

Se realizará un programa que muestre una lista de comidas para que el usuario seleccione sus comidas favoritas y al dar clic en el botón “comidas” aparezcan escritas en una etiqueta.

1. Crear un nuevo formulario con el nombre CasillaActivacionForm y colocamos los siguientes elementos: 3 etiquetas, un área de texto, un panel y dos botones, distribuidos como se muestra a continuación:



fig. 4.1.89. Componentes formulario casilla de activación.

El recuadro seleccionado es un Panel para colocar la lista de opciones. Un Panel se utiliza como un contenedor, en él se puede colocar y acomodar elementos (dentro del panel) y así luego se facilitará mover todos esos componentes relacionados de un lugar a otro solo moviendo el Panel y no cada uno de los elementos que contiene.

2. Las propiedades de texto correspondientes a cada componente son:

| Componente | Propiedad - Texto | Componente | Propiedad - Texto |
|------------|------------------------------------|------------|----------------------------|
| jLabel1 | Casilla de Activación | jTextArea | Descripción del componente |
| jLabel2 | ¿Cuáles son tus comidas favoritas? | jButton1 | Lista |
| jLabel3 | Aquí aparece la lista | jButton2 | |

3. En la propiedad *nombre* se renombrarán los componentes como sigue:

| Componente | Propiedad - Nombre | Componente | Propiedad - Nombre |
|------------|-----------------------|------------|--------------------|
| jLabel3 | <i>lblListaComida</i> | jButton1 | btnLista |

4. Para colocar la lista de comidas se selecciona el Panel y de las propiedades seleccionamos *border*

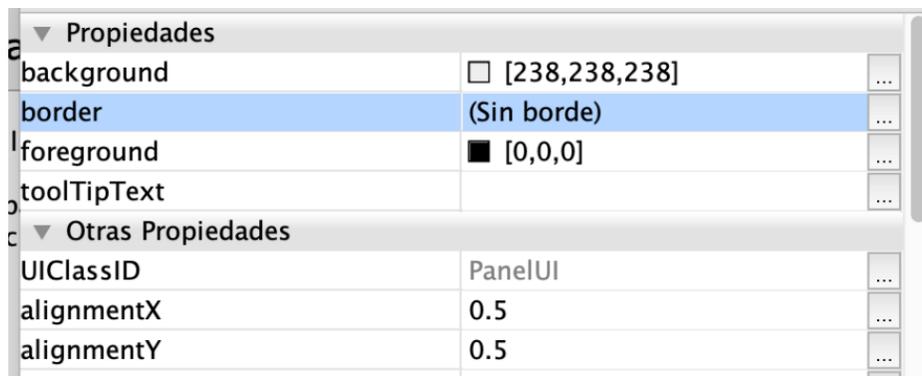


fig. 4.1.90. Propiedades etiqueta.

De allí podemos colocar Borde con título y poner *Lista de comidas*

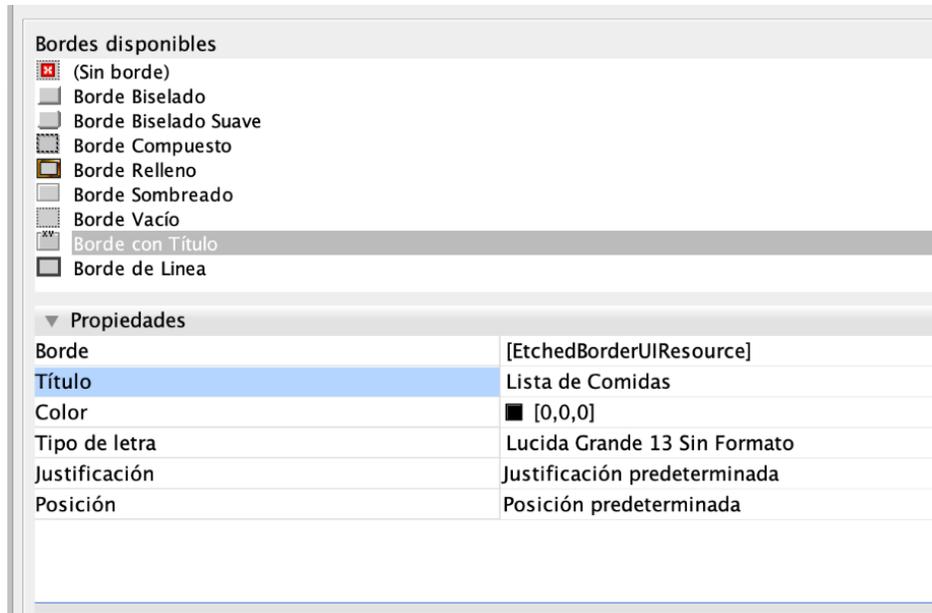


fig. 4.1.91. Ventana cambio de Borde y titulo.

El panel se muestra en la siguiente figura:

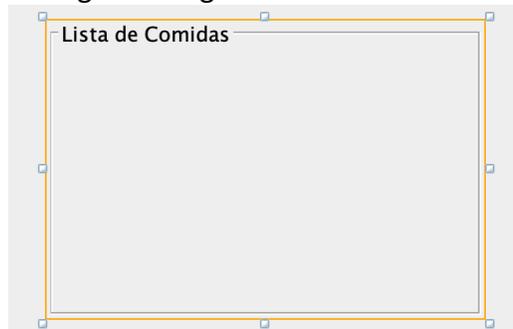


fig. 4.1.92. Panel.

5. Colocamos diseño nulo en el panel para poder acomodar la lista de opciones

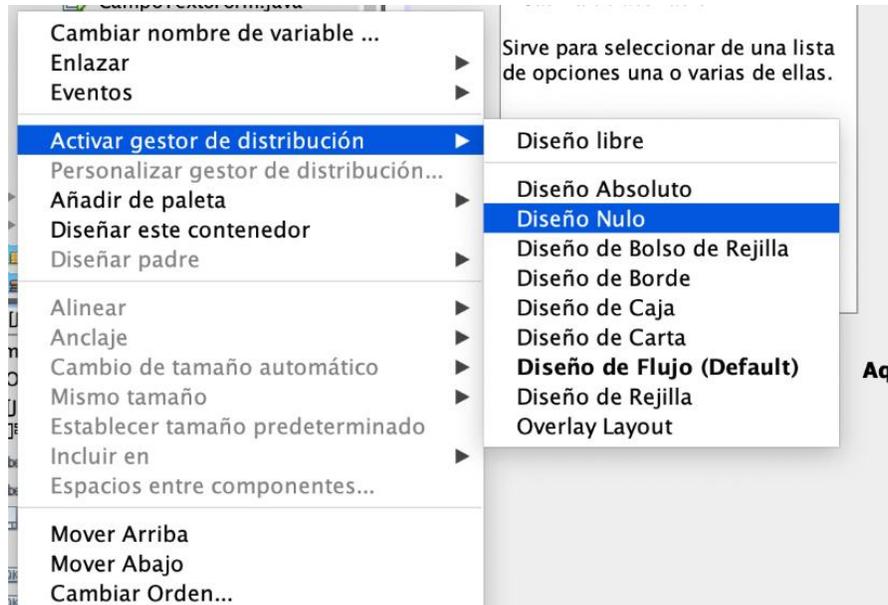


fig. 4.1.93. Cambio a diseño nulo.

6. Ahora, dentro del Panel se colocarán siete casillas de verificación y se modificará su texto con alguna de las siguientes opciones: dando doble clic sobre ella, oprimir botón derecho y seleccionar Editar texto o en la propiedad texto.

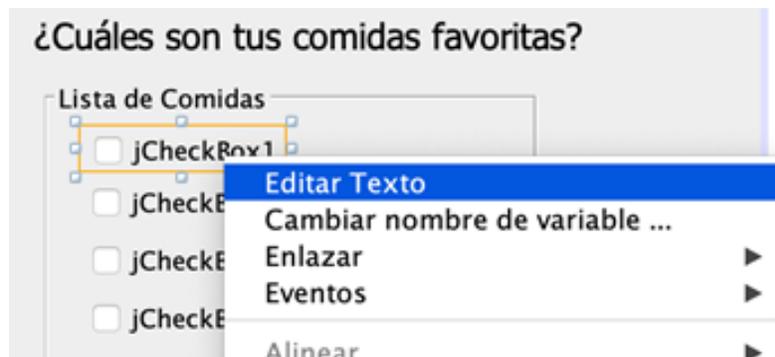


fig. 4.1.94. Panel para cambio de texto.

Los textos serán las comidas: *Tacos*, *Tortas*, *Quesadillas*, *Pescado*, *Verduras*, *Hamburguesas* y *Hotdogs*.

7. Para identificar a cada casilla de verificación cambiaremos su nombre de variable, se nombrarán según el elemento (jCheckBox) y su texto, esto es, *chkTacos*, *chkTortas*, *chkQuesadillas* y así sucesivamente.
 - a. Para ello da clic derecho sobre el jCheckBox1 y selecciona Cambiar el nombre de variable

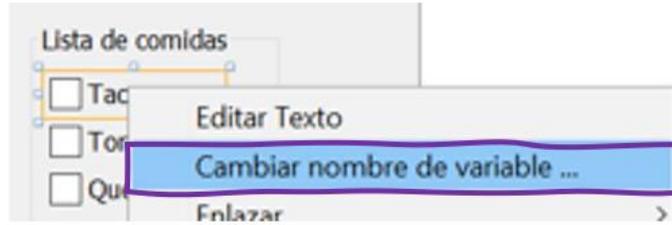


fig. 4.1.95. Propiedad cambio de variable.

En la ventana que aparece se podrá colocar el nombre nuevo al elemento.

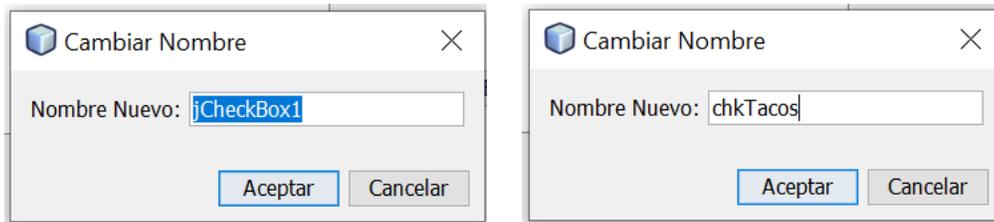


fig. 4.1.96. Cambio de variable.

Ahora se visualiza el formulario con la lista de comidas a seleccionar

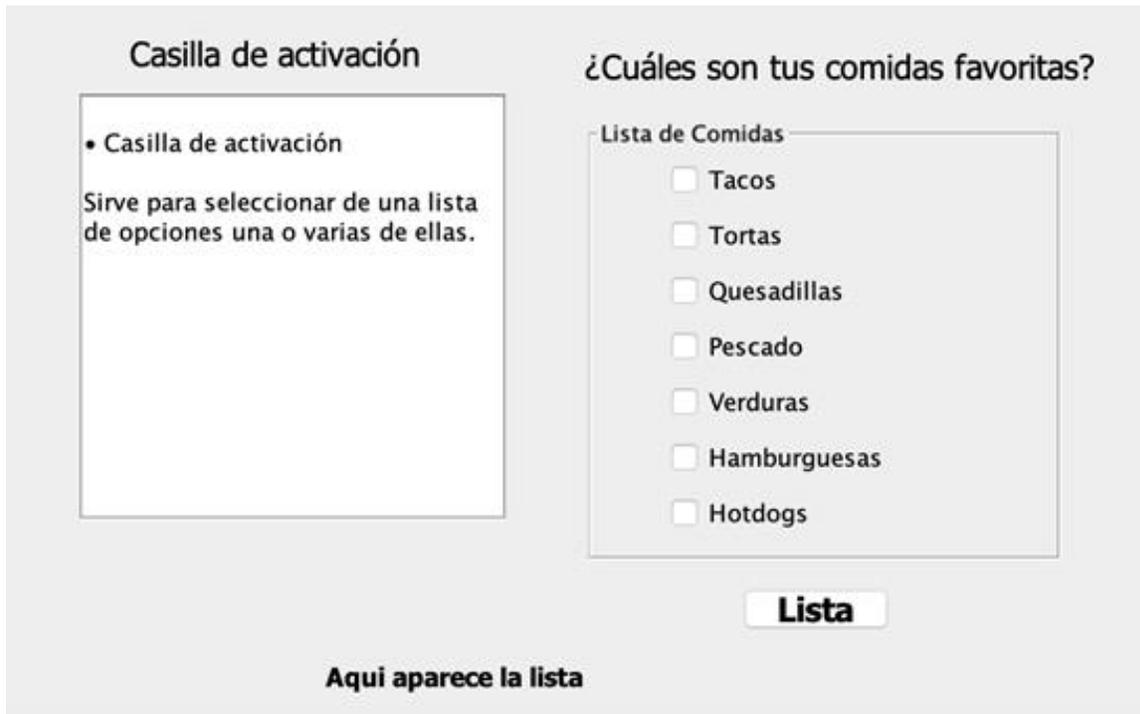


fig. 4.1.97. Panel lista de comidas

8. Seguiremos con el código del botón **Lista**, el cual permitirá colocar las comidas preferidas que se seleccionaran.
 - a. Dar doble clic al botón Lista e introducir el siguiente código.

```
private void btnListaActionPerformed(java.awt.event.ActionEvent evt) {
    String lista = "";
    //Si la casilla de Tacos se selecciona la condición se cumple
    if (this.chkTacos.isSelected()){
        //Se concatena el texto de la casilla - comida seleccionada: Tacos
        lista = lista + this.chkTacos.getText()+" ";
    }
    if (this.chkTortas.isSelected()){
        // Se concatena el texto de la casilla - comida seleccionada: Tortas
        lista = lista + this.chkTortas.getText()+" ";
    }
}
```

Revisa el código escrito para los tacos y tortas, con ello escribe en las líneas del código correcto para las comidas faltantes.

```
//Se muestra el texto de la lista con las comidas seleccionadas
this.lblListaComida.setText(lista);
}
```

9. Modifica las propiedades **icon** y **rollOverIcon** del **jButton2** para que aparezca la imagen `clase_swing` y `clase_swing2` respectivamente. Agrega en el mismo el botón su respectivo código para regresar al Mapa mental. (Este procedimiento lo puedes revisar en el punto 1 incisos d y e de la Actividad 12).
10. Agrega el código para centrar el formulario en la pantalla. (Actividad 12, inciso f)
11. En el formulario MapaMental agrega el código correspondiente al botón *Casilla de Activación* (Actividad 12, inciso f)
12. Imprime y pega la pantalla que aparece al ejecutar el programa.

Actividad 16

Ordena

Escribe el orden correcto de los pasos para generar la lista de la actividad anterior.

| | | |
|-----|--------------------------------|------------------|
| () | Título | Lista de Comidas |
| () | Borde con titulo | |
| () | jCheckBox1 | Editar Texto |
| () | Activar gestor de distribución | Diseño nulo |

4.1.12 Botón de Opción



JRadioButton – Botón de opción

fig. 4.1.98. Botón de opción.

El control de botones de opción o botones de radio, se utilizan cuando se quiere que el usuario pueda elegir solo una opción entre varias. Para que esto suceda se deben asociar los JRadioButton de las opciones posibles, esto se logra colocándolos dentro de un objeto JPanel - Panel y asociarlos a un grupo de botones. En caso contrario, será posible activar varios botones de opción a la vez al momento de ejecutar el programa.

Actividad 17

Insertando y programando el botón del componente *Botón de opción*

El uso de este componente es para seleccionar una sola opción de una lista.

Se desarrollará un programa para mostrar al usuario una lista de carreras cuando se haya elegido una, se oprime el botón "Carrera" para que aparezca en una etiqueta.

1. Crear un nuevo formulario con el nombre BotonOpcionForm y colocar los siguientes elementos: 3 etiquetas, un área de texto, un panel y dos botones, distribuidos como se muestra a continuación:



fig. 4.1.99. Componentes de formulario botón de opción.

- a. Las propiedades de texto correspondientes a cada componente son las siguientes:

| Componente | Propiedad - Texto | Componente | Propiedad - Texto |
|------------|--------------------------------------|------------|----------------------------|
| jLabel1 | Botón de Opción | jTextArea | Descripción del componente |
| jLabel2 | Elige una carrera | jButton1 | Carrera |
| jLabel3 | Aquí aparece la carrera que elegiste | | |

- b. En la propiedad nombre se renombrarán los componentes como sigue:

| Componente | Propiedad - Nombre | Componente | Propiedad - Nombre |
|------------|------------------------|------------|--------------------|
| jLabel3 | <i>lblListaCarrera</i> | jButton1 | btnLista |

- c. Se colocará título al componente *Panel*, por lo que en la propiedad *Border* se selecciona Borde con *título* y en el título se escribe **Carreras**.
- d. Se agrega un grupo de botones dentro del Panel

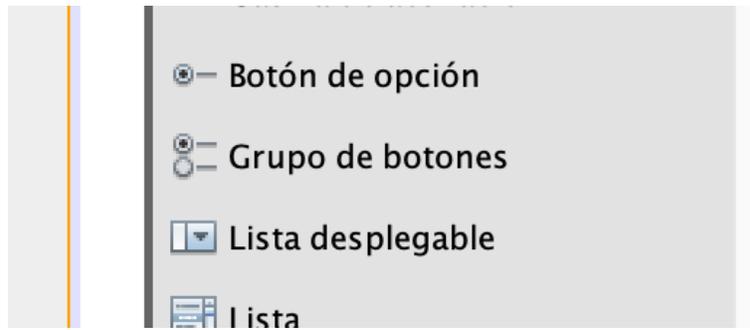


fig. 4.1.100. Panel grupo de botones.

Este grupo de botones nos va a permitir asociar los botones de opción que pertenezcan a él.

- e. Se agregarán siete botones de opción dentro del panel y se les cambiará la propiedad *texto* con las siguientes carreras: Médico, Arquitecto, Ingeniero, Abogado, Filósofo, Psicólogo, Artista.
- f. Ahora, cada elemento se asocia al grupo **buttonGroup1**, para ello, se selecciona un botón de opción y en la propiedad *buttonGroup*, se elige **buttonGroup1**. Se asocia cada botón de opción al **buttonGroup1**.

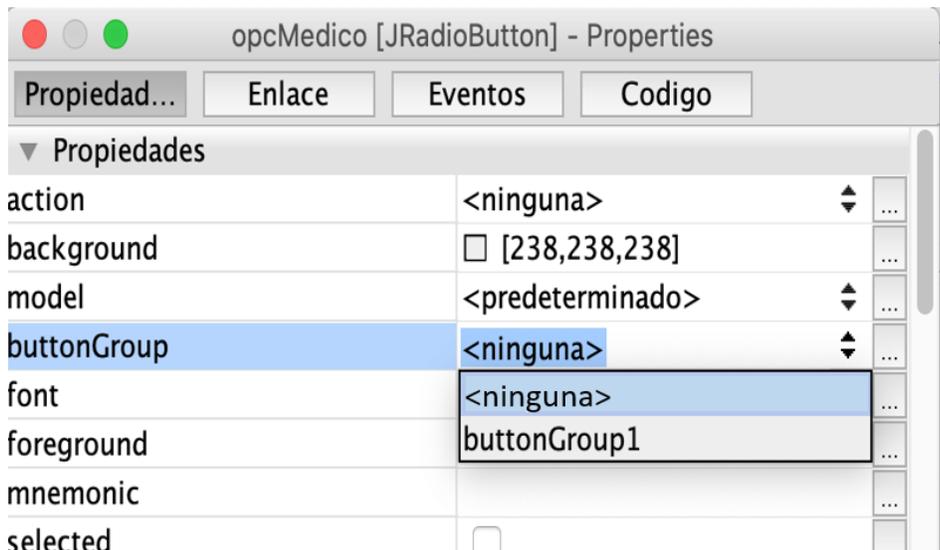


fig. 4.1.101. Propiedades de botón de opción.

Al agruparlos todos aparecen relacionados como se muestra en la siguiente figura:



fig. 4.1.102. Panel para selección de carreras.

2. Para identificar cada botón de opción se cambiará su nombre de variable, se nombrarán según el texto de la opción, *esto es, opcMédico, opcArquitecto, opcIngeniero, opcAbogado* y así sucesivamente.

Para lo anterior, recuerda que debes dar clic derecho sobre el elemento *JRadioButton1* y seleccionar *Cambiar el nombre de variable* y escribir el correspondiente. Este procedimiento se debe realizar para todas las opciones. Por ejemplo:

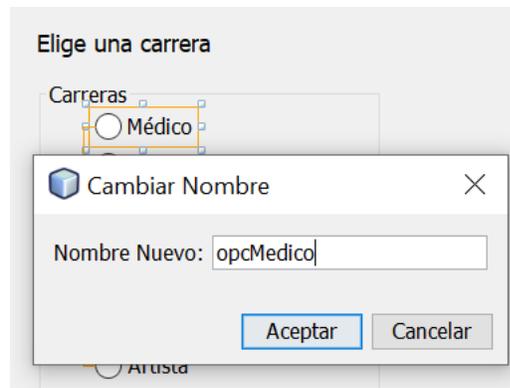


fig. 4.1.103. Cambiar Nombre.

3. Seguiremos con el código del botón Lista, el cual permitirá colocar la carrera seleccionada.
 - a. Da doble clic al botón Lista y escribe el siguiente código.

```
private void btnListaActionPerformed(java.awt.event.ActionEvent evt) {
    String carrera = "";
    //Si se selecciona la opción Médico la condición se cumple
    if (this.opcMedico.isSelected()){
```

```

//Se asigna el texto de la opción seleccionada - Médico
carrera = this.opcMedico.getText();
}

//Si se selecciona la opción Arquitecto la condición se cumple
if (this.opcArquitecto.isSelected()){
//Se asigna el texto de la opción seleccionada - Arquitecto
carrera = this.opcArquitecto.getText();
}

```

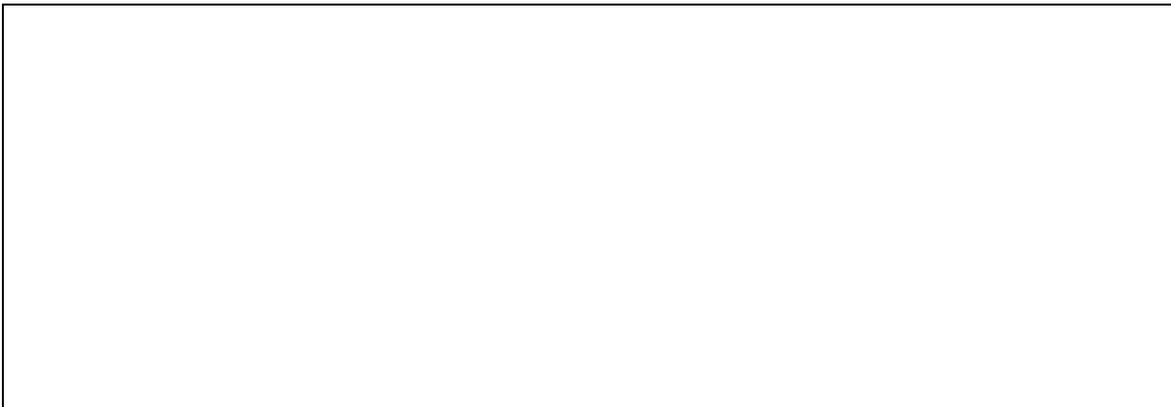
Revisa el código de las dos primeras opciones y continúa con las que hacen falta

```

}
//Se asigna la carrera seleccionada a la etiqueta
this.lblListaCarrera.setText("Elegiste: "+carrera);
}

```

4. Modifica las propiedades **icon** y **rollOverIcon** del **jButton2** para que aparezca la imagen `clase_swing` y `clase_swing2` respectivamente. Agrega en el mismo el botón su respectivo código para regresar al Mapa mental. (Este procedimiento lo puedes revisar en el punto 1 incisos d y e de la Actividad 12).
5. Agrega el código para centrar el formulario en la pantalla. (Actividad 12, inciso f)
6. En el formulario MapaMental agrega el código correspondiente al botón *Botón de opción* (Actividad 12, inciso f)
7. Imprime y pega la pantalla que aparece al ejecutar el programa.



4.1.13 Combo Box



JcomboBox – Combo Box

fig. 4.1.104. Combo Box

Este control nos permite seleccionar un valor de entre una serie de opciones, es muy útil cuando se sabe cuáles son los datos que se pueden ingresar, esto es de ayuda para evitar errores.

De una lista desplegable podemos obtener dos tipos de valores, ellos son:

- Índice o index (número de elemento).
- El ítem Seleccionado (nombre de elemento).

Actividad 18

Insertando y programando el botón del componente *Combo Box*

Este componente es para seleccionar una opción de una lista desplegable.

Se desarrollará un programa en donde se presente una lista desplegable que muestre diferentes carreras profesionales y al elegir una se indique en una etiqueta.

1. Crear un nuevo formulario con el nombre *ComboBoxForm* con los siguientes componentes:

| Componente | Propiedad - Nombre | Propiedad -Text |
|------------|--------------------|--------------------------------------|
| jLabel1 | jLabel1 | Combo Box |
| jLabel2 | jLabel2 | Elige una carrera |
| jLabel3 | <i>lblCarrera</i> | Aquí aparece la Carrera que elegiste |
| jTextArea1 | jTextArea1 | Descripción del componente Combo Box |
| jComboBox1 | | |
| jButton1 | | |

Distribuidos preferentemente como se muestra.

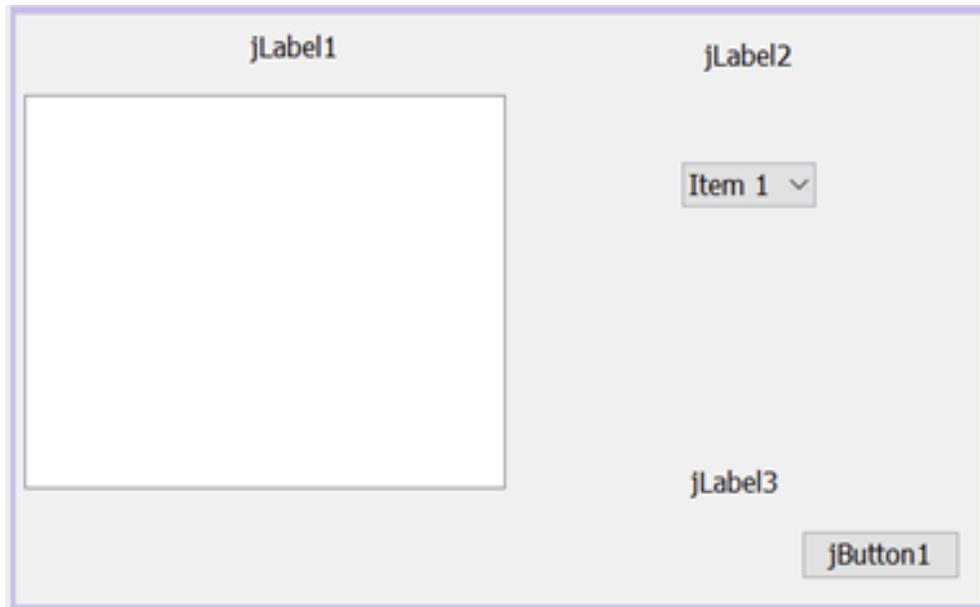


fig. 4.1.105. Componentes en el formulario para el ComboBox.

2. Para agregar la lista de las carreras se realiza lo siguiente:
 - a. Selecciona el ComboBox y nos dirigimos a las propiedades en donde se busca **model** para agregar los elementos de la lista de opciones, se da clic del lado derecho en

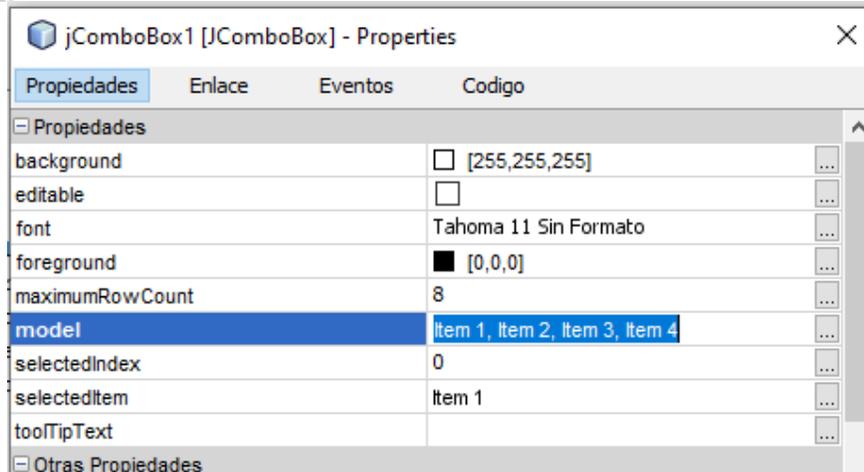


fig. 4.1.106. Propiedades ComboBox.

Y se empiezan a agregar las carreras cada una en un renglón:

Médico, Arquitecto, Ingeniero, Abogado, Filósofo, Psicólogo y Artista.

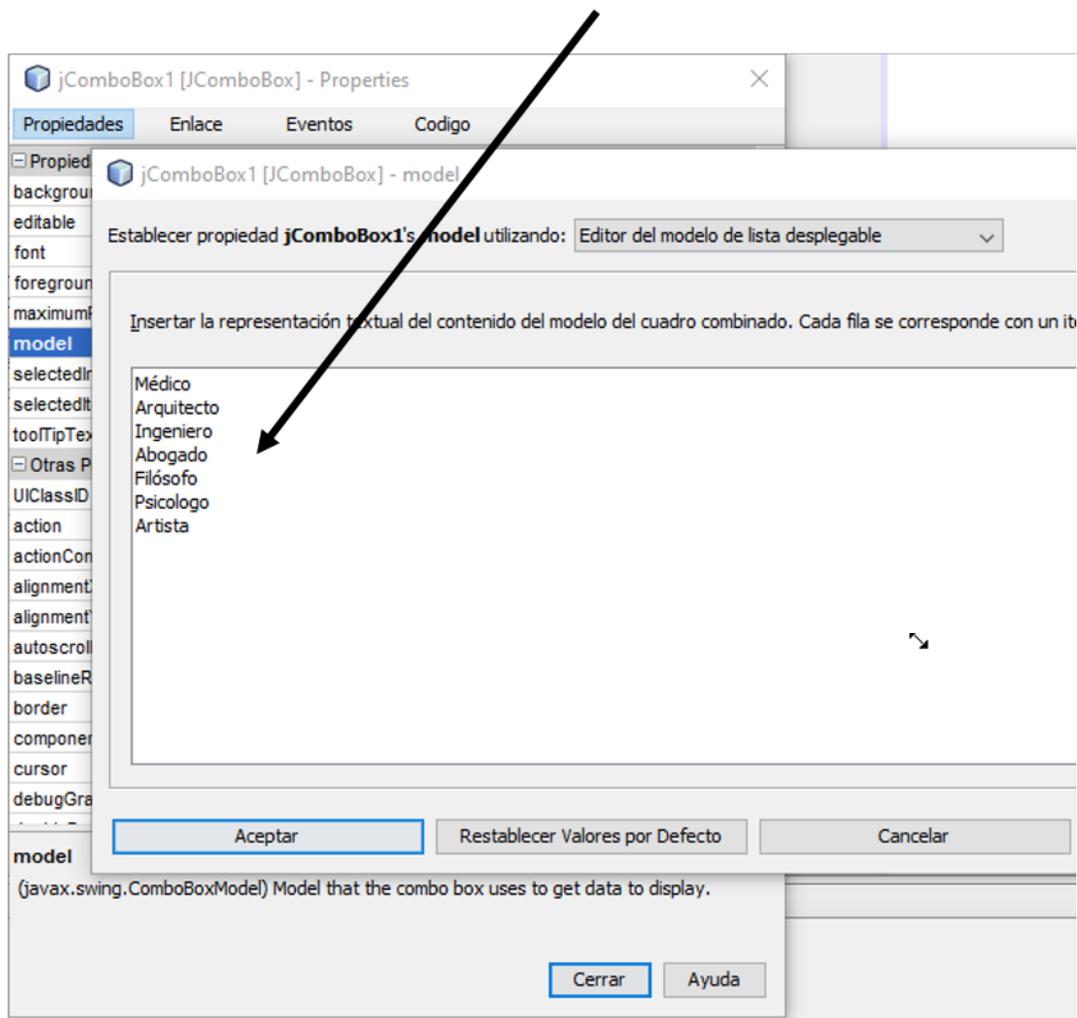


fig. 4.1.107. Pantalla agregar carreras.

Para que finalmente se visualice como en la siguiente figura:

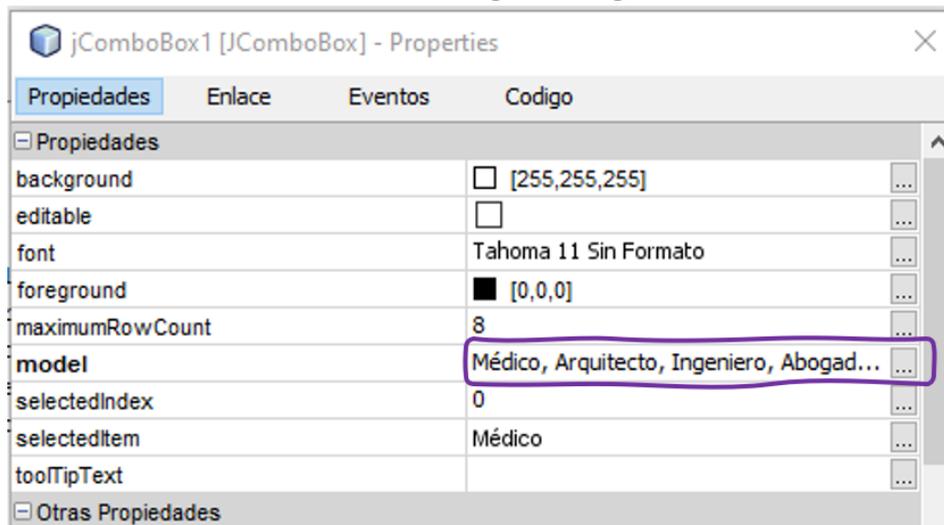


fig. 4.1.108. Visualización de carreras.

- Se da doble clic en el JComboBox para agregar el siguiente código:

```
private void JComboBox1ActionPerformed(java.awt.event.ActionEvent evt) {
    //Se obtiene el texto de la opción seleccionada y se asigna a la etiqueta
    this.lblCarrera.setText("Elegiste:
    "+this.JComboBox1.getSelectedItem().toString());
}
```

- Modifica las propiedades **icon** y **rollOverIcon** del **jButton2** para que aparezca la imagen clase_swing y clase_swing2 respectivamente. Agrega en el mismo el botón su respectivo código para regresar al Mapa mental. (Este procedimiento lo puedes revisar en el punto 1 incisos d y e de la Actividad 12).
- Agrega el código para centrar el formulario en la pantalla. (Actividad 12, inciso f)
- En el formulario MapaMental agrega el código correspondiente al botón *Combo Box* (Actividad 12, inciso f)
- Imprime y pega la pantalla que aparece al ejecutar el programa.

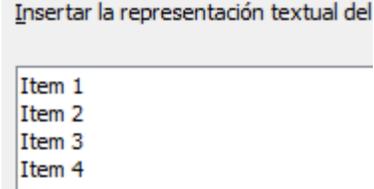


Actividad 19

Combo Box

Se realizará un formulario para elegir una carrera mediante el uso de un componente JComboBox. Se colocará el componente con la lista de carreras posibles. A continuación, están los pasos a seguir para realizar esta actividad, pero se muestran en desorden. Debemos colocarlos en el orden correcto.



| | |
|-----|---|
| () |  |
| () | Selección ComboBox Propiedades Model Presionar el botón  |
| () | Se agregan los componentes |
| () |  |
| () |  |
| () | Se da doble clic en el JComboBox y se agrega el código. |

4.1.14 Barra de Menú



fig. 4.1.109. Barra de menú.

JMenu Barra de menú

Es la barra de menú principal. Una barra horizontal alargada en la que se colocarán las distintas opciones. A cada una de estas opciones se puede agregar un **JMenu** o un **JMenuItem**.

Actividad 20

Insertando y programando el botón del componente Barra de Menú

Se realizará un formulario jForm en donde aparezca su descripción en un área de Texto y una Barra de Menú, en menú File se encontrará la opción Salir.

1. Generar un nuevo formulario con el nombre *MenuBarraForm* y colocar los siguientes componentes:

| Componente | Propiedad - Nombre | Propiedad -Text |
|------------|--------------------|-----------------|
| jLabel1 | jLabel1 | Barra Menú |

| | | |
|------------|------------|---------------------------------------|
| jTextArea1 | jTextArea1 | Descripción del componente Barra Menú |
| jFileMenu | jMenuBar | |
| jButton1 | jButton1 | |

Distribuidos preferentemente como se muestra en la figura.

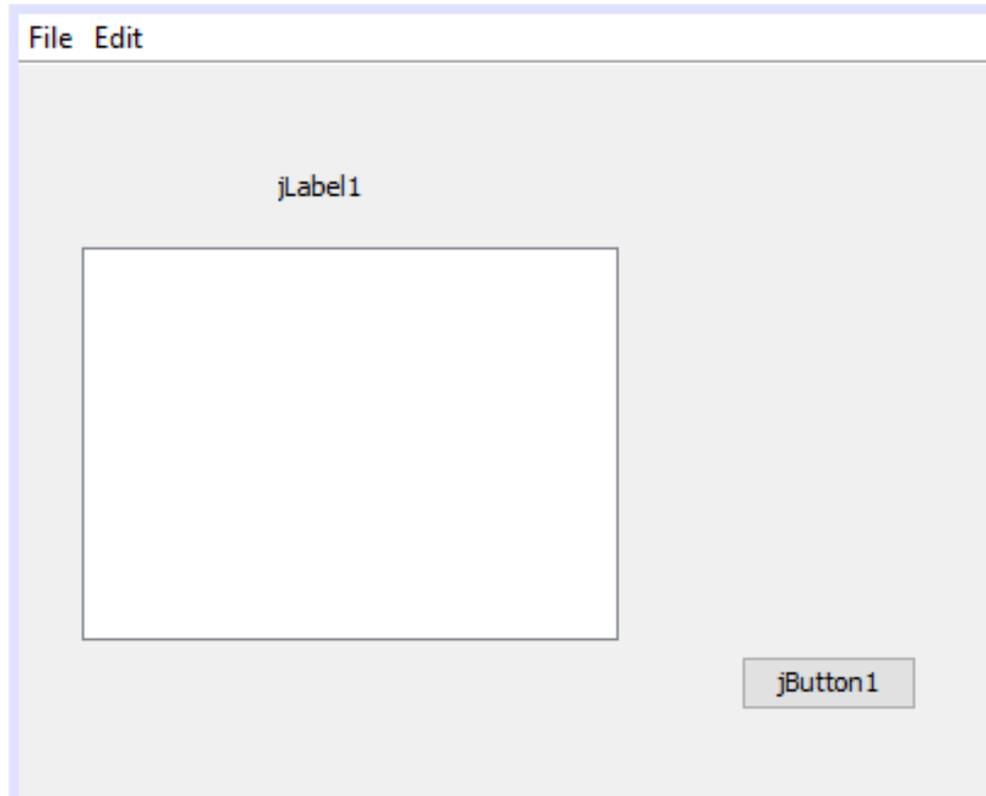


fig. 4.1.110. Componentes del formulario de MenuBarraForm.

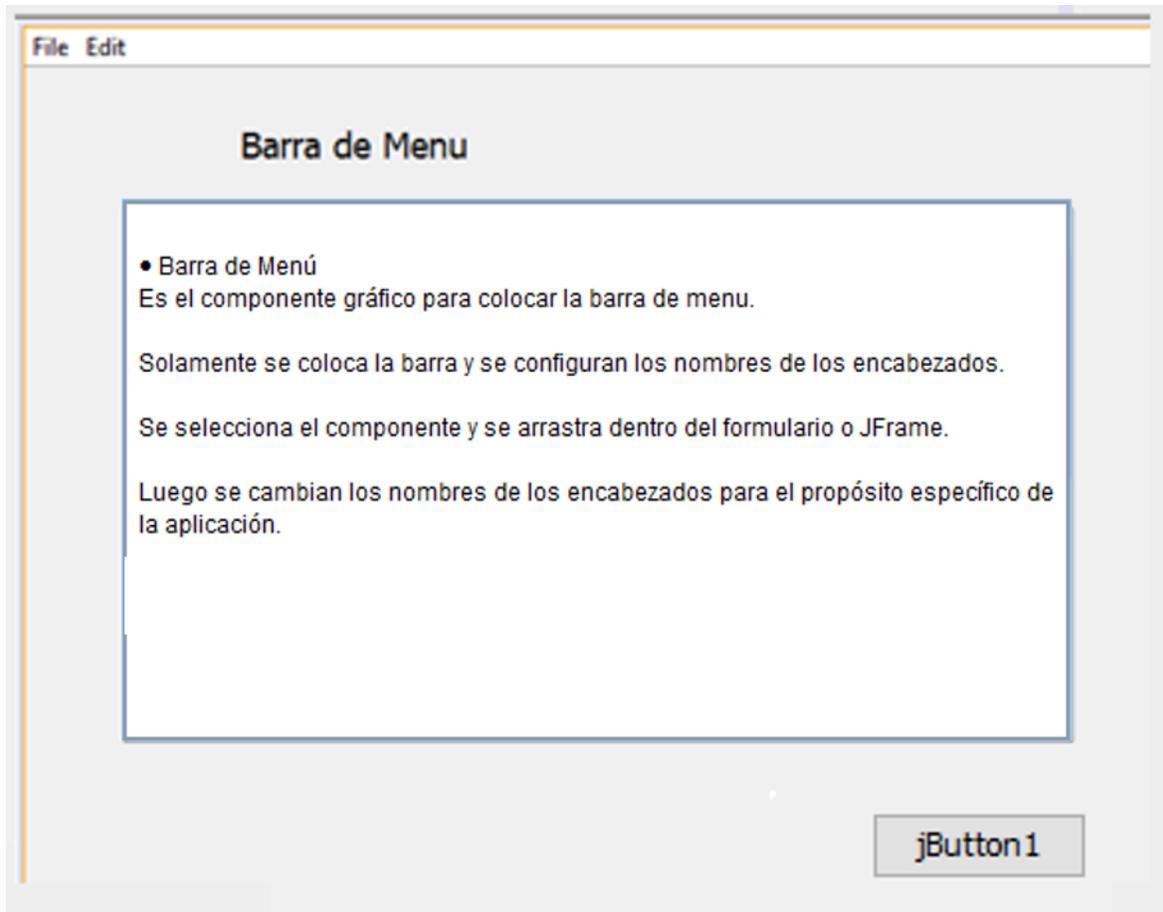


fig. 4.1.111. Resultado del formulario.

Observarás que al incluir el componente Barra Menú, inmediatamente se agrega en la parte superior de la ventana con los menús File y Edit.

2. Realiza los siguientes pasos para colocar opciones a la barra de menú.
 - a. Agregar la opción *salir* en el menú File. Para ello, se incluye un elemento de `JMenu` en File.

Se selecciona el objeto elemento de menú y debajo de *File*, como se muestra a continuación:



fig. 4.1.112. Selección del objeto MenuItem1.

- b. Se renombra el objeto `JMenuItem1` a Salir, como se muestra a continuación:



fig. 4.1.113. Selección del objeto menú.

En donde se cambiará el texto `JMenuItem1` por `Salir`. Esto se puede realizar al dar clic en el texto y modificarlo; oprimiendo botón derecho y dar clic en Editar Texto o dirigirse a la ventana de Propiedades buscando Text y modificarlo.

3. Para darle funcionalidad podemos anexar código dando doble clic sobre el elemento **Salir** y anexamos el código correspondiente.

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
    //Código para salir o terminar la ejecución
    System.exit(0);
}
```

4. Modifica las propiedades **icon** y **rollOverIcon** del **JButton2** para que aparezca la imagen `clase_swing` y `clase_swing2` respectivamente. Agrega en el mismo el botón su respectivo código para regresar al Mapa mental. (Este procedimiento lo puedes revisar en el punto 1 incisos d y e de la Actividad 12).
5. Agrega el código para centrar el formulario en la pantalla. (Actividad 12, inciso f)
6. En el formulario `MapaMental` agrega el código correspondiente al botón *Barra de menú* (Actividad 12, inciso f)
7. Imprime y pega la pantalla que aparece al ejecutar el programa.



4.1.15 Menú



JMenu – Menú

fig. 4.1.114. Menú

Es una barra alargada en la que se colocarán las distintas opciones. Es el menú principal. En cada una de estas opciones se puede agregar un JMenuitem.

Es el componente gráfico para colocar un menú dentro de los encabezados principales de la barra.

En este caso se construirá un formulario con un menú para los distintos instrumentos musicales y cada menú se asociará a los tipos de instrumentos.

Actividad 21

Insertando y programando el botón del componente Menú

Colocaremos una Barra de menú para los *Instrumentos musicales*, clasificándolos según su tipo: Cuerda, Percusión y Viento.

1. Crear el formulario correspondiente con el nombre de *MenuForm*.
2. Colocar una Barra de menú con la opción de Instrumentos musicales en él que aparezca un menú sobre los instrumentos musicales clasificándolos según su tipo: Cuerda, Percusión y Viento.

Agregar en el formulario los siguientes componentes:

| Componente | Propiedad - Nombre | Propiedad -Text |
|------------|--------------------|---------------------------------|
| jLabel1 | jLabel1 | Menú |
| jTextArea1 | jTextArea1 | Descripción del componente Menú |
| jFileMenu | jMenuBar | |
| jButton1 | jButton1 | |

3. Distribuidos preferentemente como se muestra, cambiando el texto de la opción Edit de la barra de menú a Instrumentos Musicales.

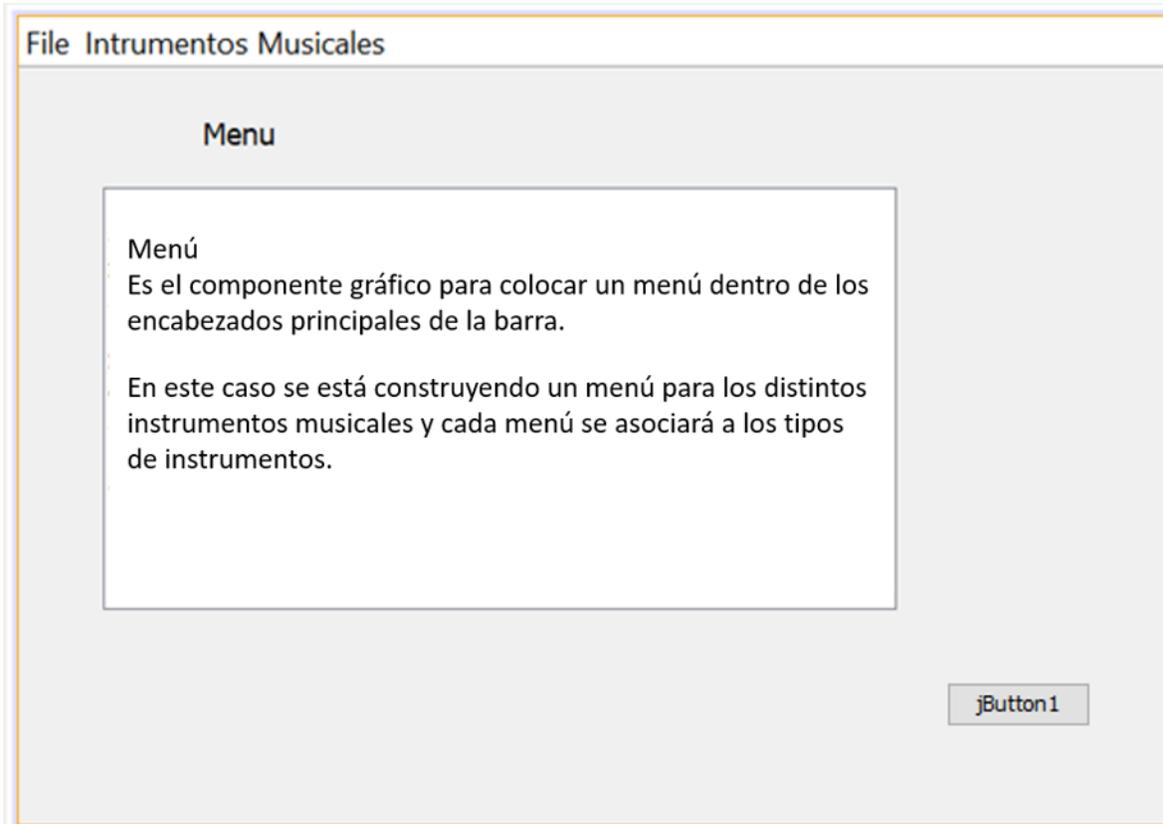


fig. 4.1.115. Salida de programa menú.

4. Colocar el componente button1, el procedimiento para que aparezca la imagen Clase **Swing** y nos pueda retornar al formulario del Mapa.
5. Agregar en la opción **File**, un menú Item que se llame **Salir**, con el código correspondiente para salir del programa.
6. Colocar un Menú principal para los **Intrumentos Musicales**.
7. Colocar tres submenús para los distintos instrumentos musicales.

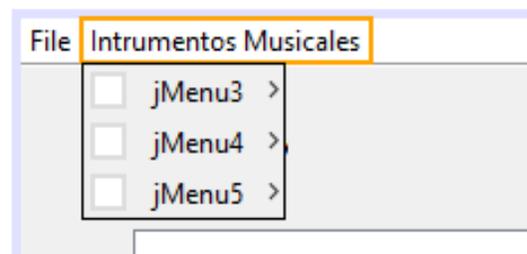


fig. 4.1.116. Submenús.

8. Cambiar el texto para que se muestren cada uno:

jMenu3 – Cuerda, jMenu4 – Percusión y jMenu5 – Viento. (Los números pueden cambiar)

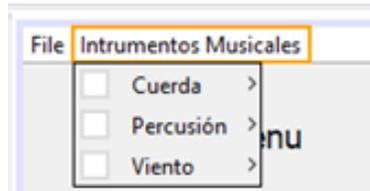


fig. 4.1.117. Agregar instrumentos musicales.

9. Modifica las propiedades **icon** y **rollOverIcon** del **jButton2** para que aparezca la imagen clase_swing y clase_swing2 respectivamente. Agrega en el mismo el botón su respectivo código para regresar al Mapa mental. (Este procedimiento lo puedes revisar en el punto 1 incisos d y e de la Actividad 12).
10. Agrega el código para centrar el formulario en la pantalla. (Actividad 12, inciso f)
11. En el formulario MapaMental agrega el código correspondiente al botón *Menú* (Actividad 12, inciso f)
12. Imprime y pega la pantalla que aparece al ejecutar el programa.



Actividad 22

Imágenes y sonidos

En esta actividad vamos a crear la carpeta para colocar las imágenes de los instrumentos musicales y también vamos a crear otra carpeta donde colocar los sonidos correspondientes para cada uno de los instrumentos musicales. Es importante considerar las ubicaciones tal como se indican para que el código funcione correctamente.

1. En la subcarpeta **src** del proyecto genera una carpeta con el nombre de *imagenes*, guarda imágenes de instrumentos musicales de cuerdas (Guitarra, Violín, Arpa, Piano), percusión (Batería, Marimba, Panderero y Maracas) y de viento (Trompeta, Flauta, Oboe, Clarinete), tamaño 236 x 200 pixeles en formato **png**, ejemplo: Piano.png



fig. 4.1.118. Piano.

2. En la subcarpeta **src** del proyecto genera una carpeta con el nombre *audios*, guarda los sonidos de instrumentos musicales en formato mp3, de los instrumentos musicales de cuerdas (Guitarra, Violín, Arpa, Piano), percusión (Batería, Marimba, Panderero y Maracas) y de viento (Trompeta, Flauta, Oboe, Clarinete).

La siguiente imagen muestra cómo deben quedar las carpetas.

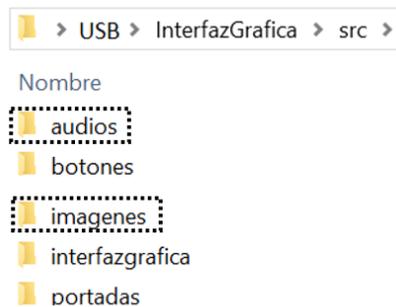


fig. 4.1.119. Carpetas.

4.1.16 JMenuItem - Elemento de Menú



JMenuItem - Elemento de Menú

fig. 4.1.120. Elemento menú.

Este objeto sirve para agregar submenús a la opción que corresponde de un JMenuItemBar. Cuando se hace clic en un menú, éste se expande para mostrar su lista de elementos de menú.

Menú Item o Elemento de menú es el componente gráfico para añadir un elemento al menú. Son los puntos terminales de los menús que corresponden a la selección deseada. Aquí es donde generalmente se da la funcionalidad o se ejecuta la acción correspondiente a la opción seleccionada.

Actividad 23

Insertando y programando el botón del componente Elemento de Menú

Con este componente se integrará los elementos de menú a cada tipo de Instrumento Musical y al seleccionar uno de ellos aparecerá su: nombre, imagen y sonido.

Recuerda que las carpetas de **audios** e **imagenes** deben estar en la subcarpeta **src** del proyecto.

1. Generar un nuevo formulario con el nombre de *MenuItemForm* y colocar los siguientes componentes:

| Componente | Propiedad - Nombre | Propiedad -Text |
|------------|--------------------|---|
| jLabel1 | jLabel1 | Menu Item – Elemento de menú |
| jLabel2 | jLabel2 | Instrumento Musical |
| jLabel3 | lblInstrumento | Aquí aparece |
| jLabel4 | lblImagen | |
| jTextArea1 | jTextArea1 | Descripción del componente Elemento de Menú |
| jFileMenu | jMenuBar | |
| jButton1 | jButton1 | |

Para el jLabel4 se cambiará la propiedad *Border*; da clic derecho sobre el componente, selecciona **Propiedades**, elige *Border* y selecciona *Borde Relleno*, también te permitirá cambiar el grosor y el color, en este momento dejamos el default.

2. Distribuidos preferentemente como se muestra.

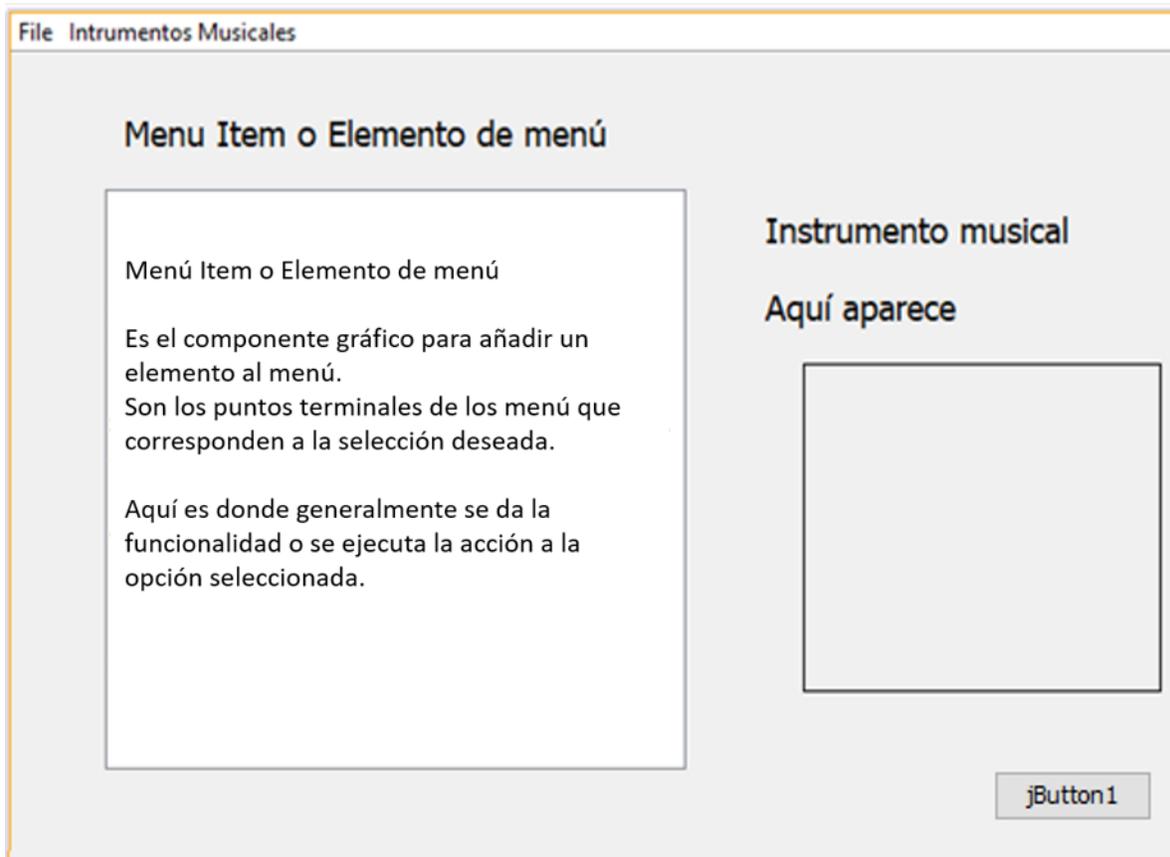


fig. 4.1.121. Componentes del Elemento menú.

3. Coloca en **File** la opción **Salir** con su código correspondiente.
4. Agrega los tipos de instrumentos musicales en el elemento *Menu* (Cuerda, Percusión y Viento)
5. Coloca en la opción de *Instrumentos musicales* de *Cuerda* cuatro Elementos de Menú, los cuales tendrán como nombre Guitarra, Violín, Arpa y Piano, respectivamente. Recuerda que puedes dar doble clic izquierdo sobre el componente y podrás cambiar el texto o dirigirte a Propiedades, buscar texto y cambiarlo.

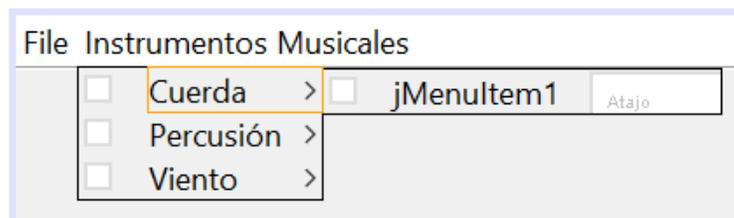


fig. 4.1.122. Menú instrumentos de cuerda.

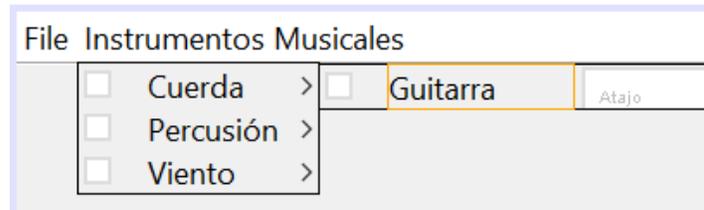


fig. 4.1.123. Agregar instrumentos cuerda guitarra.

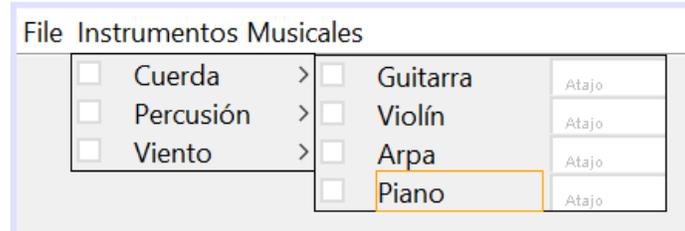


fig. 4.1.124. Agregar instrumentos de cuerda.

6. También debemos cambiar los nombres de las variables para que correspondan con el código, con botón derecho sobre el menú aparece la lista de opciones y seleccionamos cambiar nombre a la variable.

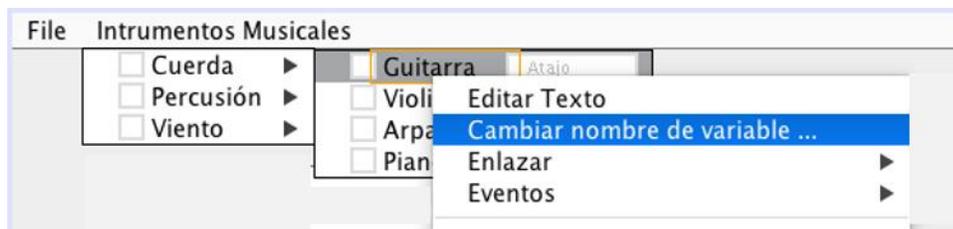


fig. 4.1.125. Opción Cambiar nombre de variable.

En el caso de Guitarra el nombre de la variable debe ser *menuGuitarra*, para Violín debe ser *menuViolin* y así con el menú de cada instrumento.

7. Agrega en el Menú Percusión elementos para la: Batería, Marimba, Panderero y Maracas. Como se muestra a continuación:

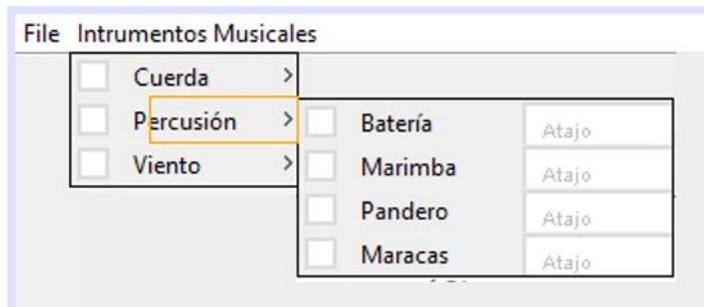


fig. 4.1.126. Selección menú percusión.

8. Coloca en el Menú Viento elementos para la: Trompeta, Flauta, Oboe y Clarinete.

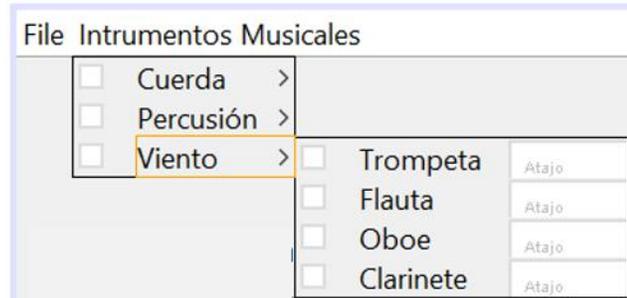
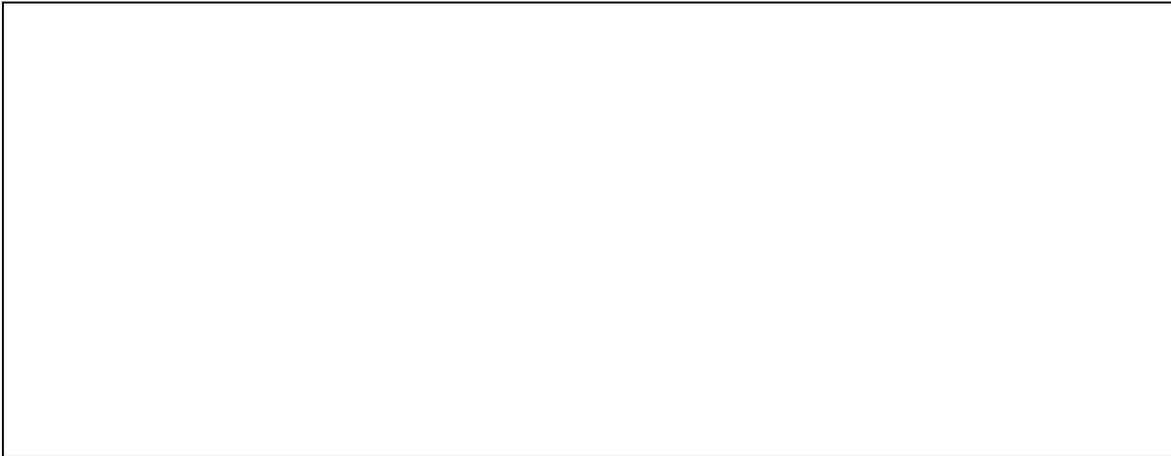


fig. 4.1.127. Menú Agregar instrumentos de Viento.

9. Modifica las propiedades **icon** y **rollOverIcon** del **JButton2** para que aparezca la imagen `clase_swing` y `clase_swing2` respectivamente. Agrega en el mismo el botón su respectivo código para regresar al Mapa mental. (Este procedimiento lo puedes revisar en el punto 1 incisos d y e de la Actividad 12).
10. Agrega el código para centrar el formulario en la pantalla. (Actividad 12, inciso f)
11. En el formulario MapaMental agrega el código correspondiente al botón *Elemento de Menú* (Actividad 12, inciso f)
12. Imprime y pega la pantalla que aparece al ejecutar el programa.



Actividad 24

Código para colocar la imagen y el sonido.

Código para que en la etiqueta `lblInstrumento` aparezca el nombre del instrumento seleccionado y en `lblImagen` se muestre la imagen y se reproduzca el sonido correspondiente al instrumento.

1. Descarga de la librería de audio y agregar sus librerías al proyecto.
 - a. Ir al siguiente link para descargar la librería de audio:

<http://www.javazoom.net/jlgui/api.html>

fig. 4.1.128. Pagina de la librería de audio.

b. Descargar la librería:



dar click en:

[[download](#)]

- c. Se descarga el archivo **basicplayer3.0.zip**, descomprimirlo para obtener la carpeta **BasicPlayer3.0**.
- d. Colocar esta carpeta dentro de la carpeta del proyecto **InterfazGrafica**.
- e. Entrar a propiedades del proyecto **InterfazGrafica**, con botón derecho seleccionar la opción de **Bibliotecas** y en el botón **Añadir JAR/Carpeta** ubicar la carpeta **BasicPlayer3.0**, dar doble clic para ver el contenido.

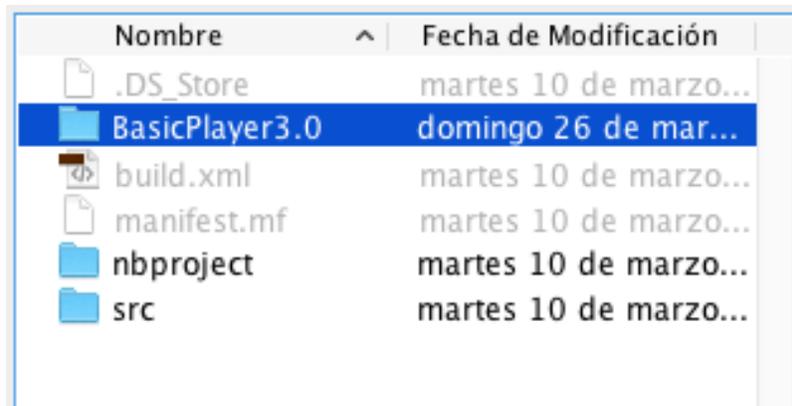


fig. 4.1.129. Carpeta Basic Player3.0.

- f. Seleccionar **basicplayer3.0.jar** y el botón **seleccionar**.

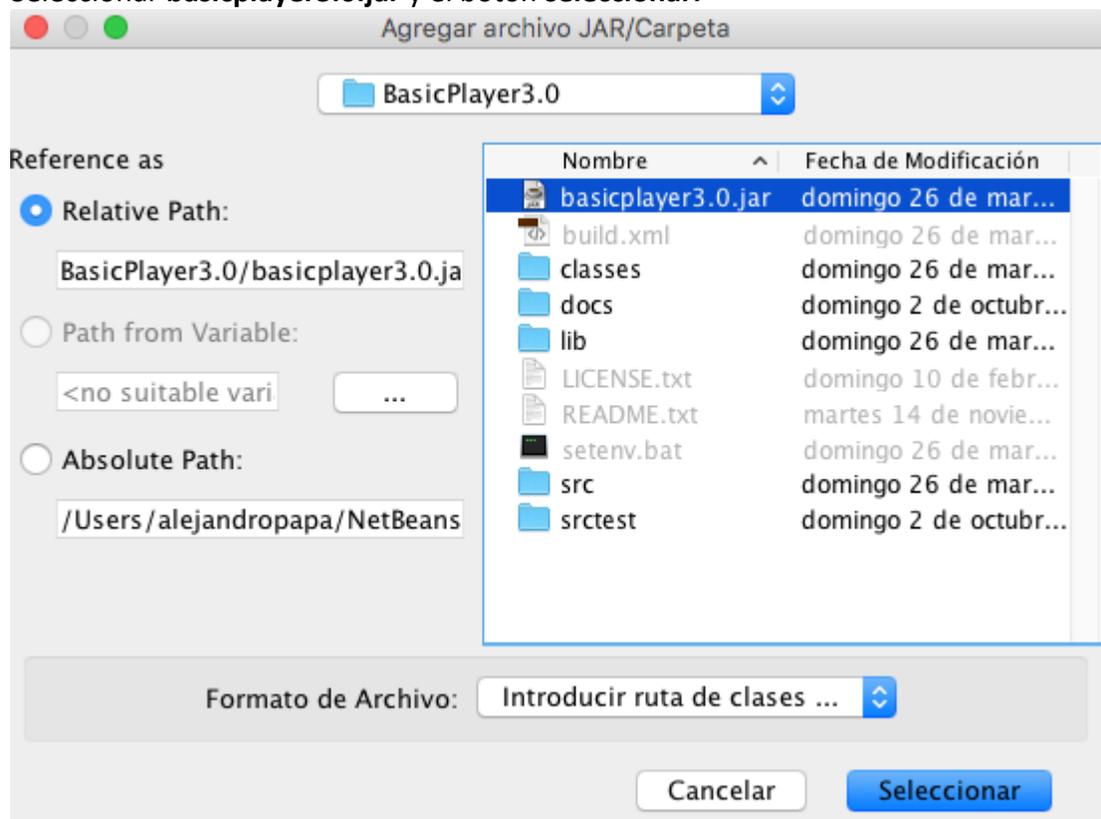


fig. 4.1.130. Seleccionar basicplayer3.0.jar.

- g. Entrar nuevamente a **Añadir JAR/Carpeta** como se indica en el inciso e, dar doble clic sobre la carpeta **lib**

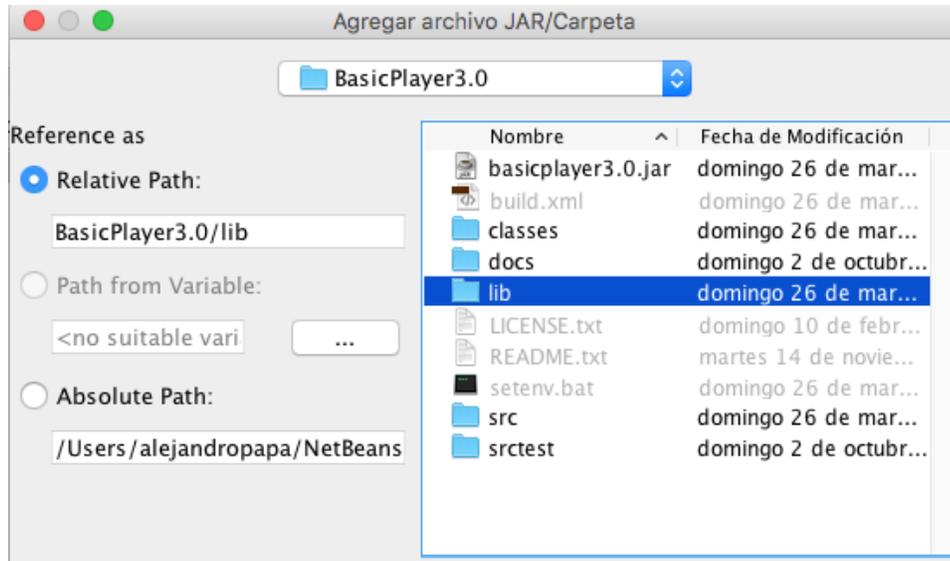


fig. 4.1.131. Seleccionar carpeta lib.

- h. Se mostrará su contenido, seleccionamos todos los archivos arrastrando el mouse con clic sobre ellos y se da clic en **Seleccionar**

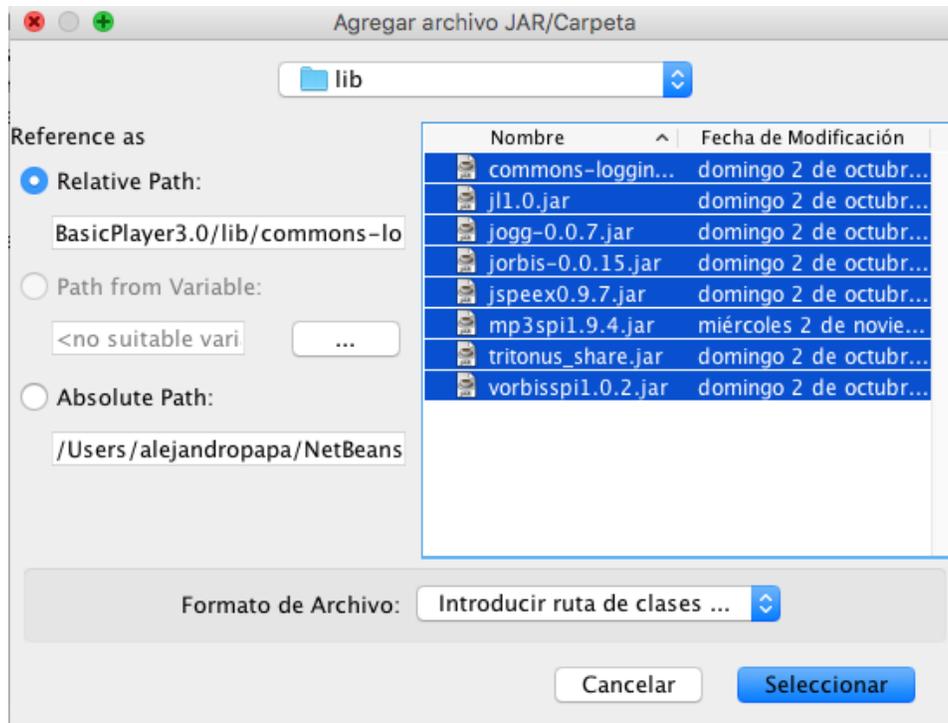


fig. 4.1.132. Selección del contenido.

- i. Se han añadido todos los **Jar** de la carpeta **lib**

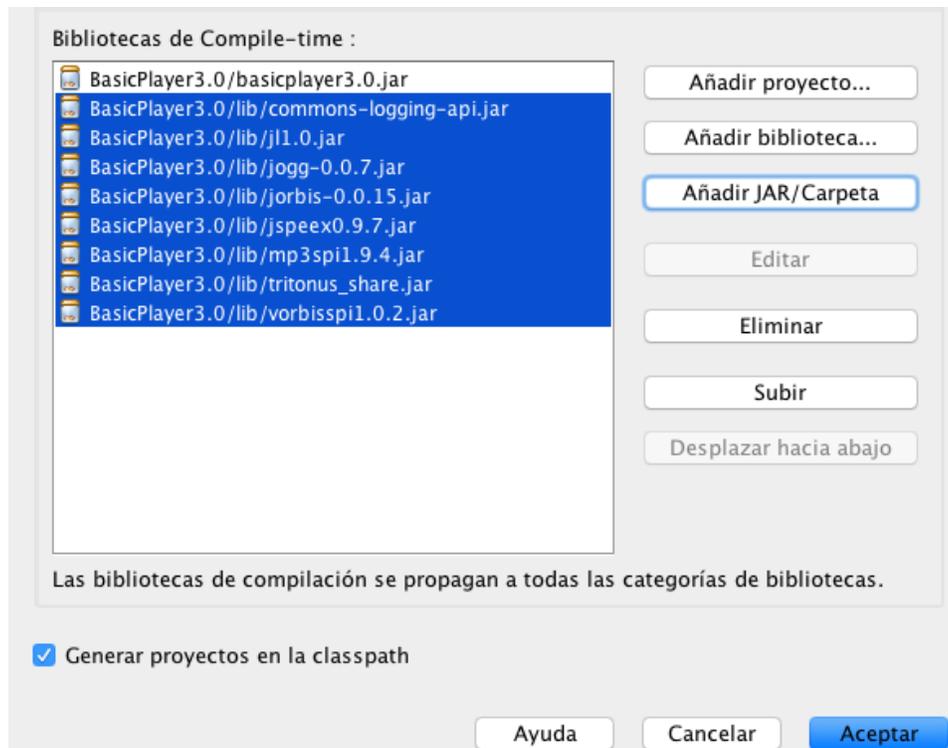


fig. 4.1.133. Integrar la librería al proyecto.

- j. Finalmente se da **Aceptar** para que la librería se integre al proyecto.
2. Desarrollar los siguientes pasos para que se visualice y reproduzca el sonido al dar clic en cada opción de menú.
 - a. Dar clic en la pestaña Source y buscar el método constructor del formulario `MenuItemForm()` lo crea NetBeans. Previamente se define reproductor como un objeto del tipo `BasicPlayer` con acceso privado para luego instanciarlo en el constructor.

```
//Se define reproductor del tipo BasicPlayer
//para poder utilizar sus métodos de manejo de audio
private BasicPlayer reproductor;
```

```
public MenuItemForm() {
    initComponents();
    //Se centra el formulario en la pantalla
    this.setLocationRelativeTo(null);
    //Se instancia reproductor
    reproductor = new BasicPlayer();
}
```

3. Escribir el siguiente código para mostrar la imagen del instrumento musical.

```

//Método para mostrar la imagen del instrumento en una etiqueta
public void muestraImagen(String instrumento) {
    //Se define imagenArchivo para asignarle la ubicación de la imagen
    String imagenArchivo = "/imagenes/" + instrumento + ".png";
    //Se instancia la imagen como un recurso ImageIcon
    ImageIcon imagenInstrumento = new
ImageIcon(getClass().getResource(imagenArchivo));
    //Se instancia la imagen como un ImageIcon con las dimensiones de la
etiqueta
    ImageIcon iconoImagen = new
ImageIcon(imagenInstrumento.getImage().getScaledInstance(this.lblImagen.getWi
dth(), this.lblImagen.getHeight(), Image.SCALE_DEFAULT));
    //Se asigna la imagen a la etiqueta
    this.lblImagen.setIcon(iconoImagen);
}

```

4. Escribir el siguiente código para mostrar la imagen del instrumento musical.

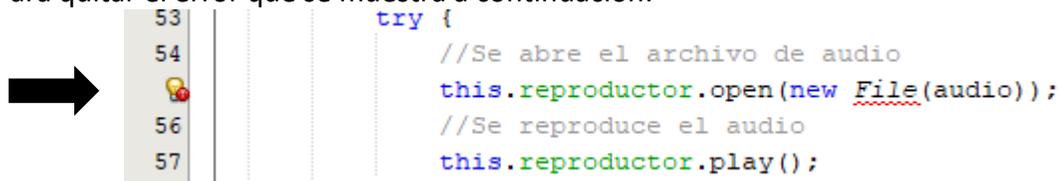
```

//Método para reproducir el audio del instrumento
public void reproduceAudio(String instrumento) {
    //Se define audio como un String para asignarle la ubicación del audio
    String audio = "src/audios/" + instrumento + ".mp3";
    //Iniciamos un try por si se produce un error
    //en los métodos del reproductor de audio
    try {
        //Se abre el archivo de audio
        this.reproductor.open(new File(audio));
        //Se reproduce el audio
        this.reproductor.play();
    } catch (BasicPlayerException ex) {
        //Se muestra el error si no existe el archivo
        //o si no se puede reproducir el audio
        System.out.println("Error: " + ex.getMessage());
    }
}

```

5. Importar la librería para abrir el archivo de audio **File(audio);**

Para quitar el error que se muestra a continuación:

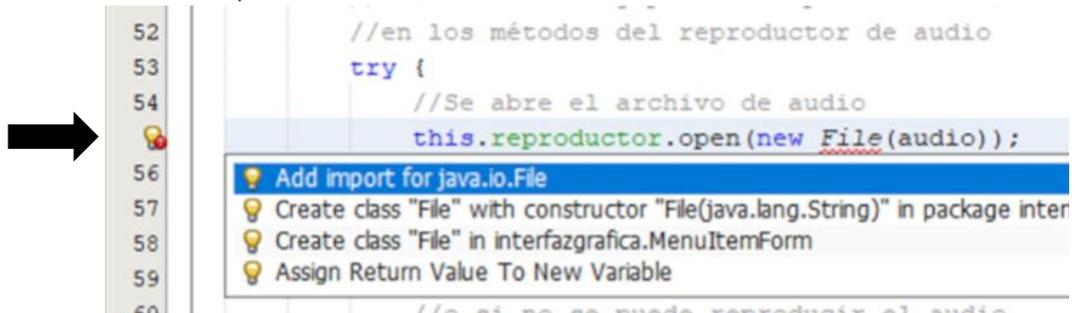


```

53 | | | | | try {
54 | | | | | //Se abre el archivo de audio
55 | | | | | this.reproductor.open(new File(audio));
56 | | | | | //Se reproduce el audio
57 | | | | | this.reproductor.play();

```

- a. Dar clic en el globito del lado izquierdo para que NetBeans importe la librería automáticamente,



Por lo anterior, se agrega la línea **import java.io.File;** automáticamente en la parte superior.

```

8 import java.awt.Image;
9 import java.io.File;
10 import javax.swing.ImageIcon;
    
```

6. Escribir el código de cada menú dando doble clic en el nombre del instrumento y agregar:

```

this.lblInstrumento.setText("Seleccionaste: " + this.menuGuitarra.getText());
muestraImagen(this.menuGuitarra.getText());
reproduceAudio(this.menuGuitarra.getText());
    
```

El código resultante de cada instrumento será:

```

private void menuGuitarraActionPerformed(java.awt.event.ActionEvent evt) {
    this.lblInstrumento.setText("Seleccionaste: " + this.menuGuitarra.getText());
    muestraImagen(this.menuGuitarra.getText());
    reproduceAudio(this.menuGuitarra.getText());
}
private void menuViolinActionPerformed(java.awt.event.ActionEvent evt) {
    this.lblInstrumento.setText("Seleccionaste: " + this.menuViolin.getText());
    muestraImagen(this.menuViolin.getText());
    reproduceAudio(this.menuViolin.getText());
}
    
```

Revisando el código anterior de los instrumentos Guitarra y Violín, escribe el correspondiente a los instrumentos Arpa, Piano, Batería, Marimba, Panderó. Maracas, Trompeta, Flauta, Oboe y Clarinete.

```
private void menuArpaActionPerformed(java.awt.event.ActionEvent evt) {
```

```
    _____  
    _____  
    _____
```

```
}
```

```
private void menuPianoActionPerformed(java.awt.event.ActionEvent evt) {
```

```
    _____  
    _____  
    _____
```

```
}
```

```
private void menuBateriaActionPerformed(java.awt.event.ActionEvent evt) {
```

```
    _____  
    _____  
    _____
```

```
}
```

```
private void menuMarimbaActionPerformed(java.awt.event.ActionEvent evt) {
```

```
    _____  
    _____  
    _____
```

```
}
```

```
private void menuPanderoActionPerformed(java.awt.event.ActionEvent evt) {
```

```
    _____  
    _____  
    _____
```

```
}
```

```
private void menuMaracasActionPerformed(java.awt.event.ActionEvent evt) {
```

```
    _____  
    _____  
    _____
```

```
}
```

```
private void menuTrompetaActionPerformed(java.awt.event.ActionEvent evt) {
```

```
    _____  
    _____  
    _____
```

```
}
```

```
private void menuFlautaActionPerformed(java.awt.event.ActionEvent evt) {
```

```
}

```

```
private void menuOboeActionPerformed(java.awt.event.ActionEvent evt) {
```

```
}

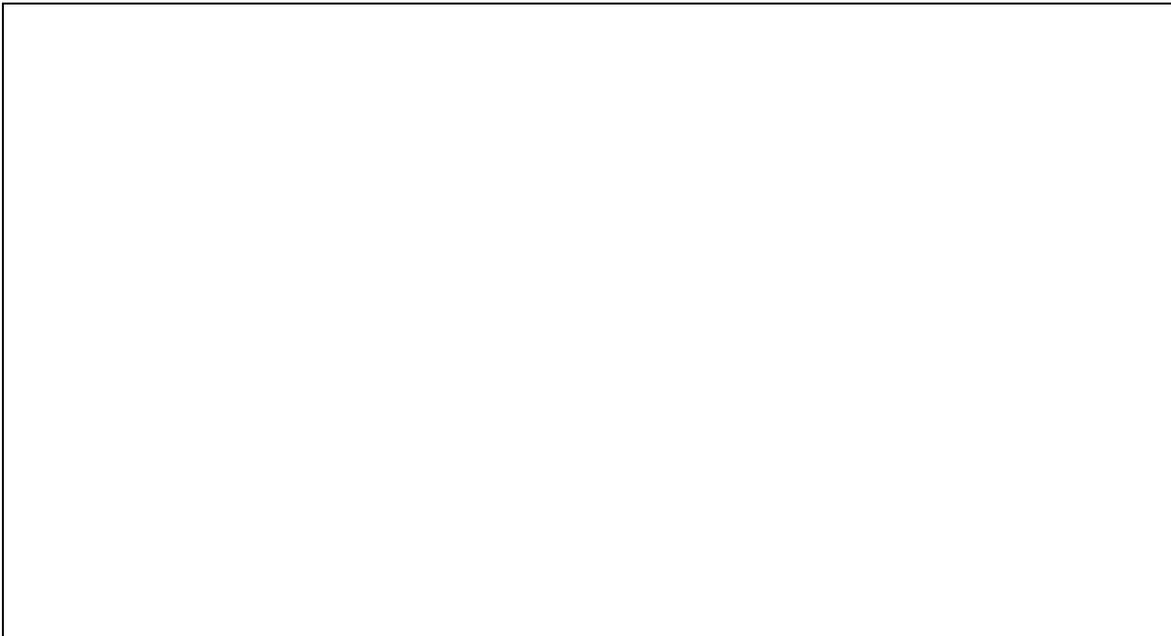
```

```
private void menuClarineteActionPerformed(java.awt.event.ActionEvent evt) {
```

```
}

```

7. Imprime la pantalla que te resulto al ejecutar el programa.



7. Envía la aplicación por correo electrónico a tu profesor

Actividad 25

Crucigrama

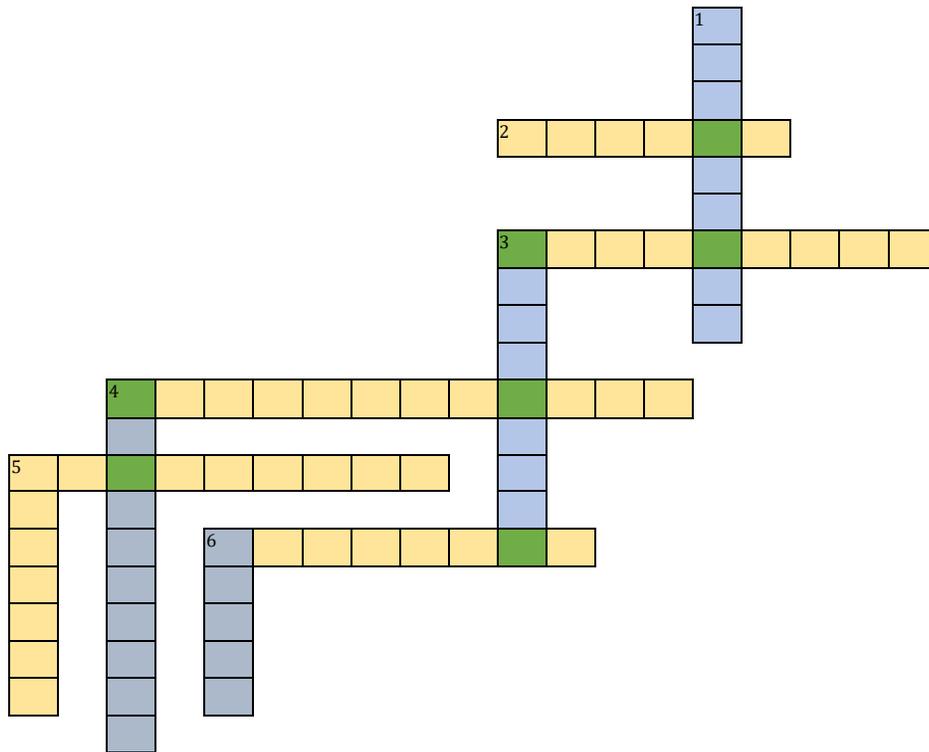
Verticales

1. Este componente nos permite seleccionar una o varias opciones de una lista.

3. Vincula un área de múltiples líneas de texto en donde se ingresa o se muestra información.
4. Es un componente que permite al usuario introducir un dato.
5. Se utiliza para crear un botón dentro de una aplicación de interfaz gráfica.
6. Este elemento es utilizado para incluir un menú.

Horizontales

2. Es un elemento que muestra una sola línea de texto de solo lectura.
3. Permite al usuario seleccionar un artículo de una lista desplegable.
4. Este componente despliega un pequeño círculo con una etiqueta y aparecen en grupos.
5. Es un elemento del menú y al ser pulsado genera un evento.
6. Es muy útil ya que representa la barra de menú que vemos siempre en todo programa.



4.2 Clase Graphics

Aprendizajes esperados.

El alumno:

- ✓ Elabora programas con interfaz gráfica de usuario aplicando las Clases: setColor, drawLine, drawRect, drawRoundRect, drawOval, drawPolygon.
- ✓ Elabora programas con interfaz gráfica de usuario aplicando las Clases: fillRect, fillRoundRect, fillOval, fillPolygon.

Contenido temático.

Clase Graphics:

- setColor.
- drawLine.
- drawRect.
- drawRoundRect.
- drawOval.
- drawPolygon.

Clase Graphics

- fillRect.
- fillRoundRect.
- fillOval.
- fillPolygon.

Objetivo:

Que el alumno conozca la Clase Graphics y emplee los métodos necesarios para elaborar dibujos por medio de programas en Java.

4.2.1 Introducción

El lenguaje de programación Java contiene herramientas que permiten realizar diversas figuras en un entorno gráfico. Para ello utiliza como base la clase AWT de la cual se hereda la clase Graphics. Mediante ella, es posible elaborar figuras básicas como líneas, rectángulos, elipses o polígonos. Adicionalmente se pueden cambiar los colores, tanto del fondo como del frente.

Para realizar figuras, Java utiliza un plano cartesiano que, con el uso de coordenadas, es posible definir un inicio y un fin para una figura en concreto. Adicionalmente es importante recordar que la unidad básica para definir las coordenadas son los pixeles.

En este apartado se hace énfasis en los métodos básicos de la clase Graphics entre los cuales podemos mencionar: setColor, drawLine, drawRect, drawRoundRect, drawOval, drawPolygon, fillRect, fillRoundRect, fillOval, fillPolygon.

4.2.2 Clase Graphics

Esta clase nos proporciona los elementos necesarios para desarrollar programas, construir gráficos y figuras independientemente del sistema operativo. Todos los métodos utilizados tienen argumentos que representan puntos iniciales y finales de coordenadas en un plano cartesiano, esquinas o ubicaciones iniciales del objeto representados como valores en el sistema de coordenadas de Java.

A continuación, se muestran los métodos más utilizados para dibujar:

| Método | Descripción |
|---|--|
| setColor(Color c) | Establece un color actual. |
| drawLine(int x1, int y1, int x2, int y2) | Dibuja una línea entre dos puntos. |
| drawRect(int x, int y, int anchura, int altura) | Dibuja las líneas de un rectángulo. |
| drawRoundRect(int x, int y, int anchura, int altura, int anchoArc, int altoArc) | Dibuja un rectángulo con las esquinas redondeadas. |
| drawOval(int x, int y, int anchura, int altura) | Dibuja el contorno de un óvalo. |
| drawPolygon(int[] x, int[] y, int puntosN) | Dibuja las líneas de un polígono cerrado. |
| fillRect(int x, int y, int anchura, int altura) | Dibuja un rectángulo relleno. |
| fillRoundRect(int x, int y, int anchura, int altura, int anchoArc, int altoArc) | Dibuja un rectángulo relleno y con las esquinas redondeadas. |
| fillOval(int x, int y, int anchura, int altura) | Dibuja un óvalo relleno de color. |
| fillPolygon(int[] x, int[] y, int puntosN) | Dibuja un polígono cerrado y relleno. |

El sistema de coordenadas de Java, se compone de la siguiente forma, tiene el origen (0,0) en la esquina superior izquierda. Los valores positivos de X están a la derecha, y los valores positivos de Y están hacia abajo. Todos los valores de pixeles son enteros, no existen pixeles parciales o fraccionarios por ejemplo la coordenada (10,10). (figura 4.2.1)

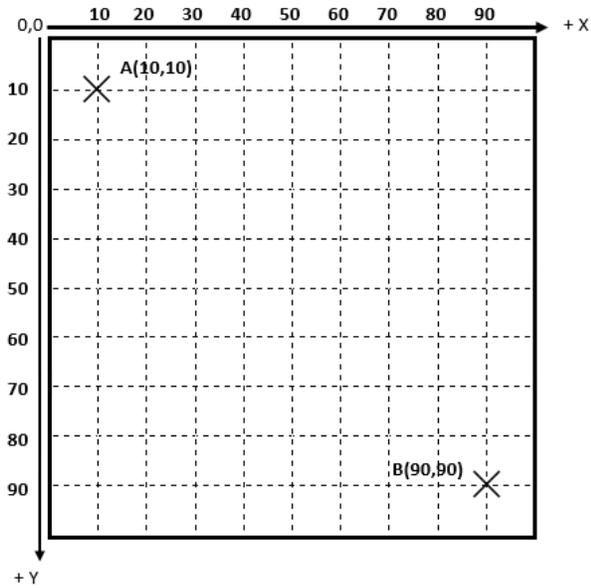


fig. 4.2.1. Sistema de coordenadas gráficas de Java.

Actividad 26

Coordenadas gráficas de Java.

Con base en la imagen que se muestra en la figura 4.2.2, responde las siguientes preguntas:

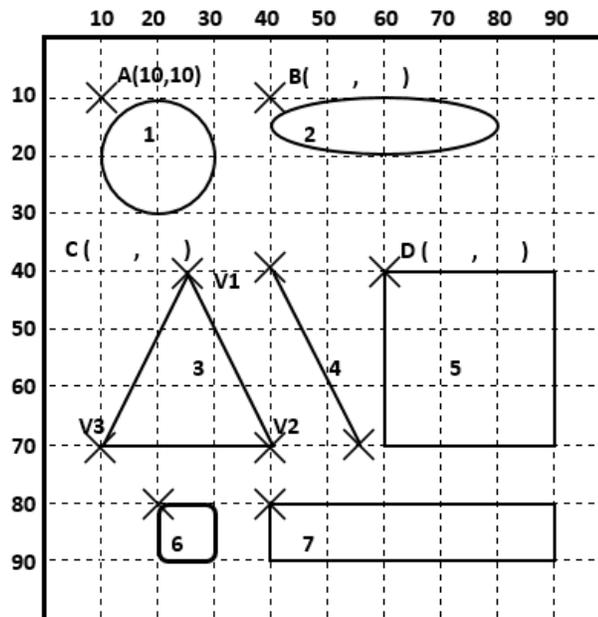


fig. 4.2.2. Actividad 26.

1.1 ¿Qué figuras geométricas observas en la figura 4.2.2?

1.2 De acuerdo con las coordenadas del punto A ¿Cuáles son las coordenadas de los puntos B, C y D?

1.3 Llena las siguientes tablas con los datos que se piden:

| Figura número | Nombre de la figura | Coordenadas de la esquina superior izquierda | Tamaño del ancho | Tamaño de lo alto |
|---------------|---------------------|--|------------------|-------------------|
| 1 | | (10,10) | 20 | 20 |
| 2 | | (40,10) | | |
| 5 | | | 30 | |
| 6 | | | | 10 |
| 7 | | | 50 | |

Tabla 4.2.1.

| Figura número | Nombre de la figura | Coordenadas del punto inicial | Coordenadas del punto final |
|---------------|---------------------|-------------------------------|-----------------------------|
| 4 | | (40,40) | |

Tabla 4.2.2.

| Figura número | Nombre de la figura | Número de vértices | Coordenadas del vértice V1 | Coordenadas del vértice V2 | Coordenadas del vértice V3 |
|---------------|---------------------|--------------------|----------------------------|----------------------------|----------------------------|
| 3 | | 3 | | | (10,70) |

Tabla 4.2.3.

4.2.3 Método setColor

Tomando la tabla 4.2.4. como referencia, este método define el color para dibujar utilizando la clase Color que se explicó en la clase Swing.

```
public void setColor(Color c)
```



| Argumento de color | Color |
|--------------------|-------------|
| Color.black | Negro |
| Color.blue | Azul |
| Color.cyan | Cian |
| Color.darkGray | Gris oscuro |
| Color.gray | Gris |
| Color.green | Verde |
| Color.lightGray | Gris claro |
| Color.magenta | Magenta |
| Color.orange | Naranja |
| Color.pink | Rosa |
| Color.red | Rojo |
| Color.white | Blanco |
| Color.yellow | Amarillo |

Tabla 4.2.4. Argumentos de color.

4.2.4 Método drawLine

Para dibujar una línea recta.

```
public void drawLine(int x1, int y1, int x2, int y2)
```

Donde x1, y1 son las coordenadas del punto inicial y x2, y2 las del punto final.

Para dibujar la línea de la figura 4.2.3, los parámetros son los siguientes:

```
drawLine (10, 10, 40, 40);
```

Punto inicial
Punto final

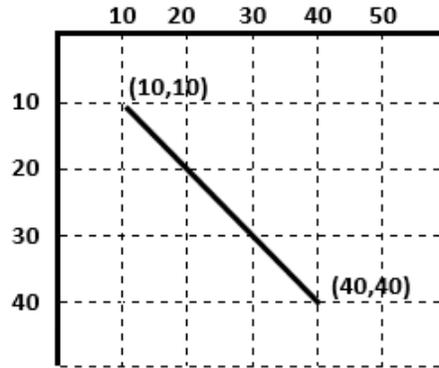


fig. 4.2.3. Línea recta.

4.2.5 Método drawRect



Para dibujar un rectángulo, se utiliza el método siguiente:

```
public void drawRect(int x, int y, int anchura, int altura)
```

Para dibujar el rectángulo de la figura 4.2.4, los parámetros son los siguientes:

```
drawRect (10, 10, 40, 30);
```

Punto de inicio
anchura
altura

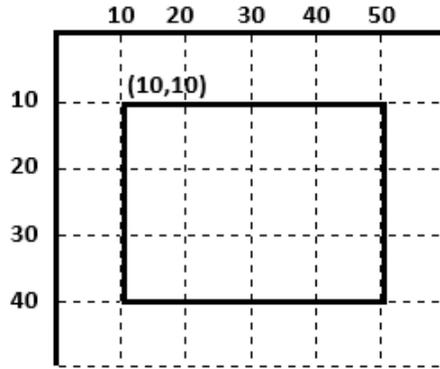


fig. 4.2.4. Rectángulo.

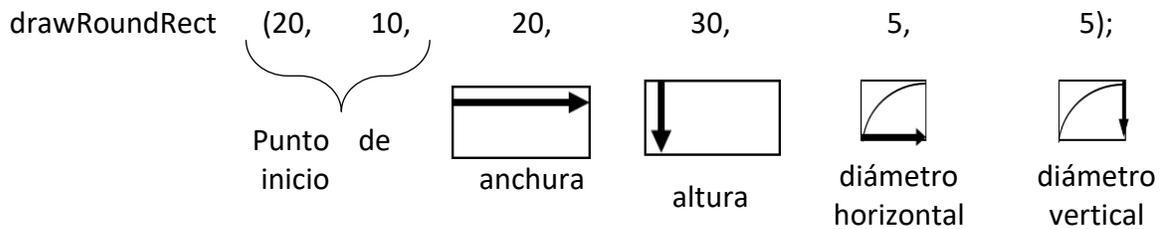
4.2.6 Método drawRoundRect

Para dibujar un rectángulo con las esquinas redondeadas se utiliza el método siguiente:

```
public void drawRoundRect(int x, int y, int anchura, int altura, int arcAncho, int arcAlto)
```

En donde arcAncho es el diámetro horizontal del arco en las cuatro esquinas y arcAlto es el diámetro vertical.

Para dibujar el rectángulo con esquinas redondeadas de la figura 4.2.5, los parámetros son los siguientes:



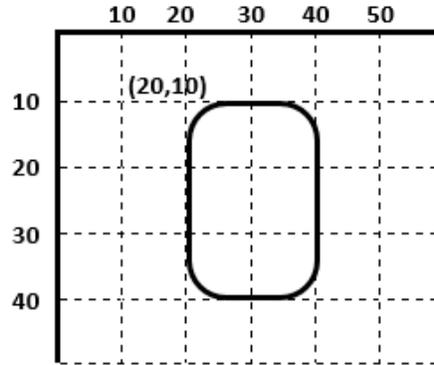


fig. 4.2.5. Rectángulo con las esquinas redondeadas

El valor de **arcAncho** es la distancia de una esquina del rectángulo hacia el eje horizontal y el valor de **arcAlto** es la distancia de la esquina sobre el eje vertical. Para un rectángulo de ancho 60 y alto 30 como el que muestra la siguiente figura, podemos ver dos tipos de esquinas redondeadas.

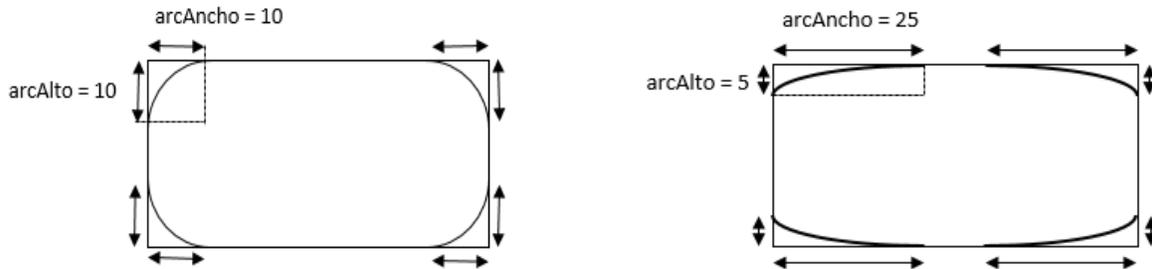


fig. 4.2.6. Ejemplo de arcAncho y arcAlto.

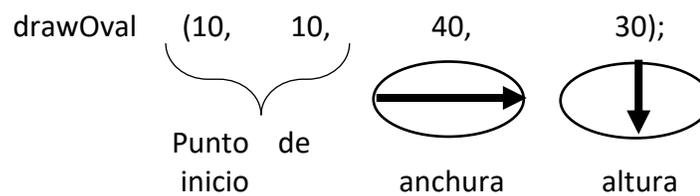
4.2.7 Método drawOval



Para dibujar una elipse o círculo si la anchura y altura son iguales, se utiliza el siguiente método:

```
public void drawOval(int x, int y, int anchura, int altura)
```

Para dibujar una elipse o círculo como la de la figura 4.2.7, los parámetros son los siguientes:



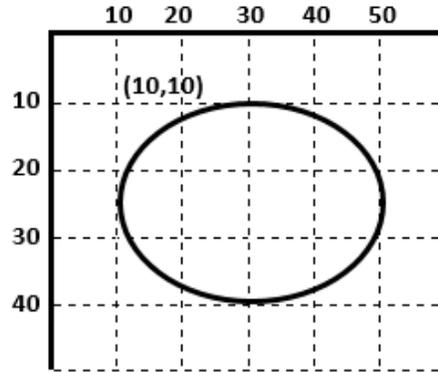


fig. 4.2.7. Elipse o círculo.

4.2.8 Método drawPolygon

Para dibujar un polígono cerrado, se utiliza el método siguiente:

```
public void drawPolygon(int X[ ], int Y[ ], int puntos)
```

En donde X, Y, son arreglos donde se almacenan pares de coordenadas.

En la figura 4.2.8 observamos los vértices y las coordenadas de cada uno de acuerdo con la tabla 4.2.5.:

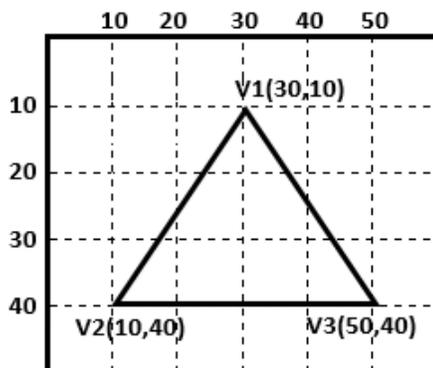


fig. 4.2.8. Polígono cerrado.

| Puntos | x[] | y[] |
|-----------|------|------|
| V1(x0,y0) | 30 | 10 |
| V2(x1,y1) | 10 | 40 |
| V3(x2,y2) | 50 | 40 |

Tabla 4.2.5. Vértices del triángulo

Se definen los arreglos de coordenadas con los valores de la tabla cinco; un arreglo de tipo entero para los valores de la columna x[]={x0,x1,x2} y otro arreglo también de tipo entero para los de la columna y[]={y0,y1,y2}. Los parámetros quedarían así:

```
int[ ] x={30,10,50};
int[ ] y={10,40,40};
```

Para dibujar el triángulo de la figura 4.2.8, los parámetros son los siguientes:

drawPolygon (x, y, 3);
 Arreglo de los puntos en x Arreglo de los puntos en y Número de puntos

Actividad 27

Aplicando los métodos.

Desarrolla el procedimiento para dibujar las figuras de la siguiente imagen, utilizando los métodos de Java explicados anteriormente.

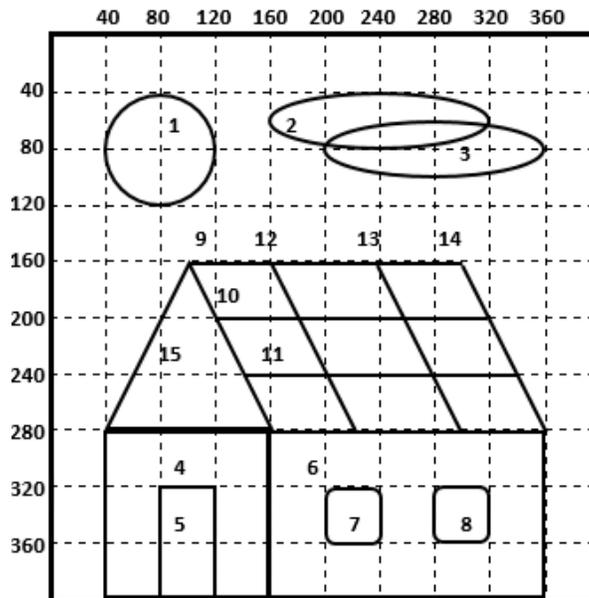


fig. 4.2.9. Actividad 27.

- Escribir los datos que faltan en las siguientes tablas, tomando como referencia la imagen mostrada en figura 4.2.9.

| Núm. | Figura | Color | Método | x | y | Anchura | Altura |
|------|--------|----------|----------|-----|----|---------|--------|
| 1 | | amarillo | drawOval | 40 | | | 80 |
| 2 | | azul | drawOval | 160 | 40 | | |
| 3 | | azul | drawOval | 200 | | 160 | |
| 4 | | azul | drawRect | | | 120 | 120 |

| | | | | | | | |
|---|--|-------------|----------|----|-----|-----|--|
| 5 | | Gris oscuro | drawRect | 80 | | 40 | |
| 6 | | morado | drawRect | | 280 | 200 | |

Tabla 4.2.6.

| Núm. | Figura | Color | Método | x | y | Anchura | Altura | Arco Anchura | Arco Altura |
|------|--------|---------|---------------|-----|-----|---------|--------|--------------|-------------|
| 7 | | naranja | drawRoundRect | 200 | 320 | | | 5 | 5 |
| 8 | | naranja | | | | 40 | 40 | | |

Tabla 4.2.7.

| Núm. | Figura | Color | Método | x1 | y1 | x2 | y2 |
|------|--------|-------|----------|-----|-----|-----|-----|
| 9 | | rojo | drawLine | 100 | | 300 | |
| 10 | | rojo | drawLine | | 200 | | 200 |
| 11 | | rojo | drawLine | 140 | | | 240 |
| 12 | | rojo | drawLine | 160 | 160 | | |
| 13 | | rojo | drawLine | | | 300 | 280 |
| 14 | | rojo | drawLine | | | | |

Tabla 4.2.8.

| Núm. | Figura | Color | Método | Vértice 1 | Vértice 2 | Vértice 3 | Puntos |
|------|--------|-------|--------|------------------|--------------|---------------|--------|
| 15 | | verde | | x1=100 y1=160 | x2=40 y2= | x2= y2=280 | 3 |

Tabla 4.2.9.

- b. Abrir el programa NetBeans y crear un nuevo proyecto que se llame **Graficos**, con un paquete que se llame **Figuras** y un JFrame que se llame **Dibujo**.
- c. Diseñar y programar una ventana con título **Dibujo**, de tamaño **500 X 500** y que no puedan modificarse sus dimensiones.
- d. Cambiar a la pestaña **Source** para importar las clases Graphics y Color antes de la clase Dibujo.

```
import java.awt.Color;
import java.awt.Graphics;
```

- e. Escribir el método paint después del método constructor, y llamar al atributo paint pasando el parámetro casa:

```
public void paint(Graphics casa)
{
```

```
super.paint(casa);
```

- f. Dibujar el círculo, definiendo el color y tomando los datos de la tabla 4.2.6:

```
casa.setColor(Color.yellow);
casa.drawOval(40, _____, _____, 80);
```

- g. Dibujar las elipses, definiendo el color y tomando los datos de la tabla 4.2.6:

```
casa.setColor(Color.blue);
casa.drawOval(160, 40, _____, _____);
casa.drawOval(200, _____, 160, _____);
```

- h. Dibujar el cuadrado, dejando el color azul y tomando los datos de la tabla 4.2.6:

```
casa.drawRect(_____, _____, 12,120);
```

- i. Dibujar el rectángulo, definiendo el color café y tomando los datos de la tabla 4.2.6:

```
casa.setColor(Color.darkGray)
casa.drawRect(80, _____, 40, _____);
```

- j. Dibujar el rectángulo, definiendo el color morado y tomando los datos de la tabla 4.2.6:

```
casa.setColor(Color.magenta)
casa.drawRect(_____, 280, 200, _____);
```

- k. Dibujar los cuadrados de color naranja y los datos de la tabla 4.2.7:

```
casa.setColor(Color._____);
casa.drawRoundRect(200,320, _____, _____, 5,5);
casa.drawRoundRect(_____, _____, 40,40, _____, _____);
```

- l. Dibujar las líneas definiendo el color y tomando las coordenadas de la tabla 4.2.8:

```
casa.setColor(Color._____);
casa.drawLine(100, _____, 300, _____);
casa.drawLine(_____, 200, _____, 200);
casa.drawLine(140, _____, _____, 240);
casa.drawLine(160, 160, _____, _____);
casa.drawLine(_____, _____, 300, 280);
casa.drawLine(_____, _____, _____, _____);
```

m. Declarar los arreglos X y Y, dibujar el triángulo con los datos de la tabla 4.2.9:

```

casa.setColor(Color._____);
int[] x={100,40,_____};
int[] y={160,_____,280};
casa._____(x,y,3);
    
```

n. Al ejecutar el programa se debe obtener una imagen como la que se muestra en la figura 4.2.10, si esto no ocurre revisar el programa, corregir y volver a ejecutar.

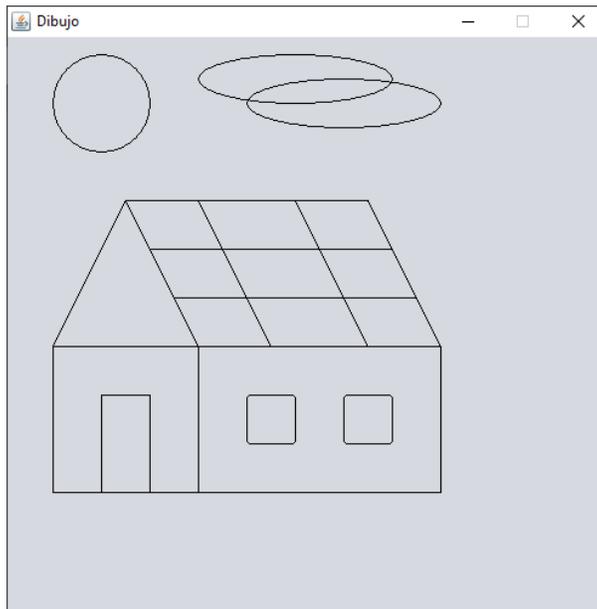


fig. 4.2.10. Salida de la Actividad 27.

Desafío 2

Diseño de imágenes con los métodos.

Dibujar las siguientes imágenes en un JFrame con nombre Dibujo2, Dibujo3 y las siguientes características.

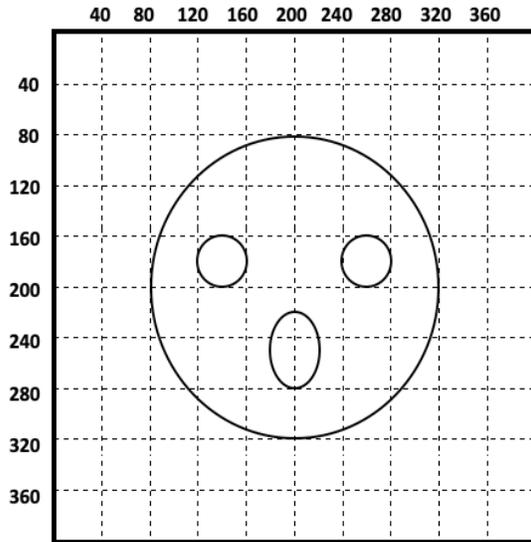


fig. 4.2.11. Dibujo 2.

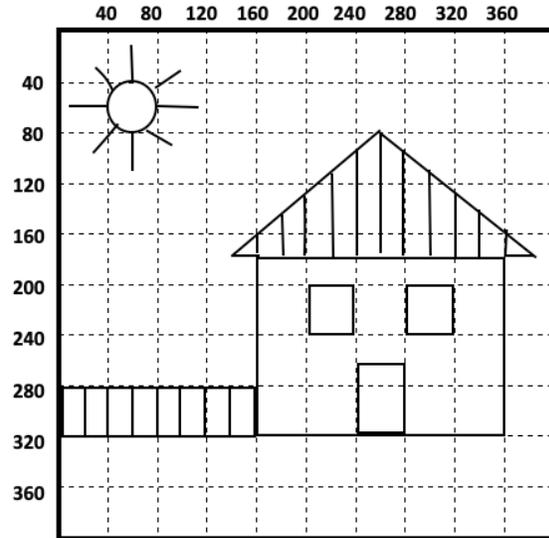


fig. 4.2.12. Dibujo 3.

Imprime la pantalla de las figuras obtenidas y pégalas en este espacio.



4.2.9 Método fillRect

Para dibujar un rectángulo relleno se utiliza el método siguiente:

```
public void fillRect(int x, int y, int anchura, int altura)
```

Para dibujar el rectángulo de la figura 4.2.13 los parámetros son los siguientes:

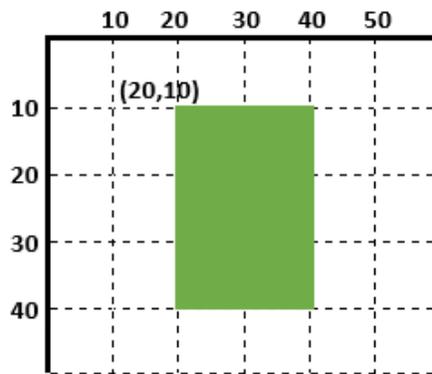
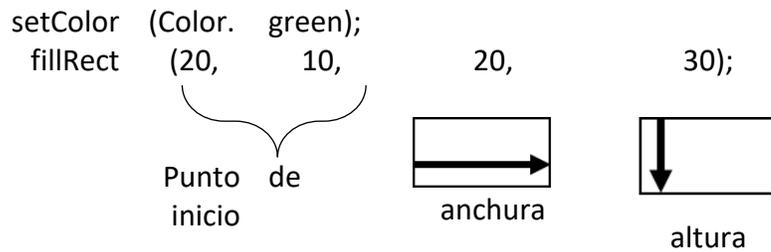


fig. 4.2.13. Rectángulo relleno.

4.2.10 Método fillRoundRect

Para dibujar un rectángulo relleno con las esquinas redondeadas se utiliza el método siguiente:

```
public void fillRoundRect(int x, int y, int anchura, int altura, int anchuraArco, int alturaArco)
```

Para dibujar el rectángulo de la figura 4.2.14 los parámetros son los siguientes:

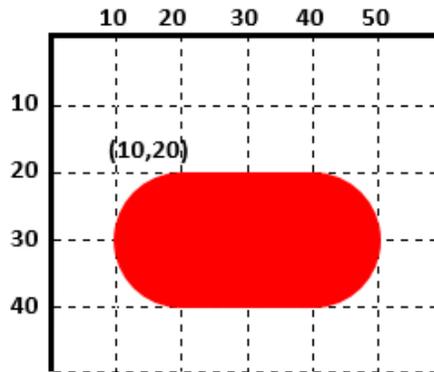
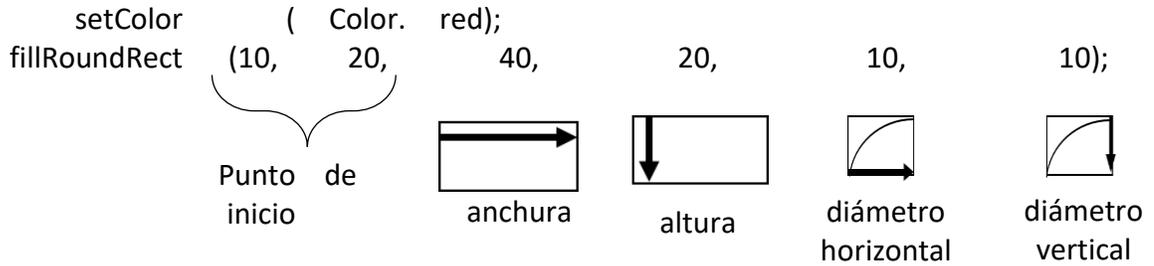


fig. 4.2.14. Rectángulo relleno con las esquinas redondeadas.

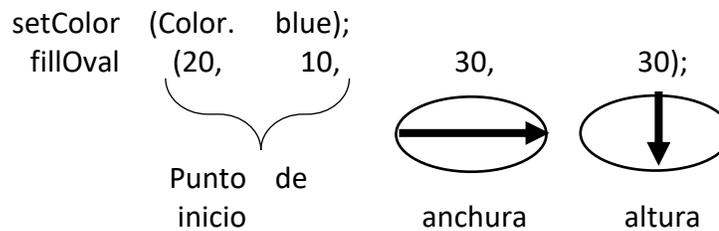
4.2.11 Método fillOval

Para dibujar una elipse o círculo relleno se utiliza el método siguiente:

```

public void fillOval(int x, int y, int anchura, int altura)
    
```

Para dibujar una elipse o círculo de la Figura 4.2.15. los parámetros son los siguientes:



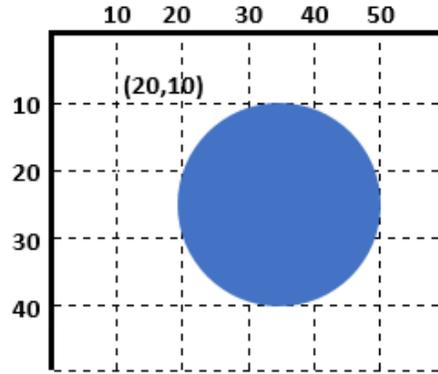


fig. 4.2.15. Elipse o círculo relleno.

4.2.12 Método fillPolygon

Para dibujar un polígono cerrado relleno se utiliza el método siguiente:

```
public void fillPolygon(int X[ ], int Y[ ], int puntos)
```

En la figura 4.2.16. observamos los vértices de un polígono cerrado y las coordenadas de cada uno de acuerdo con la tabla 4.2.10.:

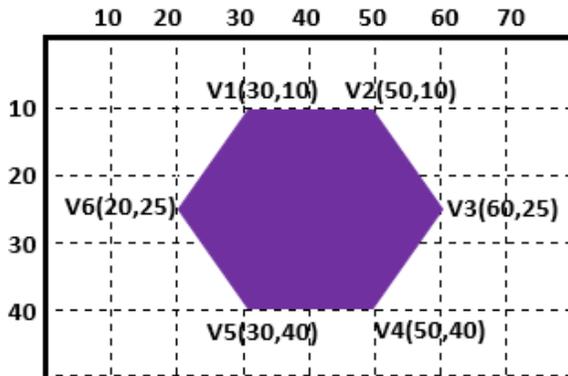


fig. 4.2.16. Polígono cerrado relleno

| Puntos | x[] | y[] |
|-----------|------|------|
| V1(x0,y0) | 30 | 10 |
| V2(x1,y1) | 50 | 10 |
| V3(x2,y2) | 60 | 25 |
| V4(x3,y3) | 50 | 40 |
| V5(x4,y4) | 30 | 40 |
| V6(x5,y5) | 20 | 25 |

Tabla 4.2.10. Vértices del hexágono

Se definen los arreglos de coordenadas con los valores de la tabla anterior; un arreglo de tipo entero para los valores de la columna x[]={x0,x1,x2,x3,x4,x5} y otro arreglo también de tipo entero para los de la columna y[]={y0,y1,y2,y3,y4,y5}. Los parámetros quedarían así:

```
int[ ] x={30,50,60,50,30,20};
int[ ] y={10,10,25,40,40,25};
```

```

setColor (Color.magenta)
drawPolygon (x, y, 6);

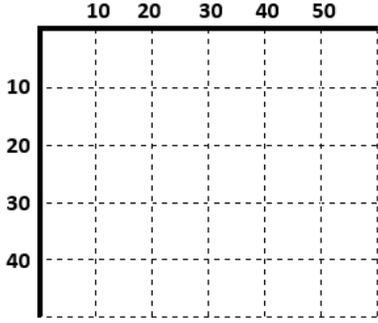
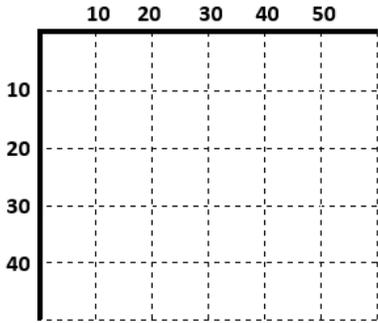
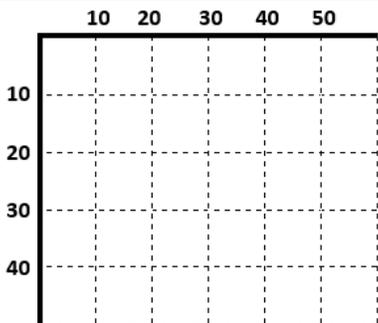
```

Arreglo de los puntos en x
Arreglo de los puntos en y
Número de puntos

Actividad 28

Aplicando métodos para colorear figuras.

Dibuja y colorea las siguientes figuras en la cuadrícula de la columna **Figura** utilizando colores de relleno según el código empleado.

| Código | Figura |
|--|--|
| <pre> setColor(Color.black); fillRect(10,30,30,10); </pre> |  |
| <pre> setColor(Color.pink); fillOval(30,10,30,10); </pre> |  |
| <pre> int[] x={30,33,50,38,42,30,18,22,10,17}; int[] y={5,20,20,30,45,38,45,30,20,20}; setColor(Color.yellow); fillPolygon(x,y,10); </pre> |  |

| | |
|--|--|
| <pre>setColor(Color.gray); fillRoundRect(10,10,40,30,20,15);</pre> | |
|--|--|

Actividad 29

Aplicando métodos para dibujar y colorear figuras.

Utilizando el programa elaborado en la Actividad 2 colorea la misma imagen, como se muestra en la figura 4.2.17 y para el techo de la casita, dibuja un polígono de cuatro lados de color rosa.

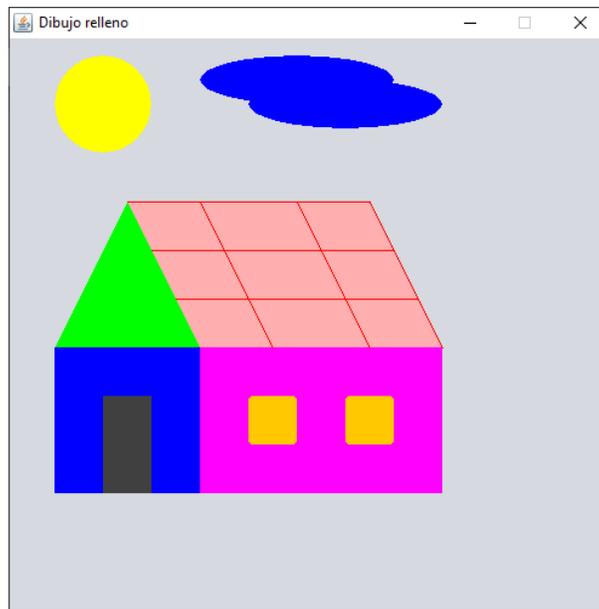
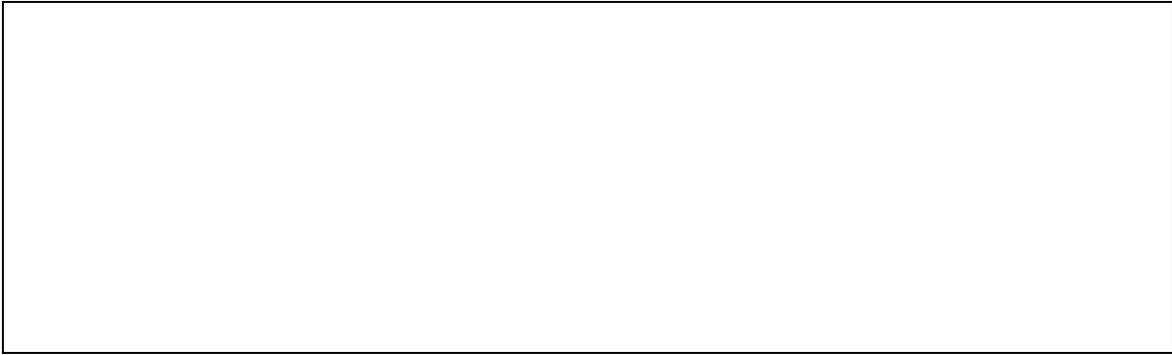


fig. 4.2.17. Salida de la Actividad 27.

Imprime la pantalla de la figura obtenida y pégala en este espacio.



Desafío 3

Relleno de color de imágenes.

Desarrollar los programas correspondientes para obtener imágenes como las mostradas en las figuras 4.2.18 y 4.2.19.

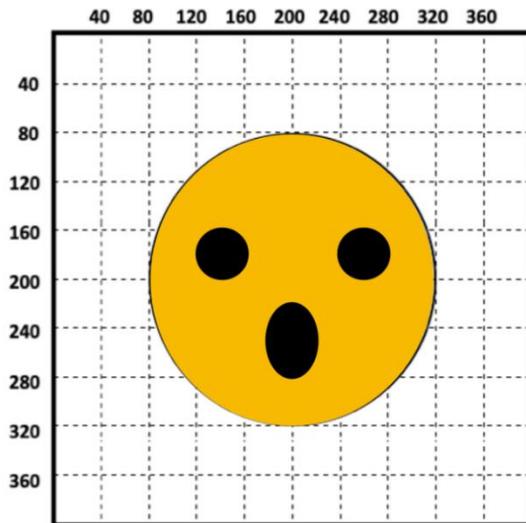


fig. 4.2.18. Dibujo 4.

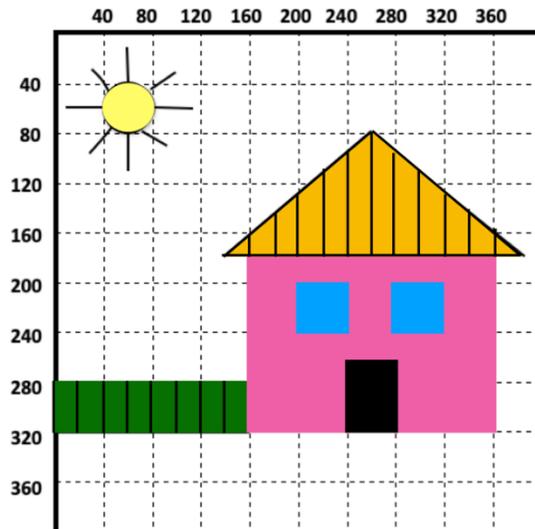
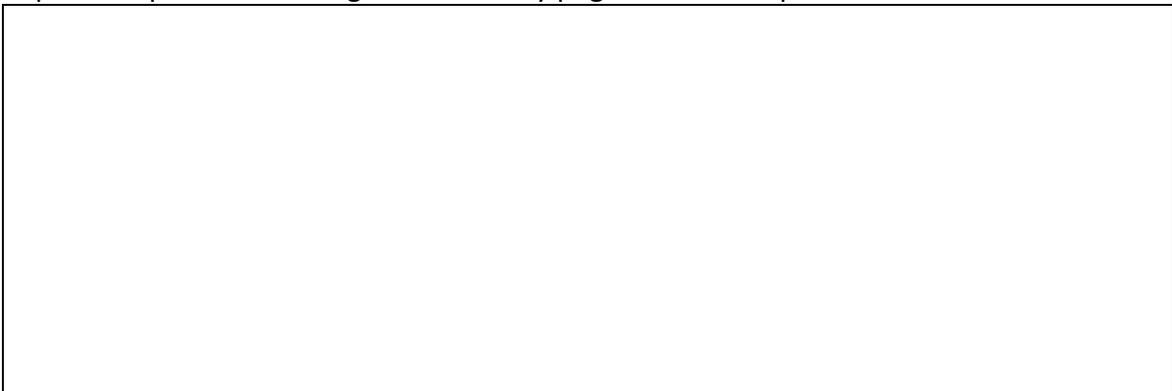


fig. 4.2.19. Dibujo 5

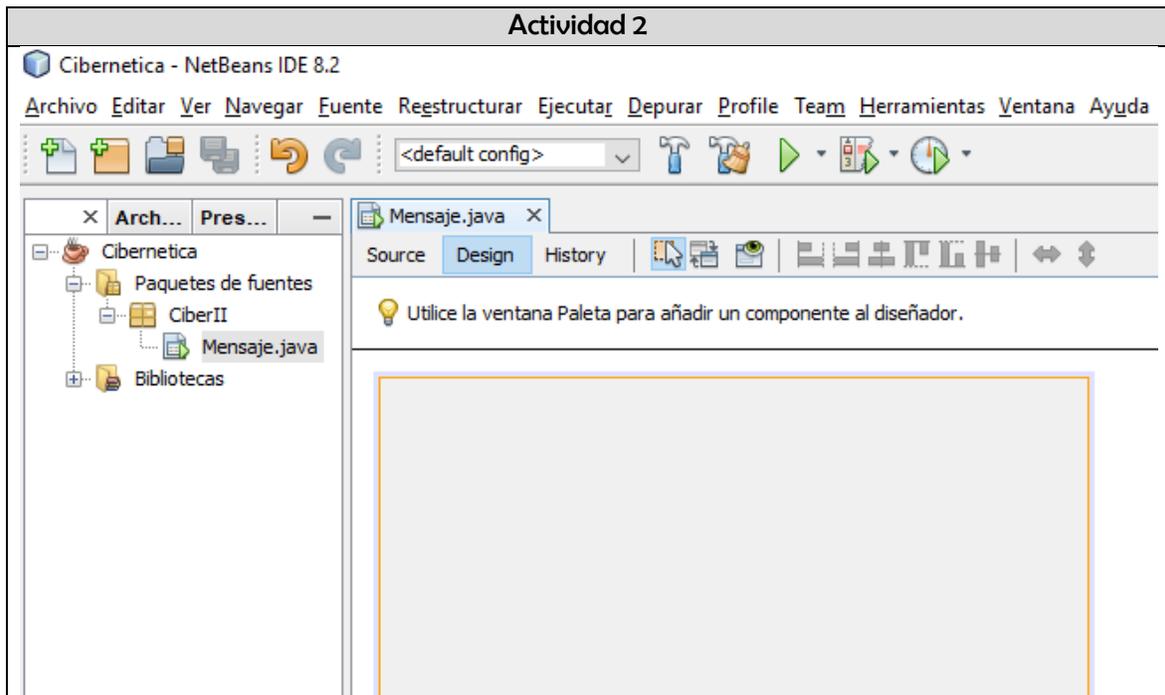
Imprime la pantalla de la figura obtenida y pégala en este espacio.



4.3 Soluciones

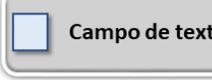
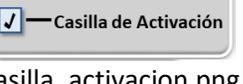
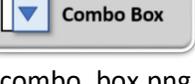
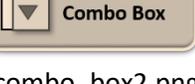
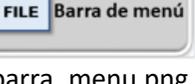
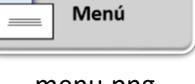
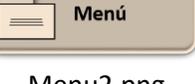
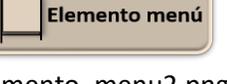
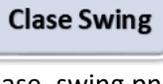
Actividades

| Actividad 1 | |
|--|---|
| 1.1. Interfaz Gráfica de Usuario, o Graphic User Interface por sus siglas en inglés. | 1.2. Una biblioteca de clases para crear interfaces gráficas de usuarios que abarca componentes como botones, tablas, marcos, etcétera. |
| 1.3. Contenedores (JFrame, JDialog, JPanel), átomicos (JLabel, JCheckBox, JComboBox, JButton, JRadioButton), de texto (JTextField, JTextArea, JTextPane) y de menu (JMenuBar, JMenu, JMenuItem). | 1.4. Antes de Swing, las interfaces gráficas con el usuario se realizaban a través de AWT. La clase Swing implementa un conjunto de componentes construidos sobre AWT y además proporciona mejoras en la visualización de los diferentes elementos. |



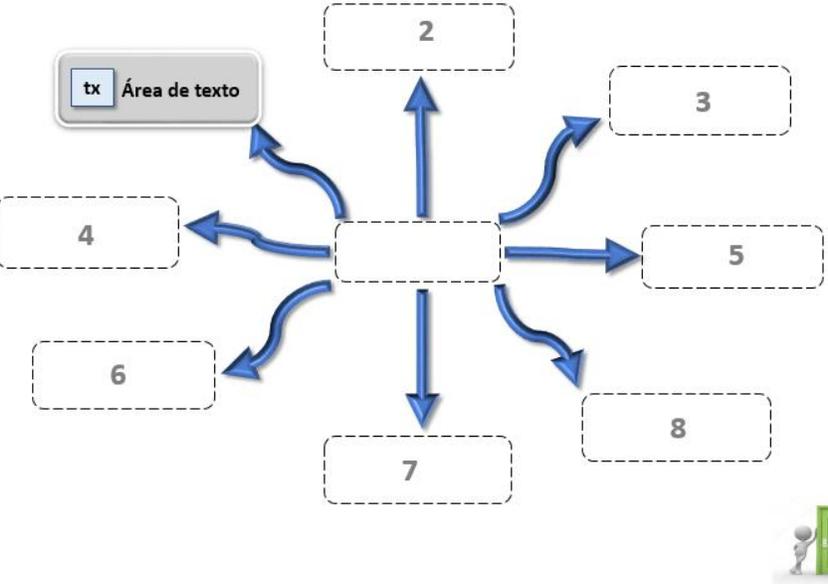
| Actividad 3 | | | | | | | | | |
|-------------|-------|-------|-------|-------|-------|-------|-------|-------|--------|
| 1.(d) | 2.(g) | 3.(j) | 4.(e) | 5.(c) | 6.(b) | 7.(i) | 8.(a) | 9.(h) | 10.(f) |

| Actividad 4 | | | | | | | | | |
|-------------|-------|-------|-------|-------|-------|-------|-------|-------|--------|
| 1.(b) | 2.(c) | 3.(a) | 4.(f) | 5.(i) | 6.(d) | 7.(g) | 8.(e) | 9.(h) | 10.(j) |

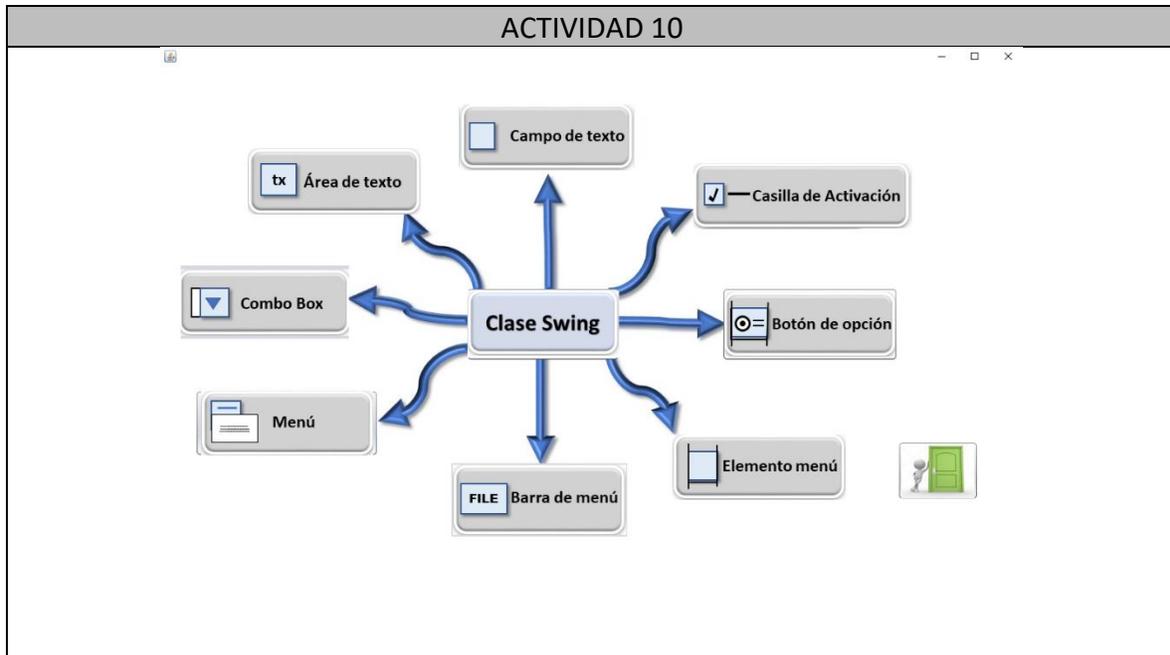
| ACTIVIDAD 8 | | | |
|---|--|---|---|
| Imágenes | | | |
|  area_texto.png |  area_texto2.png |  campo_texto.png |  campo_texto2.png |
|  casilla_activacion.png |  casilla_activacion2.png |  boton_opcion.png |  boton_opcion2.png |
|  combo_box.png |  combo_box2.png |  barra_menu.png |  barra_menu2.png |
|  menu.png |  Menu2.png |  elemento_menu.png |  elemento_menu2.png |
|  clase_swing.png |  clase_swing.png |  Salir.png |  Salir2.png |

ACTIVIDAD 9

8. Pantalla



9. Al pasar el mouse cambia el color y el tamaño.



| ACTIVIDAD 11 | |
|-----------------------------------|---|
| JTextField – Campo de Texto | Permite la introducción de texto simple. |
| JTextArea – Área de texto | Vincula un área de texto en la ventana donde se ingresa o se muestra información. |
| JMenuBar – Barra de menú | Se utiliza para incluir una barra de menú en la ventana. |
| JMenuItem – Elemento de Menú | Elemento que se encuentra en la barra de menú. |
| JComboBox – Combo Box | Muestra una lista de elementos como un combo de selección. |
| JCheckBox - Casilla de Activación | Se utiliza para elegir una o varias casillas para una selección múltiple. |
| JRadioButton –Botón de opción | Es un botón que nos permite una sola elección dentro de un grupo. |

| ACTIVIDAD 12 | |
|---------------------|--|
| Pantalla. | |

ACTIVIDAD 12

Area de Texto

- Area de texto

Es el componente gráfico para colocar texto. Sirve para colocar texto a cerca de un tema específico o para explicar alguna imagen de la que se esté haciendo referencia.
Podemos dar formato con las propiedades del componente para cualquier color y tipo y tamaño de letra. Solo que la propiedad se aplica a todo el texto, no a una porción de este. Se pueden modificar las propiedades

Clase Swing

ACTIVIDAD 13

Pantalla

Campo de Texto

- Campo de texto

Sirve para colocar espacios donde se puede ingresar datos dentro de los formularios.

Ejemplo: Sumar dos números

Número 1:

Número 2:

+

Resultado: 9.0

Clase Swing

ACTIVIDAD 14

7, 2, 8, 5, 3, 4, 6, 1

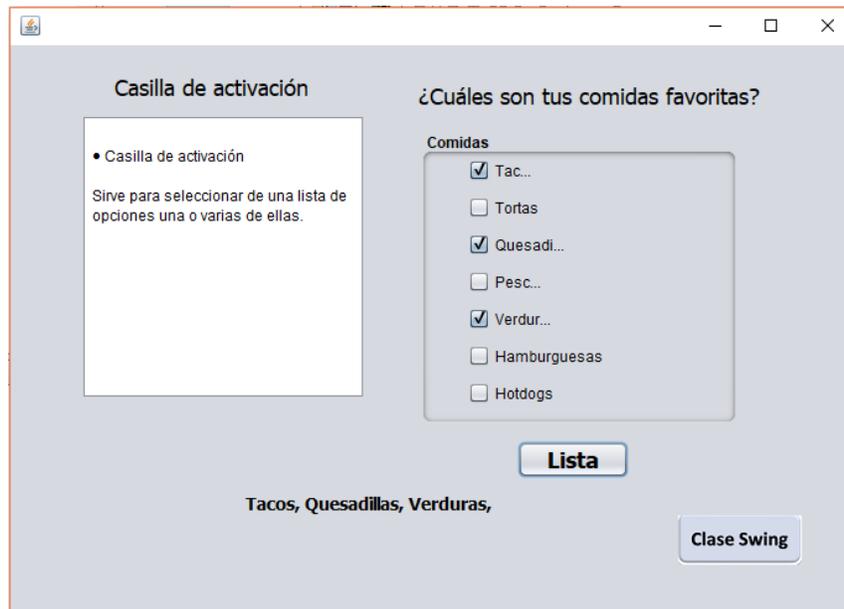
ACTIVIDAD 15

8. Código.

```

if (this.chkQuesadillas.isSelected()){
    lista = lista + this.chkQuesadillas.getText()+" ";
}
if (this.chkPescado.isSelected()){
    lista = lista + this.chkPescado.getText()+" ";
}
if (this.chkVerduras.isSelected()){
    lista = lista + this.chkVerduras.getText()+" ";
}
if (this.chkHamburguesas.isSelected()){
    lista = lista + this.chkHamburguesas.getText()+" ";
}
if (this.chkHotdogs.isSelected()){
    lista = lista + this.chkHotdogs.getText()+" ";
}
    
```

12. Pantalla



ACTIVIDAD 16

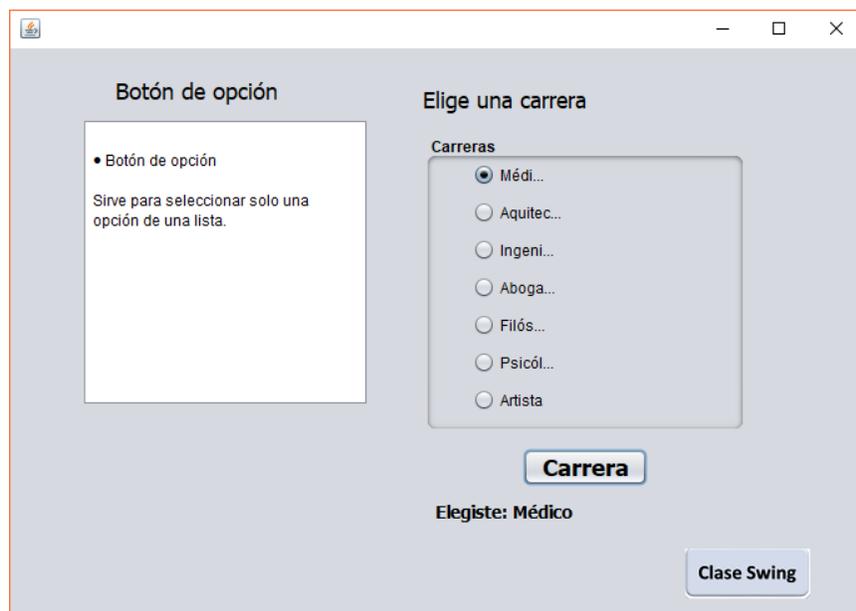
2, 1, 4, 3

ACTIVIDAD 17

3. Código del botón **Carrera**

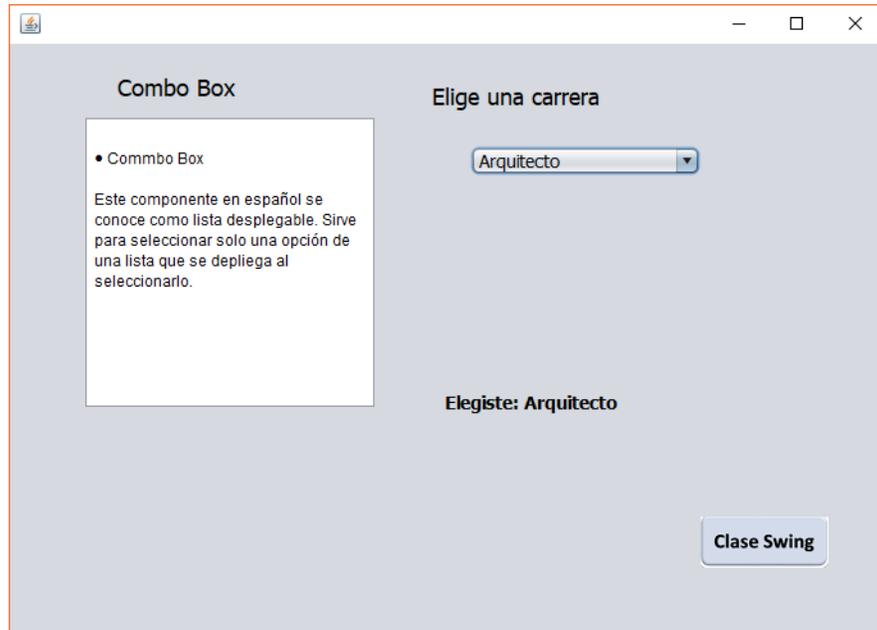
```
private void btnListaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String carrera = "";
    if (this.opcMedico.isSelected()){
        carrera = this.opcMedico.getText();
    }
    if (this.opcArquitecto.isSelected()){
        carrera = this.opcArquitecto.getText();
    }
    if (this.opcIngeniero.isSelected()){
        carrera = this.opcIngeniero.getText();
    }
    if (this.opcAbogado.isSelected()){
        carrera = this.opcAbogado.getText();
    }
    if (this.opcFilosofo.isSelected()){
        carrera = this.opcFilosofo.getText();
    }
    if (this.opcPsicologo.isSelected()){
        carrera = this.opcPsicologo.getText();
    }
    if (this.opcArtista.isSelected()){
        carrera = this.opcArtista.getText();
    }
    this.lblCarrera.setText("Elegiste: "+carrera);
}
}
```

7. Pantalla



ACTIVIDAD 18

Pantalla.

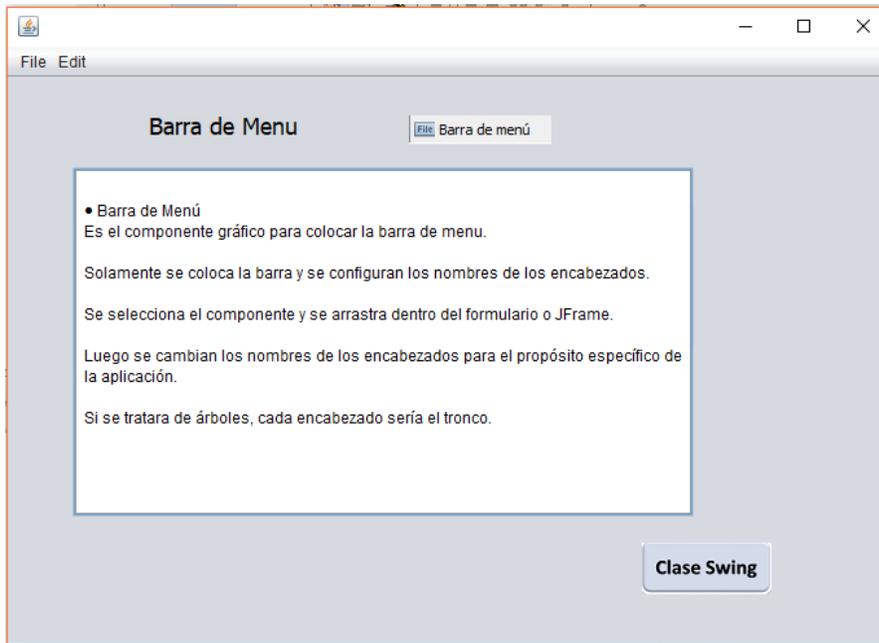


ACTIVIDAD 19

6, 1, 3, 2, 4, 5, 7

ACTIVIDAD 20

Pantalla.



ACTIVIDAD 21

Pantalla

File Instrumentos Musicales

- Cuerda ▶
- Percusión ▶
- Viento ▶

Menú

- Menú

Es el componente gráfico para colocar un menú dentro de los encabezados principales de la barra.

En este caso se está construyendo un menú para los distintos instrumentos musicales y cada menú se asociará a los tipos de instrumentos.

Clase Swing

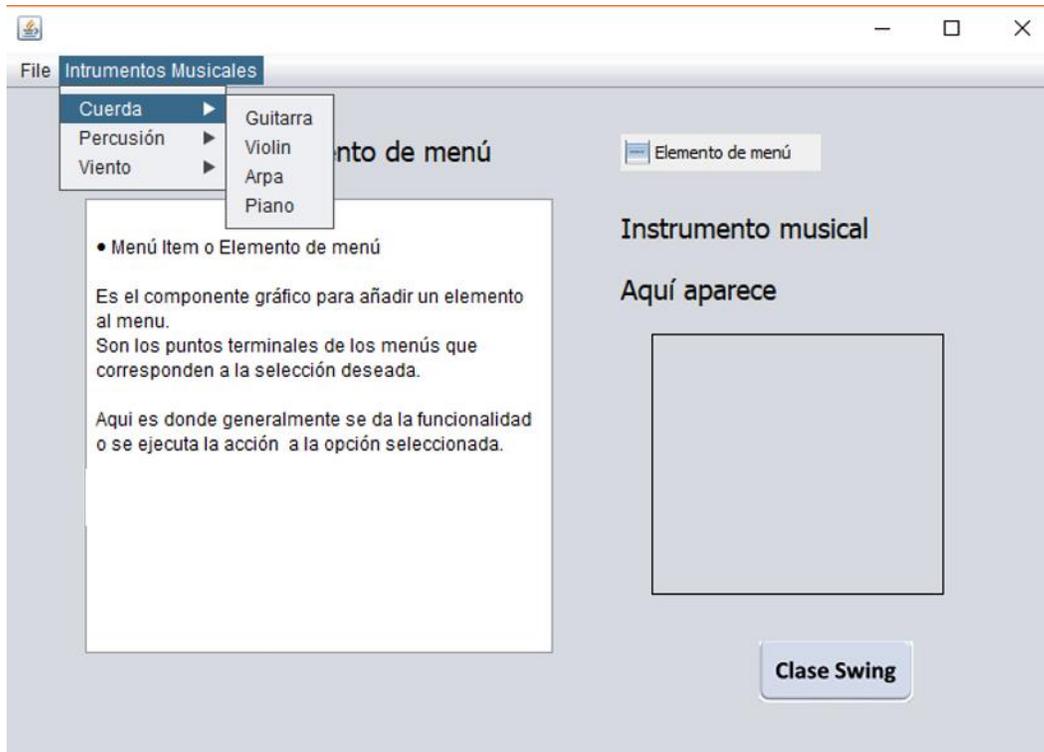
ACTIVIDAD 22

Imágenes.

| | | | | |
|---|---|---|--|---|
|  |  |  |  |  |
| arpa.png | bateria.png | clarinete.png | flauta.png | guitarra.png |
|  |  |  |  |  |
| guitarra2.png | marimba.png | piano.png | tambor.png | trompeta.png |
|  |  |  | | |
| oboe.png | pandero.png | violin.png | | |

ACTIVIDAD 23

Pantalla.



ACTIVIDAD 24

6. Código

```
private void menuArpaActionPerformed(java.awt.event.ActionEvent evt) {
    this.lblInstrumento.setText("Seleccionaste: " + this.menuArpa.getText());
    muestralmagen(this.menuArpa.getText());
    reproduceAudio(this.menuArpa.getText());
}

private void menuPianoActionPerformed(java.awt.event.ActionEvent evt) {
    this.lblInstrumento.setText("Seleccionaste: " + this.menuPiano.getText());
    muestralmagen(this.menuPiano.getText());
    reproduceAudio(this.menuPiano.getText());
}

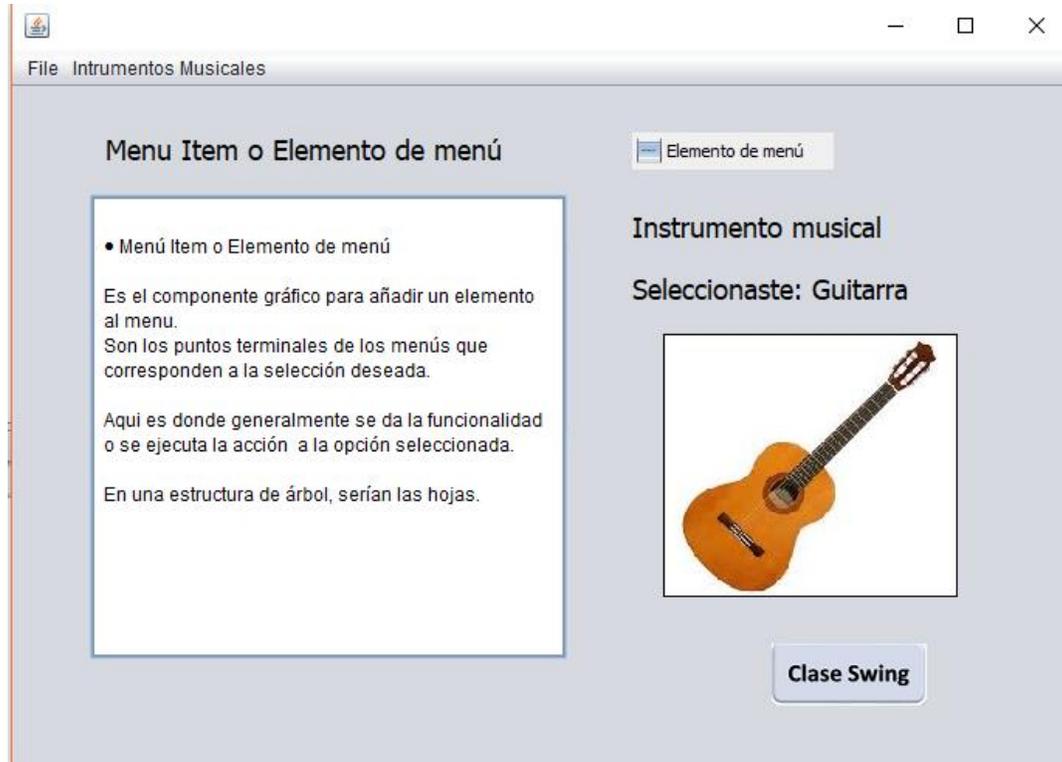
private void menuBateriaActionPerformed(java.awt.event.ActionEvent evt) {
    this.lblInstrumento.setText("Seleccionaste: " + this.menuBateria.getText());
    muestralmagen(this.menuBateria.getText());
    reproduceAudio(this.menuBateria.getText());
}
```

ACTIVIDAD 24

```
}  
  
private void menuMarimbaActionPerformed(java.awt.event.ActionEvent evt) {  
    this.lblInstrumento.setText("Seleccionaste: " + this.menuMarimba.getText());  
    muestralmagen(this.menuMarimba.getText());  
    reproduceAudio(this.menuMarimba.getText());  
}  
  
private void menuPanderoActionPerformed(java.awt.event.ActionEvent evt) {  
    this.lblInstrumento.setText("Seleccionaste: " + this.menuPandero.getText());  
    muestralmagen(this.menuPandero.getText());  
    reproduceAudio(this.menuPandero.getText());  
}  
  
private void menuMaracasActionPerformed(java.awt.event.ActionEvent evt) {  
    this.lblInstrumento.setText("Seleccionaste: " + this.menuMaracas.getText());  
    muestralmagen(this.menuMaracas.getText());  
    reproduceAudio(this.menuMaracas.getText());  
}  
  
private void menuTrompetaActionPerformed(java.awt.event.ActionEvent evt) {  
    this.lblInstrumento.setText("Seleccionaste: " + this.menuTrompeta.getText());  
    muestralmagen(this.menuTrompeta.getText());  
    reproduceAudio(this.menuTrompeta.getText());  
}  
  
private void menuFlautaActionPerformed(java.awt.event.ActionEvent evt) {  
    this.lblInstrumento.setText("Seleccionaste: " + this.menuFlauta.getText());  
    muestralmagen(this.menuFlauta.getText());  
    reproduceAudio(this.menuFlauta.getText());  
}  
  
private void menuOboeActionPerformed(java.awt.event.ActionEvent evt) {  
    this.lblInstrumento.setText("Seleccionaste: " + this.menuOboe.getText());  
    muestralmagen(this.menuOboe.getText());  
    reproduceAudio(this.menuOboe.getText());  
}  
  
private void menuClarineteActionPerformed(java.awt.event.ActionEvent evt) {  
    this.lblInstrumento.setText("Seleccionaste: " + this.menuClarinete.getText());  
    muestralmagen(this.menuClarinete.getText());  
    reproduceAudio(this.menuClarinete.getText());  
}
```

ACTIVIDAD 24

7. Pantalla



ACTIVIDAD 25

| Actividad 26 | | | | |
|--|---|--|------------------|-------------------|
| <p>1.1. Círculo, elipse, triángulo, línea, cuadrado, cuadrado esquinas redondeadas, rectángulo.</p> | <p>1.2. B(40,10), C(25,40), D(60,40)</p> | | | |
| <p>1.3</p> | | | | |
| Figura número | Nombre de la figura | Coordenadas de la esquina superior izquierda | Tamaño del ancho | Tamaño de lo alto |
| 1 | Círculo | (10,10) | 20 | 20 |
| 2 | Elipse | (40,10) | 40 | 10 |
| 5 | Cuadrado | (60,40) | 30 | 30 |
| 6 | Cuadrado esquinas redondeadas | (20,80) | 10 | 10 |
| 7 | Rectángulo | (40,80) | 50 | 10 |
| Tabla 4.2.1. | | | | |

| Figura número | Nombre de la figura | Coordenadas del punto inicial | Coordenadas del punto final |
|---------------|---------------------|-------------------------------|-----------------------------|
| 4 | Línea | (40,40) | (55,75) |

Tabla 4.2.2.

| Figura número | Nombre de la figura | Número de vértices | Coordenadas del vértice V1 | Coordenadas del vértice V2 | Coordenadas del vértice V3 |
|---------------|---------------------|--------------------|----------------------------|----------------------------|----------------------------|
| 3 | Triángulo | 3 | (25,40) | (40,70) | (10,70) |

Tabla 4.2.3.

Actividad 27

a)

| Núm. | Figura | Color | Método | x | y | Anchura | Altura |
|------|------------|-------------|----------|-----|-----|---------|--------|
| 1 | Círculo | amarillo | drawOval | 40 | 40 | 80 | 80 |
| 2 | Óvalo | azul | drawOval | 160 | 40 | 160 | 40 |
| 3 | Óvalo | azul | drawOval | 200 | 60 | 160 | 40 |
| 4 | Rectángulo | azul | drawRect | 40 | 280 | 120 | 120 |
| 5 | Rectángulo | Gris oscuro | drawRect | 80 | 320 | 40 | 80 |
| 6 | Rectángulo | morado | drawRect | 160 | 280 | 200 | 120 |

Tabla 4.2.6.

| Núm. | Figura | Color | Método | x | y | Anchura | Altura | Arco Anchura | Arco Altura |
|------|-------------------------------------|---------|---------------|-----|-----|---------|--------|--------------|-------------|
| 7 | Rectángulo con esquinas redondeadas | naranja | drawRoundRect | 200 | 320 | 40 | 40 | 5 | 5 |
| 8 | Rectángulo con esquinas redondeadas | naranja | drawRoundRect | 280 | 320 | 40 | 40 | 5 | 5 |

Tabla 4.2.7.

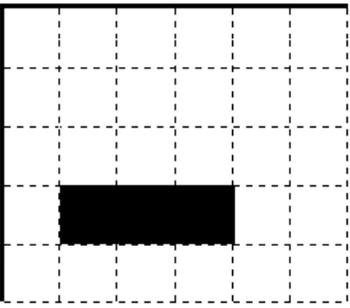
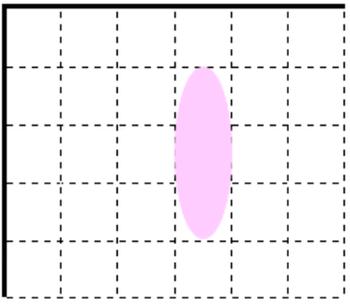
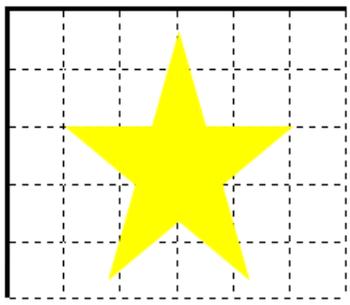
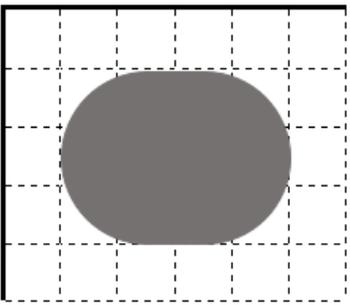
| Núm. | Figura | Color | Método | x1 | y1 | x2 | y2 |
|------|--------|-------|----------|-----|-----|-----|-----|
| 9 | Línea | rojo | drawLine | 100 | 160 | 300 | 160 |
| 10 | Línea | rojo | drawLine | 120 | 200 | 320 | 200 |
| 11 | Línea | rojo | drawLine | 140 | 240 | 340 | 240 |
| 12 | Línea | rojo | drawLine | 160 | 160 | 220 | 280 |
| 13 | Línea | rojo | drawLine | 240 | 160 | 300 | 280 |
| 14 | Línea | rojo | drawLine | 300 | 160 | 360 | 280 |

Tabla 4.2.8.

| Núm. | Figura | Color | Método | Vértice 1 | Vértice 2 | Vértice 3 | Puntos |
|------|-----------|-------|-------------|------------------|---------------|----------------|--------|
| 15 | Triángulo | verde | drawPolygon | x1=100 y1=160 | x=40 y=280 | x=160 y=280 | 3 |

Tabla 4.2.9.

```
public void paint(Graphics casa)
{
    super.paint(casa);
    casa.setColor(Color.yellow);
    casa.drawOval(40, 40, 80, 80);
    casa.setColor(Color.blue);
    casa.drawOval(160, 40, 160, 40);
    casa.drawOval(200,60,160,40);
    casa.drawRect(40,280,120,120);
    casa.setColor(Color.magenta);
    casa.drawRect(80,320,40,80);
    casa.setColor(Color.darkGray);
    casa.drawRect(160,280,200,120);
    casa.setColor(Color.orange);
    casa.drawRoundRect(200,320,40,40,5,5);
    casa.drawRoundRect(280,320,40,40,5,5);
    casa.setColor(Color.red);
    casa.drawLine(100,160,300,160);
    casa.drawLine(120,200,320,200);
    casa.drawLine(140,240,340,240);
    casa.drawLine(160,160,220,280);
    casa.drawLine(240,160,300,280);
    casa.drawLine(300,160,360,280);
    casa.setColor(Color.green);
    int[] x={100,40,160};
    int[] y={160,280,280};
    casa.drawPolygon(x,y,3);
}
```

| Código | Figura |
|--|--|
| <pre>setColor(Color.black); fillRect(10,30,30,10);</pre> |  |
| <pre>setColor(Color.pink); fillOval(30,10,10,30);</pre> |  |
| <pre>int[] x={30,33,50,38,42,30,18,22,10,17}; int[] y={5,20,20,30,45,38,45,30,20,20}; setColor(Color.yellow); fillPolygon(x,y,10);</pre> |  |
| <pre>setColor(Color.gray); fillRoundRect(10,10,40,30,20,15);</pre> |  |

```

public void paint(Graphics casa)
{
    super.paint(casa);
    casa.setColor(Color.yellow);
    casa.fillOval(40, 40, 80, 80);
    casa.setColor(Color.blue);
    casa.fillOval(160, 40, 160, 40);
    casa.fillOval(200,60,160,40);
    casa.fillRect(40,280,120,120);
    casa.setColor(Color.darkGray);
    casa.fillRect(80,320,40,80);
    casa.setColor(Color.magenta);
    casa.fillRect(160,280,200,120);
    casa.setColor(Color.orange);
    casa.fillRoundRect(200,320,40,40,5,5);
    casa.fillRoundRect(280,320,40,40,5,5);
    casa.setColor(Color.pink);
    int[] x1={100,300,360,160};
    int[] y1={160,160,280,280};
    casa.fillPolygon(x1,y1,4);
    casa.setColor(Color.red);
    casa.drawLine(100,160,300,160);
    casa.drawLine(120,200,320,200);
    casa.drawLine(140,240,340,240);
    casa.drawLine(160,160,220,280);
    casa.drawLine(240,160,300,280);
    casa.drawLine(300,160,360,280);
    casa.setColor(Color.green);
    int[] x={100,40,160};
    int[] y={160,280,280};
    casa.fillPolygon(x,y,3);
}

```

Desafíos

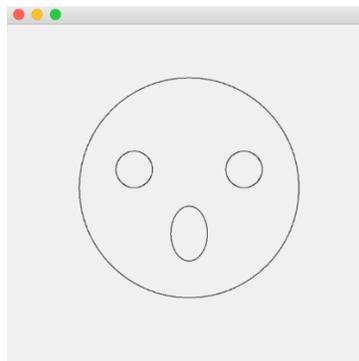
| Desafío 1 | |
|---|--|
| Al pulsar el botón Saludar aparece el mensaje Hola José. | //Código para la acción del botón Saludar. private void saludarActionPerformed(java.awt.event.ActionEvent evt) { this.jLabel1.setText("Hola José"); } |

| | |
|--|--|
| Al pulsar el botón Limpiar se borra el mensaje. | //Código para la acción del botón Limpiar private void limpiarActionPerformed(java.awt.event.ActionEvent evt) { this.jLabel1.setText(""); } |
| Al pulsar el botón Cambiar el fondo pasa a color rosa. | //Código para la acción del botón Cambiar private void cambiarActionPerformed(java.awt.event.ActionEvent evt) { this.getContentPane().setBackground(Color.pink); this.jLabel1.setBackground(Color.red); } |
| Al pulsar el botón Restablecer aparece el mensaje y el color iniciales. | //Código para la acción del botón Restablecer private void restablecerActionPerformed(java.awt.event.ActionEvent evt) { this.jLabel1.setText("Hola ..."); this.getContentPane().setBackground(Color.cyan); } |
| Al pulsar el botón Salir se cierra la ventana. | //Código para la acción del botón Salir private void salirActionPerformed(java.awt.event.ActionEvent evt) { System.exit(0); } |

Desafío 2

1) Dibujo de la cara.

```
public void paint (Graphics cara){
    super.paint(cara);
    //Dibuja un círculo con origen en x=80, y=80; ancho=240 y largo=240
    cara.drawOval(80, 80, 240, 240);
    //Dibuja un círculo con origen en x=120, y=160; ancho=40 y largo=40
    cara.drawOval(120, 160, 40, 40);
    //Dibuja un círculo con origen en x=240, y=160; ancho=40 y largo=40
    cara.drawOval(240, 160, 40, 40);
    //Dibuja un círculo con origen en x=180, y=220; ancho=40 y largo=60
    cara.drawOval(180, 220, 40, 60);
}
```



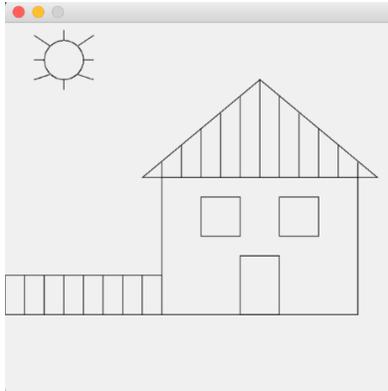
2) Dibujo de la casa.

```
public void paint (Graphics dibujo){
    super.paint(dibujo);
    //Dibujo del sol
    dibujo.drawOval(40, 40, 40, 40);
    //Dibujo de 8 rayos de sol
    dibujo.drawLine(60, 40, 60, 30);
    dibujo.drawLine(80, 60, 90, 60);
    dibujo.drawLine(60, 80, 60, 90);
    dibujo.drawLine(40, 60, 30, 60);
    dibujo.drawLine(75, 45, 90, 35);
    dibujo.drawLine(75, 75, 90, 80);
    dibujo.drawLine(45, 75, 30, 80);
    dibujo.drawLine(45, 45, 30, 35);
    //Dibujo de la casa
    dibujo.drawRect(160,180,200,140);
    //Dibujo de la primera ventana
    dibujo.drawRect(200,200,40,40);
    //Dibujo de la segunda ventana
    dibujo.drawRect(280,200,40,40);
    //Dibujo de la puerta
    dibujo.drawRect(240,260,40,60);
    //Dibujo de la cerca
    dibujo.drawRect(0,280,160,40);
    //Dibujo de las líneas de la cerca
    dibujo.drawLine(20,280,20,320);
    dibujo.drawLine(40,280,40,320);
    dibujo.drawLine(60,280,60,320);
    dibujo.drawLine(80,280,80,320);
    dibujo.drawLine(100,280,100,320);
    dibujo.drawLine(120,280,120,320);
    dibujo.drawLine(140,280,140,320);
    //Dibujo del techo
    int x[ ]={260,140,380};
    int y[ ]={80,180,180};
    dibujo.drawPolygon(x,y,3);
    //Dibujo de las líneas del techo
    dibujo.drawLine(160,180,160,165);
    dibujo.drawLine(180,180,180,147);
    dibujo.drawLine(200,180,200,130);
    dibujo.drawLine(220,180,220,115);
    dibujo.drawLine(240,180,240,97);
```

```

dibujو.drawLine(260,180,260,80);
dibujو.drawLine(280,180,280,97);
dibujو.drawLine(300,180,300,115);
dibujو.drawLine(320,180,320,130);
dibujو.drawLine(340,180,340,147);
dibujو.drawLine(360,180,360,165);
}

```



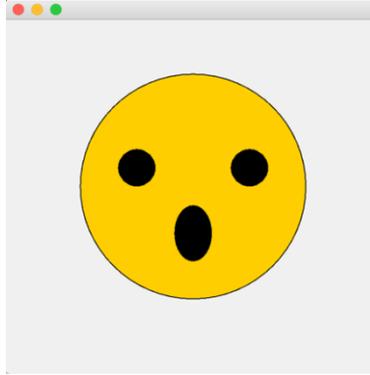
Desafío 3

1) Dibujo de la cara.

```

public void paint (Graphics cara){
    super.paint(cara);
    //Colorear la imagen
    cara.setColor(Color.orange);
    cara.fillOval(80, 80, 240, 240);
    //Dibujo del contorno
    cara.setColor(Color.black);
    cara.drawOval(80, 80, 240, 240);
    //Colorear la boca y los ojos
    cara.setColor(Color.black);
    cara.fillOval(120, 160, 40, 40);
    cara.fillOval(240, 160, 40, 40);
    cara.fillOval(180, 220, 40, 60);
}

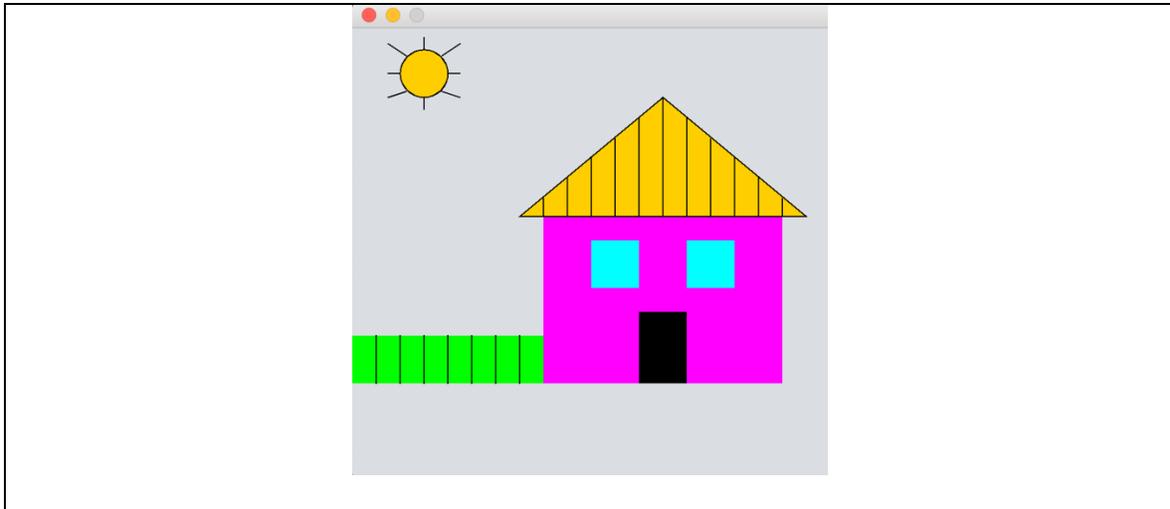
```



2) Dibujo de la casa.

```
public void paint (Graphics dibujo){
    super.paint(dibujo);
    //Colorear el sol
    dibujo.setColor(Color.orange);
    dibujo.fillOval(40, 40, 40, 40);
    //Dibujo del contorno del sol
    dibujo.setColor(Color.black);
    dibujo.drawOval(40, 40, 40, 40);
    //Dibujo de 8 rayos de sol
    dibujo.setColor(Color.black);
    dibujo.drawLine(60, 40, 60, 30);
    dibujo.drawLine(80, 60, 90, 60);
    dibujo.drawLine(60, 80, 60, 90);
    dibujo.drawLine(40, 60, 30, 60);
    dibujo.drawLine(75, 45, 90, 35);
    dibujo.drawLine(75, 75, 90, 80);
    dibujo.drawLine(45, 75, 30, 80);
    dibujo.drawLine(45, 45, 30, 35);
    //Colorear de la casa
    dibujo.setColor(Color.magenta);
    dibujo.fillRect(160,180,200,140);
    //Colorear la primera ventana
    dibujo.setColor(Color.cyan);
    dibujo.fillRect(200,200,40,40);
    //Colorear la segunda ventana
    dibujo.fillRect(280,200,40,40);
    //Colorear la puerta
    dibujo.setColor(Color.black);
    dibujo.fillRect(240,260,40,60);
    //Colorear la cerca
    dibujo.setColor(Color.green);
```

```
dibujو.fillRect(0,280,160,40);
//Dibujو de las líneas de la cerca
dibujو.setColor(Color.black);
dibujو.drawLine(20,280,20,320);
dibujو.drawLine(40,280,40,320);
dibujو.drawLine(60,280,60,320);
dibujو.drawLine(80,280,80,320);
dibujو.drawLine(100,280,100,320);
dibujو.drawLine(120,280,120,320);
dibujو.drawLine(140,280,140,320);
//Colorear el techo
dibujو.setColor(Color.orange);
int x[]={260,140,380};
int y[]={80,180,180};
dibujو.fillPolygon(x,y,3);
//Dibujو del contorno del techo
dibujو.setColor(Color.black);
int x1[]={260,140,380};
int y1[]={80,180,180};
dibujو.drawPolygon(x1,y1,3);
//Dibujو de las líneas del techo
dibujو.setColor(Color.black);
dibujو.drawLine(160,180,160,165);
dibujو.drawLine(180,180,180,147);
dibujو.drawLine(200,180,200,130);
dibujو.drawLine(220,180,220,115);
dibujو.drawLine(240,180,240,97);
dibujو.drawLine(260,180,260,80);
dibujو.drawLine(280,180,280,97);
dibujو.drawLine(300,180,300,115);
dibujو.drawLine(320,180,320,130);
dibujو.drawLine(340,180,340,147);
dibujو.drawLine(360,180,360,165);
}
```



4.4 Referencias

- Ávila, S. et. al. (2019). *Paquete didáctico para la asignatura de Cibernética y Computación II*. México: ENCCH Plantel Oriente, UNAM.
- Dean, J. (2009). *Introducción a la programación con Java*. Mc. Graw-Hill. México.
- Doherty, D. et. al.(2000). *Aprendiendo Borland JBuilder 3 en 21 días*. México. Prentice Hall Hispanoamericana.
- Zárate, U. (s.f.). *Programación con Java*. [en línea] Disponible en: <http://profesores.fi-b.unam.mx/carlos/java/> [Consultado el 1 de marzo de 2019].