Ejercicio de introducción a la programación orientada a objetos

Ejecuta la aplicación, se debería mostrar una ventana como la siguiente

Tarea 1) Hacer que cuando se pulse con un botón del ratón, se dibuje un rectángulo. Abre el archivo "Painter.java". El método click() se ejecutará cada vez que se pulse un botón del ratón, siendo x e y las coordenadas en la que se ha hecho pulsado el botón del ratón.

```
public void click(Graphics2D graphics, int width, int height, int button, int x, int y ) {
    graphics.setColor(Color.BLUE);
    graphics.fillRect(x, y, width: 30, height: 30);
}
```

Ejecuta la aplicación nuevamente, y haz click sobre la ventana con los diferentes botones del ratón.

Tarea 2) Ahora haremos que cuando se pulse el botón izquierdo se dibuje un rectángulo y cuando se pulse el botón derecho, se dibuje un circulo de color rojo.

```
public void click(Graphics2D graphics, int width, int height, int button, int x, int y ) {
    if (button == 1) {
        graphics.setColor(Color.BLUE);
        graphics.fillRect(x, y, width: 30, height: 30);
    } else if (button == 3) {
        graphics.setColor(Color.RED);
        graphics.fillOval(x, y, width: 30, height: 30);
    }
}
```

Tarea 3) Prueba a redimensionar la ventana de tu aplicación. Efectivamente, se borra todo lo dibujado hasta ahora. Esto ocurre ya que se necesita redibujar toda la ventana, junto con los rectángulos y circulos que se hayan posicionado.

Esto nos obliga a guardar los rectángulos y círculos que se vayan dibujando para poder redibujarlos cuando se requiera. Según lo que sabemos programar, podemos plantearnos tener tres ArrayList:

- listX almacenará la coordenada X de la figura.
- listY almacenará la coordenada Y de la figura.
- listTipo almacenará si la figura es un rectángulo (0) o un circulo (1)

Estas variables la debemos declarar fuera del método “click”, ya que se eliminan cada vez que se finaliza de ejecutar la función. Por eso, las pondremos a nivel del siguiente bloque “clase Painter”.

```
public class Painter {
    private ArrayList<Integer> listX = new ArrayList<>();
    private ArrayList<Integer> listY = new ArrayList<>();
    private ArrayList<Integer> listTipo = new ArrayList<>();
```

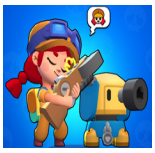
Y guardamos la coordenadas x e y, además del tipo de figura en el método “click”

```
public void click(Graphics2D graphics, int width, int height, int button, int x, int y ) {
    if (button == 1) {
        graphics.setColor(Color.BLUE);
        graphics.fillRect(x, y, width: 30, height: 30);
        listX.add(x);
        listY.add(y);
        listTipo.add(0);
    } else if (button == 3) {
        graphics.setColor(Color.RED);
        graphics.fillOval(x, y, width: 30, height: 30);
        listX.add(x);
        listY.add(y);
        listTipo.add(1);
    }
}
```

Y para que se repinte todas las figuras, lo hacemos en el método “repaint”.

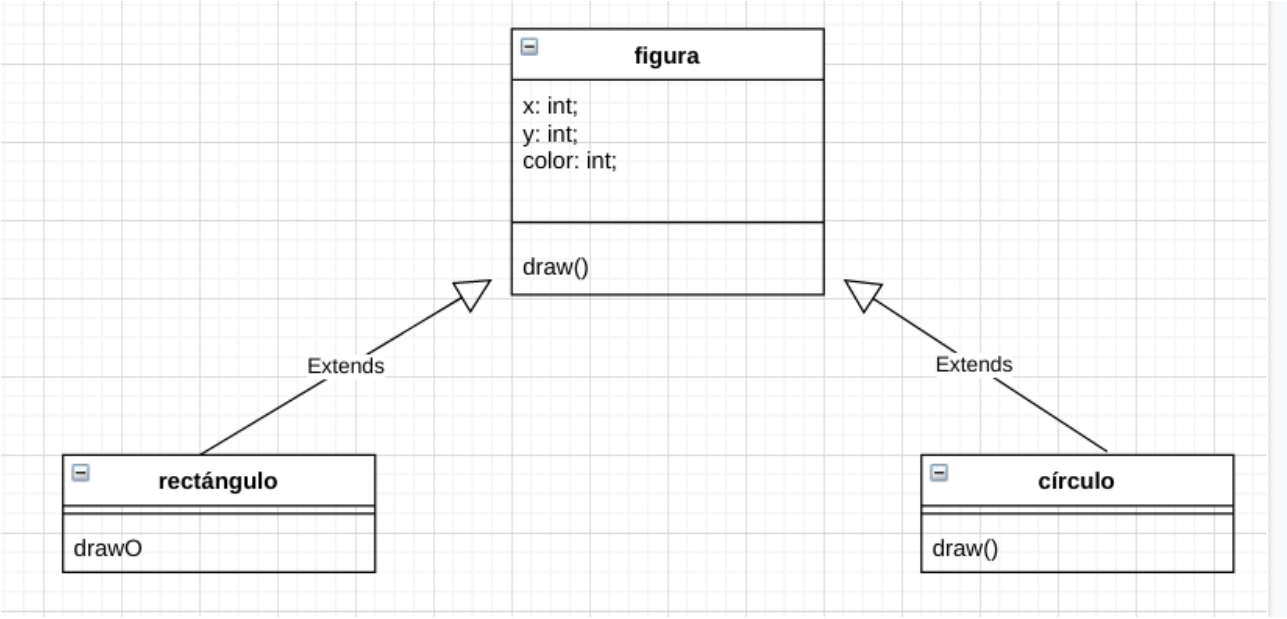
```
public void repaint(Graphics2D graphics, int width, int height) {
    for (int i = 0; i < listX.size(); i++) {
        int x = listX.get(i);
        int y = listY.get(i);
        int tipo = listTipo.get(i);

        if (tipo == 0) {
            graphics.setColor(Color.BLUE);
            graphics.fillRect(x, y, width: 30, height: 30);
        } else if (tipo == 1) {
            graphics.setColor(Color.RED);
            graphics.fillOval(x, y, width: 30, height: 30);
        }
    }
}
```

Prueba ahora a dibujar varios rectángulos y círculos y a redimensionar la ventana de tu aplicación. Se debería ver que ahora no desaparecen las figuras.



Tarea 4) Introducción a un planteamiento orientado a objetos - Esta forma de programar tiene muchos problemas que hacen que la aplicación sea poco legible y sobre todo, que sea más difícil ampliarla (por ejemplo, añadiendo nuevas figuras). Y es aquí en donde empezamos a usar la programación orientada a objetos POO

En POO lo primero es identificar los elementos de tu dominio. En nuestro caso, identificamos que tenemos dos tipos de objetos: rectángulos y círculos. Además, podemos pensar que estas no son más que tipos de un objeto más genérico “figura”.



Asociado a los objetos identificamos propiedades que tienen y métodos. Las propiedades en nuestro caso, podría ser las coordenadas x e y, el color, tamaño, etc. Los métodos indican funcionalidades, podría ser un posible método dibujar “draw”.

El método “dibujar” lo debería tener todas las figuras pero su implementación depende de la figura concreta. No es el mismo código para dibujar un rectángulo que para dibujar un círculo.

La flecha “extends” significa que es una especialización. “rectángulo” es una especialización de la clase figura. También se puede decir, que “figura” es una generalización de los rectángulos y círculos. Esto se implemente a través de la “herencia”.

Tarea 5) Creamos una clase figura llamada “Figure”. Cada clase se debe crear en un fichero java que tenga el mismo nombre de la clase.

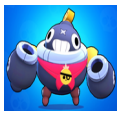


Figure.java

```java
package org.iesanaluisabenitez.informatica;

import java.awt.*;

public class Figure {
    int x;
    int y;

    void draw(Graphics2D graphics) {

    }
}
```

Tarea 6) Creamos la clase Rectangle

Rectangle.java

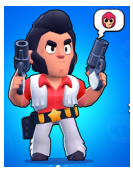
```java
package org.iesanaluisabenitez.informatica;

import java.awt.*;

public class Rectangle extends Figure {
    @Override
    void draw(Graphics2D graphics) {
        graphics.setColor(Color.BLUE);
        graphics.fillRect(x, y, width: 30, height: 30);
    }
}
```

Observa que Rectangle al heredar de Figure (extends) posee todas las propiedades y métodos que tiene la clase Figure. Por eso, no hay que volver a declarar las propiedades x e y.

Tarea 7) Creamos la clase Circle

Circle.java

```java
package org.iesanaluisabenitez.informatica;

import java.awt.*;

public class Circle extends Figure{
    void draw(Graphics2D graphics) {
        graphics.setColor(Color.RED);
        graphics.fillOval(x, y, width: 30, height: 30);
    }
}
```



Tarea 8) Hacemos uso de las nuevas clases definidas en Painter.

a) Creamos la variable (propiedad) para almacenar las figuras.

```java
public class Painter {
    private ArrayList<Figure> figures = new ArrayList<>();
```

b) Creamos las figuras, las dibujamos y las pintamos. Observa que para acceder a las propiedades y métodos de un objeto se pone el nombre de la variable que almacena el objeto, un punto y a continuación el nombre de la propiedad o método.

Painter.java

```java
     */
    public void click(Graphics2D graphics, int width, int height, int button, int x, int y ) {
        if (button == 1) {
            Rectangle rectangle = new Rectangle();
            rectangle.x = x;
            rectangle.y = y;

            rectangle.draw(graphics);
            figures.add(rectangle);
        } else if (button == 3) {
            Circle circle = new Circle();
            circle.x = x;
            circle.y = y;

            circle.draw(graphics);
            figures.add(circle);
        }
    }
```

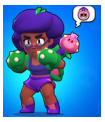
Y para repintarlas, el método “repaint” de “Painter”

```java
    public void repaint(Graphics2D graphics, int width, int height) {
        for (Figure figure: figures) {
            figure.draw(graphics);
        }
    }
```

Prueba a usar nuevamente la aplicación. Debería funcionar igual que antes ¿Pero que te parece este forma de programar orientada a objetos?

¿Te das cuenta que creamos un “objeto” rectángulo o circulo cada vez en la lista? Los objetos son instancias de las clases (Rectangle o Circle).

Tarea 9) Añadimos la posibilidad de dibujar triángulos si se pulsa el botón central del ratón.



Creamos una nueva clase “Triangle”

Triangle.java

```java
package org.iesanaluisabenitez.informatica;

import java.awt.*;

public class Triangle extends Figure{
    void draw(Graphics2D graphics) {
        graphics.setColor(Color.GREEN);

        graphics.fillPolygon(new int[] {x, x-15, x+15}, new int[] {y, y+30, y+30}, nPoints: 3);
    }
}
```

En click añadimos la posibilidad de crear triángulos si pulsa el botón central del ratón (2)

```java
public void click(Graphics2D graphics, int width, int height, int button, int x, int y ) {
    if (button == 1) {
        Rectangle rectangle = new Rectangle();
        rectangle.x = x;
        rectangle.y = y;

        rectangle.draw(graphics);
        figures.add(rectangle);
    } else if (button == 3) {
        Circle circle = new Circle();
        circle.x = x;
        circle.y = y;

        circle.draw(graphics);
        figures.add(circle);
    } else if (button == 2) {
        Triangle triangle = new Triangle();
        triangle.x = x;
        triangle.y = y;

        triangle.draw(graphics);
        figures.add(triangle);
    }
}
```

Espera, seguro que podemos plantear el mismo código de una forma más abreviada aprovechando que los rectángulos y círculos son al fin y al cabo figuras.

```java
public void click(Graphics2D graphics, int width, int height, int button, int x, int y ) {
    Figure newFigure;
    if (button == 1) {
        newFigure = new Rectangle();
    } else if (button == 2) {
        newFigure = new Triangle();
    } else if (button == 3) {
        newFigure = new Circle();
    } else {
        return;
    }

    newFigure.x = x;
    newFigure.y = y;
    newFigure.draw(graphics);
    figures.add(newFigure);
}
```

Tarea 10) Nos hemos venido arriba, ahora queremos dar algo de movimiento a las figuras que son circulo.

Circle.java

```java
package org.iesanaluisabenitez.informatica;

import java.awt.*;

public class Circle extends Figure{
    void draw(Graphics2D graphics) {
        graphics.setColor(Color.RED);
        graphics.fillOval(x, y, width: 30, height: 30);
        y++;
    }
}
```

¿No ha cambiado nada en tu programa? Dibuja varios rectángulos, triángulos y círculos. Pulsa el botón “play” ¿Qué ocurre?