UCSM Esc. Prof. de Ingeniería de Sistemas

Marzo - 2025

INFORME DE PRÁCTICAS

ESTRUCTURA DE DATOS Y ALGORITMIA

Práctica N° 01: POO - UTILIZANDO CLASES Y ARREGLOS DE OBJETOS

Elaborado por:

BEJARANO RIVERA MIGUEL
CUSIRRAMOS CHIRI SANTIAGO JESUS
YTO QUISPE JOSE CARLOS

GRUPO N° 04

POO
UTILIZANDO CLASES Y ARREGLOS DE OBJETOS

Presentado por:

71376633	Bejarano Rivera, Miguel	100%
60783993	Cusirramos Chiri, Santiago Jesus	100%
75145575	Yto Quispe, Jose Carlos	100%

RECONOCIMIENTOS

El autor de este trabajo reconoce con gratitud a los creadores de los lenguajes JAVA y otras personalidades y autores de libros de programación Bjarne Stroustrup, Dennis Ritchie, Herb Sutter, Herb Sutter, James Gosling, James Gosling, Brian Kernighan, Brian Kernighan, Ken Thompson.

PALABRAS CLAVES

Renderizado

ÍNDICE

1. RESUMEN

El desarrollo de esta práctica nos conduce por temas como la programación orientada a objetos, la definición de clases, métodos de instancia y de clase, arreglos y el uso de los mismos, así como de la palabra clave “this” en el contexto de la programación en java. Además, contiene un proyecto explicado paso a paso acerca de la generación de un programa el cual nos permite determinas diversas características de dos rectángulos generados por el usuario y trabajados por el programa creado por nosotros.

2. INTRODUCCIÓN

En esta práctica exploraremos los principios fundamentales de la Programación Orientada a Objetos (POO), un paradigma que aborda la abstracción, encapsulamiento, herencia y polimorfismo para mejorar la modularidad y reutilización del código. Se analizará la definición y manipulación de clases, la implementación de métodos de clase e instancia, y el uso del puntero implícito **THIS** para la referencia a miembros de un objeto. Además, se trabajará con arreglos de objetos, optimizando la gestión de múltiples instancias dentro de estructuras dinámicas o estáticas. A través de estos conceptos, se fortalecerá la comprensión de los principios de diseño orientado a objetos, promoviendo la construcción de software robusto y escalable. A si mismo, se recalcara la importancia de los diagramas **UML** y el como debemos de considerar el **SDLC** para el desarrollo de nuestros proyectos.

3. MARCO TEÓRICO

3.1 Definición y uso de clases.

La Programacion Orientada a Objetos es un enfoque de programación que se basa en la creación y manipulación de “objetos”, los cuales contienen atributos que representan sus características o propiedades, y métodos que definen las acciones que pueden realizar. Las clases se utilizan para definir la estructura y el comportamiento de los objetos, esta misma actuando como una plantilla o un modelo a partir del cual se crean objetos individuales con características únicas. (Blasco, 2023)

En POO se organiza el codigo de manera modular al optimizar comportamientos, combinación de datos, facilitando de esto manera la creación, mantenimiento y reutilización de codigo en le desarrollo de Software.

3.2 Definición de métodos de clase y de instancia.

Dentro de las clases, las cuales en principio actúan como la plantilla para que nuestros diversos objetos puedan actuar dentro del programa, encontramos los métodos de clases y de instancia. Los cuales pueden ser invocados de diferentes maneras. (Blasco, 2023)

3.2.1 Métodos de instancia

Un método de instancia es aquel que pertenece a una instancia específica(objeto) de una clase y opera sobre los atributos de dicha instancia. Para poder invocar estos métodos, además de crear una clase base, necesitamos generar una instancia de esa clase.

3.2.2 Métodos de clase

Por otra parte, tenemos a los métodos de clase, los cuales pertenecen a la misma clase, sin la necesidad de funcionar con una instancia especifica. Para hacer el uso de estos empleamos la palabra clase **STATIC.**

3.3 Uso de arreglos de objetos.

Para el uso de arreglos de objetos, consideramos esto como una estructura de datos importante, ya que permite manipular, controlar y almacenar una gran cantidad de datos complejos dentro del codigo. Este al ser un arreglo de objetos, significa que al ingresara a estos objetos, podremos emplear sus métodos de clase o instancia

3.4 Importancia de UML en el proceso de desarrollo.

Los diagramas de clases UML sin herramientas practicas de modelado para construir una arquitectura de software. Junto a los diagramas UML, los desarrolladores y las partes interesadas

pueden visualizar diferentes vistas de un sistema. Esto con el fin de ayudarnos a comunicar y a entender el funcionamiento del sistema, sus comportamientos y como se relacionan sus diferentes partes.

Las interacciones en un diagrama de clases UML muestran las relaciones entre las clases utilizando líneas y puntas de flecha. Las puntas de flecha tienen significados específicos para diferenciar entre los tipos de relaciones. Estas relaciones incluyen herencia, así como relaciones bidireccionales y unidireccionales. Las clases pueden agruparse en paquetes de clases que están estrechamente relacionadas. (*Diagrama de Clases: Qué Es, Cómo Hacerlo y Ejemplos | Miro*, s. f.)

Lo importante de estos diagramas es la capacidad que nos da de visualizar las clases de un sistema y sus propiedades. Además, de mostrar y analizar las relaciones entre las clases.

3.5 SDLC

El ciclo de vida del desarrollo de software (SDLC) es un proceso rentable y eficiente en términos de tiempo empleado por los equipos de desarrollo (para nuestro contexto, estudiantes de Ingeniería de Sistemas de la UCSM) para diseñar y crear software de alta calidad. El objetivo es minimizar los riesgos del proyecto por medio de una planificación anticipada que permita que el software cumpla las expectativas del cliente. Esta metodología establece una serie de pasos que dividen el proceso de desarrollo de software en tareas que se pueden asignar, completar y medir. (*¿Qué Es el SDLC? - Explicación del Ciclo de Vida del Desarrollo de Software - AWS*, s. f.)

Es importante considerar SDLC para el desarrollo de las guías practicas ya que este nos brindara una vista más profesional a comparación del modelo de trabajo que hemos estado llevando. EL desarrollo de software puede ser dificil de administrar debido a los requisitos, la colaboración interfuncional y los diferentes métodos que se pueden llegar a aplicar. (*¿Qué Es el SDLC? - Explicación del Ciclo de Vida del Desarrollo de Software - AWS*, s. f.)

Para esto el SDLC divide las tareas necesarias para la creación de software. El proceso de desarrollo pasa por varias etapas a medida que los desarrolladores agregan nuevas características, entre las que el equipo ha utilizado son:

3.5.1 Planificación

Esta fase incluyo la planificación del como el equipo iba a desarrollar las diferentes actividades solicitadas. El equipo por medio de reuniones y coordinación presencial en la universidad recopilo diferentes requisitos, así mismo en las reuniones se dieron diferentes ideas del como poder desarrollarlo.

3.5.2 Diseño

En la fase de diseño se tomo en cuenta los diferentes diagramas brindados en la guía práctica, el diagrama UML de clases sirvió/apoyo para agilizar la coordinación del trabajo.

3.5.3 Implementación - Desarrollo

En esta fase, el equipo se dedico a codificar el producto. Se analizaron los requisitos para identificar tareas de codificación más pequeñas que puedan hacerse sin necesidad de gastar tantos recursos.

3.5.4 Pruebas

El equipo realizo diferentes pruebas manuales para comprobar si el software presentaba errores, lastimosamente en los primeros productos si hubo algunos errores por falta de consideración de requisitos, sin embargo, en el segundo producto todo ha ido exitosamente, cumpliendo con las necesidades presentadas.

4. ACTIVIDADES

Realizar un programa que verifique si el rectanguilo esta sobre puesto, si esta junto o si estas dis junto. Considerar para esto arreglo de rectángulos, las distancias euclidianas y las áreas de cada rectángulo, como adicional integrar una función que retorne el área del nuevo rectángulo que se genera cuando los rectángulos iniciales están sobre puestos.

4.1 Coordenada

Integrar el nuevo UML → Tarea para santi o,o

```java
// Explciacion 4.1
public class Coordenada {
    private double x;
    private double y;

    public Coordenada() {}
    public Coordenada(double x, double y) {
        this.x = x;
        this.y = y;
    }
    public Coordenada(Coordenada c) {
        this.x = c.x;
        this.y = c.y;
    }

    public double getX() {
        return x;
    }
    public double getY() {
        return y;
    }

    public void setX(double x) {
        this.x = x;
    }
    public void setY(double y) {
        this.y = y;
    }

    @Override
    public String toString() {
        return  "(" + x + ";" + y +")";
    }
}
```

Nuestra primera integración para esta problemática es la siguiente clase:

COORDENADA: La cual se encarga de recibir los puntos cardinales de nuestros rectángulos.

Para la solución del problema es importante ya que esta nos brinda la opción de poder manejar cada punto cardinal de manera independiente, mencionar que esta con un constructor para **X** y para **Y**, cada una con sus respectivos getters y setters.

He de mencionar que se agrega un **toString** para la imprimir el objeto.

4.2 Rectángulo

En la clase rectángulo contamos con diferentes funciones, entre ellas el ordenarar, setters y getters, etc.

C**ONSTRUCTOR.**

```
public Rectangulo(Coordenada c1, Coordenada c2) {
     ordenar(c1,c2);
}
```

Para nuestro constructor del rectángulo, el usuario manda 2 grupos de coordenadas (x,y). el constructor no asigna directamente **c1** ni **c2**, lo que realiza es delegar esa función a **ordenar**, con el objetivo de que esta función pueda ordenar las variables/puntos cardinales.

ORDENAR.

```
public void ordenar(Coordenada c1, Coordenada c2) {
    double xMin = Math.min(c1.getX(), c2.getX());
    double yMin = Math.min(c1.getY(), c2.getY());
    double xMax = Math.max(c1.getX(), c2.getX());
    double yMax = Math.max(c1.getY(), c2.getY());

    this.esquina1 = new Coordenada(xMin, yMin);
    this.esquina2 = new Coordenada(xMax, yMax);
}
```

El método presente, garantiza que los puntos cardinales se almacenen de manera correcta. Es decir, se asegura de que la esquina uno sea la inferior izquierda y la esquina 2 la superior derecha.

Esto lo realizamos con las funciones de Math, con su mínimo y su máximo.

AREA

```
public double area() {
    return Math.abs((esquina2.getX() - esquina1.getX()) *
            (esquina2.getY() - esquina1.getY()));
}
```

El método presente método lo que realiza es comparar las coordenadas y retorna su área. En caso de algún negativo, usamos la función ABS para el valor absoluto y el evitar los valores negativos.

DISTANCIA – E

```
public double DistanciaE(){
    return Math.sqrt(Math.pow(esquina1.getX() - esquina2.getX(), 2) +
    Math.pow(esquina1.getY() - esquina2.getY(), 2));
}
```

El método usa Math.pow para elevar al cuadrado las diferencias de coordenadas y Math.sqrt para calcular la raíz cuadrada de la suma de esos valores.

Esto permite obtener la distancia sin importar la posición relativa de los puntos.

TO_STRING

```
@Override
public String toString() {
    return "Rectangulo {esquina1: " + esquina1 + "; esquina2: " + esquina2 +
'}';
}
```

Imprime el objeto

4.3 Verificador

Para verificador, tenemos diferentes funciones las cuales explicaremos una por una.

4.3.1 Uso en **esSobrePos** y **esJunto**

```
double r1x2 = r1.getEsquina2().getX();
double r1x1 = r1.getEsquina1().getX();
double r1y2 = r1.getEsquina2().getY();
double r1y1 = r1.getEsquina1().getY();

double r2x1 = r2.getEsquina1().getX();
double r2x2 = r2.getEsquina2().getX();
double r2y1 = r2.getEsquina1().getY();
double r2y2 = r2.getEsquina2().getY();
```

Esto se utiliza en sobre pos y en es junto, para obtener las coordenadas que se ordenaron antes.

4.3.2 ES_SOBRE_POS

```
public static boolean esSobrePos(Rectangulo r1, Rectangulo r2) {
    if (r1x2 > r2x1 && r1x1 < r2x2 && r1y2 > r2y1 && r1y1 < r2y2) {
        return true;
    }
    return false;
}
```

La logica que aqui manejamos, es una comprobación de adentro hacia afuera. Comprobamos que las coordenadas externas sean diferentes o igual, y lo mismo con las internas.

Lo trabajamos de esta manera tomando en cuenta que el usuario puede ingresar cualquier rectángulo, en cualquier orden.

4.3.3 ES_JUNTO

```
public static boolean esJunto(Rectangulo r1, Rectangulo r2) {
    if(esSobrePos(r1,r2)){
        return false;
    }

    boolean tocaX = (r1x2 == r2x1 || r1x1 == r2x2);
    boolean tocaY = (r1y2 == r2y1 || r1y1 == r2y2);

    return (tocaX && (r1y2 >= r2y1 && r1y1 <= r2y2)) ||
            (tocaY && (r1x2 >= r2x1 && r1x1 <= r2x2));
}
```

La lógica manejada aquí son comprobaciones de si algún punto cardinal es equivalente a otro, o si están en el mismo eje **X** o **Y**.

Adicionalmente, se le agrega una comprobación de mayor menor para saber si están tocándose.

4.3.4 ES_DISJUNTO

```
public static boolean esDisjunto(Rectangulo r1, Rectangulo r2) {
    return !(esSobrePos(r1, r2) || esJunto(r1, r2));
}
```

Para esta función, comprobamos que los casos anteriores sean falsos.

4.3.5 MOSTRAR_R

```
public static void mostrarR(Rectangulo re1, Rectangulo re2) {
    Rectangulo nre = rectanguloSobre(re1, re2);

    System.out.println("\nRectángulo 1: " + re1);
    System.out.println("Rectángulo 2: " + re2);
    System.out.println("Relación entre los rectángulos:");

    boolean esSobrePos = esSobrePos(re1, re2);
    boolean esJunto = esJunto(re1, re2);

    System.out.println("\nEstán sobrepuestos: " + esSobrePos);
    System.out.println("Estan juntos (se tocan sin sobreponerse): " +
esJunto);
    System.out.println("Son disjuntos : " + esDisjunto(re1, re2));

    if (esSobrePos) {
        System.out.println("\nEl rectángulo de intersección es: \n" + nre);
        System.out.println("\nEl área de la intersección es: \n" +
nre.area());
    }
}
```

La función de este método es imprimir los diferentes casos de uso, como también los rectángulos que se están utilizando.

4.3.6 RECTANGULO_SOBRE

```
public static Rectangulo rectanguloSobre(Rectangulo r1, Rectangulo r2) {
    double interseccionXMin = Math.max(r1.getEsquina1().getX(),
r2.getEsquina1().getX());
    double interseccionYMin = Math.max(r1.getEsquina1().getY(),
r2.getEsquina1().getY());
    double interseccionXMax = Math.min(r1.getEsquina2().getX(),
r2.getEsquina2().getX());
    double interseccionYMax = Math.min(r1.getEsquina2().getY(),
r2.getEsquina2().getY());

    return new Rectangulo(
            new Coordenada(interseccionXMin, interseccionYMin),
            new Coordenada(interseccionXMax, interseccionYMax)
    );
}
```

Este método retorna el nuevo rectángulo creado cuando 2 rectangulos están cumpliendo con la función EsSobrePos. Aquí aplicmaos algunas condiconales simples para comprobar los casos.

4.4 Contenedor de los Rectángulos

4.4.1 VARIABLES - CONSTRUCTOR

```
public class ContainerRect {
    Rectangulo[] rectangulos;
    double[] distanciasEcleudianas;
    double[] areas;

    static int numRect = 0;

    public ContainerRect () {
       this.rectangulos = new Rectangulo[9];
       this.distanciasEcleudianas = new double[9];
       this.areas = new double[9];
    }
```

4.4.2 ADD_RECTANGULO

```
public void addRectagulo(Rectangulo re){
    if (numRect <= 9){
       this.rectangulos[numRect] = re;
       this.distanciasEcleudianas[numRect] = re.DistanciaE();
       this.areas[numRect] = re.area();
       numRect += 1;
    } else {
       System.out.println("Container lleno");
    }
}
```

El añadir rectángulos será posible siempre y cuando **numRect** la variable que usamos siga siendo menor o igual a 9, nuestro contenedor solo podrá admitir hasta 10 rectángulos, no podrá admitir más.

4.4.3 FUNCIONES_GENERALES

```
public void showrectangulos(){
    for (int i = 0; i < numRect; i++){
       System.out.println(i + " ->  " + rectangulos[i]);
    }
}

public Rectangulo getRectangulo(int i){
    return rectangulos[i];
}
public int getNumrec(){
    return numRect;
}
```

Métodos que iremos llamando dependiendo de las necesidades del programa.

4.4.4 TO_STRING

```
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();

    sb.append(String.format("%-10s %-15s %-15s %-10s %-10s%n",
                            "Index", "Esquina M", "Esquina N", "Distancia",
"Área"));
    sb.append("-----------------------------------------------------------------
\n");

    for (int i = 0; i < numRect; i++) {
        sb.append(String.format("%-10d %-15s %-15s %-10.2f %-10.2f%n",
           i,
           rectangulos[i].getEsquina2().toString(),
           rectangulos[i].getEsquina1().toString(),
           distanciasEcleudianas[i],
           areas[i]));
    }
       return sb.toString();
}
```

En este **toString** usamos el **StringBuilder** el cual se encarga básicamente de construir la salida de un string de manera eficiente, todo trabajando con referencias.

Con el métodos **String.format( )** podemos construir el string y sus respectivas referencias, si a esto utilizamos el método **append( )** lo que hará será agregar toda la información a nuestro **StringBuilder(sb).**

Las referencias que usamos controlan el espacio y el formato de cada dato:

- %-10d → Número entero con ancho de 10 caracteres.
- %-15s → Texto alineado a la izquierda con 15 caracteres.
- %-10.2f → Número decimal con 2 decimales y ancho de 10 caracteres.
- %n → Salto de línea.

4.5 Rectángulo Grafico

Para graficar los rectangulos empleamos diferentes métodos y estrategias para un correcto trabajo.

4.5.1 VARIABLES

```
public class RectanguloGrafico extends JPanel {
    private List<Rectangulo> rectangulos;

    // Colores para los rectangulos o,o
    private static final Color[] COLORES = {
            new Color(102, 204, 255, 180),
            new Color(255, 153, 153, 180)};

    // Fondo oscuro
    private static final Color COLOR_FONDO = new Color(35, 35, 40);
    private static final Color COLOR_EJES = new Color(70, 70, 75);
    private static final Color COLOR_TEXTO = new Color(220, 220, 220);

    // Escalas, ejes y distancia entre puntos
    private static final int ESCALA = 35;
    private static final int LIMITE_EJE = 20;

    // Arrastrar el gráfico
    private Point lastPoint;
    private int offsetX = 0;
    private int offsetY = 0;

    private static JFrame frame;
    private static RectanguloGrafico panel;
```

Aquí las variables que iremos utilizando en el desarrollo de estos gráficos 2D.

4.5.2 RECTANGULO_GRAFICO

```
public RectanguloGrafico(List<Rectangulo> rectangulos) {
    this.rectangulos = rectangulos;
    setPreferredSize(new Dimension(600, 600));
    setBackground(COLOR_FONDO);
    configurarArrastre();
}
```

Esto de aquí se utiliza para configurar el tamaño de la pantalla que tendrá el grafico y para directamente agregar la configuración de Arrastre

4.5.3 CONFIGURACION_ARRASTRE

```
private void configurarArrastre() {
    MouseAdapter arrastreAdapter = new MouseAdapter() {
        @Override
        public void mousePressed(MouseEvent e) {
            lastPoint = e.getPoint();
        }
        @Override
        public void mouseDragged(MouseEvent e) {
            Point currentPoint = e.getPoint();
            offsetX += (currentPoint.x - lastPoint.x);
            offsetY += (currentPoint.y - lastPoint.y);
            lastPoint = currentPoint;
            repaint();
        }
    };

    addMouseListener(arrastreAdapter);
```

```
    addMouseMotionListener(arrastreAdapter);
}
```

Este metodo lo que hace es configurar los eventos del mouse para que pueda arrastrar el grafico. Posteriormente utilizamos la función **mousePressed()** la cual se ejecuta cuando presionamos algún botón del mouse. Además, **e.getPoint()** guarda la posición donde ha sido accionado el mouse.

MouseDragged() es un método que se ejecuta continuamente (lo interpreto como si se tratase de hilos), el uso de este **"Point currentPoint = e.getPoint();"** es para obtener la posicion actual del cursor. Las siguiente 2 de **"offsetX-Y"**, lo que realizan es comparar las coordenadas actuales con las anteriores. Posteriormente, lastPoint actualiza la referencia para el próximo calculo y repaint solicita que vuelva a pintar o dibujar.

Registra el adaptador para eventos como presionar, soltar y movimiento del ratón como arrastrar respectivamente.

4.5.4 PAINT_COMPONET

Por la complejidad de esta función, se va a subdividir.

4.5.4.1 VARIABLES

```
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D) g;

    // Punto central de la pantalla (origen de coordenadas) con offset para
arrastre
    int centerX = getWidth() / 2 + offsetX;
    int centerY = getHeight() / 2 + offsetY;
```

El Super.paintComponent lo usamos para asegurarnos que realicen las operaciones de dibujo.

También empleamos Graphics2D, ya que ofrece mas funcionalidades y mejor calidad de renderizado.

Las siguientes instrucciones que le siguen es para calcular las posiciones, horizontal y vertical, esto con el movimiento acumulado por el arrastre.

```
// Dibujar ejes X e Y
g2d.setColor(COLOR_EJES);
g2d.drawLine(centerX, 10, centerX, getHeight() - 10); // Eje Y
g2d.drawLine(10, centerY, getWidth() - 10, centerY); // Eje X

// Dibujar marcas en los ejes
g2d.setFont(new Font("Monospaced", Font.PLAIN, 10));
g2d.setColor(COLOR_TEXTO);
```

Lo que hace es ponerles color a los ejes, posteriormente

Las líneas que le siguen es para ponerle un borde de 10 pixeles, no se nota mucho, pero esto cambia la vista si se manejan los márgenes muy altos, afecta tanto a X y a Y.

El setFont() es útil para números, caracteres donde ocupan el mismo lugar. **Font.PLAIN** para un estilo de fuente normal, y el 10 es el tamaño que tomara la etiqueta.

```
g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
```

Lo que hacemos aquí es una técnica que suaviza los bordes de las figuras dibujadas (como líneas y círculos) para que no se vean pixeladas o con bordes irregulares.

RenderingHints.KEY_ANTIALIASING es la clave para cambiar a antialiasing, por otra parte **VALUE_ANTIALIAS_ON** es lo que activa esta función.

4.5.4.2 Marcas en eje X-Y y verificacion de arcas

```
for (int i = -LIMITE_EJE; i <= LIMITE_EJE; i++) {
    if (i == 0) continue;
    int x = centerX + i * ESCALA;
    if (x >= 10 && x <= getWidth() - 10) {
        g2d.drawLine(x, centerY - 3, x, centerY + 3);
        g2d.drawString(String.valueOf(i), x - 3, centerY + 15);
    }
}

for (int i = -LIMITE_EJE; i <= LIMITE_EJE; i++) {
    if (i == 0) continue;
    int y = centerY - i * ESCALA;
    if (y >= 10 && y <= getHeight() - 10) {
        g2d.drawLine(centerX - 3, y, centerX + 3, y);
        g2d.drawString(String.valueOf(i), centerX - 20, y + 4);
    }
}
```

Por esta parte, se dedica a dibujar las líneas, respetar los pixeles que no cuentan con líneas y números (ya vimos algo similar que apoya esto antes), además de dibujar pequeñas líneas y los respectivos números del grafico.

4.5.4.3 Dibujo del Rectangulo

```
for (int i = 0; i < rectangulos.size(); i++) {
    Rectangulo rect = rectangulos.get(i);

    double x1 = rect.getEsquina1().getX();
    double y1 = rect.getEsquina1().getY();
    double x2 = rect.getEsquina2().getX();
    double y2 = rect.getEsquina2().getY();

    double xMin = Math.min(x1, x2);
    double yMin = Math.min(y1, y2);
    double xMax = Math.max(x1, x2);
    double yMax = Math.max(y1, y2);

    int drawX = centerX + (int)(xMin * ESCALA);
    int drawY = centerY - (int)(yMax * ESCALA);
    int drawWidth = (int)((xMax - xMin) * ESCALA);
    int drawHeight = (int)((yMax - yMin) * ESCALA);

    // Dibujar rectángulo con color
    g2d.setColor(COLORES[i % COLORES.length]);
    g2d.fillRect(drawX, drawY, drawWidth, drawHeight);

    // Contorno
```

```
    g2d.setColor(Color.WHITE);
    g2d.drawRect(drawX, drawY, drawWidth, drawHeight);
```

Logica para dibujar el contorno, además de agregar el contorno.

4.5.4.4 Etiquetas en el rectángulo

```
// Etiqueta con número de rectángulo
g2d.setFont(new Font("SansSerif", Font.BOLD, 10));
g2d.setColor(COLOR_TEXTO);
g2d.drawString("R" + (i+1), drawX + 10, drawY + 25);

// coordenadas esquina inferior izquierda
g2d.setFont(new Font("Monospaced", Font.PLAIN, 10));
String coordsBottomLeft = String.format("(%.1f,%.1f)", xMin, yMin);
g2d.drawString(coordsBottomLeft, drawX + 5, drawY + drawHeight - 5);

// coordenadas esquina superior derecha
String coordsTopRight = String.format("(%.1f,%.1f)", xMax, yMax);
FontMetrics fm = g2d.getFontMetrics();
g2d.drawString(coordsTopRight, drawX + drawWidth -
fm.stringWidth(coordsTopRight) - 5, drawY + 15);
```

En general solo son posiciones de pixeles, y formato de letra.

El **drawString** solicita un mensaje por imprimir, las coordenas de X y Y y poco más. Podemos incluir formato de string.

Para la ultima sentencia, para la esquina superior derecha. Drawx + DrawWidth, posicional al final del rectángulo, **fm.stringWidth(coordsTopRight)** esta resta el ancho del texto para que no se salga del recntangulo, el -5 es un pequeño margen y el drawY + 15 posiciona el texto mejor.

4.5.4.5 MOSTRAR_GRAFICO

```
public static void mostrarGrafico(List<Rectangulo> rectangulos) {
    SwingUtilities.invokeLater(() -> {
        try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            e.printStackTrace();
        }

        // Reutilizar el frame existente si ya existe
        if (frame == null) {
            frame = new JFrame("Visualización de Rectángulos");
            frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
            frame.setSize(700, 700);
            frame.setLocationRelativeTo(null);

            panel = new RectanguloGrafico(rectangulos);

            JScrollPane scrollPane = new JScrollPane(panel);
            scrollPane.setBorder(BorderFactory.createEmptyBorder());

            frame.add(scrollPane);
```

```
            frame.setVisible(true);
        } else {
            // Actualizar el panel existente con los nuevos rectángulos
            panel.rectangulos = rectangulos;
            panel.repaint();

            // Asegurarse de que el frame sea visible
            if (!frame.isVisible()) {
                frame.setVisible(true);
            }
        }
    });
}
```

SwingUtilities.invokeLater((), este metodo permite que la UI se modifique correctamente sin errores de concurrencia.

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

No estamos consciente el uso de esto, solo sabemos que si lo tocamos el programa dejara de funcionar correctamente. Según GPT esto sirve para cambiar la apariencia de la UI de Swing, obteniendo el LOOK y FEEL nativo del sistema.

scrollPane.setBorder(BorderFactory.*createEmptyBorder*()); Nos ayuda a que la información y los procesos generados hasta ahora se puedan observar en la UI

5. Casos de Uso

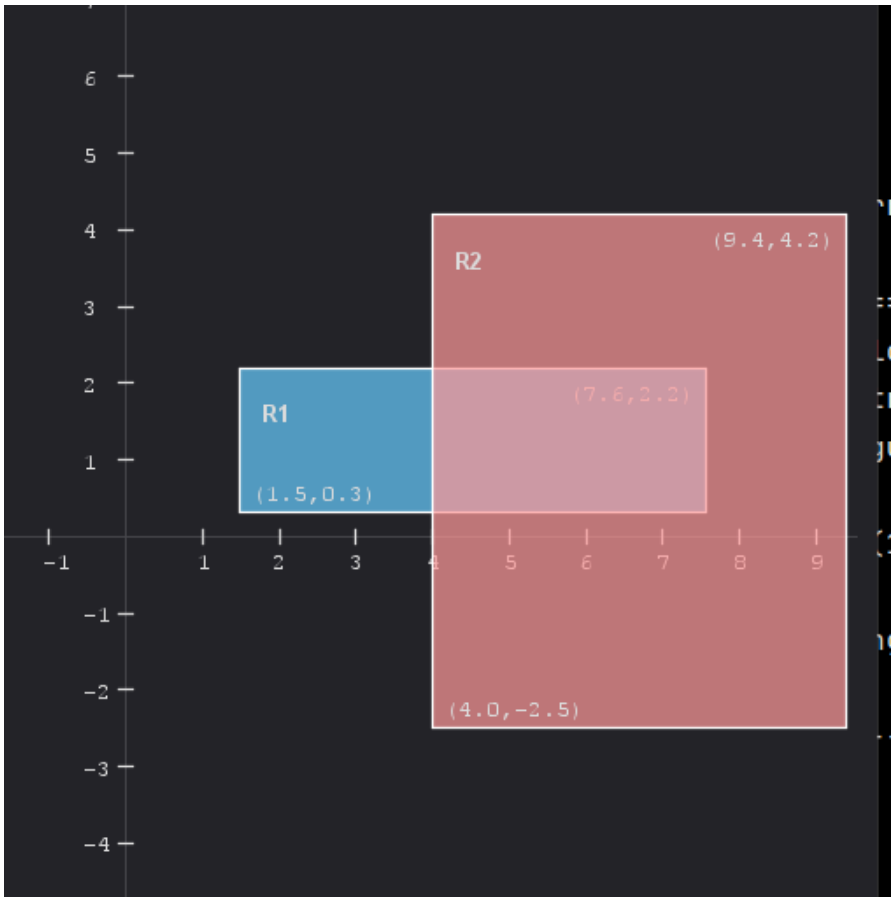
Prueba 1.

Ingreso:

```
Selecciona una opción (1-4): 1

=== Agregar un rectangulo ===
Ingrese x1: 7.6
Ingrese y1: 2.2
Ingrese x2: 1.5
Ingrese y2: 0.3
Rectángulo agregado correctamente :'p

===== MENU PRINCIPAL =====
1. Agregar un rectangulo
2. Mostrar relación entre dos rectangulos
3. Graficar dos rectangulos
4. Finalizar
Selecciona una opción (1-4): 1

=== Agregar un rectangulo ===
Ingrese x1: 9.4
Ingrese y1: -2.5
Ingrese x2: 4.0
Ingrese y2: 4.0
Rectángulo agregado correctamente :'p
```

Salida:

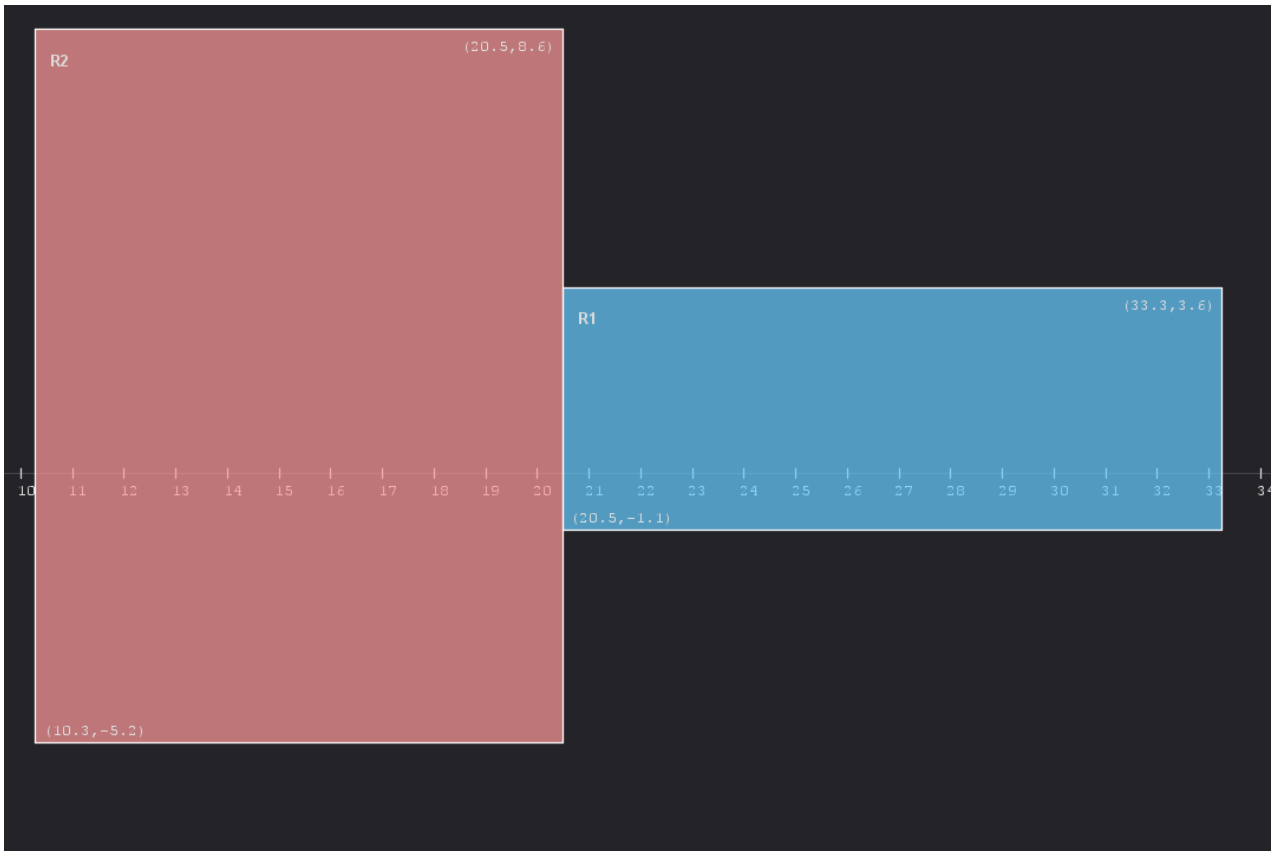
```
=== Mostrar relación entre dos rectángulos ===
Index      Esquina M       Esquina N       Distancia  Área
------------------------------------------------------------
0          (7.6;2.2)       (1.5;0.3)       6.39       11.59
1          (9.4;4.0)       (4.0;-2.5)      8.45       35.10

Selecciona el primer rectangulo (0 - 1): 0
Selecciona el segundo rectangulo (0 - 1): 1

Rectángulo 1: Rectangulo {esquina1: (1.5;0.3); esquina2: (7.6;2.2)}
Rectángulo 2: Rectangulo {esquina1: (4.0;-2.5); esquina2: (9.4;4.0)}
Relación entre los rectángulos:


Están sobrepuestos: true
Estan juntos (se tocan sin sobreponerse): false
Son disjuntos : false

El rectángulo de intersección es:
Rectangulo {esquina1: (4.0;0.3); esquina2: (7.6;2.2)}

El área de la intersección es:
6.84
```



Prueba2.

```
=== Agregar un rectangulo ===
Ingrese x1: 33.3
Ingrese y1: 3.6
Ingrese x2: 20.5
Ingrese y2: -1.1
Rectángulo agregado correctamente :'p

===== MENU PRINCIPAL =====
1. Agregar un rectangulo
2. Mostrar relación entre dos rectangulos
3. Graficar dos rectangulos
4. Finalizar
Selecciona una opción (1-4): 1

=== Agregar un rectangulo ===
Ingrese x1: 20.5
Ingrese y1: 8.6
Ingrese x2: 10.3
Ingrese y2: -5.2
Rectángulo agregado correctamente :'p
```

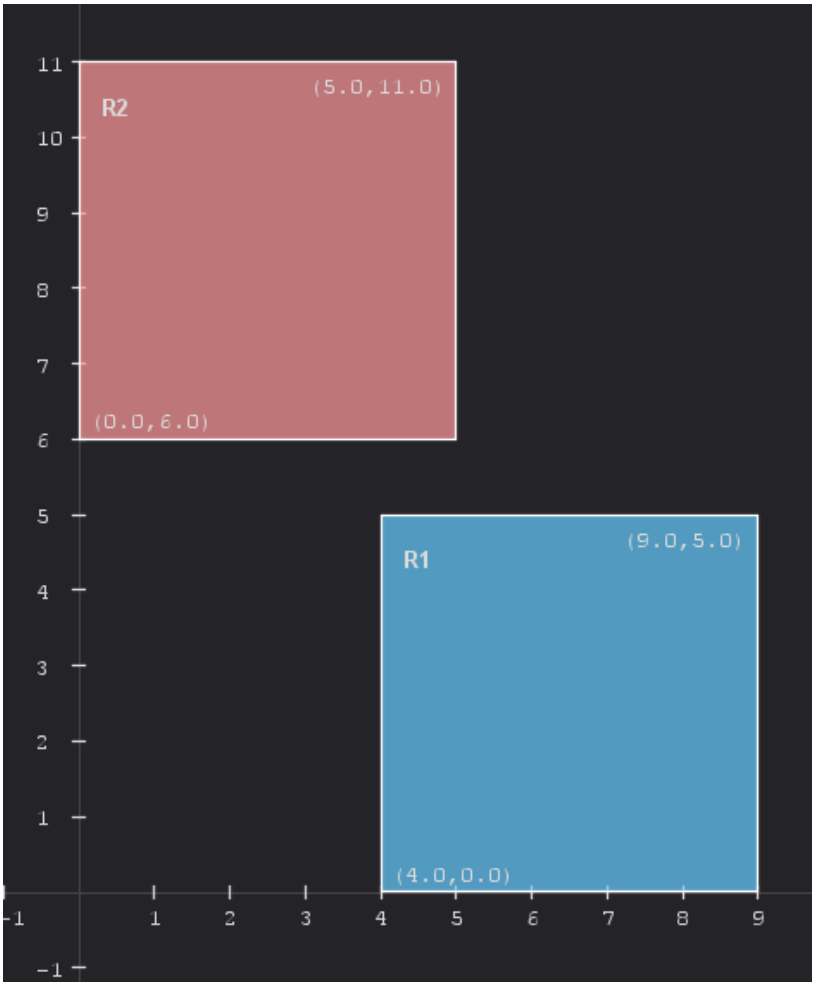
```
=== Mostrar relación entre dos rectángulos ===
Index      Esquina M       Esquina N       Distancia  Área
-------------------------------------------------------------
0          (33.3;3.6)      (20.5;-1.1)     13.64      60.16
1          (20.5;8.6)      (10.3;-5.2)     17.16      140.76

Selecciona el primer rectangulo (0 - 1): 1
Selecciona el segundo rectangulo (0 - 1): 0

Rectángulo 1: Rectangulo {esquina1: (10.3;-5.2); esquina2:
 (20.5;8.6)}
Rectángulo 2: Rectangulo {esquina1: (20.5;-1.1); esquina2:
 (33.3;3.6)}
Relación entre los rectángulos:

Están sobrepuestos: false
Estan juntos (se tocan sin sobreponerse): true
Son disjuntos : false
```



Prueba 3.

```
=== Agregar un rectangulo ===
Ingrese x1: 9
Ingrese y1: 5
Ingrese x2: 4
Ingrese y2: 0
Rectángulo agregado correctamente :'p

===== MENU PRINCIPAL =====
1. Agregar un rectangulo
2. Mostrar relación entre dos rectangulos
3. Graficar dos rectangulos
4. Mostrar Rectangulos
5. Finalizar
Selecciona una opción (1-5): 1

=== Agregar un rectangulo ===
Ingrese x1: 5
Ingrese y1: 11
Ingrese x2: 0
Ingrese y2: 6
Rectángulo agregado correctamente :'p
```

```
=== Mostrar relación entre dos rectángulos ===
Index      Esquina M       Esquina N       Distancia  Área
------------------------------------------------------------
0          (9.0;5.0)       (4.0;0.0)       7.07       25.00
1          (5.0;11.0)      (0.0;6.0)       7.07       25.00

Selecciona el primer rectangulo (0 - 1): 0
Selecciona el segundo rectangulo (0 - 1): 1

Rectángulo 1: Rectangulo {esquina1: (4.0;0.0); esquina2: (9.0;5.0)}
Rectángulo 2: Rectangulo {esquina1: (0.0;6.0); esquina2: (5.0;11.0)}
Relación entre los rectángulos:

Están sobrepuestos: false
Estan juntos (se tocan sin sobreponerse): false
Son disjuntos : true
```



6. Ejercicio

Evidencia del ejercicio, el mostrar toda la cadena de objetos.

Aclaración.

M significa la coordenada mayor. Por otro lado, N significa la coordenada menor.

```
===== MENU PRINCIPAL =====
1. Agregar un rectangulo
2. Mostrar relación entre dos rectangulos
3. Graficar dos rectangulos
4. Mostrar Rectangulos
5. Finalizar
Selecciona una opción (1-5): 4
Index      Esquina M       Esquina N       Distancia  Área
-------------------------------------------------------------
0          (9.0;5.0)       (4.0;0.0)       7.07       25.00
1          (5.0;11.0)      (0.0;6.0)       7.07       25.00
2          (7.6;2.2)       (1.5;0.3)       6.39       11.59
3          (9.4;4.2)       (4.0;-2.5)      8.61       36.18
4          (33.3;3.6)      (20.5;-1.1)     13.64      60.16
5          (20.5;8.6)      (10.3;-5.2)     17.16      140.76
```

7. CONCLUSIONES DE LA PRÁCTICA:

a) Se han desarrollado los conceptos básicos de POO a lo largo de la práctica, esto con ha ayudado ha desarrollar los diferentes métodos y funciones tanto de clase como de instancia aplicadas en el ejercicio.

b) A lo largo de la práctica, en equipo se ha desarrollado las clases e instancias que ayudaron al correcto funcionamiento del programa y sus respectivas comprobaciones.

c) Se logro aplicar los diferentes métodos de instancia y clases, esto debido a la necesidad del programa para poder optimizar ciertos procesos durante la ejecución.

d) Se logro y se determino el uso de los pilares de la programación durante la práctica, esto nos ayudó a determinar de una manera mas efectiva las variables, métodos y relaciones empleadas en el programa.

e) Se utilizaron diferentes arreglos para que puedan actuar como si fueran contenedores, esto nos ayuda a tener organizados los diferentes datos e información que el usuario iba ingresando poco a poco.

8. BIBLIOGRAFÍA

Blasco, J. L. (2023, 27 octubre). Introducción a POO en Java: Objetos y clases. *OpenWebinars.net*. https://openwebinars.net/blog/introduccion-a-poo-en-java-objetos-y-clases/#:~:text=NET%20y%20m%C3%A1s-,Qu%C3%A9%20son%20las%20clases%20en%20Java,crear%20a%20partir%20de%20ella.

Diagrama de clases: Qué es, cómo hacerlo y ejemplos | Miro. (s. f.). https://miro.com/. https://miro.com/es/diagrama/que-es-diagrama-clases-uml/

¿Qué es el SDLC? - Explicación del ciclo de vida del desarrollo de software - AWS. (s. f.). Amazon Web Services, Inc. https://aws.amazon.com/es/what-is/sdlc/

1. ANEXO: TECNOLOGÍAS, NORMAS Y ESTÁNDARES UTILIZADOS

https://github.com/Arthur845777/Algoritmia/tree/main