

1. Ejercicio 01

1.1. Descripción General

El código presentado consta de dos clases en Java: **Punto** y **Circulo**. Estas clases se utilizan para representar puntos en un plano cartesiano y círculos respectivamente. La clase **Circulo** hereda de la clase **Punto**, lo que implica que un objeto de tipo **Circulo** hereda todas las propiedades y métodos de un objeto **Punto**.

1.2. Clase Punto

1.2.1. Atributos

- `private double x`: Representa la coordenada x del punto.
- `private double y`: Representa la coordenada y del punto.

1.2.2. Constructor

- `public Punto(double x, double y)`: Inicializa las coordenadas (x, y) del punto con los valores proporcionados.

1.2.3. Métodos

- `public double distancia(Punto otroPunto)`: Calcula la distancia entre el punto actual y otro punto dado utilizando la fórmula de distancia euclidiana.
- `public double getX()`: Obtiene la coordenada x del punto.
- `public double getY()`: Obtiene la coordenada y del punto.
- `public void setY(double y)`: Establece la coordenada y del punto.
- `public void setX(double x)`: Establece la coordenada x del punto.

1.3. Clase Circulo

1.3.1. Atributos

- `private double radio`: Representa el radio del círculo.

1.3.2. Constructor

- `public Circulo(double x, double y, double radio)`: Inicializa las coordenadas (x, y) del centro del círculo y su radio. Utiliza la llamada al constructor de la clase base **Punto** mediante `super(x, y)`.

1.3.3. Métodos

- `public double getRadio()`: Obtiene el radio del círculo.
- `public void setRadio(double radio)`: Establece el radio del círculo.

1.4. Ejemplo de Uso

```
1 \begin{lstlisting}
2 \begin{lstlisting}[language=Java]
3 public class EjemploUso {
4     public static void main(String[] args) {
5         // Crear un punto en el plano cartesiano
6         Punto puntoA = new Punto(1.0, 2.0);
7
8         // Obtener las coordenadas del punto
9         double coordenadaX = puntoA.getX();
10        double coordenadaY = puntoA.getY();
11
12        // Crear un círculo con centro en el puntoA y radio 3.0
13        Circulo circuloA = new Circulo(coordenadaX, coordenadaY, 3.0);
14
15        // Obtener el radio del círculo
16        double radioCirculoA = circuloA.getRadio();
17
18        // Establecer nuevas coordenadas al puntoA
19        puntoA.setX(4.0);
20        puntoA.setY(5.0);
21
22        // Calcular la distancia entre el puntoA y el centro del círculoA
23        double distancia = puntoA.distancia(new Punto(coordenadaX, coordenadaY));
24    }
25 }
```

En este ejemplo, se muestran instancias de las clases **Punto** y **Circulo**, así como el acceso a sus métodos para obtener y establecer valores, y calcular la distancia entre el punto y el centro del círculo.

2. Ejercicio 02

2.1. Descripción General

En este segundo fragmento de código, se introduce la nueva clase: **Cilindro**. Además, se expande la clase **Circulo** que ahora hereda de la clase **Punto**. Estas clases están diseñadas para representar un cilindro tridimensional, puntos en un plano cartesiano, y círculos respectivamente.

2.2. Clase Cilindro

Atributos:

- `protected double longitud`: Representa la longitud del cilindro.

Constructor:

- `public Cilindro(double x, double y, double radio, double longitud)`: Inicializa las coordenadas (x, y) , el radio y la longitud del cilindro. Utiliza la llamada al constructor de la clase base **Circulo** mediante `super(x, y, radio)`.

Métodos:

- `public double superficie()`: Calcula la superficie del cilindro.
- `public void setLongitud(double longitud)`: Establece la longitud del cilindro.
- `public double getLongitud()`: Obtiene la longitud del cilindro.

2.3. Clase Punto (sin cambios)

Atributos:

- `protected double x`: Representa la coordenada x del punto.
- `protected double y`: Representa la coordenada y del punto.

Constructor:

- `public Punto(double x, double y)`: Inicializa las coordenadas (x, y) del punto con los valores proporcionados.

Métodos:

- `public double distancia(Punto otroPunto)`: Calcula la distancia entre el punto actual y otro punto dado utilizando la fórmula de distancia euclidiana.
- `public double getX()`: Obtiene la coordenada x del punto.
- `public double getY()`: Obtiene la coordenada y del punto.
- `public void setY(double y)`: Establece la coordenada y del punto.
- `public void setX(double x)`: Establece la coordenada x del punto.

2.4. Clase Circulo (sin cambios)

Atributos:

- `protected double radio`: Representa el radio del círculo.

Constructor:

- `public Circulo(double x, double y, double radio)`: Inicializa las coordenadas (x, y) y el radio del círculo. Utiliza la llamada al constructor de la clase base `Punto` mediante `super(x, y)`.

Métodos:

- `public double getRadio()`: Obtiene el radio del círculo.
- `public void setRadio(double radio)`: Establece el radio del círculo.

2.5. Ejemplo de Uso (sin cambios)

```
1 \begin{lstlisting}[language=Java]
2 public class EjemploUso {
3     public static void main(String[] args) {
4         // Crear un punto en el plano cartesiano
5         Punto puntoA = new Punto(1.0, 2.0);
6
7         // Obtener las coordenadas del punto
8         double coordenadaX = puntoA.getX();
9         double coordenadaY = puntoA.getY();
10
11         // Crear un círculo con centro en el puntoA y radio 3.0
12         Circulo circuloA = new Circulo(coordenadaX, coordenadaY, 3.0);
13
14         // Obtener el radio del círculo
15         double radioCirculoA = circuloA.getRadio();
16     }
17 }
```

```
17 // Establecer nuevas coordenadas al puntoA
18 puntoA.setX(4.0);
19 puntoA.setY(5.0);
20
21 // Calcular la distancia entre el puntoA y el centro del círculoA
22 double distancia = puntoA.distancia(new Punto(coordenadaX, coordenadaY));
23 }
24 }
```

3. Ejercicio 03

3.1. Descripción General

En este ejercicio, se ha implementado un conjunto de interfaces y una clase para representar la funcionalidad de un hidroavión. Se han definido dos interfaces, **Barco** y **Avion**, cada una con un método que representa la acción específica de navegar y volar, respectivamente. Además, se ha creado una clase **Hidroavion** que implementa ambas interfaces, permitiendo que un objeto de esta clase pueda realizar tanto operaciones de barco como de avión.

3.2. Código

Interfaces:

```
1 interface Barco {
2     void navegar();
3 }
4
5 interface Avion {
6     void volar();
7 }
```

Clase Hidroavion:

```
1 class Hidroavion implements Barco, Avion {
2     @Override
3     public void navegar() {
4         System.out.println("Hidroavion navegando en el agua");
5     }
6
7     @Override
8     public void volar() {
9         System.out.println("Hidroavion navegando en el aire");
10    }
11 }
```

Clase de Prueba (Ejercicio3):

```
1 public class Ejercicio3 {
2     public static void main(String[] args) {
3         // Crear una instancia de hidroavion
4         Hidroavion hidroavion = new Hidroavion();
5
6         // Llamar a los métodos de las interfaces
7         hidroavion.navegar();
8         hidroavion.volar();
9     }
10 }
```

3.3. Explicación del Código

Interfaces (Barco y Avion):

- Se han definido dos interfaces, una para la funcionalidad de barco y otra para la funcionalidad de avión. Cada interfaz contiene un método que representa la acción específica asociada con la respectiva interfaz.

Clase Hidroavion:

- Esta clase implementa ambas interfaces **Barco** y **Avion**.
- Se han proporcionado las implementaciones concretas para los métodos **navegar** y **volar** en función del comportamiento esperado de un hidroavión.

Clase de Prueba (Ejercicio3):

- En la clase de prueba, se crea una instancia de **Hidroavion**.
- Se invocan los métodos **navegar** y **volar** para demostrar la funcionalidad dual del hidroavión, capaz de navegar tanto en el agua como en el aire.

3.4. Ejemplo de Ejecución

Al ejecutar la clase **Ejercicio3**, se obtendrá la siguiente salida:

```
1 Hidroavion navegando en el agua
2 Hidroavion navegando en el aire
```

Este resultado demuestra que el hidroavión es capaz de realizar ambas acciones, navegar en el agua y volar en el aire, gracias a la implementación de las interfaces **Barco** y **Avion**.