# Practica 03

## 1. Ejercicio 01

### 1.1. Descripción General

El código presentado para el primer ejercicio 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

```java
public class EjemploUso {
  public static void main(String[] args) {
    // Crear un punto en el plano cartesiano
    Punto puntoA = new Punto(1.0, 2.0);

    // Obtener las coordenadas del punto
    double coordenadaX = puntoA.getX();
    double coordenadaY = puntoA.getY();

    // Crear un crculo con centro en el puntoA y radio 3.0
    Circulo circuloA = new Circulo(coordenadaX, coordenadaY, 3.0);

    // Obtener el radio del crculo
    double radioCirculoA = circuloA.getRadio();

    // Establecer nuevas coordenadas al puntoA
    puntoA.setX(4.0);
    puntoA.setY(5.0);

    // Calcular la distancia entre el puntoA y el centro del crculoA
    double distancia = puntoA.distancia(new Punto(coordenadaX, coordenadaY));
  }
}
```

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

Para el segundo ejercicio, 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

```java
public class EjemploUso {
  public static void main(String[] args) {
    // Crear un punto en el plano cartesiano
    Punto puntoA = new Punto(1.0, 2.0);

    // Obtener las coordenadas del punto
    double coordenadaX = puntoA.getX();
    double coordenadaY = puntoA.getY();

    // Crear un cilindro con centro en el puntoA, radio 3.0 y altura 10.0
    Cilindro cilindro = new Cilindro(coordenadaX, coordenadaY, 3.0, 10.0);

    // Obtener la superficie del cilindro
    double superficieCilindro = cilindro.superficie();
  }
}
```

# 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:**

```java
interface Barco {
  void navegar();
}

interface Avion {
  void volar();
}
```

**Clase Hidroavion:**

```java
class Hidroavion implements Barco, Avion {
  @Override
  public void navegar() {
    System.out.println("Hidroavion navegando en el agua");
  }

  @Override
  public void volar() {
    System.out.println("Hidroavion navegando en el aire");
  }
}
```

**Clase de Prueba (`Ejercicio3`):**

```java
public class Ejercicio3 {
  public static void main(String[] args) {
    // Crear una instancia de hidroavion
    Hidroavion hidroavion = new Hidroavion();

    // Llamar a los mtodos de las interfaces
    hidroavion.navegar();
    hidroavion.volar();
  }
}
```

## 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:

```
Hidroavion navegando en el agua
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`.