Bloque 2

Introducción al diseño de tipos

Objetos, interfaces, clases

Fundamentos de Programación
Departamento de Lenguajes y Sistemas Informáticos

2023/2024

Ejercicio. Tipo Punto

2023/24

- Gráficamente la clase ***Punto*** puede ser algo así:

public class Punto

private Double x;
private Double y;

public *Punto* (Double x, Double y)
public *Punto* ()
public Double *getX*()
public Double *getY*()
public void *setX*(Double x)
public void *setY*(Double y)
public String *toString*())
public *distancia* (Punto p)

2023/24

Ejercicio. Tipo Punto (Creación objetos)

Antes de continuar vamos a ver de forma gráfica como es el proceso de creación e inicialización de un objeto Paso 1.

TestPunto.java

```
Punto p1=new Punto(1.0,2d);
```

Punto.java

```
private Double x;
private Double y;

public Punto (Double x, Double y) {
   this.x=x;
   this.y=y;
}
```

1.0

2d

2023/24

Ejercicio. Tipo Punto (Creación objetos)

Antes de continuar vamos a ver de forma gráfica como es el proceso de creación e inicialización de un objeto Paso 2.

TestPunto.java

```
Punto p1=new Punto(1.0,2d);
```

Punto.java

```
private Double x;
private Double y;

public Punto (Double x, Double y) {
   this.x=x;
   this.y=y;
}
```

1.0 2d

Se realizan las asignaciones

2023/24

Ejercicio. Tipo Punto (Creación objetos)

Antes de continuar vamos a ver de forma gráfica como es el proceso de creación e inicialización de un objeto Paso 3.

TestPunto.java

```
Punto p1=new Punto(1.0,2d);
```

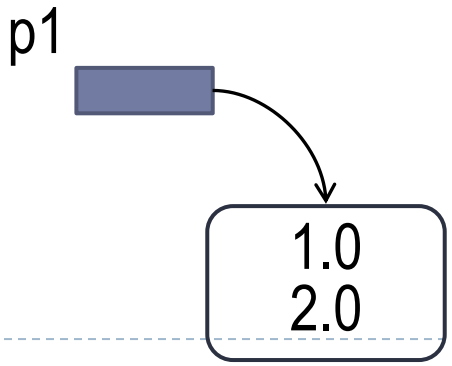
Punto.java

```
private Double x;
private Double y;

public Punto (Double x, Double y) {
   this.x=x;
    this.y=y;
}
```

1.0

2d



2023/24

Ejercicio. Tipo Punto (get)

Antes de continuar vamos a ver de forma gráfica como es el proceso de consulta de una propiedad básica o atributo

TestPunto.java

```
Punto p1=new Punto(1.0,2d);

System.out.println("Ordenada
de p1="+p1.getY());
```

2d

Punto.java

```
private Double x;
private Double y;

public Double getY(){
    return this.y;
}
```

1.0

2d

Consola

```
Ordenada de p1=2.0
```

2023/24

Ejercicio. Tipo Circunferencia

Atributos:

- Parece lógico que, los atributos sean el *centro* y el *radio* de la circunferencia

- ¿De qué tipo será el *centro*?. Nuestro proyecto ya dispone del tipo Punto, ¡aprovechémoslo!: el centro será de tipo *Punto*.
- ¿Parece lógico que el *radio* admita magnitudes con decimales?. Digamos que sí. Entonces radio deberá ser de tipo Float o Double. Escojamos *Double*.

2023/24

Ejercicio. Tipo Circunferencia

Métodos (*qué operaciones deseamos para nuestro tipo Circunferencia*):

- Un *constructor* que reciba las dos propiedades básicas.
- Dos métodos *consultores*: Uno para cada propiedad básica.
- Dos métodos *modificadores*. Uno para cada propiedad básica.
- La *representación textual* del objeto: Como la de Punto seguida de un espacio, una R: y el valor del radio. Por ejemplo: (1.0,2.0) R:2.5
- Un método *longitud* que devuelva la longitud de la circunferencia. Utiliza Math.PI para obtener el valor de π
- Un método *área* que devuelva el área del círculo interior a la circunferencia. Puedes multiplicar el radio por sí mismo o usar Math.pow(base, exponente) para calcular la potencia del radio elevado al cuadrado.

¡Hala! pues a Eclipse a definir los atributos y los métodos

2023/24

Ejercicio. Tipo Circunferencia

Test (*¡tendremos que probar que hemos implementado bien el tipo!*):

Dentro del método ***main*** que estará en la clase ***TestCircunferencia*** haremos lo siguiente:

1. Construiremos un punto "***centro***" con los valores (1, 2)
2. Construiremos una circunferencia "***miCircunferencia***" usando como parámetros del constructor el ***centro*** y un ***radio*** con una longitud de 2.5.
3. Visualizaremos ***miCircunferencia***.
4. Visualizaremos la longitud de ***miCircunferencia***.
5. Visualizaremos el área del círculo interior a ***miCircunferencia***.

Si has llegado hasta aquí sin copiar de otros ni de otras.

¡Enhorabuena! vas por buen camino

2023/24

Ejercicio. Tipo Circunferencia

Corregimos el ejercicio

Ejercicio. Tipo Circunferencia: otra implementación

Nuevo planteamiento sobre el modelado de tipos:

- Hemos conseguido modelar una circunferencia usando como atributos el centro y el radio.

- La pregunta es:
  - ¿Podríamos modelar una circunferencia usando otros atributos?
  - En caso afirmativo, ¿Qué atributos nos permitiría también modelar una circunferencia?

2023/24

Ejercicio. Tipo Circunferencia: otra implementación

Atributos:

- Qué tal, si en lugar de guardar el centro y el radio como atributos, guardamos el *centro* y, por ejemplo, el *área*.

Ejercicio. Tipo Circunferencia: otra implementación

Procedimiento para CircunferenciaImpl2:

Veamos como lo hacemos:

1. Para guardar Circunferencia como la tenemos, en eclipse la renombramos a CircunferenciaOrigen. Asimismo, la *copiamos y pegamos* en el paquete geometría y a la copia la denominamos *CircunferenciaImpl1*
2. *Copiemos y peguemos* en el mismo paquete geometría *CircunferenciaImpl1* y a la copia le denominamos *CircunferenciaImpl2*.
3. Ahora se trata de *modificar* CircunferecniaImpl2 para cambiar el atributo *radio por área* y, sin modificar las cabeceras de los métodos, hacer los *cambios en el código de los métodos* que lo necesiten para que sigan haciendo lo que ya hacían.

sigue…

2023/24

Ejercicio. Tipo Circunferencia: otra implementación

Procedimiento para CircunferenciaImpl2:

Veamos como lo hacemos:

4. Copia y pega en el paquete geometría.test el testGeometría y le denominas testGeometría2.
5. Modifica el código para que use CircunferenciaImpl1
6. Copia y pega debajo el código, pero cambiando que el objeto que creamos con la implementación 2 se llama otraCircunfreencia y comprobemos que los métodos se siguen comportando igual con independencia de la implementación.

¡Hala! pues a Eclipse a definir los nuevos atributos y los ajustes en los métodos

2023/24

Ejercicio. Tipo Circunferencia: otra implementación más

Procedimiento para CircunferenciaImpl3:

Si te apetece existe una tercera implementación.

¿te atreves?

¿Cómo se definen nuevos tipos objeto?

2023/24

- A veces, se descompone la definición de un nuevo tipo en dos elementos: una **interfaz** y una **clase**.
- Una **interfaz** es una descripción de las funcionalidades:
  - Sólo necesito conocer la interfaz para saber usar el objeto.
  - Si cambio la clase del objeto, pero se mantiene la interfaz, sigo sabiendo usar el objeto.

2023/24

Interfaces y clases

- Normalmente cuando se descompone la definición de un tipo en dos elementos: una **interfaz** y una **clase** es porque existen más de una implementación del tipo.
- Una **interfaz** es una descripción de las funcionalidades:
  - Sólo se indica qué métodos tendrá el tipo *(pero no se explica cómo se construyen esos métodos*).

```
Interfaz Coche:
   arrancar()
   pararMotor()
   acelerar()
   frenar()
   cambioDeMarcha()
   abrirPuerta(número de la puerta)
```

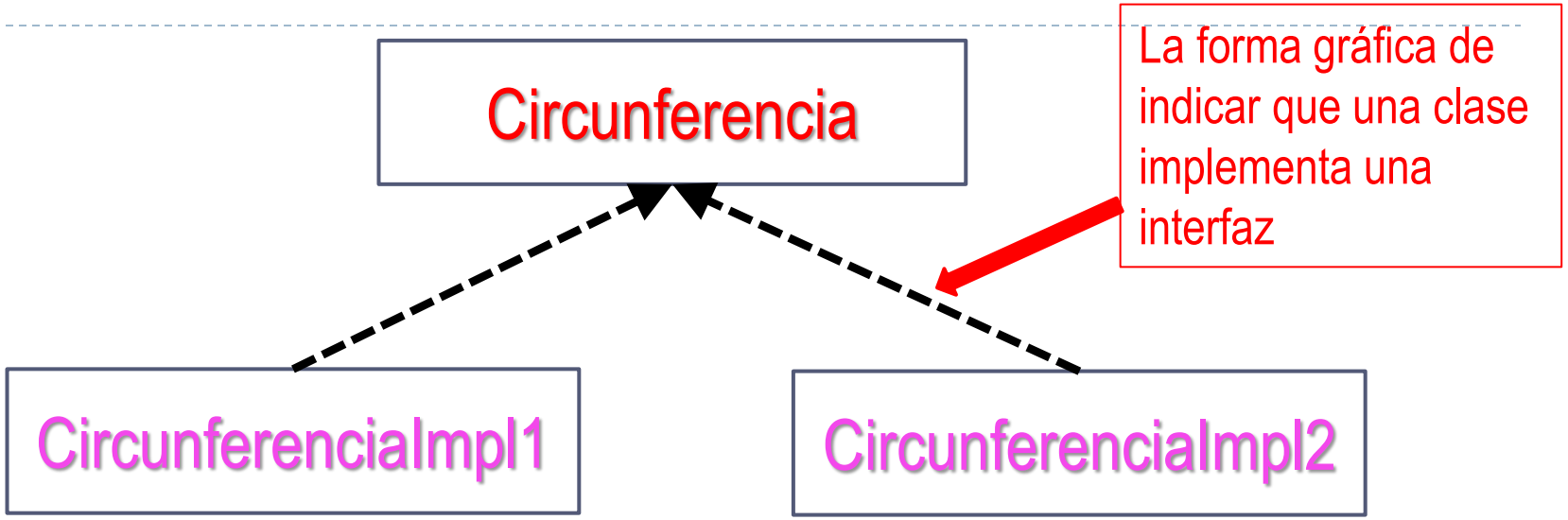
CUIDADO: esto no es código en Java es una forma de escribir

Interfaz

Una **interfaz** es una descripción de las funcionalidades:

- Sólo necesito conocer la interfaz para saber usar el objeto.
- Si cambio la clase que implementa la interfaz a la hora de construir un objeto sigo sabiendo usar el objeto.

Interfaz Circunferencia



- Definir una Interfaz consiste *en definir las funcionalidades* (los métodos) de que dispondrá el tipo.
- Se describen todos los métodos públicos *a excepción de los constructores, toString(), equals() y hashcode()* *(ya veremos estos dos últimos)*

Interfaz

Sintaxis de construcción de una interfaz:

Dentro del paquete se crea un elemento interface con la siguiente estructura:

```
public interface NombreInterfaz{
   Tipo nombreMétodo1(…);
   Tipo nombreMétodo2(…);
   …
}
```

Cuando se cree una clase que implemente la interfaz, en la cabecera de la clase hay añadir:

```
public class NombreDeLaClase implements NombreInterfaz{
…
}
```

Ello obligará a "programar" en la clase todos los métodos referenciados en la interfaz.

Interfaz Circunferencia

Interfaz Circunferencia:

- Dentro del paquete *geometría* defina la interfaz *Circunferencia*. Con las funcionalidades que se esperan de una circunferencia
- Modifique las clases *CircunferenciaImpl1* y *CircunferenciaImpl2*, para que implemente la interfaz *Circunferencia*
- En el paquete *geometría.test* copie TestCircunferencia2 .y pegue como *TestCircunferencia3* y modifique los objetos para que sean de tipo Circunferencia y mantenga las clases con la que se implementan.

¡Todo debe funcionar “redondo”!

Resumen: Clases con o sin Interfaz

2023/24

Hemos aprendido a crear **tipos** valiéndonos de dos herramientas: la Interfaz y/o la/s Clase**s** que la implementa**n**.

- El tipo *Punto* sólo tiene una implementación y por lo tanto sólo creamos la clase Punto. A la hora de construir puntos los hacemos definiéndolos y construyéndolos con el nombre de la clase:

  Punto p1=new *Punto* (1.5,-9.8);

  Punto p2=new *Punto* (-83.8, 5.389);

- El tipo *Circunferencia* tiene más de una implementación (*Impl1, Impl2 y existe Impl3 que se queda a la curiosidad del alumno*). En este caso, definimos una interfaz y las clases que la implementan. En consecuencia, definiremos las circunferencias con la interfaz y las creamos con las clases.

  Circunferencia cA=new *CircunferenciaImpl1*(new Punto(1.2,7.4), 2.5);

  Circunferencia cB=new *CircunferenciaImpl2*(new Punto(1.2,7.4), 2.5);

  Circunferencia cC=new *CircunferenciaImpl1*(new Punto(-4.2,9.4), 7D);

Elegimos la que queramos. ¡Funcionan igual!

2023/24

Clases de utilidad

- Existen otras clases que no construyen ningún objeto y se llaman ***clases de utilidad***. Estas clases no suelen tener atributos, sino una serie de métodos ***static*** que, para invocarlos, al carecer de objetos con los que invocarlos, se utiliza el ***nombre de la clase***.

  Por ejemplo:

  - *Math* (contiene métodos y constantes para trabajar con operaciones matemáticas).
    - *Math.sqrt(…)* → forma de invocar al método que calcula la $\sqrt{\ }$
  - *Checkers* (contiene métodos para comprobar las restricciones de las propiedades de los objetos)
    - *Checkers.checkNoNull(…)* → forma de invocar al método que comprueba que los parámetros no sean nulos

Records

2023/24

Es un elemento del lenguaje Java que permite crear un tipo de manera muy simplificada con las siguientes características por defecto:

- *No se especifican atributos*, sino que sus nombres son los que se utilicen como *parámetros* en la sentencia *record*. Aunque internamente si existen dentro del tipo.
- Los objetos que se crean son Inmutables (no pueden modificarse una vez creado)
- No tiene métodos explícitos. Existen los métodos *"get"* pero no hace falta escribirlos. Se denominan como los *atributos* sin la partícula *"get" delante*.
- No existen métodos *"set"* porque el tipo es inmutable.
- Tampoco hace falta escribir el método *toString()*, por defecto tiene internamente uno. Aunque se puede escribir uno a nuestras necesidades que oculta al que tiene por defecto.

2023/24

Records

Supongamos que se quiere modelar un tipo *Persona* que será inmutable (una vez creado un objeto su estado no cambia), con las siguientes propiedades básicas (que dan lugar a los atributos): dni, nombre, apellidos y la fecha de nacimiento. En este caso lo ideal es implementarlo mediante *record*

Sintaxis:

```
public record Persona (String dni, String nombre, String
                       apellidos, LocalDate fechaNacimiento) {
}
```

¡Ya tenemos todos los elementos para manejar los objetos"

Records

2023/24

```
public record Persona (String dni, String nombre, String
       apellidos, LocalDate fechaNacimiento) {
}
```

Por el hecho de haber construido este tipo a través de *record* disponemos de:

- Un constructor con las cuatro propiedades

```
Persona p1=new Persona("12345678A","Juana","Gómez
                  Pérez",LocalDate.of(2000, 1, 1));
```

- Métodos consultores (uno por cada atributo)

```
p1.dni(); p1.nombre();
p1.apellidos();  p1.fechaNacimiento();
```

- Representación textual

```
p1.toString();
```
o directamente escribir `p1`

- Criterio de igualdad y hashCode

```
p1.equals(p2);        p1.hashCode(p2);
```

2023/24

Records (*Ejemplo*)

```
public record Persona (String dni, String nombre, String
       apellidos, LocalDate fechaNacimiento) {
}
```

Se pueden incluir dentro del bloque del record, "a mano" otros métodos. Por ejemplo, *getEdad()* o *getNombre()* -este equivale al método *nombre()*-

```
public record Persona (String dni, String nombre, String
       apellidos, LocalDate fechaNacimiento) {
   public Integer getEdad() {
      return this.fechaNacimiento.
             until(LocalDate.now()).getYears();
   }
   public String getNombre() {
      return this.nombre;
   }
}
```

Practicamos. Proyecto: T02_Tipos

2023/24

Proyecto: *T02_Tipos*

Paquetes: *tipos* y *tipos.test*

Record (en tipos): *Persona*

Test (en tipos.test): TestPersona01

Método main():

1. Crear un objeto *yo* de tipo *Persona* con tu: dni, nombre, apellidos y fecha de nacimiento:
2. Crear un objeto *miCompi* de tipo *Persona* con el: dni, nombre, apellidos y fecha de nacimiento de tu compañero/a
3. Visualizar *yo* y *miCompi*.
4. Visualizar tus *apellidos*
5. Modifica *Persona* e implementa *getEdad*(). Visualiza la edad de tu compañero/a

2023/24

Practicamos. Proyecto: T02_Tipos

Proyecto: *T02_Tipos*

Paquetes: *tipos* y *tipos.test*

Record (en tipos): *Animal*

Test (en tipos.test): *TestAnimal01*

Propiedades de *Animal*:

- *Familia* familia (puede tomar los valores TERRESTRE, AVE, MARINO o ANFIBIO hay que crear en el paquete tipo el enumerado *Familia*)
- *String* nombre
- *Double* pesoMedio
- *Integer* edadMedia
- *Boolean* puedeSerDoméstico

Implementar mediante record y probar