Introducción al diseño de tipos

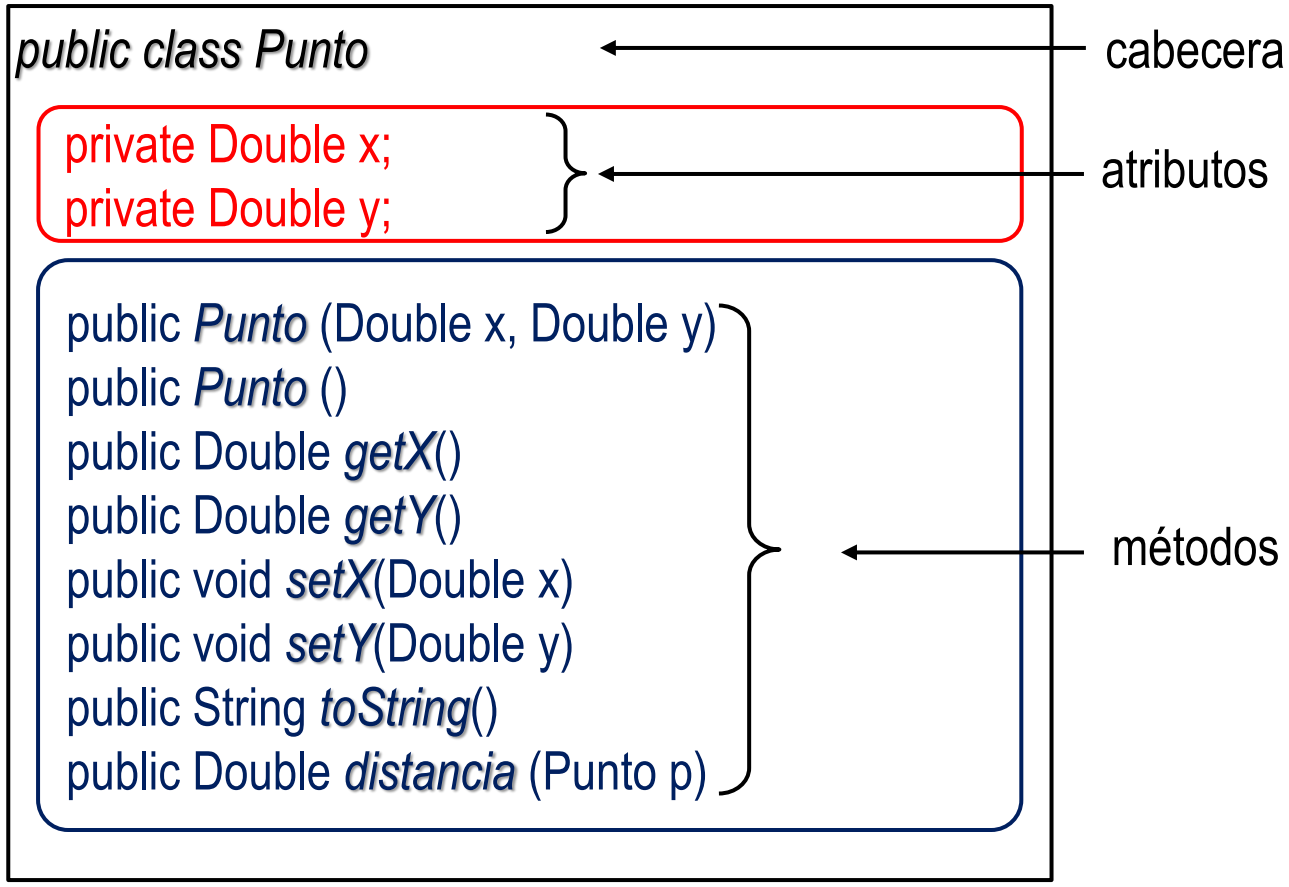
Objetos, clases, interfaces y record

Fundamentos de Programación
Departamento de Lenguajes y Sistemas Informáticos

2024/2025

Ejercicio. Tipo Punto

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



2024/25

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 p1

Paso 1.

TestPunto.java

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

2024/25

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 p1

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

2024/25

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

Se realizan las asignaciones

Ejercicio. Tipo Punto (Creación objetos)

2024/25

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

Paso 4.

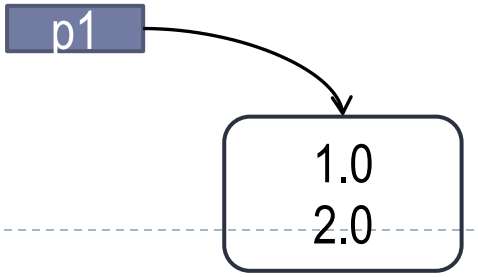
TestPunto.java

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

Punto.java

```
private Double x; 1.0
private Double y; 2d

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



2024/25

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("Ordena
da 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
```

Ejercicio. Tipo Punto

2024/25

INTENTAMOS HACER LO QUE HEMOS VISTO EN LAS DIAPOSTIVAS ANTERIORES CON *DEBUG* EN EL IDE *ECLIPSE*

Ejercicio. Tipo Circunferencia

CORREGIMOS EL EJERCICIO QUE QUEDÓ PROPUESTO EN LA ÚLTIMA SESIÓN

2024/25

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*.

2024/25

Ejercicio. Tipo Circunferencia

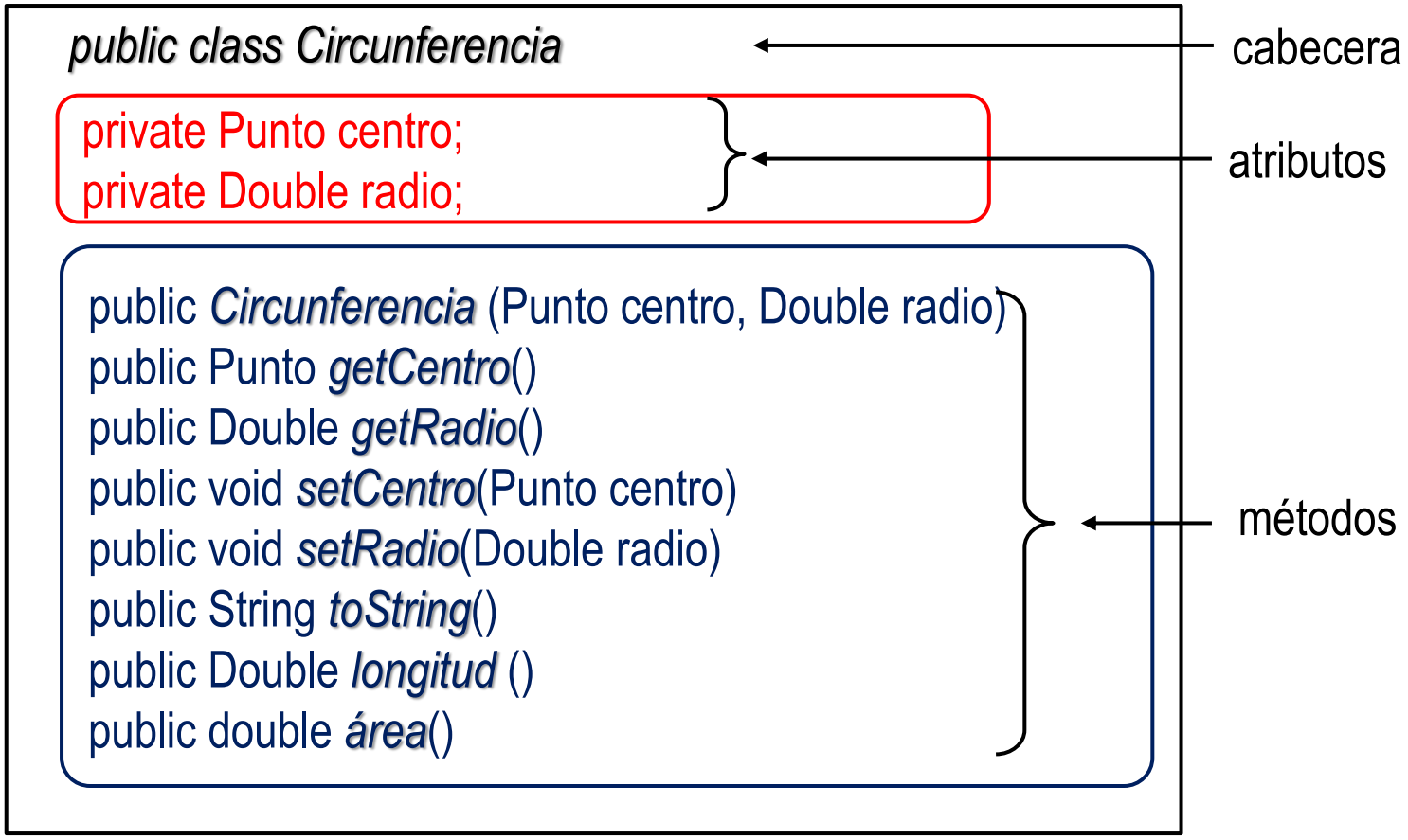
Métodos (*qué operaciones deseamos para nuestro tipo Circunferencia*):

- 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

Ejercicio. Tipo Circunferencia

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



2024/25

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

2024/25

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?

2024/25

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*.

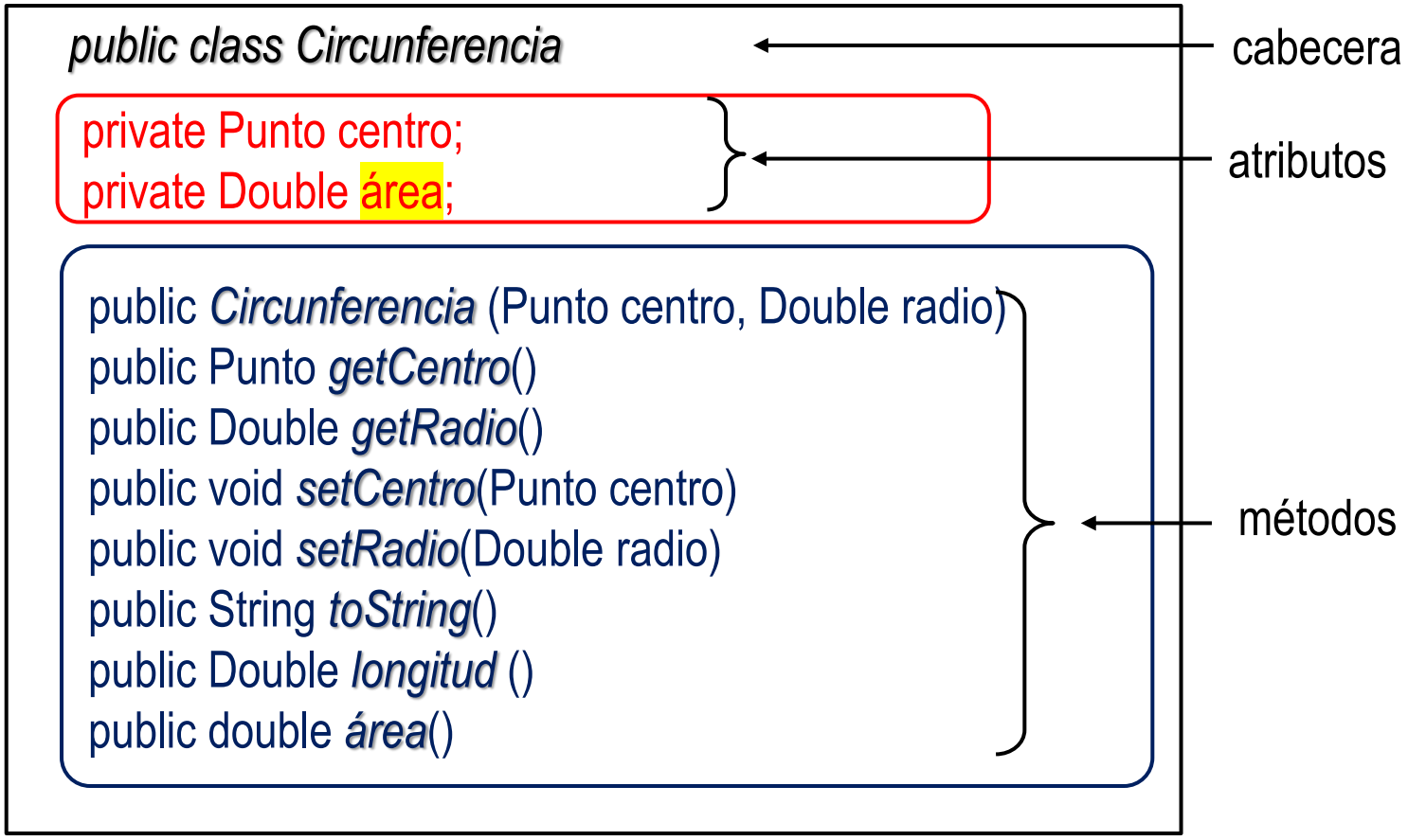
- Eso si, las cabeceras de todos los métodos no pueden cambiar:

  El constructor de Circunferencia debe seguir recibiendo el *centro* y el *radio*, pero *internamente, en la clase, los atributos* serán el *centro* y el *área*

2024/25

Ejercicio. Tipo Circunferencia: otra implementación

- Gráficamente ahora la clase *Circunferencia* sería algo así:



2024/25

Ejercicio. Tipo Circunferencia: otra implementación

Procedimiento para la nueva implementación:

Veamos como lo hacemos:

1. Vamos a renombrar la clase *Circunferencia* como *CircunferenciaImpl1*
2. *Copiemos y peguemos CircunferenciaImpl1* en el mismo paquete geometría y a la copia le denominamos *CircunferenciaImpl2*.
3. Vamos a modificar el testCircunferencia para cambiar el tipo *Circunferencia, por CircunferenciaImpl1* y pruébalo porque debe seguir funcionando
4. Ahora se trata de *modificar CircunferenciaImpl2* para cambiar el atributo *radio por área* y, sin modificar las cabeceras de los métodos, hacer también los *cambios en el código de los métodos* que lo necesiten para que sigan haciendo lo que ya hacían.

¡Hala! pues a Eclipse a cambiar los atributos y los métodos de *CircunferenciaImpl2*

2024/25

Ejercicio. Tipo Circunferencia: otra implementación

Procedimiento para la nueva implementación:

Ahora no encontramos con dos implementaciones de Circunferencia:

CircunferenciaImpl1	CircunferenciaImpl2

Para probar la nueva implementación dupliquemos la clase *TestCircunferencia* con el nombre *TestCircunferencia2* y cambiemos donde pone *Impl1* por *Impl2*. ¡Debe seguir funcionando!

2024/25

Ejercicio. Tipo Circunferencia: otra implementación más

CircunferenciaImpl3 y CircunferenciaImpl4:

Si te apetece existe una tercera y una cuarta implementación.

¿te atreves a pensarlo y realizarlas en casa?

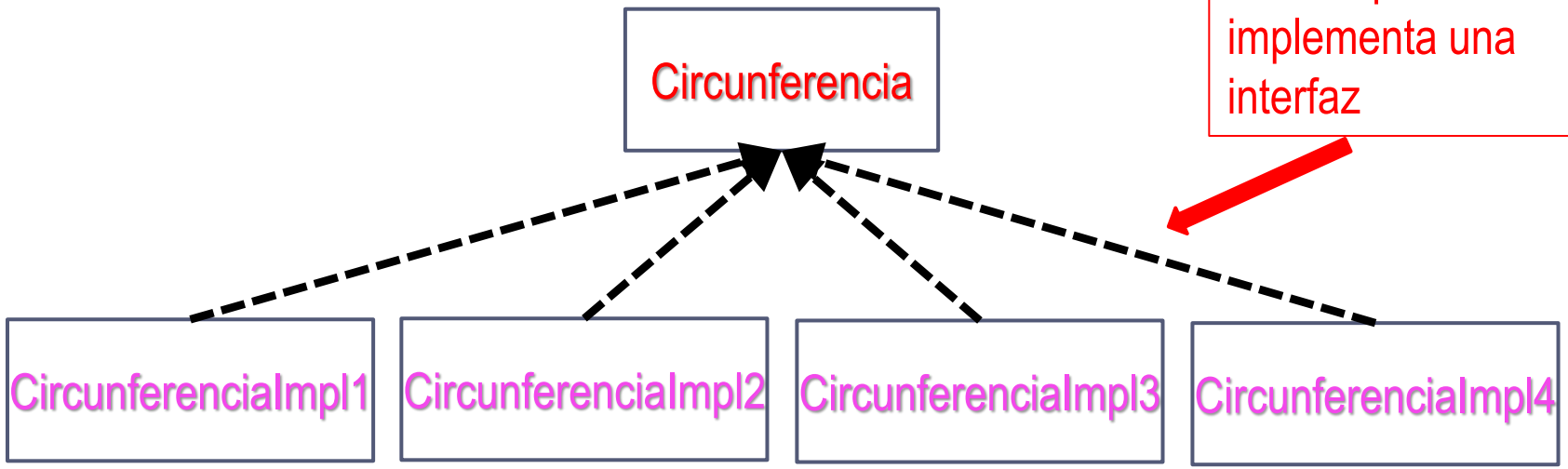
Interfaz

2024/25

- Cuando un objeto del mundo real se puede construir y manejar con las mismas funciones, pero con distintas implementaciones, surge la conveniencia de la implementación de lo que se denomina *Interfaz*.
- Una *interfaz* es una descripción de las funcionalidades de un tipo de objeto: se indica qué métodos tendrá el tipo (pero no se explica cómo internamente se construyen esos métodos).

Interfaz Coche:
- arrancarMotor()
- pararMotor()
- acelerar()
- frenar()
- cambioDeMarcha()
- abrirPuerta(número de la puerta)
- cerrarPuerta(número de la puerta)

Interfaz Circunferencia



- Tenemos dos implementaciones ¿o cuatro para los más atrevidos?. Vamos a definir la interfaz *Circunferencia* que nos permita manejar cualquier implementación de ella.

Interfaz Circunferencia

Interfaz Circunferencia:

- Dentro del paquete *geometría* defina la interfaz *Circunferencia*. Con las funcionalidades que se esperan de una circunferencia (métodos consultores y modificadores de las propiedades básicas y los otros métodos descritos. No incluir *toString()*)
- En las interfaces se escribe el *nombre* de cada método, precedido por su *tipo* y seguido de sus *parámetros*, evidentemente, entre paréntesis. (*ver imagen en la siguiente diapositiva*)

Interfaz

Sintaxis de construcción de una interfaz:

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

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

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.

2024/25

Resumen: Clases con o sin Interfaz

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 e Impl4 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!

Records

2024/25

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 ajustado a nuestras necesidades que oculta* al que tiene por defecto.

2024/25

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

2024/25

```
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 (*Constructor canónico*)

```
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: métodos *equals* y *hashCode* (hoy pasamos de estos métodos)

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

2024/24

Records (*Ejemplo*)

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

Se pueden implementar dentro del bloque del record, 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;
   }
}
```

2024/25

Practicamos. Proyecto: T02_Tipos

Proyecto: *T02_Tipos*

Paquetes: *fp.tipos* y *fp.tipos.test*

- Record (en fp.tipos): *Persona*
- Test (en fp.tipos.test): TestPersona01

Método main():

1. Crear un objeto *yo* de tipo *Persona* con tu: dni, nombre, apellidos y fecha de nacimiento -*recuerda LocalDate.of(año, mes, día)*-
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

2024/25

Creación de tipos mediante record

Inplementación de otros constructores en un record.

Hemos visto que una forma "minimalista" de crear un tipo es usando records, que proporciona un *constructor* que denominamos *canónico* que tiene un parámetro por cada atributo.

Pero ¿*y si queremos un constructor con otros parámetros*?. La solución es sencilla: Se construye un constructor en el *record* con los parámetros deseados que tiene *una primera línea* con la sentencia *this*(…). Es sentencia this(…), invoca al constructor canónico.

Ejemplo: Constructor de Persona con sólo apellidos y nombre, con el dni sin rellenar y la fecha de nacimiento en el día de hoy

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

   public Persona(String apellidos, String nombre) {
      this("", nombre, apellidos, LocalDate.now());
   }
   …
}
```

2024/25

Practicamos. Proyecto: T02_Tipos

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

2024/25

Practicamos en casa. Proyecto: T02_Tipos

Proyecto: *T02_Tipos*

Paquetes: *fp.tipos y fp.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 con un par de animales

Practicamos en casa. Proyecto: T02_Tipos

2024/25

Para el tipo *Animal* . Se trata ahora de *implementar también y probar dos nuevos métodos constructores:*

- Uno con solo el *nombre* y la *familia* y el resto de los atributos numéricos a cero y si el atributo puedeSerDoméstico a false.
- Otro con el *nombre*, *familia* y *puedeSerDoméstico* y el resto de los atributos a cero.

Haga un nuevo *TestAnimal02* en el paquete *fp.tipos.test* para construir dos objetos de tipo animal con los nuevos constructores y visualícelos.