Bloque 2

Introducción al diseño de tipos

Objetos, interfaces, clases

Fundamentos de Programación
Departamento de Lenguajes y Sistemas Informáticos

2023/2024

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

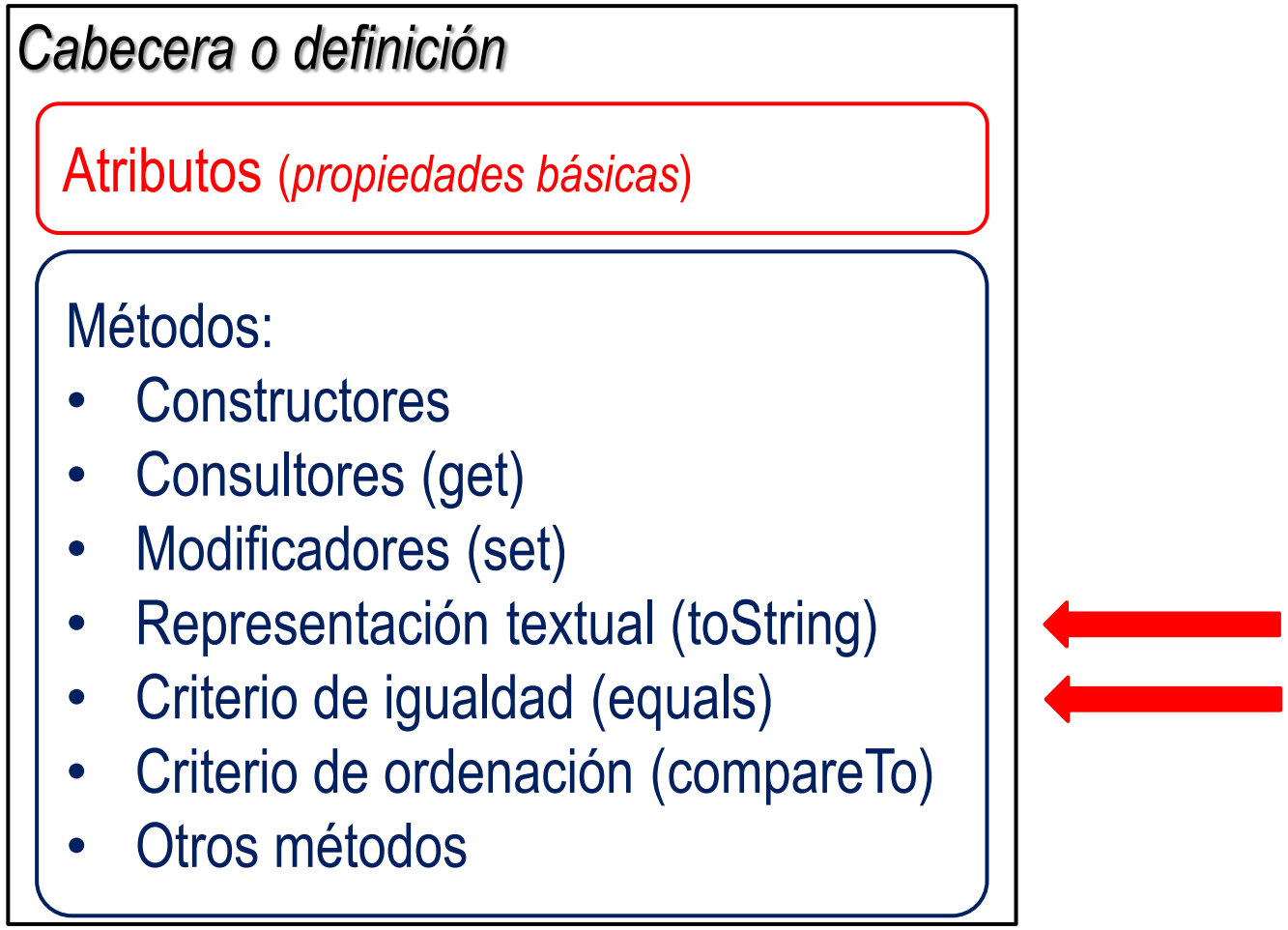
Implementar mediante record y probar

Ejercicio. Tipo Animal

Corregimos el ejercicio

Clases: Definición de un nuevo tipo

- Gráficamente una clase puede ser algo así:



Herencia

2023/24

- La herencia es una relación que se establece entre tipos.
- Se dice que un tipo **H**ijo hereda de otro tipo **P**adre si:
  - Un objeto de tipo **H** también **es un** objeto de tipo **P** (pero no al revés)
  - El objeto de tipo **H** tiene otras características propias que no tienen todos los objetos de tipo **P**

Tipo Hijo (subtipo) POLICÍA

…hereda de…

…es una…

…extiende a…

Tipo Padre (supertipo) PERSONA

En *Java* ocurre lo que en la *vida cotidiana:*

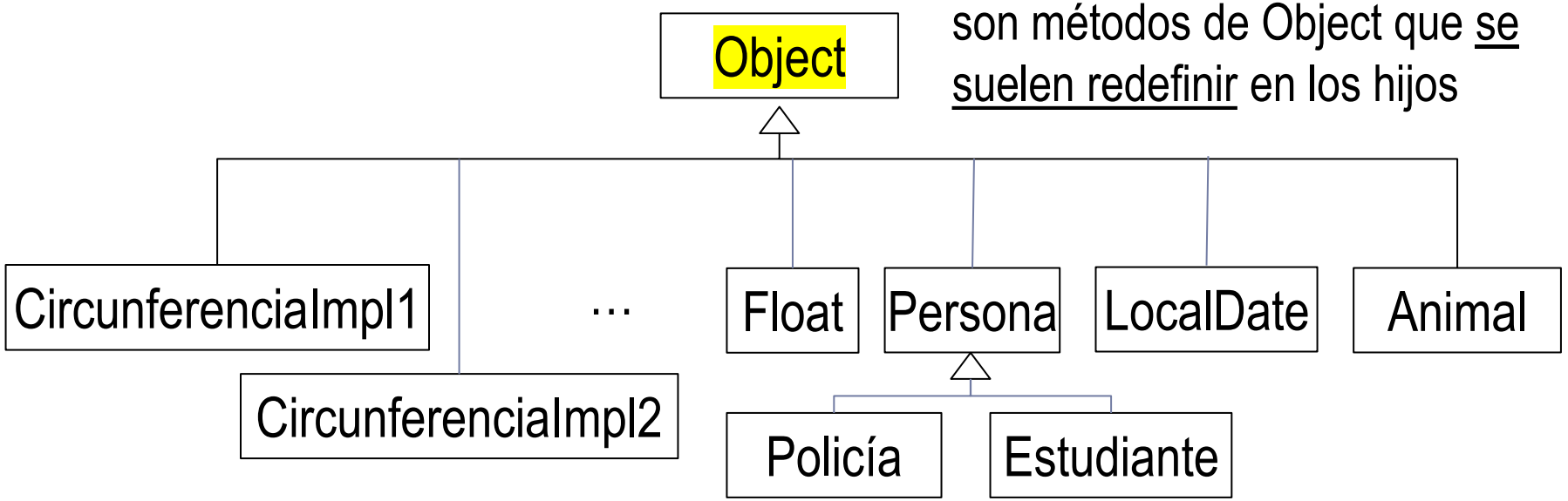
¡¡Todo lo del padre lo coge el hijo, pero el hijo no deja que su padre se meta en sus cosas!!

La clase Object

Todos los tipos objeto en Java heredan del tipo **Object**

- Object es el "padre invisible" de todos los objetos en Java.
- Integer, Double, Float, Character, String… heredan de *Object*, también CircunferenciaImpl1, CircunferenciaImpl2, Persona, Animal.

toString, *equals* y *hashCode* son métodos de Object que se suelen redefinir en los hijos



El tipo Object

2023/24

Object tiene una serie de métodos o propiedades, entre otros, se distinguen:

- String *toString()*; *(para convertir un objeto en texto)*
- boolean *equals(*Object o*)*; *(para ver la igualdad, que no la identidad)*
- int *hashCode()*; *(obtiene un "dni" único de cada objeto). Se calcula realizando una combinación lineal de los hashCode de las propiedades que intervienen en el método equals()*

Normalmente en todas las clases implementaremos estos tres métodos.

Si el tipo tiene interfaz no hay que definir en ella, porque ya los tiene definidos Object

El tipo Object (*toString del tipo Punto*)

2023/24

String *toString():* Ya lo implementamos para que se visualizara, por ejemplo: (1.0,2.0)

```
public String toString(){
   return "("+this.x+","+this.y+")";
}
```

El tipo Object (*toString del tipo Punto*)

2023/24

Si tengo creado un punto ***p1***

Test

```
Punto p1=new Punto(1d,2d);
System.out.println(p1.toString()); → (1.0,2.0)
System.out.println(p1);   → (1.0,2.0)
```

Clase

```
public String toString(){
    return "("+this.x+","+this.y+")";
}
```

El tipo Object (*equals del tipo Punto*)

2023/24

boolean *equals* (Object o): Elegimos que dos Puntos son iguales si tienen las mismas coordenadas.

```
public boolean equals(Object o) {
   boolean res=false;
   if (o instanceof Punto) {
      Punto p=(Punto)o;
      res=this.getX().equals(p.getX()) &&
          this.getY().equals(p.getY());
   }
   return res;
}
```

El tipo Object (*equals del tipo Punto*)

2023/24

Si tengo creado dos puntos *p1* y *p2*

Test

```
Punto p1=new Punto(1d,2d);
Punto p2=new Punto(1.0,2.0);
System.out.println(p1.equals(p2));   → true
```

Clase

```
public boolean equals(Object o) {
   boolean res=false;
   if (o instanceof Punto) {
      Punto p=(Punto)o;
      res=this.getX().equals(p.getX()) &&
          this.getY().equals(p.getY());
   }
   return res;
}
```

El tipo Object (*hashCode del tipo Punto*)

2023/24

int ***hashCode()***: Dado que equals utiliza la coordenada “x” y la coordenada “y” para establecer la igualdad hashCode tiene que usar las mismas propiedades. Se realiza una combinación lineal de los hashCode de “x” y de “y”, preferentemente con coeficientes que sean números primos.

```
public int hashCode() {
    return 11*this.getX().hashCode()+
           37*this.getY().hashCode();
}
```

El tipo Object (*hashCode del tipo Punto*)

2023/24

Si tengo creado un punto *p1*

Test

```
Punto p1=new Punto(1d,2d);
System.out.println(p1.hashCode());   → -11534336
```

Clase

```
public int hashCode() {
    return 11*this.getX().hashCode()+
           37*this.getY().hashCode();
}
```

2023/24

El tipo Object (*Ejercicios: Circunferencia*)

Para el tipo *Circunferencia* ya hicimos toString().

Se trata ahora de implementar y probar los métodos *equals()* y *hashCode()*

Decidimos que dos circunferencias son iguales si tienen el *mismo centro y radio*

2023/24

El tipo Object (*Ejercicios: Circunferencia*)

Para probar *equals()* y *hashCode()*

1. Cree dos circunferencias (c1 y c2) con el mismo centro y radio utilizando la implementación-1 para la primero y la implementación-2 para lal segunda.
2. Cree una tercera circunferencia (c3) con el mismo centro y radio distinto de las anteriores con la implementación que desee
3. Por último, cree una cuarta circunferencia (c4) con centro distinto y radio igual a las dos primeras con la implementación que desea.
4. Visualice los resultados de:
   - c1.*equals*(c2);
   - c1.*equals*(c3).
   - c2.*equals*(c4)y
   - c4.*equals*(p3)
5. Visualice los hashCode de las 4 circunferencias.

El tipo Object (*Ejercicio: Persona*)

2023/24

Para el tipo ***Persona*** . Se trata ahora de implementar y probar los métodos ***toString()***, ***equals()*** y ***hashCode()***

Decidimos que:

- La representación textual de una persona es su dni, apellidos y nombre separador por punto y coma y un blanco y entre paréntesis:

  (12345678A; García Gómez; Ana)

- Dos Personas son iguales si tienen el ***mismo dni, nombre y apellidos***

2023/24

El tipo Object (*Ejercicio: Persona*)

Para probar *toString()*, *equals()* y hashCode()

Cree tres personas y visualice los resultados de algunos que los métodos anteriores para las tres personas.

¡No seas vago y no visualices el mismo método para todas!

El tipo Object (*Ejercicio: Animal*)

2023/24

Para el tipo *Animal* . Se trata ahora de implementar y probar los métodos *toString()*, *equals()* y *hashCode()*

Decidimos que:

- La representación textual de un animal es su nombre seguido de la familia entre corchetes:

  Buitre[AVE]

- Dos animales son iguales si tienen el *mismo nombre y familia*

2023/24

El tipo Object (*Ejercicio: Animal*)

Para probar *toString()*, *equals()* y hashCode()

Cree tres animales y visualice los resultados de algunos que los métodos anteriores para los tres animales.

¡No seas vago y no visualices el mismo método para todos los animales!

Criterio de orden natural

2023/24

- El criterio de *orden natural* es el que *se elige* para establecer un *criterio de ordenación* entre objetos. Es decir, dados dos objetos, saber si uno de ellos es menor, mayor o igual que el otro o, de otra manera, quien va primero y quien va detrás.
- No todos los objetos tienen porqué tener criterio de orden natural.
- El método *int compareTo()* es el que *establecermos* el orden natural. Cuando tengamos, por ejemplo, dos puntos p1 y p2 y queramos saber cuál va primero y cual va detrás escribiremos:

```
p1.compareTo(p2);
```
→
- <0 si p1 < p2
- =0 si p1.= p2
- >0 si p1 > p2

2023/24

Criterio de orden natural

Procedimiento para implementar el orden natural

Java tiene definida la interfaz Comparable

```
public interface Comparable<T>{
        int compareTo(T obj);
}
```

Un tipo será comparable si implemente dicha interfaz. Es decir, en la cabecera de la *clase* o del *record* se escribirá:

```
public class Tipo implements Comparable<Tipo>{
        …
}

public record Tipo implements Comparable<Tipo>{
        …
}
```

Criterio de orden natural

2023/24

Procedimiento para implementar el orden natural

Sabemos que cuando un tipo implementa una interfaz obliga a programar los métodos que están definidos en ella. En consecuencia, en la *clase* o en *record* habrá que implementar el método *compareTo*

Una advertencia: Si el tipo ya implementa una interfaz (como pasa en nuestras circunferencias) la sintaxis es distinta: La interfaz es la *extiende* a Comparable<T>, pero el resultado es el mismo. En la clase hay que programar el método CompareTo().

```
public interface Tipo extends Comparable<Tipo>{
   …
}
```

Criterio de orden natural (Punto)

2023/24

Orden natural de Punto

Diremos que el criterio de *orden natural de Punto* será por la coordenada "x" y en caso de empate por la coordenada "y".

- El punto (1.0,2.0) será menor que (3.0, -1.5)
- El punto (1.0,2.0) será menor que (1.0, 4.5)
- El punto (2.0,2.0) será mayor que (1.0, 4.5)

```
public class Punto implements Comparable<Punto>{
   …
   public int compareTo(Punto p) {
      int res=this.getX().compareTo(p.getX());
      if (res==0) {
         res=this.getY().compareTo(p.getY());
      }
      return res;
   }
}
```

2023/24

Criterio de orden natural (Circunferencia)

Orden natural de Circunferencia

Diremos que el criterio de orden natural del tipo circunferencia es por la longitud del radio. Recuerda que nuestra implementación de Circunferencia tiene Interfaz

La circunferencia (1.0,4.5) R:2.5 es mayor que (1.0,4.5) R:2.0 y menor que (-1.0,7.5) R:1.25

¡¡Hala. A Eclipse a divertirse!!

2023/24

Criterio de orden natural (Persona)

Orden natural de Pesona

Diremos que el criterio de orden natural del tipo Persona es por el dni

¡¡Hala. A Eclipse a divertirse!!

Criterio de orden natural (Animal)

2023/24

Orden natural de Animal

Diremos que el criterio de orden natural del tipo Animal es por el pesoMedio y a igualdad de pesoMedio por edadMedia y a igualdad de edadMedia por la familia

¡¡Hala. A Eclipse a divertirse!!