Introducción al diseño de tipos

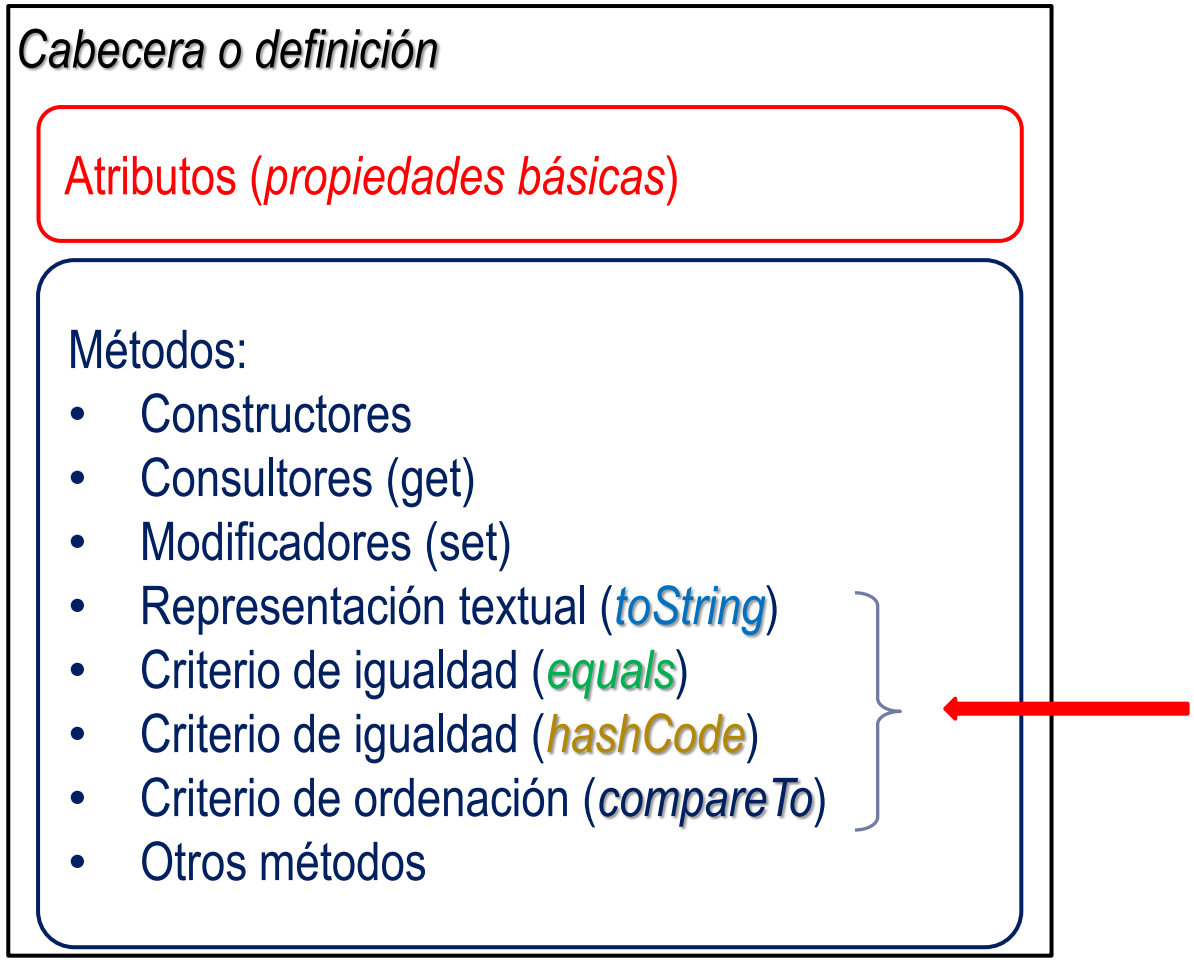
Herencia, tipo Object y Orden Natural

Fundamentos de Programación
Departamento de Lenguajes y Sistemas Informáticos

2024/2025

Clases: Definición de un nuevo tipo

- Ya hemos visto la imagen gráfica de una clase:



Herencia

2024/25

- 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!!

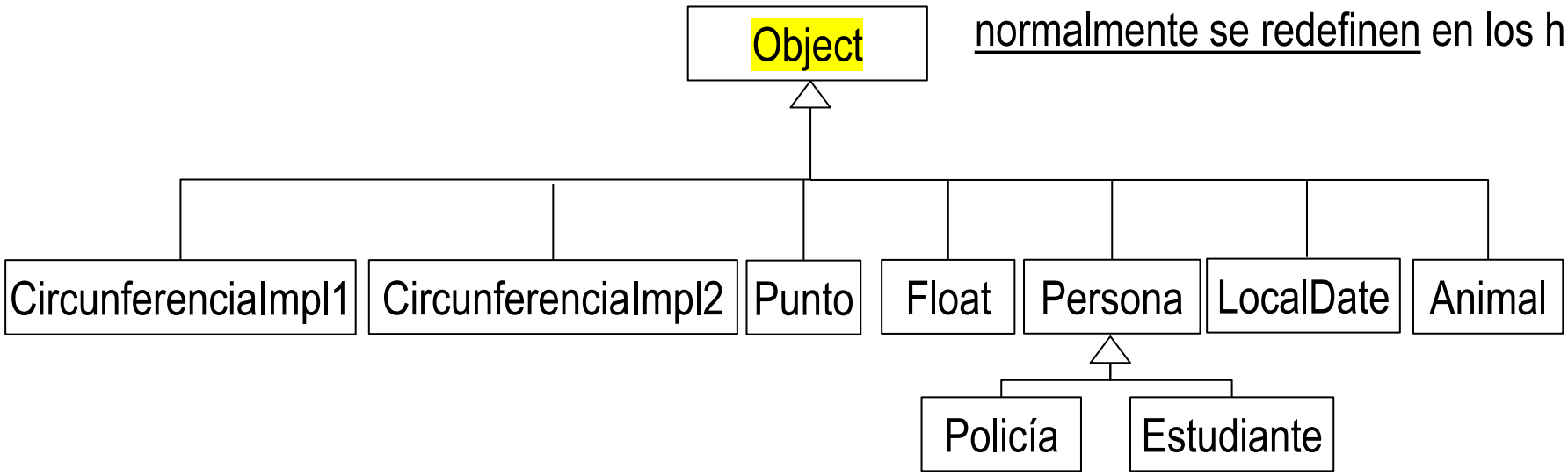
La clase Object

2024/25

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 normalmente se redefinen en los hijos



El tipo Object / La clase Object

2024/25

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

- String *toString()*; *(para convertir un objeto en una representación textual)*
- 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.

¡OJO!: Estos tres métodos no se escriben en las interfaces.

El tipo Object (*toString del tipo Punto*)

2024/25

Si se ha implementado el método toString() en la clase

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

Una vez creado un objeto "p1" podemos ver su representación escribiendo directamente el nombre del objeto "p1" o invocando al método toString()

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

Si no se implementa el método toString cuando se ejecute el test, se ejecutará el método tostring() de la clase Object.

Test
```
Punto p1=new Punto(1d,2d);
System.out.println(p1.toString()); → Punto@5674cd4d
System.out.println(p1);  → Punto@5674cd4d
```

El tipo Object (*equals del tipo Punto*)

2024/25

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

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;
}
```

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
```

El tipo Object (*equals del tipo Punto*)

2024/25

boolean *equals* (Object o): ¿Y si son objetos de distinto tipo?

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;
}
```

Si tengo creado dos objetos de tipos diferentes *p* y *a*

Test

```
Punto p=new Punto(1d,2d);
Animal a=new Animal("Gato",Familia.TERRESTRE);
System.out.println(p.equals(a));  → false
System.out.println(a.equals(p));  → false
```

El tipo Object (*hashCode del tipo Punto*)

2024/25

int *hashCode()*: Dado que equals utiliza la coordenada "x" y la coordenada "y" para establecer la igualdad, h*ashCode* 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.

Clase

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

Si tengo creado un punto *p1*

Test

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

2024/25

El tipo Object (*Ejercicios: Circunferencia*)

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

Ejercicio.-

Se trata ahora de implementar y probar los métodos *equals()* y *hashCode()* teniendo en cuenta que decidimos que dos circunferencias son iguales si tienen el *mismo centro y radio*

2024/25

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*(c3)
5. Visualice los hashCode de las 4 circunferencias.

El tipo Object (*Ejercicio: Persona*)

2024/25

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 encerrados entre paréntesis:

  (12345678A; García Gómez; Ana)

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

2024/25

El tipo Object (*Ejercicio: Persona*)

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

Haga un test en el que cree tres personas y visualice los resultados de algunos que los métodos anteriores para las tres personas.

El tipo Object (*Ejercicio: Animal*)

2024/25

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*

2024/25

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.

Criterio de orden natural

- 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

2024/25

Criterio de orden natural

Hacer comparable un tipo:

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 escribe ***implements***, pero si el tipo tiene intefaz entonces ***extends***:

```
public class Tipo implements Comparable<Tipo>{
        …
}
public record Tipo implements Comparable<Tipo>{
        …
}
public interface Tipo extends Comparable<Tipo>{
    …
}
```

2024/25

Criterio de orden natural (Punto)

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;
   }
}
```

2024/25

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:3.25

¡¡Hala. A Eclipse a divertirse!!

2024/25

Criterio de orden natural (Persona)

Orden natural de Persona

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

¡¡Hala. A Eclipse a divertirse!!

Criterio de orden natural (Animal)

2024/25

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 el orden alfabético de la familia

¡¡Hala. A Eclipse a divertirse!!