

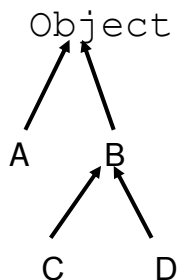
6. Herencia. Clases abstractas. Genericidad

- Clase `Object`
- Polimorfismo y ligadura dinámica
- `super`
- Herencia y creación
- Clases abstractas
- Genericidad

1

Herencia en Java

- **Herencia simple**
- **Object** es la clase raíz (paquete `java.lang`)
- **Object** describe las propiedades comunes a todos los objetos



Terminología:

- C y D son **subclases** de B
- B es la **superclase** de C y D

Herencia en Java

- Una subclase **hereda** de su superclase **métodos** y **variables** tanto de clase como de instancia.
- Una subclase puede **añadir** nuevos métodos y variables.
- Una subclase puede **redefinir** métodos heredados.

- **Sintaxis:**

```
class nombreHija extends nombrePadre {...}
```

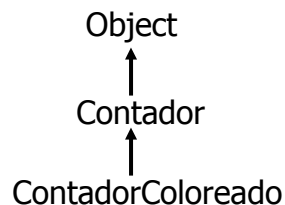
- Por defecto toda clase extiende la **clase Object**

```
class Contador {...} ⇔  
class Contador extends Object {...}
```

Ejemplo

```
public class ContadorColoreado extends Contador {  
    //nueva variable de instancia  
    private String color ;  
  
    public ContadorColoreado() {  
        super();  
        setColor("transparente");  
    }  
    public ContadorColoreado(int v, int s, String c) {  
        super(v,s);  
        setColor(c);  
    }  
    ...  
}
```

Llamada al constructor del padre



Acceso protegido

- Una subclase no puede acceder a los campos privados de la superclase
- Para permitir que un método de la subclase pueda acceder a un campo de la superclase, éste tiene que declararse como **protected**
- **Protected:** miembros visibles a las subclases y al resto de clases del paquete
- Resumen de modificadores de acceso:

- private -visible sólo en la clase
- public -visible a todas las clases
- protected -visible en el paquete y las subclases
- Sin modificador -visible en el paquete

El lenguaje de programación Java

5

Conversión de tipos

- Java es un lenguaje **fuertemente tipado**, es decir, en tiempo de compilación se comprueba la compatibilidad de tipos

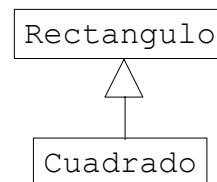
- **Conversión implícita:** (automática)

- *tipos primitivos* a uno que soporte un rango mayor de valores

```
float saldo = 300;    //podemos asignarle un entero
int codigo = 3.7;    //ERROR
```
- *conversión de referencias:* todo objeto es compatible con sus superclases

- **cast-up**
- siempre válido

```
Rectangulo r;
Cuadrado c = new Cuadrado();
r = c;
```



6

Conversión de tipos

- **Conversión explícita:**

- *tipos primitivos* perdiendo información

```
long l = 200;  
int i = (int)l;
```

- *conversión de referencias* asignar a un objeto de una subclase uno de la superclase

- *cast-down* o *narrowing*
- No siempre válido
- El error se puede producir:
 - en tiempo de ejecución (**ClassCastException**)
 - en tiempo de compilación si no es ni siquiera una subclase.

7

Conversión explícita de referencias

- Puede dar un error en ejecución:

```
Figura [] figuras = new Figura [30];  
...  
Rectangulo r = (Rectangulo)figuras[i];
```

- Comprobar el tipo antes de la conversión:

```
if (figuras[i] instanceof Rectangulo)  
    r = (Rectangulo)figuras[i]
```

- Daría error en compilación:

```
Cuenta c = (Cuenta)figuras[i];
```

Redefinición

- Un **método** de la subclase con la **misma signatura** y valor de retorno que un método de la superclase lo está **REDEFINIENDO**.
- Para evitar la redefinición de un método se utiliza el modificador `final`.
- Puede **cambiar el valor de acceso** siempre que lo relaje, es decir, puede pasar de `protected` a `public` pero no a `private`.
- Los **atributos** no se pueden redefinir, sólo se **OCULTAN** (el campo de la superclase todavía existe pero no se puede acceder)

El lenguaje de programación Java

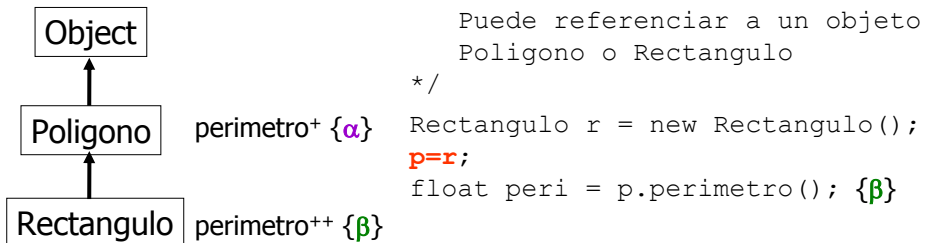
9

```
public class Poligono{
    private Punto [] vertices;
    private int numVertices;
    ...
    public float perimetro(){ //{{a}}
        double peri=0;
        Punto anterior=vertices[0];
        Punto actual;
        int ultimo=numVertices-1;
        for (int i=1; i<=ultimo; i++){
            actual=vertices[i];
            peri+=anterior.distancia(actual);
            anterior=actual;
        }
        //distancia del ultimo con el primero
        return peri+=vertices[ultimo].distancia(vertices[0]);
    }
}

public class Rectangulo{ //{{β}}
    private double lado1;
    private double lado2; ← Añade atributos
    ...
    public float perimetro(){ ← Redefine métodos
        return 2*lado1+2*lado2;
    }
}
```

Polimorfismo y ligadura dinámica

- **Polimorfismo**: una entidad puede hacer referencia a objetos de diferentes tipos en tiempo de ejecución.
- **Ligadura dinámica**: en tiempo de ejecución se elegirá la versión mas adecuada según el tipo del objeto receptor.
- Sea la jerarquía:



```
Poligono p;  
/*  
    Puede referenciar a un objeto  
    Poligono o Rectangulo  
*/  
Rectangulo r = new Rectangulo();  
p=r;  
float peri = p.perimetro(); { $\beta$ }
```

El lenguaje de programación Java

11

super

- Palabra clave disponible en todos los métodos no-static
- Se invoca a la versión del método de la primera superclase que lo contenga
- Ejemplo:

```
public class Punto{  
    ... //(x,y)  
    public void borrar(){}  
}  
  
public class Pixel extends Punto{  
    ...  
    public void borrar(){  
        super.borrar(); //borra el Punto  
        color = null;  
    }  
}
```

El lenguaje de programación Java

12

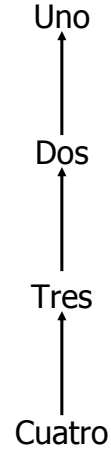
Ejemplo super

```
class Uno {
    public int test(){return 1;}
    public int result1(){return this.test();}
}

class Dos extends Uno{
    public int test(){return 2;}
}

class Tres extends Dos{
    public int result2(){return this.result1();}
    public int result3(){return super.test();}
}

class Cuatro extends Tres{
    public int test(){return 4;}
}
```



Herencia. 'super'

```
public class PruebaSuperThis{
    public static void main (String args[]){
        Uno ob1 = new Uno();
        Dos ob2 = new Dos();
        Tres ob3 = new Tres();
        Cuatro ob4 = new Cuatro();

        System.out.println("ob1.test = "+ ob1.test());
        System.out.println("ob1.result1 = " + ob1.result1());
        System.out.println("ob2.test = " + ob2.test());
        System.out.println("ob2.result1 = " + ob2.result1());
        System.out.println("ob3.test = " + ob3.test());
        System.out.println("ob4.result1 = " + ob4.result1());
        System.out.println("ob3.result2 = " + ob3.result2());
        System.out.println("ob4.result2 = " + ob4.result2());
        System.out.println("ob3.result3 = " + ob3.result3());
        System.out.println("ob4.result3 = " + ob4.result3());
    }
}
```



Clases abstractas

- Se fija un conjunto de métodos y atributos que permitan modelar un cierto concepto, que será refinado mediante la herencia.
- **Métodos abstractos:**
 - sólo cuentan con la declaración y no poseen cuerpo de definición
 - la implementación es específica de cada subclase
- Toda clase que contenga algún método abstracto (heredado o no) es abstracta. Puede tener también métodos efectivos.
- Tiene que derivarse obligatoriamente
- **NO** se puede hacer un **new** de una clase abstracta. **SI** deben definir los **constructores**.

El lenguaje de programación Java

15

Ejemplos

- ```
public abstract class Figura {
 public abstract void dibujar();
 public abstract void rotar();
 ...
}
public class Rectangulo extends Figura {
 public void dibujar(){
 //código específico para dibujar rectángulos
 ...
 }
 ...
}
```
- **Clases abstractas en Java: Number y Dictionary**

16



## Herencia. Clase Object

- Todas las clases heredan directa o indirectamente de la **clase Object**, raíz de la jerarquía.
- Toda clase tiene disponibles sus métodos:
  - `public final Class getClass()` → clase que representa el tipo del objeto
  - `public boolean equals(Object obj)` → igualdad de valores
  - `public String toString()` → Devuelve la representación del obj en un String
  - `protected Object clone()` → devuelve una copia del objeto
  - `public int hashCode()` → devuelve un código que identifica de manera única al objeto
  - `protected void finalize()` → relacionado con liberar memoria
- Hay que redefinir `equals`, `toString`, `hashCode` y `clone` para adaptarlos.

El lenguaje de programación Java

17

## Método getClass. Clase Class

- Sirve para identificar el tipo de los objetos en tiempo de ejecución
- Describe las propiedades de la clase de un objeto
- **String getName()**: devuelve el nombre de la clase

- Ejemplo:

```
Empleado e; ...
System.out.println(e.getClass().getName()
 + e.getNombre());
```

**SALIDA:** Empleado Pedro Martínez

El lenguaje de programación Java

18

## getClass VS instanceof

- instanceof  
“¿Eres de esta clase o de una clase derivada de ésta?”
- comparando los objetos Class  
“¿Eres exactamente de esta clase?”
- Ejemplo: Sea Rectangulo una subclase de la clase Figura

```
Rectangulo r = new Rectangulo();
(r instanceof Figura) → true
(r.getClass().equals(Figura.class)) → false
```

## Igualdad en Java

- Igualdad de referencias (*identidad*):

```
objPila1 == objPila2 --> false
objPila1 != objPila2 --> true
```

- Método equals
  - Disponible para todo objeto
  - **public boolean equals(Object obj)**
  - Comportamiento por defecto: `this==obj`
  - Utilizado para implementar la igualdad de objetos.

## Método equals

- Propiedades del método equals:
  - *reflexivo*: `x.equals(x) = true` para todo `x != null`
  - *simétrico*: `x.equals(y) = y.equals(x)`
  - *transitivo*: Si `x.equals(y) AND y.equals(z)`  
ENTONCES `x.equals(z)`
  - *consistente*: Llamadas repetidas a `x.equals(y)` debe devolver el mismo valor si los objetos a los que referencian `x` e `y` no han cambiado
  - `x.equals(null)` debe devolver `false`.

## Ejemplo de redefinición de equals

```
public class Punto {
 // . . .
 public boolean equals(Object otroObjeto) {
 // 1° comprobar si son idénticos
 if (this == otroObjeto) return true;
 // 2° si el argumento es null debe devolver falso
 if (otroObjeto == null) return false;
 // 3° Si son de clases distintas no pueden ser iguales
 if (getClass() != otroObjeto.getClass()) return false;
 // ahora sabemos que otherObject es un Punto no nulo
 Punto otroPunto = (Punto)otroObjeto;
 //Comprobamos si los campos tienen valores idénticos
 return x == otroPunto.x && y == otroPunto.y;
 }
}
```

## Método toString

- Devuelve un String con la información del estado actual del objeto
- Casi todas las clases deberán redefinir este método
- Son equivalentes:
  - `System.out.println(punto);`
  - `System.out.println(punto.toString());`
- Al concatenar con el operador “+” automáticamente se invoca al método toString
  - `System.out.println("Posición actual "+punto);`  
`//llama a punto.toString()`

## Ejemplo redefinición toString clase Punto

- Formato: nombreClase [atributo1=valor1, ...]
- a) Escribir el nombre de la clase:
- ```
public String toString(){
    return "Punto [x="+x
        +", y="+y
        +"]";
}
```
- b) Obtener el nombre de la clase automáticamente:
- ```
public String toString(){
 return getClass().getName()
 +" [x="+x
 +", y="+y
 +"]";
}
```

## toString y las subclases

- La opción b) hace que el `toString` funcione para las subclases

- Ejemplo:

```
class Punto3D extends Punto{
 //...
 public String toString(){
 return super.toString()
 + "[z="+z
 + "]";
 }
}
```

- Salida:

```
Punto3D [x=0,y=0][z=0]
```

## Genericidad en Java

- Hasta la versión 1.4 no existía un mecanismo de genericidad en el lenguaje al estilo de Eiffel: `Array[G]`
- Se puede definir una estructura de datos de tipo `Object`, puesto que todo tipo es compatible con la raíz
- **Inserción:**
  - Puedo insertar cualquier tipo de objetos
  - El control lo debe implementar el programador
- **Extracción:**
  - Recupero elementos de tipo `Object`
  - Hace falta hacer una conversión explícita

## Ejemplo: Clase Pila

```
public class Pila{
 Object [] contenido;
 ...
 public void push (Object obj){...}
 public Object pop(){...}
}
public class TestPila{
 public static void main(String [] args){
 Pila p = new Pila();
 Cuenta cta = new Cuenta("Titular",600.0);
 Cuenta cta2;
 Rectangulo r1 = new Rectangulo (10,20);
 Rectangulo r2 = new Rectangulo (10,20);
 p.push(r1);
 p.push(cta);
 p.push(r2);
 cta2 =(Cuenta) p.pop(); //OK! r compatible con Object
 cta2= p.pop(); // Error tej ClassCastException
 int d=(p.pop()).getDiagonal(); //Error tc tipos incompatibles
 //no existe el método getDiagonal en la clase Object
 }
}
```

El lenguaje de programación Java

27

## Clases envoltentes

- Todos los tipos primitivos tienen tipos de envoltura que los representan:

| Tipo básico | Clase envoltente |
|-------------|------------------|
| byte        | Byte             |
| short       | Short            |
| int         | Integer          |
| long        | Long             |
| float       | Float            |
| double      | Double           |
| boolean     | Boolean          |
| char        | Character        |

El lenguaje de programación Java

28

## Clases envolventes

- Funciones:
  - a) Crear un objeto que almacene el valor de un tipo primitivo para poder utilizarlo en una **estructura de datos genérica** (sólo maneja referencias a `Object`)
  - b) Recoge las operaciones y características relacionadas con un tipo. Por ejemplo, constantes que tienen información del intervalo máximo de valores, conversión en cadenas de texto.

## Ejemplo: clases envolventes

```
Object [] tabla = new Object [10];
...
for (int i=0; i<10; i++){
 tabla[i]=new Integer(i);
}
```

- También existen constructores a partir de cadenas de caracteres

```
Integer objInt = new Integer("345");
```

- Para obtener el valor:

```
int i = (Integer)tabla[i].intValue();
```

# Conversión de tipos

| Tipo    | A String                | De String                          |
|---------|-------------------------|------------------------------------|
| boolean | String.valueOf(boolean) | new Boolean(String).booleanValue() |
| int     | String.valueOf(int)     | Integer.parseInt(String)           |
| long    | String.valueOf(long)    | Long.parseLong(String)             |
| float   | String.valueOf(float)   | Float.parseFloat(String)           |
| double  | String.valueOf(double)  | Double.parseDouble(String)         |

- Forma más habitual para convertir un tipo primitivo en un string:

```
String entero = ""+56;
```

- Excepción `NumberFormatException` cuando la conversión de tipos no es válida.