

Paso de parámetros a métodos

Paso de tipos primitivos

- Al pasar un tipo primitivo a un método, estamos pasando una copia del dato.
- Si el método modifica su copia, no afecta al original

```
class Test{  
    public static void main(String[] ar){  
        Calc cl=new Calc();  
        int n=5;  
        cl.modif(n);  
        System.out.println(n); //5  
    }  
}
```

```
class Calc{  
    public void modif(int a){  
        a=a+3;  
    }  
}
```

n 5

a 5

n 5

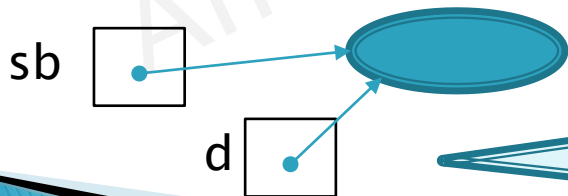
a 8

Paso de tipos objeto

➤ Al pasar un tipo objeto a un método, estamos pasando una copia de la referencia al objeto.

```
class Test{  
    public static void main(String[] ar){  
        Calc cl=new Calc();  
        StringBuilder sb=new StringBuilder("hello");  
        cl.modif(sb);  
        System.out.println(sb.toString()); //hello bye  
    }  
}
```

```
class Calc{  
    public int modif(StringBuilder d){  
        d.append(" bye");  
    }  
}
```



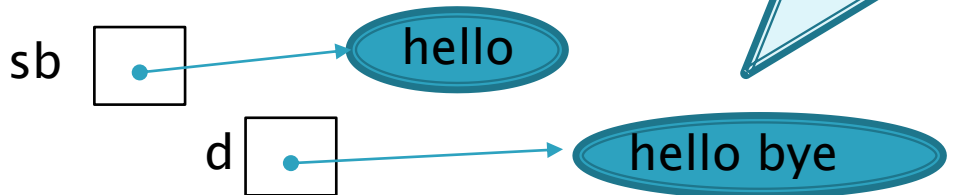
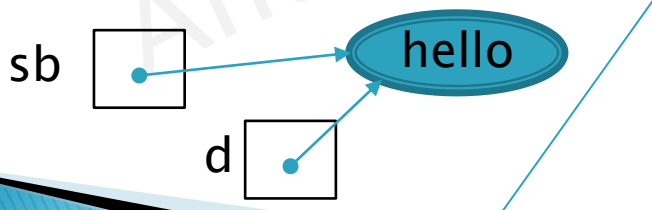
Ambas variables apuntan al mismo objeto

Paso de String

➤ Recuerda que las cadenas, aunque son objetos, son inmutables.

```
class Test{  
    public static void main(String[] ar){  
        Calc cl=new Calc();  
        String sb=new String("hello");  
        cl.modif(sb);  
        System.out.println(sb); //hello  
    }  
}
```

```
class Calc{  
    public int modif(String d){  
        d+=" bye";  
    }  
}
```



Aunque inicialmente apuntan al mismo objeto, tras la concatenación se genera un nuevo objeto al que apunta la variable parámetro

Sobrecarga

Sobrecarga de métodos

- Una clase puede contener varios métodos con el mismo nombre, pero deben diferenciarse en el número o tipo de parámetros:

```
public int sumar(int a, int b){..}  
public int sumar(int a){..}  
public int sumar(long b){..}  
public int sumar(int ...x){..}
```

- El tipo de devolución no afecta en la sobrecarga, puede ser el mismo o diferente

Llamadas a métodos sobrecargados

➤ La versión del método que será llamado se determina en función de los argumentos de la llamada:

sumar(3,9); —————> public int sumar(int a, int b){..}

sumar(10); —————> public int sumar(int a){..}

sumar(7L); —————> public int sumar(long b){..}

Precaución

➤ Cuando hay varios posibles métodos que se pueden ejecutar en una llamada: primero se intenta coincidencia exacta, después promoción de tipos y en último lugar autoboxing

metodo(4); —————> void metodo(int a)
void metodo(Integer e);

metodo(4); —————> void metodo(long a)
void metodo(Integer e);

metodo(4); —————> void metodo(Long a)
void metodo(Integer e);

Constructores

Constructor por defecto

- Si no se define un constructor de forma explícita en una clase, el compilador añade el llamado constructor por defecto, que no tiene parámetros y tampoco ninguna instrucción:

```
class Test{  
}
```



```
class Test{  
    public Test(){}  
}
```

```
Test t=new Test(); //ok
```

- Si se define explícitamente un constructor, el compilador ya no crea el constructor por defecto:


```
class Test{  
    public Test(int m){}  
}
```

```
Test t=new Test(); //error compilación
```

Llamadas a otro constructor

- Desde un constructor se puede llamar a otro constructor de la misma clase utilizando la expresión `this(argumentos)`.
- Debe ser la primera instrucción del constructor:

```
class Test{  
    public Test(){  
        this(5); //ok  
    }  
    public Test(int a){}  
    public Test(int a, int b){  
        int s=a+b;  
        this(s); //error compilación  
    }  
}
```



Bloque de inicialización instancia

➤ Son bloque de código que se ejecutan cada vez que se crea un objeto de la clase, antes del constructor.

➤ Se delimitan por llaves

```
class Test{  
    {  
        System.out.println("bloque");  
    }  
    public Test(){  
        System.out.println("constructor");  
    }  
}
```

```
class Prueba{  
    public static void main(String[] ar){  
        Test t1=new Test();  
        Test t2=new Test();  
    }  
}
```

Se imprimirá:
bloque
constructor
bloque
constructor

Bloque estáticos

- Se ejecutan una vez durante la vida de una clase.
- Solo puede acceder a otros miembros estáticos

```
class Test{  
    static int n=0;  
    static{  
        n++;  
    }  
    public int getN(){return n;}  
}
```

```
class Prueba{  
    public static void main(String[] ar){  
        Test t1=new Test();  
        Test t2=new Test();  
        System.out.println(t1.getN()); //1  
        System.out.println(t2.getN()); //1  
    }  
}
```

Sobrescritura

Reglas sobrescritura

➤ A la hora de sobrescribir un método, se deben seguir las siguientes reglas:

- El nombre y lista de parámetros debe ser idéntico
- El ámbito debe ser igual o menos restrictivo
- El tipo de devolución debe ser igual o un subtipo del original
- La nueva versión del método no debe propagar excepciones que no estén definidas en el original (esta restricción NO afecta a las excepciones Runtime)

Ejemplos sobreescripción correcta

```
class Clase1{  
    public Object test(){ }  
}  
class Clase2 extends Clase1{  
    @Override  
    public String test(){ }  
}
```

Tipo devolución
subclase de
Object

Ámbito
superior a
(default)

```
class Clase1{  
    void test(){ }  
}  
class Clase2 extends Clase1{  
    @Override  
    public void test(){ }  
}
```

```
class Clase1{  
    void test() throws IOException{ }  
}  
class Clase2 extends Clase1{  
    @Override  
    public void test() throws FileNotFoundException{ }  
}
```

Subtipo de
IOException

Ejemplos sobrescritura incorrecta

```
class Clase1{  
    public void test(){ }  
}  
class Clase2 extends Clase1{  
    @Override  
    public String test(){ } //error compilación  
}
```

El tipo
devolución
debería ser void

Ámbito inferior
a public

```
class Clase1{  
    public void test(){ }  
}  
class Clase2 extends Clase1{  
    @Override  
    void test(){ } //error compilación.  
}
```

```
class Clase1{  
    void test(){ }  
}  
class Clase2 extends Clase1{  
    @Override  
    public void test() throws SQLException{ } //error compilación.  
}
```

Excepción no declarada
en la superclase