



Escuela de Ingeniería Informática



Unidad 6: Comportamiento más sofisticado



Introducción a la Programación

Curso 2015-2016

Repaso ...

□ Bucle for-each

- Procesa **todos** los elementos de una colección
- Se puede aplicar a colecciones de **tamaño fijo o flexible**

□ Bucle for

- Permite procesar **total o parcialmente** una colección
- Se puede usar con colecciones de **tamaño fijo o flexible**
- Se puede usar para **repetir** la ejecución de un bloque de sentencias un número fijo de veces. En este caso, se usa el bucle for sin colecciones.

Repaso ...

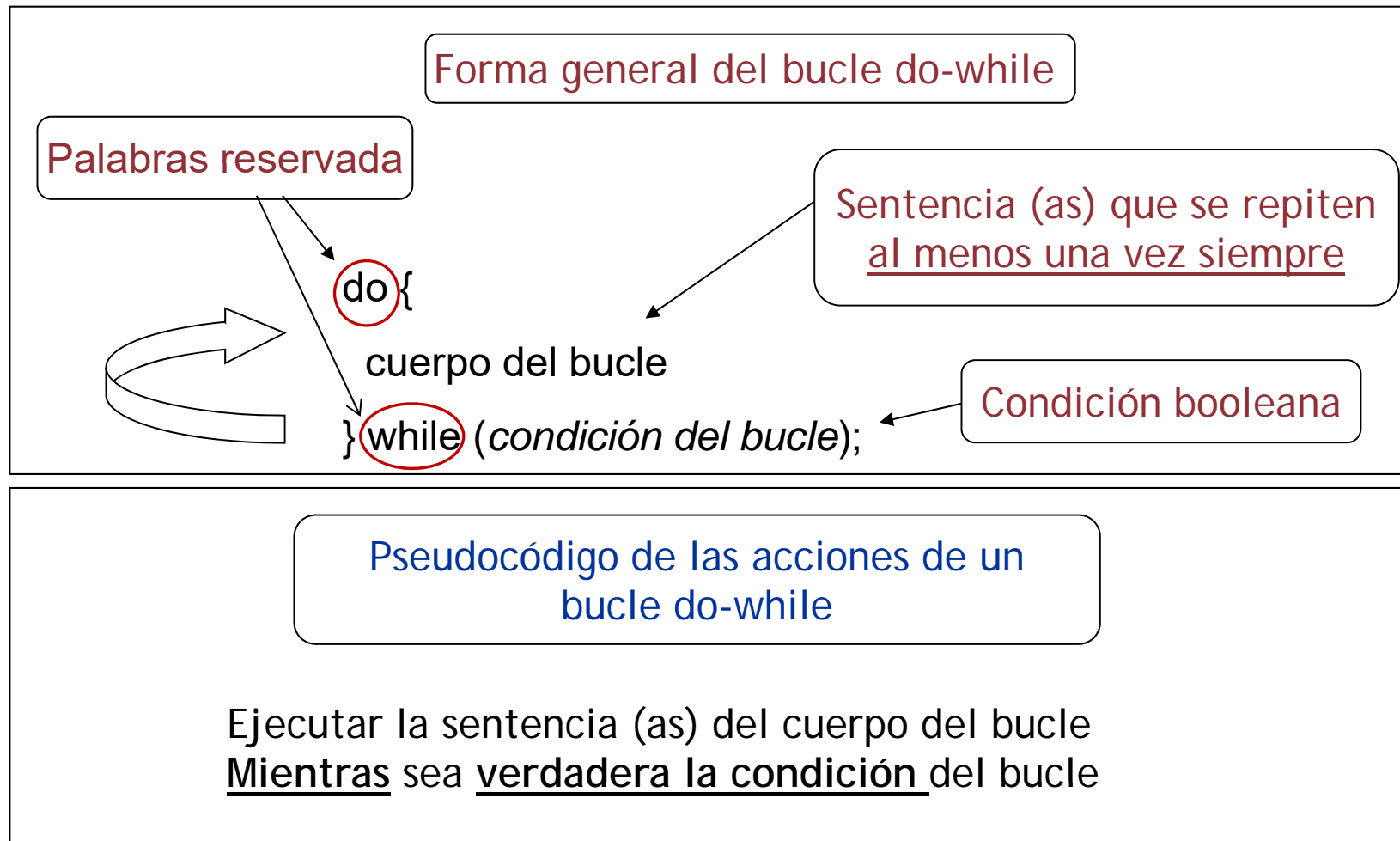
□ Bucle while

- Permite procesar **total o parcialmente** una colección
- Se puede usar con colecciones de **tamaño fijo o flexible**
- Se puede usar para **repetir** la ejecución de un bloque de sentencias un número fijo de veces. En este caso, se usa el bucle while sin colecciones.

□ Iterador (usar un objeto `Iterator`)

- Permite procesar **total o parcialmente** una colección
- Disponible para **todas las colecciones** de las clases **de las bibliotecas de Java**
- A menudo se usa con colecciones donde el **acceso indexado no es muy eficiente o es imposible**.

Bucle do-while



Bucle do-while

□ El Bucle do-while

- Permite procesar **total o parcialmente** una colección
- Se puede usar con colecciones de **tamaño fijo o flexible**
- Se puede usar para **repetir** la ejecución de un bloque de sentencias un número fijo de veces. En este caso, se usa el bucle do-while sin colecciones.

Sentencia switch-case

Forma general de la sentencia switch-case

Palabras reservada

```
switch ( expresión ) {  
    case ABC:  
        sentencia(as);  
        /* sigue la ejecución */  
    case DEF:  
        sentencia(as);  
        break;  
    case XYZ:  
        sentencia(as);  
        break;  
    default:  
        sentencia(as);  
        break;  
}
```

Sentencia multialternativa. La sentencia *break*, detiene la ejecución y salta a la siguiente sentencia que sigue a switch-case

Ejemplo switch-case

```
public void miniCalculadora (int a, int b, char op)
{
    System.out.print("El resultado es : ");
    switch ( op ) {
        case '+':
            System.out.println( a + b );
            break;
        case '-':
            System.out.println( a - b );
            break;
        case '*':
            System.out.println( a * b );
            break;
        case '/':
            System.out.println( a / b );
            break;
        default:    //default es opcional
            System.out.println("error" );
            break;   //break es opcional
    }
}
```

Ejemplo switch-case

```
public void cambiarPosicion(char orientacion)
{
    switch (orientacion) {
        case 'n':case 'N':
            posY=posY-1;
            break;
        case 'e':case 'E':
            posX=posX+1;
            break;
        case 's':case 'S':
            posY=posY+1;
            break;
        case 'o':case 'O':
            posX=posX+1;
            break;
        default:
            posX=posX+1;
            posY=posY+1;
    }
}
```


Arrays bidimensionales (Matrices)

UNIDIMENSIONALES:

```
tipo[] nombre_array = new tipo[tamaño];
```

```
tipo[] nombre_array = {v1, v2, v3, ...};
```

BIDIMENSIONALES:

```
tipo[][] nombre_array = new tipo[TFila][TCol];
```

```
tipo[][] nombre_array = {{v1, v2, ...}, ...};
```

- En **Java** una **matriz** es un **vector** de **vectores fila**, o más en concreto **un vector de referencias a los vectores fila.**

Arrays bidimensionales (Matrices)

	0	1	2	3
0	16	3	1	87
1	5	7	2	8
2	10	1	34	12
3	23	6	12	6

`int[][] matriz;` Declarar el array

`matriz = new int[4][4];` Crear el array

```
matriz[0][0] vale 16
matriz [0][1] vale 3
...
matriz [3][3] vale 6
```

Arrays bidimensionales (Matrices)

- Al contrario que otros lenguajes, Java no fuerza a especificar el mismo tamaño para cada fila

```
int [ ][ ] x = new int [ 5 ][ ];
```

```
x [0] = new int [3];
```

```
x [1] = new int [2];
```

```
x [2] = new int [3];
```

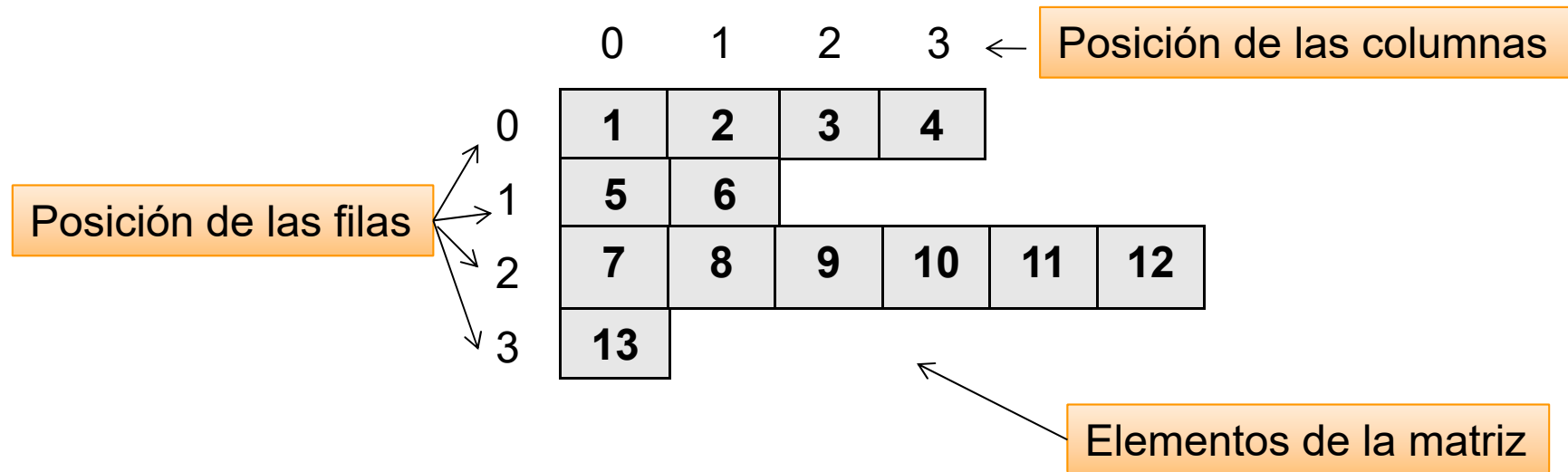
```
x [3] = new int [5];
```

```
x [4] = new int [1];
```

(0,0)	(0,1)	(0,2)		
(1,0)	(1,1)			
(2,0)	(2,1)	(2,2)		
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)				

5 filas con diferente tamaño

Arrays bidimensionales (Matrices)



```
int [][] matriz={{1,2,3,4},{5,6},{7,8,9,10,11,12},{13}};
```

Declaración, creación e inicialización

Acceso a los elementos de un array

Rellenar un array

```
for (int i=0; i < matriz.length; i++)  
{  
    for (int j=0; j < matriz[i].length; j++)  
    {  
        matriz[i][j] = generarNumeroAleatorio();  
    }  
}
```

Número de elementos en cada fila

Número de filas

Método que devuelve un número entero

Mostrar los elementos

```
for (int i=0; i < matriz.length; i++)  
{  
    for (int j=0; j < matriz[i].length; j++)  
    {  
        System.out.print(matriz[i][j]+"\\t");  
    }  
    System.out.println("");  
}
```

Uso de clases de biblioteca

- Las bibliotecas de clases estándar de Java contiene muchas clases útiles.

Se debe:

- Conocer algunas clases importantes por su nombre
- Conocer como se pueden localizar otras clases.

Importante:

- Solamente es necesario conocer la interfaz, no la implementación.

Interfaz e implementación

- **La interfaz** de **una clase** describe lo que es capaz de hacer y la manera en que se puede usar sin mostrar su implementación.

La documentación (de la biblioteca estándar de Java) incluye:

- El nombre de la clase
- Una descripción general del propósito de la clase
- Una lista de los constructores y los métodos de la clase
- Los parámetros y los tipos de retorno de cada constructor y de cada método
- Una descripción del propósito de cada constructor y cada método



La interfaz de la clase

Interfaz e implementación

- El **código completo** que define **una clase** se denomina **implementación**

*La documentación **no incluye**:*

- propiedades privadas (casi todas lo son)
- Métodos privados
- El cuerpo de cada método (código fuente)



La implementación de la clase

Interfaz e implementación

- **La interfaz** de **un método** consiste en su signatura y un comentario.

La documentación (de la biblioteca estándar de Java) incluye:

- Un modificador de acceso (public, private, ...)
- El tipo de retorno del método
- El nombre del método
- Una lista de parámetros (puede estar vacía)



La interfaz del método

- Proporciona todos los elementos para saber como se usa

Usando las librerías de clases

- Las clases de las librerías (paquetes en Java) **deben ser importadas** usando la sentencia *import* (excepto las clases de *java.lang* ya que se importan automáticamente).
 - Pertenecen a *java.lang* las clases **Object**, **String**, **Exception**, ...
- Estas clases pueden ser usadas como clases del proyecto que se esta implementando.

Paquetes e Import

- Las clases están organizadas en paquetes.

- Se puede importar una **sola clase**

```
import java.util.ArrayList;
```

- Se pueden importar **paquetes enteros**

```
import java.util.*;
```

Usando Random

- La clase Random puede ser usada para generar números aleatorios

Devuelve valores entre 0 (incluido) y 100 (excluido)

```
import java.util.Random;
...
Random randomGenerator = new Random();
...
int index1 = randomGenerator.nextInt();
int index2 = randomGenerator.nextInt(100);
```

Devuelve valores enteros entre -2147483648 y 2147483647

Ejercicios

- ❑ Escribir un método de nombre `lanzarDado` que devuelva un número comprendido entre 1 y 6 (inclusive)
- ❑ Escribir un método de nombre `generarValor` que tenga dos parámetros `min` y `max` y genere un número al azar en el rango comprendido entre `min` y `max` (inclusive)
- ❑ Escribir un método `getRespuesta` que devuelva aleatoriamente una de las siguientes cadenas: "si", "no" o "quizás"

Ejemplo

```
public void Responder()  
{  
    respuestas = new ArrayList<String>();  
    rellenarRespuestas();  
    System.out.println(generarRespuesta());  
}  
  
public String generarRespuesta()  
{  
    Random random = new Random();  
    int indice = random.nextInt(respuestas.size());  
    return respuestas.get(indice);  
}  
  
public void rellenarRespuestas()  
{  
    ...  
}
```

Encontrar el error ...

```
class Agenda {  
    private ArrayList<String> notas;  
  
    public Agenda()  
    {  
        ArrayList<String> notas = new ArrayList<String>();  
    }  
  
    public void addNota(String nota)  
    {  
        notas.add(nota);  
    }  
}
```

Principales conceptos que veremos

- Mas clases de librerías
- Escribir documentación

Conjuntos

- **Un conjunto** es una colección que almacena cada elemento individual una sola vez como máximo. No mantiene un orden específico.
 - El iterador puede devolver los elementos en diferente orden al que fueron añadidos
 - Si se añade un elemento por segunda vez, no tiene ningún efecto.
- **En un ArrayList** los elementos sí se mantienen en un orden específico, se accede a los elementos a través de un índice y puede contener el mismo elemento varias veces.

Usando conjuntos

```
import java.util.HashSet;
```

```
...
```

```
HashSet<String> miConjunto = new  
HashSet<String>();
```

```
miConjunto.add("one");  
miConjunto.add("two");  
miConjunto.add("three");
```

```
for(String elemento : miConjunto) {  
    hacer algo con cada elemento  
}
```

Comparar con el
código de
ArrayList

Dividir cadenas

```
public HashSet<String> crearConjunto(String linea)
{
    String[] palabrasArray = linea.split(" ");
    HashSet<String> palabras = new HashSet<String>();

    for(String palabra : palabrasArray) {
        palabras.add(palabra);
    }
    return palabras;
}
```

Divide una cadena en distintas subcadenas y las devuelve en un array de cadenas

Razona ...

¿ Que sucede si hay más de un espacio en blanco entre dos palabras ?

```
public HashSet<String> crearConjunto(String linea)
{
    String[] palabrasArray = linea.split(" ");
    HashSet<String> palabras = new HashSet<String>();

    for(String palabra : palabrasArray) {
        palabras.add(palabra);
    }
    return palabras;
}
```

Mapas

- **Un mapa** es una colección de pares de objetos clave/valor como entradas. Los valores se pueden buscar suministrando la clave.
 - Si se añade un elemento por segunda vez, no tiene ningún efecto.
- **En un ArrayList** los elementos sí se mantienen en un orden específico, se accede a los elementos a través de un índice y puede contener el mismo elemento varias veces.

Usando un HashMap

- Un mapa con Strings como claves y valores

:HashMap

"Carlos Rodriguez"	"985 924587"
"Lisa García"	"655 364674"
"Lucía Suarez"	"606 880123"

Usar un HashMap

```
HashMap <String, String> agenda = new HashMap<String, String>();
```

```
agenda.put(" Carlos Rodriguez ", " 985 924587 ");  
agenda.put(" Lisa García ", " 655 364674 ");  
agenda.put(" Lucía Suarez ", " 606 880123 ");
```

```
String numTelefono = agenda.get(" Lisa García ");  
System.out.println(numTelefono);
```

Razona...

- ¿Que ocurre cuando se añade una entrada al mapa con una clave que ya existe ?
- ¿Que ocurre cuando se añade una entrada en el mapa con un valor ya existente?
- ¿Cómo se puede verificar si el mapa contiene una clave determinada?
- ¿Como puedo saber cuantas entradas tiene un mapa?

Usar un HashMap

Lucía Suarez	600 111111
Carlos Rodriguez	985 924587
Lisa García	655 364674

Recorrer los elementos de la colección

```
HashMap <String, String> agenda = new HashMap<String, String>();
```

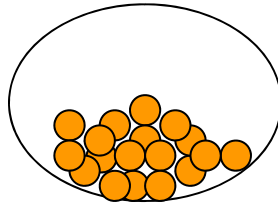
```
agenda.put(" Carlos Rodriguez ", " 985 924587 ");  
agenda.put(" Lisa García ", " 655 364674 ");  
agenda.put(" Lucía Suarez ", " 606 880123 ");  
agenda.put(" Lucía Suarez ", " 600 111111 ");
```

```
Iterator it = agenda.entrySet().iterator();  
while (it.hasNext()) {  
    Map.Entry e = (Map.Entry)it.next();  
    System.out.println(e.getKey() + " " + e.getValue());  
}
```

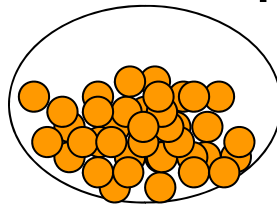
Razona...

Para el “sorteo de la Lotería de Navidad”

- ¿Qué colección usarías para almacenar los números de la lotería nacional?



- ¿Qué colección usarías para almacenar los premios?



- ¿Qué colección usarías para almacenar el resultado del sorteo?



Escribir documentación de clase

- Las clases creadas por nosotros deben estar documentadas de la misma forma que las clases de librería.
- Otras personas podrán usar estas clases sin necesidad de conocer la implementación.
- Podemos hacer que nuestras clases sean ¡clases de librería!

Elementos de documentación

La documentación de una clase debe incluir:

- El nombre de la clase
- Un comentario que describa el propósito general y las características de la clase
- Un número de versión
- El nombre del autor (o autores)
- La documentación de cada constructor y de cada método

Elementos de documentación

La documentación de cada constructor y cada método debe incluir:

- El nombre del método
- El tipo de retorno
- Los nombres y tipos de parámetros
- Una descripción del propósito y de la función del método
- Una descripción de cada parámetro
- Una descripción del valor que devuelve

Javadoc

Comentario de clase:

```
/**  
 * The Responder class represents a response  
 * generator object. It is used to generate an  
 * automatic response.  
 *  
 * @author      Michael Kölling and David J. Barnes  
 * @version     1.0   (30.Mar.2013)  
 */
```

Javadoc

Comentario de método:

```
/**
 * Read a line of text from standard input (the text
 * terminal), and return it as a set of words.
 *
 * @param line A line to print to screen.
 * @return A set of Strings, where each String is
 *         one of the words typed by the user
 */
public HashSet<String> getInput(String line)
{
    ...
}
```

Modificadores de acceso

public vs private

- ❑ Los atributos (propiedades o campos), constructores y métodos **públicos**, son accesibles dentro de la misma clase o fuera de ella (desde otras clases)
- ❑ Los atributos **no deben** ser públicos
- ❑ Los atributos **privados** son accesibles solamente desde dentro de la misma clase en la que se definen
- ❑ Solamente los métodos que son llamados desde otras clases **serán públicos**

Modificadores de acceso

Sin palabra reservada

Se puede acceder desde cualquier parte del *package* al que pertenezca la clase.

¡es la ocultación por omisión!

Ocultación de la información

- Los **detalles internos** de implementación de una **clase** deben **permanecer ocultos** para **otras clases**.

abstracción y modularización

“Si necesitásemos conocer todos los detalles internos de todas las clases que queremos usar, no terminaríamos nunca de implementar sistemas grandes”.

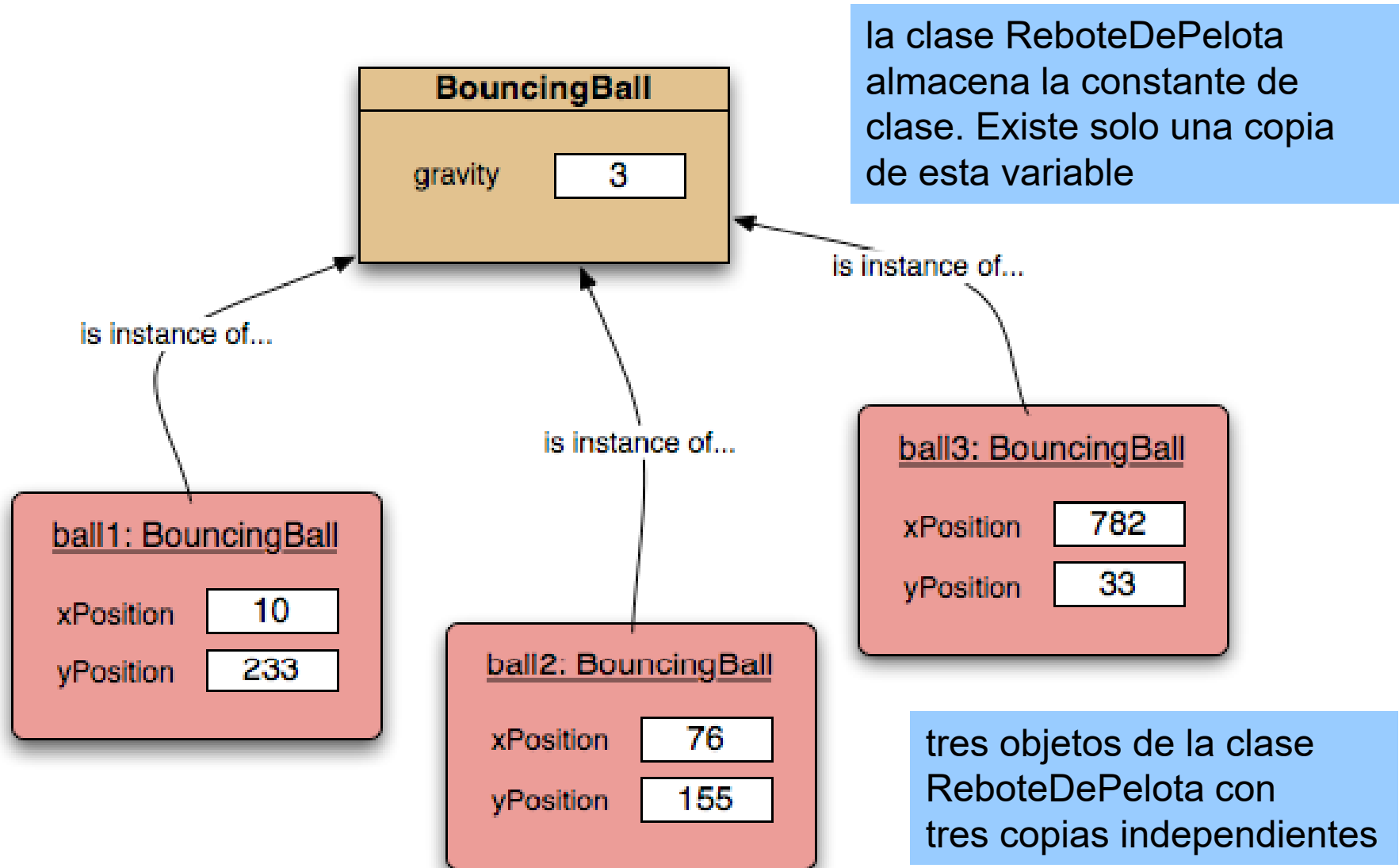
Variables y constantes de clase

- **Las variables y constantes de clase** son propiedades que se almacenan en la misma clase y no en un objeto.
- **Las variables de instancia** se almacenan en cada objeto.

```
public class Pelota
{
    //Efecto de gravedad
    private static final int GRAVEDAD = 3;

    private int posicionX;
    private int posicionY;
    // se omiten los otros campos y métodos
}
```

Variables y constantes de clase



Constantes de clase

```
private static final int gravity = 3;
```

private: modificador de acceso

static: variable de clase

final: constante

Ejemplo

```
public Class Farola
{
    public static final boolean ENCENCIDA = true;
    public static final boolean APAGADA= false;

    private boolean estado = APAGADA;

    public void setEstado (boolean valor)
    {
        this.estado = valor;
    }
    ...
}
```

```
Farola miFarola = new Farola();
miFarola.setEstado(Farola.ENCENCIDA);
```

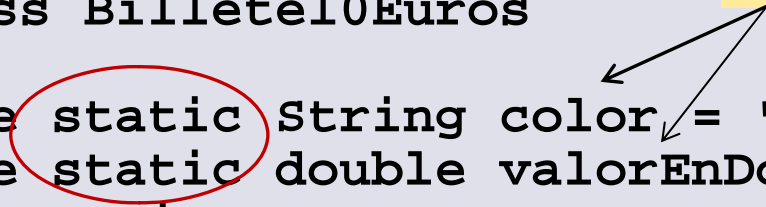
Variables y métodos de clase. Ejemplo

```
public class Billete10Euros
{
    private static String color = "rojo";
    private static double valorEnDolares = 1.25*10;
    private String numeroID;

    public Billete10Euros(String numeroID)
    {
        this.numeroID = numeroID;
    }

    // método estático para fijar la cotización
    public static void fijarCot( double newCotización)
    {
        valorEnDolares = newCotización;
    }
}
```

variables de clase



Métodos de clase (métodos estáticos)

- **Los métodos estáticos** tienen el mismo efecto independientemente del objeto
 - Se ejecutan sin necesidad de crear un objeto (están controlados por la clase).
 - Solo pueden acceder a las variables de clase.

Ejemplo La clase Math

Math.PI //constante de clase

Math.sqrt() //método de clase

Revisión

- ❑ Java tiene bibliotecas de clases muy extensas.
- ❑ Un buen programador debe estar familiarizado con las bibliotecas.
- ❑ La documentación nos proporciona lo que necesitamos conocer para usar una clase (interface).
- ❑ La implementación queda oculta (ocultación de la información).
- ❑ Es obligatorio documentar nuestras propias clases.