

Clase adicional 2

Temas

- Estructuras de control
 - § Sentencia condicional
 - § Iteración
- Clases
 - § Definir una clase
 - § Crear una instancia de una clase
 - § Campos estáticos
- Problemas de la clase adicional
- Problema de diseño

Estructuras básicas de control

Existen dos estructuras básicas de control

Sentencias condicionales: `if ... else...`

Iteración: `bucle while`, `bucle do ... while` y `bucle for`

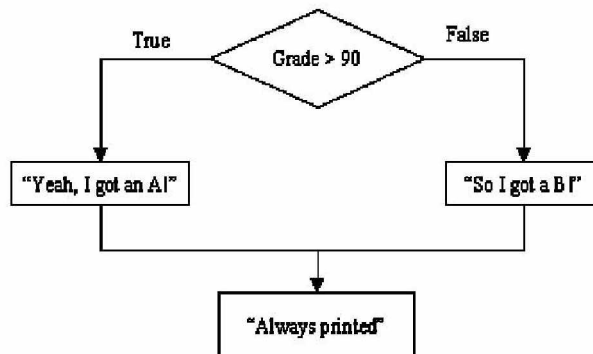
Sentencia condicional

La forma general de una sentencia *if* es:

```
if(expresión)
    sentencia1
else
    sentencia2
siguiente sentencia
```

Si la expresión es *true*, sentencia1 se ejecuta. Si es *false*, sentencia2 se ejecuta.
Por ejemplo:

```
if (grade >= 90)
    System.out.println("¡Bien! ¡He sacado un 10!");
else
    System.out.println("¡Vaya! ¡He sacado un 8!");
System.out.println ("Siempre impreso");
```



Estructura de iteración

En ocasiones, es necesario repetir las mismas sentencias varias veces. En este caso, lo que necesita es utilizar la estructura de iteración. Hay dos tipos de estructuras de iteración: bucles indeterminados y bucles determinados.

En un bucle indeterminado, se desconoce el número de veces que se ejecutará el bucle. Por lo tanto, se requerirá la ayuda de una expresión booleana:

```

while (expresión)
{
    sentencias;
    .....
}
  
```

Las sentencias se ejecutarán repetidamente siempre y cuando la expresión sea *true*. A continuación se muestra un ejemplo:

```

Grade = 0;
while (Grade <= 90) {
    System.out.println("Hay que estudiar más!");
    Grade += 10;
}
System.out.println("¡Bien hecho!");
  
```

En este caso, el mensaje "¡Hay que estudiar más!" se imprime 10 veces mientras la calificación (grade) avanza de 0 a 90. Sólo cuando llegue a 90 verá el mensaje "¡Bien hecho!".

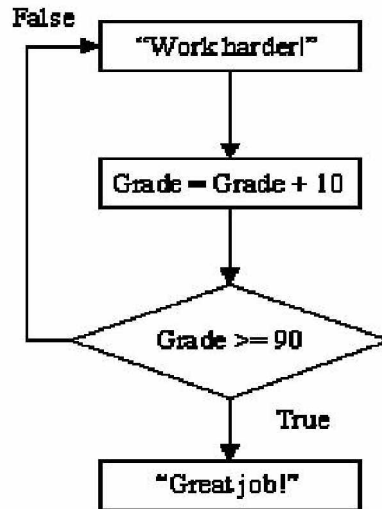
Una variante del bucle *while* es la estructura *do-while*. En vez de comprobar la expresión al principio del bucle, una estructura *do-while* realiza la comprobación al final. La forma general de esta estructura es:

```

do
    sentencia
while (expresión);
  
```

siguiente sentencia

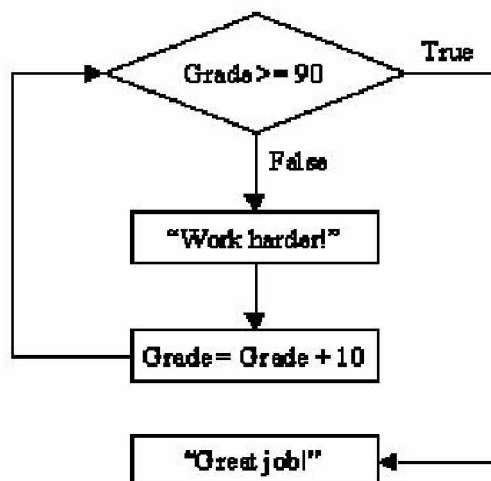
A continuación se muestra un gráfico de flujo del ejemplo anterior.



Si quiere controlar el número de veces que se ejecutarán las sentencias (“bucle determinado”), deberá utilizar el bucle *for*. En este tipo de bucle, se define un contador que se actualiza tras cada iteración. Se saldrá del bucle cuando el contador llegue a su límite. A continuación se muestra un ejemplo:

```
for(grade = 0; grade <= 90; grade+=10) {  
    System.out.println("¡Hay que estudiar más!");  
}  
System.out.println("¡Bien hecho!");
```

Aquí, la calificación se inicializa en 0. En cada iteración, agregamos 10 puntos a la calificación. Cuando llega a 90, el bucle termina y el control pasa a la siguiente sentencia. Como se puede comprobar en el siguiente gráfico de flujo, el bucle *for* hace lo mismo que el bucle *while*.



Clases

Una clase proporciona la definición de un objeto. Cada clase contiene:

Variables (o campos): atributos del objeto.

Métodos: acciones que el objeto puede ejecutar.

Por ejemplo, podemos definir una clase `Box` que contiene tres atributos: *width* (ancho), *height* (alto) y *length* (largo). También contiene tres métodos `setWidth`, `setHeight` y `setLength` que los usuarios pueden utilizar para definir el alto, el ancho y el largo del objeto `Box`.

Definir una clase

```
public class Box {
    //Atributos

    private double width;
    private double height;
    private double length;

    //Métodos

    setWidth (double w);
    setHeight (double h);
    setLength (double l);
}
```

Cada variable cuenta con un

Identificador de acceso. Una variable puede ser *public* o *private* (existen otros identificadores de acceso que abordaremos más adelante en el curso). En este ejemplo, todas las variables de la clase `Box` se han declarado como *private*, lo que implica que únicamente se puede acceder a ellas desde la clase. Por su parte, a una variable *public* puede acceder cualquier usuario, tanto dentro como fuera de la clase.

Tipo de datos. Una variable puede ser simplemente un tipo de datos como *int* o *double*, u otra clase.

Nombre. El nombre de las variables debe comenzar por una letra, distingue entre mayúsculas y minúsculas y no puede ser una palabra reservada de Java. Una buena práctica es utilizar nombres descriptivos, por ejemplo, *height* (altura) en vez de *h*

Por ejemplo, en la clase `Box`, *height* es una variables *private* del tipo *double*

Hablaremos de las definiciones de los métodos en la siguiente clase; también tienen especificadores de acceso y otras características.

Crear la instancia de una clase

Una vez que se ha definido una clase, es posible crear instancias de dicha clase. Cada instancia contiene el mismo conjunto de variables pero distintos valores. Aquí, nos gustaría crear una instancia llamada myFirstBox:

```
Box myFirstBox = new Box();
```

Tenga en cuenta que myFirstBox no es un objeto; es una referencia al objeto que se crea cuando se llama al nuevo operador. El nuevo operador invoca el constructor de la clase (un método especial al que se llama cuando se crea un objeto por primera vez) y reserva memoria para dicho objeto. (Hablaremos de los constructores de forma detallada en la siguiente clase adicional).

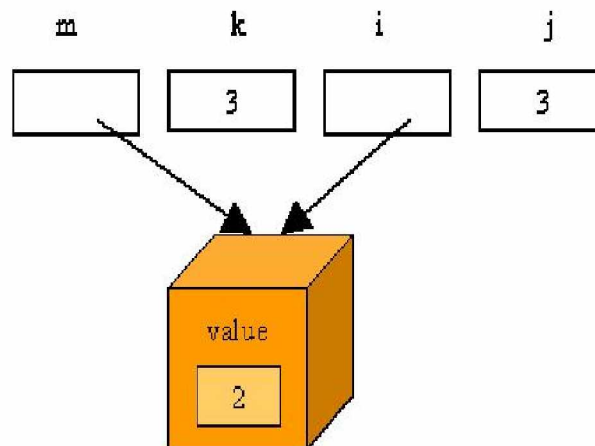
El siguiente ejemplo se centra en una explicación visual de la diferencia entre el objeto y la referencia al objeto. Aquí, hemos definido dos variables: una variable de datos primitiva (int j) y una variable de objeto (Integer i). Ambas contienen el valor entero 2. Utilizamos el cuadrado para representar el tipo de datos primitivo y la caja para representar al objeto.

```
int j = 2;  
Integer i = new Integer(2);
```

Tal como se puede observar en el siguiente dibujo, la variable primitiva j contiene el valor 2. Sin embargo, la variable del objeto, i, no contiene el objeto *Integer*, sino una referencia a dicho objeto con un valor de 2.

Entonces, ¿qué ocurre si asignamos i a otro objeto *Integer*? ¿Podría explicarlo utilizando la misma representación de caja y cuadrado?

```
int k = j;  
Integer m = i;
```



Campos *Static*

Cada instancia de objeto contiene su propio conjunto de variables de miembros y métodos. Los cambios realizados en una instancia no afectan al resto. La única excepción son los campos y los métodos estáticos (*static*).

Si define un campo como *static*, sólo podrá haber uno por clase. Veamos el siguiente ejemplo:

```
public class Box {
    //Atributos

    static int numberOfBoxes = 0;
    private double width;
    .....
}
```

Si creamos 1.000 instancias de *Box*, cada objeto *box* tendrá su propio ancho, pero sólo habrá un *numberOfBoxes* para la clase.

Dado que los campos *static* pertenecen a la clase, se hace referencia a ellos mediante el nombre de la clase, y no mediante el nombre de la instancia. Por ejemplo:

```
public static void main (String args[]) {
    Box b = new Box();
    System.out.println(b.getWidth());
    System.out.println(Box.numberOfBoxes());
}
```

Problemas de la clase adicional

1. Bucle *for*

¿Qué datos de salida se obtienen con el siguiente programa?

```
for (int i = 1; i < 4; i++) {
    for (int j = 1; j < 3; j++) {
        System.out.println("i = " + i + " + j = " + j + "\n" );
    }
}
```

Respuesta:

2. Bucle *do-while*

Escriba un pequeño programa que imprima los enteros impares desde 2 hasta 100 utilizando un bucle *do-while*:

```
int counter = 2;
do
{

} while (_____);
```

3. Bucle *for*

Intente hacer lo mismo utilizando el bucle *for*.

```
for (_____; _____; _____)
_____;
```

4. Clase

Detecte los errores de compilación del siguiente programa:

```
public class Box {
    private double width;
    private double height;
    private double length;
}

public class TestBox {
    public static void main (String argv[]) {
        Box b1 = new Box();
        b1.width = 5;
        System.out.println(b1.width);
    }
}
```

Respuesta:

5. Escriba un programa de Java que calcule el factorial (n!) de un entero no negativo *n*. La fórmula es la siguiente:

$$\text{factorial}(n) = \begin{cases} 1 & \text{if } n < 2, \\ n * (n - 1) * \dots * 1 & \text{otherwise} \end{cases}$$

Problema de diseño

Defina un conjunto de clases, con sus variables de miembros, para modelar cursos del MIT. A continuación se incluye una breve descripción:

Cada curso del MIT tiene un número de curso, un número de aula, un instructor y 2 profesores adjuntos

Cada profesor adjunto tiene un nombre y una dirección de correo electrónico

Cada profesor tiene un nombre, un título (profesor, profesor asociado, etc.), un número de teléfono y un número de despacho

Cada aula de clase tiene un número de aula y una capacidad

Tras definir las clases, cree una instancia de describa el curso 1.00. (Puede obtener toda la información necesaria en el programa del curso y la instancia sólo debe incluir 2 estudiantes: usted mismo y su compañero).