Programación en Java

Tema 3. Programación orientada a objetos en Java (Parte 1)

Luis Rodríguez Baena

Universidad Pontificia de Salamanca (campus Madrid)
Facultad de Informática

Clases y objetos (I)

- Clase
  - Cojunto de datos (*atributos*) y funciones (*métodos*) que definen la estructura de los objetos y los mecanismos para su manipulación.
  - Atributos y métodos junto con *interfaces* y clases *anidadas* constituyen los **miembros** de una clase.
- Declaración

```
[modificadores] class NombreDeClase{
    //Declaración de atributos
    //Declaración de métodos
    //Declaración de clases anidadas e interfaces
}
```

Clases y objetos (II)

- Modificadores de clase
  - `public`, todo el mundo puede acceder a ella. Sin este modificador sólo podrían acceder los miembros de su clase o los de las clases de su paquete.
  - `abstract`, clase incompleta de la que no se pueden crear instancias. Se utiliza para implementar *superclases* que las clases "hijas" deberán completar.
  - `final`, no admite subclases por lo que no se podrán sobreescribir. Todos sus métodos serán a su vez `final`, por lo que no podrán ser sobreescritos.
  - `synchronizable`, todos sus métodos son sincronizables, es decir no se puede acceder a ellos desde distintos hilos (*threads*) de ejecución.

Objetos (I)

- Instancia de una clase.
  - Para su uso es necesaria la declaración, la instanciación y la inicialización del objeto.

```
class Empleado{
   long idEmpleado = 0;
   String nombre = "SinNombre";
   double sueldo = 0;
}
```

  - Declaración.

```
Empleado e;
```

  - Instanciación.

```
e = new Empleado();
```

  - Se puede resumir en una única instrucción:

```
Empleado e = new Empleado();
```

Objetos (II)

- Constructores.
  - Constructor por omisión (constructor *no-args*): `Asignatura(){}`
  - Bloques de sentencias declarado dentro de una clase con el mismo nombre que la clase y sin valor de retorno.
  - Las sentencias inicializan la instancia y se invocan después de asignar los valores por omisión de los atributos.

```
Empleado(String nom){
    nombre = nom;
  }
…
Empleado e = new Empleado("Juan Martinez);
```

Objetos (III)

- Constructores (*continuación*).
  - Es posible utilizar las características de otro constructor utilizando la palabra reservada `this` que devuelve una referencia al objeto.
    - ✓ `this.idEmpleado`, haría referencia al código actual del objeto.
    - ✓ `this(long)`, haría referencia al constructor.

```
Empleado(long id){
   idEmpleado = id;
}

Empleado(long id, String nombre){
   this(id);
   this.nombre = nombre;
…
Empleado e = new Empleado(12345);
Empleado e1 = new Empleado(343234, "Ana Lopez")
```

Destructores y recolección de basura

- El entorno de ejecución de Java dispone de un recolector de basura (*garbage collector*) que limpia de la memoria los objetos no utilizados.
  - Cuando un objeto no se puede referenciar (acaba la vida de la variable de referencia) marca el objeto como basura.
  - Cuando lo considere oportuno el recolector de basura lo eliminará
  - No es un destructor.
- El método `finalize()` se ejecuta antes de eliminarlo.
  - Se puede utilizar para realizar otras operaciones de limpieza, cerrar archivos, etc.

Accesibilidad de las clases

- Por omisión: acceso de paquete (*friendly* o amistoso).
  - La clase sólo puede ser utilizada por otras clases del paquete.
- Modificador `public`: permite que sea utilizada por otras clases.
  - Sólo puede haber una clase pública por unidad de compilación.
  - Su nombre debe coincidir con el de la unidad de compilación.
  - Puede haber una unidad de compilación sin clases públicas.
- No hay clases `private`.

Atributos (I)

- "Variables" de la clase.
  - Se declaran igual que las variables.
- Inicialización de atributos.
  - Valor por omisión, expresión de inicialización, o mediante constructores.

Atributos (II)

- Modificadores de acceso:
  - Acceso paquete (sin modificador de acceso).
    - ✓ Pueden acceder todas los miembros de la clase y de la clase del paquete.
    - ✓ Para el resto de paquetes, se considerará como acceso privado.
    - ✓ Métodos get / set para facilitar el acceso al resto de paquetes.
  - Acceso publico (modificador `public`).
    - ✓ Disponible para todas las clases que se encuentren en el directorio de `CLASSPATH`.
    - ✓ Las clases del mismo directorio se consideran del mismo paquete (acceso de paquete).

Atributos (III)

- Modificadores de acceso (*continuación*).
  - Acceso privado (modificador `private`).
    - ✓ Ningún miembro de otra clase puede acceder al atributo.
    - ✓ Atributos útiles para implementar métodos *ayudantes* (que utilizan otros métodos de la clase).
    - ✓ Posibilidad de acceso mediante métodos get/set.

```
private double sueldo = 0;
...
public double getSueldo(){
   return nota;
}
...
//Daría error
//System.out.println(e.sueldo());
System.out.println(e.getSueldo());
```

Atributos (IV)

- Modificadores de acceso (*continuación*).
  - Acceso protegido (modificador `protected`).
    - ✓ Se utiliza para la herencia.
    - ✓ Permite a las clases hijas utilizar los atributos de la clase base aunque pertenezcan a distintos paquetes.
- Niveles de acceso.

Modificador	Clase	Subclase	Paquete	Mundo
`private`	X			
`protected`	X	X	X	
`public`	X	X	X	X
"paquete"	X		X	

Atributos (V)

- Atributos de clase (modificador `static`).
  - Para campos que compartan todas las instancias de una clase.

```
...
static long numEmpleados = 0;

Empleado(){
    numEmpleados++;
}
...
System.out.println(Empleados.numEmpleados);
```

  - Se utiliza como cualificador el nombre de la clase o de la instancia (no recomendado).

Atributos (VI)

- Atributos constantes (modificador `final`)
  - No pueden cambiarse una vez inicializados.
  - En atributos de tipos de datos primitivos se inicializan en tiempo de compilación y se deben inicializar en la declaración.
  - En tipos de referencia:
    - ✓ No puede cambiar la referencia aunque si los atributos de la clase.
    - ✓ Para definir una clase final habría que definir todos sus atributos como final.
    - ✓ Se puede hacer una inicialización tardía (*constantes final blancas* o *inicialización perezosa*).

Métodos (I)

- Proporcionan la funcionalidad a las clases y contiene el código que maneja el estado de un objeto.
- Declaración.

```
[modificadores de acceso][static][final][abstract]
   [synchronized] tipoRetorno
      nombreMétodo([listaParametrosFormales])
       [throws listadeExceptiones]{
          //cuerpo del método
}
```

- Invocación.

```
objeto.método([listaParámetrosActuales])
```

Métodos (II)

- Modificadores de acceso

Visibilidad	public	protected	private	Por omisión
Desde la propia clase	X	X	X	X
Desde otra clase del paquete	X	X		X
Desde otra clase fuera del paquete	X			
Desde otra subclase del paquete	X	X		X
Desde otra subclase fuera del paquete	X	X		

Métodos (III)

- Tipos de retorno
  - Datos primitivos, arrays, objetos, interfaces, objeto de la clase o subclase o `void`.
  - El valor lo devuelven mediante la sentencia `return`.
- Paso de argumentos
  - Siempre se pasan por valor
    - ✓ En tipos de datos primitivos pasan copias de los parámetros actuales.
    - ✓ En tipos de datos de referencia se pasa el valor de la referencia.
      - × Los atributos de la referencia se pueden modificar.

Métodos (IV)

- Sobrecarga de métodos.
  - Asigna un mismo nombre a métodos con distinta funcionalidad.

```
void aumentarSueldo(float porCiento){
   this.sueldo = (this.sueldo * porCiento / 100) + this.sueldo;
}

void aumentarSueldo(double euros){
   this.sueldo += euros;
}
```

  - El compilador los distingue por el número y tipo de argumentos.
    - ✓ ¿Es posible utilizar el tipo de retorno para distinguirlos?

```
int f1(){…};
float f1{…};
//Llamada al método entero
int x = f1();
//Llamada al método real
float y = f1();
//Llamada ¿a qué método?
f1();
```

Métodos (V)

- Métodos de clase: modificador `static`.
  - Se pueden invocar sin crear una instancia de la clase.
  - Invocación: *`NombreClase.nombreMétodo`*`([`*`listaArgumentos`*`])`

```
//Obtiene el número de empleados instanciados
static long cuantosEmpleados(){
   return numEmpleados;
}
System.out.println(Empleados.cuantasEmpleados());
```

- Métodos constantes: modificador `final`.
  - Ninguna clase puede redefinir el método.
  - Decisión de diseño: impedir que la herencia pueda modificar el comportamiento del método.
  - Mejora la eficiencia.
  - Cualquier método privado es constante: sólo es accesible desde su clase.

Métodos (VI)

- Métodos sincronizados: modificador `synchronized`.
  - Para secciones críticas en programación multihilo.
  - Bloquea un método hasta que acaba su ejecución.
    - ✓ Impide que otro hilo ejecute el método hasta que el método `synchronized` acabe.
- Métodos abstractos: modificador `abstract`.
  - Se utilizan para la herencia.
  - Métodos no implementados de una clase.
  - Se declaran sólo con el tipo de retorno, el nombre y los argumentos.

```
abstract void f1();
```

Métodos (VII)

- Métodos nativos.
  - Se utilizan para declarar la existencia de código no escrito en Java.
  - Su implementación se almacenará en alguna biblioteca dinámica (por ejemplo una DLL).
  - Perdida de compatibilidad (están compilados para un entorno determinado).
  - JNI (*Java Native Interface*), interfaz estándar para programadores de C y C++.

```
public native int nombreMetodo();
```