CENTRO DE TECNOLOGIAS DE INFORMACION Y COMUNICACION

JAVA BASICO

Instructor: LENIN LEON
Email: leninleon0720@gmail.com
Celular: 6234-5608

OBJETOS Y CLASES

Como ya hemos comentado Java es un lenguaje totalmente orientado a objetos, mucho más que, por ejemplo, C++. En Java todo es un objeto, a excepción de los tipos básicos de variables enteras, reales y char. Pero bien, ¿qué es un objeto? y ¿qué es la programación orientada a objetos?.

OBJETOS Y CLASES

Responder a estas preguntas no es en absoluto trivial, hay libros enteros escritos sobre objetos y metodologías de programación orientada a objetos sin abordar ningún lenguaje de programación en concreto . Aquí simplemente daremos unas nociones muy básicas de programación orientada a objetos que nos permitan empezar a adentrarnos en el mundo de Java.

OBJETOS Y CLASES

En los años 60 la programación se realizaba de un modo "clásico" (no orientado a objetos). Un programa era un código que se ejecutaba, los trozos de código que se podían emplear en varias ocasiones a lo largo del programa (reusar) se escribían en forma de procedimientos que se invocaban desde el programa, y esta era la única capacidad de reuso de código posible.

OBJETOS Y CLASES

Según los códigos se fueron haciendo más grandes y complejos este estilo de programación se hacía más inviable: es difícil programar algo de grandes dimensiones con este estilo de programación. La única posibilidad de repartir trozos de código relativamente independientes entre programadores son los procedimientos, y al final hay que juntar todos estos con el programa central que los llama, siendo frecuente encontrar problemas al unir estos trozos de código.

OBJETOS Y CLASES

En los años 70 se empezó a imponer con fuerza otro estilo de programación: POO, programación orientada o objetos (en la literatura suele aparecer como OOP, Object Oriented Programing). Aquí un programa no es un código que llama a procedimientos, aquí un programa es un montón de objetos, independientes entre si, que dialogan entre ellos pasándose mensajes para llegar a resolver el problema en cuestión.

OBJETOS Y CLASES

A un objeto no le importa en absoluto como está implementado otro objeto, que código tiene o deja de tener, que variables usa.... sólo le importa a que mensajes es capaz de responder. Un mensaje es la invocación de un método de otro objeto. Un método es muy semejante a un procedimiento de la programación clásica: a un método se le pasan uno, varios o ningún dato y nos devuelve un dato a cambio.

OBJETOS Y CLASES

Si hay que repartir un programa de grandes dimensiones entre varios programadores a cada uno se le asignan unos cuantos objetos, y en lo único que tendrán que ponerse de acuerdo entre ellos es en los mensajes que se van a pasar; la forma en que un programador implemente sus objetos no influye en absoluto en lo que los demás programadores hagan. Esto es así gracias a que los objetos son independientes unos de otros (cuanta mayor sea la independencia entre ellos de mayor calidad serán).

OBJETOS Y CLASES

Si analizamos lo que hemos dicho hasta aquí de los objetos veremos que estos parecen tener dos partes bastante diferenciadas: la parte que gestiona los mensajes, que ha de ser conocida por los demás, y que no podremos cambiar en el futuro sin modificar los demás objetos (sí es posible añadir nuevos métodos para dar nuevas funciones al objetos sin modificar los métodos ya existentes).

OBJETOS Y CLASES

La otra parte es el mecanismo por el cual se generan las acciones requeridas por los mensajes el conjunto de variables que se emplean para lograr estas acciones. Esta segunda parte es, en principio, totalmente desconocida para los demás objetos (a veces no es así, pero es lo ideal en una buena OOP).

OBJETOS Y CLASES

Por ser desconocida para los demás objetos podemos en cualquier momento modificarla sin que a los demás les importe, y además cada programador tendrá total libertad para llevarla a cabo como él considere oportuno.

OBJETOS Y CLASES

La OOP permite abordar con más posibilidades de éxito y con un menor coste temporal grandes proyectos de software, simplificándole además la tarea al programador.

CLASES Y HERENCIA

Una clase es la “plantilla” que usamos para crear los objetos. Todos los objetos pertenecen a una determinada clase. Un objeto que se crea a partir de una clase se dice que es una instancia de esa clase. Las distintas clases tienen distintas relaciones de herencia entre si: una clase puede derivarse de otra, en ese caso la clase derivada o clase hija hereda los métodos y variables de la clase de la que se deriva o clase padre. En Java todas las clases tienen como primer padre una misma clase: la clase Object.

CLASES Y HERENCIA

Una clase es la definición de las tareas que se van a realizar. Incluye las variables necesarias y los métodos, tanto públicos como privados. En lenguajes estructurados, los métodos equivalen a las funciones y procedimientos, como en C o Pascal.

Un objeto es la instancia de una clase. Con un objeto se pueden ejecutar las tareas definidas en la clase.

CLASES Y HERENCIA

Herencia es la capacidad de recibir todos los métodos y variables de una o más clases para realizar ciertas tareas. Por lo general, las subclases agregan métodos y modifican algunos métodos para realizar tareas diferentes. Por ejemplo, la clase Object (de la que heredan todas las demás clases en Java) define un método llamado toString() que regresa la representación textual del objeto. Cada clase modifica en cierta manera el comportamiento de toString() para regresar valores de acuerdo a la tarea para la que fue creada la clase.

Definición de una clase

La forma más general de definición de una clase en Java es:

```
[Modificador] class nombreClase [extends nombreClasePadre]
[implements interface] {
Declaración de variables;
Declaración de métodos;
}
```

Definición de una clase

Los campos que van entre corchetes son optativos. nombreClase es el nombre que le queramos dar a nuestra clase, nombreClasePadre es el nombre de la clase padre, de la cual hereda los métodos y variables. En cuanto al contenido del último corchete ya se explicará más adelante su significado.
Los modificadores indican las posibles propiedades de la clase. Veamos que opciones tenemos:

Modificadores de clases

public: La clase es pública y por lo tanto accesible para todo el mundo. Sólo podemos tener una clase public por unidad de compilación, aunque es posible no tener ninguna.
final: Indicará que esta clase no puede “tener hijo”, no se puede derivar ninguna clase de ella.
abstract: Se trata de una clase de la cual no se puede instanciar ningún objeto.

Modificadores de clases

Ninguno: La clase es “amistosa”. Será accesible para las demás clases del package. Sin embargo mientras todas las clases con las que estemos trabajando estén en el mismo directorio pertenecerán al mismo package y por ello serán como si fuesen públicas.

Modificadores de clases

Veamos un ejemplo de clase en Java:

```
class Animal{
    int edad;
    String nombre;
    public void nace(){
        System.out.println("Hola mundo");
    }
    public void getNombre(){
        System.out.println(nombre);
    }
    public void getEdad(){
        System.out.println(edad);
    }
}
```

Sobrecarga de métodos

Java admite lo que se llama sobrecarga de métodos: puede haber varios métodos con el mismo nombre pero a los cuales se les pasan distintos parámetros. Según los parámetros que se le pasen se invocará a uno u otro método:

Sobrecarga de métodos

```
class Animal{
    int edad;
    String nombre;
    public void nace(){
        System.out.println("Hola mundo");
    }
    public void getNombre(){
        System.out.println(nombre);
    }
    public void getNombre(int i){
        System.out.println(nombre +" " +edad);
    }
    public void getEdad(){
        System.out.println(edad);
    }
}
```

Constructores

Constructores son métodos cuyo nombre coincide con el nombre de la clase y que nunca devuelven ningún tipo de dato, no siendo necesario indicar que el tipo de dato devuelto es void. Los constructores se emplean para inicializar los valores de los objetos y realizar las operaciones que sean necesarias para la generación de este objeto (crear otros objetos que puedan estar contenidos dentro de este objeto, abrir un archivo o una conexión de internet.....).

Constructores

Como cualquier método, un constructor admite sobrecarga. Cuando creamos un objeto (ya se verá más adelante como se hace) podemos invocar al constructor que más nos convenga.

Constructores

```
class Animal{
	int edad;
	String nombre;
	public Animal(){
	}
	public Animal(int _edad, String _nombre){
		edad = _edad;
		nombre = _nombre;
	}
	public void nace(){
		System.out.println("Hola mundo");
	}
	public void getNombre(){
		System.out.println(nombre);
	}
	public void getNombre(int i){
		System.out.println(nombre +" " +edad);
	}
	public void getEdad(){
		System.out.println(edad);
	}
}
```

Modificadores de métodos y variables

Antes de explicar herencia entre clases comentaremos cuales son los posibles modificadores que pueden tener métodos y variables y su comportamiento:

Modificadores de métodos y variables

Modificadores de variables

public: Pública, puede acceder todo el mundo a esa variable.

protected: Protegida, sólo pueden acceder a ella las clases hijas de la clase que posee la variable y las que estén en el mismo package.

private: Privada, nadie salvo la clase misma puede acceder a estas variables. Pueden acceder a ella todas las instancias de la clase (cuando decimos clase nos estamos refiriendo a todas sus posibles instancias)

Modificadores de métodos y variables

Modificadores de variables

Ninguno: Es “amistosa”, puede ser accedida por cualquier miembro del package, pero no por otras clases que pertenezcan a otro package distinto.

static: Estática, esta variable es la misma para todas las instancias de una clase, todas comparten ese dato. Si una instancia lo modifica todas ven dicha modificación.

final: Final, se emplea para definir constantes, un dato tipo final no puede variar nunca su valor. La variable no tiene porque inicializarse en el momento de definirse, pero cuando se inicializa ya no puede cambiar su valor.

Modificadores de métodos y variables

Modificadores de variables

```
class Marciano {
	boolean vivo;
	private static int numero_marcianos = 0;
	final String Soy = "marciano";
	void quienEres(){
		System.out.println("Soy un " + Soy);
	}
	Marciano(){
		vivo = true;
		numero_marcianos++;
	}
	void muerto(){
		if(vivo){
			vivo = false;
			numero_marcianos--;
		}
	}
}
```

Modificadores de métodos y variables

Modificadores de un método

public: Pública, puede acceder todo el mundo a este método.

Ninguno: Es “amistoso”, puede ser accedida por cualquier miembro del package, pero no por otras clases que pertenecen a otro package.

protected: Protegido, sólo pueden acceder a ella las clases hijas de la clase que posea el método y las que estén en el mismo package.

private: Privada, nadie salvo la clase misma puede acceder a estos métodos.

Modificadores de métodos y variables

Modificadores de un método

static: Estática, es un método al cual se puede invocar sin crear ningún objeto de dicha clase. Math.sin, Math.cos son dos ejemplos de métodos estáticos. Desde un método estático sólo podemos invocar otros métodos que también sean estáticos.

final: Final, se trata de un método que no podrá ser cambiado por ninguna clase que herede de la clase donde se definió. Es un método que no se puede “sobrescribir”. Más adelante se explicará que es esto.

Modificadores de métodos y variables

Modificadores de un método

static: Estática, es un método al cual se puede invocar sin crear ningún objeto de dicha clase. Math.sin, Math.cos son dos ejemplos de métodos estáticos. Desde un método estático sólo podemos invocar otros métodos que también sean estáticos.

final: Final, se trata de un método que no podrá ser cambiado por ninguna clase que herede de la clase donde se definió. Es un método que no se puede “sobrescribir”. Más adelante se explicará que es esto.