INSTITUTO POLITÉCNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA DE INGENIERÍA Y CIENCIAS SOCIALES Y ADMINISTRATIVAS

Material para el curso de Programación Orientada a Objetos

Contenido

2.1.1 Creación de clases con base en estándares de codificación y su representación de clases en UML

El primer programa

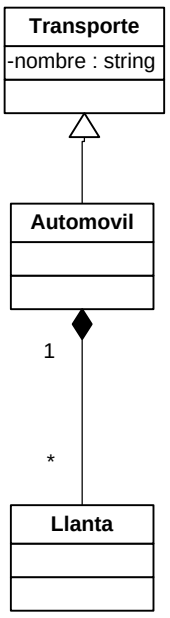
```
public class HolaMundo {
        public static void main (String args [ ])   {
                System.out.println ("Hola Mundo");
        }
}
```

Nuestro primer programa HolaMundo es una clase con un solo método *main*, que es el que nos indica que se trata de una aplicación completa, comúnmente abreviada App.

Los atributos o variables, que guardan el estado de cada objeto, pueden ser de tipos primitivos, pero también pueden ser objetos a su vez. Esto se llama inclusión o composición, y para representarla en UML (Unified Modelling Language) utilizamos el símbolo de diamante: ◊

Otra relación importante entre clases es la Herencia o derivación, que se representa en UML con un triangulo hacia arriba: Δ

Por ejemplo, la clase Automovil hereda de Transporte pues en efecto es una especialización de aquel, pero la clase Llanta no hereda de Automovil, sino que cada objeto Automovil tiene 4 atributos de la clase Llanta:



Existen en Java muchas reglas para nomenclatura. Primeramente recordemos la notación Húngara o “camello”, la cual indica que la primera letra de cada palabra debe escribirse en mayúsculas para poder separarlas visualmente sin necesidad del underscore o guion “chaparro”, excepto la primera letra de la primera palabra. Esta primera letra de la primera palabra debe ser mayúscula para clases e interfaces (y en singular), y minúscula para cualquier otro nombre, como variable, método, etc.

Sin embargo, las constantes se sugiere se escriban solo con mayúsculas, por eso aquí si son necesarios los underscores. Por ejemplo:

*final int SEGUNDOS_POR_DIA = 60 * 60 * 24;*

También habíamos visto que existen métodos especiales llamados getters y setters, para obtener y modificar respectivamente el valor de cada atributo. Sus nombres deben formarse con la palabra *get* o *set* más el nombre del atributo correspondiente. Por ejemplo:

```
class Perro {
        private int edad;
        private boolean isSick;
        public void setEdad (int e) {
                edad = e;
        }
        public int getEdad () {
                return edad;
        }
        public boolean isSick () {
                return isSick;
        }
}
```

Sin embargo, cuando el atributo es de tipo boolean, tanto el nombre del atributo como del getter deben empezar con la palabra *is* para enfatizar que se trata de un estado de “sí o no”. No existe confusión pues el método tiene paréntesis y el atributo no.

2.1.2 Manejo de métodos; sobrecarga; uso de *this;* clases internas; métodos genéricos

this y super

La palabra genérica *this* indica el objeto o clase actual, y tiene principalmente 2 usos:

- Para enfatizar en un setter o constructor cual es el atributo y cuál es el argumento, pues se recomienda que se llamen igual. Por ejemplo:

```
class Perro {
        private int edad;
        public void setEdad (int edad) {
                this.edad = edad;          // atributo = argumento
        }
}
```

- Para que un constructor pueda llamar a uno de sus hermanos. Por ejemplo, en un cronometro podemos tener un constructor que automáticamente lo arranca, y otro que además ajusta el tiempo:

```
public class StopWatch {
        long segundos;
        StopWatch(boolean start) {
                if (start)
                        arranca();
        }
        StopWatch(boolean start, long segundos) {
                if (start)
                        arranca();
                this.segundos = segundos;
        }
}
```

Pero podemos simplificar el código redundante haciendo que el constructor con más argumentos llame al otro, que es lo recomendable:

```
public class StopWatch {
        long segundos;
        StopWatch(boolean start) {
                if (start)
                        arranca();
        }
        StopWatch(boolean start, long segundos) {
                this(start);    // llama al constructor de 1 boolean
                this.segundos = segundos;
        }
}
```

Similarmente, la palabra genérica *super* indica el objeto o clase padre, y tiene también 2 usos:

- Para ejecutar la versión de la clase inmediata superior, de un método sobrescrito, por ejemplo:

```
class Persona {
        void hablar () {
                System.out.println ("soy una Persona");
        }
}
class Empleado extends Persona {
        void hablar () {
                System.out.println ("soy un Empleado");
        }
        void opinionPersonal () {
                super.hablar ();              // escribe "soy una Persona"
        }
}
```

- Para que un constructor inicialice la parte correspondiente de su clase padre, llamando al constructor de la súper clase:

```
class Persona {
        Persona (String nom, String ape) {      // constructor
                this.nom      = nom;
                this.ape      = ape;
        }
}
class Empleado extends Persona {
        Empleado (String nom, String ape, int cve) {
                super (nom, ape);          // llama al constructor padre
                this.cve      = cve;
        }
}
```

Hay que mencionar que la llamada explicita a *this ()* o a *super ()* debe ser la primera instrucción dentro del constructor, pues de otra manera tendremos un error de compilación.

Referencias

“Thinking in Java”
Bruce Eckel
Prentice Hall

“Mastering Java 2.0”
Sun Microsystems