LP00

Trabajo

- Ensayo N° 05

Nombre del profesor:

- Hancco Carpio, Rony Jordan

Nombre del alumno:

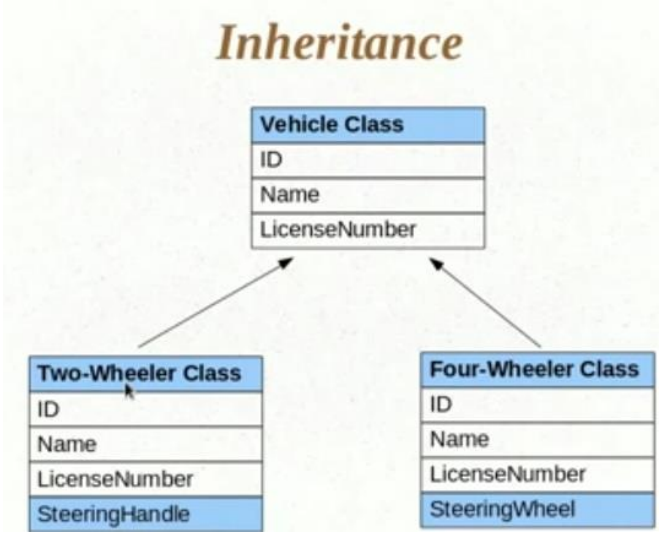
- Negrete Girano, Luis Carlo

Lima, 03 de junio del 2015

QUINTO ENSAYO SOBRE LENGUAJE DE PROGRAMACIÓN ORIENTADOS A OBJETOS

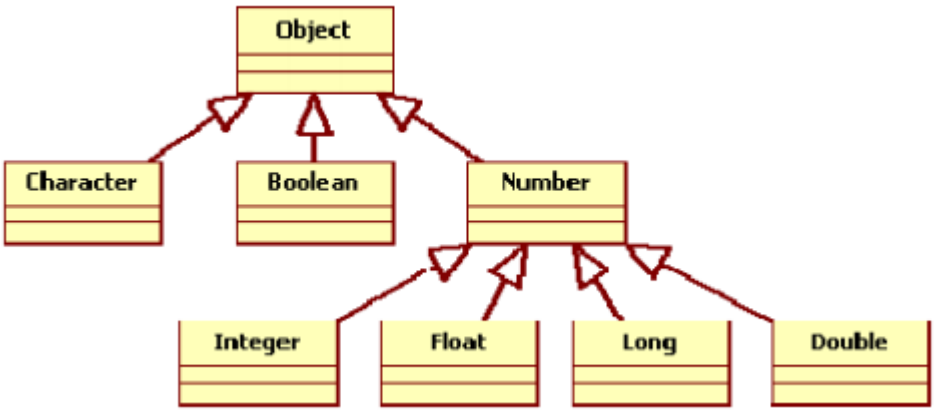
En este ensayo, se hablará sobre dos temas en particular, específicamente, sobre herencia y sobrecarga de métodos. Sin más preámbulo, pasemos a hablar sobre ellos.

En primer lugar, qué es la herencia, es una capacidad no exclusiva del lenguaje java, la cual nos permite extender clases, la cual hereda todos los atributos y los métodos declarados en la llamada “superclase”, también conocida como clase original, a la que extiende, inclusive cuando no todos sean accesibles. Cabe mencionar, que la clase nueva que surge es denominada clase derivada o subclase, observemos un gráfico para tener una idea más clara.



Observamos que los atributos de las clases son similares, esto se da gracias a la herencia. Sin embargo, debe quedar en claro que la relación no es recíproca(nótese en la dirección de las flechas), es decir, la superclase no hereda nada en lo absoluto de alguna subclase, en base a esto, podríamos decir que la herencia se puede utilizar para ahorrar líneas de código en declaraciones innecesarias de clases que tienen algunos atributos y métodos en común. En otras palabras, reutilizamos el código.

Asimismo, debemos considerar que una subclase puede convertirse en una superclase, pero ¿cómo es esto posible?; pues justamente es aquí donde entra a tallar el término de "jerarquía de herencia". La cual nos dice que una clase derivada obtenida, puede ser extendida nuevamente y así sucesivamente; esta relación es transitiva y define a lo que se denomina como "jerarquía". Un ejemplo claro de esto es que en el lenguaje Java todas las clases están relacionadas bajo una jerarquía única, tal y como se ve puede apreciar en el siguiente esquema de jerarquía de herencias de clases asociados a los tipos primitivos de java.



Ahora bien, otro punto importante a considerar, es que al aplicar la herencia, el constructor de la subclase se ve modificado, ya que este va a tener que invocar al constructor de la superclase. Así, tenemos la siguiente sintaxis en el código:

```
Syntax    accessSpecifier ClassName(parameterType parameterName, . . .)
          {
             super(parameters);
             . . .
          }
```

La lista de argumentos observada se utiliza para escoger a un constructor en específico. Puede ser vacía, en caso el constructor por omisión de la superclase es invocado. Asimismo, se debe considerar que los constructores no son heredados, y no pueden ser modificados. Su sobrecarga, tema que será explicado en breve, está restringida al conjunto de constructores declarados en una clase.

Para finalizar con el tema de herencia, veamos la sintaxis que debe tener un código al momento de extender una clase es el siguiente:

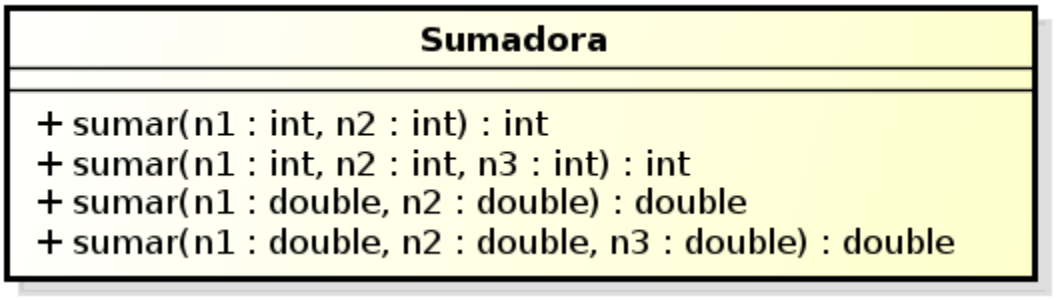
```
public class "Name-SubClass" extends "Name-SuperClass"
{

}
```

Para entender un poco mejor este tema, observar el ejemplo dentro del repositorio.

Tras finalizar con este tema, pasamos ahora a hablar brevemente del tema de “sobrecarga de métodos”, el cual es una rama del tema de polimorfismo, una manera simple de definir a la sobrecarga de métodos podría ser la creación de varios métodos con el mismo nombre pero con diferentes parámetros o argumentos. Java utiliza el número y tipo de argumentos para seleccionar cuál definición de método ejecutar.

Esto quiere decir, por ejemplo, tenemos un método “duplicar”, el cual lo declaramos más de una vez en nuestra línea de código, primero teniendo como parámetro a un entero, luego a un float, y por último a un entero y un float. Si al momento de ejecutar el método, ingresamos por ejemplo (‘5’), el cual es un entero, lo que hará nuestro IDE es buscar dentro de todos los métodos declarados, cuál de estos cumple con el requisito del dato ingresado, en este caso específico, se tomará al primero: el método declarado con parámetro entero, análogamente si se ingresa (‘5.5’) el cual es un float; sin embargo, si ingresamos (‘Joven’), el cual es un string, se producirá un error dado que este no está declarado, por lo que si deseamos arreglar este problema, simplemente declaramos nuevamente el método con el mismo nombre, pero ahora con un parámetro que acepte string. Finalmente, si tenemos dos valores de entradas, ya no se buscará en aquellos métodos que tienen un solo parámetro, sino que buscará donde hay dos parámetros y se podrá llevar a cabo el método siempre y cuando cumpla con el tipo de dato declarado, en nuestro caso (‘6’,’8.2’) sería un dato aceptado.

Otro ejemplo a citar podría ser el siguiente:



Como se puede apreciar, el método “sumar” es declarado cuatro veces y recibe diferentes parámetros; por lo tanto, según lo expuesto, el IDE, dependiendo del tipo de dato ingresado, determinará qué método satisface con los requerimientos y posteriormente realizará las acciones establecidas en el método.

De manera similar al tema anterior, para poder entender cómo es que funciona la sobrecarga, observar el ejemplo dentro del repositorio.