“UNIVERSIDAD NACIONAL DE INGENIERIA”

“Facultad de Ingeniería Industrial y de Sistemas”

Alumno: Vidal Flores, Jose Carlos

20130078D

Dirigido: HANCCO CARPIO, RONY JORDAN

Curso: LENGUAJES DE PROGRAMACIÓN ORIENTADOS A OBJETOS (ST-232 V)

“2015-I”

HERENCIA

Una característica que posee el lenguaje de programación orientada a objetos es la de herencia; esto se puede explicar cómo una capacidad que poseen las clases en java para poder extenderse a otra clase, es decir una clase llamada subclase o derivada puede incluirse dentro de otra llamada superclase o base.

Extensión de clases

Es la capacidad de las clases para poder heredar, esta capacidad les permite a las subclases heredar el comportamiento de las variables y métodos declarados en la superclase, normalmente estas subclases son una especialización de la superclase, es decir son características más particulares y los de la superclase son características más generales.

Las clases en java pueden extenderse cuantas veces se desea mediante la palabra extends, como se especifica en lo siguiente:

```
class nombredelasubclase extends nombredelasuperclase {

declaraciones de variables y métodos

}
```

Así pueden haber subclases dentro de una superclase, que a su vez puedan pertenecer a una clase mayor; si bien las extensiones de clases se puede dar cuantas veces se desee es recomendable no excederse de cinco para adquirir una buena práctica de programación. Las extensiones consecutivas de clases se les pueden denominar “clases anidadas” o “cadena de herencias”, cabe aclarar que una subclase puede tener solo una superclase pero si puede tener muchas subclases anidadas.

La relación de herencia entre clases también se aplica a los tipos que las clases definen, llamándolos “supertipos” y “subtipos”. Esto permite compatibilidad para la asignación o conversión entre tipos de referencia, así que es posible asignar una referencia a un objeto de un subtipo a una variable cuyo tipo es el supertipo.

La herencia tiene el efecto de incrementar el número de entornos que necesitan ser revisados para verificar que una variable o método está en un entorno, y si es accesible. El orden de la búsqueda es como sigue, y termina cuando el identificador es hallado:

1. Verificar el entorno local y cualquier entorno anidado.
2. Verificar el entorno de clase.
3. Verificar cada entorno de superclases, a lo largo de la cadena de herencia.

Si en algún otro entorno se encuentra una declaración del identificador de una variable, se reporta un error. Si no se encuentra una declaración en todos los entornos, entonces también se reporta un error de declaración de la variable. Si varias variables en diferentes entornos son declaradas con el mismo identificador, entonces la primera declaración encontrada es utilizada, es decir, aquella en el entorno más cercano. Las declaraciones en entornos anidados se dice están ocultas (hidden o shadowed) del entorno anidado.

Palabras Clave private, protected, y Herencia

Los modificadores de acceso influyen en las herencias de clase, si bien la herencia se define como la capacidad de adquirir las características y el comportamiento de la superclase, es decir sus atributos y métodos, estos pueden verse afectados según el tipo de modificador de acceso que se le da al atributo o método.

A continuación se verá cómo es que influye cada uno de ellos:

- La declaración private ofrece una garantía de que ninguna otra clase, incluyendo las subclases, pueden acceder al método o variable.

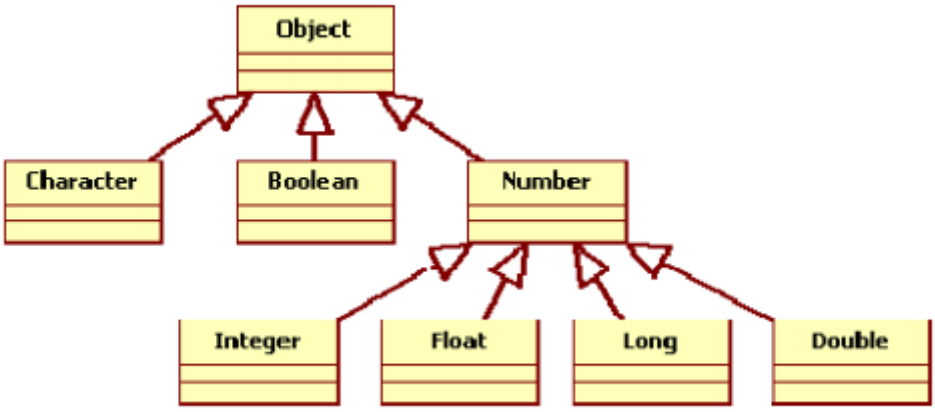
- La declaración protected permite que una subclase acceda directa y eficientemente a su superclase, incluyendo si estos están en otro package.

- La declaración public hace accesible a la subclase y a todo lo demás.

- El acceso por omisión es igual al acceso público si una subclase se encuentra en el mismo paquete, e igual al acceso privado si no.

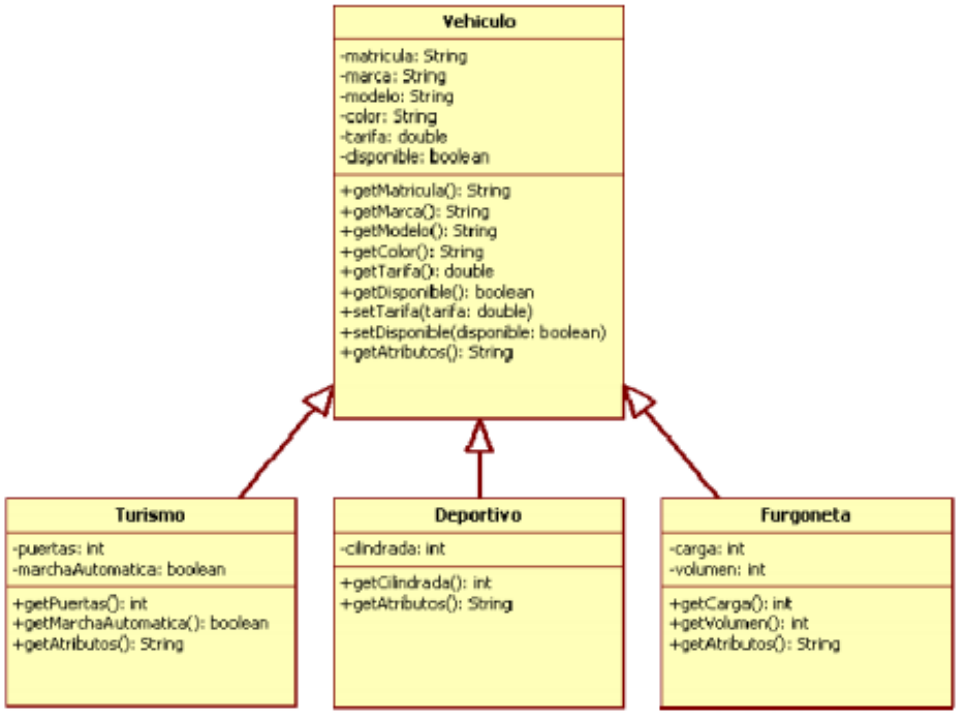
Jerarquía de Herencia

Como se ha visto para poder extender una clase de otra se hace uso de la palabra extends haciendo referencia a la superclase; en java todas las clases creadas son subclases de la clase Object a diferencia que no se necesita hacer referencia con la palabra extends, todas manejan el concepto de herencia debido a tal razón. A continuación se mostrara el esquema de la jerarquía en java:



POLIMORFISMO

Para entender mejor esto haremos uso del siguiente ejemplo:



Como se puede ver en el diagrama de clases, hay tres subclases que heredan de la superclase Vehículo, además se puede observar que en las subclases Turismo, Deportivo y Furgoneta cada uno posee su propio método con el nombre getAtributos() del mismo modo que la superclase Vehículo, entonces se dice que existe una sobrecarga de métodos; entonces ¿Cómo llamar a un método especifico de nombre getAtributos()?, para realizar esta tarea se debe instanciar una variable objeto; dependiendo del tipo de constructor que se usó para instanciarlo hace uso del método perteneciente a ese constructor.

Ejemplo: Si deseo usar el método getAtributos() de la clase Turismo; debo instanciar una variable de la siguiente forma “ Turismo miTurismo= new Turismo(); así al hacer uso de miTurismo.getAtributos() este se referirá al método perteneciente a la subclase Turismo.

Lo que sucede es que los métodos de la subclase con el mismo nombre de un método de la superclase, estos lo sobrescriben de tal forma que al crear un objeto perteneciente a una subclase, use el método de esa subclase y no el de la superclase.

Compatibilidad de tipos

En una relación de tipo herencia, un objeto de la superclase puede almacenar un objeto de cualquiera de sus subclases, es decir cualquier referencia de la superclase puede contener una instancia de la misma superclase o de cualquiera de sus subclases.

Conversión ascendente de tipos

Cuando un objeto se asigna a una referencia distinta de la clase a la que pertenece, se hace conversión de tipos. Java permite asignar un objeto a una referencia de la clase base, es decir a un objeto del tipo de la superclase se le puede referenciar un objeto de una de sus superclases, esto es válido en java. A esta conversión ascendente de tipos se le denomina “upcasting”.

Conversión descendente de tipos

Si una instancia de la clase base almacena una referencia a un objeto de un de sus clases derivadas, entonces es posible hacer una conversión descendiente de tipos, que se denomina “dowcasting”.

Para esto se debe hacer uso del nombre de la clase a la que se desea convertir, ejemplo:

```
Vehiculo miVehiculo = new Turismo();

Tursimo miNuevoTurismo = (Turismo) miVehiculo;
```