UNIVERSIDAD INTERAMERICANA DE PANAMA

FACULTAD DE INGENIERIAS

INGENIERIA ECONOMICA

PROF. LEONARDO ESQUEDA

INVESTIGACIÓN No.1

LUIS VILLARREAL 8-800-1974

ADONIS AMAYA 8-928-639

ROGER SILVA 8-951-948

LUIS BONILLA 8-878-842

ARIADNA ACEVEDO 8-712-1473

JULIO GATICA PE-10-585

JORGE URBANO 8-923-139

12 DE JULIO DE 2018

Las clases son la base de la Programación Orientada a Objetos. Una clase es una plantilla que define la forma de un objeto; en ella se agrupan datos y métodos que operarán sobre esos datos. En java, una clase se define con la palabra reservada class. Por convención, se declaran primero las variables (atributos) miembro de la clase y luego las declaraciones e implementaciones de métodos. Al diseñar una clase es conveniente tomar en cuenta lo siguiente:

• En Java no existen variables ni métodos globales. Todas las variables y métodos deben pertenecer a una clase.

• Cuando una clase extiende a otra hereda todos sus atributos y métodos.

• En Java no existe la herencia múltiple.

• Object es la base de toda la jerarquía de clases de Java. Si al definir una clase no se especifica la clase que extiende, por default deriva de Object.

**CLASES ABSTRACTAS**.

Una clase que declara la existencia de métodos pero no la implementación de dichos métodos, se considera una clase abstracta. Una clase abstracta puede contener métodos no abstractos pero al menos uno de los métodos debe ser abstracto. Para declarar una clase o método como abstractos, se utiliza la palabra reservada abstract.

Una clase abstracta no se puede instanciar(es decir no se pueden volver en objetos) pero si se puede heredar y las clases hijas serán las encargadas de agregar la funcionalidad a los métodos abstractos.

**Ventajas de su uso**

En primer lugar, evitan que los usuarios de la clase puedan crear objetos de la misma, como dice la definición de clase abstracta. En segundo lugar, permiten crear interfaces que luego deben ser implementados por las clases que hereden de la clase abstracta.

Es evidente que una clase abstracta, al no poder ser instanciada, no tiene sentido hasta que una serie de clases que heredan de ella, la implementan completamente y le dan un significado a todos sus métodos.

**INTERFACES**

Una interface es una variante de una clase abstracta con la condición de que todos sus métodos deben ser abstractos. Si la interface va a tener atributos, éstos deben llevar las palabras reservadas static final y con un valor inicial ya que funcionan como constantes por lo que, por convención, su nombre va en mayúsculas.

Una clase implementa una o más interfaces (separadas por comas) con la palabra reservada implements con el uso de interfaces se puede “simular” la herencia múltiple que Java no soporta.

**ÁMBITO DE UNA VARIABLE**

El ámbito de una variable define su alcance de uso, o lo que es lo mismo, en que secciones de código una variable estará disponible. Fuera de este ámbito, una variable no podrá ser accedida (no existe).

**Tipos de ámbito**

En Java tenemos tres tipos de ámbito que pueden aplicar a una variable:

* Local
* Global
* Estático

**Variables de ámbito local**

Las variables de ámbito local, o de bloque, son aquellas que sólo pueden ser accedidas desde el bloque de código en el que han sido declaradas (que no inicializadas).

## Variables de ámbito global

Las variables de ámbito global, o de instancia, son aquellas que pertenecen a cada instancia concreta de la clase donde han sido declaradas, y dependiendo del modificador de visibilidad usado (más tarde hablaremos sobre esto) podrían ser sólo accedidas desde la propia instancia a la que pertenecen.

## Variables estáticas

Las variables estáticas, o de clase, son aquellas que pertenecen a la propia clase donde han sido declaradas, y dependiendo del modificador de visibilidad usado podrían ser sólo accedidas desde la propia clase en la que han sido declaradas.

**Reduciendo el ámbito**

La siguiente frase debes grabarla a fuego en tu cabeza desde este mismo momento.

Toda variable debe ser declarada en el ámbito más reducido posible.

Cuando una variable es declarada con un ámbito más extenso del necesario, es sólo cuestión de tiempo que usemos esa variable de forma incorrecta, provocando confusión e invitándonos a introducir bug's (si es que el ámbito incorrecto no es en sí mismo ya un bug...).

El uso adecuado del ámbito de las variables es tan importante que, además de producir código más limpio, legible y mantenible, puede ahorrarnos muchas (MUCHAS) horas de depuración intentando corregir errores que son consecuencia de su mal uso. El ajuste de la usabilidad y la visibilidad siguen un camino paralelo, así que intenta recorrerlos sin perder de vista los demás. Tu depurador te lo agradecerá.

**MÉTODOS**

En la programación, un método es una subrutina cuyo código es definido en una clase y puede pertenecer tanto a una clase, como es el caso de los métodos de clase o estáticos, como a un objeto, como es el caso de los métodos de instancia. Análogamente a los procedimientos en lenguajes imperativos, un método consiste generalmente de una serie de sentencias para llevar a cabo una acción, un juego de parámetros de entrada que regularán dicha acción o, posiblemente, un valor de salida (o valor de retorno) de algún tipo.

La diferencia entre un procedimiento (generalmente llamado función si devuelve un valor) y un método es que este último, al estar asociado con un objeto o clase en particular, puede acceder y modificar los datos privados del objeto correspondiente de forma tal que sea consistente con el comportamiento deseado para el mismo. Así, es recomendable entender a un método no como una secuencia de instrucciones sino como la forma en que el objeto es útil (el método para hacer su trabajo). Por lo tanto, podemos considerar al método como el pedido a un objeto para que realice una tarea determinada o como la vía para enviar un mensaje al objeto y que éste reaccione acorde a dicho mensaje

**Tipos de métodos**

Como ya se mencionó, los métodos de instancia están relacionados con un objeto en particular, mientras que los métodos estáticos o de clase (también denominados métodos compartidos), están asociados a una clase en particular. En una implementación de constructores, siendo estos métodos de instancia especiales llamados automáticamente cuando se crea una instancia de alguna clase. En Java y C++ se distinguen por tener el mismo nombre de las clases a la que están asociados. Lenguajes como Smalltalk no requieren constructores ni destructores.

Los métodos de acceso son un tipo de método normalmente pequeño y simple que se limita a proveer información acerca del estado de un objeto. Aunque introduce una nueva dependencia, la utilización de métodos es preferida a acceder directamente a la información para proveer de una nueva capa de abstracción (programación orientada a objetos). Por ejemplo, si una clase que modela una cuenta bancaria provee de un método de acceso "obtenerBalance()" en versiones posteriores de la clase se podría cambiar el código de dicho método substancialmente sin que el código dependiente de la clase tuviese que ser modificado (un cambio sería necesario siempre que el tipo de dato devuelto por el método cambie). Los métodos de acceso que pueden cambiar el estado de un objeto son llamados, frecuentemente, métodos de actualización o métodos de mutación; a su vez, los objetos que proveen de dichos métodos son denominados objetos mutables.

**CONSTRUCTORES**

Un Constructor es una función, método, etc, de las clases, la cual es llamada automáticamente cuando se crea un objeto de esa clase.

Por ser métodos, los constructores también aceptan parámetros. Cuando en una clase no especificamos ningún tipo de constructor, el compilador añade uno público por omisión sin parámetros, el cual NO hace nada.

**Características de los Constructores**

1. Un constructor, tiene el mismo nombre de la clase a la cual pertenece.

2. No puede ser Heredado.

3. No retorna ningún valor (Ni void), por lo cual no debe especificarse ningún tipo de dato.

4. Debe declararse como public, sólo en casos realmente extraordinarios será de otro tipo.

Un método, si va a devolver algún valor, éste debe especificarse cuando se hace la definición del mismo, para el caso, el método Distancia, devuelve el valor de la distancia, es por ello que lo especificamos como public double. Además que, cuando realizamos la llamada de ese método, es en ese preciso lugar que aparecerá el valor que devuelve. Por otra parte, los atributos, son de acceso privado, lo cual implica que, sólo podemos acceder a ellos, en la clase Coordenadas.

**Sobrecarga**

Cuando en una clase, se definen un mismo método, con diferente número de parámetros, o bien con el mismo número de parámetros pero diferenciándose en la definición, en la cual, al menos un parámetro sea de diferente tipo, cuando esto sucede, se dice que el método está sobrecargado.

**FINALIZADORES**

Los finalizadores son métodos que vienen a completar la labor del garbage collector. Un finalizador es un método que se llama automáticamente cuando se va a destruir un objeto (antes de que la memoria sea liberada de modo automático por el sistema).

Se utilizan para ciertas operaciones de terminación distintas de liberar memoria (por ejemplo: cerrar ficheros, cerrar conexiones de red, liberar memoria reservada por funciones nativas, etc.).

Hay que tener en cuenta que el garbage collector sólo libera la memoria reservada con new. Si por ejemplo se ha reservado memoria con funciones nativas en C (por ejemplo, utilizando la función malloc()), esta memoria hay que liberarla explícitamente utilizando el método finalize().

Un finalizador es un método de objeto (no static), sin valor de retorno (void), sin argumentos y que siempre se llama finalize(). Los finalizadores se llaman de modo automático siempre que hayan sido definidos por el programador de la clase. Para realizar su tarea correctamente, un finalizador debería terminar siempre llamando al finalizador de su super-clase.

Tampoco se puede saber el momento preciso en que los finalizadores van a ser llamados. En muchas ocasiones será conveniente que el programador realice esas operaciones de finalización de modo explícito mediante otros métodos que él mismo llame.

El método System.runFinalization() “sugiere” a la JVM que ejecute los finalizadores de los objetos pendientes (que han perdido la referencia). Parece ser que para que este método se ejecute, en Java 1.1 hay que llamar primero a gc() y luego a runFinalization().ACIO