Capítulo 1

Primeros Pasos – Declaraciones - Control de Acceso

Java World

¿Querés ser un SCJP?

Revista Digital - Hecha por estudiantes para estudiantes

Bienvenidos

Si estás leyendo esto es porque te encuentras interesado en rendir la certificación de Sun SCJP (Sun Certified Java Programmer), o al menos tienes curiosidad al respecto.
El texto que leerán a continuación es un resumen que realizo por cada capítulo del libro de "Sun certified Programmer for Java 6 *Study Guide*", de *Katy Sierra y* Bert Bates. No fue ideado como alternativa para no comprar dicho libro, es más, recomiendo explícitamente que adquieran el libro.
¿Por qué adquirirlo, si me estás facilitando los resúmenes?

- Este documento es un resumen, pero a su vez una síntesis. Conceptos que para mí son conocidos, puede que para otros lectores no lo sean, y viceversa.
- Este documento contiene conceptos que han sido aportados pura y exclusivamente por el creador.
- El libro explica en forma más profunda todos los temas que se verán a continuación.
- El libro contiene una serie de exámenes al final de cada capítulo, los cuales debo decir son prácticamente indispensables, y muy tramposos en algunos casos.

Recomendaciones

Si no deseas leer mis recomendaciones, puedes pasar al siguiente capítulo “Utilizando Java desde la consola de comandos”, y comenzar a prepararte para ser un SCJP.

Los temas que se tratan pueden parecer triviales, pero llegan a ser complejos cuando se los mezcla en un todo. Es muy importante, no dar nada por sentado, y leer hasta el último párrafo (a veces, creemos que sabíamos algo, cuando realmente, nos damos cuenta que no era del todo cierto).
Es muy importante lo que sabes, pero también lo que no sabes. Una manera muy buena de realizar esta verificación es con los exámenes provistos en el libro “Sun certified Programmer for Java 6 *Study Guide*”, de *Katy Sierra y* Bert Bates. La otra (y recomiendo ambas) es escribir código. Hola mundo, 2 + 2, cualquier cosa. A cada tema que están estudiando hagan un sencillo programa, y piensen para sí mismos, cuáles serían los casos más raros que se les pueden llegar a presentar, y pruébenlos. Por estas razones, no es obligatorio, pero sería recomendable que supieran programar en Java a un nivel básico. Simplemente conocer la sintaxis de java.
Olvídense de su IDE, su computadora necesita un descanso. Utilicen la consola de comandos para compilar y ejecutar sus programas.
Al no utilizar una IDE, están propensos a cometer errores, que esta no les permitiría, o les advertiría en el momento. Cuando vayan a compilar, sabrán si algo no le agrada al compilador.
¿Pero, cómo uso es cosa que llamas “consola”?
Bien, tienes razón. Si no sabes cómo hacerlo, puede ser un poco frustrante. Para aquellos que sepan:

- Instalar la JDK
- Establecer el Path a la instalación de los binary de la JDK
- Compilar un .java desde consola
- Ejecutar un .class o .jar desde consola

Los invito a que empiecen con el capítulo de Identificadores, los que no, que les parece si comenzamos a familiarizarnos con la consola.

Utilizando java desde la consola de comandos

Vamos a hacer una diferencia entre dos componentes de Java:

- JRE
- JDK

JRE (Java Runtime Environment): Set de librerías que permiten correr una JVM en la pc para poder ejecutar un programa escrito en Java (bytecodes).
JDK (Java Development Kit): Como mencionamos arriba, los bytecodes son un lenguaje intermedio entre el código fuente y el código máquina, el cual es interpretado para la JVM. Para generar este código, necesitamos de otras funcionalidades que no contiene la JRE. Además, en la JDK encontraremos la API de Java. Un set de librerías predefinidas para que utilicemos en nuestros programas.

Los pasos a continuación representan una guía simple para poder utilizar la consola de comandos, simplemente para que puedan hacer ejercicios sobre los temas estudiados. De todas maneras, la consola de comandos se tratará más a fondo en entregas posteriores.

Instalando la JDK y configurando el entorno

1. **Si ya tienes una version de la JDK igual o superior a la 1.6.0 puedes pasar al paso 3**. Vamos a obtener la JDK desde el sitio oficial de sun `http://java.sun.com`.
   El link directo: `http://java.sun.com/javase/downloads/widget/jdk6.jsp`
   Lo que nosotros buscamos es la JDK de Java SE.
2. Una vez que hayas desccargado la JDK es tiempo de instalarla.
3. Dijimos que ibamos a utilizar la consola de comandos, el problema es que la JDK no se almacena en la variable PATH de windows, esto quiere decir que aun no podemos ejecutar los comandos de Java, salvo que nos encontremos en el directorio de los mismo `\[Directorio instalación JDK]\Bin\`
   En Windows, lo que vamos a hacer primero es rastrear este Path. Generalmente se encuentra en: `C:\Archivos de programa\Java\jdk1.6.0_XX\bin\.` Si no puedes dar con este Path, puedes buscar el archivo `javac.exe`. Una vez que tengas el directorio, guardalo en un txt.
4. Botón derecho sobre “Mi Pc” -> Propiedades. En la solapa de “Opciones Avanzadas”, buscamos la opción que dice “Variables de entorno”. En el segundo recuadro, buscamos por una variable que se llame `PATH`. Doble click sobre la misma, y abrirá la edición. Ahora, agreguen el siguiente texto sin eliminar lo que hubiera antes:
   `;”[Path copiado al txt]”;` donde Path copiado al txt es el path de los bin de la JDK (ponemos entre comillas el código por si el path contiene algún espacio).
   Luego acepta los cambios.
5. Ve a “Menu Inicio” -> “Ejecutar” -> y tipea “cmd”.
   Se debería de abrir una consola de comandos. Ahora vamos a probar que la asignación del Path fue correcta. Para ello tecleamos el comando `javac`.
   Debería de aparecerte algo como lo siguiente:

```
C:\>javac
Usage: javac <options> <source files>
where possible options include:
```

Si recibiste un mensaje que comienza como este, significa que la instalación fue exitosa, de lo contrario, vuelve a rever los pasos anteriores.

Compilando y ejecutando

Cuando escriban código en Java, cada archivo fuente contendra la extensión **java** (dificil de recordar, no?!).
En cambio, los archivos compilados contienen la extensión **class**. También pueden estar dentro de un **jar**, que no es más que un zip que los agrupa en un solo archivo (tiene otras particularidades).

Nosotros vamos a centrarnos en la creación de los **.java** y los **.class**. Los jar los dejaremos fuera por ahora.
Esta sección tiene en cuenta que el lector tiene conocimientos básicos para manejarse en la consola de comandos.

Vamos a crear nuestra primer app. Java:

1. Crea un archivo llamado **HolaJavaWorld.java** (en el directorio que quieras), y copia el código a continuación dentro del archivo.

```
public class HolaJavaWorld {
  static public void main(String[] args) {
    System.out.println("Hola Java World!");
  }
}
```

2. Ejecuta la consola de comandos, y posiciona el Prompt sobre el directorio donde creaste el archivo (Tip: si haces botón derecho sobre el directorio, verás una opción que dice “símbolo de sistema”, esto te habre una consola de comandos posicionada en dicho directorio).
3. Ejecuta **javac -g HolaJavaWorld.java** . En esta parte es donde le pedimos al compilador de Java que genere el Bytecode de dicho archivo fuente. Si llegase a existir algún error de compilación, es en este punto que se nos informa de tal.
4. Ejecuta **java HolaJavaWorld**. Aqui creamos una instancia de la JVM sobre la cual correrá nuestra aplicación, en la cual veremos un grandioso “Hola Java World!”.

Con estos conocimientos básicos de la consola, y un poco de conocimiento del lenguaje Java, ya te encuentras en condiciones de generar código para probar tus conocimientos sobre lo que vayas leyendo.

System.out.println(“Buena Suerte”);

Identificadores

Los identificadores permiten que podamos identificar una variable por un nombre y/o conjunto de caracteres alfanuméricos (generalmente, describen el contenido).
Para los identificadores, existen una serie de convenciones.

Identificadores legales

Los identificadores legales se refieren a aquellos que cumplen con las normas mínimas de manera que el compilador no genere un error en tiempo de compilación.
Las características que deben de cumplir los identificadores son:

- Debe estar compuesto de caracteres Unicode, números, `$` o `_`
- Debe comenzar con un carácter Unicode, `$` o `_`
- Luego del primer carácter, puede contener cualquier conjunto de los nombrados en el primer punto
- No tienen un límite de largo
- No es posible utilizar palabras reservadas de Java
- Son case-sensitive (distingue mayúsculas de minúsculas)

Expresión regular de la composición de un identificador legal: `[\w$_]([\w\d$_]*)`

Convenciones de código de Java Sun

Sun creó un conjunto de estándares de codificación para Java, y publicó estos estándares en un documento titulado “Convenciones de código de Java” el cual puede ser encontrado en java.sun.com.
Las características que debe de cumplir los identificadores son:

- Clases e interfaces
  - La primer letra debe ser mayúscula
  - Utiliza nomenclatura camelCase
  - Para las clases, los nombres deben de ser sustantivos
  - Para las interfaces, los nombres deben de ser adjetivos
- Métodos
  - La primer letra debe ser minúscula
  - Utiliza nomenclatura camelCase
  - Los nombres deben conformarse por el par verbo + sustantivo
- Variables
  - La primer letra debe ser minúscula
  - Utiliza nomenclatura camelCase
  - Es recomendable utilizar nombres con un significado explícito, y en lo posible, cortos
- Constantes
  - Todas las letras de cada palabra deben estar en mayúsculas
  - Se separa cada palabra con un `_`

Estándares para nombres de JavaBeans

Al utilizar estándares para nombres, se garantiza que las herramientas puedan reconocer los componentes realizados por cualquier desarrollador.
Las características que debe de cumplir los identificadores son:

- Atributos
  - Ambos (getters y setters) se conforman del prefijo especificado + el nombre del atributo con su primer letra en mayúscula
  - Getters
    - Si la propiedad no es de tipo **`boolean`** el prefijo debe ser **`get`**
    - Si la propiedad es de tipo **`boolean`** el prefijo debe ser **`is`**
    - Deben de ser **`public`**, no recibir ningún argumento, y devolver un valor del mismo tipo que el setter para la propiedad.
  - Setters
    - El prefijo debe ser **`set`**
    - Deben de ser **`public`**, recibir un parámetro del mismo tipo que la propiedad, y devolver **`void`**
- Listeners
  - Los métodos que agreguen un listener deben de comenzar con el prefijo **`add`**
  - Los métodos que quiten un listener deben de comenzar con el prefijo **`remove`**
  - El tipo del listener a agregar/quitar debe de ser pasado como argumento
  - Deben de terminar con el sufijo **`Listener`**
- Archivos de código fuente
  - Solo puede haber una clase **`public`** por cada archivo
  - Los comentarios pueden aparecer en cualquier línea
  - Si existe una clase **`public`**, el nombre de la misma debe corresponderse con el nombre del archivo
  - Si la clase pertenece a un **`package`**, esta sentencia debe ser incluida como primera línea
  - Si hay **`import`**, estos deben de ser incluidos entre la declaración del **`package`** y de la clase (**`class`**)
  - Los **`import`** y **`package`** aplican a todas las clases dentro del archivo
  - Un archivo puede tener más de una clase, siempre que esta no sea **`public`**
  - Los archivos que no tengan una clase **`public`** pueden contener cualquier nombre

Declaraciones de clases y modificadores de acceso

El acceso significa visibilidad. Dependiendo de los modificadores aplicados a la clase, atributos, o propiedades, será que estos podrán ser accedidos o no por otros.

- Modificadores de acceso
  - **public**
  - **private**
  - **protected**
  - Default (**package**). Si no se especifica un modificador, este es el que toma por defecto
- Modificadores de no acceso
  - **strictpf**
  - **final**
  - **abstract**

Modificadores de acceso a clases

package (default)

Nivel de visibilidad: clase visible para todas las clases que se encuentren dentro del mismo **package**.
Este no debe especificarse (modificador por defecto).

En algunas preguntas con lógica compleja, primero es recomendable verificar los modificadores de acceso. Si detectas alguna violación de los mismos, directamente elige la opción “Compilation Fails (Fallas de compilación)”.

public

Nivel de visibilidad: clase visible para todas las clases de todos los **package**.
Para especificar este modificador se antepone a la palabra clave **class**, la palabra **public**. De todas maneras, si la clase es utilizada por otra clase que no se encuentra en el mismo **package**, es necesario realizar el **import** de la misma.

Modificadores de no acceso

Estos modificadores pueden combinarse con aquellos que si modifican el acceso.

strictpf

No es necesario saber para la certificación como es que funciona **strictpf**. Lo único que es necesario, es saber que:

- Es una palabra reservada
- Puede modificar solo clases y métodos
- Hacer una clase **strictpf** significa que todos los métodos que la conforman cumplen con la norma IEEE 754 para punto flotante

final

El modificador **`final`** indica que una clase no puede ser heredada o redefinida por una subclase.

abstract

Características de una clase **`abstract`**:

- No puede ser instanciada (**`new`**), solo puede ser heredada.
- Si un método de la clase es **`abstract`**, esto fuerza a que la clase completa sea **`abstract`**.
- No todos los métodos de una clase **`abstract`** tienen que ser abstractos.
- Los métodos **`abstract`** no llevan cuerpo (no llevan los caracteres **`{}`**).
- La primer subclase concreta que herede de una clase **`abstract`** debe implementar todos los métodos de la superclase.

abstract y final

Es posible combinar estos dos modificadores de no acceso, pero… si interpretamos el funcionamiento de cada uno nos dicen que:

- abstract: obliga a que la subclase defina el cuerpo de los métodos.
- final: previene que la clase sea heredada o redefinida.

Un modificador se opone al otro, de manera que, aunque no recibamos un error de compilación, la clase declarada de esta manera es inutilizable.

Declaración de interfaces

Cuando creamos una interfaz, lo que estamos diciendo es lo que la clase deberá de poder hacer, pero no como lo hará. Una interfaz también es conocida como **contrato**.
Características de una interfaz:

- Todos los métodos de una interfaz son implícitamente **`public abstract`**, no es necesario especificarlo en la declaración del mismo.
- Todos los atributos de una interfaz son implícitamente constantes (**`public static final`**), no es necesario especificarlo en la declaración del mismo.
- Todas las variables de una interfaz deben de ser constantes (**`public static final`**).
- Los métodos de una interfaz no pueden ser: **`static`**, **`final`**, **`strictfp`** ni **`native`**.
- Una interfaz puede heredar (**`extends`**) de una o más interfaces.
- Una interfaz no puede heredar de otro elemento que no sea una interfaz.
- Una interfaz no puede implementar (**`implements`**) otra interfaz.
- Una interfaz debe ser declarada con la palabra clave **`interface`**.
- Los tipos de las interfaces pueden ser utilizados polimórficamente.
- Una interfaz puede ser **`public`** o **`package`** (valor por defecto).
- Los métodos toman como ámbito el que contiene la interfaz.

IMPORTANTE: cuando se menciona el implícitamente, se refiere a que si no se especifican estos son los modificadores que toma, pero se pueden especificar todos, ninguno, o algunos, siendo declaraciones semejantes, por ejemplo:

```
public interface Dibujable {
    int CONST = 1;
    public int CONST = 1;
    public static int CONST = 1;
    public final int CONST = 1;
    static int CONST = 1;
    final int CONST = 1;
    static final int CONST = 1;
    public static final int CONST = 1;

    void doCalculo ();
    public void doCalculo ();
    abstract void doCalculo ();
    public abstract void doCalculo ();
}
```

Declaración de métodos y atributos

Un método puede contener modificadores de acceso y de no acceso. Además, posee más combinaciones que la definición de clases. Dado que ambos, métodos y atributos se declaran de manera semejante, se especifican en un mismo apartado.

Modificadores de acceso

Los modificadores disponibles son:

- `public`
- `protected`
- `package`
- `prívate`

Con estos modificadores tenemos dos conceptos que entender:

- Cuando un método de una clase puede acceder a un miembro de otra clase.
- Cuando una subclase puede heredar un miembro de su superclase.

El primer caso se da cuando un método de una clase intenta acceder a un método o atributo de otra clase, valiéndose del operador `.` para invocar un método, u obtener una variable.
El segundo caso se da cuando una clase hereda de otro, permitiendo que la subclase acceda a los métodos y atributos de la superclase, a través de la herencia (`extends`).

public

Cuando un atributo o método es declarado como `public`, significa que cualquier clase en cualquier `package` puede acceder a dicho miembro.

private

Cuando un atributo o método es declarado como `private`, solo puede ser referenciado por la misma clase (ni siquiera por una subclase). Incluso si una subclase declarara un atributo o método con el mismo nombre, solo estaría generando un nuevo miembro, no sobrescribiéndolo, como puede llegar a ser malinterpretado.

package (default)

Cuando un atributo o método es declarado como `package`, solo puede ser referenciado por cualquier clase que se encuentre dentro del mismo `package`, por medio de herencia y referencia.

protected

Cuando un atributo o método es declarado como protected, este se comporta de la siguiente manera:

- Una variable declarada como `protected`, se comporta como un modificador de tipo `package` en el paquete donde se encuentra la clase que declaró el miembro.
- Una variable declarada como `protected`, solo puede ser accedida a través de la herencia (no referencia), cuando se encuentra fuera del `package`.

Esto quiere decir que, un miembro declarado como `protected` es visible para todas las clases dentro del `package` donde fue declarado (ya sean subclases de la misma o no) a través de referencia y herencia, y solo disponible a través de la herencia para las subclases que se encuentran fuera del `package`. Para el resto de las clases, se comporta como un modificador prívate (no subclases

fuera del **package** de declaración).

Modificadores de acceso en variables locales

No es posible aplicar ningún modificador de acceso a una variable local. Solo es posible aplicar un modificador de no acceso: **final**.

Modificadores de no acceso

Dentro de los modificadores de no acceso, se agregan varios de los 2 ya conocidos a la lista para modificadores de miembros.

Modificador de no acceso en métodos – final

El modificador **final** fuerza a que un método no pueda ser redefinido por una subclase.

Modificador de no acceso en atributos – final

En el caso de un modificador final para un atributo, pueden darse 2 casos:

- Si el atributo es un valor primitivo, impide de que modifique su valor (un valor primitivo es por ejemplo un **int**).
- Si el atributo es una referencia, impide que se modifique la referencia, pero no el estado de la misma.

Modificador de no acceso en métodos – abstract

El modificador **abstract** en un método indica que el mismo se ha declarado en el contrato, pero no se ha especificado su comportamiento (cuerpo del método).

Al especificar al menos 1 método como **abstract**, se fuerza a que la clase se declare como tal, por el contrario, una clase declarada como **abstract** puede no contener ningún método de este tipo.

Modificador de no acceso en métodos – synchronized

El modificador **sinchronized** indica que el método solo puede ser accedido por un **Thread** a la vez.
Este modificador puede utilizarse en conjunto con cualquiera de los 4 modificadores de acceso.

Modificador de no acceso en métodos – Native

El modificador **native** indica que el código es dependiente de la plataforma, generalmente en código C. Solo es necesario saber para la certificación que un modificador de no acceso **native** solo puede ser aplicado a un método.
El método debe especificarse como si fuera abstracto (sin los caracteres **{}**, y directamente con **;**)

Modificador de no acceso en métodos – strictfp

Al igual que para una clase, el modificador de no acceso **strictfp** indica que los caracteres de punto flotante utilizan la norma IEEE 754.

Métodos con listas de argumentos variables

A partir de Java 5.0, es posible crear métodos que reciban una cantidad de argumentos variables.

- Pueden ser de tipo primitivo o un objeto.

- La definición de var-args se hace: **tipoDato**… **nombreVariable**.
- Solo se puede especificar un solo parámetro de tipo var-args en la firma del método.
- El parámetro var-args debe de ser el último especificado en la firma del método.
- La sintaxis de estos es: TipoDeDato… nombreVariable.

Al llamar a un método sobrecargado o sobre escrito, primero se verifica si la llamada concuerda con algún método que no tenga argumentos variables, si no se encuentra ninguno de estos, se verifica con los métodos de argumentos variables.

```
public class PruebaParametrosVariables {
  static public void haceAlgo(int val) {
    System.out.println("Soy haceAlgo con un parámetro int");
  }

  static public void haceAlgo(int val1, int val2) {
    System.out.println("Soy haceAlgo con dos parámetros int");
  }

  static public void haceAlgo(int... val) {
    System.out.println("Soy haceAlgo con parámetros variables");
  }

  static public void main(String[] args) {
    PruebaParametrosVariables.haceAlgo(1);
    PruebaParametrosVariables.haceAlgo(2, 3);
    PruebaParametrosVariables.haceAlgo(4, 5, 6);
    PruebaParametrosVariables.haceAlgo(4, 5, 6, 7);
  }
```

```
Soy haceAlgo con un parámetro int
Soy haceAlgo con dos parámetros int
Soy haceAlgo con parámetros variables
Soy haceAlgo con parámetros variables
```

Declaración de constructores

En Java, cada vez que un objeto es instanciado, esto se logra a través de un constructor, el cual fue invocado con un **new**.

- Toda clase tiene al menos un constructor. Si este no se especifica, existe un constructor implícito.
- Si se especifica explícitamente al menos un constructor, el constructor implícito deja de existir.
- Un constructor se define como un método, con la salvedad de que jamás puede devolver ningún parámetro.
- Puede utilizar cualquiera los modificadores de acceso (**public**, **private**, **protected**, **package**).
- El nombre del método debe de ser siempre igual al de la clase.
- Puede existir más de un constructor. Esto se logra utilizando la sobrecarga de métodos.
- Al igual que no pueden existir dos métodos iguales, no pueden existir dos constructores iguales.
- No pueden ser: **static**, **final** ni **abstract**.
- Pueden contener en su definición como parámetro cualquier tipo primitivo, objeto, o incluso var-args.

Declaración de variables

Se dividen en dos grupos:

- Primitivas: estas pueden ser cualquiera de los tipos contenidos en la **tabla 1**
- Referencias: se utilizan para referenciar un objeto. Estas son declaradas para ser de un tipo específico, y ese tipo no podrá cambiar jamás

Declaración de variables primitivas y rango de variables

Estas pueden ser declaradas como:

- Variables de clase (**static**)
- Variables de instancia (objeto)
- Parámetros
- Variables locales

Tabla 1

*Tipo	Bits	*Bytes	Minimum range	Maximum range
byte	8	1	-2^7	(2^7)-1
short	16	2	-2^15	(2^15)-1
char	16	2	0	2^16
int	24	4	-2^23	(2^23)-1
long	32	8	-2^31	(2^31)-1
float	16	4	-2^15	(2^15)-1
double	32	8	-2^31	(2^31)-1
boolean	Dependiente de la JVM			

*Con conocer el valor de estas dos columnas, es más que suficiente para deducir el resto.

Declaración de variables de referencia

Estas pueden ser declaradas como:

- Variables de clase (`static`)
- Variables de instancia (objeto)
- Parámetros
- Variables locales

Variables de instancia

Representan los atributos de una clase. Se definen dentro de la clase, pero fuera de los métodos, y solo se inicializan cuando el objeto es inicializado.
Características:

- Pueden utilizar cualquiera de los cuatro modificadores de acceso
- Pueden ser `final`, `transient`, `volatile`
- No pueden ser `abstract`, `sinchronized`, `strictfp`, `native`, `static`

Variables locales

Son variables declaradas dentro de un método. Una variable local puede definirse con el mismo nombre que una variable de instancia. A esto se lo conoce con el nombre de `shadowing`.
Dado un caso de estos (`shadowing`), si se quisiera referenciar la variable de instancia, se debe utilizar el operador `this`. Este almacena una referencia al objeto actual.
Características:

- No son inicializadas por defecto
- Pueden ser final
- No pueden ser `transient`, `volatile`, `abstract`, `sinchronized`, `strictfp`, `native`, `static` (porque se volverían variables de clase)

Declaración de arrays

Un array es un objeto que almacena multiples variables del mismo tipo, o subclases del mismo. Estos pueden contener tanto variables primitivas como objetos, pero el array, siempre será almacenado en el `heap`.
Sintaxis:
TipoVariable`[]` NombreVariable
De esta manera, al encontrarnos con una declaración como la anterior, es simple identificar el tipo de dato, en este caso, una referencia a un array de objetos de tipo TipoVariable.
También es posible declarar arrays multidimensionales, que no son otra cosa que arrays de arrays. Por cada dimensión, se especifica un par de `[]`.

Declaración de variables final

El declarar una variable con el modificador `final` hace imposible modificar su valor una vez que esta ha sido inicializada con un valor explícito.
En el caso de ser una variable primitiva, su valor no podrá modificarse nunca.

En el caso de ser una variable de referencia, no se podrá modificar la referencia al objeto que contiene, pero si su estado.

Variables transient

Solo es aplicable a variables de instancia.
Cuando una variable es marcada como **`transient`**, cuando el objeto es serializado, esta variable es excluida.

Variables volatile

Cuando una variable es marcada como **`volatile`**, le indica a la JVM que cada vez que un **`thread`** quiera acceder a su copia local de la variable, deberá verificarla con la copia maestra en memoria.

Variables y métodos estáticos

El modificador **`static`** es utilizado para crear variables y métodos que existirán independientemente de las instancias creadas. Los atributos y métodos **`static`** existen aun antes de la primer inicialización de un objeto de dicha clase, y solo habrá una copia de estos, por más que no se realice ninguna o **`n`** instancias.
Elementos que pueden contener este modificador:

- Métodos
- Variables de clase
- Una clase anidada dentro de otra clase (pero no dentro de un método).
- Bloques de inicialización

Declaración de enumeradores

Java permite restringir una variable para que solo tenga un conjunto determinado de elementos (valores predefinidos).
Declaración:

```
enum NombreEnumerador { VALOR1, VALOR2, … , VALORn }
```

Referencia:

```
NombreEnumerador.VALOR1
```

Los enumeradores pueden ser declarados como una clase aparte, o miembros de una clase, pero **nunca dentro de un método**.

Cada valor de un enumerador es una clase del tipo del NombreEnumerador. Cada enumerador conoce su índice, de manera que **la posición en que sea declarado el valor si importa.**

Declaración de contructores, métodos y variables

Debido a que los enum son en realidad un tipo especial de clase, estos pueden contener: constructores, métodos, variables de instancia, y cuerpo constante de clase específico (`constant` specific class body).
Debes recordar las siguientes características sobre enum:

- Nunca puedes invocar un constructor enum directamente. El contructor de enum es invocado automáticamente, con los argumentos que defines luego de las constantes.
- Puedes definir más de un argumento para el constructor, y puedes sobrecargar los constructores, así como cualquier clase.

El `constant` specific class body, es una declaración de lo que a simple vista parece una clase interna. Es utilizado cuando una constante requiere sobrescribir el comportamiento de algún método.

Lo que se viene

En la próxima entrega estaremos adentrándonos en el mundo de la programación Orientada a objetos.

Conoceremos como se compone una clase, y como es la sintaxis de esta. También veremos interfaces, conceptos como herencia, polimorfismo, y otros.

Aprenderemos sobre las relaciones que existen entre objetos como Is-A (es u) y As-A (tiene un).

Y veremos cómo extender una clase, o implementar una interface.

¿Tenés lo que se necesita para ser un SCJP?

Autor
Gustavo Alberola

Diseño
Leonardo Blanco