Certificación JAVA

# Capítulo 1: Bloques de construcción Java (Java Building Blocks)

## Estructuras de las clases Java

* Para utilizar la mayoría de clases se tienen que crear objetos.
* Un objeto es una instancia en tiempo de ejecución de una clase en memoria.
* Una clase contiene variables (fields) y métodos.
* Hay 3 tipos de comentarios, línea simple (//), multilinea (/\* \*/), Javadoc (/\*\* \*\*/).
* Las clases no necesariamente tienen que ser públicas, pero si se declara pública el archivo tiene que llamarse igual que el nombre de clase.
* Si hay dos clases en el mismo archivo máximo una de ellas puede ser pública.
* Al compilar un archivo .java lo que se obtiene es un bytecode (.class).
* El método main siempre debe ser estático sino se tendrá un error, así como si al correr una clase sin main también lanzara una excepción.
* Los argumentos del main puede definirse: String[] args, o String args[], o String… args.
* Los parámetros enviados al main, al ejecutar una aplicación se separan con espacios, si deseas usar espacios en el parámetro enviado debes rodearlo con comillas.
* Los argumentos de una línea de comandos siempre son Strings aunque pongas un número.
* Si utilizas un argumento en el método que no haz enviado (por ejemplo args[1]) se lanzará una excepción por el array.
* Para compilar se necesita JDK, para solo ejecutar se necesita solo el JRE.

## Declaración de Paquetes e Imports

* Los paquetes podrían hasta ser a.b.c, las reglas son las mismas que la de los nombres de variables.
* Al colocar el .\* se obtienen todas las clases del paquete (no las clases dentro de paquetes hijos ni variables ni métodos), hay algo llamado **static import** pero se verá en el capítulo 4.
* Existe redundancia de imports, por ejemplo java.lang se importa automáticamente en la clase, importar cualquier clase dentro de este paquete o sus subpaquetes sería redundante. Tambien sería redundante importar clases del mismo paquete.
* Aquí tenemos algunos ejemplos de imports que no funcionan:

**import** java.nio.file.\*; // BIEN

**import** java.nio.file.Files; // BIEN

**import** java.nio.file.Paths; // BIEN

**import** java.nio.\*; // MAL – un comodín(wildcard \*) solo obtiene clases,

//no "file.\*Files"

**import** java.nio.\*.\*; // MAL – solo puedes tener un comodín(wildcard \*)

//y debe de estar al final

**import** java.nio.files.Paths.\*; // MAL – No puedes importer métodos

//solo clases

* También se tienen casos de conflictos de nombres, no pueden importarse clases con el mismo nombre ya sea explícito o con el comodín.

**import** java.util.\*;

**import** java.sql.\*; // NO COMPILA

**import** java.util.Date; //SI COMPILA

**import** java.sql.\*;

**import** java.util.Date;

**import** java.sql.Date; //NO COMPILA

**public** **class** Conflicts {

Date date;

// some more code

}

* Si se necesitase utilizar ambas clases, se deja una en los imports y la otra se utiliza con el nombre completo (paquete.clase) o utilizar ambos con el nombre completo.
* Al ejecutar un programa por consola no se coloca la extensión (.class) (java paquete.ClassA).
* Si la clase a ejecutar utiliza clases guardadas en otros directorios se importa las rutas al classpath:

java -cp ".:/tmp/someOtherLocation:/tmp/myJar.jar" myPackage.MyClass

* El punto al inicio indica que quieres incluir el directorio actual donde estas realizando la ejecución en el classpath, lo demás indica que se deben buscar clases y paquetes en someOtherLocation y dentro de myJar.jar.
* También se puede utilizar el comodín (wildcard \*) para agregar todos los jars dentro de un directorio, pero no agregará los jars dentro de un subdirectorio del directorio indicado (no subpaquetes de directoryWitgJars).

java -cp "C:\temp\directoryWithJars\\*" myPackage.MyClass

* Cuando en el examen, en fragmentos de código no se incluye imports debe revisarse si se tiene un número de línea explícito y que no comience en 1, porque se comienza en 1 y no tiene los imports que debería tener no compilará el código.

## Creando Objetos

* Los constructores tienen las dos siguientes características básicas: mismo nombre de la clase y no tiene tipo de retorno.
* Todo lo que se encuentre dentro de llaves ( {} ) es llamado bloque de código (code block). Bloques de código fuera de métodos son llamados inicializadores de instancia (instance initializers) (Capítulo 5).
* Hasta el momento el orden de ejecución de campos (variables) e inicializadores de instancia es de acuerdo al orden en que se encuentran en el archivo. Luego de ello se ejecuta el constructor, pese a que el constructor pueda estar declarado antes que la declaración de variables o inicializadores de instancia.

## Distinguir entre Referencias de objetos y primitivos.

* Java tiene 8 tipos primitivos: boolean, byte, short, int, long, float, double, char.

|  |  |
| --- | --- |
| **TIPO** | **BITS** |
| boolean | True/false |
| Byte | 8-bit inegral value |
| short | 16-bit inegral value |
| int | 32-bit inegral value |
| long | 64-bit inegral value |
| float | 32-bit floating point value |
| double | 64-bit floating point value |
| Char | 16-bit Unicode valie |

* Float requiere un sufijo F seguido del número.
* Los tipos numéricos tienen la siguiente cantidad de números: byte (2^8), short (2^16), int (2^32), long (2^64), considerando los negativos y considerando al cero como positivo se tendría por ejemplo para byte el rango de -128 – 127.
* Otro punto que debe conocerse es que cuando un número se presenta como tal en el código es llamado un literal, de forma predeterminada Java asume que el literal es un int. Por ejemplo

**long** max = 3123456789; // DOES NOT COMPILE

**long** max = 3123456789L; // COMPILE

* Tambien pueden utilizarse otras bases númericas: octal con prefijo 0 (017), hexadecimal con prefijo 0x (0xFF), binario con prefijo 0b (0b10).
* La ultima cosa que se debe saber es que desde Java 7 se permitió agregar guiones bajos a los números para un mejor entendimiento.

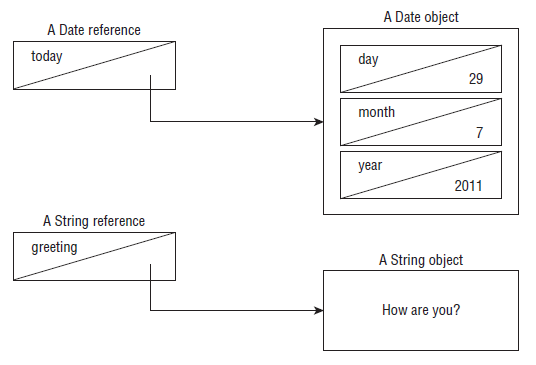
**double** notAtStart = \_1000.00; // DOES NOT COMPILE

**double** notAtEnd = 1000.00\_; // DOES NOT COMPILE

**double** notByDecimal = 1000\_.00; // DOES NOT COMPILE

**double** annoyingButLegal = 1\_00\_0.0\_0; // this one compiles

* A diferencia de los primitivos que guardan el valor, las variables de referencia solo apuntan al objeto guardado en memoria guardan la dirección de memoria donde el objeto está guardado (Java no permite conocer la dirección de memoria a diferencia de otros lenguajes como C por ejemplo).



* Existen algunas diferencias importantes entre primitivos y referencias:
  + A una variable de referencia se le puede asignar null pero a un primitivo no, en la compilación se mostraría con error.
  + Las variables de referencia pueden usarse para llamar métodos cuando no apuntan a null, los primitivos no tiene métodos a los que llamar.
  + Primitivos comienzan con minúscula, mientras que todas las clases comienzan con mayúscula, siendo una convención que debería seguirse.

## Declarando e Inicializando Variables