Clase Teórica 1 / 12

Índice de contenido

1.1 ¿ Que es JAVA ?....2
1.2 Historia de JAVA....2
1.3 Generalidades de JAVA....3
1.4 Maquina Virtual de JAVA....5
1.5 Kit de desarrollo y Entorno de ejecución (JDK, JRE)....6
1.6 Ediciones de JAVA....7
1.7 La Programación Orientada a Objetos como base de JAVA....7
1.8 Elementos del lenguaje JAVA....9
1.8.1 Tipos de datos soportados....9
1.8.2 Operadores aritméticos....10
1.8.3 Operadores relacionales y lógicos....10
1.8.4 Operadores de asignación....10
1.8.5 Precedencia de operadores....11
1.8.6 Palabras reservadas....11
1.8.7 Variables....11
1.8.8 Estructuras de control....13
1.8.9 Vectores....15
1.9 Estructura de un programa en JAVA....16
1.9.1 Estructura de una clase....17
1.9.2 Método main....18
1.9.3 Crear y manipular objetos de una clase....18
1.9.4 Sobrecarga de métodos....19
1.10 Convenciones de programación....20
Bibliografía....22
Licencia....22

www.profmatiasgarcia.com.ar

1.1 ¿ Que es JAVA ?

JAVA es un lenguaje de programación, basado en C++, multiplataforma dado que los programas compilados pueden correr bajo distintos sistemas operativos y distintas máquinas, sin la necesidad de recompilarse. El compilador de JAVA (javac) genera un archivo en código intermedio llamado bytecode que es ejecutado por la JAVA Virtual Machine (JVM) instalada en cada plataforma. La JVM contiene un compilador JIT (just in time) para traducir el archivo bytecode a un archivo ejecutable en la plataforma respectiva (similar al Framework de .Net de Microsoft, de aparición posterior, pero de uso únicamente en sistemas operativos de Microsoft). Debido a la característica de multiplataforma, los tipos de datos de JAVA tienen el mismo tamaño en todas las máquinas.

JAVA tiene muchos conceptos de sintaxis de C y C++, especialmente de C++, del que es un lenguaje derivado. Añade a C++ propiedades de gestión automática de memoria y soporte a nivel de lenguaje para aplicaciones multihilo. Por otra parte, JAVA, en principio a nivel medio, es más fácil de aprender y más fácil de utilizar que C++ ya que las características más complejas de C++ han sido eliminadas de JAVA: herencia múltiple, punteros y sentencia goto entre otras.

En realidad JAVA no solo es un lenguaje de programación. JAVA es un lenguaje, una plataforma de desarrollo, un entorno de ejecución y un conjunto de librerías para desarrollo de programas sofisticados. Las librerías para desarrollo se denominan JAVA Application Programming Interface (JAVA API).

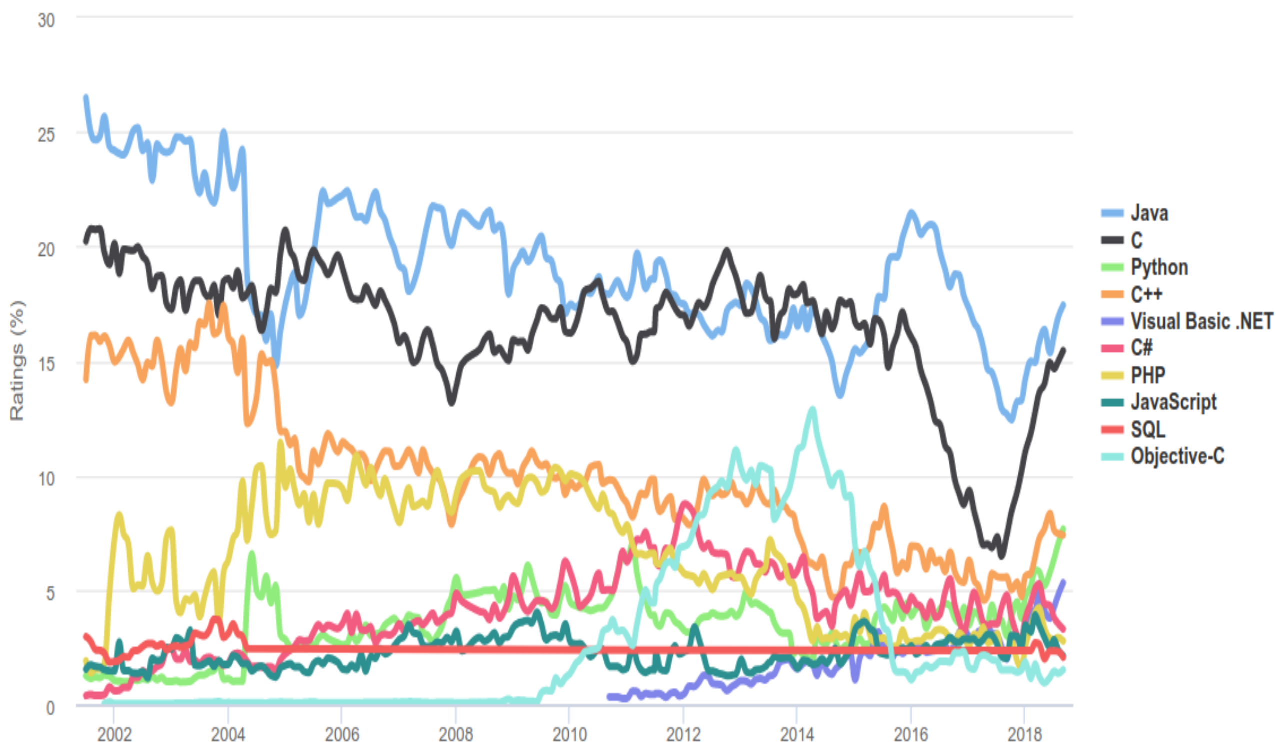
1.2 Historia de JAVA

JAVA fue creado por Sun Microsystems, anunciado formalmente en mayo de 1995. Originalmente la empresa patrocinó un proyecto interno de investigación denominado Green en 1991, por el cual trataron de diseñar un nuevo lenguaje de programación destinado a electrodomésticos, que desembocó en el desarrollo de un lenguaje basado en C++ creado por James Gosling, al cual llamó Oak debido a un roble que tenía a la vista desde su ventana en las oficinas de Sun. Posteriormente se descubrió que ya existía un lenguaje de programación con ese nombre por lo cual se lo cambió por JAVA (que es una variedad de café, que la gente de Sun se le ocurrió usar como nombre del nuevo lenguaje, mientras tomaban un café en una cafetería local).

La popularidad del servicio Web se intensificó en 1993; en ese entonces Sun vio el potencial de usar JAVA para agregar contenido dinámico, como interactividad y animaciones, a las páginas Web. JAVA generó la atención de la comunidad de negocios debido al fenomenal interés en la World Wide Web. En la actualidad, JAVA se utiliza para desarrollar aplicaciones empresariales a gran escala, para mejorar la funcionalidad de los servidores Web (las computadoras que proporcionan el contenido que vemos en nuestros exploradores Web), para proporcionar aplicaciones para los dispositivos de uso doméstico (como teléfonos celulares, teléfonos inteligentes, receptores de televisión por Internet y mucho más) y para muchos otros propósitos.

En 2009, Oracle adquirió Sun Microsystems. En la conferencia JAVAOne 2010, Oracle anunció que el 97% de todas las computadoras de escritorio, tres mil millones de dispositivos portátiles y 80 millones de dispositivos de televisión ejecutan JAVA. En la actualidad hay cerca de 12 millones de desarrolladores de JAVA, en comparación con los 4.5 millones en 2005. Ahora JAVA es el lenguaje de desarrollo de software más utilizado en todo el mundo.

www.profmatiasgarcia.com.ar



1.3 Generalidades de JAVA

En una de las primeras publicaciones que hacen referencia a JAVA (Zukowski, 1997), Sun lo describe como un sencillo, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutral, portable, de alto rendimiento, multihilo, y dinámico lenguaje.

- **Sencillo:** JAVA es un lenguaje simple. JAVA es un poco más fácil que C++, que se consideraba como el lenguaje de programación más popular hasta la implementación de JAVA. JAVA ha puesto simplicidad en la programación en comparación con C++, incorporando características medulares de C++ y eliminando algunas de éstas que hacen de C++ un lenguaje difícil y complicado. Otra facilidad que proporciona JAVA es la elegancia de su sintaxis que hacen más fácil la lectura (comprensión) y escritura de programas.
- **Orientado a objetos:** la programación en JAVA se centra en la creación, manipulación y construcción de objetos. JAVA solo trabaja con objetos con la única excepción de unos pocos tipos de datos primitivos. Lo hace a través de la definición de clases que consiste en una estructura que permite encapsular datos y los métodos que manejan esos datos. En conjunto, los datos y métodos describen el estado y el comportamiento de un objeto que es instancia de una clase.
- **Distribuido:** La computación distribuida implica que varias computadoras trabajan juntas en la red. JAVA proporciona una librería de clases para aplicaciones en red, que permiten abrir sockets e intercomunicarse con servidores y clientes remotos.
- **Interpretado y compilado:** El compilador de JAVA genera bytecode para la Máquina Virtual de

JAVA (JVM), en lugar de compilar al código nativo de la máquina donde se ejecutará. El bytecode es independiente de la plataforma y se puede hacer uso de él en cualquier máquina (no necesariamente una computadora) que tenga un intérprete de JAVA. El archivo con el código en JAVA se compila traduciéndolo a código intermedio bytecode que luego será interpretado por el JIT de la JVM que carga y traduce de a partes pero de manera optimizada para no traducir lo mismo varias veces.

- **Arquitectura Neutral y Portable:** Debido a que el lenguaje JAVA se compila en un formato de arquitectura propia o mejor dicho neutra llamada bytecode, un programa en JAVA puede ejecutarse en cualquier sistema, siempre y cuando éste tenga una implementación de la JVM y esta es la característica tal vez más notable que tenga JAVA.
  Se pueden, por ejemplo, ejecutar applets desde cualquier navegador de cualquier sistema operativo que cuente con una JVM y más allá todavía, hacer sistemas autónomos que se ejecuten directamente sobre el sistema operativo. En las aplicaciones que se desarrollan hoy en día, muy probablemente se necesite tener la misma versión ejecutándose en un ambiente de trabajo con UNIX, Linux, Windows o Mac. Y más aún, con las diferentes versiones de procesadores y dispositivos (celulares, celulares inteligentes, consolas de video juegos, entre muchos otros) soportados por estos sistemas, las posibilidades se pueden volver interminables y la dificultad para mantener una versión de la aplicación para cada uno de ellos crece de igual manera, interminable.
- **Multiplataforma:** La misma definición de bytecode hace que JAVA sea multiplataforma y no tenga la necesidad de complicados temas al portar la aplicación entre dispositivos como tipos de datos y su longitud, características y capacidades aritméticas; caso contrario, por ejemplo, del lenguaje C donde un tipo int puede ser de 16, 32 ó 64 bits dependiendo de la plataforma de compilación y ejecución. Los programadores necesitan hacer un solo esfuerzo por terminar sus asignaciones, esto se puede lograr apegándose al eslogan que hizo tan famoso Sun: “Write Once, Run Anywhere” (Escribe una vez, ejecuta donde sea).
- **Multihilo:** JAVA es un lenguaje multihilo ya que soporta la ejecución de múltiples tareas al mismo tiempo y cada uno de esos hilos puede soportar la ejecución de una tarea específica diferente. Soporta sincronización de múltiples hilos de ejecución. Puede por ej. permitir que un hilo se encargue de la comunicación, otro de la interacción con el usuario, otro de una animación en pantalla y otro realice cálculos.
- **Dinámico:** En cualquier instante de la ejecución JAVA puede ampliar sus capacidades mediante el enlace de clases que estén ubicadas en la máquina residente, en servidores remotos o cualquier sitio de la red (intranet/Internet), esto lo permite el sistema de ejecución en tiempo real de JAVA que es dinámico. Caso contrario de lenguajes como C++ que hacen esto al momento de la compilación, después ya no se puede. Se puede extender libremente métodos y atributos de una clase sin afectar la ejecución corriente, las capacidades descritas se encuentran dentro del API Reflections.
- **Robusto:** JAVA fue diseñado para ser un lenguaje de programación que genere aplicaciones robustas. JAVA no elimina la necesidad del aseguramiento de calidad en el software; de hecho es muy posible y probable tener errores al programar en JAVA. No elimina tampoco la mayoría de los errores que se comenten al utilizar cualquier lenguaje de programación.
  Sin embargo al ser fuertemente tipado se asegura que cada variable o método que se utilice

www.profmatiasgarcia.com.ar

corresponda en realidad a lo que el programador quiso utilizar y no que se escapen errores en conversión de tipos de dato, por ejemplo. JAVA requiere declaración explicita de métodos, cosa que no se permite en otros lenguajes, como en C. JAVA ha sido pensado en la fiabilidad, eliminando la mayoría (o todas) las posibles partes de otros lenguajes de programación propensas a errores, por ejemplo elimina los punteros y soporta el manejo de excepciones en tiempo de ejecución para proporcionar robustez a la programación. JAVA utiliza recolección de basura en tiempo de ejecución en vez de liberación explícita de memoria. En lenguajes como C++ es necesario borrar o liberar memoria una vez que el programa ha terminado.

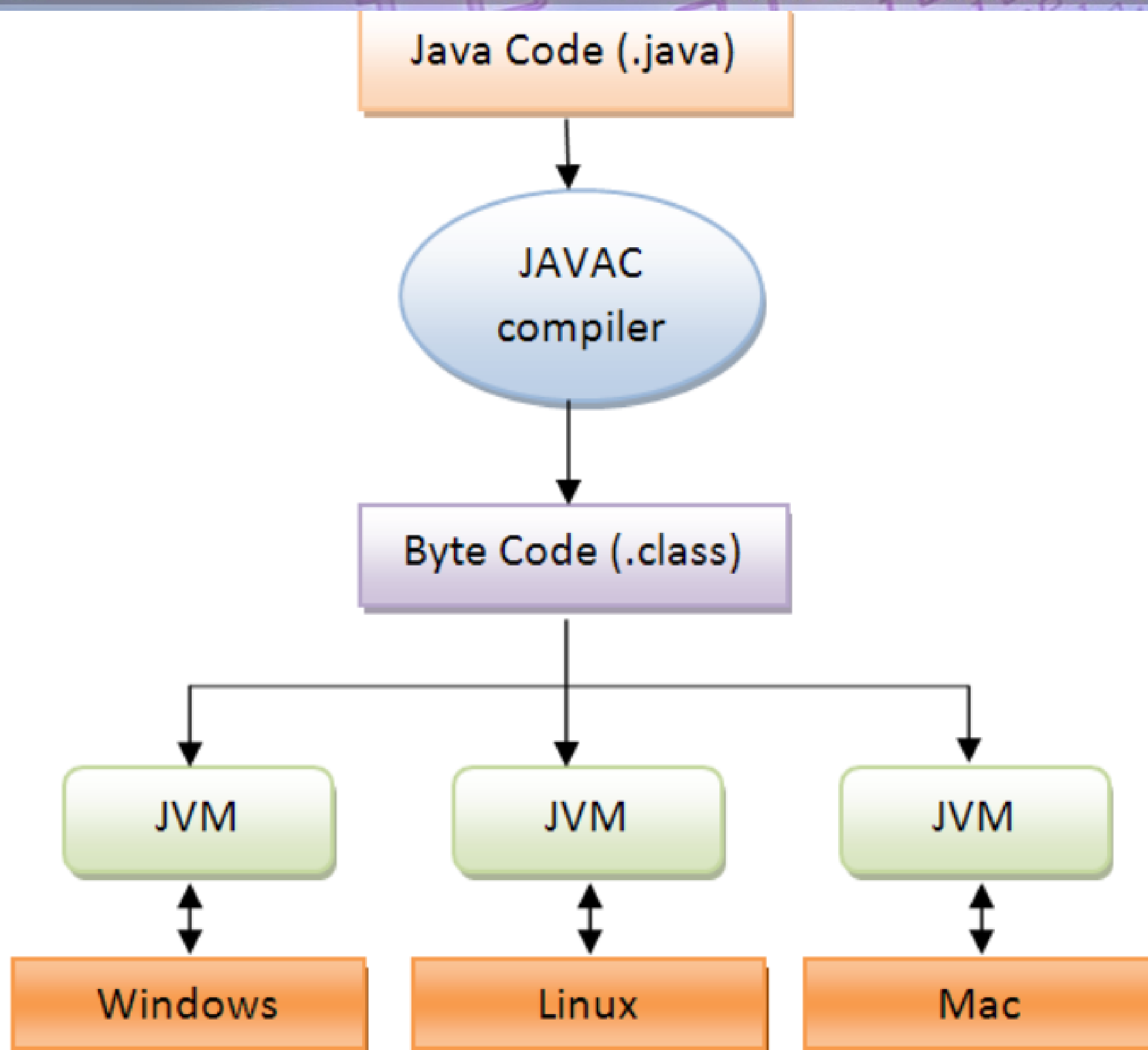
- **Seguro:** Uno de los aspectos más sobresalientes de JAVA es el de ser un lenguaje seguro, lo que es especialmente interesante debido a la naturaleza de la distribución de aplicaciones en JAVA. Sin un aseguramiento el usuario no estará tranquilo de bajar y ejecutar código en su computadora desde Internet o cualquier otra fuente. JAVA fue diseñado con la seguridad como punto principal y tiene disponibles muchas capas que gestionan la seguridad de la aplicación que se escribe, bloqueando al programador si intenta distribuir código malicioso escrito en lenguaje JAVA. Los programadores de JAVA no tienen permitido cierto tipo de acciones que se consideran altamente vulnerables o que el típico caso de ataque contra un usuario común, por ejemplo el acceso a memoria, desbordamiento de arreglos entre muchos otros.
  Otra capa de seguridad dentro de JAVA es el modelo sandbox que hace una ejecución controlada de cualquier bytecode que llega a la JVM, por ejemplo si se lograr evadir la regla del código no malicioso al momento de compilar, este modelo en su ejecución controlada evitaría que las repercusiones lleguen al mundo real.

1.4 Maquina Virtual de JAVA

Una Máquina Virtual de JAVA (JVM – JAVA Virtual Machine), es un software de proceso nativo, es decir, está desarrollada para una plataforma específica que es capaz de interpretar y ejecutar un programa o aplicación escrito en un código binario (el ya mencionado bytecode) que se genera a partir del compilador de JAVA.

La JVM es una máquina de computación de tipo abstracto. Emulando a una máquina de componentes reales, tiene un conjunto de instrucciones y utiliza diversas áreas de memoria asignada para ella. Las versiones actuales de la JVM están disponibles para muchas (casi todas) las plataformas de computación más populares. Debe entenderse sin embargo que, la JVM no asume una tecnología específica o particular, esto es lo que la vuelve multiplataforma. Además no es de por sí interpretada, y que sólo pueda funcionar de esta manera, de hecho es capaz de generar código nativo de la plataforma específica e incluso aceptar como parámetro de entrada, en lugar del bytecode, código objeto compilado, por ejemplo funciones nativas de C. De igual manera puede implementarse en sistemas embebidos o directamente en el procesador.

www.profmatiasgarcia.com.ar



1.5 Kit de desarrollo y Entorno de ejecución (JDK, JRE)

El Kit de desarrollo conocido como JDK (JAVA Development Kit) provee de un compilador, un mecanismo para comprimir un proyecto en un solo archivo de tipo JAR (que es compatible con ZIP) y un entorno de ejecución para nuestros binarios.

Cuando nuestro proyecto terminado se prepara para distribuir, no es necesario tener el compilador y la mayoría de las herramientas que se proveen en el JDK, entonces podemos prescindir de dicho JDK y utilizar el entorno de ejecución que es más pequeño en cuestiones sólo de espacio en disco. Este JRE (JAVA Runtime Environment) también puede redistribuirse sin problemas de licencias.

Ambos incluyen la JVM que permite la ejecución de los programas.

El entorno JDK no es el más adecuado para el desarrollo de aplicaciones JAVA, debido a funcionar única y exclusivamente mediante comandos de consola. Hoy en día la programación se suele ayudar de entornos visuales IDEs (Integrated Development Environment), como Eclipse y Netbeans, que facilitan enormemente la tarea.

1.6 Ediciones de JAVA

Existen varias Ediciones de JAVA, cada una de ellas diseñada para cierto ambiente en particular. Estas ediciones son:

JAVA Standard Edition es la edición que se emplea en computadoras personales (desktops y laptops). Se le conoce también como JAVA Desktop (escritorio) y es la versión que tienes que instalar para poder programar en JAVA en tu computadora, aunque tus programas estén destinados para alguna de las otras ediciones. JAVA Platform, Standard Edition o JAVA SE (conocido anteriormente hasta la versión 5.0 como Plataforma JAVA 2, Standard Edition o J2SE), es una colección de APIs del lenguaje de programación JAVA útiles para muchos programas de la Plataforma JAVA. La Plataforma JAVA 2, Enterprise Edition incluye todas las clases en el JAVA SE, además de algunas de las cuales son útiles para programas que se ejecutan en servidores sobre workstations.

JAVA Micro Edition es la edición que se emplea en dispositivos móviles, tales como los teléfonos celulares y smartphones. Es una versión recortada del JAVA SE con ciertas extensiones enfocadas a las necesidades particulares de esos tipos de dispositivos. La plataforma JAVA Micro Edition, o JAVA ME (anteriormente J2ME), es una colección de APIs en JAVA orientadas a productos de consumo como PDAs, teléfonos móviles o electrodomésticos. JAVA ME se ha convertido en una buena opción para crear juegos en teléfonos móviles debido a que se puede emular en un PC durante la fase de desarrollo y luego cargarlos fácilmente al teléfono. Al utilizar tecnologías JAVA el desarrollo de aplicaciones o videojuegos con estas APIs resulta bastante económico de portar a otros dispositivos.

JAVA Enterprise Edition es la edición que se emplea para hacer aplicaciones. Incluye a toda la Standard Edition y muchas, muchas más extensiones. J2EE es un grupo de especificaciones diseñadas por Sun que permiten la creación de aplicaciones empresariales, esto sería: acceso a base de datos (JDBC), utilización de directorios distribuidos (JNDI), acceso a métodos remotos (RMI/CORBA), funciones de correo electrónico (JAVAMail), aplicaciones Web (JSP y Servlets)...etc. Aquí es importante notar que J2EE es solo una especificación, esto permite que diversos productos sean diseñados alrededor de estas especificaciones algunos son Tomcat y Weblogic.

JAVA Card es la versión de JAVA enfocada a aplicaciones que se ejecutan en tarjetas de crédito con chip. Es una versión muy recortada de JAVA. Una JAVA Card es una tarjeta capaz de ejecutar mini-aplicaciones JAVA. En este tipo de tarjetas el sistema operativo es una pequeña máquina virtual JAVA (JVM) y en ellas se pueden cargar dinámicamente aplicaciones desarrolladas específicamente para este entorno.

1.7 La Programación Orientada a Objetos como base de JAVA

La programación orientada a objetos (POO) es la base de JAVA y constituye una forma de organización del conocimiento en la que las entidades centrales son los objetos. En un objeto se unen una serie de datos con una relación lógica entre ellos, a los que se denomina atributos o variables de instancia, con las rutinas o funciones necesarias para manipularlos, a las que se denomina métodos. Los objetos se comunican unos con otros mediante interfaces bien definidas a través del uso de mensajes; en PO0 los mensajes están asociados con métodos, de forma que cuando un objeto recibe un mensaje, ejecuta el método asociado.

Cuando se escribe un programa utilizando programación orientada a objetos, no se definen verdaderos objetos, sino clases; una clase es como una plantilla para construir varios objetos con características similares. Los objetos se crean cuando se define una variable de su clase lo que se denomina instanciar una clase. En las clases pueden existir unos métodos especiales denominados *constructores* que se llaman siempre que se crea un objeto de esa clase y cuya misión es iniciar el objeto. Los *destructores* son otros métodos especiales que pueden existir en las clases y cuya misión es realizar cualquier tarea final que corresponda realizar en el momento de destruir el objeto. Las propiedades fundamentales de la POO son:

- El **encapsulamiento**, que consiste en la combinación de los datos y las operaciones que se pueden ejecutar sobre esos datos en un objeto, impidiendo usos indebidos al forzar que el acceso a los datos se efectúe siempre a través de los métodos del objeto. En JAVA, la base del encapsulamiento es la clase, donde se define la estructura y el comportamiento que serán compartidos por el grupo de objetos pertenecientes a la misma.
- La **herencia** es la capacidad para crear nuevas clases (descendientes) que se construyen sobre otras existentes, permitiendo que éstas les transmitan sus propiedades. En programación orientada a objetos, la reutilización de código se efectúa creando una subclase que constituye una restricción o extensión de la clase base, de la cual hereda sus propiedades.
- El **polimorfismo** consigue que un mismo mensaje pueda actuar sobre diferentes tipos de objetos y comportarse de modo distinto. El polimorfismo adquiere su máxima expresión en la derivación o extensión de clases; es decir, cuando se obtienen nuevas clases a partir de una ya existente mediante la propiedad de derivación de clases o herencia.

HERENCIA

ENCAPSULAMIENTO

POLIMORFISMO

ABSTRACCION

www.profmatiasgarcia.com.ar

1.8 Elementos del lenguaje JAVA

Como cualquier lenguaje de alto nivel, JAVA cuenta con tipos de datos primitivos, palabras reservadas, operadores, estructuras de control, etc. Ahora veremos la sintaxis que corresponde a cada uno de estos elementos en JAVA.

1.8.1 Tipos de datos soportados

Tipo de dato	Tamaño en bits	Rango de valores	Descripción
Números enteros			
`byte`	8 bits complemento a 2	-128 a 127	Entero de un byte
`short`	16 bits complemento a 2	-32.768 a 32.767	Entero corto
`int`	32 bits complemento a 2	-2.147.483.648 a 2.147.483.647	Entero
`long`	64 bits complemento a 2	- 9.223.372.036.854.775.808L a 9.223.372.036.854.775.807L	Entero largo
Números reales			
`float`	32 bits IEEE 754	+/- 3.4E+38F (7 cifras decimales equivalentes)	Coma flotante con precisión simple
`double`	64 bits IEEE 754	+/- 1.8E+308 (15 cifras decimales equivalentes)	Coma flotante con precisión doble
Otros tipos			
`boolean`	8 bits	True y False	Un valor booleano
`char`	16 bits Carácter	Conjunto de caracteres Unicode ISO	Un solo carácter
`String`	Variable	Conjunto de caracteres	Cadena de caracteres

Muchas veces es necesario realizar conversiones de tipos cuando se evalúa una expresión aritmética. JAVA permite “castear” una expresión o variable de un tipo de dato a otro indicando entre paréntesis el nuevo tipo de dato.

```
double centigrados = ((fahrenheit - 32.0) * 5.0) / 9.0;
int temperatura = (int) centigrados;
```

Los Strings son una secuencia de caracteres. En JAVA, las cadenas de caracteres no son datos primitivos en realidad, son objetos y la plataforma proporciona la clase **`String`** para crear y manipular dichas cadenas.

```
String nombre = "Matias";
```

Esta es la forma de declaración básica, aunque la clase **`String`** tiene mas de diez constructores diferentes para generar objetos.

Al ser un objeto cuenta con varios métodos para poder manipularlo, como por ejemplo el método `length()` que indica el tamaño total de caracteres en el string. El método `concat()` para concatenar cadenas de caracteres. O `toUpperCase()` que convierte todo el string a mayúsculas.

`System.out.println(nombre.toUpperCase());` Imprime MATIAS

1.8.2 Operadores aritméticos

Operador	Significado
++	Incremento
--	Decremento
+	Unario + (positivo)
-	Unario - (negativo)
*	Multiplicación
/	División
%	Resto div. entera (módulo)
+	Suma
-	Resta

1.8.3 Operadores relacionales y lógicos

Operador	Significado
==	Igual (comparación igualdad)
!=	Distinto a
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
!	Negado (Not)
&&	Conjunción lógica (And)
||	Disyunción lógica (Or)

1.8.4 Operadores de asignación

Operador	Uso	Equivalente
=	a= 5	a= 5
+=	a+= 4	a = a + 4
-=	a-= 3	a= a - 3
*=	a*= 2	a = a * 2
/=	a/=3	a= a / 3
%=	a%= 4	a = a % 4
++ , --	a++ o ++a, a-- o --a	a = a + 1, a = a - 1

www.profmatiasgarcia.com.ar

1.8.5 Precedencia de operadores

Operadores	Representación
Operadores posfijos	`[ ], ( ), a++, a--`
Operadores pre y unarios	`++a, --a, +a, -a, ~, !`
Creación o conversión	`new (tipo) a`
Multiplicación	`*, /, %`
Suma	`+, -`
Comparación	`==, !=, <, >, < =, >=`
Operadores lógicos	`&&, ||`
Condicional	`? :     ej= (a > b) ? a : b;`
Asignación	`=, +=, -=, *=, /=, &=`

1.8.6 Palabras reservadas

abstract	do	implements	protected	throw
boolean	double	import	public	throws
break	else	instanceof	rest	transient
byte	extends	int	return	true
case	false	interface	short	try
catch	final	long	static	void
char	finally	native	strictftp	volatile
class	float	new	super	while
const*	for	null	switch	
continue	goto*	package	synchronized	
default	if	private	this	

1.8.7 Variables

Una variable se define especificando el tipo y el nombre de dicha variable, por ejemplo: `int numero`;. Estas variables pueden ser tanto de tipos primitivos como referencias a objetos de alguna clase perteneciente al API de JAVA o generada por el usuario. Si no se especifica un valor en su declaración, las variable primitivas se inicializan a cero (salvo **`boolean`** y **`char`**, que se inicializan a false y '\0').

Análogamente las variables de tipo referencia son inicializadas por defecto a un valor especial: **`null`**.

Para declarar una constante simbólica se debe anteponer la palabra reservada **`final`**, por ejemplo: `final double PI = 3.1416;` Si la constante es miembro de la clase (atributo), convendrá agregar el modificador **`static`** para que solo exista esa constante para todos los objetos que se creen de esa clase.

```
final static int MAX = 600;
```

A diferencia de C/C++, los tipos de datos en JAVA están perfectamente definidos en todas y cada una de las posibles plataformas. Por ejemplo, un int ocupa siempre la misma memoria y tiene el mismo

www.profmatiasgarcia.com.ar

rango de valores, en cualquier tipo de ordenador.

Se utiliza la palabra `void` para indicar la ausencia de un tipo de dato para una variable determinada.

Es importante distinguir entre la referencia a un objeto y el objeto mismo. Una referencia es una variable que indica dónde está guardado un objeto en la memoria del ordenador (a diferencia de C/C++, JAVA no permite acceder al valor de la dirección, pues en este lenguaje se han eliminado los punteros). Al declarar una referencia todavía no se encuentra "apuntando" a ningún objeto en particular (salvo que se cree explícitamente un nuevo objeto en la declaración), y por eso se le asigna el valor null. Si se desea que esta referencia apunte a un nuevo objeto es necesario crear el objeto utilizando el operador `new`. Este operador reserva en la memoria del ordenador espacio para ese objeto (atributos y métodos). También es posible igualar la referencia declarada a otra referencia a un objeto existente previamente.

Un tipo particular de referencias son los arrays o vectores, sean éstos de variables primitivas (por ejemplo, un vector de enteros) o de objetos. En la declaración de una referencia de tipo array hay que incluir los corchetes [].

Se entiende por visibilidad, ámbito o scope de una variable, la parte de la aplicación donde dicha variable es accesible y por lo tanto puede ser utilizada en una expresión. En JAVA todas las variables deben estar incluidas en una clase. En general las variables declaradas dentro de unas llaves {}, es decir dentro de un bloque, son visibles y existen dentro de estas llaves. Por ejemplo las variables declaradas al principio de una función existen mientras se ejecute la función; las variables declaradas dentro de un bloque if no serán válidas al finalizar las sentencias correspondientes a dicho if y las variables miembro de una clase (es decir declaradas entre las llaves {} de la clase pero fuera de cualquier método) son válidas mientras existe el objeto de la clase.

Las variables miembro de una clase (atributos) declaradas como `public` son accesibles a través de una referencia a un objeto de dicha clase utilizando el operador punto (.). Las variables miembro declaradas como `private` no son accesibles directamente desde otras clases. Las funciones miembro de una clase tienen acceso directo a todas las variables miembro de la clase sin necesidad de anteponer el nombre de un objeto de la clase. Sin embargo las funciones miembro de una clase B derivada de otra A, tienen acceso a todas las variables miembro de A declaradas como `public` o `protected`, pero no a las declaradas como `private`. Una clase derivada sólo puede acceder directamente a las variables y funciones miembro de su clase base declaradas como `public` o `protected`. Otra característica del lenguaje es que es posible declarar una variable dentro de un bloque con el mismo nombre que una variable miembro, pero no con el nombre de otra variable local que ya existiera. La variable declarada dentro del bloque oculta a la variable miembro en ese bloque. Para acceder a la variable miembro oculta será preciso utilizar el operador `this`, en la forma `this.varname`.

Uno de los aspectos más importantes en la programación orientada a objetos (OOP) es la forma en la cual son creados y eliminados los objetos. En JAVA la forma de crear nuevos objetos es utilizando el operador `new`. Cuando se utiliza el operador `new`, la variable de tipo referencia guarda la posición de memoria donde está almacenado este nuevo objeto. Para cada objeto se lleva cuenta de por cuántas variables de tipo referencia es apuntado. La eliminación de los objetos la realiza el programa denominado **garbage collector**, quien automáticamente libera o borra la memoria ocupada por un objeto cuando no existe ninguna referencia apuntando a ese objeto. Lo anterior significa que aunque una variable de tipo referencia deje de existir, el objeto al cual apunta no es eliminado si hay otras referencias apuntando a ese mismo objeto.

www.profmatiasgarcia.com.ar

En JAVA los nombres de las variables y los métodos siguen ciertas reglas, por ejemplo las variables se acostumbra a nombrarlas con todas sus letras en minúsculas, los métodos con la primer palabra que la identifica en minúsculas y las palabras siguientes con la primer letra en mayúsculas y los nombres de las Clases de objetos con la primer letra de cada palabra que la define en mayúsculas.

1.8.8 Estructuras de control

Condición

Permiten ejecutar una de entre varias acciones en función del valor de una expresión lógica o relacional. Se tratan de estructuras muy importantes ya que son las encargadas de controlar el flujo de ejecución de un programa.

```
if (<condición>)
    <instrucción>;
```

```
if (<condición>){
    <varias instrucciones> }
```

```
if (<condición>)
    <instrucción>;
else
    <instrucción>;
```

```
if (<condición>){
    <varias instrucciones> }
else {
    <varias instrucciones> }
```

```
if (<condición>)
    <instrucción>;
else {
    <varias instrucciones> }
```

```
if (<condición>){
    <varias instrucciones> }
else
    <instrucción>;
```

```
if (<condición>) {
    <varias instrucciones> }
    else if (<condición2>) {
          <varias instrucciones> }
    else if (<condición3>) {
          <varias instrucciones> }
    else {
          <varias instrucciones> }
```

```
(<cond>) ? <inst_true> : <inst_false>
```

www.profmatiasgarcia.com.ar

CONDICIÓN MÚLTIPLE

```
switch (<expresión>) {
    case valor1:
          <instruccion>;
          break;
    case valor2:
          <instruccion1>;
          <instruccion2>;
          break;
    case valor3:
          <instruccion>;
          break;
    ...
    [default: <instruccion>;]
 }
```

El valor de la expresión y de las constantes tiene que ser de tipo **char**, **byte**, **short** o **int**. La sentencia **break** es optativa. A partir de JAVA SE 7, es posible utilizar objetos String en la expresión de control de una instrucción **switch**, y en las etiquetas **case**.

Al evaluar la expresión del **switch**, el intérprete busca una constante con el mismo valor. Si la encuentra, ejecuta las sentencias asociadas a esta constante hasta que tropiece con un **break**. La sentencia **break** finaliza la ejecución de esta estructura. Si no encuentra ninguna constante que coincida con la expresión, busca la línea **default**. Si existe, ejecuta las sentencias que le siguen. La sentencia **default** es opcional.

www.profmatiasgarcia.com.ar

REPETITIVAS

Se utilizan para realizar un proceso repetidas veces. El código incluido entre las llaves {} (opcionales si el proceso repetitivo consta de una sola línea), se ejecutará mientras se cumpla unas determinadas condiciones.

```
while (<condición>)
    <instrucción>;


while (<condición>){
    <varias instrucciones> }


do {
    <varias instrucciones>
} while(<condición>);


for(<inic> ; <condición> ; <actualiz>)
    <instrucción>;


for(<inic> ; <condición> ; <actualiz>){
    <varias instrucciones> }
```

1.8.9 VECTORES

Para manejar colecciones de objetos del mismo tipo estructurados en una sola variable se utilizan los vectores, también llamados arrays y matrices. En JAVA, los vectores son en realidad objetos y por lo tanto se puede llamar a sus métodos.

Existen dos formas equivalentes de declarar vectores en JAVA:

1) `tipo nombreDelVector [ ] ;`

2) `tipo [ ] nombreDelVector;`

Los vectores, al igual que las demás variables pueden ser inicializados en el momento de su declaración. En este caso, no es necesario especificar el número de elementos máximo reservado. Se reserva el espacio justo para almacenar los elementos añadidos en la declaración. Por ejemplo:

```
String diasLaborables[]={"Lunes","Martes","Miércoles","Jueves","Viernes"};
```

Una simple declaración de un vector no reserva espacio en memoria, a excepción del caso anterior, en el que sus elementos obtienen la memoria necesaria para ser almacenados. Para reservar la memoria hay que llamar explícitamente a un constructor `new` de la siguiente forma `new tipo [numElementos];` Por ejemplo:

```
int matriz[][];
matriz = new int[4][7]; //tendrá 4 filas y 7 columnas

double vector[] = new double[5];
```

Para hacer referencia a los elementos particulares del vector, se utiliza el identificador del vector junto con el índice del elemento entre corchetes. El índice del primer elemento es el cero (0) y el del último, el número de elementos menos uno. `vector[2] = 4.5;`

El intento de acceder a un elemento fuera del rango de la matriz, a diferencia de lo que ocurre en C, provoca una excepción (error) que, de no ser manejado por el programa, será el compilador quien aborte la operación.

La instrucción `for` mejorada itera a través de los elementos de un arreglo sin utilizar un contador, con lo cual evita la posibilidad de “salirse” del arreglo. La sintaxis de una instrucción for mejorada es:

```
for (tipo parámetro : nombreVector){
        instrucciones
}
```

en donde parámetro tiene un tipo y un identificador (por ejemplo, `int numero`), y nombreVector es el arreglo a través del cual se iterará. El tipo del parámetro debe coincidir con el de los elementos en el arreglo.

La instrucción `for` mejorada itera a través de valores sucesivos en el arreglo, uno por uno. El encabezado del `for` mejorado se puede leer como “para cada iteración, asignar el siguiente elemento del arreglo a la variable parámetro, después ejecutar la siguiente instrucción”.

La instrucción `for` mejorada se puede utilizar en lugar de la instrucción `for` controlada por contador, cuando el código que itera a través de un arreglo no requiere acceso al contador que indica el subíndice del elemento actual del arreglo.

www.profmatiasgarcia.com.ar

Para pasar un argumento tipo vector a un método, se especifica el nombre del vector sin corchetes. Por ejemplo, si el vector temperaturasPorHora se declara como

```
double [] temperaturasPorHora = new double[24];
```

entonces la llamada al método

```
modificarVector(temperaturasPorHora);
```

pasa la referencia del vector temperaturasPorHora al método `modificarVector`. Todo objeto vector "conoce" su propia longitud (a través de su campo `length`). Por ende, cuando pasamos a un método la referencia a un objeto vector, no necesitamos pasar la longitud del mismo como un argumento adicional.

Para que un método reciba una referencia a un vector a través de una llamada a un método, la lista de parámetros del método debe especificar un parámetro tipo vector. Por ejemplo, el encabezado para el método modificarVector podría escribirse así:

```
void modificarVector (double [] b){
```

indica que modificarVector recibe la referencia de un vector **`double`** en el parámetro b. La llamada a este método pasa la referencia al vector temperaturaPorHoras, de manera que cuando el método llamado utiliza la variable b tipo vector, hace referencia al mismo objeto vector como temperaturasPorHora en el método que hizo la llamada.

1.9 Estructura de un programa en JAVA

Ya que JAVA es un lenguaje orientado a objetos, un programa en JAVA se compone solamente de objetos. Recuerda que un objeto es la instanciación de una clase, y que una clase equivale a la generalización de un tipo especifico de objetos. La clase define los atributos del objeto así como los métodos para manipularlos. Muchas de las clases que utilizaras pertenecen a la biblioteca de JAVA, ya están escritas y compiladas, pero otras tendrás que escribirlas dependiendo del problema que se deba resolver.

Partiendo desde la concepción más simple de un programa en JAVA, este se puede escribir a partir de una sola clase y conforme la complejidad de la aplicación vaya en aumento, muy probablemente (de hecho lo es) el número de clases va a ir en aumento, se podría entender como una correlación directa. Por definición cada clase necesita un punto de entrada por donde comenzará la ejecución del programa, y la clase escrita debe contenerlo como el programa, rutina o método principal: **`main()`** y dentro de este método contiene las llamadas a todas las subrutinas, métodos o demás partes de la clase o incluso de la aplicación en general.

Una estructura simple pero funcional de un programa en JAVA sería:

```
public class inicio{
    public static void main(String[] args){
            Primer_sentencia;
            Sentencia_siguiente;
            Otra_sentencia;
            // ...
            Sentencia_final;
    } //cierre del método main
```

www.profmatiasgarcia.com.ar

```
} // cierre de la clase
```

1.9.1 Estructura de una clase

Apegándose estrictamente a los lineamientos que genera la documentación de JAVA, una clase deberá contener las siguientes partes:

- **Package**: Es el apartado que contendrá la definición de directorios o jerarquía de acceso a donde estará alojada la clase. Un paquete es un conjunto de clases, lógicamente relacionadas, agrupadas bajo un nombre. Ayudan a organizar las clases en grupos para facilitar el acceso cuando las necesitemos.
  Para referirnos a una clase de un paquete, tenemos que hacerlo utilizando su nombre completo, excepto cuando el paquete haya sido importado implícita o explícitamente. Ej `java.lang.System` hace referencia a la clase System del paquete `java.lang`. Las clases que guardamos en un archivo cuando escribimos un programa, pertenecen al paquete predeterminado a menos que hayamos creado uno para su guardado.

- **Import**: Sentencia necesaria para poder “invocar” todos las clases y paquetes necesarios (contenidos en otros packages) para que la aplicación funcione. Por ejemplo, al tratar de utilizar una conexión hacia una base de datos, necesitaremos importar los paquetes que proporcionan la funcionalidad necesaria. Algunos paquetes de JAVA se importan automáticamente, no es necesario hacer un import explícito, como la librería `java.lang`. También se puede importar un paquete completo de clases utilizando como comodín un asterisco en lugar del nombre especifico de una clase, ej: `import java.lang.*`

- **Class**: Definición del nombre de clase. Se determina el nombre identificador que contendrá la clase.

- **Variables de ámbito de clase**: Se determinan las variables que se usarán en la clase para guardar datos y hacer operaciones. Estas corresponden a los atributos que tendrá el objeto instanciado de la clase.

  ```
  [modificador] tipo-dato nombreAtributo [= valor_inicializacion];
  ```

- **Constructor de la clase**: El constructor es un tipo especial de método, que se invoca automáticamente al hacer una instancia del objeto. Sirve para inicializar los objetos, las variables o cualquier configuración inicial que necesite la clase para funcionar. Se puede sobrecargar.

- **Métodos**: Bloques de código que contienen las instrucciones y declaraciones que ejecutarán algunas tareas especificas del programa. Los métodos exponen la interfaz de una clase. La definición de un método consta de una cabecera y del cuerpo del método encerrado entre llaves. Si el tipo-resultado es `void`, el método no devolverá nada, no hace falta sentencia `return`.

  ```
  [modificador] tipo-resultado nombreMetodo (lista de parámetros) {
        declaraciones de variables locales;
        sentencias;
        [return (expresión)]
  }
  ```

Los modificadores de los atributos y los métodos corresponden al nivel de protección que estos

tendrán, pueden ser paquete, publico, privado y protegido.

Un miembro de una clase declarado como **private** puede ser accedido unicamente por los métodos de su clase.

Un miembro de una clase declarado **public** es accesible desde cualquier método definido dentro o fuera de la clase o paquete actual.

A un miembro de una clase declarado como **protected** solo se puede acceder desde la propia clase o desde una clase que hereda de ella.

Generalmente los atributos de una clase de objetos se declaran privados, estando así ocultos para otras clases, siendo posible el acceso a los mismos unicamente a través de los métodos públicos de dicha clase. Esta practica se conoce como encapsulación, que corresponde al proceso de ocultar la estructura interna de datos de un objeto y permitir el acceso solo a través de la interfaz publica definida, o sea, los miembros públicos de la clase.

El nivel de protección predeterminado para un miembro de una clase es el de paquete. Un miembro de una clase con este nivel puede ser accedido desde todas las otras clases del mismo paquete.

El modificador **static** en la declaración de un atributo de una clase almacena información común a todos los objetos de esa clase y existe aunque no haya objetos definidos de esa clase.

1.9.2 Método main

Toda aplicación JAVA tiene un método denominado **main**, y solo uno. Este método es el punto de entrada a la aplicación y también el punto de salida.

```
public static void main(String[] args){
        // cuerpo del método
}
```

El método **main** es público (**public**) para poder ser accedido desde el exterior de la clase de ser necesario, estático (**static**) porque es un método común para todos los objetos de la clase y no devuelve nada (**void**) y tiene un argumento de tipo String que almacenara los argumentos pasados en la linea de comandos cuando se invoque la aplicación para su ejecución.

1.9.3 Crear y manipular objetos de una clase

Las clases son como plantillas para crear objetos. Para crear un objeto de una clase se debe utilizar el operador **new**.

```
CGrados grados = new Cgrados();
```

Observamos que para crear el objeto `grados` debemos especificar luego del operador **new** el nombre de la clase del objeto seguido de paréntesis, ya que corresponde al método predeterminado especial denominado igual que la clase que es necesario invocar para crear un objeto; este método se denomina constructor de la clase.

www.profmatiasgarcia.com.ar

Otro ejemplo; para crear un objeto de una clase de la biblioteca JAVA como puede ser el paquete `java.util` que proporciona la clase GregorianCalendar que permite crear objetos que representan una fecha y opcionalmente la hora podremos usar:

```
GregorianCalendar fecha1 = new GregorianCalendar();
GregorianCalendar fecha2 = new GregorianCalendar(2016, 3, 22);
GregorianCalendar fecha3 = new GregorianCalendar(2016, 3, 22, 12, 45, 15);
```

Los objetos `fecha1` `fecha2` `fecha3` son validos porque la clase GregorianCalendar proporciona varias formas de construir un objeto, todas difieren en la cantidad de parámetros que tiene cada uno. Por defecto, JAVA crea automáticamente un constructor sin parámetros para cualquier clase que definamos. En cuanto definimos un constructor, ya no podemos utilizar el constructor por defecto de la clase.

Cuando se crea un nuevo objeto utilizando **`new`**, JAVA asigna automáticamente la cantidad de memoria necesaria para ubicar ese objeto. Si no hubiera suficiente espacio de memoria disponible, el operador **`new`** lanzara una excepción OutOf-MemoryError. JAVA se encarga de liberar la memoria en cuanto el objeto no se utilice, o sea, cuando ya no exista ninguna referencia al mismo. Para tal efecto JAVA cuenta con una herramienta denominada **Garbage Collector** que busca objetos que no se utilizan con el fin de destruirlos y liberar la memoria que ocupan.

Cuando tenemos un objeto de un tipo determinado y queremos acceder a uno de sus miembros sólo tenemos que poner el identificador asociado al objeto (esto es, el identificador de una de las variables de nuestro programa) seguido por un punto y por el identificador que hace referencia a un miembro concreto de la clase a la que pertenece el objeto. Por ejemplo: `cuenta1.saldo = 355;`

De esta forma, para enviar un mensaje a un objeto se invocara a uno de sus métodos. Por ejemplo: `cuenta1.asignarLimite(3000);` La llamada al método hace que el objeto realice la tarea especificada en la implementación del método, tal como esté definida en la definición de la clase a la que pertenece el objeto.

1.9.4 Sobrecarga de métodos

Lenguajes como JAVA permiten que existan distintos métodos con el mismo nombre siempre y cuando su signatura no sea idéntica (algo que se conoce con el nombre de sobrecarga). El nombre de un método, los tipos de sus parámetros y el orden de los mismos definen la signatura de un método. En la creación de constructores es importante disponer de esta característica.

No es válido definir dos métodos con el mismo nombre que difieran únicamente por el tipo del valor que devuelven.

De todas formas, no conviene abusar demasiado de esta prestación del lenguaje, porque resulta propensa a errores (en ocasiones, creeremos estar llamando a una “versión” de un método cuando la que se ejecuta en realidad es otra).

www.profmatiasgarcia.com.ar

Ejemplo:

```
System.out.println(...);
- System.out.println()
- System.out.println(boolean)
- System.out.println(char)
- System.out.println(char[])
- System.out.println(double)
- System.out.println(float)
- System.out.println(int)
- System.out.println(long)
- System.out.println(Object)
- System.out.println(String)
```

Recuerda que al llamar a un método, el método utiliza una copia local de los parámetros (que contiene los valores con los cuales fue invocado). Como las variables de tipos no primitivos son, en realidad, referencias a objetos, lo que se copia en este caso es la referencia al objeto (no el objeto en sí). Como consecuencia, podemos modificar el estado de un objeto recibido como parámetro si invocamos métodos de ese objeto que modifiquen su estado. La referencia al objeto no cambia, pero sí su estado.

1.10 CONVENCIONES DE PROGRAMACIÓN

- **Paquetes (Packages)**: El nombre de los paquetes deben ser sustantivos escritos con minúsculas. `package conversor.temperatura`
- **Clases (Classes)**: Los nombres de las clases deben ser sustantivos, mezclando palabras, con la primera letra de cada palabra en mayúscula (capitalizada). Recuerda que las clases públicas deben estar definidas en archivos con extensión *.JAVA* cuyo nombre coincida exactamente con el identificador asignado a la clase (¡ojo con las mayúsculas y las minúsculas, que en JAVA se consideran diferentes!). `class MiClase`
- **Interfaces (Interfaces)**: El nombre de la interfaz debe ser capitalizado como los nombres de las clases. `interface CuentaCorriente`
- **Métodos (Methods)**: Los métodos deben ser nombrados con verbos, mezclando palabras, con la primera letra en minúscula. Dentro del nombre del método, la primera letra de cada palabra capitalizada. `asignarTemperatura()`
- **Variables (Variables)**: Todas las variables deben ser combinaciones de palabras, con la primera letra en minúscula. Las palabras son separadas por las letras capitales. Moderar el uso de los guiones bajos, y no usar el signo de pesos ($) porque tiene un significado especial dentro de las clases. `char primerLetra` Las variables deben tener significados transparentes para exponer explícitamente su uso al lector casual. Evitar usar simples caracteres (i, j, k) como variables excepto para casos temporales como ciclos.
- **Constantes (Constants)**: Las constantes deben nombrarse totalmente en mayúsculas, separando las palabras con guiones bajos. Las constantes tipo objeto pueden usar una mezcla de letras en

www.profmatiasgarcia.com.ar

mayúsculas y minúsculas. `static final double PI = 3.1416;`

- **Estructuras de Control (Control Structures)**: Usar Llaves ({ }) alrededor de las sentencias, aun cuando sean sentencias sencillas, son parte de una estructura de control, tales como un if-else o un ciclo for.
- **Espaciado (Spacing)**: Colocar una sentencia por línea, y usar de dos a cuatros líneas vacías entre sentencias para hacer el código legible. El número de espacios puede depender mucho del estándar que se use.
- **Sangrado (Indented)**: Se debe sangrar el texto que aparece entre las llaves para que resulte más fácil delimitar el ámbito de los distintos elementos de nuestro programa. De esta forma se mejora la legibilidad de nuestro programa.
- **Comentarios (Comments)**: Utilizar comentarios para explicar los segmentos de código que no son obvios. Utilizar `//` para comentar una sola línea; para comentar extensiones más grandes de información encerrar entre los símbolos delimitadores `/*    */`.

Utilizar los símbolos `/**    */` para documentar los comentarios para proveer una entrada al JAVAdoc y generar un HTML con el código.

```
/** Un comentario para propósitos de documentación.
* @author nombre del autor del código
* @see otra clase para más información
*/
```

Nota: la etiqueta @see es especial para JAVAdoc para darle un efecto de “también ver” a un link que referencia una clase o un método.

www.profmatiasgarcia.com.ar

BIBLIOGRAFÍA

- Berzal Galiano, Fernando, "Apuntes de programación orientada a objetos en JAVA: Fundamentos de programación y principios de diseño" (2006)
- Ceballos, Fco. Javier, "JAVA 2 Curso de programación" 4ta Ed. (Ra-Ma 2010)
- Deitel, Paul y Deitel, Harvey, "JAVA Cómo programar" 9na Ed. (Pearson 2012)
- Eckel, Bruce, "Piensa en JAVA" 4ta Ed. (Prentice-Hall 2007)
- Kuhn, Mónica, "Apuntes de Programación II" INSPT/UTN (2014)
- Otero, Abraham, "Tutorial básico de JAVA" 3ra Ed. (JAVAhispano.org 2007)
- Pérez, Gustavo Guillermo, "Aprendiendo JAVA y Programación Orientada a Objetos" (2008)
- Sánchez, Jorge, "JAVA 2" (2004)

LICENCIA

Este documento se encuentra bajo Licencia Creative Commons 2.5 Argentina (BY-NC-SA), por la cual se permite su exhibición, distribución, copia y posibilita hacer obras derivadas a partir de la misma, siempre y cuando se cite la autoría del **Prof. Matías E. García** y sólo podrá distribuir la obra derivada resultante bajo una licencia idéntica a ésta.

Matías E. García

Prof. & Tec. en Informática Aplicada
www.profmatiasgarcia.com.ar
info@profmatiasgarcia.com.ar

www.profmatiasgarcia.com.ar