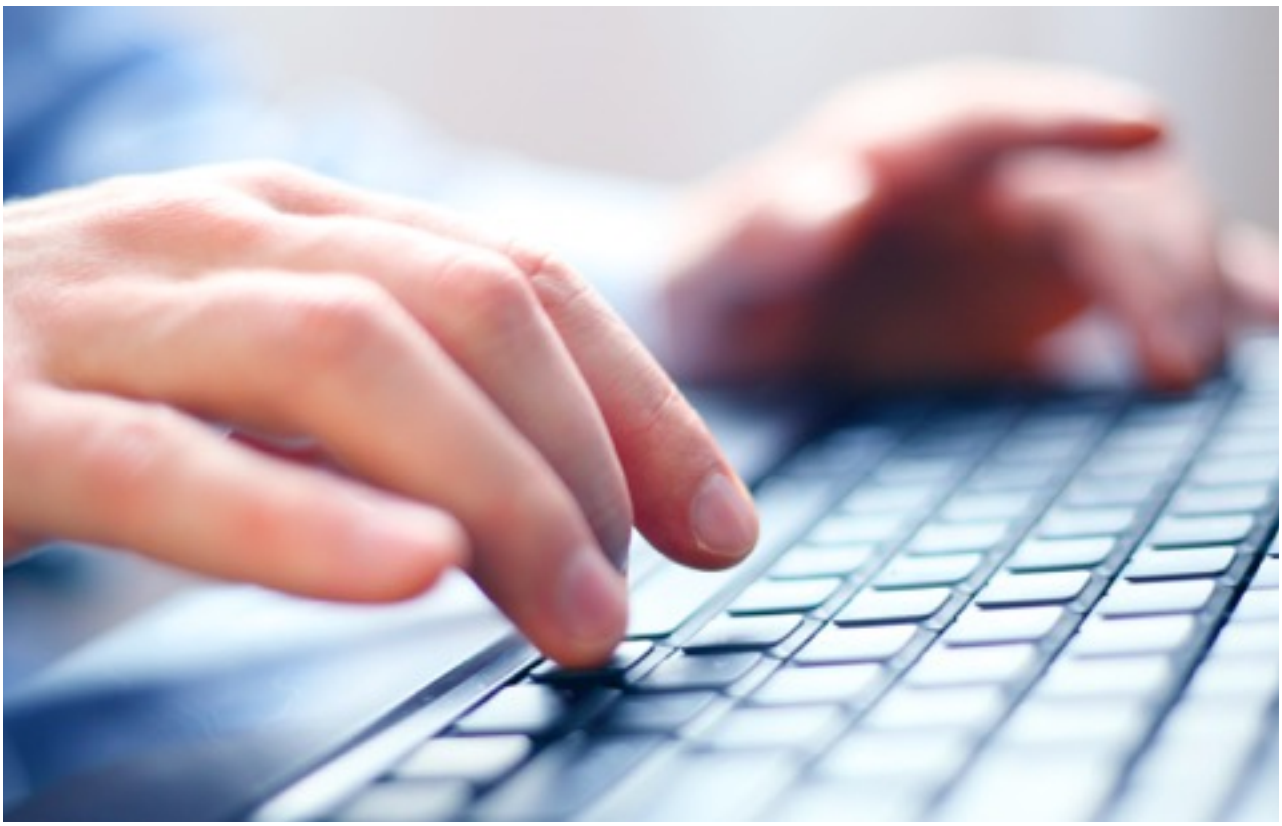
Introducción a Java

Fundamentos de la Programación.



Curso de Programación en Java
Juan Francisco Maldonado León

Introducción

Las computadoras están especialmente diseñadas para todas aquellas aplicaciones que requieren una operación o conjunto de ellas que deben repetirse muchas veces.

Un tipo muy importante de estructura es el algoritmo necesario para repetir una o varias acciones un numero determinado de veces.

Capítulo 2

Lenguaje de programación Java

Introducción

Un lenguaje de programación es aquel elemento dentro de la informática que nos permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que pone a disposición del programador para que este pueda comunicarse con los dispositivos hardware y software existentes.

Los lenguajes de programación se determinan según el nivel de abstracción, según la forma de ejecución y según el paradigma de programación que poseen cada uno de ellos, detallados a continuación:

Según su nivel de abstracción

- *Lenguajes de bajo nivel*: Los lenguajes de bajo nivel son lenguajes de programación que se acercan al funcionamiento de una computadora. El lenguaje de más bajo nivel es, por excelencia, el código máquina. A éste le sigue el lenguaje ensamblador, ya que al programar en ensamblador se trabajan con los registros de memoria de la computadora de forma directa.
- *Lenguajes de medio nivel*: Hay lenguajes de programación que son considerados por algunos expertos como lenguajes de medio nivel (como es el caso del lenguaje C) al tener ciertas características que los acercan a los lenguajes de bajo nivel pero teniendo, al mismo tiempo,

ciertas cualidades que lo hacen un lenguaje más cercano al humano y, por tanto, de alto nivel.

- *Lenguajes de alto nivel* Los lenguajes de alto nivel son normalmente fáciles de aprender porque están formados por elementos de lenguajes naturales, como el inglés. Por ejemplo Visual Basic permite construir aplicaciones armando elementos en un contenedor, sin siquiera tener que escribir las sentencias para ello.

Según la forma de ejecución

- *Lenguajes compilados*: Naturalmente, un programa que se escribe en un lenguaje de alto nivel también tiene que traducirse a un código que pueda utilizar la máquina. Los programas traductores que pueden realizar esta operación se llaman compiladores. Éstos, como los programas ensambladores avanzados, pueden generar muchas líneas de código de máquina por cada proposición del programa fuente. Se requiere una corrida de compilación antes de procesar los datos de un problema. Los compiladores son aquellos cuya función es traducir un programa escrito en un determinado lenguaje a un idioma que la computadora entienda (lenguaje máquina con código binario). Al usar un lenguaje compilado, el programa desarrollado nunca se ejecuta mientras haya errores, sino hasta que luego de haber compilado el programa, ya no aparecen errores en el código
- *Lenguajes interpretados:* Se puede también utilizar una alternativa diferente de los compiladores para traducir lenguajes de alto nivel. En vez de traducir el programa fuente y grabar en forma permanente el código objeto que se produce durante el proceso de compilación para utilizarlo posteriormente, el programador sólo carga el programa fuente en la computadora junto con los datos que se van a procesar. A continuación, un programa intérprete, almacenado en el sistema

operativo del disco, o incluido de manera permanente dentro de la máquina, convierte cada proposición del programa fuente en lenguaje de máquina conforme vaya siendo necesario durante el proceso de los datos. No se graba el código objeto para utilizarlo posteriormente. la siguiente vez que se utilice una instrucción, se le debe interpretar otra vez y traducir a lenguaje máquina. Por ejemplo, durante el procesamiento repetitivo de los pasos de un ciclo, cada instrucción del ciclo tendrá que volver a ser interpretado cada vez que se ejecute el ciclo, lo cual hace que el programa sea más lento en tiempo de ejecución (porque se va revisando el código en tiempo de ejecución) pero más rápido en tiempo de diseño (porque no se tiene que estar compilando a cada momento el código completo). El intérprete elimina la necesidad de realizar una corrida de compilación después de cada modificación del programa cuando se quiere agregar funciones o corregir errores; pero es obvio que un programa objeto compilado con antelación deberá ejecutarse con mucha mayor rapidez que uno que se debe interpretar a cada paso durante una corrida de producción.

Según el paradigma de programación

Un paradigma de programación representa un enfoque particular o filosofía para la construcción del software. No es mejor uno que otro sino que cada uno tiene ventajas y desventajas. También hay situaciones donde un paradigma resulta más apropiado que otro. Atendiendo al paradigma de programación, se pueden clasificar los lenguajes en:

Programación imperativa

- BASIC
- COBOL
- C

- Pascal

Programación funcional

- Puros: Haskell , Miranda, Erlang
- Híbridos: Lisp, Écheme, Ocaml, Standard ML, ML , Scala

Programación lógica

- Prolog

Programación orientada a objetos

- Simula
- Smalltalk
- C++
- C#
- VB.NET
- Visual FoxPro
- Eiffel
- Java
- Objective-C

Historia del lenguaje Java

El lenguaje de programación Java, fue diseñado por la compañía **Sun Microsystems Inc.**, con el propósito de crear un lenguaje que pudiera funcionar en redes computacionales heterogéneas (redes de computadoras formadas por más de un tipo de computadora, ya sean PC, MAC's, estaciones de trabajo, etc.), y que fuera independiente de la plataforma en la que se vaya a ejecutar. Esto significa que un programa de Java puede ejecutarse en cualquier máquina o plataforma.

Inicialmente fue un proyecto que se transfirió durante mucho tiempo por distintos departamentos de SUN, como un lenguaje de pequeños electrodomésticos.

El mercado inicial de JAVA (a cargo de la filial de Sun Microsystems, FirsPerson Inc.) eran los equipos como microondas, tostadores y fundamentalmente televisión interactiva.

James Gosling, miembro del equipo, con mas experiencia en lenguajes de programación había estado trabajando en su tiempo libre en un lenguaje de programación que el llamó OAK, cuyas bases fueron C++.(1990), y es el precursor de Java

Características del lenguaje Java

El lenguaje de programación Java se caracteriza por los siguientes logros de diseño:

Simple. Elimina la complejidad de los lenguajes como "C" y da paso al contexto de los lenguajes modernos orientados a objetos. Orientado a Objetos. La filosofía de programación orientada a objetos es diferente a la programación convencional.

Familiar. Como la mayoría de los programadores están acostumbrados a programar en C o en C++, la sintaxis de Java es muy similar al de estos.

Robusto. El sistema de Java maneja la memoria de la computadora por ti. No te tienes que preocupar por punteros, memoria que no se esté utilizando, etc. Java realiza todo esto sin necesidad de que uno se lo indique.

Seguro. El sistema de Java tiene ciertas políticas que evitan se puedan codificar virus con este lenguaje. Existen muchas restricciones, especialmente para los applets, que limitan lo que se puede y no puede hacer con los recursos críticos de una computadora.

Portable. Como el código compilado de Java (conocido como bytecode) es interpretado, un programa compilado de Java puede ser utilizado por cualquier computadora que tenga implementado el interprete de Java.

Independiente a la arquitectura. Al compilar un programa en Java, el código resultante un tipo de código binario conocido como byte code. Este código es interpretado por diferentes computadoras de igual manera, solamente hay que implementar un intérprete para cada plataforma. De esa manera Java logra ser un lenguaje que no depende de una arquitectura computacional definida.

Multithreaded. Un lenguaje que soporta múltiples threads (multihilos) es un lenguaje que puede ejecutar diferentes líneas de código al mismo tiempo.

Interpretado: Java corre en máquina virtual, por lo tanto es interpretado.

Dinámico. Java no requiere que compiles todas las clases de un programa para que este funcione. Si realizas una modificación a una clase Java se encarga de realizar un Dynamic Bynding (enlace dinámico) o un Dynamic Loading (carga dinámica) para encontrar las clases.

Java puede funcionar como una aplicación sola o como un "applet", que es un pequeño programa hecho en Java. Los applets de Java se pueden "pegar" a una página de Web (HTML), y con esto se puede tener un programa que cualquier persona que tenga un browser compatible podrá usar.

Compilador, Máquina Virtual de Java, Bytecode

La Máquina virtual de Java (JVM) es un programa que se ejecuta en todos los computadores. La JVM crea un software de simulación de una CPU y de una memoria. Gestiona todas las comunicaciones entre un programa Java, el sistema operativo y el hardware. El programa Java piensa que la computadora que se ejecuta en la JVM.

La JVM toma y traduce el bytecode (código binario resultante de la compilación) en código binario para la CPU que se utiliza para ejecutar el programa. Este código binario también es conocido como código nativo o código del procesador. En versiones anteriores de 1,2 de Java, la JVM es principalmente era un intérprete para el bytecode y no un compilador para el bytecode. Las actuales versiones de Java HotSpot incluir un

compilador "just in time". El compilador HotSpot permite a la JVM compilar el bytecode en el código nativo para una CPU al momento de la ejecución.

El lenguaje Java es compilado, esto significa que todo programa de java debe pasar el siguiente proceso:

- El programa se compila , es decir el compilador de Java (Javac) revisa la sintaxis de las instrucciones del programa, verificando que estén bien escritas, respetando las reglas establecidas por el lenguaje
- Si hay errores, estos se despliegan en la pantalla para que sean resueltos
- Si no hay errores se genera un archivo con código intermedio (bytecode), que es lenguaje binario.
- Este bytecode ingresa a la Máquina Virtual de Java, quién lo convierte a código Nativo y lo ejecuta.

El JDK, Java Development Kit, es el set de Desarrollo y ejecución de Java, tiene el siguiente contenido:

- Herramientas y utilidades del JDK (bin)
- Las librerías del JDK (lib)
- Los archivos C/C++ utilizados para construir la JVM (include)
- Una variedad de ejemplos escritos en Java (demo)
- JRE (Java Runtime Environment), es la JVM sin herramientas de desarrollo
- El código fuente de las APIs comprimido (src.zip)

El JDK está compuesto de:

- Compilador de Java (Javac)
- Intérprete de Java (Java.exe)
- Intérprete de Applets (los Applets son aplicaciones java que corren en un browser) (Appletviewer)
- Depurador de Java (jdb)
- Generador de Documentación (javadoc)
- Integrador de C y C++ (JNI) (javah)
- Desamblador (javap)

Versiones del Lenguaje Java:

Las versiones son mejoras que se realizan a lo que ya existe, entre ellas tenemos:

Java 1.0: el lenguaje en sí.

Java 1.1.0: Esta versión incorpora una gran cantidad de funcionalidades entre las más importantes java RMI, JavaBeans, nuevo modelo de eventos, JDBC.

Java 1.2: Una de las principales características JFC (Java Foundations classes, una biblioteca que contiene los componentes de Swing)

Java 1.3: No agrega funcionalidades, solo corrige procesos.

Java 1.4: Incorpora nuevas mejoras, como las Aserciones, llamada java 2.0

Java 1.5: Se le conoce como el Tigger de Java, o Java 5.0, provee generics, metadatos, static import, administración del garbage collector (recolector de basura de Java), clases genéricas, argumentos variables, enumeraciones.

Java 6.0: (Mustang) incorpora un API para desarrollo Web propio, crea clases para la interacción con la Consola, mejoras para Java en la red.

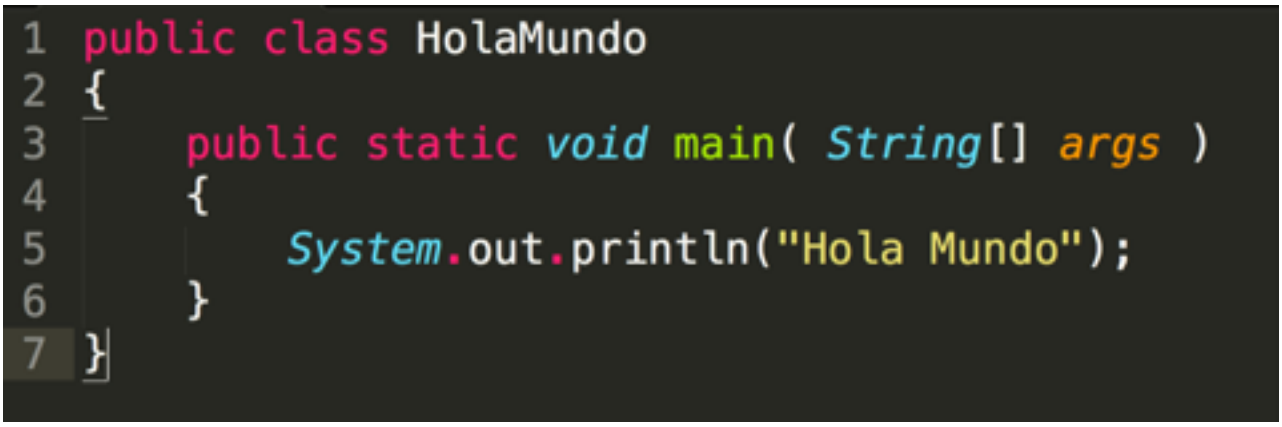
Java 7 (Dolphin**)** Soporte para XML dentro del propio lenguaje, un nuevo concepto de superpaquete, Soporte para closures, Introducción de anotaciones estándar para detectar fallos en el software, entre otras.

Java 8 Diferentes mejoras en seguridad y concurrencia. Añade funcionalidad para programación funcional mediante expresiones Lambda, Mejora la integración de JavaScript, Nuevas API para manejo de fechas y tiempo (date - time), entre otras.

Nuestro Primer Programa

Consideremos una aplicación sencilla que muestre una linea de texto que diga “Hola Mundo”.

En un editor de texto escribiremos lo siguiente.



* Los números no son parte de lo que escribiremos en el editor.

La línea 1, comienza con una declaración de clase, para la clase *HolaMundo*. Todo programa en java consiste al menos de una declaración de clase. La palabra reservada *class* introduce una declaración de clase, la cual debe ir seguida inmediatamente por el nombre de la clase (HolaMundo).

Por convención, todos los nombres de clases en Java comienzan con una letra mayúscula, y la primera letra de cada palabra en el nombre de la clase debe ir con mayúscula, por ejemplo EjemploDeNombreDeClase.

El nombre de la clase se conoce como identificador, y esta compuesto por una serie de caracteres que pueden ser letras, dígitos, guiones bajos y

signos, sin embargo, no puede comenzar con un dígito ni tener espacios en blanco. Java es sensible a mayusculas y minúsculas; es decir, las toma como identificadores diferentes, por lo que a1 y A1 son distintos, pero ambos son identificadores validos.

Cuando usted guarda su declaración de clase en un archivo, el nombre de este debe ser el nombre de la clase, seguido de la extension ".java". Para nuestro programa el nombre del archivo debe ser HolaMundo.java.

Una llave izquierda (línea 2), **{**, comienza el cuerpo de la declaración de una clase. Su correspondiente llave derecha (linea 7), **}** , debe terminar la declaración de una clase. Observe que las lineas de la 3 a la 6 tienen sangría. Esta es una convención de espaciado para identificar lo que esta dentro del cuerpo de la clase.

La línea 3 ("*public static void main(String[] args)*") es el punto de inicio de toda aplicación en Java. Los paréntesis después de **main** indican que este es un bloque de construcción del programa, al cual se le llama método. Las declaraciones de clases en java generalmente contiene uno o mas métodos. En una aplicación en java, solo uno de estos métodos debe llamarse **main** y debe definirse como muestra la línea 3; de no ser así, el interprete java no ejecutara la aplicación.

La llave izquierda en la línea 4 comienza el cuerpo de la declaración del método. Su correspondiente llave derecha (línea 6) debe terminar el cuerpo de la declaración del método.

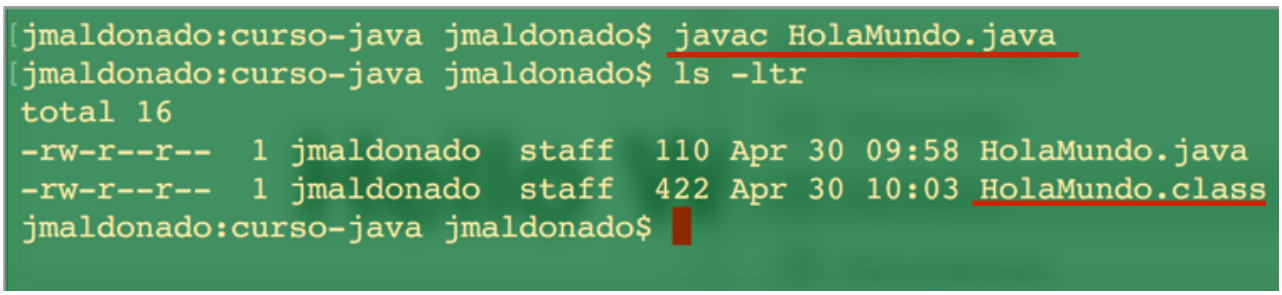
La línea 5 (*System.out.println("Hola Mundo")*), indica a la computadora que realice un acción; es decir que imprima la cadena de caracteres contenida entre los caracteres de comillas dobles. A una cadena también se le denomina String o cadena de caracteres. El compilador no ignora los caracteres de espacio en blanco dentro de las cadenas.

Compilar y Ejecutar

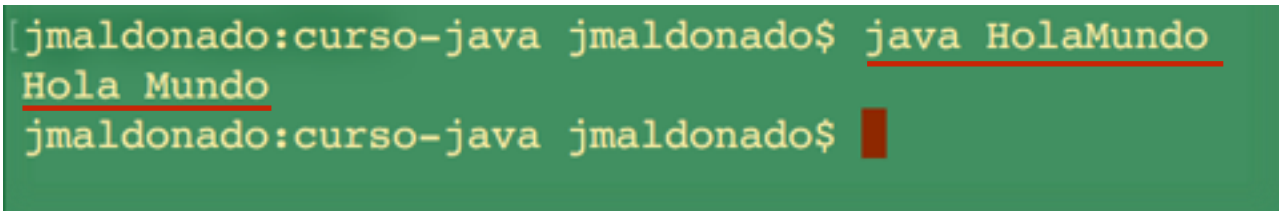
Ahora estamos listos para compilar y ejecutar nuestro programa. para compilar el programa, abra una ventada de comandos, cambie al directorio donde esta guardado el programa y escriba lo siguiente.

```
javac HolaMundo.java
```

Si el programa no contiene errores de sintaxis, el comando anterior crea un nuevo archivo llamado HolaMundo.class, conocido como el archivo de clase para HolaMundo, el cual contiene el Byte Code de Java que representan nuestra aplicación. Estos códigos de byte serán interpretados por el interprete java cuando le indiquemos que debe ejecutar el programa.



Para ejecutar el programa escribimos **java HolaMundo** , con lo que se indica al interprete de Java que debe cargar el archivo .class para la clase HolaMundo. Observe que la extension ".class" del nombre del archivo se omite en el comando anterior; de no ser así, el interprete no ejecutara el programa. El interprete llama al método main. A continuación, la instrucción de la linea 5 muestra "Hola Mundo".



Fundamentos de Programación Orientada a Objetos

Objetos

La Programación Orientada a Objetos trata de utilizar una visión real del mundo dentro de nuestros programas. Esta visión que se tiene del mundo dentro, se encuentra formada por objetos.

Un objeto es un elemento o entidad del mundo real o del espacio posible de conceptos, que puede ser claramente definido y representado, por ejemplo, un elemento tangible, Una casa.

Todo objeto del mundo real tiene:

- **Identidad**: La identidad es la propiedad que permite a un objeto diferenciarse de otros. Generalmente esta propiedad es tal, que da nombre al objeto. Tomemos por ejemplo el "verde" como un objeto concreto de una clase color; la propiedad que da identidad única a este objeto es precisamente su "color" verde. Tanto es así que para nosotros no tiene sentido usar otro nombre para el objeto que no sea el valor de la propiedad que lo identifica.
- **Características**: es la propiedad que se refiere a las cualidades o atributos que posee un objeto al que se le deben asignar valores
- **Comportamientos**: es la propiedad que se refiere a las operaciones que se pueden realizar con el objeto

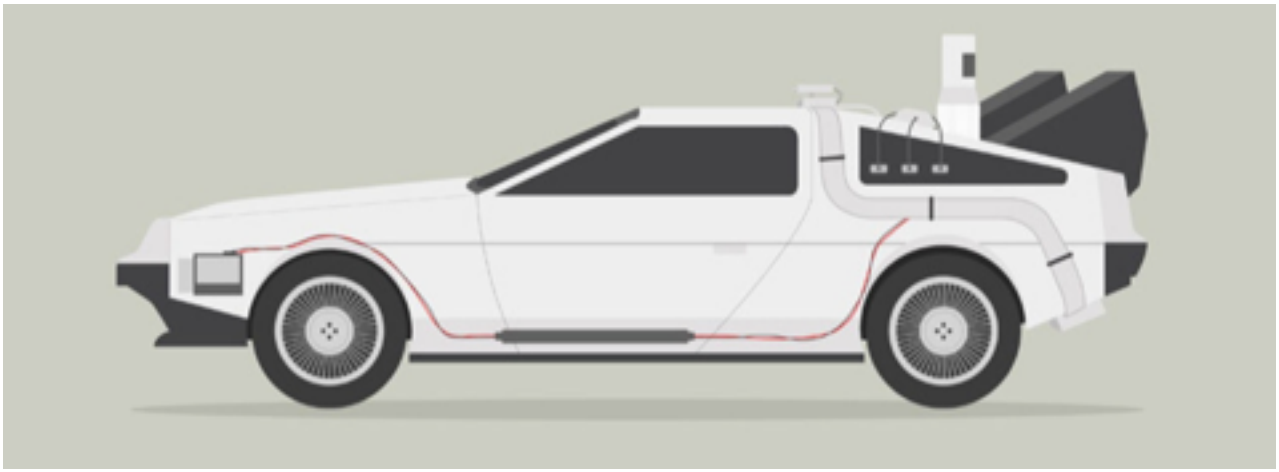
- **Estado**: es la propiedad que se refiere a los valores de los atributos en un momento determinado
- **Tiempo de vida**: La duración de un objeto en un programa siempre está limitada en el tiempo. La mayoría de los objetos sólo existen durante una parte de la ejecución del programa. Los objetos son creados mediante un mecanismo denominado instanciación, y cuando dejan de existir se dice que son destruidos.

Por ejemplo, los automóviles tienen características

- **marca** : DMC
- **modelo** : DeLorean
- **cantidad de puertas** : 2
- **velocidad máxima** : 175 km/h

Comportamiento de un auto.

- frenar
- acelerar
- retroceder



Clases

Las clases son declaraciones de objetos, también se podrían definir como abstracciones de objetos. Esto quiere decir que la definición de un objeto es la clase en sí. Cuando programamos un objeto y definimos sus características y comportamientos, en realidad lo que estamos haciendo es programar una clase. Por tanto una clase es un molde de un concepto que plasmamos en cada objeto que se crea de ella.

En el mundo real, normalmente tenemos muchos objetos del mismo tipo. Por ejemplo, nuestro teléfono celular es sólo uno de los miles que hay en el mundo. Si hablamos en términos de la programación orientada a objetos, podemos decir que nuestro objeto celular es una instancia de una clase conocida como "celular". Los celulares tienen características (marca, modelo, sistema operativo, pantalla, teclado, etc.) y comportamientos (hacer y recibir llamadas, enviar mensajes multimedia, transmisión de datos, etc.).

Teóricamente una clase es un modelo o prototipo que define las variables y métodos comunes a todos los objetos de cierta clase. También se puede decir que una clase es una plantilla genérica para un conjunto de objetos de similares características.

Atributos

Se entiende por atributo a toda aquella característica o cualidad que identifica a un objeto en particular, el que pertenece a una cierta clase. También es cada una de las características que definen un elemento.

La clase a la que pertenece el objeto decide qué atributos tendrá, por tanto cada objeto se instancia o crea con el mismo molde de la clase, lo que varía es los valores en los atributos.

Por ejemplo: **Clase Ropa**

Atributos:

- Tipo
- Color
- Talla

Un objeto de la Clase Ropa: Vestido de Fiesta

- Tipo: de fiesta
- Color: verde
- Talla: medium

Los atributos siempre se representan por sustantivos: tipo, color, talla.

Comportamientos

El comportamiento de un objeto está directamente relacionado con su funcionalidad y determina las operaciones que este puede realizar o a las que puede responder ante mensajes o solicitudes enviados por otros objetos. La funcionalidad de un objeto está determinada, primariamente, por su responsabilidad.

Se entiende, entonces, que un comportamiento es una acción que se puede realizar con un objeto determinado, el que está implementado por la clase y que tiene acceso a todos los componentes de dicha clase.

Por ejemplo: Clase Ropa

Comportamientos:

- Lavarla
- Usarla
- Comprarla
- Confeccionarla
- Repararla.

Los comportamientos se denominan **Métodos**, y son quienes tienen acceso directo a los atributos de la clase, pudiendo mostrar el contenido del atributo, cambiar el contenido del atributo o manipularlo con operadores.

Estado de un Objeto

El estado de un objeto se refiere al conjunto de atributos que describen a un objeto así como sus valores en un instante de tiempo dado. El comportamiento de un objeto puede modificar el estado de éste. Cuando una operación de un objeto modifica su estado se dice que esta ha mutado o cambiado su estado.

Por ejemplo: en la clase Ropa, el objeto Vestido tiene color verde, entonces su atributo color es verde, si cambio el color (teñido) a azul, ahora el color es azul, es decir el atributo color ha cambiado, lo que lleva a que el estado del objeto ha cambiado.

Características:

- El estado de un objeto evoluciona con el tiempo.
- Algunos atributos pueden ser constantes.
- El comportamiento agrupa las competencias de un objeto y describe las acciones y reacciones de ese objeto.
- Las operaciones de un objeto son consecuencia de un estímulo externo representado como mensaje enviado desde otro objeto
- Cada objeto está en un estado en cierto instante.
- El estado está caracterizado parcialmente por los valores algunos de los atributos del objeto.

- El estado en el que se encuentra un objeto determina su comportamiento.
- Se puede especificar el ejecutar una acción como consecuencia de entrar, salir, estar en un estado, o por la ocurrencia de un evento.

Reglas de Sintaxis

Definición de Sintaxis: Forma correcta en que deben estar dispuestos los símbolos que componen una instrucción ejecutable por el ordenador.

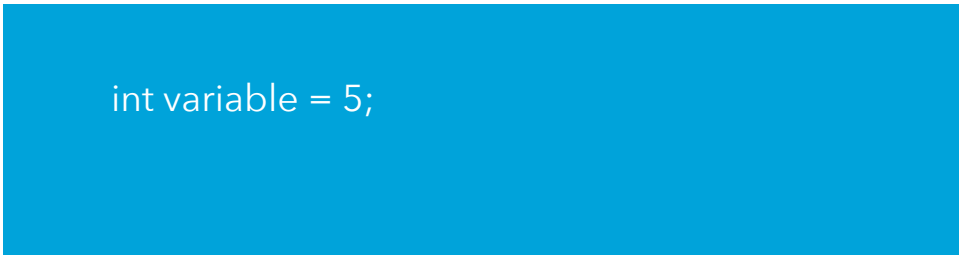
Elementos básicos de Sintaxis:

- El uso de llaves: "{}"

  Este símbolo se utiliza para determinar bloques de código, es decir un conjunto de sentencias Java que se ubican dentro de estas llaves.

- El uso de punto y coma: " ; "

  Este símbolo permite finalizar una sentencia java.



- **Uso de espacios**: Java permite el uso de espacios en blanco entre elementos del código fuente (Clase)

- **Indentación del código**: Todos los desarrolladores utilizan algún estilo de colocación de los elementos del código fuente dentro del texto que los contiene. Esto se define como el estilo de codificación. Las normas de indentación indican la posición en la que se deben colocar los diferentes elementos que se incluyen en el código fuente, por lo que forman parte del estilo de codificación. Otro ejemplo de ello es la separación con espacios en blanco entre los diferentes elementos que componen las líneas de código

  El objetivo fundamental de la indentación del código fuente es facilitar su lectura y comprensión. Hay dos tipos de posibles lectores del código fuente: programas y personas. A los programas les da igual la indentación, leen bien nuestro código siempre que cumpla la sintaxis del lenguaje. Luego la indentación debe centrarse en la lectura y comprensión del código por personas.

  Por ejemplo:

  Un código **NO** identado:

```
public void setValor(int valor)
{
this.valor = valor;
}
```

Un código SI identado

```
public void setValor(int valor)
{
        this.valor = valor;
}
```

Se puede observar que es más fácil la lectura del segundo código que del primero.

- **Identificador**: Un identificador es un "nombre" o secuencia de caracteres, que nos permite dirigirnos específicamente a una de las entidades propias del lenguaje, es decir, son los nombres que podemos ponerles a nuestros/as variables, métodos, clases, interfaces y objetos.

  La única restricción en la formación de identificadores es que ***tienen que comenzar*** **por letra, guión bajo (underscore) o por el signo '$',** pudiéndoles seguir después letras o números. Además puede estar formado por dos o más palabras

  En Java como en otros muchos lenguajes de programación se distinguen las mayúsculas y las minúsculas.

  Ejemplo de identificadores válidos, según la sintaxis del lenguaje:

nombreUsuario

nombre_usuario

_nombre

_nombre_usuario

$nombre_usuario

nombre1

El identificador **nunca debe coincidir con una palabra reservada del lenguaje**. Una palabra reservada es aquella que tiene un significado único dentro de la sintaxis del lenguaje, por tanto cada vez que la utilizamos debe significar lo mismo, por eso no debemos usar estas palabras en identificadores.

Comentarios en Java

Un comentario es una frase que comienza con un carácter especial y que la JVM no los revisa, solo los detecta. En otras palabras los comentarios permiten documentar las distintas partes del código, ya sea internamente (para el desarrollador) o externamente (para la generación de API)

Tipos de comentarios en Java

- **Comentario de Línea**. Este comentario se inicializa con "**//**" y se finaliza con el término de la línea (o al hacer <enter>, y su utilidad es para resaltar ciertas partes de código que el desarrollador está ordenando, por ejemplo:

```
// comienzo de las declaraciones
```

- **Comentario de Multilínea:** Este comentario se inicializa con "**/***" y finaliza con "***/**", permitiendo contener dentro de esos dos símbolos, varias líneas de información. Cada archivo fuente (archivo.java) debe llevar un comentario inicial de multilínea indicando los datos generales del proceso que se está codificando, por ejemplo:

```
/*      este es un comentario
        de multilíne
        en java
*/
```

- **Comentario de Documentación**: Este comentario se inicializa con "**/****" y finaliza con "***/**". Este tipo de comentario son utilizados para la documentación que es generada por la utilidad Javadoc. Esta información es la que aparece en las API de cada clase y permiten conocer la utilidad de la clase y la forma de utilizarla.

  Además, en este comentario se agregan marcadores que permiten determinar ciertas características propias de cada método o de la clase en particular.

  Los marcadores comienzan con el carácter "**@**" y siguen con una palabra especial, que se detallan a continuación:

  Palabra reservada 1.0

  Descripción

  @author El autor del método, clase.

  @deprecated El método ha sido reemplazado por otro.

  @exception La excepción que puede generar.

  @param Después viene el parámetro, esto se hace para documentar todos los parámetros que requiere el método.

  @return El valor que regresa

  @see Una referencia a otra clase o método de utilidad

  @since La versión desde que se incluye.

  @throws Es un sinónimo para @exception.

  @version La versión del método, clase.

Por ejemplo:

```
/**
   Esta clase define objetos de tipo Animal
    @author Juan Maldonado León
    @version 1.0
    @since 1.0
*/
```

Convenciones de Escritura

Convenciones para escribir identificadores:

Una convención es un acuerdo o estándar que se debe respetar en un cierto ámbito. En java existen las convenciones de escritura del lenguaje que están detalladas en el documento **Java Language Epecification**, creado por SUN.

Existen muchas razones por qué usar estas convenciones, pero dentro de ellas podemos destacar las siguientes:

1. Las convenciones de código mejoran la lectura del software, permitiendo entender código nuevo mucho más rápidamente y más a fondo.
2. Si distribuyes tu código fuente como un producto, necesitas asegurarte de que está bien hecho y presentado como cualquier otro producto.

3. Para que funcionen las convenciones, cada persona que escribe software debe seguir la convención. **Todos**

El estándar para escribir identificadores es:

- Los nombres de clases deben comenzar con letra mayúscula y seguir con caracteres en minúscula.
- Los nombres de variables deben comenzar con letra minúscula y escribirse completamente en minúscula.
- Los **nombres de métodos** deben escribirse completamente en minúscula.
- Los nombres de constantes deben escribirse completamente en mayúscula.
- Cuando un identificador está compuesto de dos o más palabras, desde la segunda palabra debe comenzar con letra mayúscula, el resto con minúscula.

Convenciones para escribir bloques de código:

Un bloque de código es una o más sentencias del lenguaje Java, y estas siempre deben ir entre llaves, por ejemplo:

```
{
    sentencia de java;
}
```

Convenciones para escribir una Clase

El orden en que se escribe una clase es:

A. Comentarios javadoc encabezado
B. Declaración de la clase
C. Declaración de Atributos
D. Declaración de métodos
E. Constructores
F. Accesadores y Mutadores
G. Comportamientos adicionales.

Estructura de una clase

Todo en java es una clase, por lo que se hace indispensable conocer como debemos codificar cada clase personal, qué reglas debemos respetar y que elementos del lenguaje usaremos par esto.

Lo primero es entender qué elementos constituyen una clase, para esto definiremos la Anatomía de una clase, la que se expresa en el cuadro de abajo:

```
/*Anatomía de una clase*/

public class NombreClase //cáscara
{
        declaraciones de atributos
        definiciones de constructor(es)
        definiciones de métodos accesadores y mutadores
        definición de métodos
 }
```

Una clase se compone de los siguientes elementos:

1. **Cáscara:** se refiere al encabezado de la clase, cuya sintaxis es:

[modificador de acceso] **class** NombreClase

Donde:

modificador de acceso: determina la accesibilidad que tendrán otras clases sobre ella.

class: palabra reservada del leguaje que determina la definición de una clase

NombreClase: identificador que representa el nombre de la clase.

2. Declaración de campos o atributos:

Un campo es una variable, es decir un espacio de memoria que tiene un tamaño determinado (lo da el tipo de dato), un valor a asignar, que debe ser compatible con el tipo de datos y un identificador asociado a dicho espacio, que es su acceso. La declaración de atributos va debajo de la cáscara, a continuación de la llave que abre el bloque de código completo.

Sintaxis de declaración:

[modificador de acceso] tipo de datos nombreCampo;

Donde:

modificador de acceso: determina la accesibilidad que tendrán otras clases sobre ella.

Los atributos de utilizan **private** (que permite mantener protegida la información, es decir no hay acceso a ellos desde fuera de la clase).

Tipo de Datos: se refiere a que valores son posibles que la variable guarde y en qué espacio debe hacerse. Para esto java provee dos **categorías** en los tipos de datos: Primitivos y Complejos.

Tipo Primitivos: son aquellos que están más cerca de la máquina, y son:

- **boolean**: permite almacenar valores de tipo verdadero o falso, ocupan 1 bit de almacenamiento y pueden contener los siguientes valores: true o **false.**
- **char:** permite almacenar valor de tipo carácter, requiere 16 bits (2 bytes) de almacenamiento y pueden contener cualquier carácter que se define en la tabla UNICODE codificación internacional. Se inicializan con un carcacter entre comillas simple: **'a'.**
- **byte:** permite almacenar valores numéricos enteros de baja precisión, ocupan 16 biots (2 bytes) de almacenamiento, y puede tener valores en el rango **-128 a 127**
- **short:** permite almacenar calores numéricos enteros de precisión simple, requiere 16 bits (2 bytes) de almacenamiento, y puede tener valores en el rango **–32.768 a 32.767**

- **int:** permite almacenar valores numéricos enteros, ocupan 32 bits (4 bytes) de almacenamiento, y puede tener valores en el rango de **-2.147.483.648 a 2.147.483.648.** Es el tipo de dato por defecto de todos los literales de tipo int(por ejemplo: 4, 2344556).
- **long:** permite almacenar valores numéricos entero de doble precisión, requiere 64 bits (8 bytes) de almacenamiento, y puede tener valores en el rango: -9,223,372,036,854,775,808L a 9,223,372,036,854,775,807L
- **float:** permite almacenar valores numérico con decimales de precisión simple (o coma flotante), requiere 32 bits (4 bytes) de almacenamiento, y puede contener valores +/- 3.4E+38F (6-7 dígitos significativos).
- **Double:** permite almacenar valores numérico con decimales de precisión doble (o coma flotante), requiere 64 bits (8 bytes) de almacenamiento, y puede contener valores +/- 1.8E+308 (15 dígitos importantes)

nombreCampo: corresponde al identificador del atributo o campo, debe comenzar con minúscula y debe usar un conjunto de caracteres que identifique lo que se quiere representar, por ejemplo: el campo color se refiere al colo de algo, si le hubiésemos puesto campo1 deberíamos haber explicado que queríamos representar, en cambio como lo definimos no es necesario.

Tipo Complejos: en esta categoría están todas las referencias de objetos de Clases que existan, por tanto el tipo de dato depende del nombre de la clase.

Para poder explicar este tema, pensemos en las cápsulas de los remedios, sabemos que dentro de ellas hay sustancias que tienen un fin, pero no las vemos. Esto es similar para las clases y sus atributos, si permitimos que cualquier clase desde fuera manipule nuestros atributos, probablemente la clase no se comporte como esperamos y los datos que guardemos no sean consistentes con lo que se quiere lograr.

El encapsulamiento consiste en unir en la Clase las características privadas y comportamientos públicos, esto es, las variables y métodos. Es tener todo esto es una sola entidad.

Para esto Java provee modificadores de acceso que permiten determinar el acceso a los distintos componentes de la clase, ya sean campos o métodos. Ellos son:

- public: este modificador permite que el acceso sea libere a ese miembro de la clase
- private: este modificador no permite acceso a ese miembro desde fuera de la clase, sino solamente interno.

Entonces las variables de instancia son siempre private y los métodos son generalmente public.

Constructores

Después de definir una clase en Java esta queda lista para ser utilizada. Por tanto lo primero que se debe hacer para usarla es construir objetos de dicha clase, para esto existe un método encargado de construir objetos, es decir de darle valores iniciales a los atributos

Antes de revisar la sintaxis del método constructor debemos aprender dos conceptos previamente:

Asignación

Para poder darle valores iniciales a cualquier variable, incluidos los campos o atributos, se requiere utilizar la operación de Asignación, cuya misión es precisamente guardar valores en las referencias de los indicadores. El símbolo de esta operación es "=".

Esta operación asigna el valor del lado derecho del signo igual a la referencia del identificador del lado izquierdo del signo igual.

Sintaxis:

```
color = "c"; //esta sentencia asigna el carácter c al identificador color.
```

Parámetro

Se entiendo por parámetro a aquellas variable que permite ingresar valores a un método, para esto deben ser declarados con el tipo de dato que vana permitir entrar. Esta declaración es de la siguiente manera:

```
[Tipo de dato]  nombreParametro
```

Donde:

Tipo de dato: corresponde al tipo de dato primitivo o complejo que va a contener la variable

nombreParametro: Corresponde al identificador de la variable

Se declaran tantos parámetros como sea necesario, y ellos se separan con el símbolo "," en la declaración.

Sintaxis del método constructor:

El método constructor lleva el mismo identificador que el usado en el nombre de la clase

```
/**
  Comentario javadoc
*/
[modificador de acceso]    NombreClase([parámetros])
{
        campoUno = valor;
        campados = valor;
         …
}
```

Donde:

/…*/: c**omentario javadoc donde se debe especificar cuál es la utilidad del método constructor

Modificador de acceso: determina la accesibilidad que tendrán otras clases sobre

NombreClase: corresponde al identificador dl nombre del método constructor

Parámetros: son los parámetros de entrada al métod0, no siempre lleva parámetros, pero si los lleva cada uno debe ser definido como se explicó anteriormente.

Dentro del bloque de código que encierran las llaves van las asignaciones de valores iniciales a todos los campos o atributos de la clase

- **public Lapiz()** es el constructor de la clase Lapiz que no recibe parámetros
- **public Lapiz(char categoria, double tamaño, int cantidadColores, boolean esTransparente)** es el constructor de la clase Lapiz que recibe parámetros para asignarle valor a cada uno de los campos de la clase.
- **categoria= 'a'**; asigna el carácter 'a' al atributo categoria
- **this.categoria=categoria;** asigna el valor del parámetro categoria al atributo categoria, en este caso como los dos identificadores son iguales es necesario indicar cual de los dos es el atributo, para esto se antepone el modificador **this** seguido de un punto (“.”), al campo.

Accesadores y Mutadores

Los atributos de la clase son declarados como privados (private) eso significa que es imposible poder acceder o cambiar su contendido desde fuera de la clase. Para poder programar estos dos comportamientos básicos existen ciertos métodos que tienen una función única y específica dentro de la clase, y que su ausencia podría significar que la Clase no es operable. Ellos son: método Accesador y método Mutador.

Método Accesador

El método Accesador tiene como objetivo mostrar el contenido de un campo o atributo a quien lo solicite.

Este método utiliza la sentencia return, que hace dos cosas

1. Entrega el valor de respuesta del método
2. Provoca el fin de la ejecución del método inmediatamente

Cabe destacar que la única regla del uso es que el tipo de dato de la expresión al lado derecho del return debe ser compatible con el tipo de retorno de la firma del método, es decir, si la variable al lado derecho del return es de tipo char, el tipo de retorno del método debe ser char.

Cualquier cosa que quede bajo el return no se va a ejecutar nunca, por lo tanto, el return debe ser la última instrucción del método

Sintaxis:

```
/**  javadoc  */
[modificador de acceso]  tipo de retorno  getNombreCampo()
{
    return nombreCampo;
}
```

Como se ve en el ejemplo La primera línea del método es el **encabezado o firma.** El resto del método (lo que va entre llaves) es el **cuerpo**.

- El **encabezado o firma** muestra quien puede tener acceso a este método, que tipo de dato retorna y el nombre. Esto es lo que se necesita conocer para llamar al método.

  Por convención del lenguaje, el identificador de todos los métodos accesadores comienzan con la palabra get, seguido del nombre del campo pero comenzado con mayúscula, por ejemplo: **getColor.**

  Antes de la firma se ubica el comentario javadoc que debe indicar la responsabilidad del método y el valor de retorno con el marcado @return

- El **cuerpo** es el conjunto de instrucciones que le dicen que hacer al computador cuando el método es invocado, en este caso se activa el return y el valor es retornado.

Método Mutador

El método Mutador tiene como objetivo o responsabilidad permitir el cambio de valor para un atributo en particular, esto a través de un parámetro que recibe el nuevo valor y la asignación que permite almacenar el nuevo valor en la referencia indicada (campo).

Sintaxis:

```
[modificador de acceso] void setNombreCampo(tipo dato parametro)
{
    nombreCampo = parametro;
}
```

En este caso se reemplaza el tipo de retorno por la palabra reservada void que indica que el método no retorna valor alguno, por tanto no debe llevar la sentencia return.

Como se ve en el ejemplo La primera línea del método es el **encabezado o firma.** El resto del método (lo que va entre llaves) es el **cuerpo**.

El **encabezado o firma** muestra quien puede tener acceso a este método, la palabra reservada void, el nombre y que información requiere el método. Esto es lo que se necesita conocer para llamar al método.

Por convención del lenguaje, el identificador de todos los métodos mutadores comienzan con la palabra set, seguido del nombre del campo pero comenzado con mayúscula, por ejemplo: **setColor.**

Antes de la firma se ubica el comentario javadoc que debe indicar la responsabilidad del método y el valor del parámetro de entrada con el marcado @param.

El **cuerpo** es el conjunto de instrucciones que le dicen que hacer al computador cuando el método es invocado, en este caso se asigna el valor del parámetro al identificador del lado izquierdo del signo igual (operador de asignación).