

Programando con Java para aplicaciones Android

Indice

Módulo 1: Introducción.....	2
Generalidades y componentes de java.....	2
Módulo 2: Básicos de Java.....	4
Objetivo.....	4
Hola mundo.....	4
IDES.....	7
HolaMundo en Eclipse.....	7
Tipos de datos.....	10
Enteros.....	10
Float y Double.....	11
Char y boolean.....	11
Naming y cast de variables.....	12
Arreglos.....	12
Operadores.....	14
Operadores aritméticos y de asignación.....	14
Operadores de incremento y decremento.....	16
Operadores de Equidad y Relacionales.....	16
Operadores Lógicos.....	17
Control de Flujo.....	18
Introducción.....	18
If, else, switch.....	18
While, for y foreach.....	21
Módulo 3: Programación orientada a objetos.....	24
Objetivo.....	24
Introducción.....	24
Clases.....	25
Declaración de métodos.....	26
Declaración e instanciado de objetos.....	28
Creación de objetos.....	28
Métodos constructores.....	29
Utilizando objetos y accedendo a métodos.....	29
Métodos y miembros estáticos.....	30
import.....	31
Sobrecarga de métodos y constructores.....	32
Sobrecarga de constructores.....	32
Ejemplo de sobrecarga.....	34
Modificadores de acceso.....	35
Ejemplo de modificadores de acceso.....	36
Herencia y Polimorfismo.....	37
Herencia.....	37
Polimorfismo.....	40
Ejemplo de Herencia y Polimorfismo.....	41

Getters y Setters.....	41
Para qué sirven.....	41
Objetos vs variables.....	44
Módulo 4: Lenguaje adicional de Java.....	47
Objetivo.....	47
Bibliografía.....	47

Módulo 1: Introducción

Generalidades y componentes de java

Es muy importante antes de que comiences a tirar cualquier línea de código, conocer la filosofía de Java. La filosofía de Java es Write Once Run Anywhere. Esto significa que el código que desarrolles podrá correr en todos lados. ¿Qué significa en todos lados? Bueno, en cualquier sistema operativo, you sea que tú utilices un sistema operativo Mac, un sistema operativo Windows o un sistema operativo Linux. La máquina virtual hace todo el trabajo para que no te desgastes desarrollando código para cada sistema operativo. Así que, ¿qué es la máquina virtual? Bueno, no te preocupes, vamos a verlo más adelante.

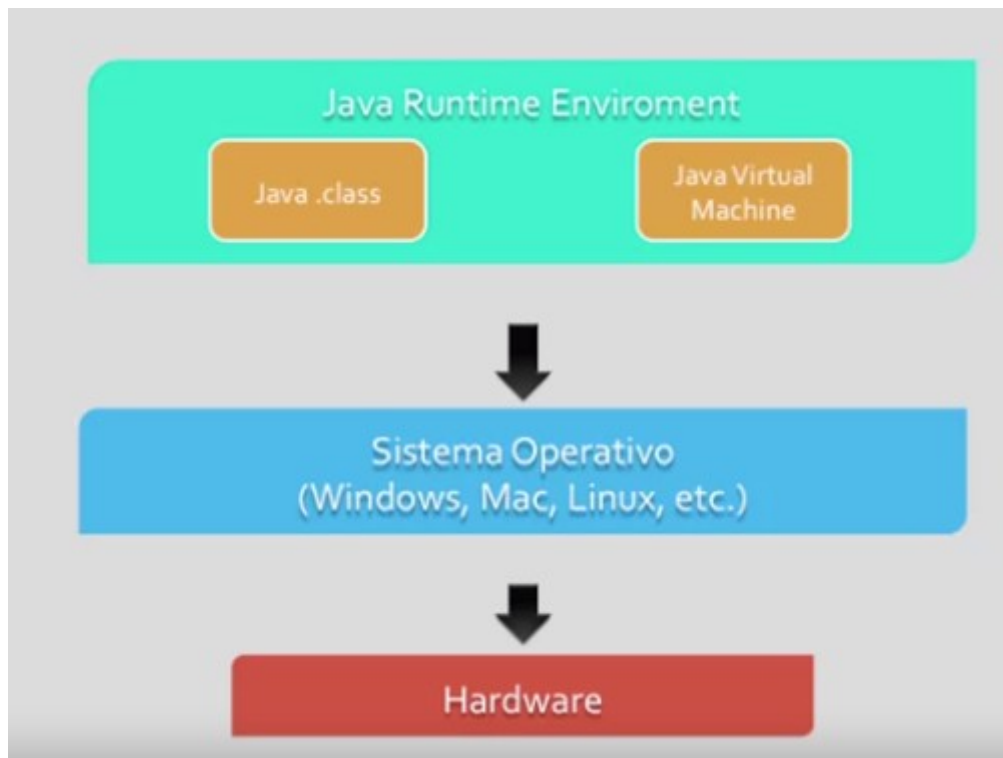
Lo siguiente que debes saber es que Java maneja varias versiones. Varias versiones que puedes encontrar para desarrollar en algunos entornos de desarrollo, o en algunos IDE:

- **Java EE.** Así es normalmente conocida pero significa Java Enterprise Edition.
se ha utilizado comúnmente para desarrollar proyectos en web, proyectos que tienen que involucrar un navegador web e incluso también un entorno de desarrollo en backend.
- **Java Server Enterprise (Java SE)** Java Standard Edition, esa es la que se utiliza comúnmente para desarrollar nuestra aplicaciones móviles
- **Java Micro Edition (J3ME)** que se ha desarrollado para enfocarse en dispositivos de muy muy bajo nivel. ¿Cómo? Pues, dispositivos que tenemos en nuestra vida diaria, como un control de remoto, licuadora, etcétera. Muchos electrodomésticos que tenemos en nuestra casa, funcionan con esta versión de Java, Java Micro Edition.

¿Qué es el JRE de Java?

El JRE de Java es mejor conocido como la máquina virtual. La máquina virtual es aquella que hace toda la magia de nuestros programas.

Lo que hace ella es que toma todos estos archivos compilados de Java. Esos archivos compilados los transforma y se los da a tu sistema operativo, para que lo pueda leer fácilmente, tu sistema operativo Windows, Linux o Mac, lo que utilices. Tu sistema operativo está encargado de mantener la conexión con el hardware, tal cual como se está viendo en la siguiente figura.



¿Qué es el JDK de Java?

El JDK, que significa Java Development Kit, es decir, es un kit para que tú desarrolles tus programas en Java. Lo que contiene este kit de desarrollo son todas las clases y todas las librerías que Java you tiene listas para tí para que las puedas utilizar y que puedas partir de ahí para desarrollar nuevos programas de Java.

Bien, ya hemos elegido qué versión de Java utilizaremos para desarrollar nuestras aplicaciones. Java Standard Edition. Ya hemos hablado de los componentes que necesitas para desarrollar tus programas, el JDK (en el curso usa JDK8) de Java y también el JRD (Java Runtime Environment) de Java.

A continuación te voy a explicar, cómo puedes observar si tu computadora ya tiene instalado alguno de estos componentes. Fíjate bien, lo primero que tenemos que saber es, abrir una ventana de línea de comandos. Si estás en Windows, lo que tienes que abrir es el CMD. Pero si estás en Mac o en Linux, puedes abrir la terminal. Los comandos que tienes que ejecutar son los siguientes, los puedes ver aquí en tu pantalla, el primero es:

```
java -version
```

Este comando te dirá, qué versión de Java tienes instalado, si es que tienes you instalado alguna. El siguiente comando es el comando `javac`. Este comando nos sirve para verificar si tienes instalado el compilador de Java, que es muy importante para que podamos desarrollar nuestros programas.

Módulo 2: Básicos de Java

Objetivo

Creación de un programa en Java siguiendo una matriz específica

Hola mundo

Muy bien, una vez teniendo instalados todos nuestros componentes. ya tenemos la versión de Java adecuada y también tenemos el compilador de Java. Vamos a comenzar nuestro primer programa, nuestro "Hola mundo" de Java. Lo único que necesitas para este ejercicio es simplemente tu editor de código favorito, el que quieras. Puedes utilizar Notepad, si es que así te gusta, o puedes utilizar cualquier otro editor de código. Y segundo, necesitarás la terminal o tu consola de comandos. Vamos a hacerlo. Bien, comenzaré a hacer mi primer "Hola mundo".

Lo primero que voy a hacer es crear en el escritorio una carpeta que se llame Java (el nombre de la carpeta es a gusto del consumidor, lo importante es que, al ejecutar se trabaje en el mismo directorio).

Ahora desde la consola, iré a esa dirección. Antes de tirar cualquier línea de código, lo primero que haré será guardar mi archivo.

Para guardar este archivo, colocaré Hola con h mayúscula, Mundo, sin espacios, y con la extensión .java.

Más adelante te voy a explicar por qué utilizamos esta sintaxis al guardar nuestros archivos. Vamos a comenzar por definir la clase. Public class, y el nombre de la clase debe tener exactamente el mismo nombre del archivo (sin la extensión).

```
public class HolaMundo{  
}
```

Las llaves me indicarán que todo lo que yo coloque dentro pertenecerá a esta clase.

Ahora vamos a comenzar a definir un método.

Que es un método especial, el cual es nuestro método main. `Public static void main()`. Y este método especial, lo que recibe es un arreglo de `String`, que se puede llamar `args`. Voy a abrir mis llaves y a cerrar.

```
public class HolaMundo{  
    public static void main(String[] args){  
  
    }  
}
```

De la misma forma que con la clase, todo lo que yo coloque dentro de estas llaves pertenecerá a este método main.

Ahora, ¿qué es el método main?

El método main, como su nombre lo dice, es el método principal de nuestra aplicación. ¿Qué quiere decir con principal? Principal porque será el método de entrada de nuestro programa. Todo lo que queremos que muestre nuestro programa, o todo lo que queremos que se reúna, que se conjunte, en nuestro programa, debe estar declarado en este método. Si no lo declaras, es como si no existiera. Si tú creas una clase o creas un objeto, cualquier cosa que tú quieras que cobre vida, debe estar definido en el método main. Lo único que voy a hacer dentro de este método, es simplemente imprimir el mensaje que diga "Hola Mundo".

Voy a comenzar escribiendo con ese mayúscula `System.out.println`.

No olvides el punto y coma. Todas las instrucciones en Java siempre deben llevar un punto y coma.

Dentro de estos paréntesis, voy a colocar unas comillas y ahí imprimiré mi mensaje "Hola Mundo :)".

```
public class HolaMundo{  
    public static void main(String[] args){  
        System.out.println("Hola Mundo :)");  
    }  
}
```

Listo, ya tenemos escrito nuestro primer código de Java, nuestro primer Hola Mundo. Lo siguiente, ahora es ejecutarlo.

Para ejecutarlo, espero que recuerdes nuestros comandos que vimos al principio. El comando java, y el comando javac. Abriremos nuestra terminal para ejecutar esos comandos.

Debemos estar posicionados en la carpeta que contiene este programa o este archivo. Lo primero que hay que hacer es compilar el programa. Es decir, verificar que el programa no tenga errores. Para eso se usa el compilador, el javac.

Voy a colocar javac, espacio, HolaMundo.java. Y le voy a dar Enter.

Listo, al parecer pasó la prueba, no nos salió ningún mensaje de error.

Si hubiésemos tenido algún mensaje de error, por ejemplo, que yo hubiera olvidado colocar el punto y coma. Si lo quitamos, lo guardamos y compilamos de nuevo, inmediatamente el compilador nos indica que tenemos un error. Dice que tenemos un error en la línea tres y que el error es que esperaba un punto y coma. Efectivamente, estamos en la línea tres y necesitamos colocar el punto y coma. Bien, lo ponemos nuevamente, lo guardamos y, de nuevo, vamos a compilar.

Ya que hemos compilado nuestro programa, como puedes observar en este lado, nos ha generado un archivo .class.

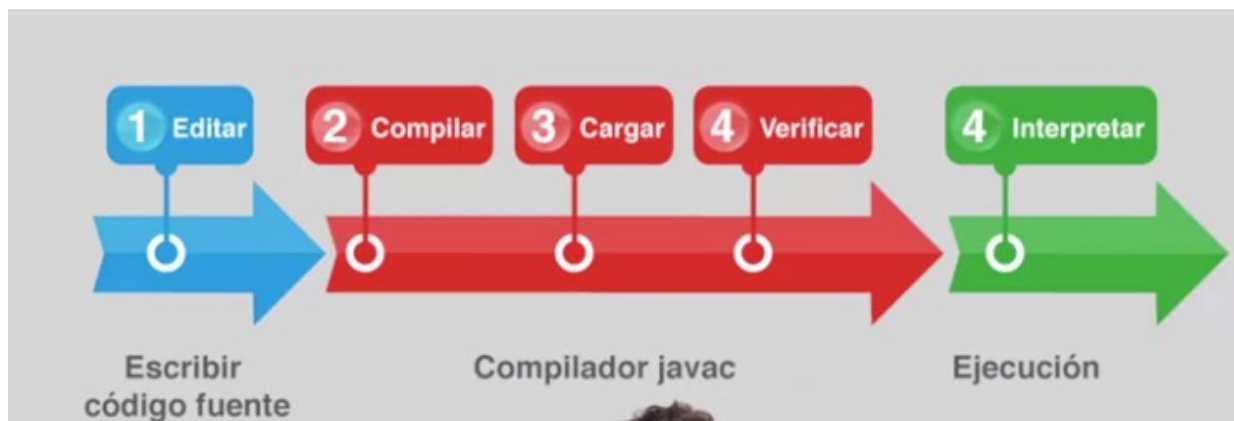
Ahora no sólo tenemos el archivo .java, ahora también tenemos un archivo .class. Bien, vamos a correr nuestro programa. Utilizaré el comando java y pondré HolaMundo pero sin la extensión:

```
java HolaMundo
```

Le damos enter y voilá, nos aparecen nuestro mensaje "Hola Mundo :)". Bien, vamos a explicar cómo es que sucedió todo esto.

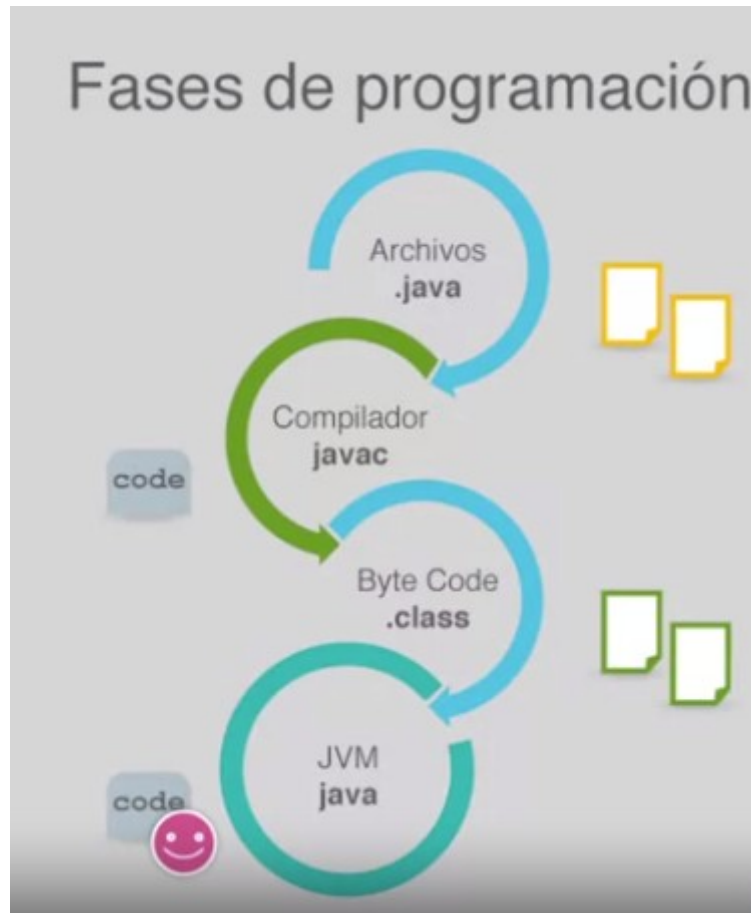
Para eso necesitamos conocer las fases de la programación en java. Las fases de la programación en java son sumamente sencillas. Únicamente constan de tres pasos:

1. Editar el código. Es básicamente lo que hicimos al principio, escribir nuestra clase. Escribir nuestro método main y todo el código que contenga el archivo.
2. Compilar el código. Aquí es dónde comienza el trabajo de nuestro comando javac. Aquí se llevan a cabo tres fases más.
 1. Compilación
 2. Carga.
 3. Verificación. El comando javac, lo que hace es que verifica que todo nuestro código esté bien escrito y que coincida con las reglas de sintaxis de Java. Una vez hecho esto nos genera un archivo .class.
3. La ejecución de nuestro programa. Aquí es donde nuestra máquina virtual de Java entra en acción. Lo que hace la máquina virtual de java, es que lee ese archivo .class. La máquina virtual nunca va a leer los archivos de código, es decir tus archivos .java. A la máquina virtual, lo único que le interesan son los archivos ejecutables, es decir los archivos .class. Cuando la máquina virtual tiene esos archivos .class, lo que hace es que interpreta.



Archivos .class

Esos archivos .class están definidos como archivos de Byte Code, es decir, lenguaje máquina. Lo que tu hardware o la máquina o lo que tengas de muy bajo nivel, necesita leer. La máquina virtual hace ese trabajo (JVM Java Virtual Machine). Esos archivos .class, los traduce finalmente en lo que puedes ver en pantalla. Es decir, nuestro "Hola Mundo :)".



IDES

IDE es un acrónimo que se define como Integrated Development Environment, es decir un entorno de desarrollo integrado. Pero, ¿qué cosas tiene integradas? Bueno, como acabamos de ver teníamos nuestro editor de código y teníamos también nuestra terminal. Justamente lo que tiene integrado un IDE es eso. Tenemos también nuestro depurador de código y un constructor de interfaz gráfica. Esto es solamente cuando estamos desarrollando interfaces gráficas. Si nosotros solamente desarrollamos puro código, pues no tendremos acceso a esta opción. El entorno de desarrollo nos va a ayudar a que todo sea mucho más fácil. A que ya no utilicemos los comandos. A que simplemente apretando un botón automáticamente el depurador se pone en marcha. Automáticamente el compilador se pone en marcha. El entorno de desarrollo que estaremos utilizando para este curso será Eclipse. Eclipse, es un entorno de desarrollo que tiene muchos años de experiencia, como te comenté hay muchos otros más. Está también NetBeans y también, si te gusta, está IntelliJ.

HolaMundo en Eclipse

Para iniciar un programa nuevo de cero, se va desde el menú File → Other → java project

Cuando se hace ésto, se crea el proyecto y se crean nuevas carpetas, siendo una de ellas, src, que es donde va el código fuente, y ahí se crean los packages. Para crear uno nuevo, se hace click derecho sobre la carpeta src y luego New → Package.

Los packages son un sistema de carpetas que permiten controlar mejor el proyecto y sus nombres son `com.<nombre significativo>.<nombre del proyecto>`, por ejemplo `com.jenifer.holamundo`.

`com.<nombre significativo>` es el company domain o sea, si uno pertenece a una compañía en internet, eso es lo que va en `<nombre significativo>`, lo que hace es funcionar como un identificador que hace que distingas la compañía que desarrolla la aplicación.

En cuanto a `<nombre del proyecto>` va exactamente eso, todo en minúsculas.

Una vez creado el package, sobre el archivo del package, en mi caso `com.jenifer.holamundo`, se crea una clase nueva con click derecho New → Class, y en el nombre de la clase se pone el nombre del archivo sin extensiones, en nuestro caso `HolaMundo`.

Algo importante acá es que hay que señalarle que ésta clase es la que contiene el main como se ve en la figura.

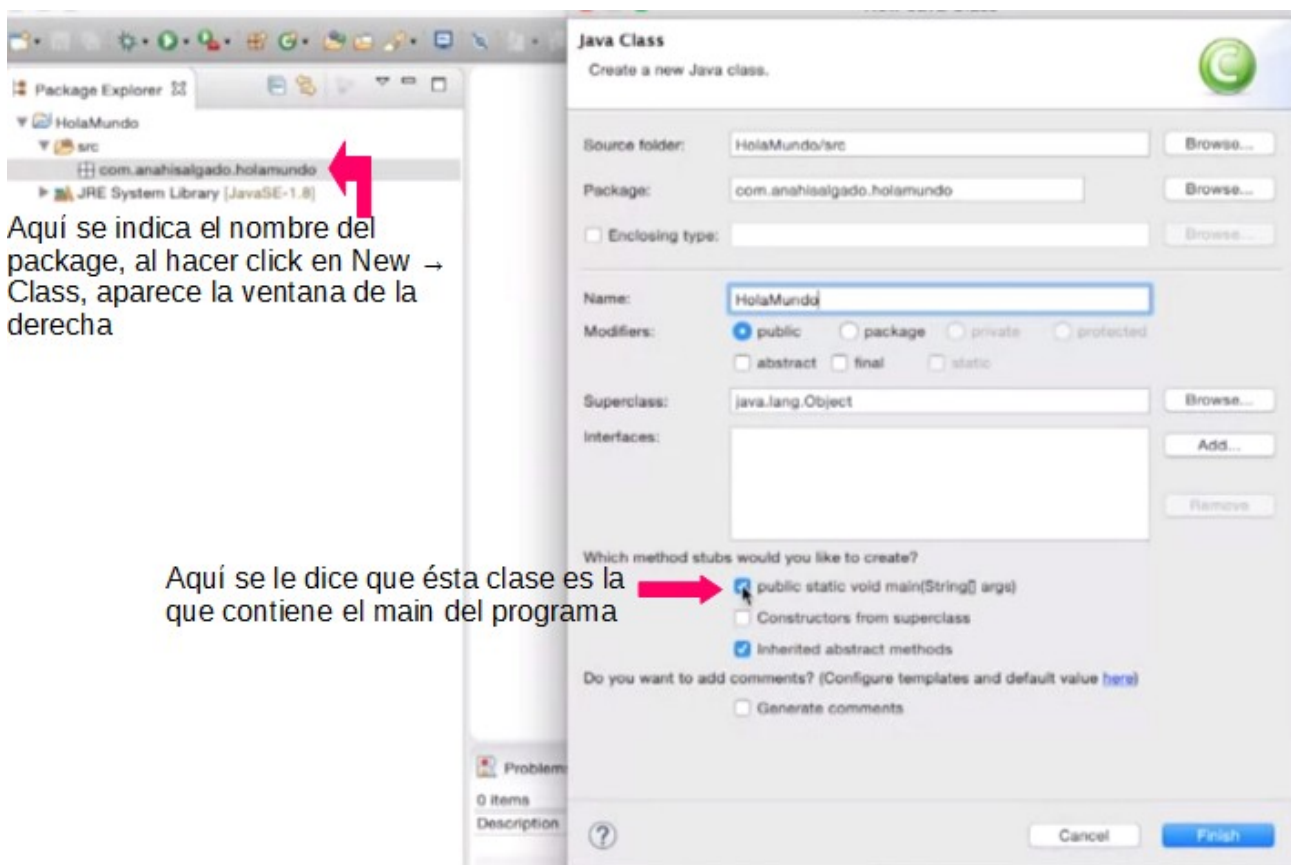


Figura 1: Aquí se ve detallado cómo se crea una clase dentro de un package

Al darle finish crea el archivo `HolaMundo.java` dentro del package.

Notas:

- En java, los comentarios son con `//`, (ver figura 2)

- Si en eclipse no sale sugerido al escribir código, se puede activar con ctrl+barra espaciadora

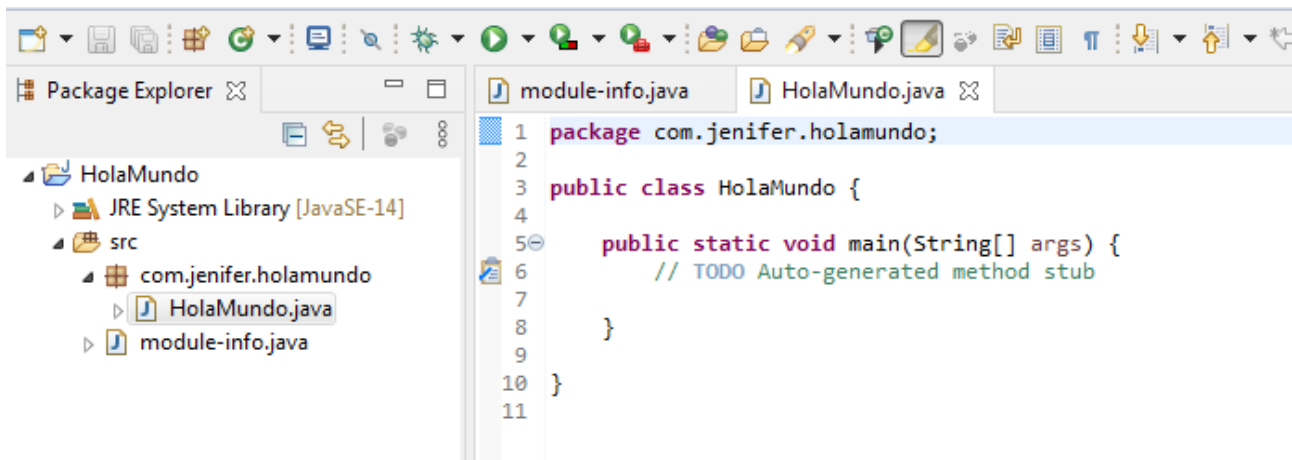


Figura 2: En la figura a la izquierda se ve que creó el archivo .java dentro del package y a la derecha se ve en el código que la clase pertenece al package creado (com.jenifer.holamundo), creó la clase HolaMundo e incluye lo que es de main y aparece una línea comentada que indica que está vacío y queda pendiente lo que hace el main.

Por otra parte está la consola del IDE, que se encuentra abajo, si no está en la parte de abajo, se va a window → show view → console

Una vez escrito, se le da click al botón verde de run, y lo que va a hacer es compilar y correr (Figura 3)

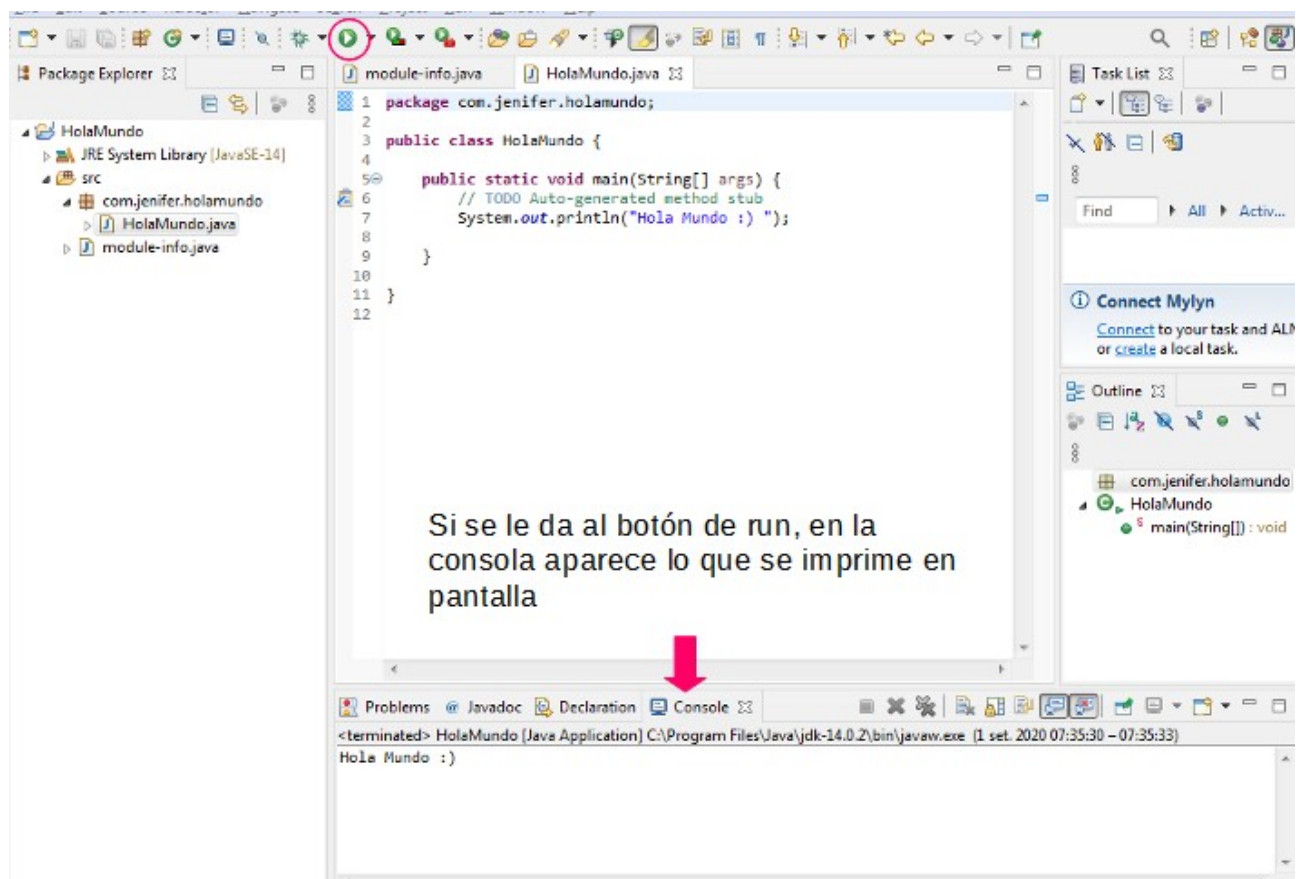


Figura 3: En la figura se muestra a la derecha el navegador de archivos, a la derecha el archivo HolaMundo.java y abajo en la consola el resultado de correr System.out.println. El botón de run está señalado con círculo rojo

Pero si se quiere correr todo el proyecto, se hace file → run as → java application

Tipos de datos

Enteros

En java hay 2 tipos de datos, Primitivos y Objetos, siendo los enteros un tipo de datos primitivo, los cuales se van ordenando de acuerdo a su tamaño y capacidad.

Dentro de los enteros estan:

1. byte → Rango de -128 a 127 = 1 byte. El rango quiere decir que solo puede almacenar desde el número -128 al 127 en 1 byte
2. short → Rango -32768 a 32767 = 2 bytes
3. int → Short y Byte son pequeños, para almacenar números más grandes se utiliza éste tipo de dato. Rango -2.147.483.648 a 2.147.483.647 y ocupa 4 bytes. Es el más usado y es el recomendable.
4. long → Ésto es para almacenar enteros muy grandes y ocupa 8 bytes. Su rango es de -2^{63} a $2^{63}-1$ los cuales son numeros enormes por lo que sólo se recomienda éste tipo de datos

cuando se trabaja con números grandes, por ejemplo, el tiempo en milisegundos que pasaron de una fecha a otra.

En android es muy importante cuidar el espacio en memoria

Notas:

- java es un lenguaje declarativo, así que, cuando quiero crear una variable nueva, se le debe poner el tipo y si hay variables repetidas, lo bueno de trabajar con el IDE es que avisa cuando se comete el error de declarar una variable 2 veces.
- El IDE también avisa cuando no le da el rango a una variables
- Los avisos del IDE en eclipse se ven en el margen de la parte derecha.
- Se recomienda que, al declarar una variable, inicializarla

Float y Double

Se utilizan para guardar números decimales y hay de 2 tipos:

1. Float: Ocupa un espacio en memoria (o sea, una dimensión) de 4 bytes, lo cual es un rango sumamente grande. (igual que int en cuanto a espacio en memoria). Para declararlo se usa como en el siguiente ejemplo:

```
float mivariable = 1.80F;
```

Sin la F no funciona porque si no se pone, el compilador piensa que es un double y da error por falta de consistencia

2. Double: Tiene una dimensión de 8 bytes (igual que long en cuanto a espacio en memoria). Para declararlos por ejemplo

```
double mivariable = 958.96783538;
```

Acá no es necesario poner nada a diferencia de float, que precisa una F.

Char y boolean

1. char: Ocupa 2 bytes en memoria y almacena un solo carácter.

Ejemplo: `char genero = 'F'`

Debe estar declarado con comillas simples, si se ponen comillas dobles, java da error porque las comillas dobles son para strings (sobre esto, mas adelante)

2. boolean: sólo pueden tomar valores de verdadero o falso, y en la sintaxis va todo en minúsculas

Ejemplo: `boolean status = true;`

Se usa para conocer cosas lógicas, de si algo existe o no.

Naming y cast de variables

Naming

Java es muy estricto en cuanto al declarar nombres de variables y de clases

Convenciones para variables

- Debe comenzar con una letra aunque admite el símbolo \$ y el guión bajo, las letras posteriores pueden incluir esos caracteres.
- Puede tener números en el medio, pero no al principio
- Es sensible a mayúsculas y minúsculas
- Por convención se utiliza la técnica “camello” ejemplo: `char miVariable = 'a'`
- Para variables, métodos y objetos se usa Lower Camell Case, las iniciales van en minúscula.
- Las constantes se escriben en MAYUSCULAS y pueden contener guión bajo

Convenciones para las clases

- Se usa la técnica de Upper Cammel Case, la inicial siempre va en mayuscula ejemplo:
`class HolaMundo`
- si es más de una palabra, cada palabra debe empezar con mayúscula

Cast

Cast es la situación en la cual se desea cambiar un tipo de variable a otro.

Ejemplo:

```
double d = 100.79;
```

```
int i = (int) d;
```

El cast se indica entre paréntesis el tipo de dato destino. Por otra parte, se puede usar con cualquier tipo de dato menos los booleanos.

Generalmente ejecutamos un cast, cuando el tipo de dato destino, es más grande que el tipo original. De esta forma, la variable puede caber muy fácilmente en el tipo de dato destino. Por lo tanto podemos observar que muy a menudo podemos castear un tipo de dato byte o char a un int. O un tipo de dato int a un long. Siempre y cuando el tipo de dato destino, sea más grande que el original.

Arreglos

Los arreglos se pueden definir como objetos que pueden almacenar más de una variable siempre y cuando se incluya los tipos de datos almacenados.

Por otra parte pueden tener más de 1 dimensión, en la figura se muestra un ejemplo de arreglo en 2 dimensiones:

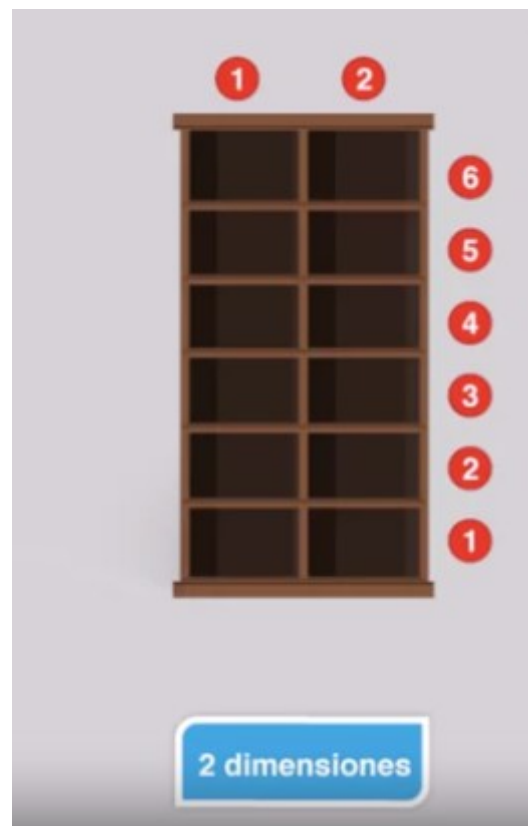


Figura 4: Comparación de un arreglo de 2 dimensiones en java con una estantería

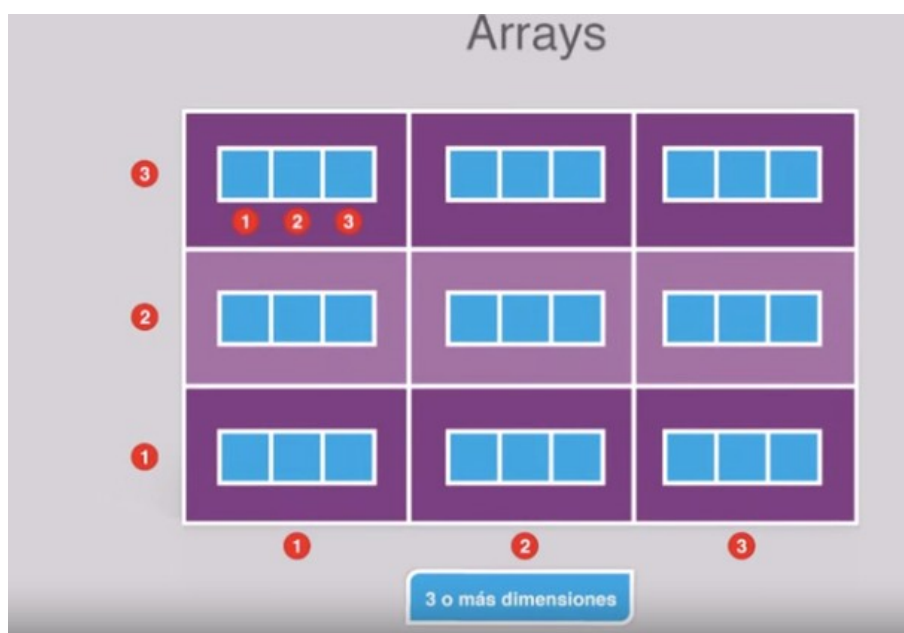


Figura 5: Arreglos de 3 o más dimensiones representados en memoria

Para hacer cosas con arreglos en java, se debe usar de la siguiente manera:

```
nombreVariable = new <tipo de dato> [capacidad]
```

Siendo la capacidad, los slots que quiero reservar, para lo cual se asignan índices, los cuales, comienzan en 0.

Luego para asignar valores:

```
nombreVariable[índice] = valor; //para una dimensión
```

```
nombreVariable[índice] [índice]= valor; //para 2 dimensiones
```

Y así sucesivamente.

Notas:

- Para declarar los arreglos es necesario colocar corchetes antes o después del nombre del objeto, y para dimensionarlo es necesario instanciarlo.
- Si bien los índices comienzan en 0, al declarar la dimensión es n: Ejemplo

```
int[] miArreglo = int [5]; //el array tiene 5 elementos y es de una dimensión todos los cuales van a ser enteros
```

- Si se tiene más de una dimensión, es obligatorio solamente definir la primera, las otras pueden quedar pendientes. Ejemplo:

```
double[][] miArreglo = new double[3][];
```

Por otra parte, si por ejemplo asigno `miArreglo [1][1]= 4;` ese 4 se va a comportar como 4.0 porque el tipo de dato es double.

- Hay que tener cuidado con los índices porque si por alguna razón, en el código, aon variables que adquieren un rango que no está, da error. Ejemplo:

```
double[][] miArreglo = new double[3][3];
```

```
miArreglo[3][1] = 4.5; //esto da error porque el índice va de 0 a 2
```

Operadores

Operadores aritméticos y de asignación

Ahora ya sabemos crear nuestras variables. Podemos utilizar esas variables para formar nuevas expresiones que nos devuelven un valor. ¿Y cómo lo hacemos?, pues utilizando los operadores.

Operadores aritméticos

Éstos involucran el operador suma, el operador resta, el operador multiplicación, división y también el módulo. El módulo significa el residuo de una división.

El operador suma, además de poder utilizarlo para realizar sumas, también nos sirve en java para concatenar cadenas o caracteres, también puede concatenar números.

Por ejemplo

```
int a =2;
int b = 3;
int c = 0;
c = a + b; //acá el operador suma sumó a "a" y "b"
System.out.println("Suma: "+c); //el operador suma concatenó el
string con un int
```

En el caso de la división, cuando los tipos de variables son de tipo int, se va a quedar con la parte entera del número, o sea, si se sigue el ejemplo anterior pero con $c = a/b$, el resultado va a ser 0

Operador	Nombre	Ejemplo
+	Adición	$a + b$
-	Substracción	$a - b$
*	Multiplicación	$a * b$
/	División	a / b
%	Módulo	$a \% b$

Figura 6: Operadores aritméticos

Operadores de asignación

Los siguientes que tenemos son los operadores de asignación. Y éstos se basan mucho en los anteriores, solamente que añadimos un "=", podemos tener "+=", "-=", "*=", "/=", y también "%=". Ésto significa, o se puede desglosar de la siguiente forma. Por ejemplo, si tuviéramos una variable a y una variable b y tuviéramos la operación "+=", "a+=b" podríamos desglosarlo de la siguiente forma. "a= a+b", es decir, estamos acarreado el valor de "a", se lo sumamos a "b" y todo el resultado se lo asignamos nuevamente a, "a". De esta forma es como podemos utilizar los operadores de asignación, anteponiendo una operación.

Las asignaciones siempre van de derecha hacia izquierda.

Operador	Utilización	Desglose
<code>+=</code>	<code>a += b</code>	<code>a = a + b</code>
<code>-=</code>	<code>a -= b</code>	<code>a = a - b</code>
<code>*=</code>	<code>a *= b</code>	<code>a = a * b</code>
<code>/=</code>	<code>a /= b</code>	<code>a = a / b</code>
<code>%=</code>	<code>a %= b</code>	<code>a = a % b</code>

Figura 7: Operadores de asignación

Operadores de incremento y decremento

Éstos operadores, ++ y -- incrementan o decrementan respectivamente, una variable en un int de 1, o sea a++ es lo mismo que a = a + 1.

Por otra parte de estos tenemos dos tipos:

- prefijo: ++a primero incrementa la variable a y luego la asigna.
- postfijo: a++ primero asigna la variable a y luego la incrementa.

La diferencia radica en que si por ejemplo tengo

```
int i =3;
```

```
System.out.println("Suma Postfijo: "+ (i++)); //esto imprime 3
```

```
System.out.println("Valor de i : "+ i); //ésto imprime 4 porque la suma se hizo despues
```

Entonces para poder utilizar estos operadores, tienes que saber pues qué tipo de datos quieres saber en ese momento. En el caso de i++ sabrás será el valor de i que traes y en el caso de ++i lo primero que sabrás será el valor de i incrementado.

Operadores de Equidad y Relacionales

Todas las expresiones creadas con equidad y operadores relacionales regresa un valor booleano (true o false), dependiendo de si la comparación cumple o no cumple los requisitos.

Operadores de equidad

Lo que hacen es comparar igualdad o de desigualdad (Ver figura 8)

Operador	Nombre	Ejemplo
==	Comparación igualdad	a == b
!=	Comparación desigualdad	a != b

Figura 8: Operadores de Equidad

El == significa que estaremos comparando si una variable es igual a otra. Y el otro caso, !=, estaremos comparando si una variable es diferente a otra. Eso devuelve true o false, por ejemplo en el caso de que una variable “a” sea igual a la variable “b” la comparación a==b da true y a!=b va a dar false, ¿cahai?

Operadores Relacionales

Los operadores relacionales son < , >, >=, <= y funcionan como se espera que funcionen. Por ejemplo:

```
int a=5;
int b=5;
int c = 4;
System.out.println("a <=b? "+(a <= b)); //da true
System.out.println("a <=c? "+(a <= c)); //da false
```

Operadores Lógicos

Todos los operadores lógicos regresan un valor booleano, y ellos son and (&&), or (||) y not (!). Ver sintaxis en la Figura 9. Siguen las llamadas tablas de verdad.

Operador	Nombre	Ejemplo
&&	AND	a && b
	OR	a b
!	NOT	!a

Figura 9: Operadores en java

En el caso de and, para que el resultado sea verdadero, ambos tienen que ser verdaderos. Tanto el valor de a y el valor de b, tienen que ser verdaderos para que el resultado sea verdadero. Cualquier otra combinación va a ser falso. Si a vale falso y si b vale falso, el resultado será falso. Ver figura 10

a	b	a && b
f	f	f
f	v	f
v	f	f
v	v	v

AND

a	b	a b
f	f	f
f	v	v
v	f	v
v	v	v

OR

Figura 10: Tablas de verdad para and y or

Por otra parte, or quiere decir, que si el valor de a o el valor de b, es verdadero, siempre será verdadero. O sea el único caso en el cual será falso es cuando ambos valores valen falso.

En el caso de NOT, lo que hace es devolver el inverso, si la variable a es true !a es false.

Control de Flujo

Introducción

Hasta este momento hemos colocado puras líneas de código que se van ejecutando secuencialmente, todas de arriba hacia abajo. Pero hay veces en que nosotros queremos controlar un poco ese flujo.

Es decir, que ese flujo no vaya solamente desde arriba hasta abajo, sino que a lo mejor se detenga el algún lado, o que se omita alguna línea de código, o que incluso se repita esa línea de código varias veces dependiendo de alguna condición.

Para eso es este tema que estamos viendo que se llama Control de flujo.

La idea es que nosotros podemos controlar el flujo que va teniendo nuestro programa.

If, else, switch

if/else

La primera forma de controlar nuestro flujo es utilizando una sentencia que se llama if/else. Lo que hace un if, compara una condición. If traducido al español significaría decir, si, o si hago algo, o si sucede algo, entonces ejecuta esto.

Si la condición que estamos colocando es verdadera, es decir, que sí se cumple, entonces se ejecutará lo que está dentro de las llaves. Si la condición es falsa, o sea que no se cumple, entonces ejecutará lo que está dentro del else. Recuerda que el else puede ser opcional. Puede no estar.

Sintaxis:

```
if (<condición>) {
    //codigo
```

```
}else{  
    //codigo  
}
```

Para anidar ifs la sintaxis es:

```
if (<condición>) {  
    //codigo  
}else if(condicion){  
    //codigo  
}
```

Ejemplo

```
int a=5;  
int b=5;  
if (a > b){  
    System.out.println("a es MAYOR que b");  
  
}else if(a<b) {  
    System.out.println("a es MENOR que b");  
} else if (a == b){  
    System.out.println("a y b SON IGUALES");  
}
```

Esto va a imprimir "a y b SON IGUALES"

Ejemplo 2

```
int a=5;  
int b=20;  
if (a > b){  
    System.out.println("a es MAYOR que b");  
  
} else if (a == b){  
    System.out.println("a y b SON IGUALES");  
} else {
```

```
        System.out.println("a no es mayor a b y tampoco son  
iguales");  
    }
```

Esto va a imprimir "a no es mayor a b y tampoco son iguales" o sea, el else solo, hace que se ejecute el código si ninguno de los ifs se cumple.

Switch

Lo que hace el switch como tal no compara una condición, sino que compara un valor. El valor se pone a inspeccionar dentro de los paréntesis del switch. Entonces si ese valor coincide con algún caso de los que están ahí, se ejecutará lo que esté dentro de ese caso. Básicamente, en un switch inspeccionamos si esa variable cumple con alguno de los casos que tenemos en cuestión. Si no tienes algún caso ahí en el cual la variable cumpla, entonces se ejecutará lo que está en esta líneas de default.

Sintaxis:

```
switch (a) {  
    case valor1:  
        break;  
    case valor 2:  
        break;  
    .  
    .  
    .  
    deafult:  
        break;  
}
```

El switch es un poco similar también a un If. Aquí también estaremos comparando una variable. Pero más que comparar, estaremos inspeccionando el valor de la variable.

Ejemplo

```
int c = 1009;  
switch (c) {  
    case 200:  
        System.out.println("c es IGUAL a 200");
```

```
        break;
    case 1009:
        System.out.println("c es IGUAL a 1009");
        break;
    default:
        System.out.println("c es NO ES IGUAL a 200");
        break;
}
```

Esto imprime "c es IGUAL a 1009"

While, for y foreach

while

while se puede traducir como mientras que. O sea, mientras se cumpla una condición dada, va a ejecutar repetidamente el código que esté dentro del while.

Sintaxis:

```
while (<condición>){
    //codigo
}
```

Ejemplo:

```
int a = 1;
while (a <= 5) {
    System.out.println("Hola, soy el Número "+a);
    a++; //si no se pone ésto, entra en un bucle infinito
}
```

Va a imprimir:

```
Hola, soy el Número 1
Hola, soy el Número 2
Hola, soy el Número 3
Hola, soy el Número 4
Hola, soy el Número 5
```

Otro ejemplo:

```
int x = 1;
while (x <= 10){
    System.out.println(++x);
}
```

Imprimirá del 2 al 11 porque usa prefijo e imprime el valor ya incrementado

for

Si queremos que las líneas de código igualmente se sigan repitiendo. Pero ahora queremos saber cuántas veces se va repitiendo el ciclo, de una forma muy sencilla. Podemos utilizar este controlador de flujo que se llama for.

Sintaxis:

```
for (int i= un numero;i< otro numero, ++i){
    //codigo
}
```

El ++i incrementa en 1 a i, y “*un numero*” usualmente es 0.

También es válido haciendo con - - , ejemplo:

```
for (int i = 10; i >= 0; --i ) {
    System.out.println(i + " elefantes se columpiaban");
}
```

Imprime

```
10 elefantes se columpiaban
9 elefantes se columpiaban
8 elefantes se columpiaban
7 elefantes se columpiaban
6 elefantes se columpiaban
5 elefantes se columpiaban
4 elefantes se columpiaban
3 elefantes se columpiaban
2 elefantes se columpiaban
1 elefantes se columpiaban
0 elefantes se columpiaban
```

O sea, lo que va como argumentos para el for es, el valor inicial de la variable (que es un número), luego va la condición que se debe cumplir y por último, como se incrementa o decrementa la variable

Un uso muy común es usarlo para recorrer arreglos usando el índice:

```
int[] miArreglo = new int[5]; //no olvidarse del new
for (int j=0; j < miArreglo.length; j++) {
    miArreglo[j]= j*6;
    System.out.println("Hola, soy el indice "+j+ " del arreglo y
mi valor es "+ miArreglo[j]);
}
```

Esto imprime

```
Hola, soy el indice 0 del arreglo y mi valor es 0
Hola, soy el indice 1 del arreglo y mi valor es 6
Hola, soy el indice 2 del arreglo y mi valor es 12
Hola, soy el indice 3 del arreglo y mi valor es 18
Hola, soy el indice 4 del arreglo y mi valor es 24
```

foreach

Un Foreach significa por cada. Es decir, lo que recibirá como parámetro un Foreach en vez de un contador, o en vez de una variable que lleve todo el flujo, recibirá un arreglo o una lista o una colección de cosas. El foreach lo que hará será decir por cada elemento que tenga nuestra lista, dará una vuelta en el ciclo. O ejecutará esa línea. Por ejemplo si pasas una lista de cinco cosas, automáticamente el foreach ya sabe que tiene que dar cinco vueltas y ejecutará esa línea de código 5 veces.

Sintaxis:

```
for (<tipo de dato> elemento : colección){
    //codigo
}
```

O sea va a tomar a cada elemento de una colección de datos (generalmente arreglos pero pueden ser otras cosas) y va a ejecutar el código para cada elemento.

Si usamos el ejemplo del arreglo generado en el for:

```
for (int i : miArreglo) {
    System.out.println("Hola, soy el valor " + i+" de mi
arreglo");
}
```

Esto imprime:

```
Hola, soy el valor 0 de mi arreglo
```

Hola, soy el valor 6 de mi arreglo

Hola, soy el valor 12 de mi arreglo

Hola, soy el valor 18 de mi arreglo

Hola, soy el valor 24 de mi arreglo

Nota:

- Para comentar más de una línea se abre el bloque de código con `/*` y se cierra con `*/`

Módulo 3: Programación orientada a objetos

Objetivo

Análisis de un problema de la vida real a partir de la programación orientada a objetos para transformarlo en un código de Java

Introducción

Hemos visto ya todo lo básico sobre Java, hemos visto variables, arreglos, etcétera.

Vamos ahora a un nuevo tema que es la programación orientada a objetos, la programación orientada a objetos la mayoría de las veces nos puede sonar como algo difícil de entender o algo difícil de poder aplicar en nuestros programas, pero la verdad es que hay que partir sobre algo muy sencillo, algo muy simple, lo primero que tenemos que hacer es cambiar nuestra forma de pensar.

¿Cómo voy a cambiar mi forma de pensar? Bueno, es decir, ya no tenemos que ver el mundo o las cosas como comúnmente las vemos, es decir, ahora tenemos que aprender a ver las cosas como objetos.

Precisamente para eso es la programación orientada a objetos, cuando tengas un problema que necesites resolver con software, debes comenzar a definir tus objetos o a identificarlos, es decir, no resolver el problema sino primero visualizar, observar, identificar todos nuestros objetos.

¿Y cómo puedo identificar objetos? ¿Qué cosas podrían decirme que eso es un objeto o que no es un objeto? Los objetos pueden ser algo físico o algo conceptual. Podemos entender algo físico como algo que podemos tocar, algo que sí es tangible y algo conceptual es algo que no es tangible, algo que solamente puede existir en nuestra mente o conceptualmente.

Un objeto físico podría ser por ejemplo una persona, podría ser un cliente, podría ser un producto, podría ser todas las cosas físicas que sí existen. Pero un objeto conceptual podría ser por ejemplo la cuenta de un cliente, la cuenta de un cliente no es algo que puedas tocar como tal, sino es algo que existe solamente por su concepto, algo conceptual.

Los nombres de los objetos comúnmente suelen ser sustantivos; y generalmente los atributos también suelen ser sustantivos como el peso, como el precio de algo, como el color, etc. Los comportamientos o las funcionalidades de los objetos suelen ser verbos o también verbos combinados con sustantivos, por ejemplo enviar pedido, o también el verbo mostrar o el verbo imprimir, el verbo comprar, etc. Todos esos verbos serán nuestras funcionalidades y será lo que defina a un objeto como un **método**.

Ejemplo el objeto teléfono.

El objeto teléfono si se lo ve como un producto físico tiene los atributos:

- id
- marca
- modelo

Entre muchos otros comportamientos, podemos destacar 2 por simplicidad:

- llamar
- colgar

Ahora, si lo vemos como un producto en una tienda para vendelo a clientes tiene los atributos:

- id
- marca
- modelo
- precio

Y el único comportamiento del producto en la tienda es:

- mostrarDatos

El comportamiento son los métodos.

Clases

Hemos analizado un objeto teléfono. El siguiente paso en la programación orientada a objetos es partir de ese objeto, para definir una plantilla. Esa plantilla que nos permite generar más objetos se llama clase.

Una clase es la forma en cómo defines un objeto para generar más objetos. Las clases únicamente son descriptivas, las clases únicamente nos ayudan a definir una plantilla que nos permita posteriormente tener más objetos o mas diversidad de cosas.

En Java las clases se definen con las siguientes palabras,

```
public class <nombre de la clase>{  
  
}
```

y siempre van a ir seguidas de unas llavecitas, que dentro de esas llaves debemos colocar los atributos y los comportamientos tal cual como si estuvieras escribiendo una receta, una receta para cocinar cualquier cosa. En una receta lo primero que colocas son los ingredientes y después colocas toda la forma en cómo vas a desarrollar esa receta. Igualmente en una clase primero pondrás tus atributos o tus ingredientes,

Partiendo de nuestro objeto teléfono podemos definir una clase como podemos observar en el siguiente código:

```
public class Telefono{  
    //atributos  
    int id;  
    String marca;  
    String modelo;  
    double precio;  
  
    //comportamientos o métodos  
    public void mostrarDatos() {  
  
    }  
}
```

Siempre van primero los atributos y luego los comportamientos.

Declaración de métodos

Sintaxis, va en éste orden:

1. Su modificador de acceso.
2. valor de regreso (en realidad es el tipo de dato que devuelve): un método siempre o la mayoría de las veces va a recibir algo y nos va a devolver algo, la mayoría de las veces, no es obligatorio. Entonces nosotros indicamos qué valor o qué tipo de dato es lo que va a regresar ese método.
3. nombre del método.
4. Lo siguiente es colocar el valor de entrada o los datos de entrada para ese método

Ejemplo en la figura 11

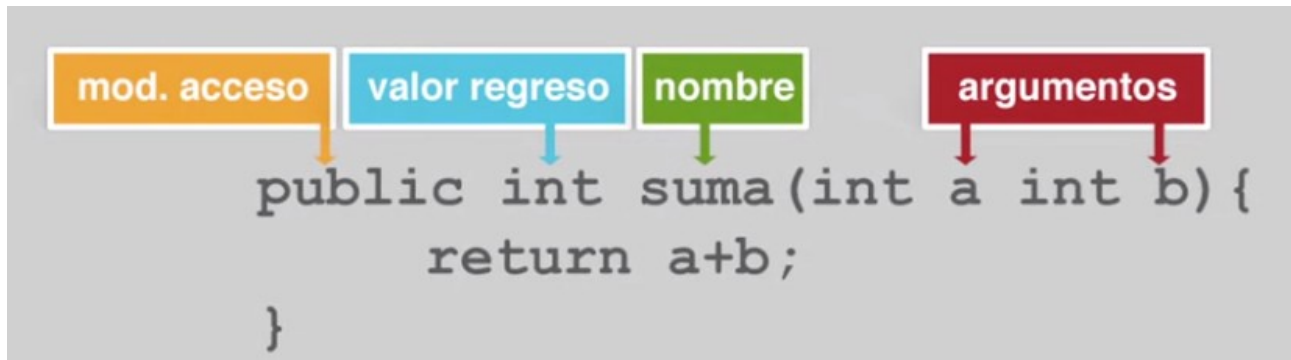


Figura 11: Sintaxis que devuelve la suma de a y b

En el ejemplo de la figura 11, tenemos 2 datos de entrada, uno que es un int a, es un tipo de dato, es una variable a, con tipo de dato primitivo, después viene una variable b, con tipo de dato primitivo, enteros ambos.

Entonces lo que hace ese método pues es simplemente una suma, sumar a más b y si ambos son tipos enteros y se hace una suma de enteros el resultado será pues también un entero. Entonces esa es la forma en cómo declararemos nuestros métodos, modificador de acceso, valor de regreso, nombre del método, parámetros.

Si el método nunca va a devolver algo, si no devuelve nada, colocaremos la palabra reservada **void**.

Volviendo al ejemplo de teléfono, como se ve en la figura 12, al no hacer return, se usa void

```
public void mostrarDatos() {  
  
    System.out.println("Datos Telefono");  
    System.out.println("ID:" + id);  
    System.out.println("Marca:" + marca);  
    System.out.println("Modelo:" + modelo);  
    System.out.println("Precio:" + precio)  
}
```

Figura 12: Ejemplo de uso de void siguiendo el ejemplo del teléfono

Importante

Se refiere a que no se puede declarar una función dentro de la otra, es decir:

No puedes tener:

```
public void mostrarDatos(){  
    public int calcularDatos(int a, int b){  
        return a * b;  
    }  
}
```

```
}
```

Lo que si puedes tener es:

```
public void mostrarDatos(){  
    calcularDatos();  
}
```

pero tampoco es una practica recomendada instanciar funciones dentro de otras funciones.

Declaración e instanciado de objetos

Creación de objetos

Para poder llamar esta clase, lo que yo necesitaré será crear un objeto. ¿Recuerdas nuestro método main, recuerdas nuestro método de entrada? Recuerdas que dijimos que todo lo que queremos que nuestro programa viva o todo lo que queremos que nuestro programa se llame, tiene que estar declarado en el método main.

¿Cómo declaramos un objeto de la clase teléfono? Lo primero es que tienes que ir ahí a ese método main y colocar lo siguiente:



Primero va el tipo de objeto y luego, el nombre del objeto. La inicial en minúscula y la siguiente con mayúscula. De esta forma es como declaramos un objeto

Pero todavía no le hemos dado vida, es decir es como si ahorita solamente estuviera presente, pero está inerte todavía, todavía no lo podemos utilizar. Para poder utilizarlo necesitamos instanciarlo, o ponerlo a volar en la memoria. Para hacer esto necesitaremos utilizar el nombre del objeto espacio igual y con la palabra reservada **new**. Al momento que yo utilizo esta palabra reservada new, quiere decir que estoy dándole vida a mi objeto, estoy poniéndolo en la memoria.

Para poderlo utilizar es que se necesita **instanciarlo**:

```
Teléfono miTelefono;  
miTeléfono = new Telefono();
```

Cuando creo un objeto, se dice que lo estoy instanciando. Ese es el término instanciación. Es decir, poner un objeto en nuestra memoria. Hemos declarado e instanciado nuestro objeto teléfono, siendo Teléfono el método constructor.

La instanciación es colocar un objeto en memoria.

Se puede hacer ésto en una linea sola:

```
Telefono miTelefono = new Telefono();
```

A la izquierda del signo igual se declara el objeto y a la derecha se lo está instanciando

Métodos constructores

Los métodos constructores son los que permiten la instanciación de objetos, es decir, asignar valores en memoria.

Si no se declara Java proporciona uno por default.

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

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

Características de los Constructores:

1. Un constructor, tiene el mismo nombre de la clase a la cual pertenece.
2. No puede ser Heredado.
3. No retorna ningún valor (Ni void), por lo cual no debe especificarse ningún tipo de dato.
4. Debe declararse como public, sólo en casos realmente extraordinarios será de otro tipo

Utilizando objetos y accedendo a métodos

Una vez que hemos declarado nuestro método constructor, y ya hemos puesto en memoria ese objeto, bueno, ahora ya podemos utilizar sus atributos y sus métodos, ya podemos tener acceso a él.

Ejemplo:

En el archivo donde se define las clases y los métodos:

```
public class Telefono {  
    String id;  
    String marca;  
    String modelo;  
    double precio;  
    public void mostrarDatos() {  
        System.out.println("id: " + id);  
        System.out.println("marca: " + marca);  
        System.out.println("modelo: " + modelo);  
    }  
}
```

```
        System.out.println("precio: " + precio);  
    }  
}
```

y en el archivo principal:

```
Telefono miTelefono = new Telefono();  
miTelefono.id="123456";  
miTelefono.marca="Motorola";  
miTelefono.modelo="V3";  
miTelefono.precio=500;  
miTelefono.mostrarDatos();
```

miTelefono es el objeto y con el punto se accede a los métodos y a sus atributos.

Importante: El archivo donde se definen las clases deben estar accesibles al archivo main.

Clase Math:

Ésta es una clase estática que tiene métodos estáticos y los accesa desde el nombre de la clase, seguido por punto, seguido por sus métodos:

```
Math.randon();  
Math.sqrt(25);  
Math.PI;
```

Métodos y miembros estáticos

En algunas situaciones, es necesario utilizar a algunos miembros de una clase que estén disponibles para todo el programa. Para esto son los miembros estáticos. Si hablamos de un método estático o static, podemos definir las siguientes características:

1. está definido para toda la clase
2. se define con la palabra reservada **static**
3. puede ser accesado a partir del nombre de la clase, punto, nombre del método. Es decir, no necesitas tener un objeto para poder acceder a este tipo de métodos.

Ejemplo en la figura 13. Este método está definido como estático, es decir que, desde cualquier lugar lo puedes acceder a partir del nombre de la clase, punto, nombre del método.

```
public class Calculadora{  
  
    public static int suma(int a, int b){  
        return a+b;  
    }  
  
}
```

Figura 13: Ejemplo de método estático, es una calculadora que recibe 2 parámetros int y retorna la suma

Lo mismo funciona para atributos (Figura 14).

```
public class Calculadora{  
  
    public static final double PI = 3.141592653589793;  
  
    public static int valor = 0;  
  
}
```

Figura 14: Aquí declara atributos que indican que son estáticos, la palabra final es reservada que indica que ese valor no puede cambiar

Los atributos y métodos, al ser públicos, pueden ser accedidos y modificados por otras clases

import

Hay una forma, que podemos definir en nuestro programa, para acceder todavía de una forma mucho más sencilla. Y esto es, importando toda la clase estática. Ver figura 15

```
import static com.anahisalgado.operaciones.Calculadora.*  
import static java.lang.Math.*;  
  
public class Principal{  
  
    public static void main(String[] args){  
        System.out.println(suma(3, 5));  
        System.out.println(PI);  
    }  
  
}
```

Figura 15: Import de clases. Como importa todo con el comodín "", no hace falta poner el nombre de la clase a la que pertenece, con sólo llamarlo, ya lo sabe.*

Entonces, con el * en el ejemplo, como importa los atributos y los métodos, importa la función suma de Calculadora y el número PI de Math.

Sobrecarga de métodos y constructores

A veces necesitamos tener dos o más métodos con el mismo nombre pero tengan diferentes argumentos. A éste concepto se le llama sobrecarga de métodos o overload.

Por ejemplo, en el caso de la figura 16, donde se ve una calculadora, hay dos métodos suma, solo que uno recibe 2 enteros mientras que el otro recibe 1 entero y un float.

```
public class Calculadora{
```

```
    public static void main(String[] args){  
        System.out.println(suma(3, 5));  
        System.out.println(PI);  
    }
```

```
    public float suma(float a, float b){  
        return a+b;  
    }
```

```
    public float suma(int a, float b){  
        return a+b;  
    }
```

```
}
```

Figura 16: Sobrecarga de métodos. Hay 2 métodos suma, uno que opera con enteros solamente y retorna un entero y el otro opera con otro entero y un float (no sé que retorna éste, pero supongo que retorna un float)

Entonces, en éste caso se dice que suma está sobrecargado porque conserva su nombre pero tiene diferentes argumentos y además de eso, el valor de retorno también es diferente. Entonces, cuando tenemos un método que tiene el mismo nombre repetidas veces, podemos decir que estamos sobrecargando ese método.

Sobrecarga de constructores

Además de sobrecargar métodos que están definidos dentro de nuestra clase, ¿te acuerdas de nuestro método constructor? Bueno, también este podemos sobrecargarlo, como puedes observar en el siguiente código:

```
Telefono miTelefono= new Telefono();
```

la sobrecarga de constructores se utiliza para inicializar objetos, ese es su uso adecuado.

Veamos nuestra clase teléfono, tenemos declarados primeramente nuestros atributos, y luego tenemos nuestro método constructor. Si no tenemos uno declarado, el compilador nos provee de uno, pero nosotros podríamos también declararlo o indicarlo. En este caso está indicado, está indicado el método constructor para la clase teléfono (Figura 17).


```
public class Telefono {  
    int id;  
    String marca;  
    String modelo;  
    double precio;  
  
    public Telefono() {  
  
    }  
  
    public Telefono(int id, String marca, String precio) {  
        this.id = id;  
        this.marca = marca;  
        this.precio = precio;  
    }  
}
```

Figura 17: Observación del constructor Teléfono. Primero se declara los atributos y luego el método constructor. El “this” funciona casi como el “self” en otros lenguajes (creo)

Primero se declara los atributos y luego el método constructor.

En este caso está indicado, está indicado el método constructor para la clase teléfono. Y si observas más abajo, tenemos nuevamente el método constructor teléfono pero ahora este está recibiendo parámetros que son distintos para el antiguo método constructor. El antiguo no tiene ninguno y este tiene el id, la marca y el precio.

En este caso tener un método constructor sobrecargado de esta forma, nos va a ayudar para poder inicializar nuestro objeto cuando lo creamos o cuando lo instanciamos. Es decir, nos puede ayudar para definir que cuando creamos un objeto, como mínimo para que el objeto exista, necesita tener a fuerzas un id, una marca y un precio.

Si el objeto no cumple con esto, no podremos crearlo. Si el objeto no tiene definido un id, una marca y un precio no podemos crearlo, por eso es importante sobrecargar el método constructor para poder definir nuestro objeto con un mínimo de requerimientos.

Si no se hace nada de esto, es decir:

Si en la clase Telefono tenemos esto:

```
public class Telefono {  
    String id;  
    String marca;  
    String modelo;  
    double precio;  
  
    public void mostrarDatos() {
```

```
        System.out.println("id: " + id);  
        System.out.println("marca: " + marca);  
        System.out.println("modelo: " + modelo);  
        System.out.println("precio: " + precio);  
    }  
}
```

Y en el main tenemos esto:

```
Telefono miTelefono = new Telefono();  
miTelefono.mostrarDatos();
```

Va a imprimir null.

Ejemplo de sobrecarga

En la figura 17, tenemos 2 métodos constructores con el mismo nombre solo que uno no recibe parámetros y el otro sí. Ésto hace que, en el main si llamo al método constructor sin parámetros, va a poner parámetros por default mientras que, si lo llamo con parámetros, va a asignarle los valores que se le pasan como parámetros.

```
int id;  
String marca;  
String modelo;  
double precio;  
  
public Telefono(){  
    this.id = 0;  
    this.marca = "Motorola";  
    this.modelo = "Moto E";  
    this.precio = 1.0;  
}  
  
public Telefono(int id, String marca, double precio){  
    this.id = id;  
    this.marca = marca;  
    this.precio = precio;  
}
```

Figura 18: Ejemplo de Sobrecarga. Tenemos 2 métodos constructores con el mismo nombre solo que uno no recibe parámetros y el otro sí. Ésto hace que, en el main si llamo al método constructor sin parámetros, va a poner parámetros por default mientras que, si lo llamo con parámetros, va a asignarle los valores que se le pasan como parámetros.

Nota: hay que tener cuidado con el tema de tipos de datos, en el ejemplo, precio es de tipo double, si se le pone otro tipo, el programa va a dar cosas inesperadas.

Entonces en el archivo main, se pueden crear objetos con valores por default o setearlos (Ejemplo: Figura 19)

```
public class Principal {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        Telefono miTelefono = new Telefono();  
        miTelefono.mostrarDatos();  
  
        Telefono tuTelefono = new Telefono(1, "Nexus", 2.0);  
        tuTelefono.mostrarDatos();  
    }  
}
```

Figura 19: Aquí se definen 2 objetos que son de tipo teléfono. El objeto miTelefono tiene valores por default mientras que el objeto tuTelefono, el modelo es null porque eso no está definido en el constructor que usé (Ver figura 18)

De esta forma es como puedo estar sobrescribiendo los métodos.

Modificadores de acceso

Los modificadores de acceso son palabras clave que se usan para especificar la accesibilidad declarada de un miembro o un tipo. Lo que permite es controlar la visibilidad de los atributos y los métodos. En java existen 4 tipos:

- **public:** El modificador public lo que hace es que te da acceso público a todo, a todo lo que quieras. Es decir, si defines un método como public, ese método estará disponible en cualquier lugar. you sea en una clase o en una clase que esté en otro paquete, o incluso en una clase que herede de otra, que esa es considerada como una subclase.

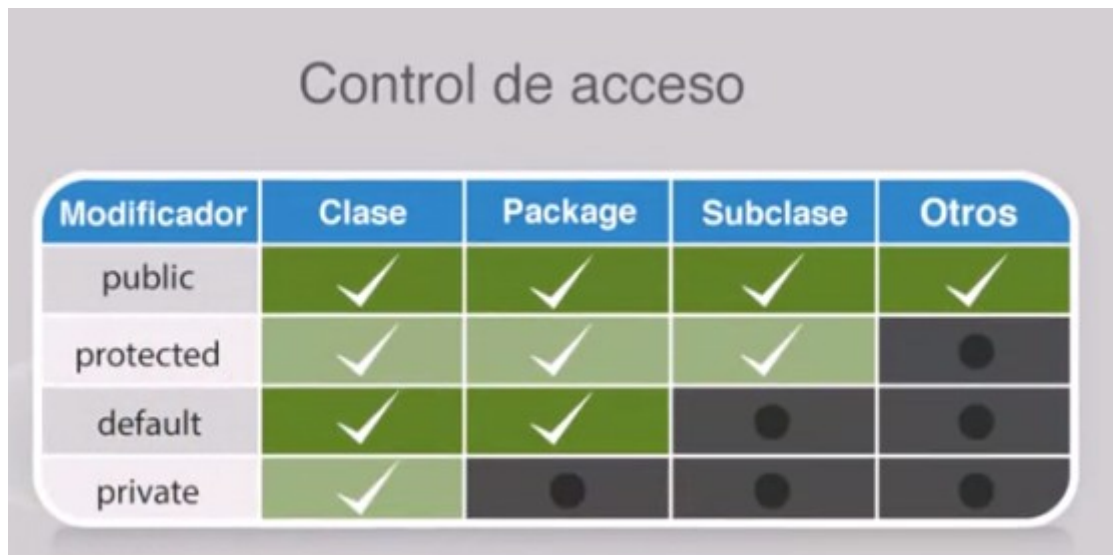
Entonces el modificador public, cualquier cosa que tu defines como public va a ser accesible a cualquier nivel en cualquier clase, no importa donde estés.

No te confundas con un método estático, recuerda que el método estático lo accedas a partir del nombre de la clase. En este caso si defines un método public puede ser o no estático. Es decir puede ser accesado a partir de un objeto, o puede ser accesado también, a partir del nombre de la clase.

- **protected:** Protected nos da acceso a elementos que estén dentro de la clase y también elementos que estén en otras clases. Elementos que estén en clases de diferentes paquetes e incluso también en clases que estén heredando de otras.
- **default:** El modificador default solamente es accesible para elementos que estén entre varias clases. O clases que estén en el mismo paquete.

- **private:** Es el más riguroso, es nuestro modificador privado, o private. Este modificador solamente nos permite tener acceso a los elementos que están dentro de nuestra clase. Si queremos acceder a ellos, ni siquiera los podremos ver.

En la figura 20 se muestra la visibilidad que da cada uno



Control de acceso

Modificador	Clase	Package	Subclase	Otros
public	✓	✓	✓	✓
protected	✓	✓	✓	●
default	✓	✓	●	●
private	✓	●	●	●

Ejemplo de modificadores de acceso

Estamos en nuestra clase principal, en el método main, recuerdas que hace un momento generamos un método constructor que tiene estos valores obligatorios. Es decir para que el objeto sea creado necesito definir como mínimo estos datos, estos datos es: el id, la marca y el precio de ese teléfono.

Entonces pues esto es un riesgo cuando nosotros tenemos este tipo de elementos así, es un riesgo para la seguridad pues los elementos pueden ser cambiados accediendo directamente a los atributos.

Entonces, para evitar eso es que se le dice que es privado y la única forma de asignar o modificar atributos, van a ser los que provee la clase.

En el archivo de la clase Telefono:

```
public class Telefono {  
    private int id;  
    private String marca;  
    String modelo;  
    private double precio;  
  
    public Telefono() {  
        this.id = 0;  
        this.marca = "Motorola";  
        this.modelo = "V3";  
    }  
}
```

```
        this.precio = 1.0;
    }

    public Telefono(int id, String marca, double precio) {
        this.id = id;
        this.marca = marca;
        this.precio = precio;
    }

    public void mostrarDatos() {
        System.out.println("id: " + id);
        System.out.println("marca: " + marca);
        System.out.println("modelo: " + modelo);
        System.out.println("precio: " + precio);
    }
}
```

Acá se definieron variables privadas y la única que es pública es Modelo, entonces con el código asó como está, si creo el objeto miTelefono, la asignación `miTelefono.modelo="V6"`, pero `miTelefono.marca="Motorola"`, no porque es inaccesible.

Por otra parte, si uso la función Telefono con parámetros, usa ése constructor por lo que modelo, al no estar definido, me va a dar null.

Herencia y Polimorfismo

Herencia

Recuerda que en la programación orientada a objetos lo que se planea es tratar de modelar cuestiones de la vida real y pasarlos a código, a software. Generalmente la mayoría de los software llegan a ser un poco complicados en construirse. Precisamente la programación orientada a objetos nos ayuda a modelar esas situaciones de la vida real, esos escenarios de nuestra vida real, tratarlas de pasar a código de la forma más sencilla posible.

Precisamente uno de estos temas que nos ayudan a poder pasar cosas de la vida real, a transformarlas a código de una forma súper sencilla es la herencia. Podemos ver la herencia en muchas facetas de nuestra vida por ejemplo, yo podría tener mi abuelo, y también podría tener por ejemplo hijos. Yo podría tener algunas características que tiene mi abuelo, podría heredarlas, el color de ojos, la forma del cabello, el tono de piel, etc. Y así mismo, podría de alguna una forma, digamos reutilizar, esas características, y dárselas también a mis hijos. Algunos podrían tenerlas y otros no. Esos son algunos de los detalles que también veremos en la herencia. Mis hijos, algunos

de mis hijos podrían tener el color de ojos también o podrían tener incluso la misma forma del cabello.

Justamente así también es como trabajamos en la herencia con código. Básicamente tomaremos una clase, tendremos también una clase padre y una clase hijo. Tomaremos la clase padre. Esa clase padre tendrá definidos atributos, comportamientos, etc, etc. Y esa clase padre podrá heredárselas a una clase hijo. La clase hijo podrá decidir si tomar todas las características o no. Y también esa clase hija, podría heredárselas a nuevas clases hijas. De esta forma, es como vamos viendo una jerarquía en el código. La herencia nos ayuda a reutilizar código, a reutilizar características, y también comportamientos.

O sea:

1. Se establece una relación padre-hijo entre 2 objetos diferentes
2. La idea de la herencia es permitir la creación de clases nuevas basadas en clases existentes
3. En algunas circunstancias, es necesario utilizar el estado y comportamiento de una clase en conjunto con otras clases

Ejemplo, se pueden tener las clases Playera, Jeans y Calcetines (Figura 20).

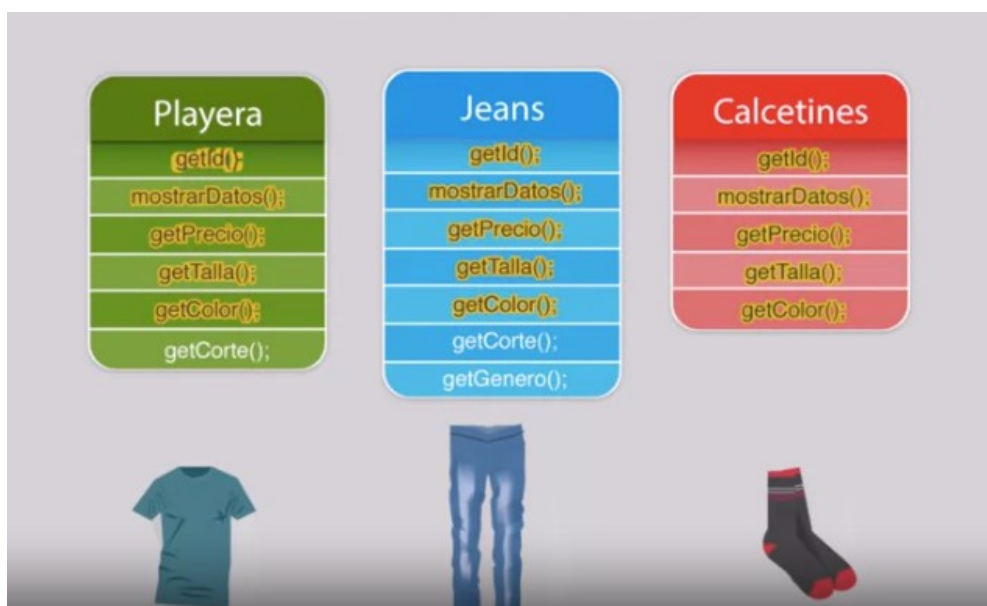


Figura 20: Ejemplo de 3 clases con métodos en común, los cuales se muestran subrayados en amarillo

Como observas estas tres clases tienen algunos atributos en común. Y en contexto de Getters y Setters comparten también estos métodos. Por ejemplo getId, getPrecio, getTalla y getColor. Adicional cada clase tiene algunos atributos propios de ella. Es decir, hacer una familia con estas clases, de tal forma que puedan compartir entre ellas estos atributos y estos métodos.

Tenemos el método getId, el getPrecio, getTalla y getColor son métodos que están presentes y que todas las tienen en común. También el método mostrarDatos. De esta forma, podemos hacer una clase padre que contenga todos estos métodos y atributos. Y así pueda heredarlos a sus clases hijas. Y de esta forma, podemos estar reutilizando código.

Entonces se genera una superclase, en el caso ejemplo Ropa y todas las clases hijas, como por ejemplo la que está aquí que es playera, será nuestra subclase, o clase hija.(Ver Figura 21)

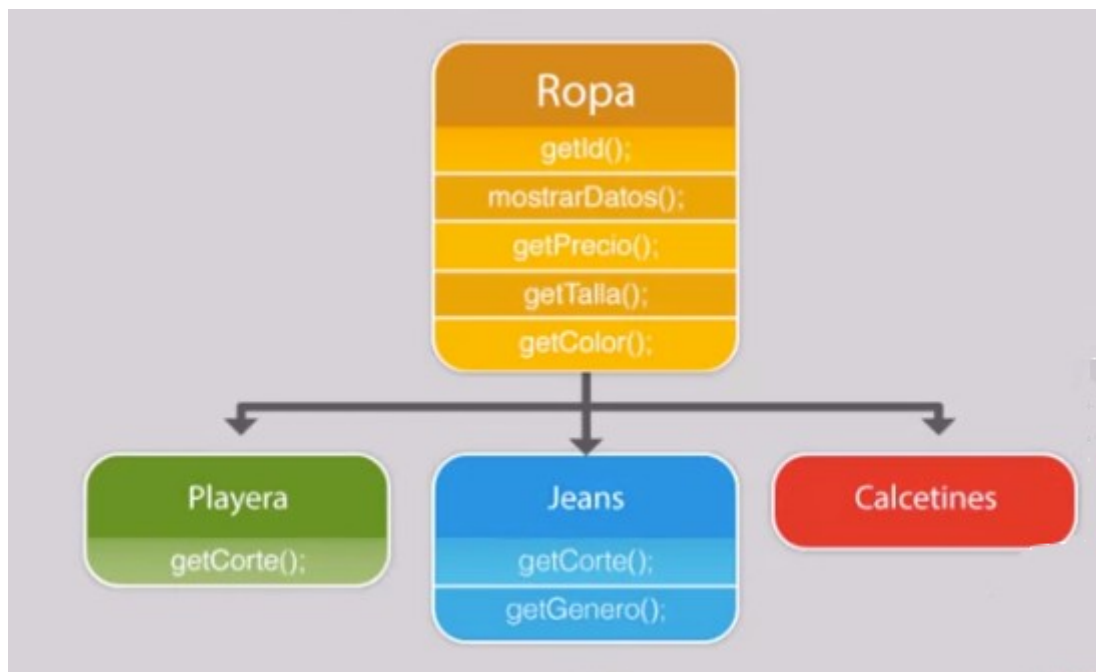


Figura 21: Creación de una superclase Ropa que tiene los métodos en común con las clases hijas

Para hacer ésto en código, lo que se hace es usa la palabra clave `extends`.

Sintaxis:

```
<modificador de acceso> <nombre de clase hija> extends <nombre de clase padre>{  
}
```

Ver figura 22

```
public class Ropa{  
    Súper Clase  
}  
public class Playera extends Ropa{  
    Subclase  
}
```

Figura 22: Sintaxis de superclases y subclases con el ejemplo de Ropa

Cuando utilicemos la herencia en nuestro código, habrá dos palabras muy importantes, que son muy comunes utilizar en la herencia:

- `super`: todos los elementos de mi súper clase, de mi clase padre.
- `this`: todos los elementos de esta clase, de la clase en la que estoy ubicado.

Sobreescritura de métodos (override)

Se da cuando una clase hereda de otra y en la clase hija se redefine un método con una implementación distinta a la de la clase padre.

Solamente heredar los métodos y que estén ahí, no es suficiente; a veces necesitamos dar nuevas implementaciones a esos elementos o darles otro comportamiento. Cuando nosotros implementamos un método y le añadimos comportamiento o le añadimos código, es decir, implementamos el método con nuevo código. A eso se le llama sobreescritura de métodos.

Algo que es muy importante decir es que no puedes sobreescribir todos los métodos de una clase padre. Por ejemplo, los métodos marcados como finales, osea **final**, o como estáticos, osea con la palabra reservada `static`. Estos no los podemos sobre escribir, estos se conservan en la clase padre.

Sobreescritura de constructores

Los métodos constructores se pueden sobreescribir, y básicamente lo que estás aquí implementando es la creación de una clase hijo a partir de una clase padre. Es decir, yo estaré sobrescribiendo el método constructor de mi clase padre, en mi clase hijo. Y de esta forma tendré una nueva clase hijo a partir de la clase padre.

Sintaxis en Figura 23.

```
/** constructor de una subclase */
public Subclase (parámetros...) {
    //invoca el constructor de la superclase
    super (parámetros para la superclase);
    //inicializada sus atributos
    ...
}
```

Figura 23: Sobreescritura de parámetros de una superclase

Polimorfismo

En programación orientada a objetos, polimorfismo es la capacidad que tienen los objetos de una clase en ofrecer respuesta distinta e independiente en función de los parámetros (diferentes implementaciones) utilizados durante su invocación. Dicho de otro modo el objeto como entidad puede contener valores de diferentes tipos durante la ejecución del programa.

O en otras palabras, es la posibilidad de construir varios métodos con el mismo nombre, pero con relación a la clase que puede tener cada uno, con comportamientos diferentes. Es decir, voy a tener

un mismo método con diferentes comportamientos y cuando tengo diferentes comportamientos de una sola cosa a eso se le llama Polimorfismo. Polimorfismo quiere decir muchas formas. Eso es lo que sucede cuando nosotros sobreescribimos un método, el método ahora tiene muchas formas de comportarse.

Ejemplo de Herencia y Polimorfismo

Getters y Setters

Para qué sirven

Los métodos Getters y Setters nos ayudarán a leer y a escribir específicamente el valor de una variable miembro.

Hasta el momento hemos ocultado valores, en el ejemplo hemos ocultado el id, la marca y también el precio, en nuestro ejercicio anterior. Pero ¿qué pasa si en un futuro quisiéramos actualizar el precio o quisiéramos cambiarlo de un dólar o 2, pasarlo a 4 dólares? Para eso tenemos unos métodos especiales.

Estos métodos especiales nos permiten acceder a un atributo, obtener sus datos e incluso actualizar un atributo, cambiarle el valor que tenga. Estos métodos se llaman Getters y Setters, que nos ayudará a obtener el dato de una variable, y tendremos otro método set que nos ayudará a actualizar o cambiar el valor de una variable.

Éstos métodos deben formar parte de la clase donde están definidas las variables a cambiar.

Un método get y un método set, ambos giran en torno a la variable. Entonces podemos decir que por cada atributo que tenga nuestra clase sería bueno definir un método get y un método set para poder estar accediendo a los datos y actualizándolos para no dejar la variable expuesta. Como ves tener este tipo de métodos es una buena práctica para estar trabajando en nuestras variables.

Sintaxis de get:

```
public int get<nombre de variable empezando con mayuscula>() {  
    return <variable>;  
}
```

Sintaxis de set:

```
public void set<nombre de variable empezando con mayuscula>(<tipo  
de dato> <argumento>) {  
    this.<nombre de variable> = <argumento>;  
}
```

Notas:

- Lo de “nombre de variable empezando con mayuscula” es por convención
- Los metodos setter y getters conviene escribirlos debajo de los métodos constructores.
- Getters y setters tienen que ser de acceso público, si no, el main no los ve

El IDE eclipse ofrece crear automáticamente los setters y getters. Para eso hay que posicionar el cursor donde se desea colocarlos, hacer click derecho, luego Source → Generate Getters and Setters, y de allí seleccionar qué getters y setters quiero colocar, en verde aparecen los atributos públicos y en rojo los privados.

Por otra parte, cuando se hace ésto el argumento lo pone con el mismo nombre de las variables que se definen al principio.

Ejemplo:

En el archivo de la clase:

```
public class Telefono {

    private int id;
    private String marca;
    String modelo;
    private double precio;

    public Telefono() {
        this.id = 0;
        this.marca = "Motorola";
        this.modelo = "V3";
        this.precio = 1.0;
    }

    public Telefono(int id, String marca, double precio) {
        this.id = id;
        this.marca = marca;
        this.precio = precio;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
}
```

```
public String getMarca() {  
    return marca;  
}  
  
public void setMarca(String marca) {  
    this.marca = marca;  
}  
  
public String getModelo() {  
    return modelo;  
}  
  
public void setModelo(String modelo) {  
    this.modelo = modelo;  
}  
  
public double getPrecio() {  
    return precio;  
}  
  
public void setPrecio(double precio) {  
    if (precio < 1){  
        System.out.println("El precio no es válido");  
    }else{  
        this.precio = precio;  
    }  
}  
}
```

Y si se quiere setear un precio al objeto miTelefono, en el main:

```
miTelefono.setPrecio(-4.0);  
miTelefono.mostrarDatos();
```

En éste caso, va a setear el precio a 0 porque en el método setPrecio, le dije que, si le pasaba un precio menor a 0, lo seteara en 0.

Por otra parte, si quiero saber el valor de una variable perteneciente a un objeto en determinado momento, uso el método get que haya definido. Por ejemplo:

```
Telefono tuTelefono = new Telefono(1, "Nexus", 2.0);  
System.out.println(tuTelefono.getPrecio()); //ésto imprime que el  
precio es 2  
tuTelefono.setPrecio(-4.0); //ésto va a imprimir "El precio no es  
válido"  
tuTelefono.mostrarDatos(); //como setee un numero negativo,  
mostrar datos me da precio 2.0 que es el que le asigné antes
```

Ahora, puedo cambiar el comportamiento del if:

```
public void setPrecio(double precio) {  
    if (precio < 1){  
        this.precio = 1.0;  
    }else{  
        this.precio = precio;  
    }  
}
```

Eso, lo setea a 1.

Entonces mis métodos getters y setters me ayudan a mantener la integridad de mis datos y poder actualizarlos sin tener problemas en un futuro.

Objetos vs variables

Entonces con lo que acabamos de ver, podemos concluir que las variables son diferentes a los objetos. Como acabamos de observar, vimos todas las características que puede tener un objeto. Un objeto se denomina a partir de una clase, además, un objeto puede contener métodos, puede contener otras variables e incluso le pueden pertenecer también otros objetos. Como vemos, hay una diferencia clara entre las variables y los objetos.

Tenemos un tipo de dato Byte, un tipo de dato Short, Integer, Long, Float, Double, Character, Boolean y String. A simple vista podemos observar que estos comienzan con mayúscula. Esa es una característica de los objetos, el origen de los objetos, que son las clases. Las clases siempre van a comenzar con mayúscula, lo que estamos viendo aquí son clases o son tipos de datos objeto. Es decir, tenemos Byte, Short, Integer. Estos no son primitivos. Lo que hizo Java fue crear un tipo de dato un poco más complejo a un primitivo.

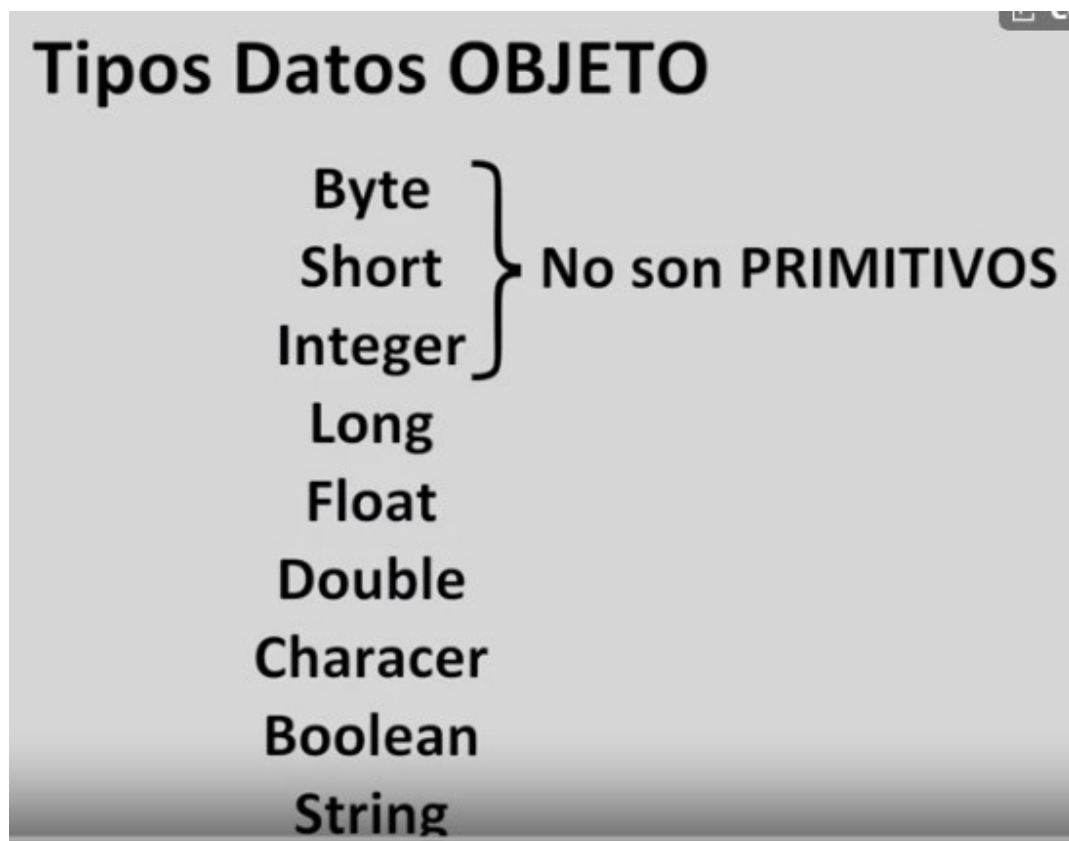


Figura 24: Clases de datos en java. Éstas son clases implementadas

Como observaste, en los primitivos no teníamos un tipo de dato que nos permitiera almacenar textos o que nos permitiera almacenar más allá de un solo caracter. Entonces lo que hizo Java fue darnos una clase especial donde pudiéramos almacenar más de un solo caracter, y esta clase es la clase String. La clase String se compone de muchos métodos que en realidad son muy conocidos hoy en día en Java.

Además de métodos tenemos también variables, tenemos incluso objetos que componen esta misma clase, y a partir de esta clase **String** yo puedo crear más objetos.

Lo mismo sucederá para un objeto de tipo Byte, un objeto de tipo Float, por ejemplo. Estas también son clases que se componen de métodos y de otros atributos que componen al objeto, por ejemplo, que yo pueda darle mayor trato a mis cadenas, que yo pueda cortar en un momento dado una cadena o que yo pueda, incluso, obtener, a partir de un índice, una letra en particular de alguna cadena, o incluso si yo quiero, transformar un tipo de dato a otro tipo de dato. Estas clases tienen todos los métodos y tienen toda la lógica para realizar esto.

Como vemos, las variables son entidades muy sencillas, entidades elementales, es decir, una variable solamente será un espacio en memoria que nos permita almacenar un número, un caracter o un valor verdadero o falso, pero los objetos son entidades más complejas. Estas pueden estar formadas por la agrupación de muchas variables y métodos. Los objetos son entidades más complejas, más complejas que solamente almacenar un solo número o un solo caracter. (Figura 25)

Variables ≠ Objetos

Variables son entidades elementales muy sencillas

Un número

Un carácter

Verdadero o Falso

Objetos son entidades complejas que pueden estar formadas por la agrupación de muchas variables o métodos

Figura 25: Diferencias entre variables y objetos

Como se vé en la Figura 26, vamos a observar cómo se comportan a nivel de memoria una variable versus un objeto. Tenemos, en primer lugar, la variable `i`, `int i = 0`. Como observamos en la memoria, en la zona verde, el 0 solamente está almacenándose en una localidad de memoria. Es una variable sencilla, es un espacio en memoria muy sencillo, pero si tenemos, por ejemplo, el objeto `mi teléfono`, vemos que está instanciado y posteriormente vemos `tuTeléfono`, vemos que también está instanciado, es decir, están puestos en la memoria. Como vemos, lo que se está almacenando aquí, en realidad es una referencia o una dirección de memoria en la cual nos indica en dónde realmente está almacenado todos los datos de ese objeto.

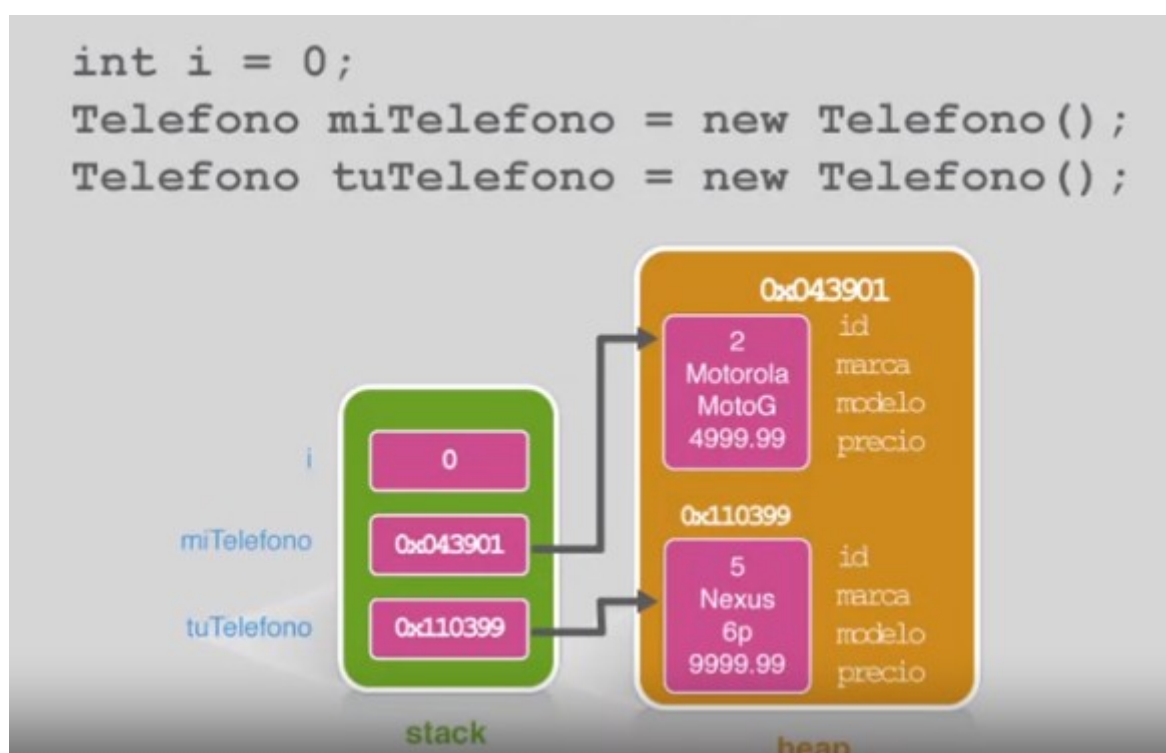


Figura 26: Ejemplo de cómo hace java para almacenar variables y objetos en memoria

Por ejemplo miTeléfono tiene la referencia de memoria 0x043901. Este, en realidad está viviendo en la zona naranja, donde ahí tengo que mi teléfono tiene un id número 2, tiene una marca Motorola, tiene un modelo Moto G y el precio es el que se muestra ahí. Lo mismo para tuTeléfono, pues observamos que solamente está almacenada la referencia de la memoria en donde realmente se ubica el objeto tuTelefono, y como observamos, en la zona naranja, ahí están almacenados todos los datos de tuTelefono.

Así es como realmente se pueden observar las variables versus los objetos en la memoria. Por eso las variables son distintas que los objetos.

Módulo 4: Lenguaje adicional de Java

Objetivo

Utilizar la clase ArrayList e implementar excepciones en el código Java

Bibliografía

Deitel, P. Y Deitel H. (2012). Cómo programar en Java Novena Edición. Pearson