**Academia Java – JAVA**

***Introducción***

Tabla de contenido

INTRODUCCIÓN 3

1. Rompiendo esquemas 4

1.1. ¿Cómo trabaja Java? 4

1.2. ¿Qué harás en Java? 5

1.3. Una breve repaso de la historia de Java 6

1.4. Mira qué fácil es codificar en Java! 7

1.5. Estructura del código en Java 8

1.6. Anatomía de una clase 10

1.7. Codificando una clase con un main 10

1.8. ¿Qué puedes hacer en un método main? (…y en cualquier método) 11

1.9. Bucles…bucle - ando ando… 13

1.9.1. Pruebas booleanas simples 13

1.9.2. Ejemplo de aplicación: Bucle 14

1.10. Condicionales…y si… 15

1.10.1. Ejemplo de aplicación: Bucle + If 15

1.11. Codificando una seria aplicación de negocios 16

1.12. Sé el compilador 18

# INTRODUCCIÓN



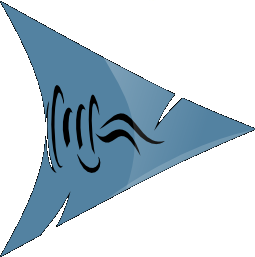
Java te lleva a nuevos lugares.

Desde su humilde lanzamiento al público en su versión 1.02, viene seduciendo a los programadores debido a su fácil sintaxis, características orientadas a objetos, gestión de memoria, y lo mejor de todo: la **promesa de la portabilidad**.

Respecto a eso, su atractivo de codificar/ejecutar en cualquier lugar es **demasiado** fuerte.

En los siguientes capítulos revisaremos todo el camino que seguirás para poder ingresar de lleno al Java World, que comentarios a parte…está más asfaltado y señalado que en tiempos anteriores. La potencia que ofrece el lenguaje es realmente increíble y sobretodo es elegante.





# Rompiendo esquemas

Procederemos a revisar algunos de los temas más importantes de Java de forma introductoria.

# ¿Cómo trabaja Java?

Para explicar cómo es que Java funciona (y también cual es la magia que permite portabilidad) usaremos el siguiente ejemplo.

Implementaremos una aplicación (en este ejemplo, una invitación a una fiesta) y haremos que funcione en cualquier dispositivo para que tus amigos puedan revisarla donde sea.

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| Crear un documento usando como algún protocolo (ejem…en este caso Java) | Corre tu documento usando un compilador de código fuente. Este verificará si existen errores y si es así no permitirá compilar hasta que todo esté correcto. | El compilador crea un nuevo documento, codificado en Java bytecode. Cualquier dispositivo capaz de  ejecutar Java podrá interpretar/traducir este archivo en algo que pueda ser ejecutado. La capacidad de ejecución del bytecode es independiente de plataforma. | Tus amigos no tienen una physical Java machine pero si que tienen una virtual Java (implementado en software) ejecutándose en sus gadgets. Esta virtual machine leerá y ejecutará el biytecode. |

# ¿Qué harás en Java?

Ahora que ya sabes que es lo que debe hacer el ejemplo que hemos planteado. Vas a escribir el documento o archivo de código fuente, lo compilarás usando el compilador javac y ejecutarás el bytecode compilado en la Java virtual machine.

|  |  |
| --- | --- |
|  | Escribe tu código fuente y guárdalo como Party.java. |
| Compilar el archivo Party.java ejecutando javac. Si no tienes errores se generará otro archivo de nombre Party.class. |  |
|  | El compilador generará el archivo Party.class compuesto por bytecodes. |
| Ejecuta el programa iniciando la java Virtual machine (JVM) usando el archivo Party.class. La JVM traduce el bytecode en algo que la plataforma entiende y ejecuta el tu programa. |  |

# Una breve repaso de la historia de Java

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| Java 1.02 | Java 1.1 | Java 2 (1.2 – 1.4) | Java 5 (1.5 a +) |
| 250 clases | 500 clases. | 2300 clases. | 3500 clases |
| Lento. | Un poco más rápido. | Mucho más rápido | Más poder y más sencillo para desarrollar. |
| Nombre y logos lindos. Divertido de usar. Demasiados bugs. Los Applets son la gran novedad.  http://tonibitt.blogia.com/upload/20110319231656-no-bugs.png | Más capacidad, amigable. Aumentando su popularidad. Mejor codificación par GUI. | Puede (a veces) ejecutarse a velocidades nativas. Gran alcance. Tres presentaciones: Micro Edition (J2ME), Standard Edition (J2SE) y Enterprise Edition (J2EE). Se convierte en la nueva elección para desarrollos enterprise y móviles.  http://movilzone.org/files/2008/10/blackberry-pearl-8220-04.jpg | Además, de tener más de un millar de clases adicionales y cambios importantes en el lenguaje mismo, por lo que (al menos en teoría) es más sencillo para los programadores. Absorbió las características más populares de otros idiomas. |

# Mira qué fácil es codificar en Java!

|  |  |
| --- | --- |
| **int** size = 27; | Declarando una variable integer llamada “size” y asignándole el valor 27. |
| String name = "Fido"; | Declarando una variable string llamada “name” y asignándole el valor “Fido”. |
| Dog myDog = **new** Dog(name, size); | Declarando una nueva variable Dog llamada “myDog” y asignándole valores a “name” y “size”. |
| **int** x = size - 5; | Sustrayendo 5 desde 27 (valor de “size”) y asignando el resultado a la variable “x”. |
| **if** (x < 15) myDog.bark(8); | Y así sucesivamente… |
| **while** (x > 3) { |
| myDog.play(); |
| } |
|  |
| **int**[] numList = { 2, 4, 6, 8 }; |
|  |
| System.*out*.print("Hello"); |
| System.*out*.print("Dog: " + name); |
| String nurn = "8"; |
|  |
| **int** z = Integer.*parseInt*(nurn); |
|  |
| **try** { |
| *readTheFile*("myFile.txt"); |
| } **catch** (FileNotFoundException ex) { |
| System.*out*.print("File not found."); |
| } |

# Estructura del código en Java

|  |  |
| --- | --- |
|  | |
| **¿Qué va en el archivo de código fuente?**  El archivo con extensión.java tiene la definición de una clase. La cual representa una pieza de tu programa…aunque si la aplicación es de una sola clase…☺  La clase debe ir dentro de un par de llaves. |  |
| **¿Qué va dentro de una clase?**  Una clase tiene uno o más métodos. En la clase Dog, el método bark indicará como es el que una instancia de la clase debe ladrar. Los métodos deben estar declarados dentro de una clase. Es decir, dentro de las llaves de la clase. ☺ |  |
| **¿Qué va dentro de un método?**  Dentro de las llaves de un método, se deben escribir las instrucciones que le indiquen al método que es lo que debe realizar. El código del método es básicamente un conjunto de declaraciones. Se puede pensar en él como una función o procedimiento. |  |

# Anatomía de una clase

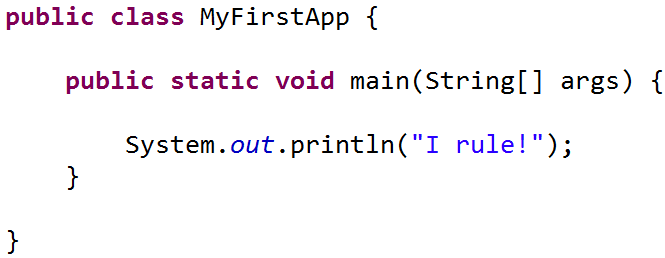
Cuando la JVM se pone en marcha busca la clase que se le indica vía la línea de comandos y busca un método especial, idéntico al siguiente:

**public** **static** **void** main(String[] args) {

//Tu código va aquí ☺

}

La JVM ejecutará todo lo que se encuentre dentro de las llaves {}. Cada aplicación tiene al menos una clase y un método principal.



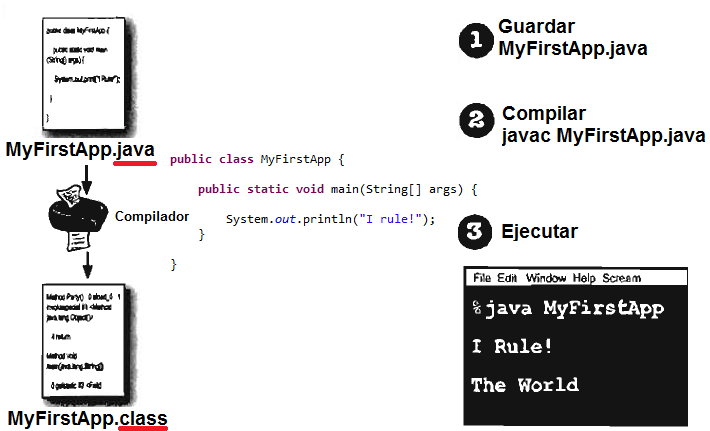
# Codificando una clase con un main

En Java, todo es una clase. Codificarás tu archivo de código fuente (.java), luego lo compilarás para obtener el archivo class (.class). Y cuando ejecutes tu programa…realmente ejecutarás un a clase.

A todo esto ¿qué es ejecutar un programa? Básicamente es decirle a la JVM que “cargue la clase MyFirstApp, ejecute el método main() y que continúe ejecutándose hasta que todo el código del método termine”.

Hay muchos detalles que se verán en adelante, por ahora te centrarás en pensar ¿cómo escribiré código Java que se ejecute? Todo comienza en el main()! Este es el método donde tu programa inicia a ejecutarse.

Sin importar cuantas clases uses, debe haber un método main() para que todo eche andar.



# ¿Qué puedes hacer en un método main? (…y en cualquier método)

Empieza la diversión. Realmente, puedes realizar en tu código lo que normalmente realizar en otros lenguajes para que la computadora haga lo que tu deseas que haga.

Por ejemplo, tu código puedes decirle a la JVM:

|  |  |
| --- | --- |
| * Cada sentencia **debe** terminar en punto y coma:   x = x + 1;   * Los comentarios de una sola línea deben ir con dos slash:   x = 22;  //Esta línea me perturba...   * Más de un espacio en blanco no es problema:   x = 3 ;   * Las variables son declaradas con un nombre y un tipo:   **int** weight;  //tipo: int, nombre: weight   * Las clases y métodos deben ser definidos con un par de llaves de apertura y cierre.   **public** **void** go() {  // tu codigo aqui  } |  |

# Bucles…bucle - ando ando…

Java tiene tres directivas estándar para el uso de bucles: while, do-while y for.

La sintaxis es tan sencilla que probablemente se aburrirán muy pronto. Así que vamos a definir que hacen estas tres directivas de forma rápida.

En las tres directivas tenemos que mientras la condición sea verdadera se ejecutará todo el código interno al bucle, el cual está delimitado por un par de llaves.

La clave está precisamente en la condicional. En Java una prueba condicional es una expresión que da como resultado un valor booleano (verdadero o falso).

Por ejemplo, si dices algo como

**while** siElPracticanteNoLaTumba {

seguirá en el team;

}

Claramente se observa que la condicional acerca que si el practicante del área de sistema no multiplique los ahorros de los clientes del banco de manera errónea, va a seguir formando parte del equipo.

# Pruebas booleanas simples

Puedes hacer pruebas simples verificando el valor de una variable, usando un operador de comparación:

< (menor que)

> (mayor que)

== (igualdad, son dos signos iguales)

Debes notar la diferencia entre el operador de asignación (signo igual simple) y el operador igual (dos signos igual). Es un error común usar = en lugar de ==.

**int** x = 4; //asignándole 4 a la variable x

**while** (x > 3) {

//este código se ejecutará porque

//x es mayor que 3

x = x - 1; //podría ejecutarse infinitamente? Si.

}

**int** z = 27;

**while** (z == 17){

//este código no se ejecutará porque

// z no es igual a 17

}

# Ejemplo de aplicación: Bucle

**public** **class** Loopy {

**public** **static** **void** main(String[] args) {

**int** x = 1;

System.*out*.println("Before the Loop");

**while** (x < 4) {

System.*out*.println("In the loop");

System.*out*.println("Value of x is " + x);

x = x + 1;

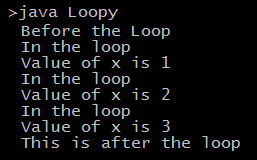
}

System.*out*.println("This is after the loop");

}

}

El resultado que debes obtener es:



|  |  |
| --- | --- |
|  | * Las sentencias terminan en punto y coma ; * Los bloques de código son definidos por un par de llaves {} * Declarar una variable con un nombre y un tipo: int x; * El operador para asignación es un signo igual = * El operador de igualdad es el uso de dos signos igual == * El bucle while ejecuta todas las sentencias que se encuentras dentro de su bloque definido por dos llaves mientras la condicional sea verdadera. * Si la condicional no se cumple o deja de cumplirse, la ejecución continuará desde las sentencias luego de finalizar el bloque delimitado por las dos llaves. * Si la condicional es false, el bloque de código del while no se ejecutará. * Siempre coloca la condicional dentro de paréntesis: while (x == 4) |

# Condicionales…y si…

En Java, un if es básicamente lo mismo que el condicional en un while – pero en lugar de decir “Mientras haya cerveza…”, debe leerlo como “Si aún hay cerveza…”.

**public** **class** IfTest {

**public** **static** **void** main(String[] args) {

**int** x = 3;

**if** (x == 3){

System.*out*.println("x must be 3");

}

System.*out* .println("This runs no matter what");

}

}

El resultado que debes obtener es:



Además, también podemos agregar la directiva else a la condición. Lo cual se traduciría como “Si hay cerveza seguir codificando, de lo contrario conseguir más cerveza y luego continuar…”.

**public** **class** IfTest2 {

**public** **static** **void** main(String[] args) {

**int** x = 2;

**if** (x == 3) {

System.*out*.println("x must be 3");

} **else** {

System.*out*.println("x is NOT 3");

}

System.*out*.println("This runs no matter what ");

}

}

El resultado que debes obtener es:



# Ejemplo de aplicación: Bucle + If

Dada la siguiente salida:



Completa el siguiente código:

**public** **class** DooBee {

**public** **static** **void** main(String[] args) {

**int** x = 1;

**while** (x < \_\_ }{

System.out.\_\_("Doo");

System.out.\_\_("MBee");

x = x + 1;

}

**if**(x == \_\_ }{

System.out.print("DoH");

}

}

}

# Codificando una seria aplicación de negocios

El siguiente código está sintácticamente correcto pero no funciona según se espera. ¿Puedes encontrar el error y mejorarlo?

**public** **class** ElefanteSong {

**public** **static** **void** main(String[] args) {

**int** elefanteNum = 1;

String palabra = "elefantes";

**while** (elefanteNum < 100) {

**if** (elefanteNum == 1) {

palabra = "elefante"; // simple

}

System.*out*.println(elefanteNum + " " + palabra

+ " se columpiaba sobre la tela de una araña, como veia que resistia fueron a llamar otro elefante más!");

elefanteNum = elefanteNum + 1;

**if** (elefanteNum > 1) {

System.*out*.println(elefanteNum + " " + palabra

+ " se columpiaban sobre la tela de una araña, como veia que resistia fueron a llamar otro elefante más!");

}**else** {

System.*out*.println(elefanteNum + " " + palabra

+ " se columpiaba sobre la tela de una araña.");

} // fin del else

**if** (elefanteNum >= 100) {

System.*out*.println("La tela de araña no aguantó más!");

}

} // fin del bucle while

} // fin del metodo main

} // fin de la clase

# Sé el compilador

Para seguir ejercitándonos haremos unos pequeños ejercicios donde nos convertiremos (momentáneamente) en el compilador de Java:

|  |  |
| --- | --- |
| **public** **class** Exerciselb {  **public** **static** **void** main(String[] args) {  **int** x = 1;  **while** ( x < 10) {  **if** ( x > 3) {  System.*out*.println("big X");  }  }  }  } |  |
| **public** **static** **void** main(String[] args) {  **int** x = 5;  **while** ( x > 1 ){  x = x - 1;  **if** ( x < 3) {  System.*out*.println("medium X");  }  }  } |  |
| **public** **class** Exerciselb {  **int** x = 5;  **while** ( x > 1 ){  x = x - 1;  **if** ( x < 3) {  System.*out*.println("small X");  }  }  } |  |