Primeros pasos Java

Programa básico en JAVA

Toda aplicación básica de consola en Java contiene una clase principal con un método **main**. El método **main** es lo primero que se ejecuta cuando se inicia la aplicación desde la línea de comandos, es decir que la ejecución del programa comienza desde allí. De esta forma podemos deducir que cualquier llamado, declaración o método que no sea utilizado de manera directa o indirecta desde el método **main** nunca se ejecutará, con lo de manera indirecta me refiero a que no se haga directamente desde el **main** sino desde un método que llama a otros el cual es llamado desde el **main**, así se ejecutaría de manera indirecta.

En Java el método **main** recibe como argumento un **array** de **String** (más adelante veréis que son los arrays). Este **array** contiene los argumentos enviados por la línea de comandos para la invocación del programa. Generalmente en una aplicación de Java básica, no se hace uso de éste argumento, sin embargo hay algunos casos específicos en los que sí.

No te preocupes si te es difícil esto del argumento, más adelante te quedará más claro, por ahora veamos algo de Java básico, algo acerca de la estructura mínima de un programa escrito en Java.

La estructura básica de un programa desarrollado usando Java es la siguiente:

```
1  public class NombreClase
2  {
3      public static void main (String args[])
4      {
5          //bloque de sentencias;
6          System.out.print("Hola mundo!");
7      }
8  }
```

Línea 1 del programa Java básico:

La primera línea de nuestro código ha declarado una clase llamada "nombre_clase" (allí puedes poner el nombre que desees), la cual es pública (public), esto quiere decir que tendrás acceso a

dicha clase, desde cualquier parte del código sin ningún tipo de complicaciones, ahora, lógicamente tú le podrás poner el nombre que te guste más.

Línea 2 del programa Java básico:

Nuestra línea 2, tiene una llave abriendo "{", esto quiere decir que todo lo que haya después de esta llave forma parte de la clase "nombre_clase" (o como le hayas llamado) e indica el comienzo de las líneas de sentencia de ésta, en este caso, esta la declaración de la función **main()** que está más adelante.

Línea 3 del programa Java básico:

La línea número tres contiene la declaración del método **main**, en cualquier aplicación de Java básica o avanzada, el método **main** es indispensable, éste método es **público**, pero hay algo especial en él: el método **main** es del tipo **void**, osea vacío (una función puede retornar valores del tipo entero, decimal, cadena de texto, entre otros) al poner **void**, estamos queriendo decir que la función **main** no retornará nada al finalizar.

Si se le pusiera por ejemplo **int** (entero) entonces al final de la función, ésta debería retornar algún número o variable de tipo entero, sin embargo al hacer esto, ya no estaríamos hablando del método principal de nuestro programa sino que sería una función creada por nosotros llamada **main** y que es del tipo entero, eso es ya otro tema muy distinto llamado sobrecarga de funciones que se verá más adelante.

No te preocupes si no entiendes bien esto de retorno o no retorno, entero y eso, pues más adelante se trabajará sobre ello a fondo. Lo realmente importante hasta ahora es que comprendas que existe un método principal llamado main y una clase principal pública que lo contiene y que desde este método comienza la ejecución del programa.

La línea tres, también tiene un argumento para el método **main**, es lo que está dentro de los paréntesis "**String args[]**", esto quiere decir que la función **main**, debe recibir un argumento el cual es de tipo **String** y es además un **array**, no te preocupes mucho de esto por ahora. Aunque tu programa sea Java básico y no hagas uso de éste argumento, es **obligatorio** declararlo o tendremos un error de compilación, así que siempre debes ponerlo en tu código.

Línea 4 del programa Java básico:

La línea 4 de nuestro programa básico contiene una llave abriendo que indica el inicio del bloque de instrucciones del método **main.**

Línea 5:

La línea 5 contiene un comentario indicando el bloque de sentencias del método **main**, es decir, las líneas de código que nuestro programa seguirá durante su ejecución, es de notar que cada sentencia termina con punto y coma ";".

Se entiende como línea de sentencia, a las líneas de código que en ellas contienen la declaración de alguna variable, o alguna operación,(suma resta, división, etc.) o cualquier acción, como modificar, asignar, o eliminar algo. Evidentemente el bloque de sentencias puede estar compuesto por una cantidad ilimitada de instrucciones (no es una única línea). Una vez que éste bloque de sentencias llega a su fin, también lo hará la ejecución de nuestra aplicación.

Línea 6:

La línea 6 contiene una sentencia que imprime por pantalla el texto **Hola mundo!.** La función que ejecuta es **System.out.print()**, esta función pertenece al paquete de funciones **java.lang** que forma parte de las librerías estandars de la máquina virtual. Estas librerías las importa JAVA por defecto sin que nosotros se lo tengamos que indicar al programa.

Línea 7 y 8:

Estas líneas contienen una llave cerrando "}" eso nos indica que en esta termina la ejecución de alguna función (para identificar esta función que se cierra, sigue la indentación y seguramente sabrás cual es la que estamos cerrando). Pues bien, si no te has equivocado, habrás notado que la función que esta llave está cerrando es **main()**, y luego está la línea 8 que cierra la definición de nuestra clase y da por terminado el código de nuestro programa básico en Java.

Ejemplo de otro programa de Java básico

El programa que pondré a continuación simplemente mostrará en pantalla la tabla del 12, algo bastante sencillo y que podrás probar tu mismo en tu casa copiando el código. Recuerda no preocuparte mucho por comprender a la perfección todo el bloque de sentencias dentro del **main**.

```
1  public class NombreClase
2  {
3      public static void main (String args[])
4      {
5          for(int i = 0; i <= 12; i++) {
6              System.out.print("12 * "+ i + " = " + 12 * i + "\n");
7          }
8
9      }
0  }
```

¿Serias capaz de predecir el resultado del programa anterior?

Deberías intentar cambiar el código, mover algún número, cambiar nombres, valores y todo lo que se te ocurra en él, te aseguro que en el proceso aprenderás unas cuantas cosas valiosas que te van a facilitar el proceso de aprendizaje y mejorarán tus conocimientos, recuerda que **la única forma de aprender a programar es programando**, así que.... ¿Qué esperas?

¿Que son los paquetes en Java?

Los paquetes en Java (**packages**) son la forma en la que Java nos permite agrupar de una manera lógica los componentes de nuestra aplicación que estén relacionados entre sí.

Los paquetes permiten poner en su interior casi cualquier cosa como: clases, interfaces, archivos de texto, entre otros. De este modo, los paquetes en Java ayudan a darle una buena organización a la aplicación ya que permiten modularizar o categorizar las diferentes estructuras que componen nuestro software.

Para utilizar los elementos de un paquete es necesario importarlo, usando para ello la sentencia **import**.

La funcionalidad de una aplicación Java se implementa habitualmente en múltiples clases, entre las que suelen existir distintas relaciones. Las clases se agrupan en paquetas y cada módulo de código establece, mediante la palabra clave **package** al inicio, a qué paquete pertenece. Con la cláusula **import** cualquier módulo de código puede hacer referencia a elementos definidos en otros paquetes.

Ya sabemos para qué sirven los paquetes en Java y sus características principales. Vamos ahora a aprender cómo usarlos y qué cambios generan estos en la estructura de nuestro proyecto.

Para declarar un paquete en Java se hace uso de la palabra reservada "package" seguido de la "ruta" del paquete, como se muestra a continuación:

```
package ruta.del.paquete;
```

Hay varias cosas que clarificar aquí. Como verás, la sintaxis es bastante simple y comprensible pero hay algunos detalles a tener en cuenta, veamos.

- *El paquete en Java se declara antes que cualquier otra cosa:*

  La declaración del paquete debe estar al principio del archivo Java, es decir, es la primera línea que se debe ver en nuestro código o archivo **.java**. Primero se declara el **paquete**, y luego podremos poner los **imports** y luego las clases, interfaces, métodos, etc.

- *Cada punto en la ruta del paquete es una nueva carpeta:*

  Cuando se escribe la ruta del paquete en Java, se pueden especificar una ruta compleja usando el punto ".", por ejemplo en el código anterior (**package ruta.del.paquete;**), nuestra clase, interfaz o archivo estará ubicado en una carpeta llamada "paquete" la cual a su vez está en la carpeta "del" y esta también se encuentra dentro de una carpeta llamada "ruta".

De este modo si queremos por ejemplo que una clase quede al interior de una carpeta llamada "mi_paquete" y ésta al interior de una carpeta llamada "otro_paquete" pondríamos:

```
package otro_paquete.mi_paquete;

 /*Se usa el punto para separar cada carpeta equivale a la
     ruta otro_paquete/mi_paquete dentro del proyecto */

public class mi_clase

 { ....

 }
```

- *Prácticas recomendadas para nombrar paquetes en Java.*

  También hay varias buenas prácticas y recomendaciones para crear y declarar paquetes en Java. En general, los paquetes en java se declaran siempre en minúsculas y en caso de ser necesario las palabras se separan usando un guión bajo "_".

- *Si no se declara un paquete (paquete por defecto):*

  Si decidimos no declarar un paquete para nuestra clase, ésta quedará en un paquete que se conoce como paquete por defecto (**default package**), en este paquete estarán todas las clases que no tengan un paquete declarado. Aunque esto no genera errores de compilación ni nada parecido, siempre es recomendable declarar un paquete a cada componente de nuestro programa Java para poder darle diferentes niveles de seguridad o acceso a dichos componentes y mantener todo ordenado.

- *Para evitar nombres conflictivos, en ambientes profesionales los paquetes reciben los nombres del nombre de dominio de la compañía en orden inverso, por ejemplo: com.guru99. com.microsoft, com.infosys etc.*

En estos momentos ya tenemos claro qué es y para qué sirve un paquete en Java, también sabemos cómo se declaran los paquetes en Java y probablemente tendremos una noción de los cambios que estos generan al interior de la estructura de nuestro proyecto.

Librerías en Java

En Java y en varios lenguajes de programación más, existe el concepto de librerías. Una librería en Java se puede entender como un conjunto de clases, que poseen una serie de métodos y atributos. Lo realmente interesante de estas librerías para Java es que facilitan muchas operaciones. De una forma más completa, las librerías en Java nos permiten reutilizar código, es decir que podemos

hacer uso de los métodos, clases y atributos que componen la librería evitando así tener que implementar nosotros mismos esas funcionalidades.

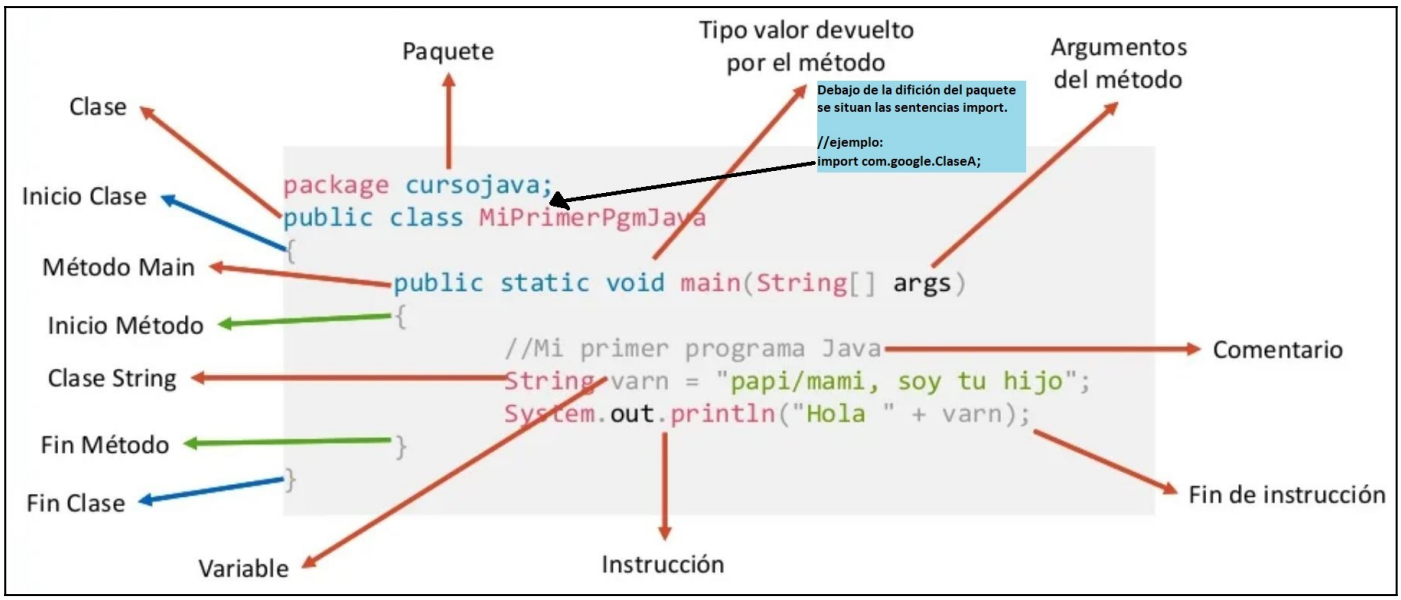
Imaginemos cómo hacer un código para conseguir que nuestra aplicación imprima algo por pantalla o conseguir que nuestra aplicación pueda hacer uso de arreglos dinámicos con cualquier tipo de dato. Por medio de librerías, para imprimir texto en pantalla en Java basta con usar **System.out.print()** y se puede hacer uso de **ArrayList**, para tener arreglos dinámicos y demás, en Java, esto es así de fácil y todo es gracias a las librerías, la **clase System** que pertenece a la librería **java.lang** (ésta es la librería estándar de Java y no es necesario importarla). Ocurre de la misma forma con el **ArrayList**, no podríamos hacer uso de éste si no existiera la **librería java.util**. Cabe mencionar que estas librerías no solo poseen estas clases sino que poseen una gran cantidad de clases Java adicionales que nos permiten llevar a cabo operaciones muy complejas con gran facilidad.

Lo más interesante de las librerías en Java, es que nosotros también podemos crearlas y hacer uso de ellas al interior de nuestros proyectos. Básicamente un paquete en Java puede ser una librería, sin embargo una librería Java completa puede estar conformada por muchos paquetes más. Al importar un paquete podemos hacer uso de las clases, métodos y atributos que lo conforman, así que eso de las librerías no es solo trabajo de otros sino que podemos crear nuestras propias librerías y usarlas, incluso podemos hacer uso de un proyecto completo nuestro al interior de otro.

Pues bien, hasta ahora todo suena muy impresionante y se entiende que es bastante útil y que nos facilita enormemente la vida, pero **¿cómo importar una librería en Java? ¿cómo se usa una librería de Java? ¿qué es importar una librería?** o mejor aún **¿qué es import en Java?** pues bien, vamos a resolver todas estas dudas a continuación.

¿Donde se localizan los imports en Java?

Al igual que sucede con la declaración de los paquetes, realizar el import en Java también tiene una localización especifica al interior del archivo java. La importación de librerías y demás siempre debe ir después de la declaración del paquete, si la clase no posee alguna declaración de paquete (pertenece al paquete por defecto), entonces los **import** deberán estar en las primeras líneas del archivo.



¿Cómo importar clases de nuestro proyecto?

Ejemplo 1: Sintaxis para import en Java

```
package paquete.mipaquete;

import paquete2.otropaquete.*;

class Ejemplo1{}
```

En el código anterior básicamente estamos diciendo que la clase contenida en el archivo Java, estará en el paquete "**paquete/mipaquete**", es decir, esa será la ruta de acceso a esa clase en el proyecto. En la siguiente línea hemos realizado el **import** de un paquete. Por medio del "*" hemos indicado a Java que queremos importar todas las clases pertenecientes a dicho paquete, puede ser una o más.

Ejemplo 2: Sintaxis para import en Java

```
package paquete.mipaquete;

import paquete3.otropaquete2.MiClase;

class Ejemplo2{}
```

En el código anterior hemos usado el mismo paquete para la clase pero hay una pequeña diferencia al momento del **import**, en esta ocasión hemos importado una clase directamente (MiClase), es decir, como no usamos el "*" no hemos importado todas las clases pertenecientes al paquete "**paquete3.otropaquete2**" sino que solo hemos importado a "**MiClase**"

Ejemplo 3: Sintaxis para import en Java

```
import paquete.mipaquete.*;

class Ejemplo3{}
```

En esta ocasión se puede apreciar que no hemos declarado paquete alguno para nuestra clase y que únicamente hemos realizado el **import**, sin embargo este **import** como tal es algo particular para nosotros, pues hemos importado TODAS las clases pertenecientes al paquete "**paquete.mipaquete**" que es el paquete al cual pertenecen las clases del ejemplo 1 y 2, esto quiere decir que al interior de la clase Ejemplo3, podemos usar las clases Ejemplo1 y Ejemplo2.

¿Cómo importar las clases y librerías de la API Java?

Importar clases propias de Java es prácticamente igual que importar clases de nuestros propios paquetes, sin embargo, ¿cuáles son esas clases?, ¿cómo conocerlas?, ¿cuál es el paquete al que pertenecen?, pues bien, de una vez te comento que Java está conformado por una enorme cantidad de clases y paquetes y desde mi punto de vista creo que es bastante complejo conocerlas todas, aunque si es importante conocer las más comunes y útiles, básicamente lo que debes hacer es ingresar a la API de Java y allí encontrarás todas las clases que componen el lenguaje, el paquete al que pertenecen, los métodos que poseen, los atributos y una breve explicación de cada uno de estos. Si ingresas a la documentación de Java verás que por ejemplo existe una clase llamada ArrayList, ésta pertenece al paquete **java.util**, esto quiere decir que para importarla debemos usar esa dirección, veamos:

Ejemplo 4: Importando clases de Java

```
import java.util.ArrayList;

class Ejemplo4{}
```

Aquí hemos importado la clase **ArrayList** y podremos hacer uso de sus componentes, siempre y cuando sean públicos.

Ejemplo 5: Importando clases de Java

```
import java.util.*;

class Ejemplo5{}
```

Aquí hemos importado TODAS las clases pertenecientes al paquete "**java.util**" lo cual incluye la clase **ArrayList** y por lo tanto es funcional y válido.

Ejemplo 6: Importando clases de Java

```
import java.util.ArrayList;
import java.io.* //importando clases para entrada y salida (IN OUT)
import java.math.* //clases utiles para operaciones y valores matematicos

class Ejemplo6{}
```

Como podrás ver, existen muchas librerías muy útiles y es bastante sencillo hacer uso de ellas cuando sea necesario y así ahorrarnos mucho tiempo, trabajo y esfuerzo.

Muy bien, hasta ahora hemos aprendido todo lo necesario acerca del **import** en Java, sabemos cómo importar nuestra propias clases, importar todas las clases de un paquete, también vimos que Java posee una API y que está conformada por una enorme cantidad de Clases, interfaces, excepciones, etc. y que en cualquier momento podemos hacer uso de cualquiera de estas.