

PROGRAMACIÓN III

Una primera aproximación a JAVA

Capítulo 1

- **Características de Java**
- **Máquina Virtual de Java (JVM)**
- **Garbage Collection**
- **Documentación de Java (API)**
- **Un Entorno de Desarrollo para Java**
- **Un primer programa en Java**
- **Compilando y Ejecutando programas sencillos**
- **Documentando nuestros programas (JavaDocs)**
- **Bases del Lenguaje**
- **Estructuras de Control**
- **Strings en Java**
- **Clases en Java**

Características de Java

- **Orientado a Objetos**

- La programación es siempre orientada a objetos. Distinto de C++, que combina orientación a objetos con Programación Estructurada.

- **Multiplataforma**

- Un programa en JAVA puede compilarse bajo una plataforma y luego ejecutarse en esa u otras plataformas

Ejemplo: Puedo compilar mi programa en una máquina Windows y ejecutarlo en una máquina Linux.

- **Interpretado**

- Al compilar un programa JAVA no se genera un ejecutable como en C++. Se genera un código especial que es traducido (en tiempo de ejecución) a la máquina donde se esté ejecutando el programa

- **Multitarea**

- En JAVA se pueden escribir programas (procesos) que luego ejecuten en forma concurrente.

Máquina Virtual de Java (JVM)

¿ Qué es la Máquina Virtual de Java ?

- Es una “máquina imaginaria” que está implementada mediante software sobre una máquina verdadera (por ejemplo, Windows o Linux)
- Cuando “instalamos Java” en una máquina, lo que hacemos es instalar :
 - La máquina Virtual de Java (JVM)
 - Las bibliotecas de clases predefinidas de Java

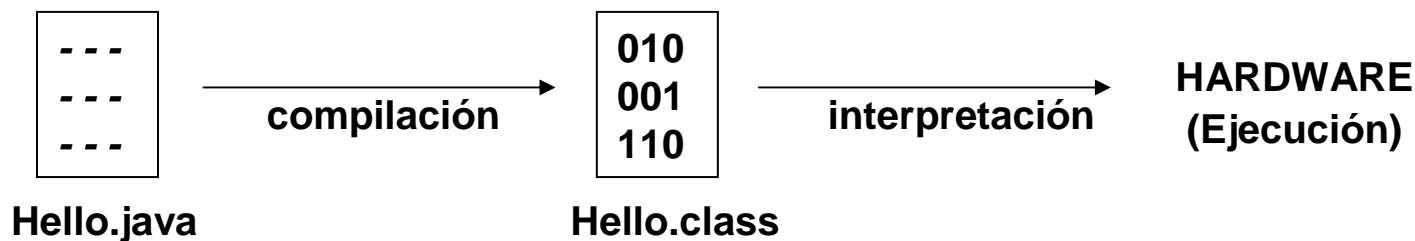
En definitiva, todo lo necesario para poder trabajar con Java.

- La JVM es quien se encarga de compilar nuestros programas y hacer que se ejecuten en la plataforma sobre la cual esté instalado el paquete (no importa si es la misma máquina donde se compiló o no).

Máquina Virtual de Java (JVM)

¿ Cómo funciona la Máquina Virtual de Java ?

- Cuando escribimos un programa, lo guardamos en un archivo con extensión “.java” (contiene el código fuente del programa)
- Luego, al compilar nuestro programa, la JVM lo “traduce” a un código especial llamado “código de bytes”. Dicho código es almacenado en un archivo con extensión “.class”
- Cuando ejecutamos el programa, la JVM toma las instrucciones del código de bytes y las **interpreta**, traduciéndolas al lenguaje de la máquina donde esté ejecutando (puede ser otra diferente a la máquina donde se compiló).



Máquina Virtual de Java (JVM)

Observación interesante:

- El código de bytes generado por una JVM instalada sobre una plataforma puede ser interpretado por una JVM instalada en otra plataforma.

Ejemplo: El código de bytes generado por una JVM sobre una máquina Windows puede ser interpretado por una JVM sobre Linux.

- Esta propiedad es la que permite que Java sea multiplataforma. Puedo compilar mi programa en una máquina y ejecutarlo en otra, sin preocuparme por su plataforma.
- La única restricción para que esto funcione es que ambas máquinas tengan instalado el paquete de Java.
- Existen JVM's disponibles para múltiples plataformas (Hay una JVM para Windows, otra para Ubuntu, etc.)

Garbage Collection

- Cuando en otros lenguajes se trabaja con memoria dinámica, es responsabilidad del programador liberar toda la memoria pedida.
- Java también maneja memoria dinámica, pero a diferencia de otros lenguajes, provee un mecanismo que **exime al programador de la responsabilidad de liberar la memoria**

¿Cómo funciona este mecanismo?

- Java posee un proceso de bajo nivel llamado *Garbage Collector* (recolector de basura). Este proceso se ejecuta automáticamente y en forma periódica (durante la ejecución de nuestro programa). Lo que hace es encontrar y liberar memoria que no esté accedida desde ningún lugar.
- Por lo tanto, lo único que debemos hacer para liberar memoria es **desreferenciarla** (o sea, lo que en C++ se conoce como “dejar basura”) Luego el “basurero” (Garbage Collector) se encarga de liberarla.

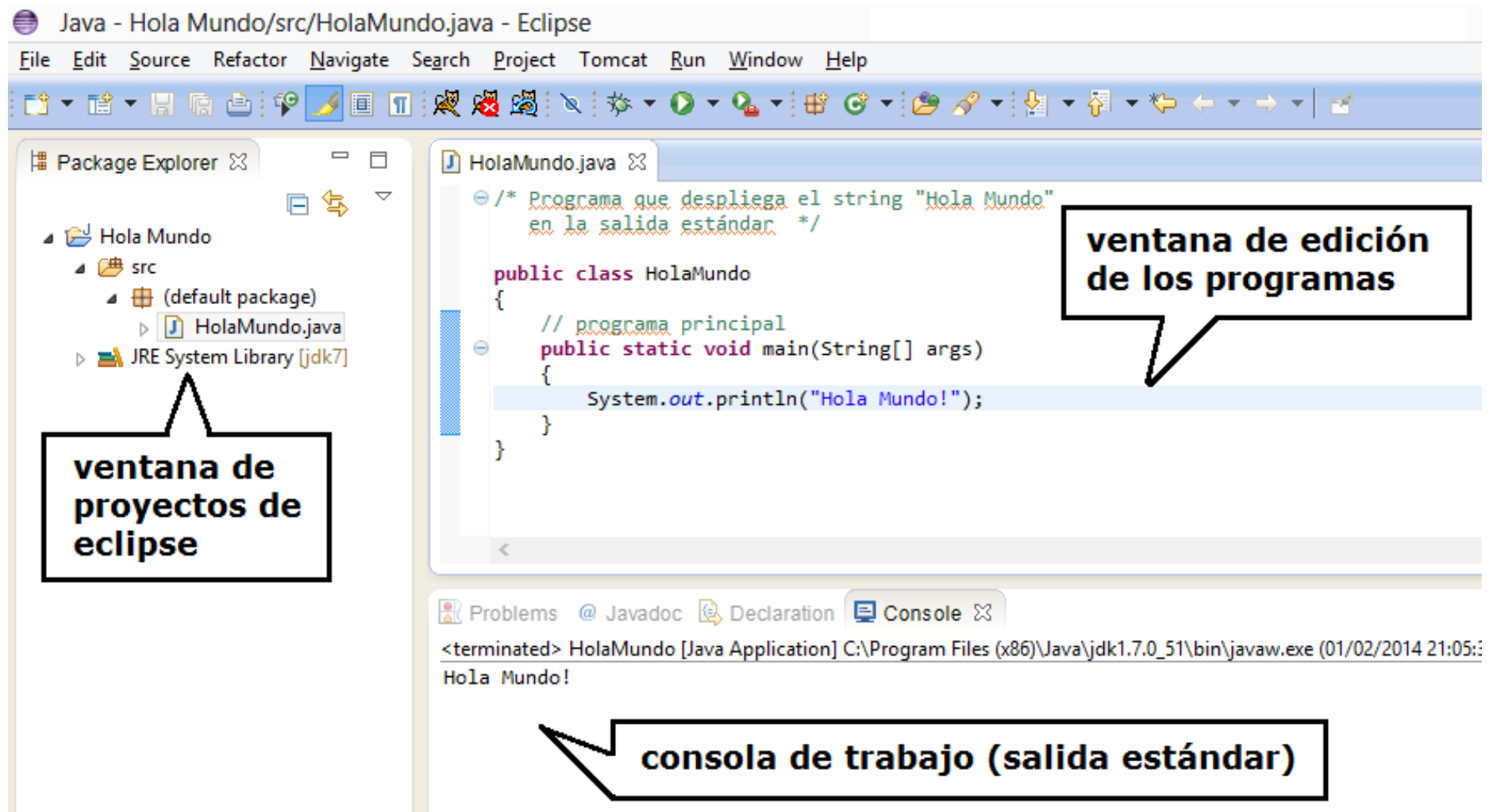
Documentación de Java (API)

- Consiste en un conjunto de páginas web que contienen información muy útil e interesante sobre las múltiples bibliotecas de clases que vienen con el paquete Java
- Navegando a través de dichas páginas web se puede :
 - Entender el funcionamiento de clases predefinidas de Java
 - Investigar qué atributos y métodos posee una clase dada
 - Conocer y descubrir nuevas clases
 - Aprender más acerca de Java
- La API para la versión de Java JDK8 también se puede consultar en forma on-line en <https://docs.oracle.com/javase/8/docs/api/>

Un Entorno de Desarrollo para Java

- A diferencia de otros lenguajes, la instalación de Java puede ser ***independiente*** del editor (IDE o entorno de desarrollo) que utilicemos para escribir nuestros programas.
- Esto significa que podemos utilizar ***diferentes*** editores para escribir, compilar y ejecutar nuestros programas.
- Existen ***muchos*** editores posibles para desarrollar en Java. Por ejemplo:
 - NotePad, WordPad (editores de texto plano, genéricos)
 - NetBeans, Eclipse (editores específicos para Java)
- Los editores específicos son mucho más apropiados para desarrollar en Java ya que poseen facilidades incorporadas que mejoran y facilitan el proceso de desarrollo (Debugger, Proyectos, Asistentes, etc.)
- En este taller vamos a utilizar ***Eclipse***, que es un editor muy completo y además es de libre distribución (al igual que Java).

Plataforma Eclipse



Un primer programa en Java

```
1.  /* Programa que despliega el string
2.     "Hola Mundo" en la salida estándar */
3.
4.  public class HolaMundo
5.  {
6.      // programa principal
7.      public static void main (String args [])
8.      {
9.          System.out.println ("Hola Mundo");
10.     }
11. }
```

Un primer programa en Java

Descripción de HolaMundo

➤ Líneas 1 y 2

```
1. /* Programa que despliega el string "Hola Mundo"  
2.   en la salida estándar */
```

- Corresponden a comentarios, delimitados de igual forma que en C++

➤ Línea 4

```
4. public class HolaMundo
```

- Especifica el nombre de la clase contenida dentro del programa.
- En Java, todo programa contiene una única clase pública, cuyo nombre **debe coincidir** con el nombre del archivo donde se guarda (En este caso es **HolaMundo.java**)

Un primer programa en Java

Descripción de HolaMundo (continuación)

➤ Línea 6

6. `// programa principal`

- Corresponde a un comentario que ocupa toda la línea, igual que en C++

➤ Línea 7

7. `public static void main (String [] args)`

- Corresponde al cabezal del único método que posee la clase (el **main**).
- Al igual que en C++, el método **main** especifica el bloque inicial del programa, donde éste comienza su ejecución.

Un primer programa en Java

Descripción de HolaMundo (continuación)

- `public` - Esta etiqueta indica que el método es accesible desde fuera de la clase (incluso para el intérprete de Java).
- `static` - Indica al compilador que el método `main` es un método de la clase y no de sus objetos (más adelante volveremos sobre esto).
- `void` - Al igual que en C++ , especifica que el método no retorna nada (es un ***procedimiento***).
- `String args []` - El método `main` recibe un arreglo de strings. Es una forma de ingresarle datos al programa al momento de iniciar su ejecución (más adelante veremos cómo)

Un primer programa en Java

Descripción de HolaMundo (continuación)

➤ Línea 9

```
9. System.out.println ("Hola Mundo");
```

- Especifica una **invocación** a un método predefinido de Java que muestra mensajes en la salida estándar.
- Este método se llama `println` y está contenido dentro del atributo `out` contenido dentro de la clase `System`.

Referencia:

API - `java.lang.System`

Allí se pueden encontrar otros métodos que muestran mensajes en la salida estándar.

Un primer programa en Java

Sintaxis de Java - Muy parecida a la de C++

- ***Coincide*** con la de C++ en que...
 - Las instrucciones finalizan con punto y coma
 - Los distintos bloques se encierran entre llaves
 - Las variables se declaran de igual forma que en C++
 - La asignación se realiza mediante el operador `=`
 - Las expresiones se construyen de idéntica forma a C++ y con los mismos operadores (lógicos y aritméticos)
 - Las estructuras de control (`if`, `switch`, `while`, `do-while`, `for`) se mantienen idénticas a las de C++
- ***Difiere*** de la de C++ en que...
 - Las declaraciones y los encabezados de los métodos pueden ser ligeramente diferentes
 - Hay variación en muchas de las palabras reservadas

Compilando Programas Sencillos

Desde una Consola del Sistema Operativo

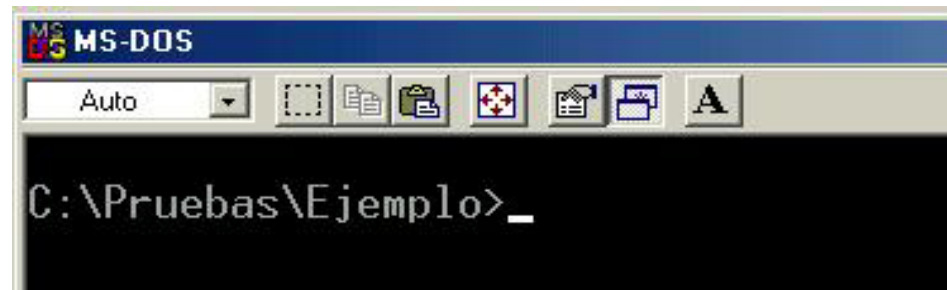
- Esta es la primer manera que tenemos de compilar un programa sencillo en Java. Se trata de abrir una consola de trabajo del Sistema Operativo y digitar un comando que permita compilar el programa.

Ejemplo:

Supongamos que en Windows tenemos nuestro archivo fuente “HolaMundo.java” en la siguiente ubicación:

`c:\Pruebas\Ejemplo`

Debemos abrir una consola del Sistema y posicionarnos en dicha ubicación.



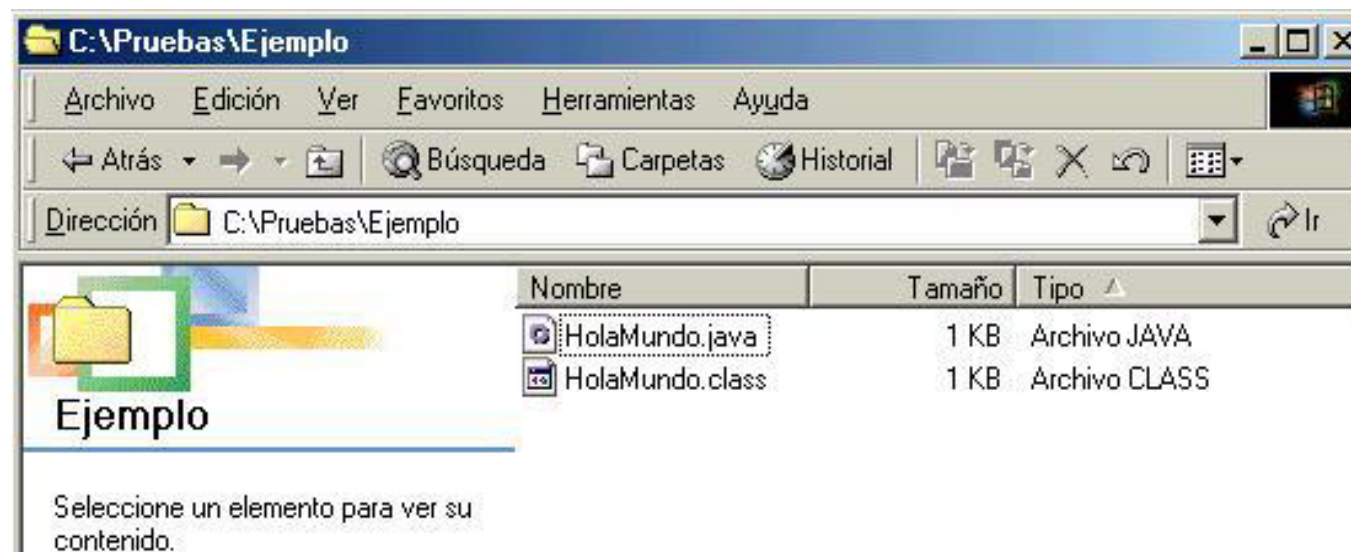
Compilando Programas Sencillos

Desde una Consola del Sistema Operativo (continuación)

Una vez allí debemos digitar el siguiente comando:

```
javac HolaMundo.java
```

El resultado será la generación del archivo “HolaMundo.class” ubicado en el mismo directorio que el archivo fuente (“HolaMundo.java”)

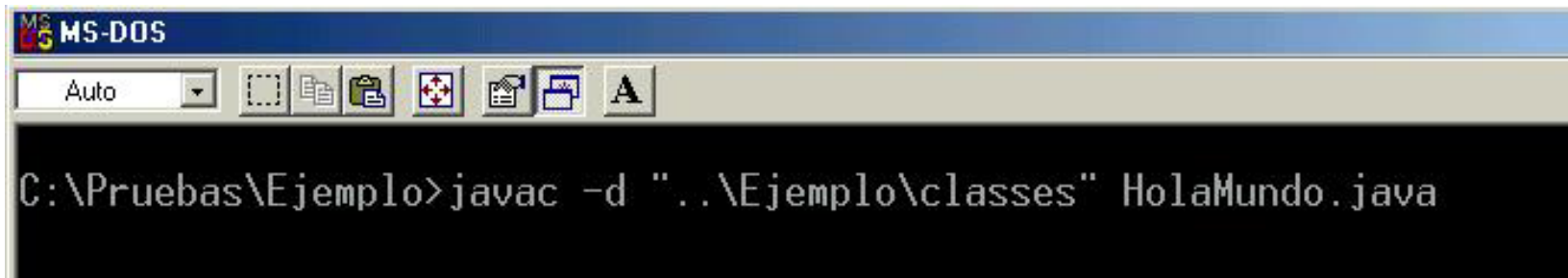


Compilando Programas Sencillos

Desde una Consola del Sistema Operativo (continuación)

➤ Otro Ejemplo:

En ocasiones nos interesa tener los archivos fuente (.java) separados de los archivos de bytes (.class). El comando `javac` nos permite almacenar los (.class) en otro directorio.



```
C:\Pruebas\Ejemplo>javac -d "../Ejemplo/classes" HolaMundo.java
```

En este ejemplo estamos almacenando el archivo “HolaMundo.class” dentro de la siguiente ubicación: `c:\Pruebas\Ejemplo\classes`

Convención: Es muy común entre programadores Java almacenar los archivos fuente en una carpeta llamada **src** y los archivos de bytes en otra carpeta llamada **classes** o **bin**

Compilando Programas Sencillos

Desde un Entorno de Desarrollo

- Los entornos de desarrollo para Java permiten compilar los programas sin necesidad de digitar comandos en una consola. Típicamente lo que hacen es crear **Proyectos** para compilar los archivos fuente.
- Los Proyectos permiten compilar todas las clases de la aplicación que estamos desarrollando en forma colectiva y con facilidad.
- Se acostumbra que los Proyectos respeten la división de archivos fuente y archivos de bytes mencionada anteriormente (carpetas **src** y **classes**).
- Ahora vamos a ver los pasos necesarios para crear un Proyecto de trabajo en **Eclipse** (el entorno de desarrollo elegido para el taller).

Compilando Programas Sencillos

Pasos para crear un Proyecto de Trabajo en *Eclipse Oxygen*

1. Crear (a nivel del Sistema Operativo) una carpeta para el Proyecto.
2. Abrir Eclipse y elegir la opción: File -> New -> Project.
3. Elegir “Java” -> “Java Project”, presionar “Next” y ponerle un nombre al Proyecto.
4. En “Project Layout” seleccionar “create separate source and output folders” y presionar “Configure default...”.
5. Nombrar “src” y “bin” a las carpetas que contendrán los archivos fuente (.java) y de bytes (.class) respectivamente (“Source folder name” y “Output folder name”). Presionar “Apply and Close”.
6. En “JRE” presionar “Configure JREs...” y verificar que en Installed JREs figure la ruta a nuestra instalación de Java. En caso de no ser así, presionar “Edit” y en el campo “JRE home” cargar la ruta a nuestra carpeta de instalación de Java (jdk1.8.0_161).

Compilando Programas Sencillos

Pasos para crear un Proyecto de Trabajo en *Eclipse Kepler*

7. De nuevo en la ventana anterior, **destildar** la opción “Use default location”. Presionar “browse” y elegir la carpeta creada para el proyecto en el paso 1 (allí es donde lo guardaremos).
8. Por último, presionar el botón “Next” y luego “Finish”. Una vez creado el Proyecto, el mismo queda accesible desde la ventana de Proyectos de Eclipse (Ventana “Package Explorer”).

También es posible importar a eclipse otro project creado anteriormente:

1. Elegir la opción File -> Import -> General -> Existing Projects into Workspace. Presionar “next”.
2. En “Select root directory:” presionar “Browse” y elegir la carpeta del Sistema Operativo que contiene los archivos del project a importar.
3. Presionar “Finish”. El project habrá sido importado en eclipse.

Compilando Programas Sencillos

Desde un Entorno de Desarrollo (continuación)

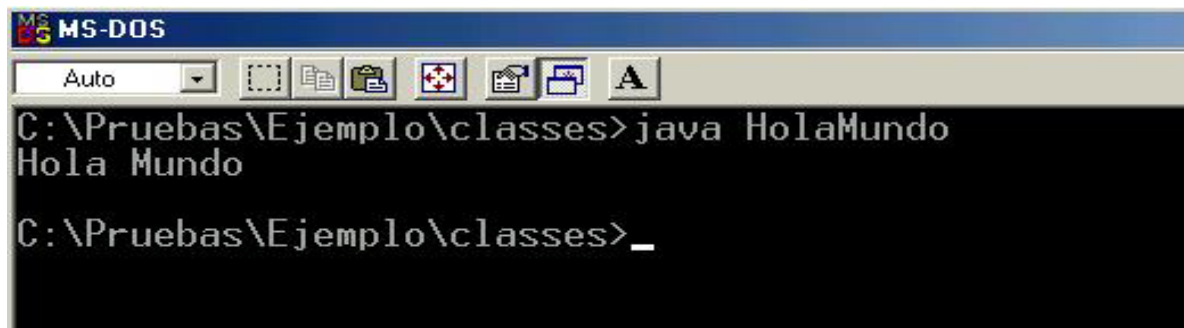
- Una vez creado el Proyecto de trabajo en **Eclipse** vamos a ver los pasos necesarios para escribir y compilar el programa.
- 1. Abrir el Proyecto creado y elegir la opción: File->New->Class.
- 2. Ponerle un nombre a la nueva clase (Ej: “HolaMundo”) y guardarla dentro de la sub-carpeta **src** del Proyecto. Al presionar “Finish” ya podemos escribir el código fuente de la clase.
- 3. En **Eclipse**, la compilación se realiza en tiempo de edición (como si fuera un corrector ortográfico). Si hay algún error de compilación, el mismo es subrayado con un color. Basta posicionar el mouse sobre la sección subrayada y el error es desplegado.
- 4. Cuando el archivo compiló correctamente, el archivo (.class) es automáticamente generado y guardado dentro de la sub-carpeta **classes** del Proyecto.

Ejecutando Programas Sencillos

Desde una Consola del Sistema Operativo

- Para ejecutar un programa de esta manera debemos posicionarnos en el directorio que contiene al archivo (.class) y digitar el siguiente comando: `java NombrePrograma arg0 arg1 arg2 ...`

Ejemplo:



```
MS-DOS
Auto
C:\Pruebas\Ejemplo\classes>java HolaMundo
Hola Mundo
C:\Pruebas\Ejemplo\classes>_
```

Los mensajes que el programa despliegue en pantalla serán mostrados en la misma consola a continuación del comando digitado. Nótese que **NO** incluimos ninguna extensión al nombre del programa cuando ejecutamos el comando.

Ejecutando Programas Sencillos

Desde un Entorno de Desarrollo (Eclipse)

- Para ejecutar un programa desde Eclipse, tenemos que tener abierto el Proyecto correspondiente al programa y seguir los siguientes pasos:
 1. Seleccionar la ventana que despliega el código fuente del programa (la clase que contiene el método `main`).
 2. Elegir la opción: Run -> Run As -> Java Application. Los mensajes que el programa despliegue serán mostrados en la consola de trabajo (Salida estándar) de Eclipse.

Observación: Para que una clase pueda ser ejecutada, la misma debe contar con el método `main`. En caso contrario, el intérprete de Java no se podrá ejecutar dicha clase. (Más adelante veremos que no todas las clases poseen un método `main`).

Documentando nuestros programas

- En C++ escribíamos **dos** archivos por cada clase que programábamos (archivo .h y archivo .cpp). Lo hacíamos con la finalidad de separar la **interfaz** de la clase (los cabecales de los métodos) de su correspondiente **implementación**
- Típicamente, el archivo (.h) contenía **comentarios** en los cabecales a efectos de que otro programador pudiera utilizar los métodos sin necesidad de conocer su implementación.
- En Java solamente escribimos **un** archivo por cada clase (.java), el cual cumple la función del archivo (.cpp) en C++ (es decir, proveer la implementación de los métodos de la clase).
- Para poder proveer una **interfaz** similar a la provista por los archivos (.h) de C++, en Java contamos con los **JavaDocs**. Los mismos son archivos HTML que presentan documentación acerca de nuestras clases para que las mismas puedan ser usadas por otros programadores sin necesidad de conocer su implementación.

Generando JavaDocs

- Al generar el **JavaDoc** correspondiente a una clase, podemos incluir en ella comentarios especiales delimitados por `/**` y `*/`. La información encerrada en ellos será incluida en el **JavaDoc**.

Ejemplo:

```
/** Clase de ejemplo que muestra la sintaxis
 * elemental de un programa en java
 * @author Federico Gómez
 * @version 1.0
 */
public class HolaMundo
{
    /** Método main de la clase
     * @param args un arreglo de Strings
     * @return No retorna nada */
    public static void main (String [] args)
    { ... }
}
```

Generando JavaDocs

Desde una Consola del Sistema Operativo

- Debemos posicionarnos en la carpeta que contiene al archivo fuente de nuestra clase y digitar el siguiente comando:

```
javadoc NombreClase.java
```

Desde un Entorno de Desarrollo (Eclipse)

- Vamos a la opción Project -> Generate JavaDoc. Allí indicamos la ruta al comando de java que genera javadocs ("javadoc.exe" contenido dentro de la carpeta **bin** bajo el directorio de instalación de java), elegimos las clases del Proyecto que deseamos documentar en el JavaDoc y por último le indicamos dónde almacenarlo.

El **JavaDoc** resultante es un conjunto de archivos HTML que contiene toda la información que pusimos en los comentarios especiales. Dichos archivos lucen exactamente igual que los archivos de la API de Java y sirven para que otros programadores puedan utilizar nuestras clases sin tener que conocer su implementación.

Generando JavaDocs



Bases del Lenguaje

Identificadores

- Son nombres que les damos a las clases, variables y métodos.
- Sólo pueden contener letras, dígitos o infraguiones.
- Deben constar de una sola palabra
- Sólo pueden comenzar con letra o infraguión
- No pueden ser palabras reservadas (Ej: class, void, boolean, etc.)

Convención:

- Nombres de clases empiezan con mayúscula
- Nombres de métodos, variables o instancias empiezan con minúscula
- Todos ellos van en minúscula, cambiando a mayúscula para simular un cambio de palabra dentro del identificador

Ejemplos: `public class MiPropiaClase`
 `int miVariableEntera`
 `public void miPropioMetodo()`

Bases del Lenguaje

Tipos de datos, variables, expresiones y asignación

➤ Tipos de Datos

- `char` (caracteres Unicode)
- `int` (enteros entre -2^{15} y $2^{15} - 1$)
- `long` (enteros entre -2^{31} y $2^{31} - 1$)
- `double` (reales de 64 bits)
- `boolean` (`false`, `true`)

➤ Variables, Expresiones y Asignación

```
int num = 5;
boolean cierto = false;
double x,y,z;
char c = '\n';
x = 2*num + 35.5;
cierto = (num == 5) && (!true);
final double IVA = 1.23; // constante simbólica
```

Estructuras de Control

Selección Simple (if – else)

➤ Ejemplo:

```
// SeleccionSimple.java

public class SeleccionSimple
{
    public static void main (String args[])
    {
        double x = Math.random();
        if (x < 0.5)
            System.out.println ("Cerca de cero");
        else
            System.out.println ("Cerca de uno");
    }
}
```


Estructuras de Control

Selección Múltiple (switch)

➤ Ejemplo:

```
// SeleccionMultiple.java

public class SeleccionMultiple
{
    public static void main (String args[])
    {
        int m = (int) (Math.random()*10) ;
        switch (m)
        {
            case 0 : System.out.println ("cero");
                    break ;
            case 1 : System.out.println ("uno");
                    break ;
            case 2 : System.out.println ("dos") ;
                    break ;
            default: System.out.println ("mayor");
        }
    }
}
```

Estructuras de Control

Iteración con control previo (while)

➤ Ejemplo:

```
// ControlPrevio.java
```

```
public class ControlPrevio
{
    public static void main (String args[])
    {
        double x, y=0;

        while (y < 20)
        {
            x = Math.random();
            y = y + x;
        }

        System.out.println (y);
    }
}
```

Estructuras de Control

Iteración con control posterior (do – while)

➤ Ejemplo:

```
// ControlPosterior.java
```

```
public class ControlPosterior
{
    public static void main (String args[])
    {
        int num;

        do
        {
            num = (int) (Math.random() * 10);
        }
        while ((num > 2) && (num < 8));

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

Estructuras de Control

Iteración por subrangos (for)

➤ Ejemplo:

```
// Subrangos.java
public class Subrangos
{
    public static void main (String args[])
    {
        int num=0;
        for (int i=0; i<10; i++)
            if (i % 2 != 0)
                num = num + (int) (Math.random() * 10);
            else
                num = num + (int) (Math.random() * 20);

        System.out.println (num);
    }
}
```

Strings en Java

- Conforman una clase, **NO** un tipo de datos.
- Ya son predefinidos por Java. Por lo tanto, no tenemos que programarlos cada vez que los necesitemos, sino que usamos la clase **String** de Java.
- Su manipulación es mucho más fácil que en C++.

Ejemplos:

```
String s = "Buenos días";  
String t = new String ("Buenas tardes");  
String u = s + t; // concatenación  
u = u + "Buenas noches";  
  
String mensaje;  
int numero = 27;  
mensaje = "El valor es :" + numero;  
System.out.println (mensaje);
```

Strings en Java

- La clase String provee una gran variedad de métodos para trabajar con Strings. Por ejemplo, los siguientes :

```
int length ();  
// retorna el largo del String sobre el cual se  
// aplica el método  
  
String substring (int beginIndex, int endIndex);  
// retorna el substring comprendido entre las  
// posiciones beginIndex y endIndex del String  
  
char charAt (int index);  
// retorna el carácter en la posición index  
  
boolean equals (String anotherString);  
// retorna true si el String sobre el cual se aplica  
// el método es igual al String recibido por  
// parámetro
```

Strings en Java

➤ Ejemplo:

```
String miCadena = "Hola Mundo";  
int largo = miCadena.length();  
String otro = miCadena.substring(0,4);  
char firstLetter = miCadena.charAt(0);  
if (miCadena.equals(otro))  
    System.out.println ("Strings iguales");  
else  
    System.out.println ("Strings diferentes");
```

Referencia:

API - `java.lang.String`

Allí se puede encontrar información sobre la clase String y todos sus métodos

Clases en Java

Un Primer Ejemplo

- La clase `Vehiculo`. Almacenada en el archivo `vehiculo.java`. Esta clase está definida por el programador, **no** es predefinida por Java.

```
1. public class Vehiculo
2. {
3.     private String modelo;
4.     private String matricula;
5.     private double precio;
6.
7.     public Vehiculo (String mod, String mat, double pre)
8.     {
9.         modelo = mod;
10.        matricula = mat;
11.        if (pre > 0)
12.            precio = pre;
13.        else
14.            precio = 0;
15.    }
```


Clases en Java

Un Primer Ejemplo (continuación)

```
16.     public String getModelo()  
17.     { return modelo; }  
18.  
19.     public String getMatricula()  
20.     { return matricula; }  
21.  
22.     public double getPrecio()  
23.     { return precio; }  
24.  
25.     public void setMatricula (String matricula)  
26.     { this.matricula = matricula; }  
27.  
28.     public void setPrecio (double precio)  
29.     {     if (precio > 0)  
30.         this.precio = precio;  
31.     }  
32. } /* fin de la clase */
```

Clases en Java

Descripción de la Clase Vehiculo

➤ Líneas 3 – 5

```
3.  private String modelo;  
4.  private String Matricula;  
5.  private double precio;
```

- Corresponden a los **atributos** de la clase, etiquetados como **private**.
- Los atributos de una clase pueden ser tanto objetos (en este caso Strings) como valores de tipos primitivos (en este caso double)

Clases en Java

Descripción de la Clase Vehiculo (continuación)

➤ Líneas 7 – 15

```
7.  public Vehiculo (String mod, String mat, double pre)
8.  {
9.      modelo = mod;
10.     matricula = mat;
11.     if (pre > 0)
12.         precio = pre;
13.     else
14.         precio = 0;
15. }
```

- Corresponde al **constructor común** de la clase. Posee igual nombre que ella y siempre se etiqueta como **public**. Sus parámetros pueden ser tanto objetos como valores de tipos primitivos.

Clases en Java

Descripción de la Clase Vehiculo (continuación)

En una clase se pueden definir tantos constructores como se quiera.

Por ejemplo, se podrían agregar también los siguientes constructores:

```
public Vehiculo (String mod, String mat)
{
    modelo = mod;
    matricula = mat;
    precio = 15000;
}
```

```
public Vehiculo (String mat)
{
    this ("Corsa", mat);
}
```

```
public Vehiculo ()
{
    this ("AAA 403");
}
```

Lo más frecuente es incluir solamente el **constructor común** con todos los parámetros y a veces también el **constructor por defecto**.

Clases en Java

Descripción de la Clase Vehiculo (continuación)

➤ Líneas 16 – 23

```
16.    public String getModelo()  
17.    {    return modelo;    }  
18.  
19.    public String getMatricula()  
20.    {    return matricula; }  
21.  
22.    public double getPrecio()  
23.    {    return precio;    }
```

- Corresponden a los **métodos selectores** de la clase, implementados como **funciones** de Java.
- Las **funciones** en Java tienen casi la misma sintaxis que en C++. La única diferencia es que pueden contener una etiqueta (en ese caso **public**) delante del tipo de retorno. Otras posibles etiquetas son **private** y **protected**

Clases en Java

Descripción de la Clase Vehiculo (continuación)

➤ Líneas 25 – 31

```
25.    public void setMatricula (String matricula)
26.    {    this.matricula = matricula;    }
27.
28.    public void setPrecio (double precio)
29.    {        if (precio > 0)
30.            this.precio = precio;
31.    }
```

- Corresponden a los **métodos modificadores** de la clase, implementados como **procedimientos** de Java.
- Los **procedimientos** en Java tienen casi la misma sintaxis que en C++. La única diferencia es que (al igual que las funciones) pueden tener las etiquetas vistas antes delante del **void**

Clases en Java

Funciones y Procedimientos (más ejemplos)

- La siguiente **función** podría agregarse a la clase **Vehiculo**. Lo que hace es retornar un String conteniendo la información del vehículo.

```
public String toString ()
{
    String resu;

    resu = "\n Modelo :" + modelo;
    resu = resu + "\n Matricula :" + matricula;
    resu = resu + "\n Precio :" + precio;

    return resu;
}
```

Clases en Java

Funciones y Procedimientos (más ejemplos)

- El siguiente **procedimiento** podría agregarse a la clase **Vehiculo**. Lo que hace es desplegar la información del vehículo en la salida estándar.

```
public void printVehiculo ()  
{  
    String datos = toString();  
    System.out.println ("Datos del Vehiculo :");  
    System.out.println (datos);  
}
```

Pregunta: ¿Qué es conceptualmente incorrecto con respecto a este procedimiento, desde el punto de vista de la separación en capas?