

Bienvenidos

Clase 1. Java Básico



Java Básico

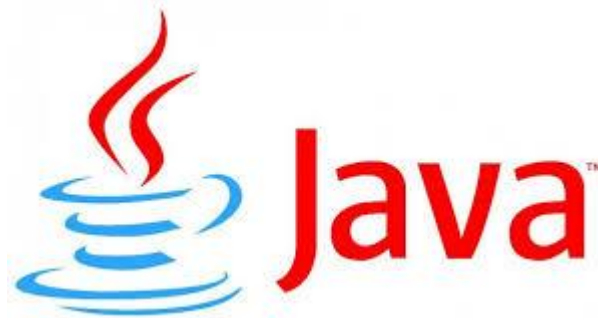
José Mauricio Jaramillo Gómez

Email Contacto

Contenido de la clase

- Presentación Docente
- Presentación del Curso
- Introducción a Java
- Que es un IDE
 - Espacio de Trabajo y Proyectos
 - Instalación de IDE de Desarrollo
 - Paquetes de clases
 - Creación de Aplicaciones
 - Creación de Proyectos
 - Configuración del Entorno
 - Opciones de Personalización del IDE
- Primer Programa
- Compilador
- Instrucciones, declaraciones y Control de Acceso
 - Identificadores, Operadores, Tipos de datos
 - Declaración de variables y métodos
 - Instrucciones, condicionales e iteraciones
 - Paquetes e importaciones
 - Constantes

Introducción a Java



Introducción a Java

Java es un lenguaje de programación con el que podemos realizar cualquier tipo de programa. En la actualidad es un lenguaje muy extendido y cada vez cobra más importancia tanto en el ámbito de Internet como en la informática en general. Fue creado inicialmente por la compañía Sun Microsystems con gran dedicación y siempre enfocado a cubrir las necesidades tecnológicas más punteras. Actualmente se encuentra en propiedad de Oracle, después que ésta adquiriera a Sun.

Una de las principales características por las que Java se ha hecho muy famoso es que es un lenguaje independiente de la plataforma. Eso quiere decir que si hacemos un programa en Java podrá funcionar en cualquier ordenador del mercado. Es una ventaja significativa para los desarrolladores de software, pues antes tenían que hacer un programa para cada sistema operativo, por ejemplo Windows, Linux, Apple, etc. Esto lo consigue porque se ha creado una Máquina virtual de Java para cada plataforma, que hace de puente entre el sistema operativo y el programa de Java y posibilita que este último se entienda perfectamente.

La independencia de plataforma es una de las razones por las que Java es interesante para Internet, ya que muchas personas deben tener acceso con ordenadores distintos. Pero no se queda ahí, Java está desarrollándose incluso para distintos tipos de dispositivos además del ordenador, como móviles, agendas y en general para cualquier cosa que se le ocurra a la industria.

Además, el hecho que Java fuera elegido por Google como lenguaje nativo para la programación de aplicaciones Android, ha provocado una nueva juventud de la tecnología y aún mayor demanda en el ámbito profesional.

Que es un IDE

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, o sea, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

Los IDE ofrecen un marco de trabajo amigable para la mayoría de los lenguajes de programación

Un IDE debe tener las siguientes características:

- Multiplataforma
- Soporte para diversos lenguajes de programación
- Integración con Sistemas de Control de Versiones
- Reconocimiento de Sintaxis
- Extensiones y Componentes para el IDE
- Integración con Framework populares
- Depurador
- Importar y Exportar proyectos
- Múltiples idiomas
- Manual de Usuarios y Ayuda

Beneficios Entorno de Desarrollo Integrado (IDE)

Un IDE (Integrated Development Environment) Entorno de Desarrollo Integrado. No es tan solo un editor de código como por ejemplo el Notepad, con el que desarrollamos nuestro primer programa en Java, el HolaMundo. Un IDE, es un conjunto de herramientas diseñadas para facilitarnos la creación/desarrollo de programas o aplicaciones. Vamos a ver algunos de ellos:

- **Editor de código:** nos facilita un editor de texto donde podremos picar nuestro código directamente.
- **Depurador:** el depurador o debugger (en inglés), nos permite localizar y solucionar errores de una manera mucho más rápida y fácil. A la acción de depurar errores se la suele conocer como debuggear.
- **Un compilador:** No tenemos que compilar el programa con los comandos `javac [nombredelarchivo.class]` y después ejecutarlo con `java [nombredelarchivo]`. Como hicimos en clases anteriores, sino que el mismo IDE se encarga automáticamente.
- **Interfaz gráfica (GUI):** Graphical User Interface nos permite en vez de mostrar los mensajes de una manera más gráfica, que no tiene nada que ver a como estamos haciendo hasta ahora (en consola).

Tipos de IDE's para Java

What's your favorite IDE
for Java Development?



 eclipse	Eclipse
	IntelliJ IDEA
 NetBeans	NetBeans
	BlueJ
	JDeveloper
 drjava	DrJava
 Android Studio	Android Studio
	Other

Instalación de IDE de Desarrollo

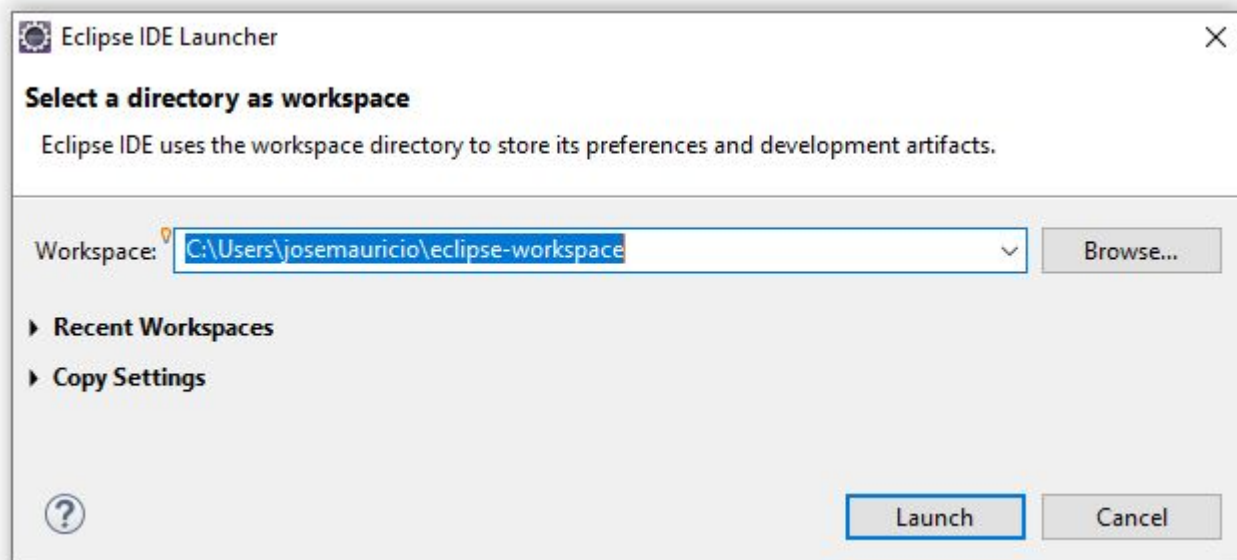
El IDE que vamos a usar en el curso es: ECLIPSE.

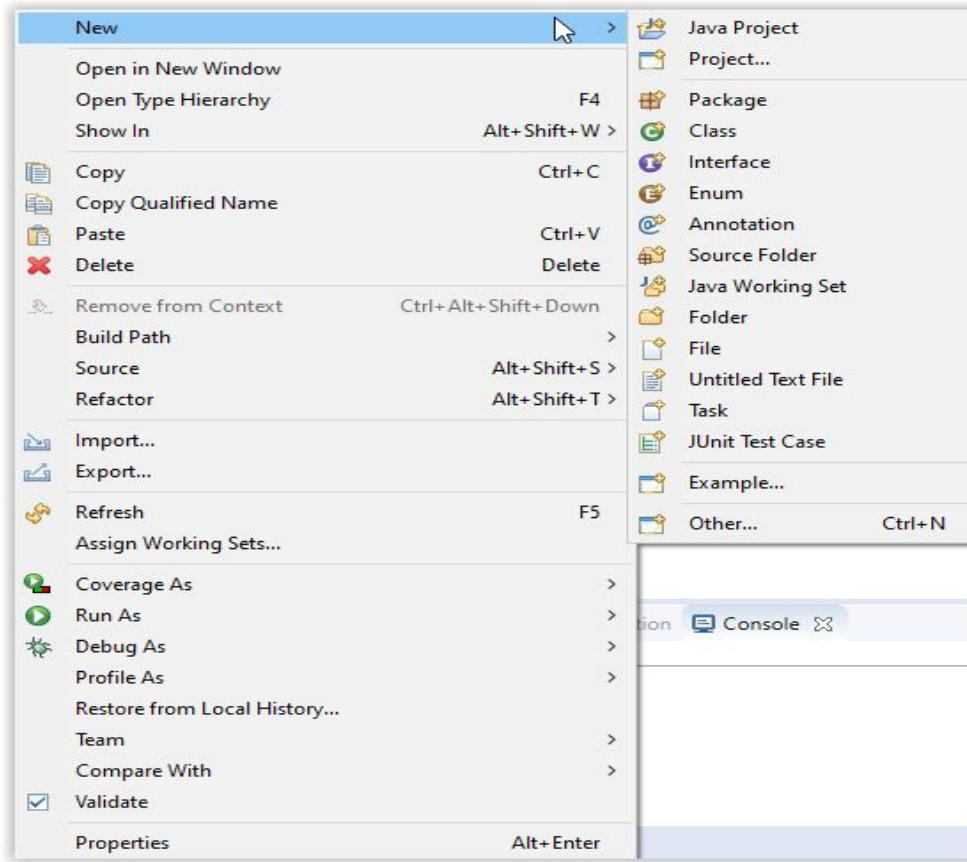
A continuación encontrarán el enlace para la descarga: <https://www.eclipse.org/downloads/packages/>

Y veremos el proceso de configuración e instalación



Espacio de Trabajo y Proyectos

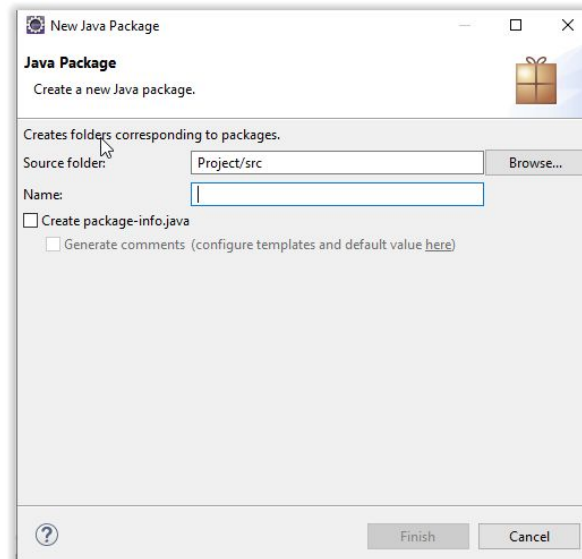




Paquetes de clases

Los paquetes son el mecanismo que usa Java **para facilitar la modularidad del código**. Un paquete puede contener una o más definiciones de interfaces y clases, distribuyéndose habitualmente como un archivo. Para utilizar los elementos de un paquete es necesario importar este en el módulo de código en curso, usando para ello la sentencia `import`.

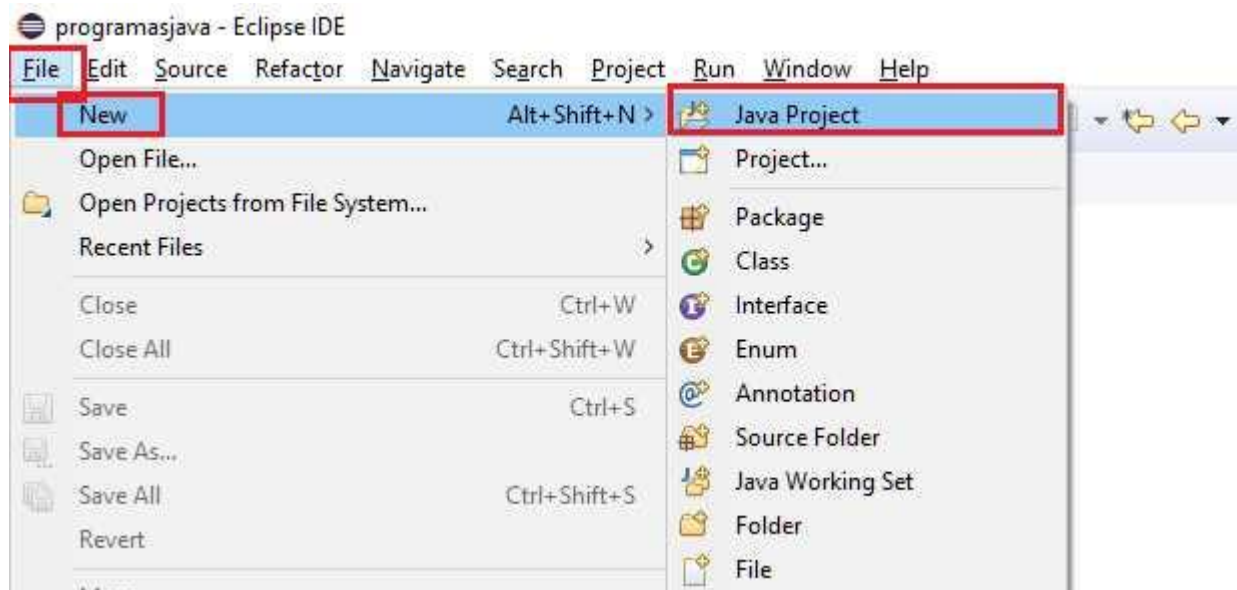
La funcionalidad de una aplicación Java se implementa habitualmente en múltiples clases, entre las que suelen existir distintas relaciones. **Las clases se agrupan en unidades de un nivel superior, los paquetes**, que actúan como **ámbitos de contención de tipos**. Cada módulo de código establece, mediante la palabra clave `package` al inicio, a qué paquete pertenece



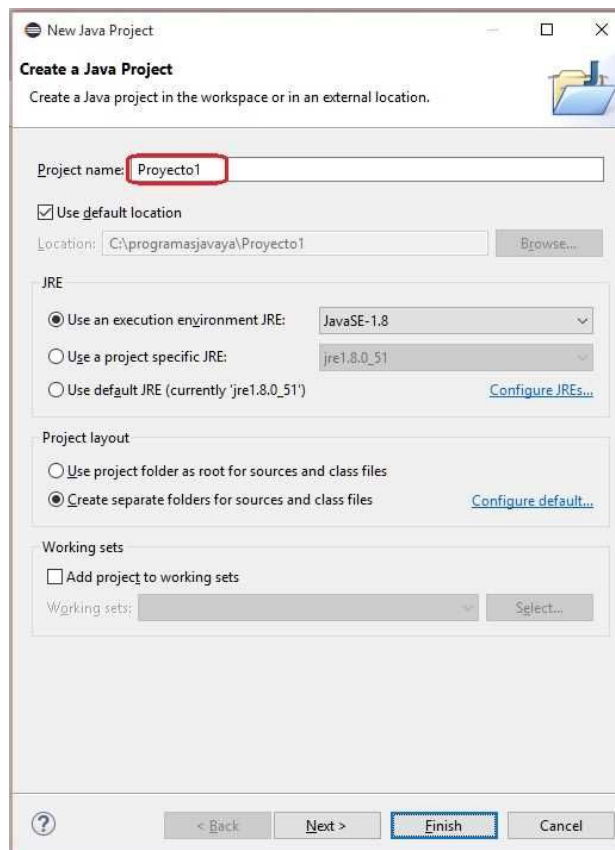
Primer Programa

El Eclipse es un entorno de trabajo profesional, por lo que en un principio puede parecer complejo el desarrollo de nuestros primeros programas.

Todo programa en Eclipse requiere la creación de un "Proyecto", para esto debemos seleccionar desde el menú de opciones:



Ahora aparece el diálogo donde debemos definir el nombre de nuestro proyecto:



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: **Proyecto1**

☒ Use default location

Location: C:\programasjava\Proyecto1 [Browse...](#)

JRE

☒ Use an execution environment JRE: JavaSE-1.8

☐ Use a project specific JRE: jre1.8.0_51

☐ Use default JRE (currently 'jre1.8.0_51') [Configure JREs...](#)

Project layout

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files [Configure default...](#)

Working sets

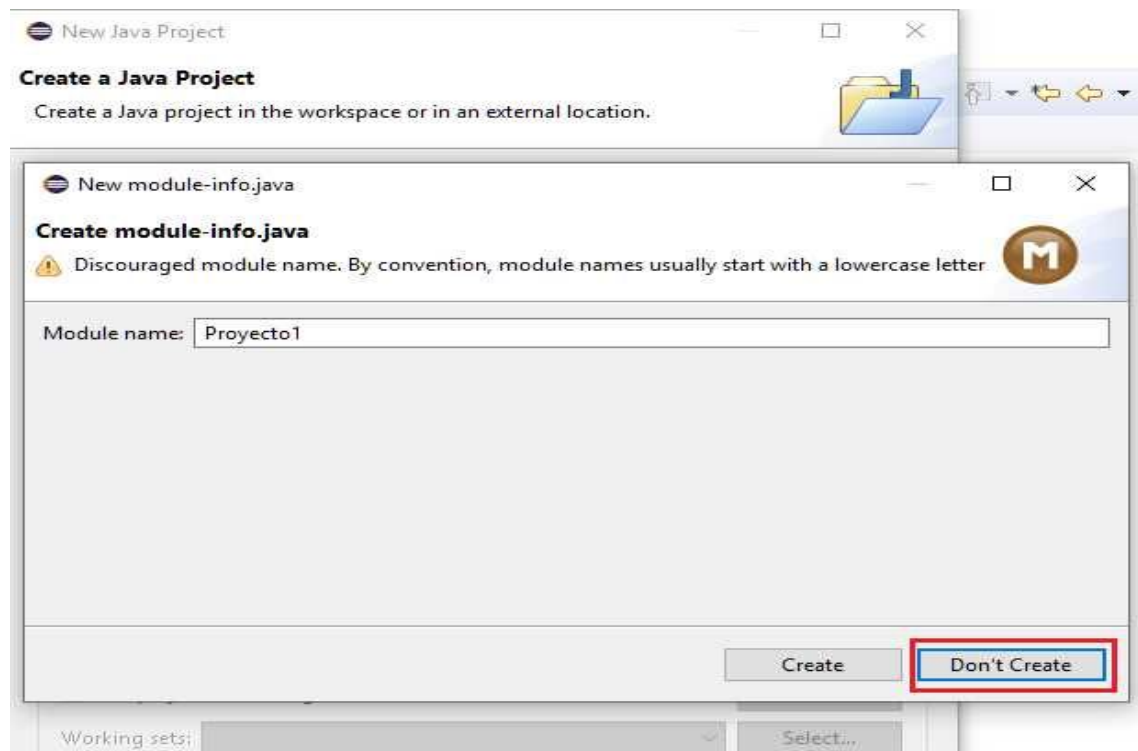
☐ Add project to working sets

Working sets: [Select...](#)

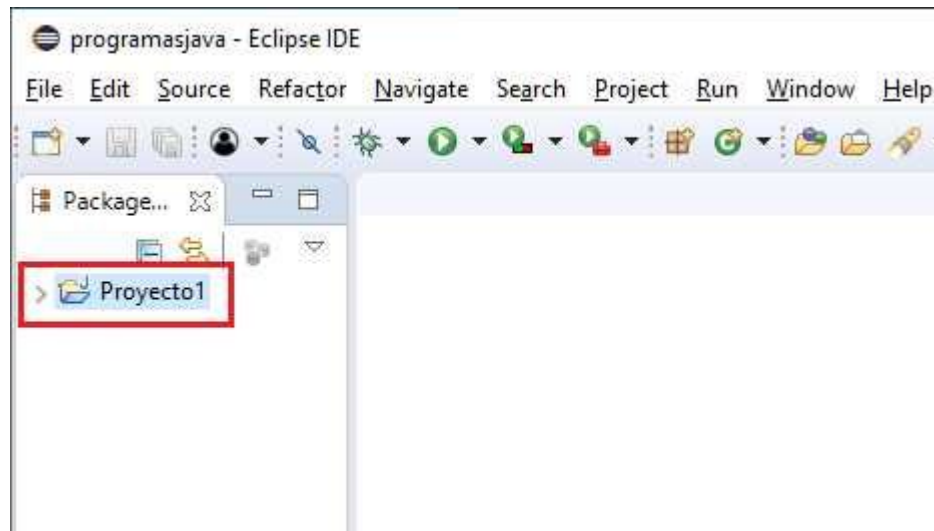
[? < Back](#) [Next >](#) **Finish** [Cancel](#)

En el campo de texto "Project Name" ingresamos como nombre: Proyecto1 y dejamos todas las otras opciones del diálogo con los valores por defecto. Presionamos el botón "Finish".

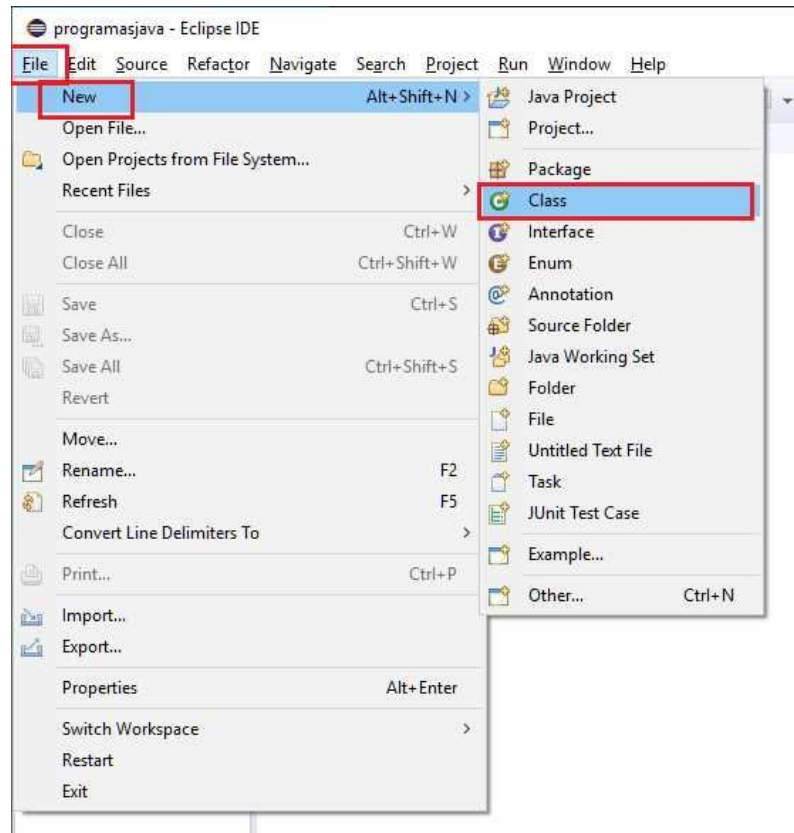
Aparece un nuevo diálogo que nos pide si queremos crear un módulo para nuestro proyecto, elegimos la opción para que no lo crea:



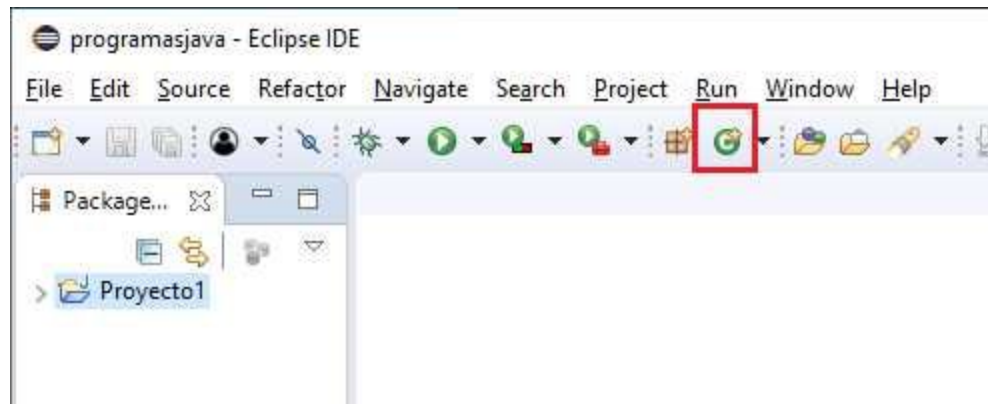
Ahora en la ventana de "Package Explorer" aparece el proyecto que acabamos de crear:



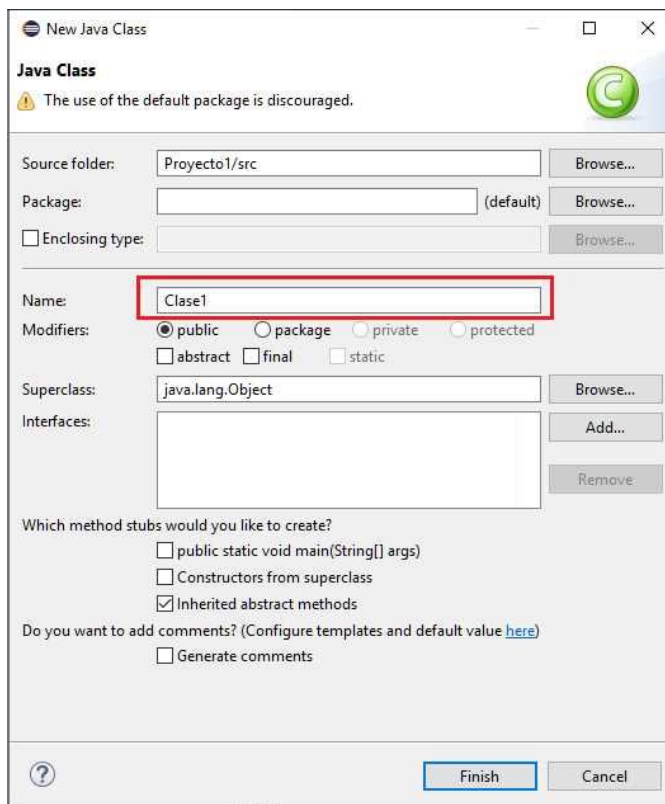
Como segundo paso veremos que todo programa en Java requiere como mínimo una clase. Para crear una clase debemos seleccionar desde el menú de opciones:



O desde la barra de íconos del Eclipse:



En el diálogo que aparece debemos definir el nombre de la clase (en nuestro primer ejemplo la llamaremos Clase1 (con mayúscula la letra C), luego veremos que es importante definir un nombre que represente al objetivo de la misma), los otros datos del diálogo los dejamos con los valores por defecto:



New Java Class

Java Class

The use of the default package is discouraged.

Source folder: Proyecto1/src Browse...

Package: (default) Browse...

Enclosing type: Browse...

Name: Clase1

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add... Remove

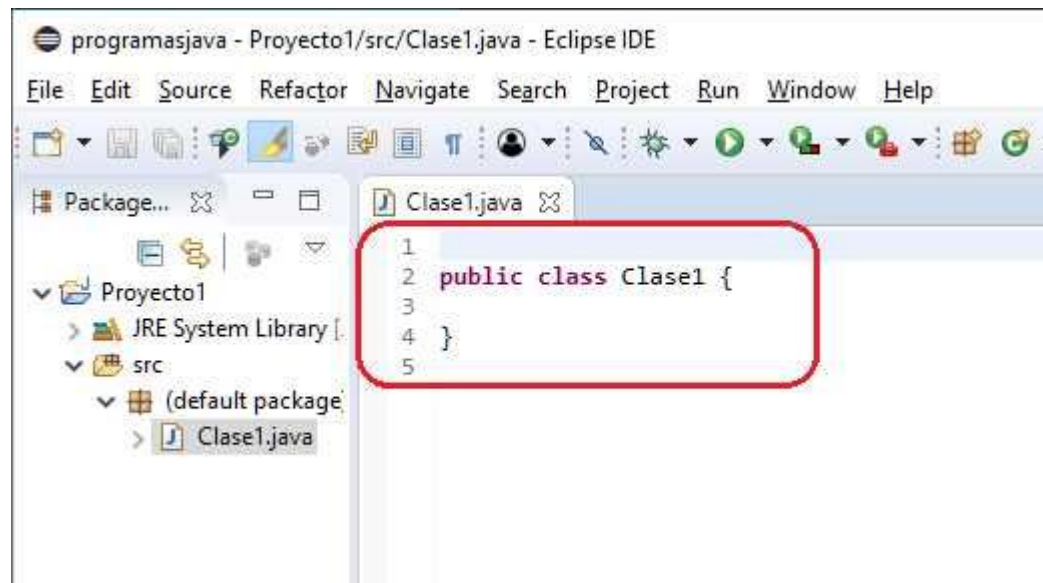
Which method stubs would you like to create?

☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

Finish Cancel

Luego de presionar el botón "Finish" tenemos el archivo donde podemos codificar nuestro primer programa:



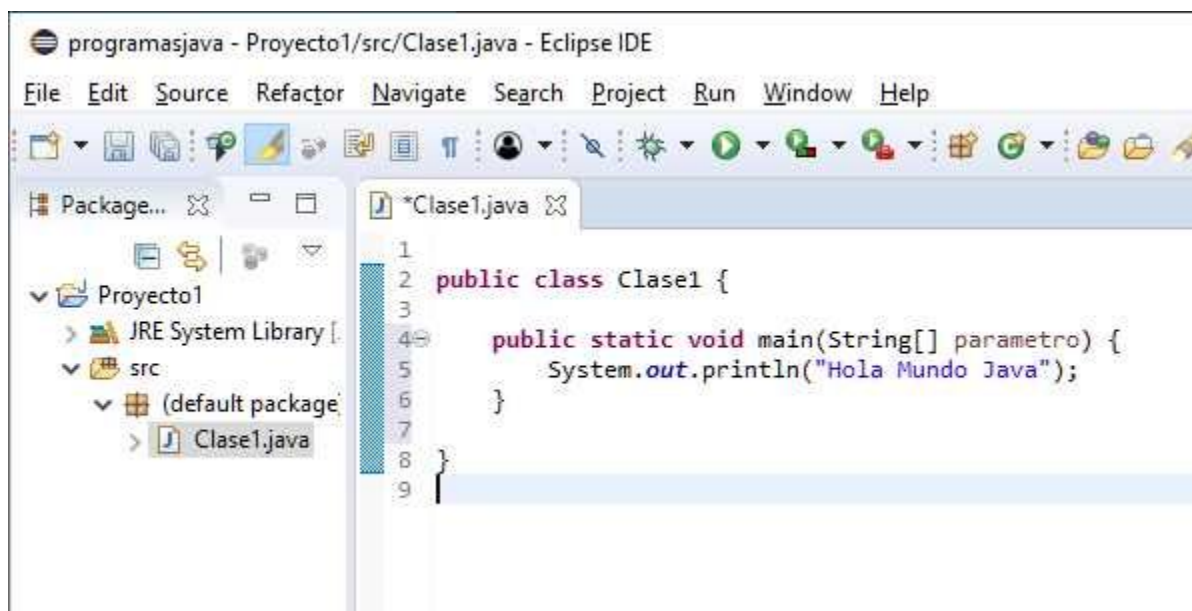
En la ventana de edición ya tenemos el esqueleto de una clase de Java que el entorno Eclipse nos creó automáticamente.

```
public class Clase1 {  
  
}
```

Todo programa en Java debe definir la función main. Esta función la debemos codificar dentro de la clase: "Clase1".
Procedemos a tipear lo siguiente:

```
public class Clase1 {  
  
    public static void main(String[] parametro) {  
        System.out.println("Hola Mundo Java");  
    }  
  
}
```

Es decir tenemos codificado en el entorno del Eclipse nuestro primer programa:

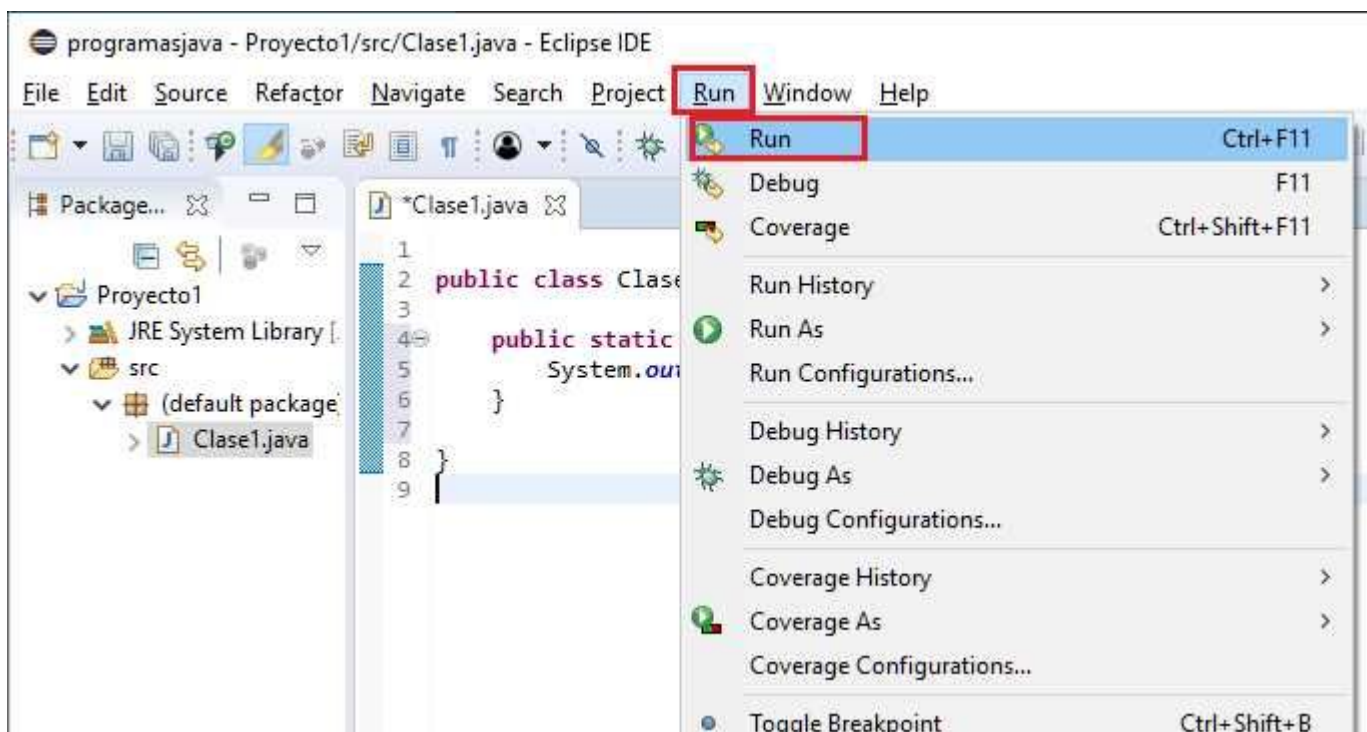


The screenshot shows the Eclipse IDE interface. The title bar reads "programasjava - Proyecto1/src/Clase1.java - Eclipse IDE". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations, editing, and running. The Package Explorer on the left shows the project structure: Proyecto1, JRE System Library, src, (default package), and Clase1.java. The main editor window displays the following Java code:

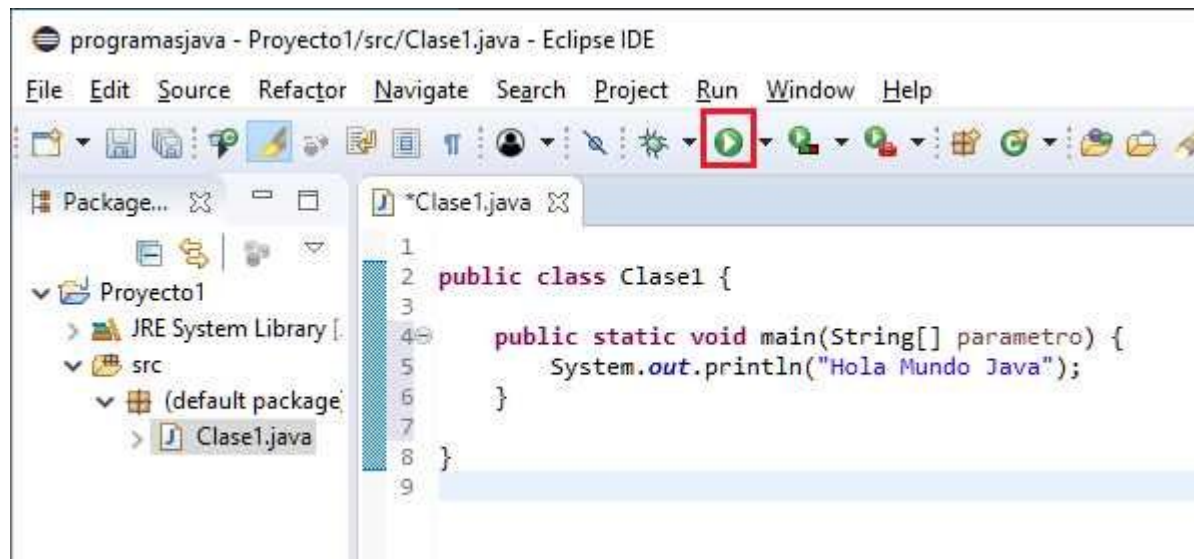
```
1
2 public class Clase1 {
3
4     public static void main(String[] parametro) {
5         System.out.println("Hola Mundo Java");
6     }
7
8 }
9
```

Compilador y Debug

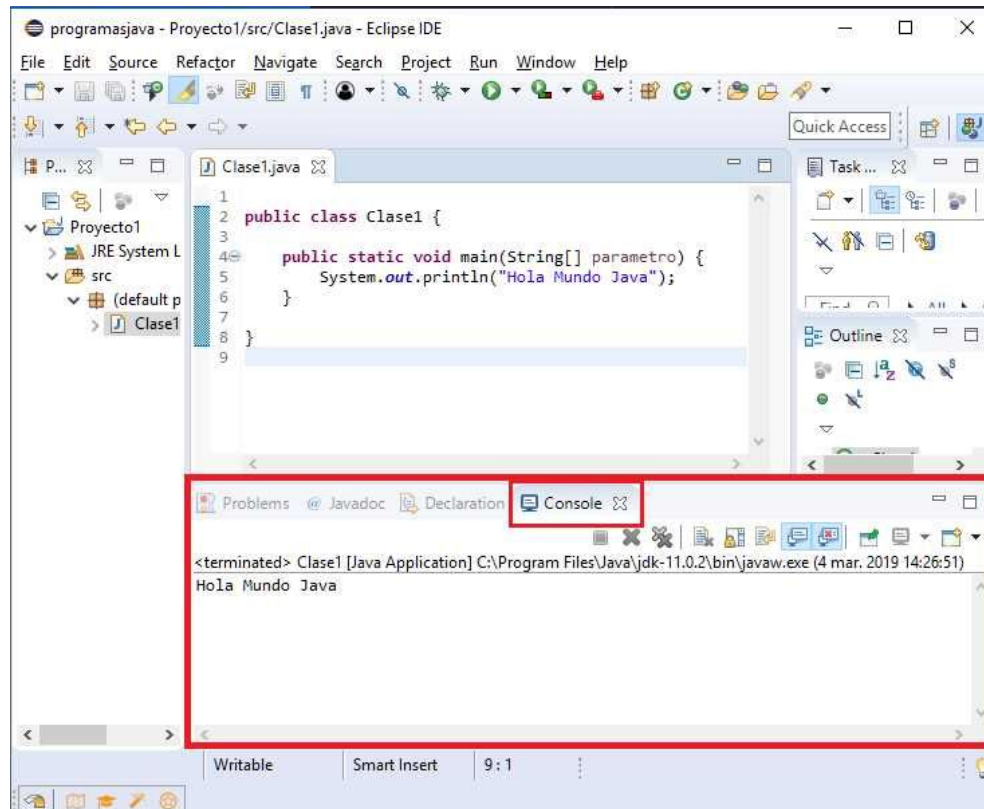
Como último paso debemos compilar y ejecutar el programa, esto lo podemos hacer desde el menú de opciones:



O desde la barra de íconos del Eclipse:



Si no hay errores de codificación debemos ver el resultado de la ejecución en una ventana del Eclipse llamada "Console" que aparece en la parte inferior (puede aparecer un diálogo pidiendo que grabemos el archivo, el cual confirmamos):



Instrucciones, declaraciones y Control de Acceso

- Private
- Public
- Static

- Void
- tipo_Dato

Paquetes e importaciones

En el lenguaje de programación Java
Podemos traer clases de paquetes o importar
librerías para su uso en el programa a crear.
Vamos a ver varios ejemplos en el proceso.

Declaración de variables y métodos

¿Que es una variable?

Una variable es un espacio de memoria (RAM) que asignamos al almacenamiento información que como su propio nombre dice (variable) puede variar para usarla en un futuro.

Una variable está formada por:

- **Definición del tipo de variable:** que usaremos (mirar la tabla de abajo y valor del campo tipo de variable).
Ejemplos: byte, short, int, long, float, char...
- **Identificarla:** (vamos ponerle un nombre) que la identifica de manera única. Un nombre cualquiera. Ejemplos: a, b, c, edad, día...
- **Asignación de un valor a la variable:** este último no es obligatorio definirlo cuando definimos una variable, pero si muy recomendable, ya que sino, no podemos mostrar su valor. Aparte de que no tiene sentido almacenar un valor en memoria sin valor.

Normas para los identificadores de las variables:

Las variables necesitan ser identificadas con identificadores y por lo tanto, tienen las mismas normas. Aquí las volvemos a repasar:

- **No utilizar ninguna palabra reservada** por el lenguaje de programación (vistas en la parte de arriba).
- **Empezar siempre por una letra , símbolo de dolar (\$) o barra baja (_).**
- **Una vez empezamos la variable con letra , símbolo de dolar (\$) barra baja (_), podemos añadirle todo lo anterior además de números.** Ejemplo: `a2_$`
- **Key Sensitive:** distingue entre mayúsculas y minúsculas.
- **Se desaconseja todo uso de espacios:** usar espacios es una mala práctica en Java debido a que en las clases funciona pero en las variables no. Por ello mejor no usarlos y así nos evitaremos futuros problemas (más abajo, os explicaremos el CamelCase).
- **Los identificadores no tienen límite de longitud:** es decir, no tienen caracteres máximos.

Tipos de Variables: Primitivas y Objetos

Tipos de Variables

Tipo
primitivos

Tipo
objetos

Tipo Primitivos

TIPO DE VARIABLE:	TIPO:	BYTES QUE OCUPA:	BITS QUE OCUPA:	RANGO DE VALORES:	EJEMPLO:
byte	Número entero (corto)	1 byte	8 bits	-128 a 127	byte a = 0;
short	Número entero (medio)	2 bytes	16 bits	-32768 a +32767	short a = 0;
int	Número entero (normal)	4 bytes	32 bits	-2.147.483.648 a +2.147.486.647	int a = 0;
long	Número entero (largo)	8 bytes	64 bits	$-9 \cdot 10^{18}$ a $+9 \cdot 10^{18} - 1$	long a = 0l; (con una l al final)
float	Numero real	4 bytes	32 bits	$-3,4 \cdot 10^{38}$ a $+3,4 \cdot 10^{38} - 1$	float a = 3.1f; (con una f al final)
double	Numero real	8 bytes	64 bits	$-1,79 \cdot 10^{308}$ a $+1,79 \cdot 10^{308} - 1$	double a = 125.2333
boolean	Lógico	1/8 bytes	1 bit	True – False	boolean a = true;
char	Carácter	2 bytes	16 bits	Cualquier carácter unicode	char a = 'z';

Tipo Objeto

- Tipos de la biblioteca estándar de Java: Pueden ser de tipo String (cadenas de texto) o por poner otro ejemplo, el famoso tipo Scanner, con el que más adelante y en lecciones posteriores, veremos cómo introducir texto por el usuario.
- Arrays: una variable que puede almacenar varios valores (del mismo tipo) en una sola variable.
- Cualquier tipo primitivo tratado como objetos (concepto que se conoce como envoltorio o wrapper)
- Las que el propio programador puede definir. Por ejemplo: coche, camión...

Declaración, Inicialización y utilización

Declaración

Definimos el tipo de variable (int, float, double, char...) para que sistema sepa que tipo que espacio debe guardar en memoria.



Inicialización

Asignamos el primer valor a la variable. Es necesario siempre inicial un valor antes de usar una variable.



Utilización

Utilizamos la variable. Hay que recordar que la variable, podrá cambiar su valor.



Declaración, Inicialización y utilización

- **Declaración:** donde definimos el nombre (identificador) y el tipo de dato (ver en

tabla). `byte dia;`

- **Iniciación:** donde definimos el valor de la variable. Es necesario para poder utilizar una variable primeramente definir (iniciacilizar) un valor en la variable antes de usarla. Se considera el no hacerlo una mala práctica que puede conllevar warnings (bombillas de advertencia) que pueden conllevar un mal

funcionamiento del programa. `dia = 3;`

- Otra opción para la **iniciación y la declaración**, es unir la declaración más la

iniciación de tal manera: `byte dia = 3;`

- **Utilización:** donde usamos la variable. `System.out.println(dia);`

Instrucciones, condicionales e iteraciones

Durante el proceso de programación tenemos procesos que nos ayudan a realizar los programas y al reuso de código.

A continuación vamos a ver las formas de programar y el uso de los condicionales.

Estructura de programación secuencial

Cuando en un problema sólo participan operaciones, entradas y salidas se la denomina una estructura secuencial.

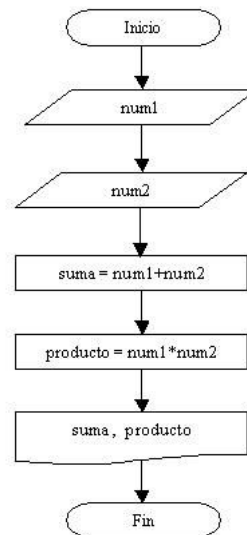
Los problemas diagramados y codificados previamente emplean solo estructuras secuenciales.

La programación requiere una práctica ininterrumpida de diagramación y codificación de problemas.

Problema:

Realizar la carga de dos números enteros por teclado e imprimir su suma y su producto.

Diagrama de flujo:



Tenemos dos entradas num1 y num2 (recordar cuáles son los nombres de variables correctas), dos operaciones: realización de la suma y del producto de los valores ingresados y dos salidas, que son los resultados de la suma y el producto de los valores ingresados. En el símbolo de impresión podemos indicar una o más salidas, eso queda a criterio del programador, lo mismo para indicar las entradas por teclado.

```
import java.util.Scanner;

public class SumaProductoNumeros {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int num1,num2,suma,producto;
        System.out.print("Ingrese primer valor:");
        num1=teclado.nextInt();
        System.out.print("Ingrese segundo valor");
        num2=teclado.nextInt();
        suma=num1 + num2;
        producto=num1 * num2;
        System.out.print("La suma de los dos valores es:");
        System.out.println(suma);
        System.out.print("El producto de los dos valores es:");
        System.out.println(producto);
    }
}
```

Estructuras condicionales simples y compuestas

No todos los problemas pueden resolverse empleando estructuras secuenciales. Cuando hay que tomar una decisión aparecen las estructuras condicionales.

En nuestra vida diaria se nos presentan situaciones donde debemos decidir.

¿Elijo la carrera A o la carrera B?

¿Me pongo este pantalón?

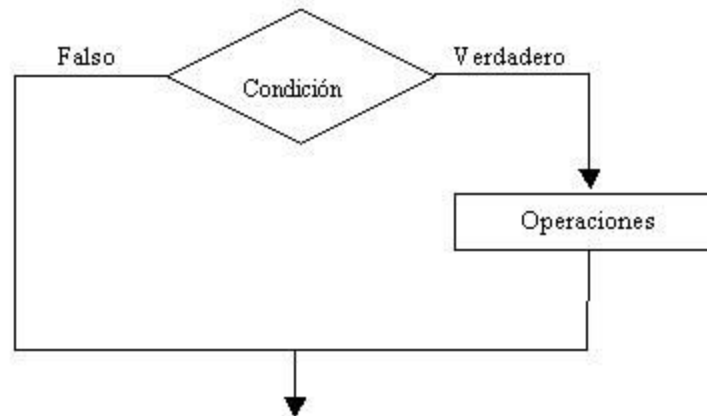
Para ir al trabajo, ¿elijo el camino A o el camino B?

Al cursar una carrera, ¿elijo el turno mañana, tarde o noche?

Por supuesto que en un problema se combinan estructuras secuenciales y condicionales.

Estructura condicional simple.

Cuando se presenta la elección tenemos la opción de realizar una actividad o no realizar ninguna.
Representación gráfica:



Podemos observar: El rombo representa la condición. Hay dos opciones que se pueden tomar. Si la condición da verdadera se sigue el camino del verdadero, o sea el de la derecha, si la condición da falsa se sigue el camino de la izquierda.

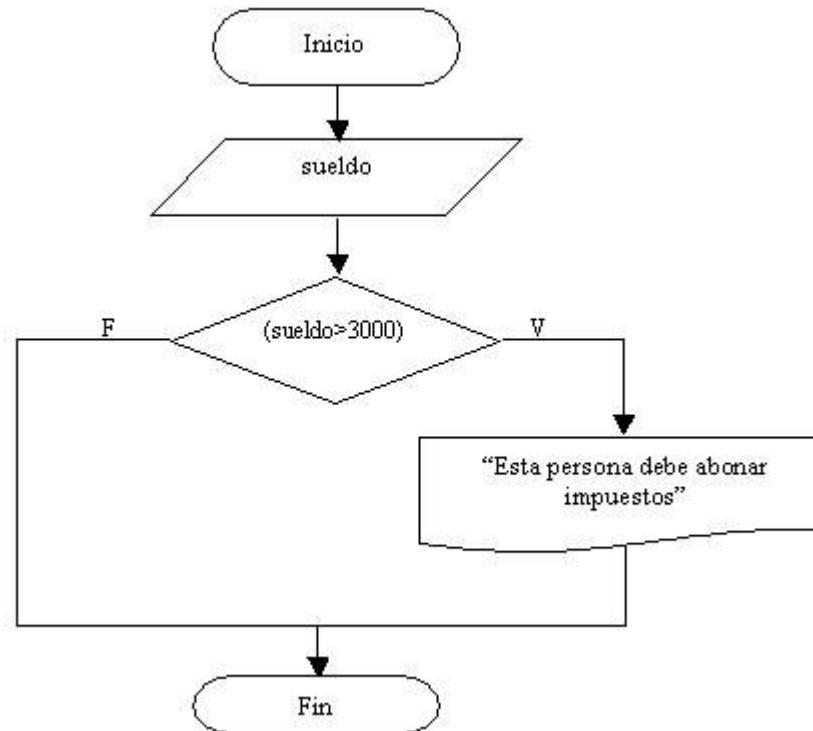
Se trata de una estructura **CONDICIONAL SIMPLE** porque por el camino del verdadero hay actividades y por el camino del falso no hay actividades.

Por el camino del verdadero pueden existir varias operaciones, entradas y salidas, inclusive ya veremos que puede haber otras estructuras condicionales.

Problema:

Ingresar el sueldo de una persona, si supera los 3000 pesos mostrar un mensaje en pantalla indicando que debe abonar impuestos.

Diagrama de flujo:



Podemos observar lo siguiente: Siempre se hace la carga del sueldo, pero si el sueldo que ingresamos supera 3000 pesos se mostrará por pantalla el mensaje "Esta persona debe abonar impuestos", en caso que la persona cobre 3000 o menos no aparece nada por pantalla.

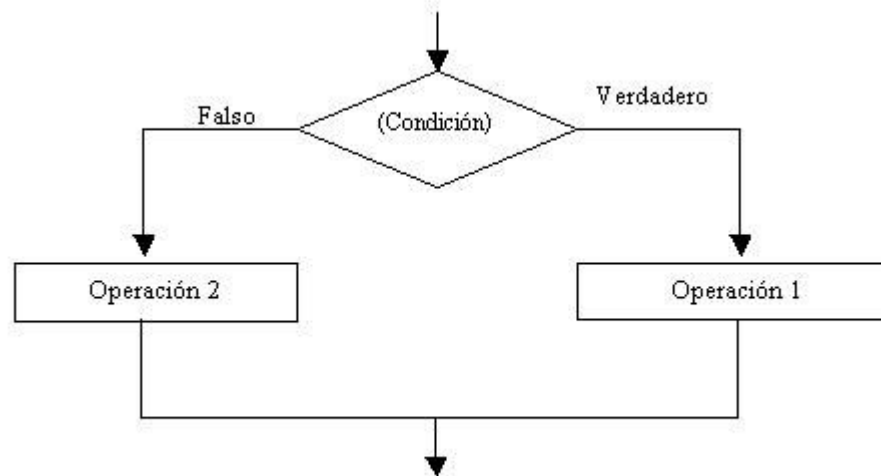
```
import java.util.Scanner;

public class EstructuraCondicionalSimple1 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        float sueldo;
        System.out.print("Ingrese el sueldo:");
        sueldo=teclado.nextFloat();
        if (sueldo>3000) {
            System.out.println("Esta persona debe abonar impuestos");
        }
    }
}
```

Estructura condicional compuesta.

Cuando se presenta la elección tenemos la opción de realizar una actividad u otra. Es decir tenemos actividades por el verdadero y por el falso de la condición. Lo más importante que hay que tener en cuenta que se realizan las actividades de la rama del verdadero o las del falso, NUNCA se realizan las actividades de las dos ramas.

Representación gráfica:

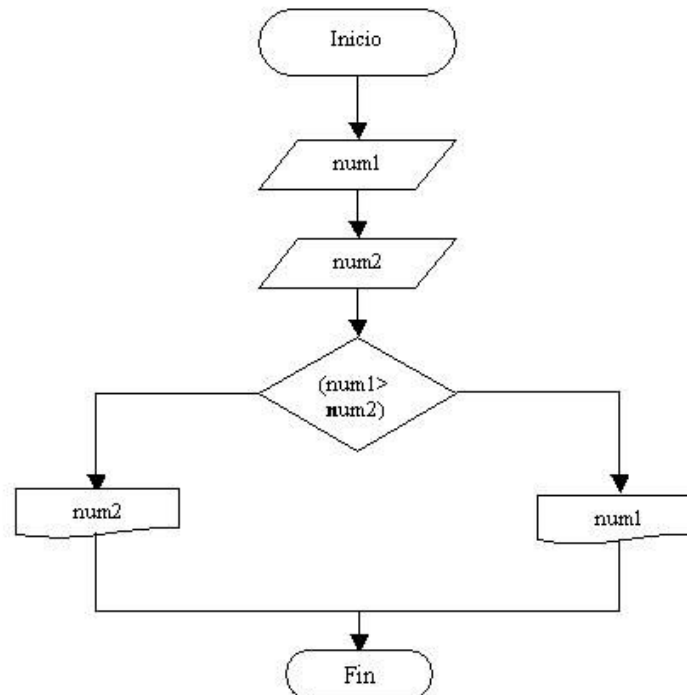


En una estructura condicional compuesta tenemos entradas, salidas, operaciones, tanto por la rama del verdadero como por la rama del falso.

Problema:

Realizar un programa que solicite ingresar dos números distintos y muestre por pantalla el mayor de ellos.

Diagrama de flujo:



Se hace la entrada de num1 y num2 por teclado. Para saber cuál variable tiene un valor mayor preguntamos si el contenido de num1 es mayor (>) que el contenido de num2, si la respuesta es verdadera vamos por la rama de la derecha e imprimimos num1, en caso que la condición sea falsa vamos por la rama de la izquierda (Falsa) e imprimimos num2.

Como podemos observar nunca se imprimen num1 y num2 simultáneamente.

Estamos en presencia de una ESTRUCTURA CONDICIONAL COMPUESTA ya que tenemos actividades por la rama del verdadero y del falso.

```
import java.util.Scanner;

public class EstructuraCondicionalCompuesta1 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int num1,num2;
        System.out.print("Ingrese primer valor:");
        num1=teclado.nextInt();
        System.out.print("Ingrese segundo valor:");
        num2=teclado.nextInt();
        if (num1>num2) {
            System.out.print(num1);
        } else {
            System.out.print(num2);
        }
    }
}
```

Estructura repetitiva while

Hasta ahora hemos empleado estructuras SECUENCIALES y CONDICIONALES. Existe otro tipo de estructuras tan importantes como las anteriores que son las estructuras REPETITIVAS.

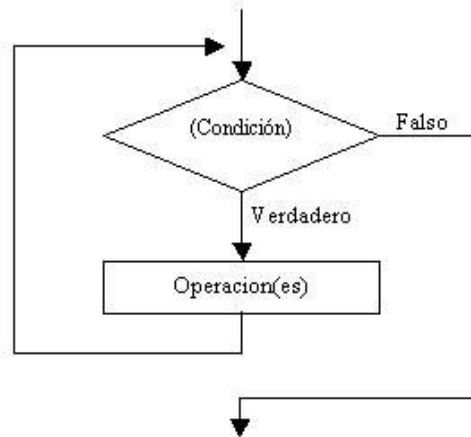
Una estructura repetitiva permite ejecutar una instrucción o un conjunto de instrucciones varias veces.

Una ejecución repetitiva de sentencias se caracteriza por:

- La o las sentencias que se repiten.
- El test o prueba de condición antes de cada repetición, que motivará que se repitan o no las sentencias.

Estructura repetitiva while

Representación gráfica de la estructura while:



No debemos confundir la representación gráfica de la estructura repetitiva while (Mientras) con la estructura condicional if (Si)

Funcionamiento: En primer lugar se verifica la condición, si la misma resulta verdadera se ejecutan las operaciones que indicamos por la rama del Verdadero.

A la rama del verdadero la graficamos en la parte inferior de la condición. Una línea al final del bloque de repetición la conecta con la parte superior de la estructura repetitiva.

En caso que la condición sea Falsa continúa por la rama del Falso y sale de la estructura repetitiva para continuar con la ejecución del algoritmo.

El bloque se repite MIENTRAS la condición sea Verdadera.

Importante: Si la condición siempre retorna verdadero estamos en presencia de un ciclo repetitivo infinito. Dicha situación es un error de programación, nunca finalizará el programa.

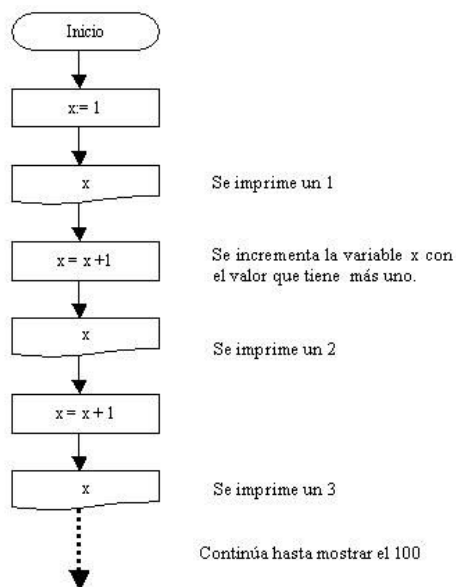
Problema 1:

Realizar un programa que imprima en pantalla los números del 1 al 100.

Sin conocer las estructuras repetitivas podemos resolver el problema empleando una estructura secuencial.

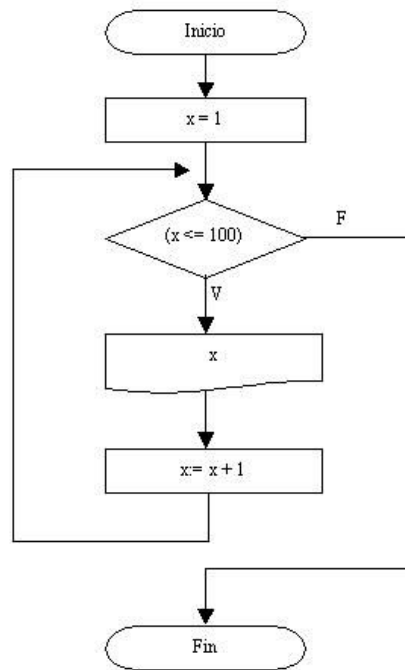
Inicializamos una variable con el valor 1, luego imprimimos la variable, incrementamos nuevamente la variable y así sucesivamente.

Diagrama de flujo:



Si continuamos con el diagrama no nos alcanzaría las próximas 5 páginas para finalizarlo. Emplear una estructura secuencial para resolver este problema produce un diagrama de flujo y un programa en Java muy largo.

Ahora veamos la solución empleando una estructura repetitiva while:



Programa

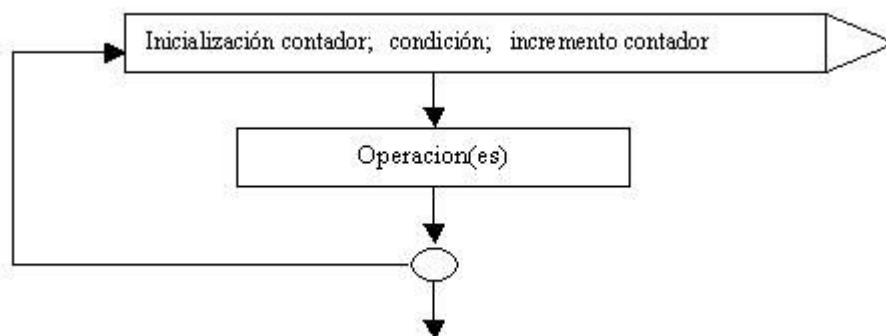
```
public class EstructuraRepetitivaWhile1 {  
    public static void main(String[] ar) {  
        int x;  
        x=1;  
        while (x<=100) {  
            System.out.print(x);  
            System.out.print(" - ");  
            x = x + 1;  
        }  
    }  
}
```

Estructura repetitiva for

Cualquier problema que requiera una estructura repetitiva se puede resolver empleando la estructura while. Pero hay otra estructura repetitiva cuyo planteo es más sencillo en ciertas situaciones.

En general, la estructura for se usa en aquellas situaciones en las cuales CONOCEMOS la cantidad de veces que queremos que se ejecute el bloque de instrucciones. Ejemplo: cargar 10 números, ingresar 5 notas de alumnos, etc. Conocemos de antemano la cantidad de veces que queremos que el bloque se repita. Veremos, sin embargo, que en el lenguaje Java la estructura for puede usarse en cualquier situación repetitiva, porque en última instancia no es otra cosa que una estructura while generalizada.

Representación gráfica:

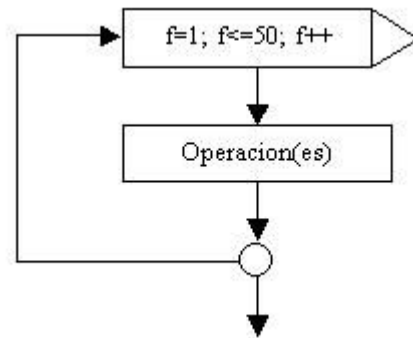


En su forma más típica y básica, esta estructura requiere una variable entera que cumple la función de un CONTADOR de vueltas. En la sección indicada como "inicialización contador", se suele colocar el nombre de la variable que hará de contador, asignándole a dicha variable un valor inicial. En la sección de "condición" se coloca la condición que deberá ser verdadera para que el ciclo continúe (en caso de un falso, el ciclo se detendrá). Y finalmente, en la sección de "incremento contador" se coloca una instrucción que permite modificar el valor de la variable que hace de contador (para permitir que alguna vez la condición sea falsa)

Cuando el ciclo comienza, antes de dar la primera vuelta, la variable del for toma el valor indicado en la sección de "inicialización contador". Inmediatamente se verifica, en forma automática, si la condición es verdadera. En caso de serlo se ejecuta el bloque de operaciones del ciclo, y al finalizar el mismo se ejecuta la instrucción que se haya colocado en la tercer sección.

Seguidamente, se vuelve a controlar el valor de la condición, y así prosigue hasta que dicha condición entregue un falso.

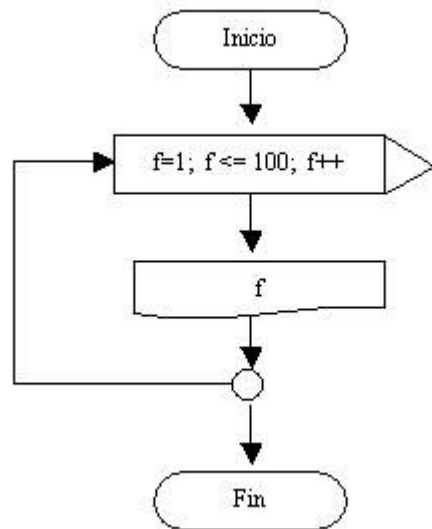
Si conocemos la cantidad de veces que se repite el bloque es muy sencillo emplear un for, por ejemplo si queremos que se repita 50 veces el bloque de instrucciones puede hacerse así:



Problema 1:

Realizar un programa que imprima en pantalla los números del 1 al 100.

Diagrama de flujo:



Podemos observar y comparar con el problema realizado con el while. Con la estructura while el CONTADOR x sirve para contar las vueltas. Con el for el CONTADOR f cumple dicha función.

Inicialmente f vale 1 y como no es superior a 100 se ejecuta el bloque, imprimimos el contenido de f, al finalizar el bloque repetitivo se incrementa la variable f en 1, como 2 no es superior a 100 se repite el bloque de instrucciones.

Cuando la variable del for llega a 101 sale de la estructura repetitiva y continúa la ejecución del algoritmo que se indica después del círculo.

La variable f (o como sea que se decida llamarla) debe estar definida como una variable más.

Programa

```
public class EstructuraRepetitivaFor1 {  
    public static void main(String[] ar) {  
        int f;  
        for(f=1;f<=100;f++) {  
            System.out.print(f);  
            System.out.print("-");  
        }  
    }  
}
```

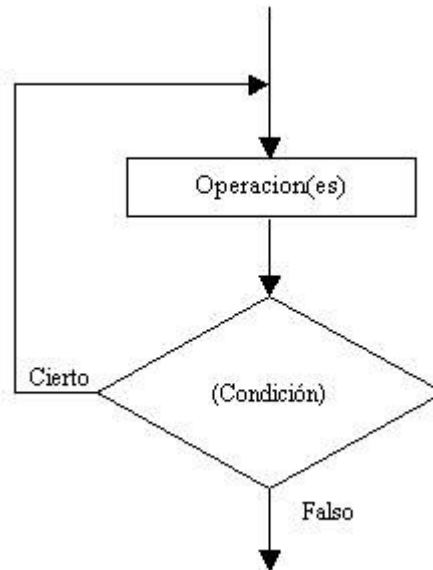
Estructura repetitiva do while

La estructura do while es otra estructura repetitiva, la cual ejecuta al menos una vez su bloque repetitivo, a diferencia del while o del for que podían no ejecutar el bloque.

Esta estructura repetitiva se utiliza cuando conocemos de antemano que por lo menos una vez se ejecutará el bloque repetitivo.

La condición de la estructura está abajo del bloque a repetir, a diferencia del while o del for que está en la parte superior.

Representación gráfica:



El bloque de operaciones se repite MIENTRAS que la condición sea Verdadera.

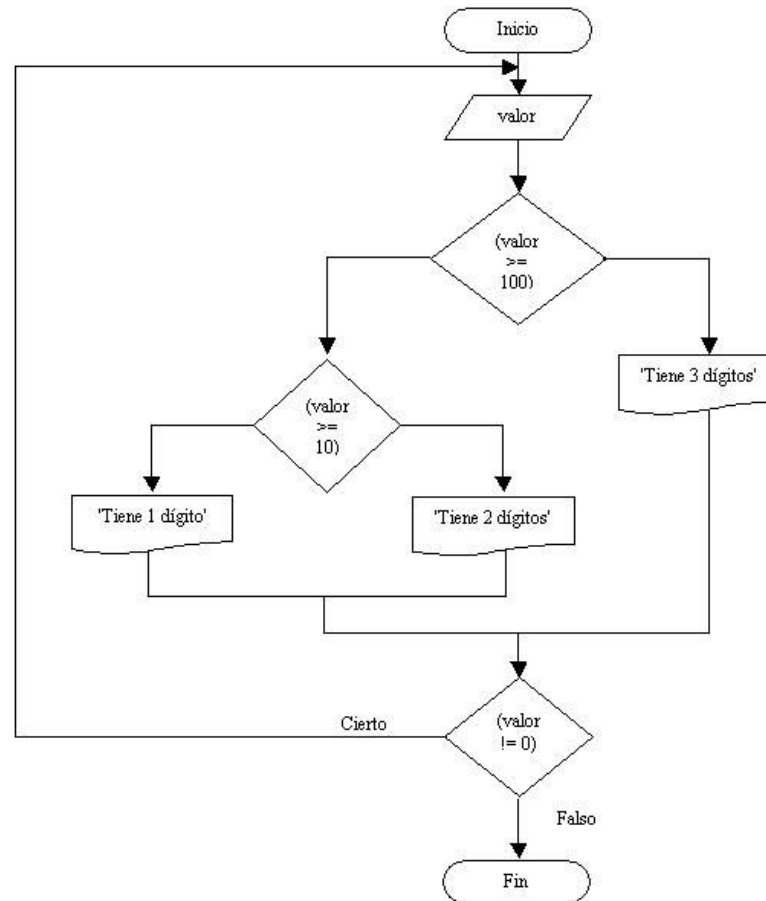
Si la condición retorna Falso el ciclo se detiene. En Java, todos los ciclos repiten por verdadero y cortan por falso.

Es importante analizar y ver que las operaciones se ejecutan como mínimo una vez.

Problema 1:

Escribir un programa que solicite la carga de un número entre 0 y 999, y nos muestre un mensaje de cuantos dígitos tiene el mismo. Finalizar el programa cuando se cargue el valor 0.

Diagrama de flujo:



No hay que confundir los rombos de las estructuras condicionales con los de las estructuras repetitivas do while. En este problema por lo menos se carga un valor. Si se carga un valor mayor o igual a 100 se trata de un número de tres cifras, si es mayor o igual a 10 se trata de un valor de dos dígitos, en caso contrario se trata de un valor de un dígito. Este bloque se repite hasta que se ingresa en la variable valor el número 0 con lo que la condición de la estructura do while retorna falso y sale del bloque repetitivo finalizando el programa.

Programa

```
import java.util.Scanner;

public class EstructuraRepetitivaDoWhile1 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int valor;
        do {
            System.out.print("Ingrese un valor entre 0 y 999 (0 finaliza):");
            valor=teclado.nextInt();
            if (valor>=100) {
                System.out.println("Tiene 3 dígitos.");
            } else {
                if (valor>=10) {
                    System.out.println("Tiene 2 dígitos.");
                } else {
                    System.out.println("Tiene 1 dígito.");
                }
            }
        } while (valor!=0);
    }
}
```

Constantes

Las constantes son un tipo de variables, con unas características especiales. A diferencia de las variables (cuyo valor podía variar), el valor de una constante, tal como dice su nombre, será el definitivo, no variará (todo el rato el mismo). Por ello, con constante nos referimos que una vez hacemos realizado una declaración, no se podrá modificar dicho valor.

La sintaxis de una constante es idéntica a la de las variables en Java, con la diferencia que debemos añadir la palabra **final** (sin mayúsculas) al principio de todo.

Estructura de una constante:

Su estructura está compuesta por:

`final [tipodedato] [nombredelaconstante]; //Declaración`

`final [tipodedato] [nombredelaconstante] = 10; //Declaración + iniciación`

```
1  public class Constantes
2  {
3      public static void main(String[] args)
4      {
5          final byte a = 1;
6          System.out.println(a);
7          final short b = 2;
8          System.out.println(b);
9          final int c = 3;
10         System.out.println(c);
11         final long d = 4L; //OJO acabado en L
12         System.out.println(d);
13         final float e = 1.253F; //OJO acabada en F y con los decimales en . NO EN COMA!
14         System.out.println(e);
15         final double f = 2.25132514;
16         System.out.println(f);
17         final boolean g = true;
18         System.out.println(g);
19         final char h = 'C'; //OJO entre comillas simples
20         System.out.println(h);
21     }
22 }
```

Gracias...