

## Entornos de Desarrollo (Interface Development Environment, IDE)

Java no cuenta con un entorno de desarrollo propio, por esa razón, se puede utilizar desde un bloc de notas hasta entornos de desarrollo avanzados como NetBeans. Netbeans es un poderoso entorno de desarrollo que permite desarrollar aplicaciones complejas con interacción web, UML, base de datos, aplicaciones para telefonía móvil e inclusive Inteligencia Artificial (IA). Debido a su performance, se abordará la programación de aplicaciones en este IDE.

En él podemos realizar todas las tareas asociadas a la programación – Editar el código – Compilarlo – Ejecutarlo – Depurarlo.

## Historia de NetBeans

NetBeans comenzó en 1996 como un proyecto estudiantil en República Checa (originalmente llamado Xelfi), bajo la tutoría de la Facultad de Matemáticas y Física en la Universidad Carolina en Praga. La meta era escribir un entorno de desarrollo integrado (IDE) para Java parecida a la de Delphi. Xelfi fue el primer entorno de desarrollo integrado escrito en Java, con su primer prerelease en 1997. El proyecto atrajo suficiente interés, por lo que los estudiantes, después de graduarse, decidieron que lo podían convertir en un proyecto comercial. Prestando espacios web de amigos y familiares, formaron una compañía alrededor de esto. Casi todos ellos siguen trabajando en NetBeans.

Tiempo después, fueron contactados por Roman Stanek, un empresario que ya había estado relacionado con varias iniciativas en la República Checa. Estaba buscando una buena idea en la que invertir, y encontró en Xelfi una buena oportunidad. Así, tras una reunión, el negocio surgió. El plan original era desarrollar unos componentes JavaBeans para redes. Jarda Tulach, quien diseñó la arquitectura básica de la IDE, propuso la idea de llamarlo NetBeans, a fin de describir este propósito. Cuando las especificaciones de los Enterprise JavaBeans salieron, decidieron trabajar con este estándar, ya que no tenía sentido competir contra él, sin embargo permaneció el nombre de NetBeans.

## ¿Por qué usarlo?

- Simplifica alguna de las tareas que, sobretudo en proyectos grandes, son tediosas
- Nos asiste (parcialmente) en la escritura de código, aunque no nos libera de aprender el lenguaje de programación
- Nos ayuda en la navegación de las clases predefinidas en la plataforma (miles)
- Aunque puede ser costoso su aprendizaje, los beneficios superan las dificultades

## Comentarios

En Java existen tres tipos de comentarios:

- Comentarios de una sola línea como en C++  

```
// Esta es una línea comentada.
```
- Comentarios de bloques como en C.

```
/* Aquí empieza el bloque comentado  
y aquí acaba */
```

- **Comentarios de documentación.**

```
/** Los comentarios de documentación se comentan de este modo */
```

## Palabras reservadas

El conjunto de palabras reservadas en Java es el que aparece en la tabla Tabla

abstract	continue	for	new	switch
boolean	default	goto	null	synchronized
break	do	if	package	this
byte	double	implements	private	threadsafe
byvalue	else	import	protected	throw
case	extends	instanceof	public	transient
catch	false	int	return	true
char	final	interface	short	try
class	finally	long	static	void
const	float	native	super	while

## Separadores

En Java existen seis separadores distintos. A continuación se muestra el uso de cada uno de ellos.

- Los paréntesis ():
  - Delimitan listas de parámetros.
  - Modifican la precedencia de una expresión.
  - Delimitan condiciones.
  - Indican el tipo en las coerciones.
- Las llaves {}:
  - Definen bloques de código.
  - Delimitan las lista de valores iniciales de los arrays.
- Los corchetes []:
  - Declaran vectores y permiten acceder a sus elementos.
- El punto y coma «;»:
  - Terminan instrucciones.
- La coma «,»:
  - Separan identificadores en declaraciones.
  - Encadenan expresiones.
- El punto «.»:
  - Acceden a los atributos y métodos de una clase.
  - Acceden a un subpaquete de un paquete.

# Que es una Clase?:

La unidad fundamental de programación en Java es la clase. Un programa Java está formado por un conjunto de clases. La programación orientada a objetos (POO) abstrae las entidades del mundo real como objetos y las relaciones entre ellos como paso de mensajes. Los objetos son instancias o ejemplares de una clase o plantilla y poseen como características atributos (valores) y métodos (acciones).

Entonces una clase es una “plantilla” que describe un conjunto de objetos con atributos y comportamiento similares.

Un programa Java en ejecución crea y manipula (mediante llamadas a métodos) objetos concretos (ejemplares o instancias).

Por convención, se declaran primero las variables (atributos) miembro de la clase y luego las declaraciones e implementaciones de métodos.

## Objeto:

Es un elemento declarado de un tipo de clase. Se conoce también como una instancia de clase. La sintaxis para definir un objeto es

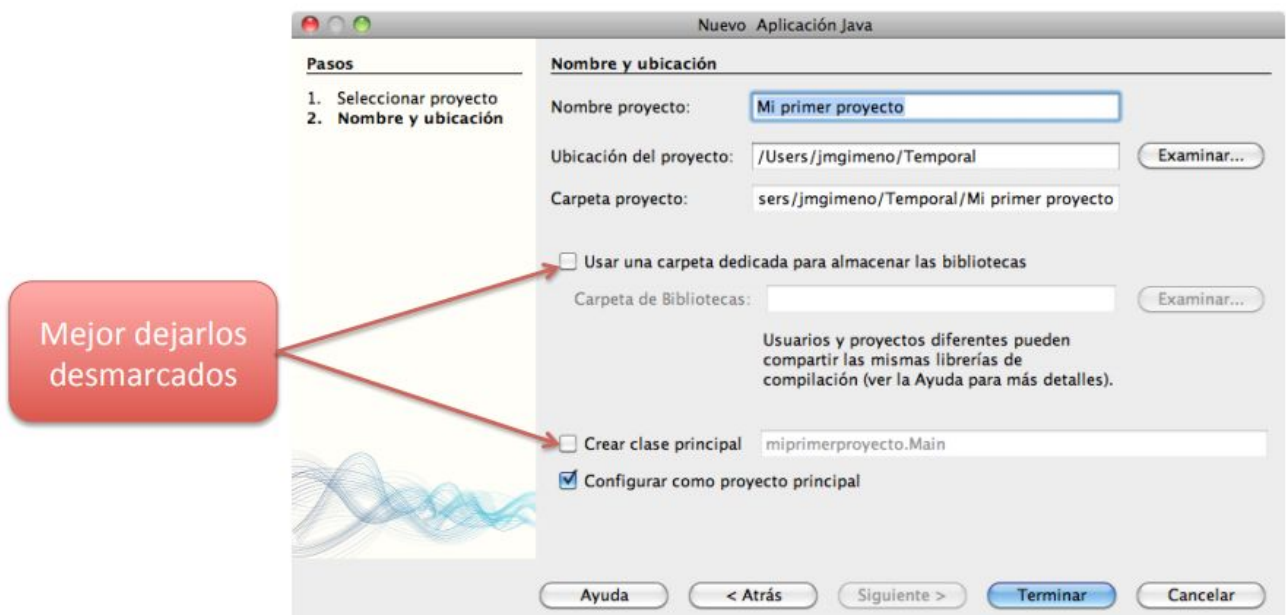
```
nombreDeClase nombreObjeto;  
nombreDeClase otroObjeto;  
Ejemplos: Transporte auto;
```

## Concepto de proyecto

- Netbeans no trabaja a nivel de archivo sino a nivel de proyecto
- Un proyecto incluye todos los recursos necesarios para construir un programa:
  - Archivos con el código
  - Bibliotecas externas (p.e. ACM Task Force)
  - Imágenes, sonidos, etc.
- Físicamente un proyecto Netbeans no es más que un directorio con una organización especial.

## Creamos un proyecto Nuevo:

Para crear un nuevo Proyecto, seleccionamos la Categoría Java y el Proyecto Java.



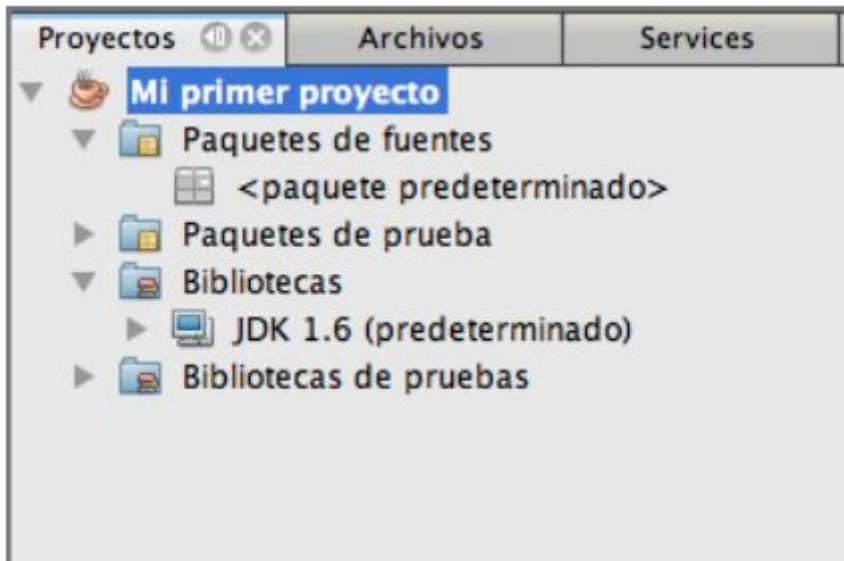
## Estructura del proyecto

Después de pulsar **Terminar** se crea un nuevo proyecto en el área de proyecto.

Este proyecto no incluye aún

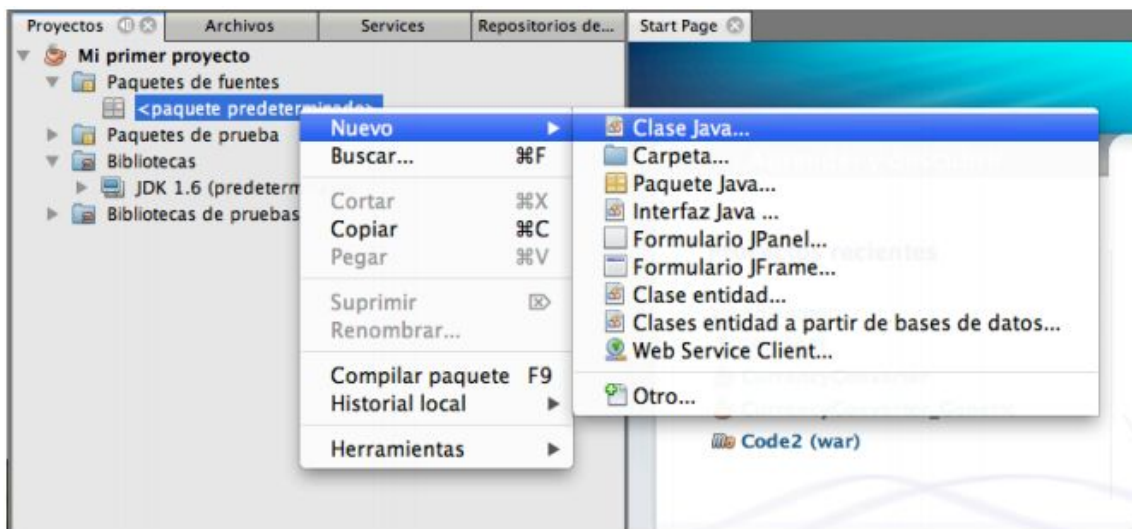
- Ninguna clase
- Biblioteca

Empecemos creando una clase:

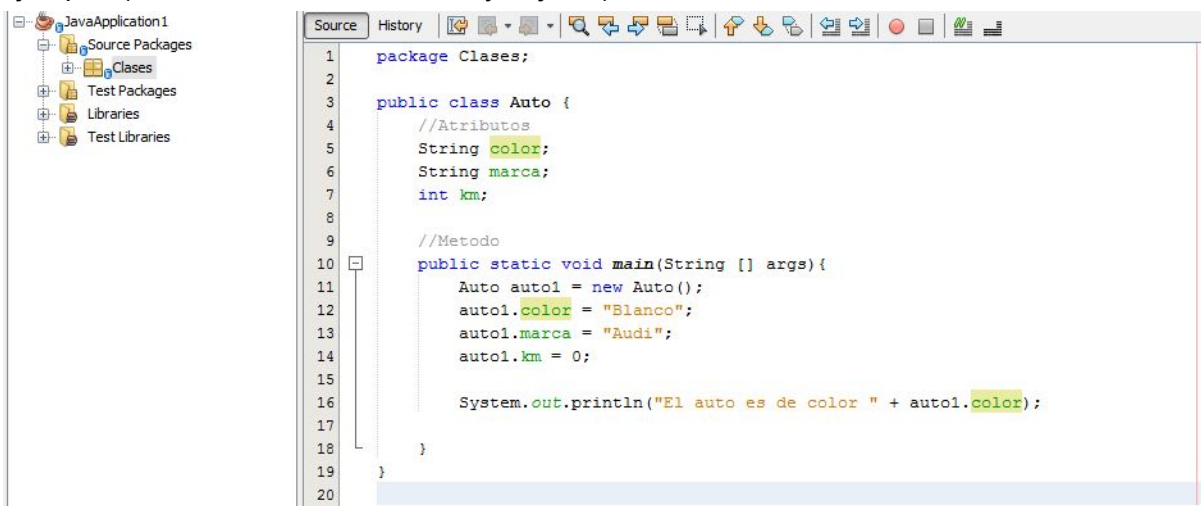


## Añadiendo una clase

- Abrimos el menú contextual sobre el **<paquete determinado>**
- Seleccionamos **Nuevo y Clase Java**



## Ejemplo: (Cómo definimos una clase y objetos)



## Métodos en Java

Un **método en Java** es un conjunto de instrucciones definidas dentro de una clase, que realizan una determinada tarea y a las que podemos invocar mediante un nombre.

Cuando se llama a un método, la ejecución del programa pasa al método y cuando éste acaba, la ejecución continúa a partir del punto donde se produjo la llamada.

Utilizando métodos:

- Podemos construir programas modulares.
- Se consigue la reutilización de código. En lugar de escribir el mismo código repetido cuando se necesite, por ejemplo para validar una fecha, se hace una llamada al método que lo realiza.

En Java un método siempre pertenece a una clase.

Todo programa java tiene un método llamado **main**. Este método es el punto de entrada al programa y también el punto de salida.

## ESTRUCTURA GENERAL DE UN MÉTODO JAVA

La estructura general de un método Java es la siguiente:

```
[especificadores] tipoDevuelto nombreMetodo ([lista parámetros]) [throws  
listaExcepciones] {  
    // instrucciones  
    [return valor;]  
}
```

Los elementos que aparecen entre corchetes son opcionales.

**especificadores** (opcional): determinan el tipo de acceso al método. Se verán en detalle más adelante.

**tipoDevuelto**: indica el tipo del valor que devuelve el método. En Java es imprescindible que en la declaración de un método, se indique el tipo de dato que ha de devolver. El dato se devuelve mediante la instrucción return. Si el método no devuelve ningún valor este tipo será void.

**nombreMetodo**: es el nombre que se le da al método. Para crearlo hay que seguir las mismas normas que para crear nombres de variables.

**Lista de parámetros** (opcional): después del nombre del método y siempre entre paréntesis puede aparecer una lista de parámetros (también llamados argumentos) separados por comas. Estos parámetros son los datos de entrada que recibe el método para operar con ellos. Un método puede recibir cero o más argumentos. Se debe especificar para cada argumento su tipo. **Los paréntesis son obligatorios** aunque estén vacíos.

**throws listaExcepciones** (opcional): indica las excepciones que puede generar y manipular el método.

**return**: se utiliza para devolver un valor. La palabra clave return va seguida de una expresión que será evaluada para saber el valor de retorno. Esta expresión puede ser compleja o puede ser simplemente el nombre de un objeto, una variable de tipo primitivo o una constante.

El tipo del valor de retorno debe coincidir con el tipoDevuelto que se ha indicado en la declaración del método.

Si el método no devuelve nada (tipoDevuelto = void) la instrucción return es opcional.

Un método puede devolver un tipo primitivo, un array, un String o un objeto.

Un método tiene un único punto de inicio, representado por la llave de inicio {. La ejecución de un método termina cuando se llega a la llave final } o cuando se ejecuta la instrucción return.

La instrucción return puede aparecer en cualquier lugar dentro del método, no tiene que estar necesariamente al final.

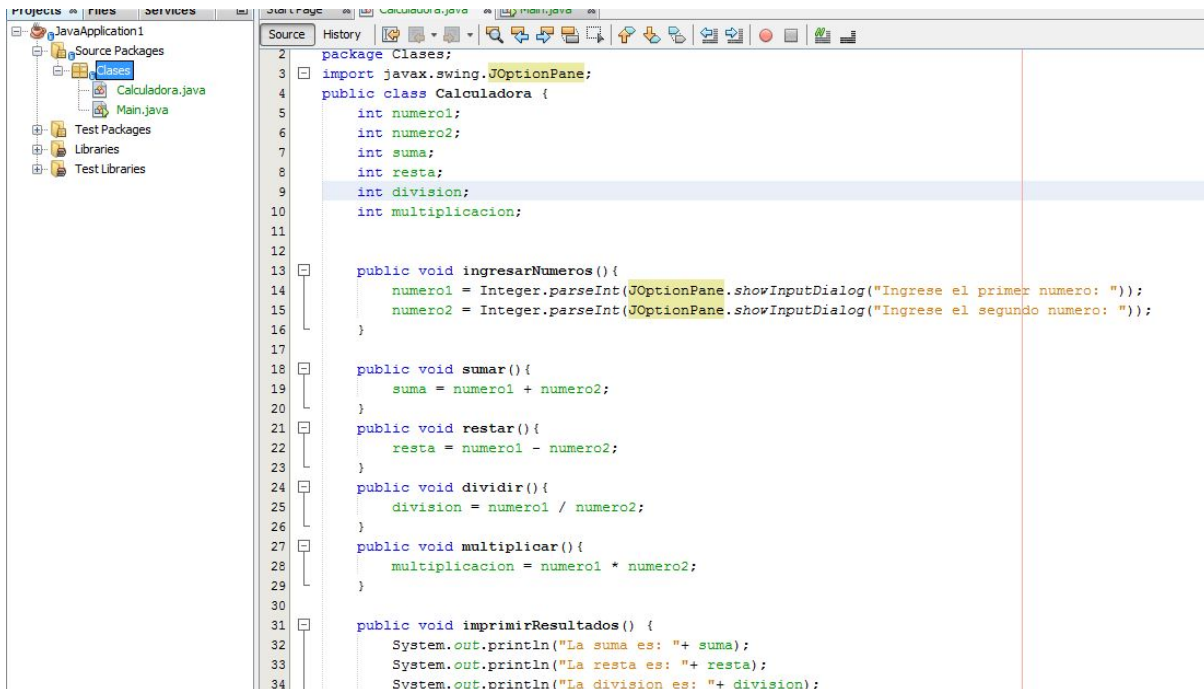
## IMPLEMENTACIÓN DE MÉTODOS EN JAVA

Pasos para implementar un método:

1. Describir lo que el método debe hacer
2. Determinar las entradas del método
3. Determinar los tipos de las entradas
4. Determinar el tipo del valor retornado
5. Escribir las instrucciones que forman el cuerpo del método
6. Prueba del método: diseñar distintos casos de prueba

Ejercicio: (Atributos y Métodos de una Clase)

Crear la clase Calculadora:



```
1 package Clases;
2 import javax.swing.JOptionPane;
3 public class Calculadora {
4     int numero1;
5     int numero2;
6     int suma;
7     int resta;
8     int division;
9     int multiplicacion;
10
11
12
13     public void ingresarNumeros() {
14         numero1 = Integer.parseInt(JOptionPane.showInputDialog("Ingresa el primer numero: "));
15         numero2 = Integer.parseInt(JOptionPane.showInputDialog("Ingresa el segundo numero: "));
16     }
17
18     public void sumar() {
19         suma = numero1 + numero2;
20     }
21     public void restar() {
22         resta = numero1 - numero2;
23     }
24     public void dividir() {
25         division = numero1 / numero2;
26     }
27     public void multiplicar() {
28         multiplicacion = numero1 * numero2;
29     }
30
31     public void imprimirResultados() {
32         System.out.println("La suma es: " + suma);
33         System.out.println("La resta es: " + resta);
34         System.out.println("La division es: " + division);
35     }
36 }
```



```
package Clases;
```

```
public class Main {  
    public static void main(String [] args){  
        Calculadora cal = new Calculadora();  
        cal.ingresarNumeros();  
        cal.sumar();  
        cal.restar();  
        cal.dividir();  
        cal.multiplicar();  
        cal.imprimirResultados();  
    }  
}
```

## Packages

Un paquete es una agrupación de clases. El programador puede crear sus propios paquetes agregando una línea con la palabra package y el nombre del paquete al inicio de un programa java.

Un ejemplo sería la siguiente línea: package mis.paquetes; Los nombres de los paquetes suelen escribirse con minúsculas, para diferenciarlos de las clases; el nombre de un paquete puede estar formado por varias palabras separadas por puntos, por ejemplo java.awt.event. Todas las clases que forman parte de un paquete deben estar en el mismo directorio.

Para utilizar un paquete dentro de un programa se debe de importar. Al importarlo no se cargan todas las clases que contiene el paquete, sólo se cargan las clases public que se vayan a utilizar. Al importar un paquete no se importan los subpaquetes. Para importarlos se debe hacer explícitamente ya que en realidad son paquetes diferentes.

Existen dos formas de utilizar import: para una clase y para todo un package:

```
import mis.paquetes.Teclado; // Importa solo la clase Teclado
```

```
import mis.paquetes.*; // Importa todas las clases contenidas en mis.paquetes
```

## Argumentos y parámetros:

Parámetro: Es una declaración de variable u objeto.

Argumento: Es un valor que se envía

Declaración de método:

### Parámetros



```
public void sumar(int numero1, int numero2){  
    suma = numero1 + numero2;  
}
```



Invocación de método:

## Argumentos



```
cal.sumar(nro1, nro2);
```

```
package Clases;
public class Calculadora {
    int suma;
    int resta;
    int division;
    int multiplicacion;

    public void sumar(int numero1, int numero2){
        suma = numero1 + numero2;
    }
    public void restar(int numero1, int numero2){
        resta = numero1 - numero2;
    }
    public void dividir(int numero1, int numero2){
        division = numero1 / numero2;
    }
    public void multiplicar(int numero1, int numero2){
        multiplicacion = numero1 * numero2;
    }

    public void imprimirResultados() {
        System.out.println("La suma es: "+ suma);
        System.out.println("La resta es: "+ resta);
        System.out.println("La division es: "+ division);
        System.out.println("La multiplicacion es: "+ multiplicacion);
    }
}
```

```

package Clases;

import javax.swing.JOptionPane;

public class Main {
    public static void main(String [] args){

        int nro1 = Integer.parseInt(JOptionPane.showInputDialog("Ingrese el primer numero: "));
        int nro2 = Integer.parseInt(JOptionPane.showInputDialog("Ingrese el segundo numero: "));

        Calculadora cal = new Calculadora();
        cal.sumar(nro1, nro2);
        cal.restar(nro1, nro2);
        cal.dividir(nro1, nro2);
        cal.multiplicar(nro1, nro2);
        cal.imprimirResultados();
    }
}

```

## Retorno de valores:

```

package Clases;
public class Calculadora {

    public int sumar(int numero1, int numero2){
        int suma = numero1 + numero2;
        return suma;
    }

    public int restar(int numero1, int numero2){
        int resta = numero1 - numero2;
        return resta;
    }

    public int dividir(int numero1, int numero2){
        int division = numero1 / numero2;
        return division;
    }

    public int multiplicar(int numero1, int numero2){
        int multiplicacion = numero1 * numero2;
        return multiplicacion;
    }

    public void imprimirResultados(int suma, int resta, int division, int multiplicacion) {
        System.out.println("La suma es: "+ suma);
        System.out.println("La resta es: "+ resta);
        System.out.println("La division es: "+ division);
        System.out.println("La multiplicacion es: "+ multiplicacion);
    }
}

```

```

package Clases;

import javax.swing.JOptionPane;

public class Main {
    public static void main(String [] args){

        int nro1 = Integer.parseInt(JOptionPane.showInputDialog("Ingrese el primer numero: "));
        int nro2 = Integer.parseInt(JOptionPane.showInputDialog("Ingrese el segundo numero: "));

        Calculadora cal = new Calculadora();
        int suma = cal.sumar(nro1, nro2);
        int resta = cal.restar(nro1, nro2);
        int division = cal.dividir(nro1, nro2);
        int multiplicacion = cal.multiplicar(nro1, nro2);
        cal.imprimirResultados(suma, resta, division, multiplicacion);
    }
}

```

## Método Constructor

Un método constructor es un método especial de una clase que se invoca siempre que se crea un objeto de esa clase.

Cuando se crea un objeto ocurren 3 cosas:

- Se asigna memoria para el objeto.
- Se inicializan los atributos de ese objeto.
- Se invoca al constructor de la clase que puede ser uno entre varios.

Ejemplo:

```
Auto auto1 = new Auto();
```

## Características:

Tienen el mismo nombre de la clase

No devuelven ningún valor. Por lo que no se le coloca nada (son los únicos métodos que no tienen tipo de dato que retornan)

Deben declararse como public

Ejemplo:

```

//Metodo Constructor
public Auto(String _color, String _marca, int _km) {
    color = _color;
    marca = _marca;
    km = _km;
}

...
public class Main {
    public static void main(){
        Auto auto1 = new Auto("Rojo", "Ferrari", 10);
        auto1.mostrarDatos();
    }
}

```

Introduccion *this*.

Ejercicios: Realizar una clase que represente Alumnos que tenga como atributos: nombre, apellido, edad, curso. Realizar el constructor de la clase y un método que muestre los datos de los alumnos.