Semana 03

Términos de Java

- **Java Virtual Machine:** Ejecuta los bytecodes.
- **Bytecode:** Producto del compilador para ejecutarse por JVM.
- **Java Development Kit:** Software para crear, compilar y ejecutar programas en Java.
- **Java Runtime Environment:** Permite la ejecución de aplicaciones creadas en Java, así como de applets.
- **Recolector de basura:** Recolecta los objetos no referenciados para recuperar memoria.

Características principales

- Independiente de la plataforma. Los bytecode se ejecutan sin importar el sistema operativo.
- Posee estructura de funcionamiento dentro de la Programación Orientada a Objetos, permite la implementación de los conceptos de abstracción, en encapsulación, herencia y polimorfismo.
- Simpleza. No requiere de apuntadores, sobrecarga de operadores, herencia múltiple o alojamiento explícito de memoria.
- Robusto, es confiable y reduce al mínimo los errores de funcionamiento.
- Seguro en manejo de memoria.
- Distribuido.
- Multi-hilos.
- Portable.
- Alto Rendimiento.
- Flexibilidad dinámica con otros lenguajes.

Introducción a Java

Detrás de un Hola Mundo

```
// Importando clases de paquetes
import java.io.*;

// Clase main
public class Hola {

    // Método main
    public static void main(String[] args)
```

```
    {

        // Declaración de impresión
        System.out.println("Hola mundo");
    }
}
```

Explicación

1. **Comentarios:** Los comentarios se usan para explicar el código y se usan de manera similar en Java o C o C++. Los compiladores ignoran las entradas de comentarios y no las ejecutan. Los comentarios pueden ser de una sola línea o de varias líneas.

   Ejemplos:
   // Comentario de una sola línea
   /* Comentarios de varias líneas*/

2. **import java.io*:** Esto significa que se pueden importar todas las clases del paquete io. El paquete Java io proporciona un conjunto de flujos de entrada y salida para leer y escribir datos en archivos u otras fuentes de entrada o salida.

3. **class:** La clase contiene los datos y métodos que se utilizarán en el programa. Los métodos definen el comportamiento de la clase. La clase Hola solo tiene un método Main en JAVA.

4. **static void main(): static** es una palabra clave que nos dice que se puede acceder a este método sin instanciar la clase.

5. **void:** las palabras clave dicen que este método no devolverá nada. El método main() es el punto de entrada de nuestra aplicación.

6. **System.out:** Este es el flujo de salida estándar que se utiliza para producir el resultado de un programa en un dispositivo de salida como la pantalla de la computadora.

7. **println():** Este método en Java también se usa para mostrar texto en la consola. Imprime el texto en la consola y el cursor se mueve al comienzo de la siguiente línea en la consola. La siguiente impresión tiene lugar a partir de la siguiente línea.

8. **String [ ]args:** Este es el argumento pasado a la función principal, que es un arreglo de cadenas con el nombre de arreglo args. Uno puede elegir su propio nombre flexible, pero muchos desarrolladores usan este nombre.

Elementos básicos de sintaxis

Compilar y ejecutar

javac EjemploDeCodigo.java
java EjemploDeCodigo

Nombre de archivo

El nombre de un archivo fuente debe coincidir exactamente con el nombre de la clase pública con la extensión .java. El nombre del archivo puede ser un nombre diferente si no tiene ninguna clase pública. Suponga que tiene un archivo de nombre Ejemplo de clase pública.

Ejemplo.java // Sintaxis válida
ejemplo.java // Sintaxis inválida

Case Sensitive

Java es un lenguaje que distingue entre mayúsculas y minúsculas, lo que significa que los identificadores AB, Ab, aB y ab son diferentes en Java.

System.out.println("Alice"); // sintaxis válida
system.out.println("Alice"); // sintaxis inválida

Nombres de clase

class MyJavaProgram // sintaxis válida
class 1Program // sintaxis inválida
class My1Program // sintaxis válida
class $Program // sintaxis válida, pero no se recomienda
class My$Program // sintaxis válida, pero no se recomienda
class myJavaProgram // sintaxis válida, pero no se recomienda

Palabras reservadas en Java

abstract assert boolean break

byte	case	catch	char
class	const	continue	default
do	double	else	enum
extends	final	finally	float
for	goto	if	implements
import	instanceof	int	interface
long	native	new	package
private	protected	public	return
short	static	strictfp	super
switch	synchronized	this	throw
throws	transient	try	void
volatile	while		

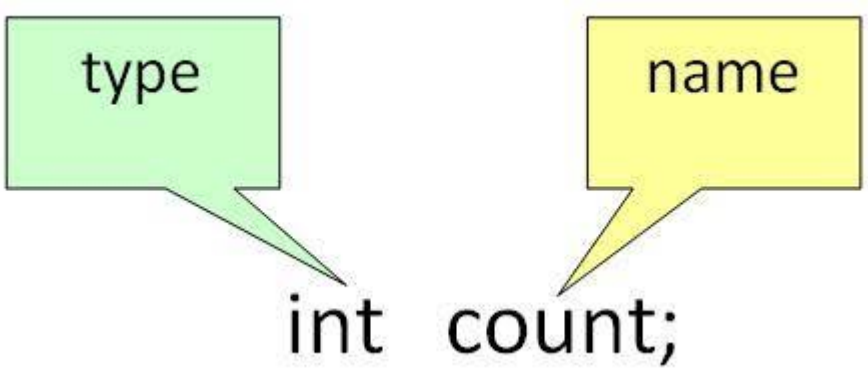
Variables

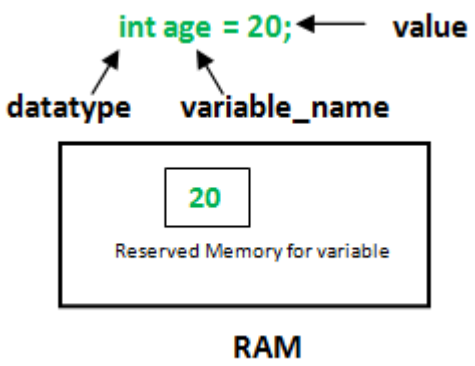
Las variables son un contenedor de datos que guarda los valores de los datos durante la ejecución del programa Java. A cada variable se le asigna un tipo de datos que designa el tipo y la cantidad de valor que puede contener. Una variable es un nombre de ubicación de memoria para los datos.

Una variable es un nombre dado a una ubicación de memoria. Es la unidad básica de almacenamiento en un programa.

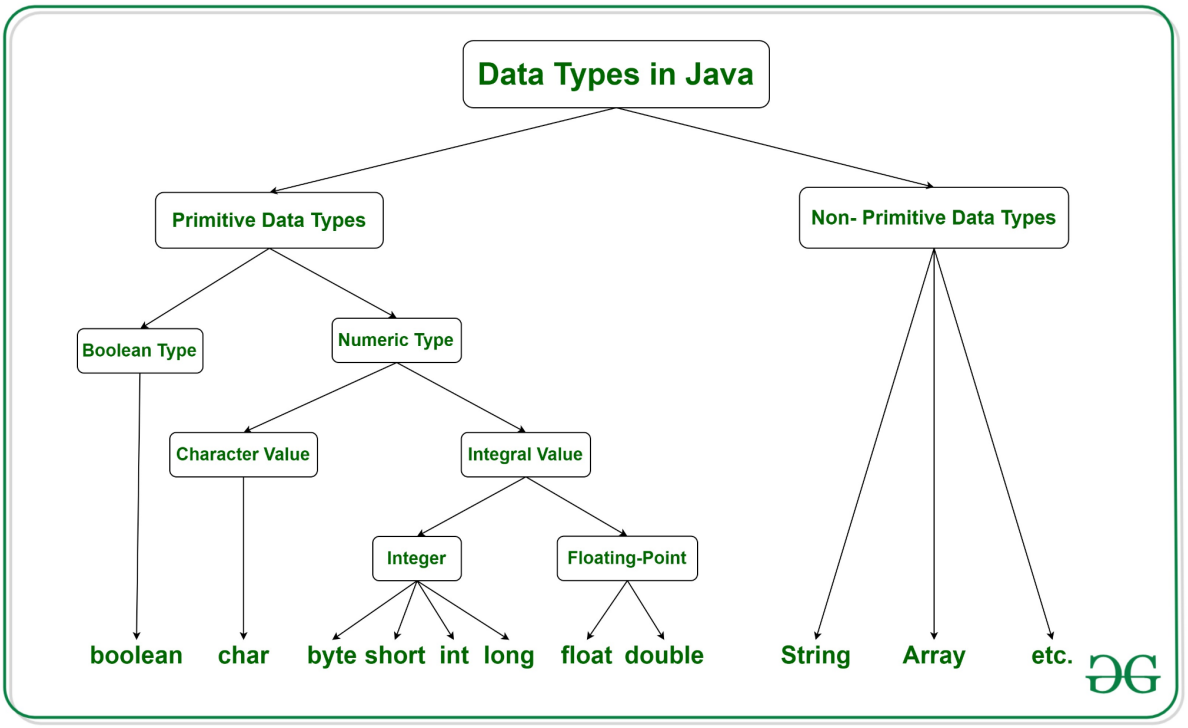
- El valor almacenado en una variable se puede cambiar durante la ejecución del programa.
- Una variable es solo un nombre dado a una ubicación de memoria. Todas las operaciones realizadas en la variable afectan esa ubicación de memoria.
- En Java, todas las variables deben declararse antes de su uso.

Declaración de variables



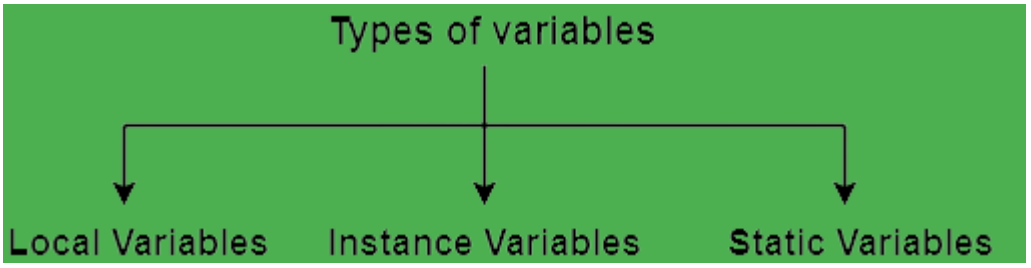


Tipos de Dato



TYPE	DESCRIPTION	DEFAULT	SIZE	EXAMPLE LITERALS	RANGE OF VALUES
boolean	true or false	false	1 bit	true, false	true, false
byte	twos complement integer	0	8 bits	(none)	-128 to 127
char	unicode character	\u0000	16 bits	'a', '\u0041', '\101', '\\', '\'','\n',' β'	character representation of ASCII values 0 to 255
short	twos complement integer	0	16 bits	(none)	-32,768 to 32,767
int	twos complement integer	0	32 bits	-2, -1, 0, 1, 2	-2,147,483,648 to 2,147,483,647
long	twos complement integer	0	64 bits	-2L, -1L, 0L, 1L, 2L	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	IEEE 754 floating point	0.0	32 bits	1.23e100f, -1.23e-100f, .3f, 3.14F	upto 7 decimal digits
double	IEEE 754 floating point	0.0	64 bits	1.23456e300d, -1.23456e-300d, 1e1d	upto 16 decimal digits

Tipos de Variables



Locales

Una variable definida dentro de un bloque, método o constructor se denomina variable local.

- Estas variables se crean cuando se ingresa al bloque, o se llama a la función y se destruye después de salir del bloque o cuando la llamada regresa de la función.
- El alcance de estas variables existe solo dentro del bloque en el que se declaran las variables, es decir, podemos acceder a estas variables solo dentro de ese bloque.
- La inicialización de la variable local es obligatoria antes de usarla en el ámbito definido.

import java.io.*;

```
class Ejemplo {
	public static void main(String[] args)
	{
		int var = 10; // Declaración de variable local
		// Esta variable sólo está visible para el método main
		System.out.println("Variable local: " + var);
	}
}
```

Instancia

Las variables de instancia son variables no estáticas y se declaran en una clase fuera de cualquier método, constructor o bloque.

- Como las variables de instancia se declaran en una clase, estas variables se crean cuando se crea un objeto de la clase y se destruyen cuando se destruye el objeto.
- A diferencia de las variables locales, podemos usar especificadores de acceso para variables de instancia. Si no especificamos ningún especificador de acceso, se utilizará el especificador de acceso predeterminado.
- La inicialización de una variable de instancia no es obligatoria. Su valor por defecto es 0.
- Solo se puede acceder a las variables de instancia mediante la creación de objetos.

```
import java.io.*;

class Ejemplo {

	public String alumno; // Declaración de variable de instancia

	public Ejemplo()
	{ // Constructor por defecto

		this.alumno = "Computólogo"; // Variable de instancia inicializada
	}
//Método main
	public static void main(String[] args)
	{

		// Creación de objeto
		Ejemplo nombre = new Ejemplo();
		// Mostrando la salida
```

```
            System.out.println("Mi nombre es: " + nombre.alumno);
        }
}
```

Estáticas

Las variables estáticas también se conocen como variables de clase.

- Estas variables se declaran de manera similar a las variables de instancia. La diferencia es que las variables estáticas se declaran usando la palabra clave static dentro de una clase fuera de cualquier método, constructor o bloque.
- A diferencia de las variables de instancia, sólo podemos tener una copia de una variable estática por clase, independientemente de cuántos objetos creemos.
- Las variables estáticas se crean al comienzo de la ejecución del programa y se destruyen automáticamente cuando finaliza la ejecución.
- La inicialización de una variable estática no es obligatoria. Su valor por defecto es 0.
- Si accedemos a una variable estática como una variable de instancia (a través de un objeto), el compilador mostrará un mensaje de advertencia, que no detendrá el programa. El compilador reemplazará el nombre del objeto con el nombre de la clase automáticamente.
- Si accedemos a una variable estática sin el nombre de la clase, el compilador agregará automáticamente el nombre de la clase.

```
import java.io.*;

class Ejemplo {

public static String nombre = "Compu";              //Declaración

        public static void main (String[] args) {

        //Variable nombre se puede accesar sin creación
        //Mostrando en pantalla
            System.out.println("Variable nombre es : "+Ejemplo.nombre);
        }
}
```

Alcance de Variables

El alcance de una variable es la parte del programa donde se puede acceder a la variable.

Variables Miembro

Estas variables deben declararse dentro de la clase (fuera de cualquier función). Se puede acceder directamente desde cualquier lugar de la clase. Veamos el siguiente ejemplo:

```
public class Test
{
   // Todas las variables están definidas en la clase
   // son variables miembro
   int a;
   private String b;
   void method1() {....}
   int method2() {....}
   char c;
}
```

Variables de Bloque

Una variable declarada dentro de un par de corchetes "{" y "}" en un método tiene alcance solo dentro de los corchetes.