Tipos de datos de Java

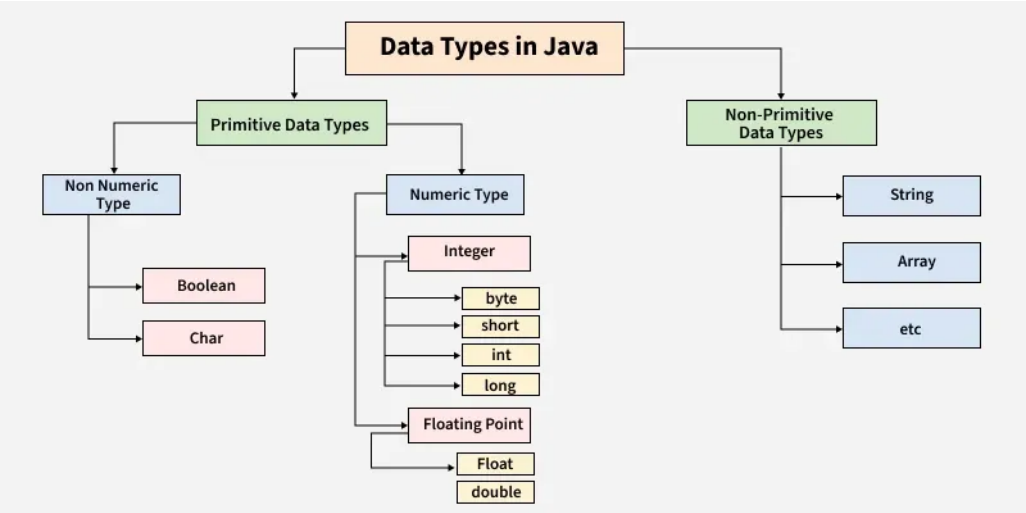
https://www.geeksforgeeks.org/java/java-data-types/

Java es un lenguaje de programación de **tipado estático**, lo que significa que *el tipo de dato de cada variable se conoce en tiempo de compilación*. El compilador garantiza la seguridad de tipos y evita asignaciones inválidas como:

```
int x = "GfG"; // Error de compilación
```

Los **tipos de datos** en Java definen el *tipo de datos que una variable puede almacenar y la memoria necesaria para ello*. Se dividen, a grandes rasgos, en dos categorías:

- **Tipos de datos primitivos**: Almacenan valores simples directamente en memoria.
- **Tipos de datos no primitivos** (**de referencia**): Almacenan referencias de memoria a objetos.



Tipos de datos primitivos

Los tipos de datos primitivos **almacenan valores simples directamente en la memoria**. Java proporciona **ocho** tipos de datos primitivos, cada uno con un tamaño y un rango fijos, que se resumen a continuación:

Tipo	Descripción	Default	Tamaño	Ejemplo	Rango
boolean	Valores logicos	false	No definido en JVM	true, false	true o false
byte	8-bit entero con signo	0	1 byte	10	-128 a 127
char	16-bit caracter Unicode	\u0000	2 bytes	'A', '\u0041'	0 a 65,535
short	16-bit entero con signo	0	2 bytes	2000	-32,768 a 32,767
int	32-bit entero con signo	0	4 bytes	1000, -500	-2,147,483,648 a 2,147,483,647
long	64-bit entero con signo	0L	8 bytes	123456789L	±9.22e18
float	32-bit punto flotante	0.0f	4 bytes	3.14f	~ 6-7 digitos de precisión
double	64-bit punto flotante	0.0d	8 bytes	3.14159d	~ 15-16 digitos de precisión

1. Tipo de dato `boolean`

Representa uno de dos valores lógicos: `true` o `false`. Se usa comúnmente en condiciones y sentencias de control.

Sintaxis:

```
boolean booleanVar;
```

```
public class Geeks {
    public static void main(String[] args) {
        boolean isJavaFun = true;
        boolean isFishTasty = false;

        System.out.println("¿Es divertido Java? "
                           + isJavaFun);
        System.out.println("¿Es sabroso el pescado? "
                           + isFishTasty);
    }
}
```

Salida:

```
¿Es divertido Java? verdadero
¿Es sabroso el pescado? falso
```

2. Tipo de dato `byte`

Un entero con signo de 8 bits que se utiliza para ahorrar memoria en grandes matrices numéricas.

Sintaxis:

```
byte byteVar;
```

```
public class Geeks {
    public static void main(String[] args) {
        byte edad = 25;
        byte temperatura = -10;

        System.out.println("Edad: " + edad);
        System.out.println("Temperatura: " + temperatura);
    }
}
```

Salida:

```
Edad: 25
Temperatura: -10
```

3. Tipo de dato `short`

Un entero con signo de 16 bits que se usa a menudo cuando la memoria es limitada y los valores son de tamaño moderado.

Sintaxis:

```
short shortVar;
```

```
public class Geeks {
    public static void main(String[] args) {
        short estudiantes = 1000;
        short temp = -200;

        System.out.println("Número de estudiantes: "
                           + estudiantes);
        System.out.println("Temperatura: " + temp);
    }
}
```

Salida:

```
Número de estudiantes: 1000
Temperatura: -200
```

4. Tipo de dato `int`

Un entero con signo de 32 bits y el tipo de dato numérico más utilizado.

Sintaxis:

```
int intVar;
```

Tamaño: 4 bytes (32 bits)

```
public class Geeks {
    public static void main(String[] args) {
        int poblacion = 2000000;
        int distancia = 150000000;

        System.out.println("Población: " + poblacion);
        System.out.println("Distancia: " + distancia);
    }
}
```

Salida:

```
Población: 2.000.000
Distancia: 150.000.000
```

5. Tipo de dato `long`

Un entero con signo de 64 bits que se utiliza cuando int no es suficiente para valores grandes.

Sintaxis:

```
long longVar;
```

Tamaño : 8 bytes (64 bits)

```
public class Geeks {
    public static void main(String[] args) {
        long poblacionMundial = 7800000000L;
        long annosLuz = 9460730472580800L;

        System.out.println("Población mundial: "
                           + poblacionMundial);
```

```
        System.out.println("Distancia en años luz: "
                           + annosLuz);
    }
}
```

Salida:

```
Población mundial: 7800000000
Distancia en años luz: 9460730472580800
```

6. Tipo de dato `float`

Tipo de punto flotante de precisión simple de 32 bits utilizado para valores fraccionarios.

Sintaxis:

```
float floatVar;
```

Tamaño : 4 bytes (32 bits)

```
public class Geeks {
    public static void main(String[] args) {
        float pi = 3.14f;
        float gravedad = 9.81f;

        System.out.println("Valor de Pi: " + pi);
        System.out.println("Gravedad: " + gravedad);
    }
}
```

Salida:

```
Valor de Pi: 3,14
Gravedad: 9,81
```

7. Tipo de dato `double`

Tipo de coma flotante de doble precisión de 64 bits y el predeterminado para números decimales.

Sintaxis:

```
double doubleVar;
```

Tamaño: 8 bytes (64 bits). Se recomienda revisar los errores de redondeo en Java.

```
public class Geeks {
    public static void main(String[] args) {
        double pi = 3.141592653589793;
        double avogadro = 6.02214076e23;

        System.out.println("Valor de Pi: " + pi);
        System.out.println("Número de Avogadro: " + avogadro);
    }
}
```

Salida:

```
Valor de Pi: 3.141592653589793
Número de Avogadro: 6.02214076E23
```

8. Tipo de dato `char`

Un carácter Unicode de 16 bits utilizado para almacenar símbolos o letras individuales.

Sintaxis:

```
char charVar;
```

Tamaño: 2 bytes (16 bits)

Ejemplo: Este ejemplo muestra cómo usar el tipo de dato `char` para almacenar caracteres individuales.

```
public class Geeks {
    public static void main(String[] args) {
        char grado = 'A';
        char simbolo = '$';

        System.out.println("Calificación: " + grado);
        System.out.println("Símbolo: " + simbolo);
    }
}
```

Salida:

```
Calificación: A
Símbolo: $
```

Tipos de datos no primitivos (de referencia)

Los tipos de datos no primitivos almacenan referencias (direcciones de memoria) en lugar de valores reales. Son creados por el usuario e incluyen tipos como `String`, `Class`, `Object`, `Interface` y `Array`.

1. `String`

Una cadena/*String* representa una secuencia de caracteres entre comillas dobles. A diferencia de C/C++, las cadenas en Java son objetos e inmutables.

Sintaxis:

```
String str = "Hello";
```

```
public class Geeks {
    public static void main(String[] args) {
        String nombre = "Geek1";
        String mensaje = "Bienvenido a Java";
        System.out.println("Nombre: " + nombre);
        System.out.println("Mensaje: " + mensaje);
    }
}
```

Salida:

```
Nombre: Geek1
Mensaje: Bienvenido a Java
```

Nota: Las cadenas no se pueden modificar después de su creación. Utilice `StringBuilder` para manipulaciones complejas de cadenas.

2. `Class`

Una clase/*Class* es un modelo definido por el usuario que define variables y métodos. Representa un tipo de objeto y constituye la base de la programación orientada a objetos.

```
class Car {
    String modelo;
    int year;

    Car(String model, int year) {
        this.model = modelo;
        this.year = year;
    }

    void display() {
        System.out.println(modelo + " " + year);
    }
}

public class Geeks {
    public static void main(String[] args) {
        Car myCar = new Car("Toyota", 2020);
        myCar.display();
    }
}
```

Salida:

```
Toyota 2020
```

3. Object

Un objeto/*Object* es una instancia de una clase que representa entidades del mundo real. Posee estado (datos), comportamiento (métodos) e identidad (referencia única).

```
class Car {
    String modelo;
    int year;

    Car(String modelo, int year) {
        this.modelo = modelo;
        this.year = year;
    }
}

public class Geeks {
    public static void main(String[] args) {
        Car myCar = new Car("Honda", 2021);
        System.out.println("Modelo de coche: " + myCar.modelo);
        System.out.println("Año del coche: " + myCar.year);
```

```
    }
}
```

Salida:

```
Modelo de coche: Honda
Año del coche: 2021
```

4. Interface

Una interfaz/*Interface* define un contrato de métodos abstractos que las clases que la implementan deben definir. Proporciona una forma de lograr la abstracción y la herencia múltiple en Java.

```
interface Animal {
    void sonido();
}

class Dog implements Animal {
    public void sonido() {
        System.out.println("Ladra");
    }
}

public class Geeks {
    public static void main(String[] args) {
        Animal dog = new Dog();
        dog.sonido();
    }
}
```

Salida:

```
Ladra
```

5. Array

Un arreglo/*array* almacena varios elementos del mismo tipo en una sola estructura. Los arreglos de Java son objetos, se asignan dinámicamente y se indexan desde 0.

```
public class Geeks {
    public static void main(String[] args) {
        int[] numeros = {1, 2, 3, 4, 5};
        String[] nombres = {"Geek1", "Geek2", "Geek3"};
        System.out.println("Primer número: " + numeros[0]);
        System.out.println("Segundo nombre: " + nombres[1]);
    }
}
```

Salida:

```
Primer número: 1
Segundo nombre: Geek2
```