**Java Core**

Java is a class-based, **object-oriented programming language** that is designed to have as few implementation dependencies as possible. It is intended to let application developers **Write Once and Run Anywhere (WORA)**, meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.

Java is known for its simplicity, robustness, and security features, making it a popular choice for enterprise-level applications. Java applications are compiled to byte code that can run on any Java Virtual Machine. The syntax of Java is similar to C/C++.

## Key Features of Java

### **1. Platform Independent**

**Compiler** converts source code to [**byte code**](https://www.geeksforgeeks.org/byte-code-in-java/)and then the JVM executes the bytecode generated by the compiler. This byte code can run on any platform be it Windows, Linux, or macOS which means if we compile a program on **Windows**, then we can run it on**Linux** and vice versa. Each operating system has a different**JVM**, but the output produced by all the OS is the same after the execution of the **byte code**. That is [**why we call java a platform-independent language.**](https://www.geeksforgeeks.org/java-platform-independent/)

### **2. Object-Oriented Programming**

**Java**is an [**object-oriented language**](https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/), promoting the use of**objects** and **classes**. Organizing the program in the terms of a collection of objects is a way of object-oriented programming, each of which represents an instance of the class.

The**four main concepts of Object-Oriented programming** are:

* [***Abstraction***](https://www.geeksforgeeks.org/abstraction-in-java-2/)
* [***Encapsulation***](https://www.geeksforgeeks.org/encapsulation-in-java/)
* [***Inheritance***](https://www.geeksforgeeks.org/inheritance-in-java/)
* [***Polymorphism***](https://www.geeksforgeeks.org/polymorphism-in-java/)

### 3. Simplicity

[**Java’s syntax**](https://www.geeksforgeeks.org/java-basic-syntax/) is simple and easy to learn, especially for those familiar with **C** or **C++**. It eliminates complex features like pointers and multiple inheritances, making it easier to ***write, debug,***and ***maintain code.***

### 4. Robustness

**Java language is robust which means reliable**. It is developed in such a way that it puts a lot of effort into checking errors as early as possible, that is why the java compiler is able to detect even those errors that are not easy to detect by another programming language. The main features of java that make it robust are garbage collection, exception handling, and memory allocation.

### 5. Security

**In java, we don’t have pointers**, so we cannot access [**out-of-bound arrays**](https://www.geeksforgeeks.org/array-index-out-of-bounds-exception-in-java/)i.e it shows **ArrayIndexOutOfBound Exception** if we try to do so. That’s why several security flaws like stack corruption or buffer overflow are impossible to exploit in Java. Also, java programs run in an environment that is independent of the **os(operating system)** environment which makes java programs more secure.

### **6.** **Distributed**

**We can create distributed applications using the java programming language.** Remote Method Invocation and Enterprise Java Beans are used for creating distributed applications in java. The java programs can be easily distributed on one or more systems that are connected to each other through an internet connection.

### **7.** **Multithreading**

[**Java supports multithreading**](https://www.geeksforgeeks.org/multithreading-in-java/), enabling the**concurrent execution** of multiple parts of a program. This feature is particularly useful for applications that require high performance, such as games and real-time simulations.

### 8. Portability

As we know, java code written on one machine can be run on another machine. The platform-independent feature of java in which its platform-independent bytecode can be taken to any platform for execution makes java portable. [**WORA(Write Once Run Anywhere)**](https://www.geeksforgeeks.org/why-is-java-write-once-and-run-anywhere/)makes java application to generates a [**‘.class’ file**](https://www.geeksforgeeks.org/java-class-file/) that corresponds to our applications(program) but contains code in binary format. It provides architecture-neutral ease, as bytecode is independent of any machine architecture. It is the primary reason java is used in the enterprising IT industry globally worldwide.

### **9. High Performance**

**Java architecture** is defined in such a way that it reduces overhead during the runtime and at some times java uses[**Just In Time (JIT) compiler**](https://www.geeksforgeeks.org/just-in-time-compiler/)where the compiler compiles code on-demand basis where it only compiles those methods that are called making applications to execute faster.

## Essential Java Terminologies You Need to Know

Before learning Java, one must be familiar with these common terms of Java.

### **1. Java Virtual Machine(JVM)**

The [**JVM**](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/) is an integral part of the**Java platform**, responsible for executing Java bytecode. It ensures that the output of Java programs is consistent across different platforms.

* Writing a program is done by a java programmer like you and me.
* The compilation is done by the **JAVAC** compiler which is a primary Java compiler included in the Java development kit (JDK). It takes the Java program as input and generates bytecode as output.
* In the Runningphase of a program,**JVM** executes the bytecode generated by the compiler.

The Java Virtual Machine (JVM) is designed to run the bytecode generated by the Java compiler.Each operating system has its own version of the JVM, but all JVMs follow the same rules and standards. This means Java programs can run the same way on any device with a JVM, regardless of the operating system**.** This is why Java is called a platform-independent language.

### **2. Bytecode**

**Bytecode** is the intermediate representation of Java code, generated by the Java compiler. It is platform-independent and can be executed by the JVM.

### **3. Java Development Kit(JDK)**

While we were using the term[**JDK**](https://www.geeksforgeeks.org/jdk-in-java/) when we learn about bytecode and JVM. So, as the name suggests, it is a complete Java development kit that includes everything including **compiler,** **Java Runtime Environment (JRE)**, J**ava Debuggers**, J**ava Docs**, etc. For the program to execute in java, we need to install JDK on our computer in order to create, compile and run the java program.

### **4. Java Runtime Environment (JRE)**

JDK includes JRE. [**JRE**](https://www.geeksforgeeks.org/jre-in-java/) installation on our computers allows the java program to run, however, we cannot compile it. JRE includes a browser, JVM, applet support, and plugins. For running the java program, a computer needs JRE.

### **5. Garbage Collector**

In Java, programmers can’t delete the objects. To delete or recollect that memory JVM has a program called [**Garbage Collector**](https://www.geeksforgeeks.org/garbage-collection-java/). Garbage Collectors can recollect the objects that are not referenced. So**Java** makes the life of a programmer easy by **handling memory management**. However, **programmers should be careful about their code whether they are using objects that have been used for a long time**. Because Garbage cannot recover the memory of objects being referenced.

### **6. ClassPath**

The **[Classpath](https://www.geeksforgeeks.org/classpath-in-java/)**is the file path where the java runtime and Java compiler look for **.class** files to load. By default, JDK provides many libraries. If you want to include external libraries they should be added to the classpath.

Basically everything in java is represented in Class as an object including the main function.

## Advantages of Java

* **Platform independent:**Java code can run on any platform that has a**Java Virtual Machine (JVM)** installed, which means that applications can be written once and run on any device.
* **Object-Oriented:** Java is an object-oriented programming language, which means that it follows the principles of encapsulation, inheritance, and polymorphism.
* **Security:** Java has built-in security features that make it a secure platform for developing applications, such as automatic memory management and type checking.
* **Large community:** Java has a large and active community of developers, which means that there is a lot of support available for learning and using the language.
* **Enterprise-level applications:**Java is widely used for developing enterprise-level applications, such as web applications, e-commerce systems, and database systems.

## Disadvantages of Java

1. **Performance:**Java can be slower compared to other programming languages, such as C++, due to its use of a virtual machine and automatic memory management.
2. **Memory management:** Java’s automatic memory management can lead to slower performance and increased memory usage, which can be a drawback for some applications.

**1. How Java Works?**

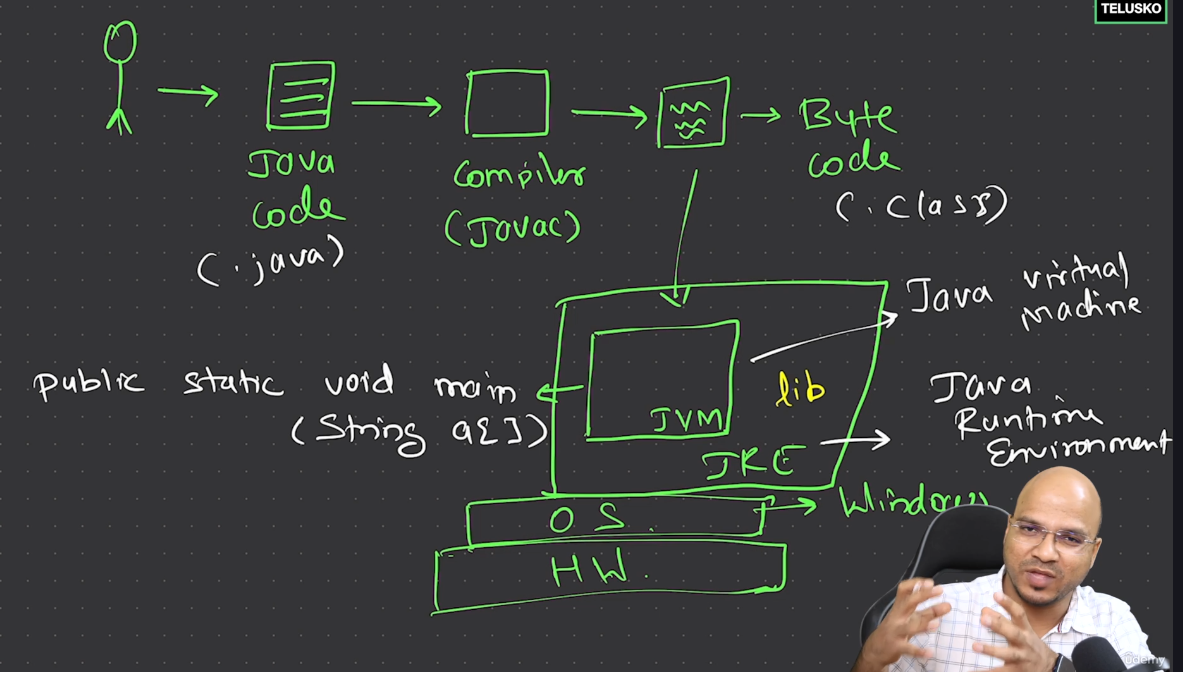
public class \_1\_Hello {

    public static void main(String[] args) {

        System.out.println("Hello, World!");

    }

}

****

### **Java Execution Process (Explained with Diagram)**

#### 1. **Writing Code**

* You write Java code (.java file), e.g., \_1\_Hello.java.
* This file contains the main() method, which is the entry point for execution.

#### 2. **Compiling Code**

* You use the **Java Compiler (javac)** to compile the code:

bash

CopyEdit

javac \_1\_Hello.java

* This converts the code into **Bytecode** (.class file) which is platform-independent.

#### 3. **Bytecode**

* The compiled .class file contains instructions in Bytecode format.
* This bytecode is not directly executed by the operating system.

#### 4. **JRE (Java Runtime Environment)**

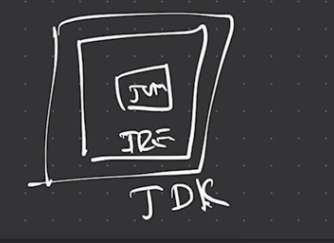
* The JRE contains:
  + **JVM (Java Virtual Machine)** — executes bytecode.
  + **Libraries (lib)** — support runtime functionalities.

#### 5. **JVM**

* The **JVM** reads the .class bytecode and **interprets or JIT-compiles** it to native code.
* The native code is then executed on the **underlying OS (like Windows)**, which runs on **hardware (HW)**.

#### 6. **Output**

* The output "Hello, World!" is printed to the console via System.out.println

****

**2. Variables**

variables are containers that store data in memory. Variable: is used initialize and store data.

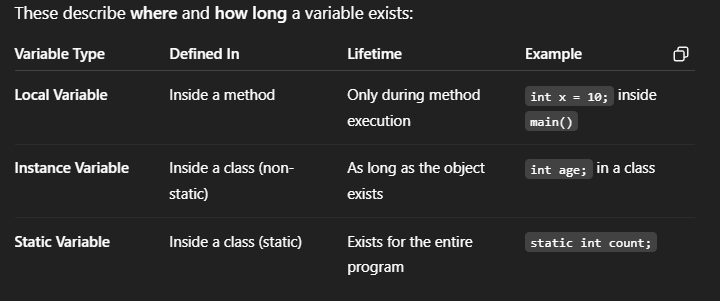
**Key Components of Variables in Java:**

A variable in Java has three components, which are listed below:

* **Data Type:**Defines the kind of data stored (e.g., int, String, float).
* **Variable Name:**A unique identifier following Java naming rules.
* **Value:** The actual data assigned to the variable.

**There are three types of variables in Java:**

**Types of Variables in Java (Based on Scope)**

****

**Type casting in Java:**

**Implicit casting(widening)**

Byte -> short->int->long->float->double ->

byte = 8bit (integer type)

short = 16bit (integer type)

int = 32bit(integer type)

long = 64bit(integer type)

float= 32 bit (floating type)

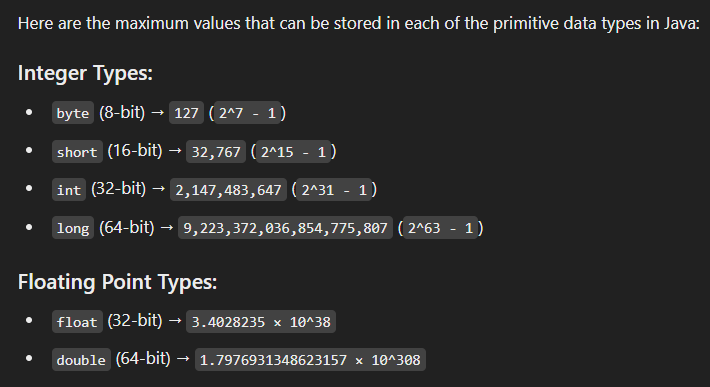
double = 64 bit(floating type)

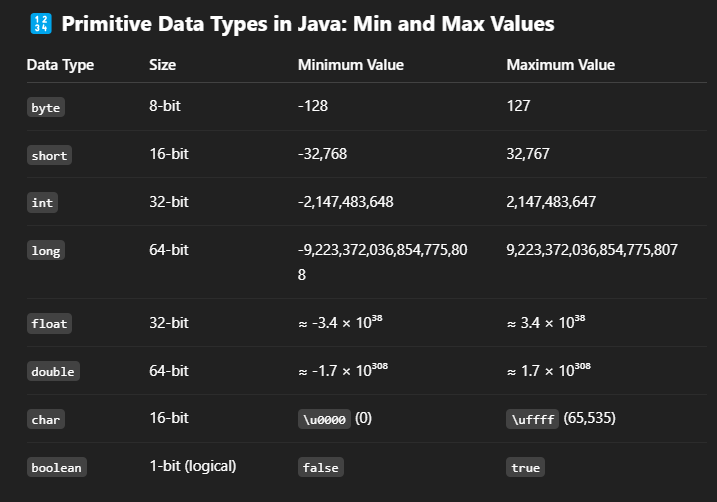
char = 16bit (UNICODE)

Boolean = True/False

**Explicitly casting (Narrowing)**

double -> float-> long -> int -> short -> byte ->



****

**3. Data types:**

**Data types in Java**are of different sizes and values that can be stored in a variable that is made as per convenience and circumstances to cover up all test cases.

[Java](https://www.geeksforgeeks.org/java/) is**statically typed** and also a **strongly typed** language because each type of**data (such as integer, character, hexadecimal, packed decimal, and so forth)** is predefined as part of the programming language and all constants or variables defined for a given program must be declared with the specific data types.

### Why Data Types Matter in Java?

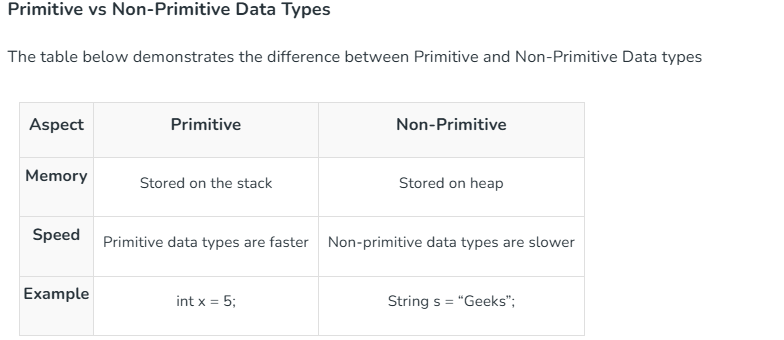
Data types matter in Java because of the following reasons, which are listed below:

* **Memory Efficiency:** Choosing the right type (byte vs int) saves memory.
* **Performance:** Proper types reduce runtime errors.
* **Code Clarity:** Explicit typing makes code more readable.

## **Java Data Type Categories**

Java has two categories in which data types are segregated

* **Primitive Data Type:**These are the basic building blocks that store simple values directly in memory. Examples of primitive data types are**boolean, char, byte, short, int, long, float, and double.**
* **Non-Primitive Data Types (Object Types):**These are reference types that store memory addresses of objects. Examples of Non-primitive data types are String, Array, Class, Interface, and Object

****

**Primitive Data Types:**

### **1. boolean Data Type**

The boolean data type represents a logical value that can be either true or false. Conceptually, it represents a single bit of information, but the actual size used by the virtual machine is implementation-dependent and typically at least one byte (eight bits) in practice. Values of the boolean type are not implicitly or explicitly converted to any other type using casts.

**Syntax:**

*boolean booleanVar;*

### **2. byte Data Type**

The byte data type is an 8-bit signed two’s complement integer. The byte data type is useful for saving memory in large arrays.

**Syntax:**

*byte byteVar;*

### **3. short Data Type**

The short data type is a 16-bit signed two’s complement integer. Similar to byte, a short is used when memory savings matter, especially in large arrays where space is constrained.

**Syntax:**

*short shortVar;*

### **4. int Data Type**

It is a 32-bit signed two’s complement integer.

**Syntax:**

*int intVar;*

### **5. long Data Type**

The long data type is a 64-bit signed two’s complement integer. It is used when an int is not large enough to hold a value, offering a much broader range.

**Syntax:**

*long longVar;*

### **6. float Data Type**

The float data type is a single-precision 32-bit IEEE 754 floating-point. Use a float (instead of double) if you need to save memory in large arrays of floating-point numbers. The size of the float data type is 4 bytes (32 bits).

**Syntax:**

*float floatVar;*

### **7. double Data Type**

The double data type is a double-precision 64-bit IEEE 754 floating-point. For decimal values, this data type is generally the default choice. The size of the double data type is 8 bytes or 64 bits.

**Syntax:**

*double doubleVar;*

### **8. char Data Type**

The char data type is a single 16-bit Unicode character with the size of 2 bytes (16 bits).

**Syntax:**

*char charVar;*

### **Why is the Size of char 2 bytes in Java?**

Unlike languages such as C or C++ that use the **ASCII character** set, Java uses the Unicode character set to support internationalization. Unicode requires more than 8 bits to represent a wide range of characters from different languages, including Latin, Greek, Cyrillic, Chinese, Arabic, and more. As a result, Java uses 2 bytes to store a char, ensuring it can represent any **Unicode** character.

## **Non-Primitive (Reference) Data Types**

### **1. Strings**

[Strings](https://www.geeksforgeeks.org/strings-in-java/)are defined as an array of characters. The difference between a character array and a string in Java is, that the string is designed to hold a sequence of characters in a single variable whereas, a character array is a collection of separate char-type entities. Unlike C/C++, Java strings are not terminated with a null character.

* A Java library example is [Comparator Interface](https://www.geeksforgeeks.org/comparator-interface-java/). If a class implements this interface, then it can be used to sort a collection.

### **5. Array**

An [Array](https://www.geeksforgeeks.org/arrays-in-java/) is a group of like-typed variables that are referred to by a common name. Arrays in Java work differently than they do in C/C++. The following are some important points about Java arrays.

* In Java, all arrays are dynamically allocated. (discussed below)
* Since arrays are objects in Java, we can find their length using member length. This is different from C/C++ where we find length using size.
* A Java array variable can also be declared like other variables with [] after the data type.
* The variables in the array are ordered and each has an index beginning with 0.
* Java array can also be used as a static field, a local variable, or a method parameter.
* The **size**of an array must be specified by an int value and not long or short.
* The direct superclass of an array type is Object.
* Every array type implements the interfaces [Cloneable](https://www.geeksforgeeks.org/marker-interface-java/)and [java.io.Serializable](https://www.geeksforgeeks.org/serialization-in-java/).

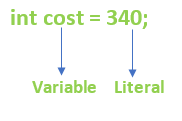
**Key Points to Remember:**

* **Strong Typing**: Java enforces strict type checking at compile-time, reducing runtime errors.
* **Memory Efficiency**: Choosing the right data type based on the range and precision needed helps in efficient memory management.
* **Immutability of Strings**: Strings in Java cannot be changed once created, ensuring safety in multithreaded environments.
* **Array Length**: The length of arrays in Java is fixed once declared, and it can be accessed using the length attribute

**4. Literals:**

## **Literals**

In Java, **literals** are the constant values that appear directly in the program. It can be assigned directly to a variable. Java has various types of literals. The following figure represents a literal.



## **Types of Literals in Java**

There are the majorly **four** types of literals in Java:

1. Integer Literal
2. Character Literal
3. Boolean Literal
4. String Literal

**5. Type Conversion and Type Casting:**

Here's a **complete theory explanation** of the code in your public class \_5\_TypeConversion\_and\_TypeCasting with **clear concepts**, **code walkthrough**, and **best practices** in Java.

## 🔄 Type Conversion & Type Casting in Java (Theory)

In Java, converting one data type to another is a common operation. This process is categorized into two main types:

### 🔹 1. **Type Conversion (Widening Conversion / Implicit Casting)**

**Definition:**  
Widening conversion occurs **automatically** when a smaller data type is assigned to a larger data type. Since there's no risk of data loss, Java handles it implicitly.

#### 🔁 Widening Order:

byte → short → char → int → long → float → double

#### ✅ Characteristics:

* No data loss
* Performed automatically
* Also called **implicit casting** or **type conversion**

#### 🔍 Code Walkthrough:

byte b = 10;

short s = b; // byte to short

int i = s; // short to int

long l = i; // int to long

float f = l; // long to float

double d = f; // float to double

#### 🧾 Output:

Byte: 10

Short: 10

Int: 10

Long: 10

Float: 10.0

Double: 10.0

### 🔹 2. **Type Casting (Narrowing Conversion / Explicit Casting)**

**Definition:**  
Narrowing conversion is used when a **larger data type** is converted into a **smaller one**. This **must be done manually** using a cast operator, because there's potential for **data loss**.

#### 🔁 Narrowing Order:

double → float → long → int → char → short → byte

#### ✅ Characteristics:

* Possible data loss
* Must be explicitly cast
* Also known as **explicit casting** or **type casting**

#### 🔍 Code Walkthrough:

double d2 = 9.78;

float f2 = (float) d2; // double to float

long l2 = (long) f2; // float to long

int i2 = (int) l2; // long to int

short s2 = (short) i2; // int to short

byte b2 = (byte) s2; // short to byte

#### 🧾 Output:

Double: 9.78

Float: 9.78

Long: 9

Int: 9

Short: 9

Byte: 9

### 🔹 3. **Type Casting with Characters**

Characters in Java are represented using the **ASCII/Unicode** values. You can convert:

* char → int (gives ASCII code)
* int → char (gives character for the ASCII code)

#### 🔍 Example:

char c = 'A';

int charToInt = c; // 65 (ASCII)

char intToChar = (char) 65; // 'A'

#### 🧾 Output:

Character: A

Character to Int: 65

Int to Character: A

### 🔹 4. **Type Conversion with Strings (Parsing)**

Strings cannot be **type-cast** directly into numeric types, but you can **parse** them using built-in methods.

#### 🔍 Common Methods:

* Integer.parseInt("123") → returns int
* Double.parseDouble("123.45") → returns double

#### 🔍 Example:

String str = "123";

int strToInt = Integer.parseInt(str);

double strToDouble = Double.parseDouble(str);

#### 🧾 Output:

String to Int: 123

String to Double: 123.0

### 🔹 5. **Type Conversion with Booleans**

**⚠️ Important:**  
Java **does not allow** converting between boolean and other types.

boolean bool = true;

// int x = (int) bool; ❌ Not allowed

#### 🧾 Output:

Boolean: true

## 🔑 Summary Table: Parsing vs. Type Conversion

| **Feature** | **Parsing** | **Type Conversion (Casting)** |
| --- | --- | --- |
| Purpose | Convert **String → Data Type** | Convert **One Type → Another** |
| Use | Reading input (files, APIs, user) | Working with numeric types |
| Method | parseXxx() e.g. Integer.parseInt | Implicit or explicit casting |
| Data Loss Risk | No | Possible (in narrowing) |
| Example | "123" → 123 | 10.5 → 10 (double to int) |

## ✅ Best Practices

* Use **widening conversions** whenever possible to avoid data loss.
* Use **explicit casting** only when you’re sure of the value range.
* Use **parsing** methods to read numeric values from string-based input.
* Be cautious with **booleans** — they cannot be cast to numbers or strings.

Here's the **theory explanation** of **Type Promotion** in Java, using your code example:

### 🔹 **What is Type Promotion?**

**Type Promotion** refers to the automatic **conversion of smaller data types to larger types** during **arithmetic or binary operations** to prevent data loss and ensure accurate results.

### 🔹 **Why It Happens?**

When performing operations involving **different data types**, Java **promotes** the smaller type to the **larger type** so that both operands are of the **same type**, ensuring:

* No loss of precision
* Correct mathematical results

### 🔍 Example Code:

int x = 10;

float y = 5.5f;

float result = x + y; // int is promoted to float

### ⚙️ What happens internally?

* x is of type int
* y is of type float
* Since float is a larger type than int, the value of x is **promoted** to float automatically.
* So the operation becomes:  
  float result = (float)x + y;

### 🧾 Output:

Type Promotion:

Int: 10

Float: 5.5

Result of Int + Float: 15.5

### 🔹 Type Promotion Hierarchy in Java (Low → High):

byte → short → int → long → float → double

Note: char is also promoted to int when used in arithmetic expressions.

### 🔹 More Examples:

byte a = 40;

byte b = 50;

byte c = 100;

int result = a \* b / c; // byte promoted to int during arithmetic

char ch = 'A';

int ascii = ch + 1; // 'A' is 65, so result is 66

### 🚫 Important Notes:

* **byte, short, and char** are always promoted to **int** during arithmetic operations.
* You **cannot assign** the result of an expression involving byte + byte directly into a byte without casting.

### ✅ Best Practices:

* Be aware of **type promotion** to avoid unexpected results.
* Always **check result types** in mixed-type expressions.
* If needed, use **explicit casting** to control the final result type.