

# Practical 1: Handling Various Data Types

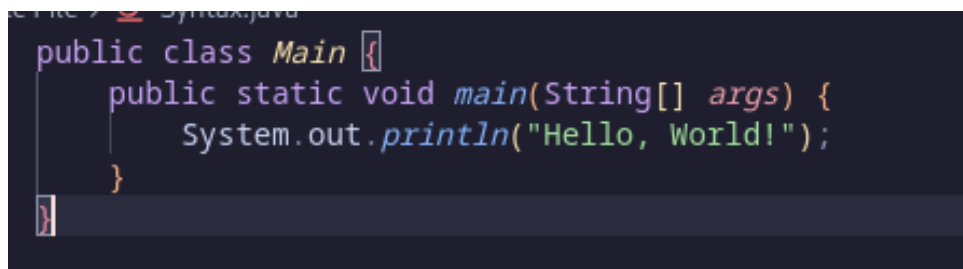
## 1 Introduction to Java

JAVA was developed by James Gosling at Sun Microsystems Inc in May 1995 and later acquired by Oracle Corporation. It is a simple programming language. Java makes writing, compiling, and debugging programming easy. It helps to create reusable code and modular programs. Java is a class-based, object-oriented programming language and is designed to have as few implementation dependencies as possible. A general-purpose programming language made for developers to write once run anywhere that is compiled Java code can run on all platforms that support Java. Java applications are compiled to byte code that can run on any Java Virtual Machine. The syntax of Java is similar to C/C++.

Java is widely used for developing applications for desktop, web, and mobile devices. Java is known for its simplicity, robustness, and security features, making it a popular choice for enterprise-level applications.

## 2 Java Syntax

Java syntax is the set of rules defining how a Java program is written and interpreted.

A screenshot of a code editor showing a Java program. The code is: 

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

Figure 1: Basic Syntax of Java Program

### **public class Main**

- **public:** An access modifier indicating that the class is accessible from other classes.
- **class:** A keyword used to define a class in Java.
- **Main:** The name of the class. By convention, class names in Java start with an uppercase letter.

**public static void main(String[] args)**

- **static:** A keyword indicating that the method belongs to the class, not to instances of the class. It can be called without creating an object of the class.
- **void:** The return type of the method, indicating that it does not return any value.
- **main:** The name of the method. This is the entry point of any Java application.
- **String[] args:** An array of String arguments passed to the method. These are command-line arguments.

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

- **System:** A built-in class in the `java.lang` package.
- **out:** A static field in the **System** class, which is an instance of **PrintStream**.
- **println:** A method of **PrintStream** that prints a message to the standard output (usually the console) followed by a newline.
- **"Hello, World!":** A string literal that is the message to be printed.

## 3 Variables in Java

Variables are containers for storing data values. In Java, every variable must be declared before it is used. A variable declaration includes the data type followed by the variable name. Java supports different types of variables, including:

- **Local Variables:** Declared inside a method and accessible only within that method.
- **Instance Variables:** Declared inside a class but outside any method. They are accessible from any method in the class.
- **Static Variables:** Declared with the `static` keyword. These are shared among all instances of the class.

## 4 Data Types in Java

Java has two categories of data types: **Primitive Data Types** and **Reference/Object Data Types**.

## # Primitive Data Types

Primitive data types are the most basic data types available in Java.

A screenshot of a Java IDE window titled 'Data.java'. The code defines a public class 'Data' with a static void main method. Inside the main method, several primitive variables are declared and assigned values: an integer 'myNum' (5), a float 'myFloatNum' (5.99f), a double 'myDoubNum' (5.9999d), a char 'myLetter' ('D'), a boolean 'myBool' (true), and a String 'myText' ('Hello'). Each assignment is followed by a comment indicating the variable's type. Below the declarations, the program uses 'System.out.println' to print the values of each variable. The IDE interface includes a menu bar (File, Edit, View, Run, Debug, Window, Help) and a toolbar with icons for running and debugging the code. The code is color-coded: keywords in blue, literals in red, and identifiers in green.

```
File > Edit > View > Run > Debug > Window > Help
public class Data {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        int myNum = 5; // Integer (whole number)
        float myFloatNum = 5.99f;
        double myDoubNum = 5.9999d; // Floating point number
        char myLetter = 'D'; // Character
        boolean myBool = true; // Boolean
        String myText = "Hello"; // String

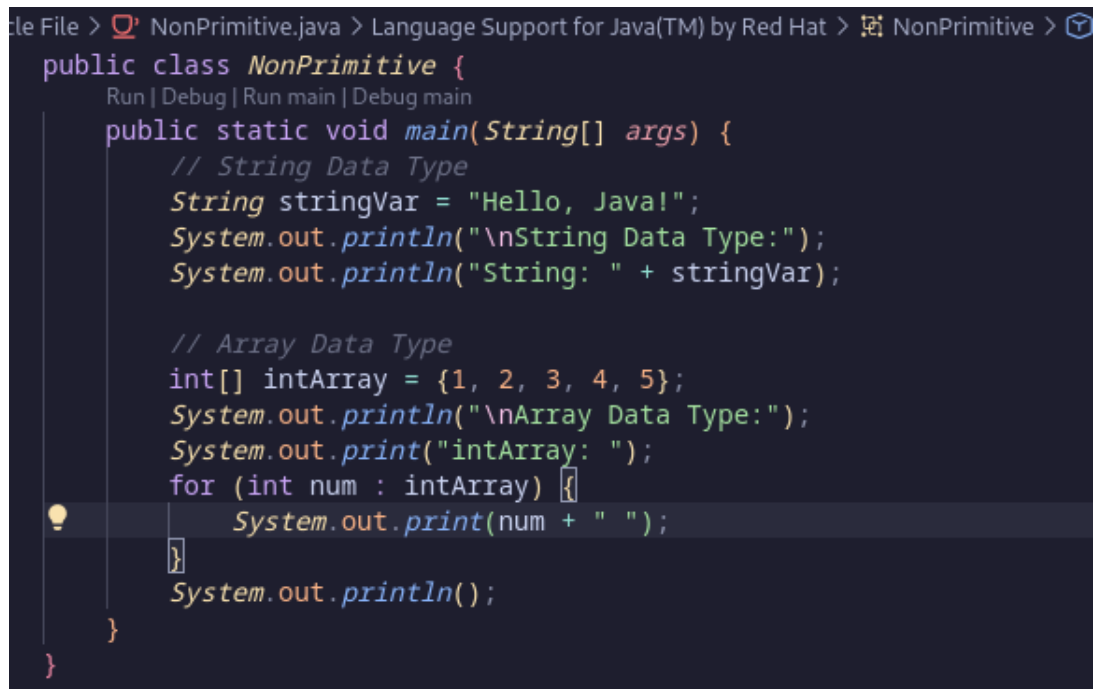
        System.out.println(myNum);
        System.out.println(myFloatNum);
        System.out.println(myDoubNum);
        System.out.println(myLetter);
        System.out.println(myBool);
        System.out.println(myText);
    }
}
```

Figure 2: Primitive Data types in Java

- **byte**: 8-bit signed integer. Range: -128 to 127.
- **short**: 16-bit signed integer. Range: -32,768 to 32,767.
- **int**: 32-bit signed integer. Range:  $-2^{31}$  to  $2^{31}-1$ .
- **long**: 64-bit signed integer. Range:  $-2^{63}$  to  $2^{63}-1$ .
- **float**: 32-bit floating-point number.
- **double**: 64-bit floating-point number.
- **char**: 16-bit Unicode character.
- **boolean**: Represents two values: true and false.

## # Non Primitive Data Types

Reference types in Java are Strings and arrays:



```
File > NonPrimitive.java > Language Support for Java(TM) by Red Hat > NonPrimitive >
public class NonPrimitive {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        // String Data Type
        String stringVar = "Hello, Java!";
        System.out.println("\nString Data Type:");
        System.out.println("String: " + stringVar);

        // Array Data Type
        int[] intArray = {1, 2, 3, 4, 5};
        System.out.println("\nArray Data Type:");
        System.out.print("intArray: ");
        for (int num : intArray) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

Figure 3: Non-Primitive Data types in Java

- **Strings:** Sequences of characters.
- **Arrays:** Containers that hold multiple values of the same type.

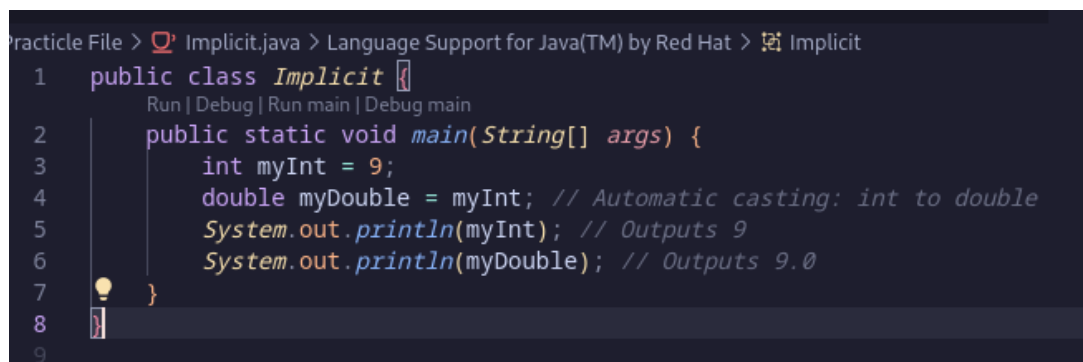
## Practical 2: Type casting

Type casting is when you assign a value of one primitive data type to another type. There are two types of type casting, implicit typecasting and explicit typecasting which are explained below:

### 1 Implicit Type Casting

Implicit type casting is done automatically when passing a smaller size type to a larger size type.

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



The screenshot shows a Java file named 'Implicit.java' in an IDE. The code defines a public class 'Implicit' with a main method. Inside the main method, an 'int' variable 'myInt' is assigned the value 9. Then, a 'double' variable 'myDouble' is assigned the value of 'myInt'. The IDE's tooltip for the assignment of 'myDouble' shows the text 'Automatic casting: int to double'. The code then prints 'myInt' (outputting 9) and 'myDouble' (outputting 9.0).

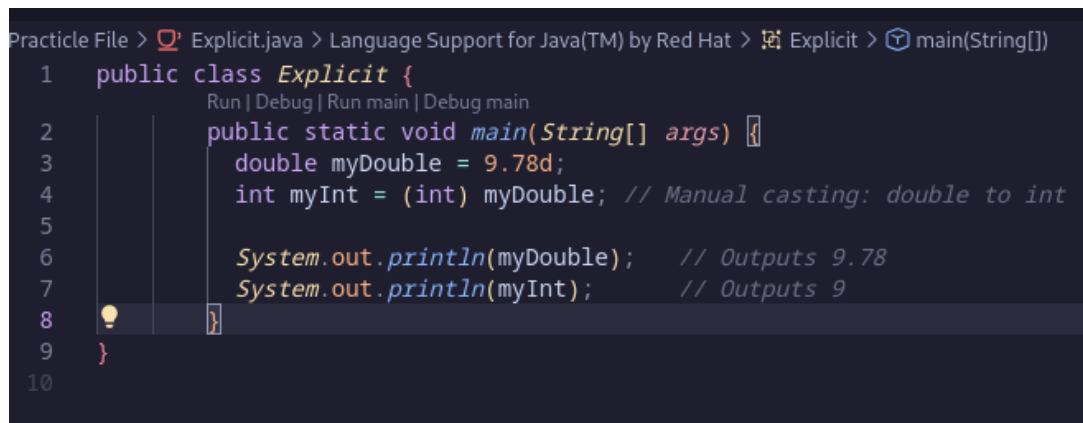
```
1 public class Implicit {  
2     public static void main(String[] args) {  
3         int myInt = 9;  
4         double myDouble = myInt; // Automatic casting: int to double  
5         System.out.println(myInt); // Outputs 9  
6         System.out.println(myDouble); // Outputs 9.0  
7     }  
8 }  
9
```

Figure 4: Implicit Type Conversion

### 2 Explicit Type Casting

Explicit type casting must be done manually by placing the type in parentheses () in front of the value.

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



The screenshot shows a Java file named 'Explicit.java' in an IDE. The code defines a public class 'Explicit' with a main method. Inside the main method, a 'double' variable 'myDouble' is assigned the value 9.78d. Then, an 'int' variable 'myInt' is assigned the value of 'myDouble', with a tooltip showing 'Manual casting: double to int'. The code then prints 'myDouble' (outputting 9.78) and 'myInt' (outputting 9).

```
1 public class Explicit {  
2     public static void main(String[] args) {  
3         double myDouble = 9.78d;  
4         int myInt = (int) myDouble; // Manual casting: double to int  
5  
6         System.out.println(myDouble); // Outputs 9.78  
7         System.out.println(myInt); // Outputs 9  
8     }  
9 }  
10
```

Figure 5: Explicit Type Conversion

## Practical 3: Addition Of Two Numbers

### 1 Add Two Numbers by taking input from the user

# Code:

```
Practicle File > AddNum.java > Language Support for Java(TM) by Red Hat > AddNum > main(String[])
1  import java.util.Scanner;
2
3  public class AddNum {
    Run | Debug | Run main | Debug main
4      public static void main(String[] args) {
5          System.out.println("# Adding Two Numbers #");
6          System.out.println("Taking input for two numbers: \n");
7          Scanner sc = new Scanner(System.in);    Resource leak: 'sc' is never closed
8          System.out.println("Enter the first number: ");
9          int a = sc.nextInt();
10         System.out.println("Enter the second number: ");
11         int b = sc.nextInt();
12         System.out.println("Sum is: ");
13         int sum = a + b;
14         System.out.println(sum);
15     }
16 }
17
```

Figure 6: Code of addition of two numbers

# Output:

```
mayank in fed in ~/Documents/Java/Practicle File | took 5ms
> java AddNum.java
# Adding Two Numbers #
Taking input for two numbers:

Enter the first number:
10
Enter the second number:
20
Sum is:
30
mayank in fed in ~/Documents/Java/Practicle File | took 7s
>
```

Figure 7: output of addition of two numbers