

# Practical 1: Handling Various Data Types

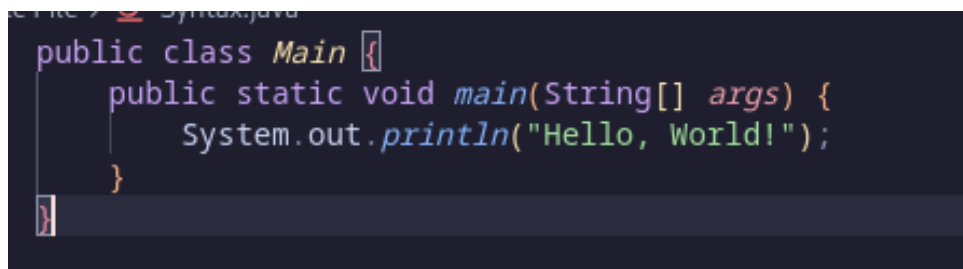
## 1 Introduction to Java

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.



```
File 7 Syntax.java
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.



```
File > Data.java > ...
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

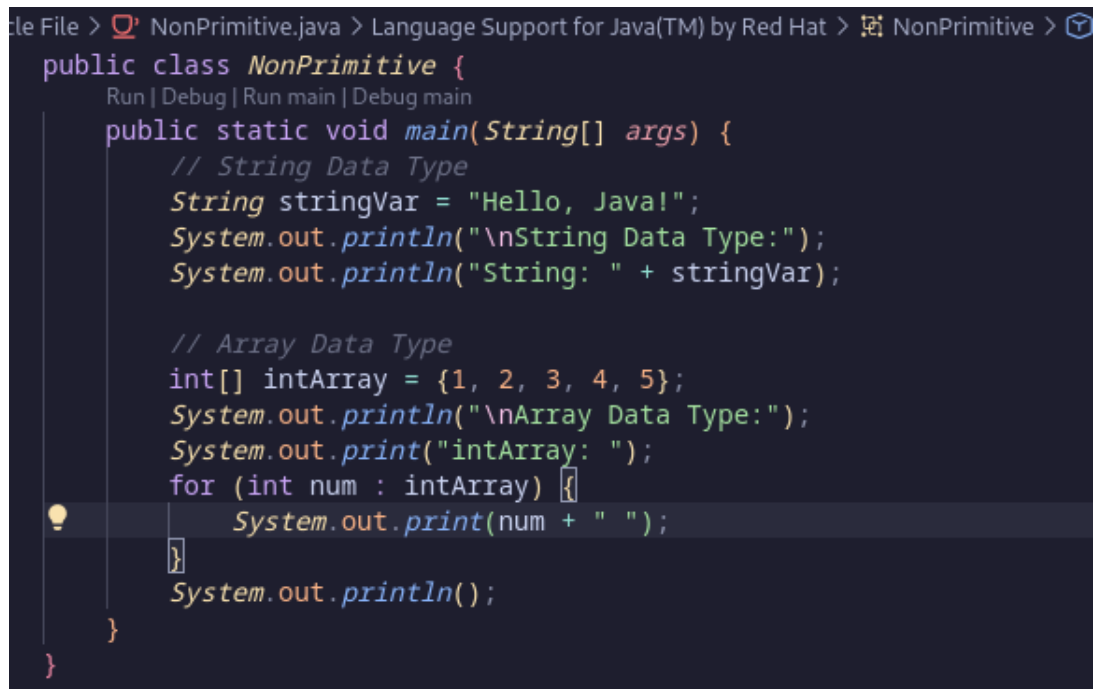
        System.out.println(myNum);
        System.out.println(myFloatNum);
        System.out.println(myDoubNum);
        System.out.println(myLetter);
        System.out.println(myBool);
        System.out.println(myText);
    }
}
```

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.

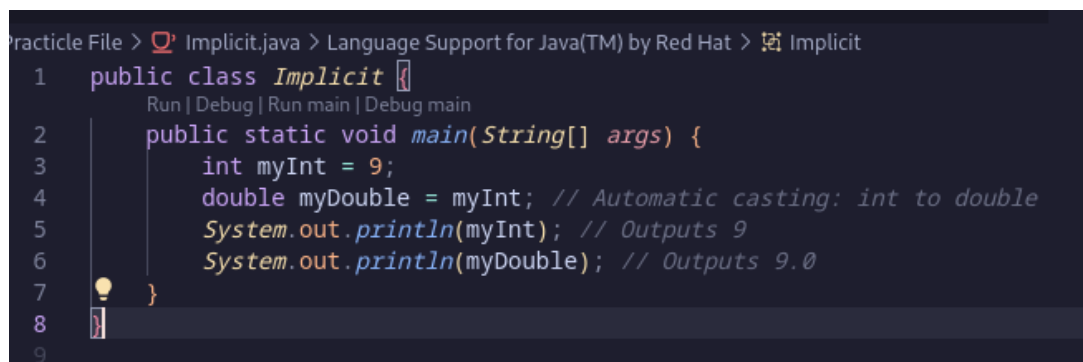
## Practical 2: Type casting

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



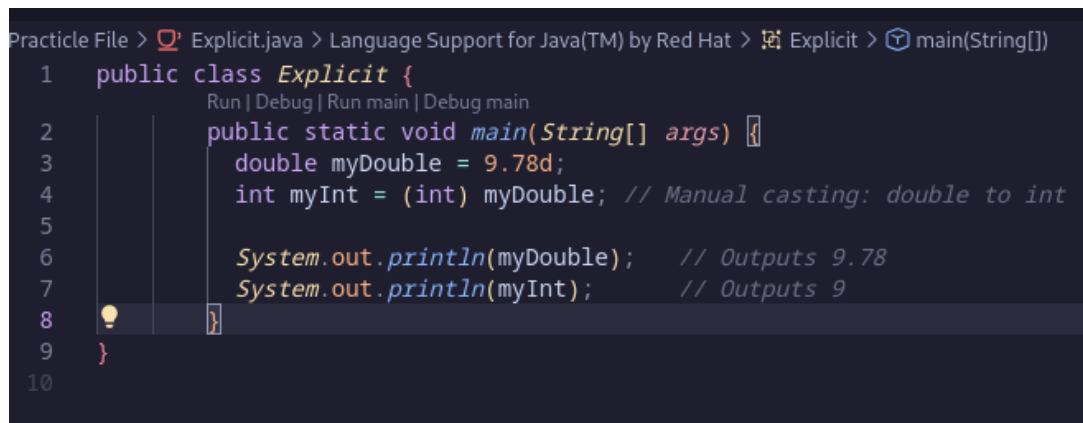
```
Practicle File > Implicit.java > Language Support for Java(TM) by Red Hat > Implicit
1 public class Implicit {
2     public static void main(String[] args) {
3         int myInt = 9;
4         double myDouble = myInt; // Automatic casting: int to double
5         System.out.println(myInt); // Outputs 9
6         System.out.println(myDouble); // Outputs 9.0
7     }
8 }
```

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



```
Practicle File > Explicit.java > Language Support for Java(TM) by Red Hat > Explicit > main(String[])
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 }
```

Figure 5: Explicit Type Conversion

# Practical 3: Array 1D and 2D

## 1 1-Dimensional Array

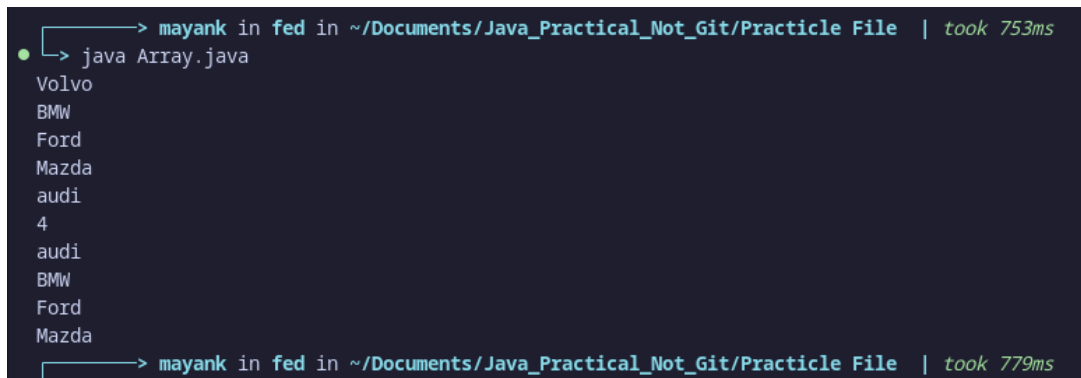
Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

# Code:

```
Practicle File > Array.java > Language Support for Java(TM) by Red Hat > Array > main(String[])
1 // Program for 1 Dimensional Array
2
3 public class Array {
4     Run | Debug | Run main | Debug main
5     public static void main(String[] args) {
6         // 1D Array
7         // Declaration and initialization of an array
8         String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
9         // int[] nyNum = {10, 20, 30, 40};
10
11         // Accessing the value of an array
12         System.out.println(cars[0]); // Volvo
13         System.out.println(cars[1]); // BMW
14         System.out.println(cars[2]); // Ford
15         System.out.println(cars[3]); // Mazda
16
17         // Changing an element of an array
18         cars[0] = "audi";
19         System.out.println(cars[0]); // Output will change to Audi from Vo.
20
21         // Length of an array
22         System.out.println(cars.length);
23
24         // Loop through an array
25         for (String arr : cars) {
26             System.out.println(arr);
27         }
28     }
29 }
```

Figure 6: 1 Dimensional Array

## # Output:



```
> mayank in fed in ~/Documents/Java_Practical_Not_Git/Practicle File | took 753ms
• java Array.java
Volvo
BMW
Ford
Mazda
audi
4
audi
BMW
Ford
Mazda
> mayank in fed in ~/Documents/Java_Practical_Not_Git/Practicle File | took 779ms
```

Figure 7: output of 1-D array

## 2 Multi Dimensional Array

A multidimensional array is an array of arrays. Multidimensional arrays are useful when you want to store data as a tabular form, like a table with rows and columns.

## # Code:



```
Practicle File > Array2D.java > Language Support for Java(TM) by Red Hat > Array2D > main(String[])
1 // Program for Multi-Dimensional Array
2
3 public class Array2D {
4     Run | Debug | Run main | Debug main
5     public static void main(String[] args) {
6         // 2D Array
7         // Declaration and initialization of an 2D array
8         int[][] my2DArr = {{10, 20, 30, 40}, {50, 60, 70}};
9
10        // Accessing Elements of array
11        System.out.println(my2DArr[0][0]); // 10
12        System.out.println(my2DArr[1][2]); // 70
13
14        // change element of array
15        my2DArr[0][0] = 100;
16        System.out.println(my2DArr[0][0]); //100
17
18        // Loop through a multi dimensional array
19        System.out.println("Looping through an array");
20        for (int[] row : my2DArr) {
21            for(int i : row) {
22                System.out.println(i);
23            }
24        }
25    }
26}
```

Figure 8: Multi Dimensional Array

## # Output:

```
> mayank in fed in ~/Documents/Java_Practical_Not_Git/Practicle File | took 754ms
● → java Array2D.java
10
70
100
Looping through an array
100
20
30
40
50
60
70
○ → > mayank in fed in ~/Documents/Java_Practical_Not_Git/Practicle File | took 866ms
○ →
```

Figure 9: output of Multi-D array



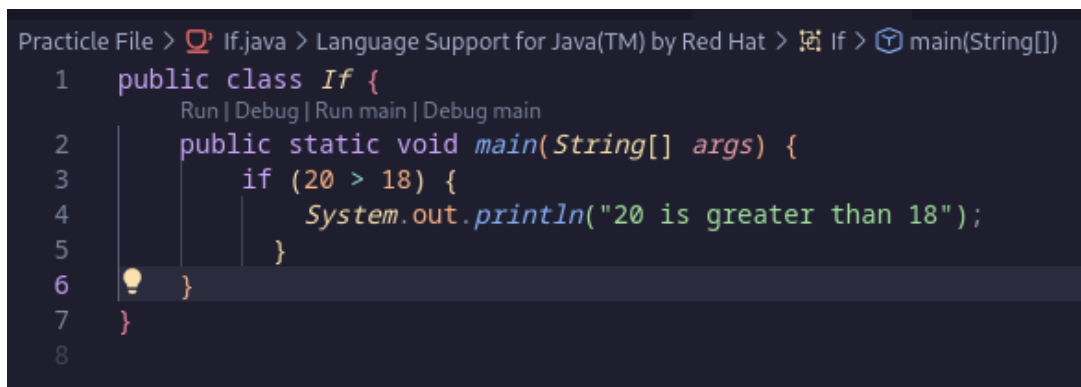
## Practical 4: Various Control Structures

In Java, control structures are constructs that dictate the order in which statements are executed in a program. They enable conditional execution, looping, and altering the normal sequential flow of control.

### 1 The IF statement

Use the if statement to specify a block of Java code to be executed if a condition is true.

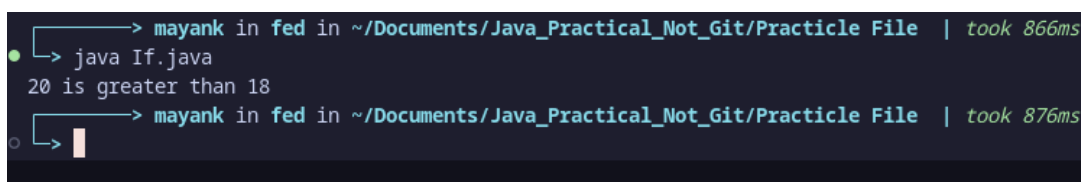
# Code:



```
Practicle File > If.java > Language Support for Java(TM) by Red Hat > If > main(String[])
1  public class If {
    Run | Debug | Run main | Debug main
2      public static void main(String[] args) {
3          if (20 > 18) {
4              System.out.println("20 is greater than 18");
5          }
6      }
7  }
8
```

Figure 10: if statement

# Output:



```
> mayank in fed in ~/Documents/Java_Practical_Not_Git/Practicle File | took 866ms
● > java If.java
20 is greater than 18
○ >
```

Figure 11: output of if statement

### 2 The IF-Else statement

Executes one block of code if its condition evaluates to true, and another block of code if it evaluates to false.

## # Code:

```
Practicle File > IfElse.java > ...  
1 // IF-ELSE Statement  
2 public class IfElse {  
    Run | Debug | Run main | Debug main  
3     public static void main(String[] args) {  
4         int time = 20;  
5         if (time < 18) {  
6             System.out.println("Good day.");  
7         } else {  
8             System.out.println("Good evening.");  
9         }  
10    }  
11 }  
12
```

Figure 12: if-else statement

## # Output:

```
> mayank in fed in ~/Documents/Java_Practical_Not_Git/Practicle File | took 881ms  
• > java IfElse.java  
Good evening.  
> mayank in fed in ~/Documents/Java_Practical_Not_Git/Practicle File | took 741ms
```

Figure 13: output of if-else statement

## Practical 4: 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 {
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);
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 14: 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
○ >
```

Figure 15: output of addition of two numbers