

ASSIGNMENT - 2

Type Conversion

Q) What is the difference between implicit and explicit type conversion in Java?

Ans-

1. Implicit Type Conversion (Widening Conversion):

- Happens automatically
- No data loss occurs
- Converting smaller data type to larger data type
- No special syntax required

Eg:

```
byte b = 100;
```

```
int i = b; // byte to int
```

```
int x = 100;
```

```
double d = x; // int to double
```

```
char ch = 'A'; int ascii = ch;
```

2. Explicit Type Conversion:

- Done manually by programmer
- Potential data loss may occur
- Converting larger data type to smaller data type
- Requires casting operator ()

Implementation:

```
1
2 public class TypeConversion {
3
4     public static void main(String[] args) {
5         // Implicit conversion (int to float)
6         int integerValue = 25;
7         float floatValue = integerValue; // implicit conversion
8         System.out.println("Implicit Conversion:");
9         System.out.println("Integer value: " + integerValue);
10        System.out.println("Float value after implicit conversion: " + floatValue);
11
12        // Explicit conversion (double to int)
13        double doubleValue = 100.75;
14        int convertedInt = (int)doubleValue; // explicit casting
15        System.out.println("\nExplicit Conversion:");
16        System.out.println("Double value: " + doubleValue);
17        System.out.println("Integer value after explicit conversion: " + convertedInt);
18    }
19
20 }
21
```

OUTPUT:

```
<terminated> TypeConversion [Java Application] C:\Users\sagar\.p2\pool\plugins\o
Implicit Conversion:
Integer value: 25
Float value after implicit conversion: 25.0

Explicit Conversion:
Double value: 100.75
Integer value after explicit conversion: 100
```

Q) What are literals in Java? List different types with examples.

ANS- In Java, literals are fixed values that can be assigned to variables. They are constant values that appear directly in the program. Here are the different types of literals in Java with examples:

Integer Literals

// Decimal (base 10)

```
int decimal = 42;
```

// Octal (base 8, starts with 0)

```
int octal = 052; // equals 42 in decimal
```

// Hexadecimal (base 16, starts with 0x)

```
int hex = 0x2A; // equals 42 in decimal
```

// Binary (base 2, starts with 0b)

```
int binary = 0b101010; // equals 42 in decimal
```

// Long literal (ends with L or l)

```
long longNum = 42L;
```

Floating-Point Literals

// Float (ends with f or F)

```
float f1 = 3.14f;
```

```
float f2 = 3.14F;
```

// Double (can end with d or D, but not required)

```
double d1 = 3.14;
```

```
double d2 = 3.14d;
```

```
double d3 = 3.14D;
```

```
// Scientific notation
```

```
double d4 = 3.14e2; //  $3.14 \times 10^2 = 314.0$ 
```

Character Literals

Represent single characters enclosed in single quotes (').

- **Example:** `char letter = 'A';`

String Literals

Represent sequences of characters enclosed in double quotes (").

- **Example:** `String greeting = "Hello, Java!";`

Boolean Literals

Represent true or false values.

```
boolean b1 = true;
```

```
boolean b2 = false;
```

Null Literal

Used to indicate that an object reference does not point to any memory location.

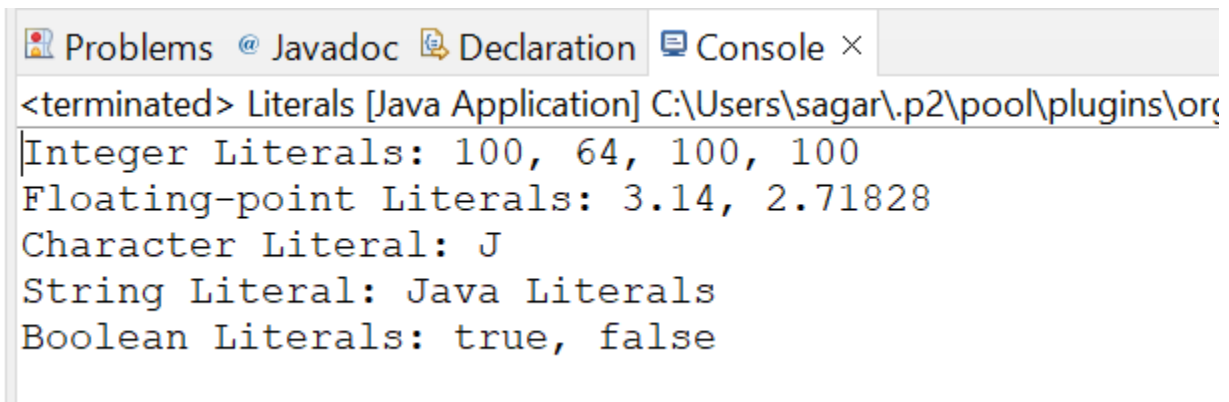
```
String str = null;
```

```
Object obj = null;
```

Implementation:

```
2 public class Literals {
3
4     public static void main(String[] args) {
5         // Integer Literals
6         int decimal = 100;
7         int octal = 0100; // 64 in decimal
8         int hex = 0x64;    // 100 in decimal
9         int binary = 0b1100100; // 100 in decimal
10
11        // Floating-point Literals
12        float floatVal = 3.14f;
13        double doubleVal = 2.71828;
14
15        // Character Literal
16        char character = 'J';
17
18        // String Literal
19        String text = "Java Literals ";
20
21        // Boolean Literals
22        boolean isJavaFun = true;
23        boolean isCodingTough = false;
24
25        // Print all the literals
26        System.out.println("Integer Literals: " + decimal + ", " + octal + ", " + hex + ", " + binary);
27        System.out.println("Floating-point Literals: " + floatVal + ", " + doubleVal);
28        System.out.println("Character Literal: " + character);
29        System.out.println("String Literal: " + text);
30        System.out.println("Boolean Literals: " + isJavaFun + ", " + isCodingTough);
31    }
32 }
```

Output:



The screenshot shows an IDE window with tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the output of the Java application. The output consists of five lines, each corresponding to a category of literals defined in the code: Integer Literals, Floating-point Literals, Character Literal, String Literal, and Boolean Literals. The values are printed in a single line for each category, separated by commas.

```
<terminated> Literals [Java Application] C:\Users\sagar\.p2\pool\plugins\org
Integer Literals: 100, 64, 100, 100
Floating-point Literals: 3.14, 2.71828
Character Literal: J
String Literal: Java Literals
Boolean Literals: true, false
```

3. Assignment Operator:

The purpose of assignment operators in Java is to assign or modify values in variables. Here's a detailed explanation of their purposes and uses:

1. Basic Assignment (=)

// Assigns a value to a variable

```
int x = 10;
```

```
String name = "John";
```

2. Compound Assignment Operators

```
int num = 5;
```

// Addition and assignment

```
num += 3; // same as: num = num + 3
```

// Subtraction and assignment

```
num -= 2; // same as: num = num - 2
```

// Multiplication and assignment

```
num *= 4; // same as: num = num * 4
```

// Division and assignment

```
num /= 2; // same as: num = num / 2
```

// Modulus and assignment

```
num %= 3; // same as: num = num % 3
```

3. Main Purposes:

- Value Assignment

```
int age = 25; // Assigns value 25 to variable age
```

- Variable Modification

```
int count = 0;
```

```
count += 1; // Increments count by 1
```

- Multiple Assignment

```
int x, y, z;
```

```
x = y = z = 100; // Assigns 100 to all three variables
```

- Expression Results

```
int result = (x + y) * z; // Assigns result of expression
```

4. Benefits:

- Makes code more concise
- Reduces typing and potential errors
- Improves code readability
- Combines operations efficiently

Implementation:

```
1
2 public class Assignment {
3
4     public static void main(String[] args) {
5         int num = 10;
6         System.out.println("Initial value: " + num);
7
8         num += 5;
9         System.out.println("After +=: " + num);
10
11        num -= 3;
12        System.out.println("After -=: " + num);
13
14        num *= 2;
15        System.out.println("After *=: " + num);
16
17        num /= 4;
18        System.out.println("After /=: " + num);
19
20        num %= 3;
21        System.out.println("After %=: " + num);
22    }
23 }
24
```

OUTPUT:

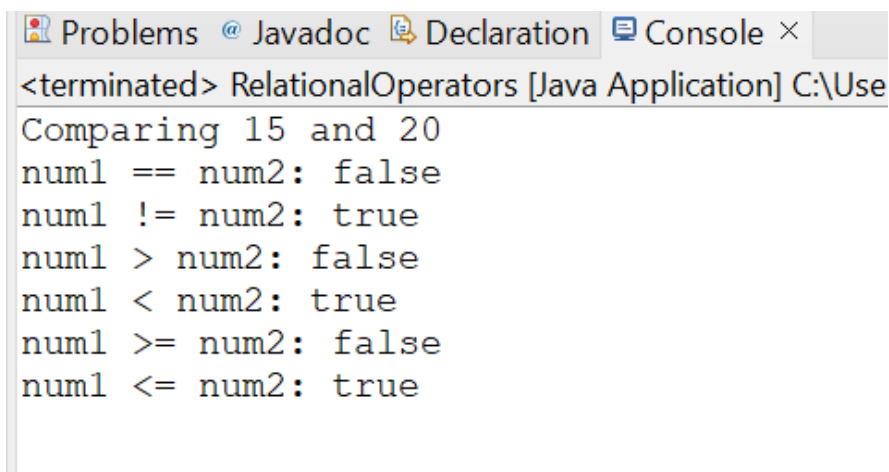
```
<terminated> Assignment [Java Application] C:\Users\sagar\
Initial value: 10
After +=: 15
After -=: 12
After *=: 24
After /=: 6
After %=: 0
```


4.Relational Operators:Relational operators in Java are used to compare two values or expressions. These operators return a boolean result (**true** or **false**) based on the relationship between the operands. They are commonly used in conditional statements like **if**, **while**, and **for**.

Program to Compare Two Integers Using Relational Operators

```
1
2 public class RelationalOperators {
3
4     public static void main(String[] args) {
5         int num1 = 15;
6         int num2 = 20;
7
8         // Using relational operators
9         System.out.println("Comparing " + num1 + " and " + num2);
10
11        // Equal to
12        System.out.println("num1 == num2: " + (num1 == num2));
13
14        // Not equal to
15        System.out.println("num1 != num2: " + (num1 != num2));
16
17        // Greater than
18        System.out.println("num1 > num2: " + (num1 > num2));
19
20        // Less than
21        System.out.println("num1 < num2: " + (num1 < num2));
22
23        // Greater than or equal to
24        System.out.println("num1 >= num2: " + (num1 >= num2));
25
26        // Less than or equal to
27        System.out.println("num1 <= num2: " + (num1 <= num2));
28    }
29 }
```

OUTPUT:



The screenshot shows an IDE window with tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the output of the Java application. The output is as follows:

```
<terminated> RelationalOperators [Java Application] C:\Use
Comparing 15 and 20
num1 == num2: false
num1 != num2: true
num1 > num2: false
num1 < num2: true
num1 >= num2: false
num1 <= num2: true
```

5. Logical Operator

Logical operators in Java are used to perform logical operations on boolean expressions. The main logical operators are:

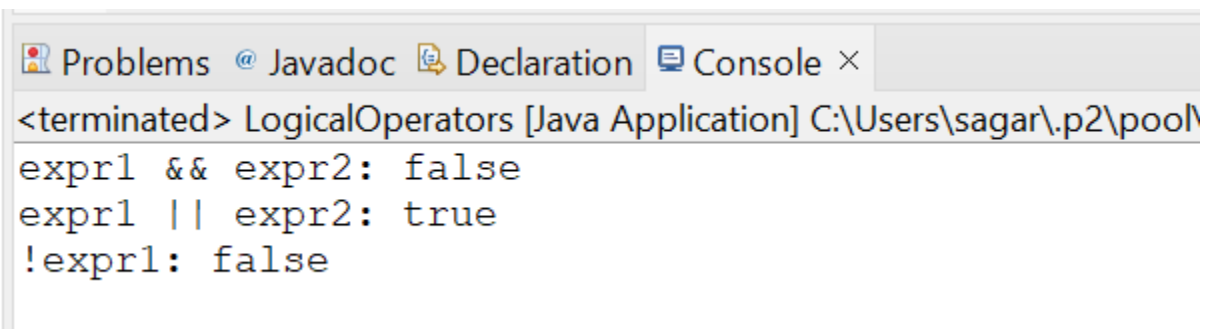
- **&&** (Logical AND): Returns **true** if both operands are true.
- **||** (Logical OR): Returns **true** if at least one operand is true.
- **!** (Logical NOT): Reverses the logical state of its operand.

Difference between **&&** and **||**:

- **&&** checks if both conditions are true; if one is false, the entire expression is false.
- **||** checks if either condition is true; the expression is true even if one condition is true.

```
1
2 public class LogicalOperators {
3
4     public static void main(String[] args) {
5         boolean expr1 = true;
6         boolean expr2 = false;
7
8         System.out.println("expr1 && expr2: " + (expr1 && expr2));
9         System.out.println("expr1 || expr2: " + (expr1 || expr2));
10        System.out.println("!expr1: " + (!expr1));
11    }
12 }
13
14
15
```

OUTPUT:



The screenshot shows an IDE window with tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the output of the Java application. The output consists of three lines: "expr1 && expr2: false", "expr1 || expr2: true", and "!expr1: false".

```
<terminated> LogicalOperators [Java Application] C:\Users\sagar\.p2\pool\
expr1 && expr2: false
expr1 || expr2: true
!expr1: false
```

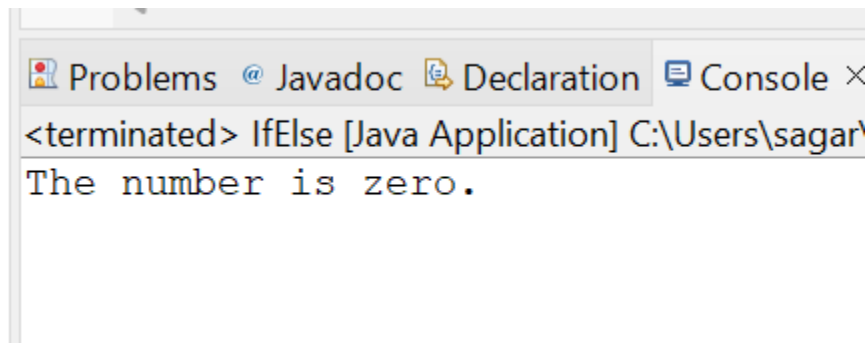
6. If-Else Statement

The syntax for an if-else statement in Java is:

```
if (condition) {  
    // code if condition is true  
}  
else {  
    // code if condition is false  
}
```

```
1  
2 public class IfElse {  
3     public static void main(String[] args) {  
4         int number = 0;  
5  
6         if (number > 0) {  
7             System.out.println("The number is positive.");  
8         } else if (number < 0) {  
9             System.out.println("The number is negative.");  
10        } else {  
11            System.out.println("The number is zero.");  
12        }  
13    }  
14 }  
15  
16
```

Output:



7.If-Else-If Ladder:

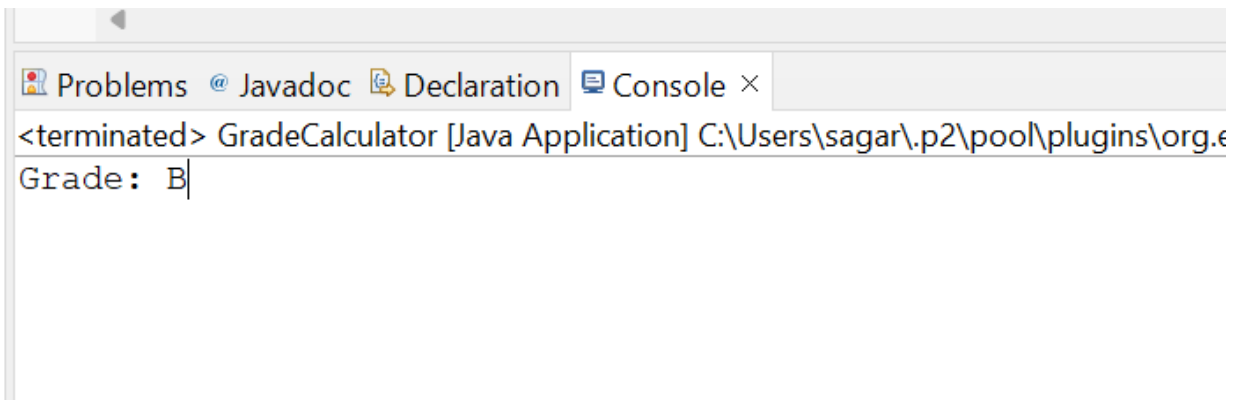
An if-else-if ladder should be used when multiple conditions need to be checked sequentially, where only one block of code will execute. This is more efficient and easier to read than multiple independent if statements.

```

1
2 public class GradeCalculator {
3
4
5     public static void main(String[] args) {
6         int marks = 85;
7
8         if (marks >= 90) {
9             System.out.println("Grade: A");
10        } else if (marks >= 80) {
11            System.out.println("Grade: B");
12        } else if (marks >= 70) {
13            System.out.println("Grade: C");
14        } else if (marks >= 60) {
15            System.out.println("Grade: D");
16        } else {
17            System.out.println("Grade: Fail");
18        }
19    }
20 }
21

```

OUTPUT:



8. Ternary Operator

The ternary operator is a concise way to write an **if-else** statement in Java.

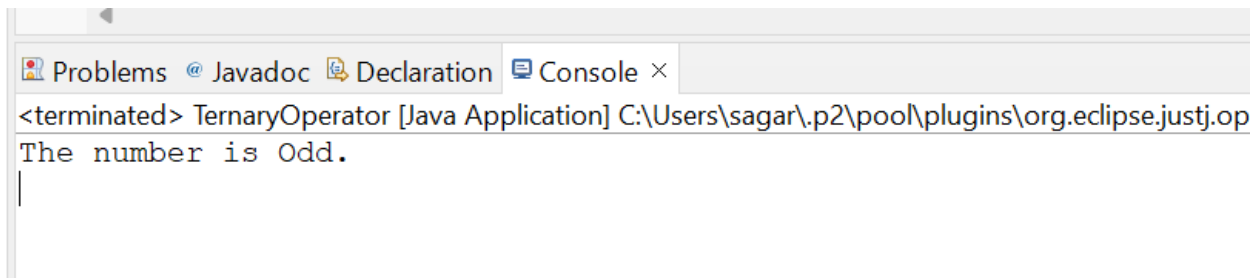
Syntax: `result = (condition) ? valueIfTrue : valueIfFalse;`

Difference from if-else:

- The ternary operator is a single-line expression, while if-else uses multiple lines.
- Ternary is suitable for simple conditions, while if-else handles more complex logic.

```
1
2 public class TernaryOperator {
3
4     public static void main(String[] args) {
5         int number = 5;
6
7         String result = (number % 2 == 0) ? "Even" : "Odd";
8         System.out.println("The number is " + result + ".");
9     }
10 }
11
```

OUTPUT :



The screenshot shows the Eclipse IDE's console window. The title bar includes tabs for 'Problems', 'Javadoc', 'Declaration', and 'Console'. The console output displays the result of running the Java application: '<terminated> TernaryOperator [Java Application] C:\Users\sagar.p2\pool\plugins\org.eclipse.justj.op' followed by the printed line 'The number is Odd.'.