Unit 01

Java Fundamentals, Data Types, Operators and Control Statements

1.2. Object Oriented Programming

Object-Oriented Programming (OOP) is a programming model that uses "objects" to design applications and computer programs. It utilizes several key concepts to create software that is modular, flexible, and easy to maintain.

- Classes and Objects
- Inheritance
- Polymorphism
- Encapsulation
- Abstraction

1.3. Java Development Kit

The Java Development Kit (JDK) is a software development environment used for developing Java applications and applets. It provides the tools needed to write, compile, debug, and run Java programs. Here's an overview of the key components and functionalities of the JDK.

1.5. Packages in Java

Packages in Java are a mechanism for organizing Java classes and interfaces into namespaces, providing modularity and encapsulation. They help manage the complexity of large software projects by grouping related classes and interfaces together.

1.6. Java's Data Types

Java supports various data types, which are classified into two categories: primitive data types and reference data types.

1.6.1 Integers

In Java, integers are a group of data types used to store whole numbers. They come in different sizes to accommodate various ranges of values and memory requirements. Java provides four types of integer data types: byte, short, int, and long

1.6.2 Characters

In Java, the char data type is used to represent single 16-bit Unicode characters. It can hold any character from the Unicode character set, allowing Java to support internationalization and localization. Characters are enclosed in single quotes (' '), and Java uses Unicode encoding internally, allowing for a wide range of characters to be represented.

1.6.3 Floating Point Types

In Java, floating-point types are used to represent real numbers (numbers with a fractional part). Java provides two floating-point types: float and double, each with different sizes and precision.

1.6.4 Strings

In Java, strings are objects that represent sequences of characters. They are widely used for storing and manipulating text data. Strings in Java are immutable, meaning once a string object is created, its value cannot be changed. Java provides a rich set of methods and operators for working with strings.

1.6.5 Arrays

In Java, an array is a data structure that stores a fixed-size sequential collection of elements of the same type. Arrays are used to store multiple values of the same data type under a single variable name. They provide a convenient way to access and manipulate a group of related data items.

1.6.6 The Boolean Types

In Java, the Boolean type represents a data type with two possible values: true and false. Booleans are commonly used for expressing conditions and controlling the flow of a program through conditional statements such as if, while, and for loops.

1.7. Literals

Java allows you to specify integer literals in hexadecimal (base 16), octal (base 8), and binary (base 2) formats.

1.7.1. Hex, Octal and Binary

Java allows you to specify integer literals in hexadecimal (base 16), octal (base 8), and binary (base 2) formats.

- **Hexadecimal:** Prefixed with 0x or 0X, followed by hexadecimal digits (0-9, A-F). Example: int hexValue = 0xA1B2;
- **Octal:** Prefixed with 0, followed by octal digits (0-7). Example: int octalValue = 0764;
- **Binary:** Prefixed with 0b or 0B, followed by binary digits (0 or 1). Example: int binaryValue = 0b101010;

1.7.2. Character Escape Sequences

Character escape sequences in Java are special sequences of characters that represent special characters or control characters. They are used within character and string literals.

- \n: Newline
- \t: Tab
- \r: Carriage return
- ': Single quote
- ": Double quote
- \: Backslash
- \uXXXX: Unicode character (where XXXX is a hexadecimal Unicode code point)

1.7.3. String Literals

String literals are sequences of characters enclosed in double quotes ("). They represent constant values that can be assigned to variables of type String.

Example: String greeting = "Hello, World!";

String literals can also be concatenated using the + operator:

String firstName = "John";

String lastName = "Doe";

```
String fullName = firstName + " " + lastName;
```

1.8. Variables and Constants

Variables and constants are fundamental concepts in programming languages like Java, used to store and manipulate data within a program.

Variables: Variables are named storage locations in computer memory that can hold data values. They can be of different types, such as integers, floating-point numbers, characters, or custom data types like objects. variables are used to store and manipulate data dynamically during program execution. They can be declared with a specific data type and assigned values that can change over time.

Constants: Constants are similar to variables but represent fixed values that do not change during the execution of a program. Constants are declared using the final keyword, indicating that their values cannot be modified once initialized.

1.9. Operators

Operators in Java are special symbols or keywords used to perform operations on operands, such as variables, constants, or expressions. They enable various computations, comparisons, and logical operations within a program.

Arithmetic Operators

Arithmetic operators perform mathematical operations on numeric operands.

- Addition (+): Adds two operands.
- Subtraction (-): Subtracts the second operand from the first.
- Multiplication (*): Multiplies two operands.
- Division (/): Divides the first operand by the second.
- Modulus (%): Returns the remainder of the division.

Relational Operators

Relational operators compare two operands and return a boolean result indicating the relationship between them.

- Equal to (==): Checks if two operands are equal.

- Not equal to (!=): Checks if two operands are not equal.
- Greater than (>): Checks if the first operand is greater than the second.
- Less than (<): Checks if the first operand is less than the second.
- Greater than or equal to (>=): Checks if the first operand is greater than or equal to the second.
- Less than or equal to (<=): Checks if the first operand is less than or equal to the second.

1.9. Operators

Operators in Java are special symbols or keywords used to perform operations on operands, such as variables, constants, or expressions. They enable various computations, comparisons, and logical operations within a program.

Arithmetic Operators

Arithmetic operators perform mathematical operations on numeric operands.

- Addition (+): Adds two operands.
- Subtraction (-): Subtracts the second operand from the first.
- Multiplication (*): Multiplies two operands.
- Division (/): Divides the first operand by the second.
- Modulus (%): Returns the remainder of the division.

```
int a = 10;
int b = 5;
int sum = a + b;
int difference = a - b;
int product = a * b;
int quotient = a / b;
int remainder = a % b;
```

Relational Operators

Relational operators compare two operands and return a boolean result indicating the relationship between them.

- Equal to (==): Checks if two operands are equal.
- Not equal to (!=): Checks if two operands are not equal.

- Greater than (>): Checks if the first operand is greater than the second.
- Less than (<): Checks if the first operand is less than the second.
- Greater than or equal to (>=): Checks if the first operand is greater than or equal to the second.
- Less than or equal to (<=): Checks if the first operand is less than or equal to the second.

```
int x = 10;
int y = 5;
boolean isEqual = (x == y);
boolean isGreaterThan = (x > y);
boolean isLessThan = (x < y);
```

Logical Operators

Logical operators perform logical operations on boolean operands.

- Logical AND (&&): Returns true if both operands are true.
- Logical OR (||): Returns true if at least one of the operands is true.
- Logical NOT (!): Returns true if the operand is false and vice versa.

Assignment Operators

Assignment operators assign values to variables.

- Assignment (=): Assigns the value of the right operand to the left operand.
- Addition assignment (+=): Adds the value of the right operand to the left operand.
- Subtraction assignment (-=): Subtracts the value of the right operand from the left operand.
- Multiplication assignment (*=): Multiplies the value of the left operand by the right operand.
- Division assignment (/=): Divides the value of the left operand by the right operand.
- Modulus assignment (%=): Assigns the remainder of the division of the left operand by the right operand to the left operand.

Bitwise Operators

Bitwise operators perform operations on individual bits of integer operands.

- Bitwise AND (&)
- Bitwise OR (|)
- Bitwise XOR (^)
- Bitwise NOT (~)
- Left shift (<<)
- Right shift (>>)
- Unsigned right shift (>>>)

1.10. Type Casting

Type casting in Java refers to the process of converting a value from one data type to another. It is necessary when assigning a value of one data type to a variable of another data type, especially when the target data type has a smaller range or precision than the source data type.

Implicit Casting (Widening Conversion)

Implicit casting, also known as widening conversion, occurs automatically when converting a smaller data type to a larger data type. There is no loss of information in this process.

```
int intValue = 10;

double doubleValue = intValue; // Implicit casting from int to double
```

Explicit Casting (Narrowing Conversion)

Explicit casting, also known as narrowing conversion, is performed manually when converting a larger data type to a smaller data type. It requires the use of a cast operator, and there may be a loss of information if the value cannot be represented accurately in the target data type.

```
double doubleValue = 10.5;

int intValue = (int) doubleValue; // Explicit casting from double to int
```

1.11. Control Statements

Control statements in Java are used to control the flow of execution in a program. They allow you to make decisions, iterate over code blocks, and alter the flow based on certain conditions.

1.11.1. if statement

The if statement is used to execute a block of code if a specified condition is true.

```
int x = 10;

if (x > 5) {

    System.out.println("x is greater than 5");

}
```

1.11.2. switch statement

The switch statement allows you to select one of many code blocks to be executed.

```
int day = 3;

switch (day) {

    case 1:

        System.out.println("Monday");

        break;

    case 2:

        System.out.println("Tuesday");

        break;

    // Add more cases...

    default:

        System.out.println("Invalid day");

}
```

1.11.3. loop statement

Loop statements allow you to execute a block of code repeatedly.

for Loop: Executes a block of code a fixed number of times.

```
for (int i = 0; i < 5; i++) {
    System.out.println("Iteration: " + i);
}
```

while Loop: Executes a block of code as long as a specified condition is true.

```
int i = 0;
while (i < 5) {
    System.out.println("Iteration: " + i);
    i++;
}
```

do-while Loop: Executes a block of code once and then repeats the execution as long as a specified condition is true.

```
int i = 0;
do {
    System.out.println("Iteration: " + i);
    i++;
} while (i < 5);
```

1.11.4. continue statement

The continue statement is used to skip the current iteration of a loop and continue with the next iteration.

```
for (int i = 0; i < 5; i++) {
    if (i == 2) {
        continue; // Skip iteration if i is 2
    }
    System.out.println("Iteration: " + i);
}
```

1.11.5. break statement

The break statement is used to terminate the loop or switch statement and transfer control to the statement following the terminated statement.

```
for (int i = 0; i < 5; i++) {
    if (i == 3) {
        break; // Terminate the loop if i is 3
    }
    System.out.println("Iteration: " + i);
}
```