DIGITAL NOTES
ON
JAVA PROGRAMMING

FOR

COMPUTER ENGG. STUDENT

PROVIEDED BY: -

SHAKTI RAJ SINGH, PROGRAMMER

UNIT II

Language Constructs, Classes, and Objects in Java

1. Variables, Types, and Type Declarations

Variables: In Java, a variable is a container for storing data values. Variables must be declared with a type, and the data type determines what kind of data the variable can hold.

Definition of Variable:

A variable is a named location in memory used to store a value that can be modified during program execution.

- **Declaration**: The process of defining a variable by specifying its type and name.
  Example:
- int x; // Declares an integer variable named 'x'

Primitive Data Types:

Java provides several **primitive data types** to store basic data values. These are:

- **byte**: 1 byte, stores integers from -128 to 127.
- **short**: 2 bytes, stores integers from -32,768 to 32,767.
- **int**: 4 bytes, stores integers from -2^31 to 2^31-1.
- **long**: 8 bytes, stores large integers.
- **float**: 4 bytes, stores single-precision floating-point numbers.
- **double**: 8 bytes, stores double-precision floating-point numbers.
- **char**: 2 bytes, stores a single character.
- **boolean**: 1 byte, stores true or false values.

Reference Data Types:

- These store references (memory addresses) to objects rather than actual data. Examples include arrays, classes, and interfaces.

Example:

String name = "Java"; // Reference variable holding the address of a String object

2. Data Types

Data Types specify the kind of data that can be stored in a variable or returned from a method. Java has two categories of data types:

Primitive Data Types:

- **Integer types**: byte, short, int, long
- **Floating-point types**: float, double
- **Character type**: char
- **Boolean type**: boolean

Reference Data Types:

- **Arrays**: A collection of elements of the same type.
- **Classes**: User-defined types representing objects.
- **Strings**: A class used for text manipulation.

Example:

int age = 25; // Primitive type

String name = "John"; // Reference type

3. Increment and Decrement Operators

Increment and Decrement Operators are used to modify a variable's value by 1. They are essential for looping and iterating over sequences.

Definition of Increment/Decrement Operators:

- **Increment Operator (++)**: Increases the value of a variable by 1.
- **Decrement Operator (--)**: Decreases the value of a variable by 1.

Types:

1. **Post-Increment (x++)**: Increases the value after using the current value in the expression.
2. **Pre-Increment (++x)**: Increases the value before using it in the expression.
3. **Post-Decrement (x--)**: Decreases the value after using the current value in the expression.
4. **Pre-Decrement (--x)**: Decreases the value before using it in the expression.

Example:

```
int x = 5;
System.out.println(x++);  // Prints 5, then x becomes 6
System.out.println(++x);  // Prints 7
```

4. Relational and Logical Operators

Relational Operators are used to compare two values or expressions, returning a boolean result (true or false).

Relational Operators:

- ==: Equal to
- !=: Not equal to
- <: Less than
- >: Greater than
- <=: Less than or equal to
- >=: Greater than or equal to

Logical Operators are used to combine multiple boolean expressions:

- &&: Logical AND
- ||: Logical OR
- !: Logical NOT

Example:

```
int a = 10;
int b = 20;
System.out.println(a == b);  // false
System.out.println(a < b);   // true
System.out.println(a && b);  // Cannot combine integers with &&, only boolean
```

5. Conditional Statements

Conditional statements allow you to execute different code based on whether a condition is true or false.

If-Then-Else:

An **if-else** statement is used to execute a block of code if a condition is true and another block if it is false.

Definition:

The **if-else statement** is a control structure used to perform decision-making based on a condition.

```
int x = 10;
if (x > 5) {
   System.out.println("x is greater than 5");
} else {
   System.out.println("x is less than or equal to 5");
}
```

Ternary Operator:

A shorthand version of the if-else statement. It evaluates a condition and returns a value based on whether the condition is true or false.

```
int x = 10;
String result = (x > 5) ? "Greater" : "Smaller or Equal";
System.out.println(result);  // "Greater"
```

6. Loops

Loops allow repeated execution of a block of code as long as a specified condition holds true.

Types of Loops:

1. **For Loop**: Used when the number of iterations is known in advance.

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

2. **While Loop**: Used when the number of iterations is unknown and depends on a condition.

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

3. Do-While Loop: Similar to while but guarantees at least one execution of the loop, as the condition is checked after the loop.

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

7. Switch-Case Statement
The **switch-case** statement is used to execute one of several possible blocks of code, based on the value of a variable.

Definition of Switch-Case:
A **switch-case** statement evaluates a variable or expression, and executes the block of code corresponding to the matching case.

```
int day = 3;
switch (day) {
   case 1:
      System.out.println("Monday");
      break;
   case 2:
      System.out.println("Tuesday");
      break;
   case 3:
      System.out.println("Wednesday");
      break;
   default:
      System.out.println("Invalid day");
}
```

8. Arrays
An **array** is a collection of variables of the same type, stored in contiguous memory locations.

Definition of Array:
An **array** in Java is a fixed-size data structure that holds a collection of values of the same data type.

Declaration and Initialization:

```
int[] arr = new int[5];  // Array of 5 integers
arr[0] = 10;  // Assign value to the first element
```

Accessing Array Elements:
Array elements can be accessed using their index, starting from 0.

```
System.out.println(arr[0]);  // Prints 10
```

Multi-Dimensional Arrays:
Arrays can be multi-dimensional (like a matrix). For example, a 2D array is an array of arrays.

```
int[][] matrix = new int[3][3];
matrix[0][0] = 1;
System.out.println(matrix[0][0]);  // Prints 1
```

9. Methods
Methods in Java are blocks of code designed to perform a specific task and can be reused throughout the program.

Definition of Method:
A **method** is a block of code that only runs when it is called. It can take parameters and return a value.
Method Declaration:

```
public returnType methodName(parameters) {
   // Method body
   return value;
}
```

Method Calling:
You call methods using their name and passing necessary arguments (if any).

```
public int add(int a, int b) {
   return a + b;
}
int result = add(5, 10);  // Calls the add method
```

10. Classes and Objects
Classes are blueprints for creating objects. **Objects** are instances of classes.

Definition of Class:
A **class** is a template or blueprint for creating objects, defining the properties and behaviors (variables and methods) that objects of the class can have.

Creating Objects:
Objects are created using the new keyword.

```
class Car {
   String model;
   int year;

   void drive() {
      System.out.println("Driving " + model);
   }
}

Car myCar = new Car();  // Creating an object of class Car
myCar.model = "Toyota";
myCar.year = 2020;
myCar.drive();  // Prints "Driving Toyota"
```

Accessing Members:

- **Instance Variables**: Each object has its own copy of these variables.
- **Static Members**: These are shared across all instances of the class and can be accessed using the class name.

Example of Static Members:

```
class Car {
   static int numberOfCars = 0;

   public Car() {
      numberOfCars++;
   }

   public static void displayCount() {
```

```
        System.out.println("Number of cars: " + numberOfCars);
    }
}

Car car1 = new Car();
Car car2 = new Car();
Car.displayCount();  // Prints "Number of cars: 2"
```