Java

Chapter 1

Introduction to Java

What is JAVA

Java is a popular, high-level, general-purpose programming language that was developed by Sun Microsystems. It was created by James Gosling in 1995. Java is used for building a wide range of applications, from web and mobile apps to desktop software.

JAVA Features:

- **Simple:** Java is simple to understand, easy to learn.
- **Platform Independent:** Java is often referred to as a "write once, run anywhere" language. This means that once the code is written, it may be run on any software or hardware system.
- **Object-Oriented Programming:** Java is an object-oriented programming language, which means it emphasizes the use of objects and classes to structure code.
- **Portable:** Java applications are highly portable because they run on any platform with a compatible JVM.
- **Robust:** Java's design includes features like strong type checking, exception handling, and automatic memory management to help developers write robust and error-free code.
- **Multi-threaded:** Java supports multithreading, making it easier to write programs that can execute multiple tasks concurrently.
- **High-performance:** With the introduction of features like the Just-In-Time (JIT) compiler and other optimizations, Java applications can achieve good performance.

First Java Program

Example:

```
public class Main {


    public static void main(String[] args) {

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

    }

}
```

Output:

```
Hello World!
```

Java Comments

Comments are used In Java to make the code more readable and understandable for developers.

Single Line Comments

- Single-line comments start with two forward slashes (//).
- Any content between // and the end of the line is ignored by the Java.

Example:

```
public class Main {
  public static void main(String[] args) {
    // This is a single line comment
    System.out.println("Hello World!!");
  }
}
```

Multi-line comments

- Multi-line comments start with /* and ends with */.
- Any content between /* and */ will be ignored by the Java.

Example:

```
public class Main {
  public static void main(String[] args) {
    /* This is a
    multi line
    comment */
    System.out.println("Hello World!!");
  }
}
```

Chapter 2

Java Fundamentals

Java Variables

Variables are containers for storing data values.

Syntax:

```
datatype variableName = value
```

Create Variables

To define a variable, you must specify its type and assign it a value.

```
int number = 50;
```

Types of Variables in Java

There are three types of variables in java:

- Local Variables
- Instance Variables

- Static Variables

Local Variables

A variable declared within a block, method, or constructor is known as a local variable.

Example:

```
int number = 50;
```

Instance Variables

Instance variables, known as non-static variables, are declared in a class outside of any method, constructor, or block.

Static Variables

Static variables are defined with the `static` keyword within a class but outside of any method, constructor, or block.

Example:

```
public class Main {
  public static String name;

  public static void main(String[] args) {
    name = " John";
    System.out.println("Hello" + name);
  }
}
```

Output:

```
Hello John
```

Java Data Types

Data types are used in Java to classify the various types of data that can be stored in a variable. There are two types of datatypes in Java:

- Primitive data type
- Non-Primitive data type

Primitive Data Types

There are 8 primitive data types are available in java.

- **bool:** Boolean data type represents one bit of information either true or false.
- **char:** The char data type is a single 16-bit Unicode character.
- **byte:** Byte data type is an 8-bit signed two's complement integer.
- **short:** Short data type is a 16-bit signed two's complement integer.
- **int:** It is a 32-bit signed two's complement integer.
- **long:** Long data type is a 64-bit signed two's complement integer.
- **float:** The float data type is a single-precision 32-bit IEEE 754 floating point.
- **double:**The double data type is a double-precision 64-bit IEEE 754 floating-point.

Data Type	Size	Range
bool	1 bit	true, false
char	2 byte	0 to 255
byte	1 byte	-128 to 127
short	2 byte	-32,768 to 32,767
int	4 byte	-2,147,483,648 to 2,147,483,647
long	8 byte	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 byte	upto 7 decimal digits
double	8 byte	upto 16 decimal digits

Java Operators

Operators are symbols that perform operations on variables and values.

Java operators are classified into five types:

- Arithmetic Operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators

Arithmetic Operators

Arithmetic operators are used to perform mathematical operations.

Operator	Description	Syntax
+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b
/	Division	a / b
%	Modulus	a % b

Assignment operators

Assignment operators are used to assign values to a variable.

Operator	Name	Syntax
=	Assignment	a = b
+=	Addition assignment	a += b
-=	Subtraction assignment	a -= b
*=	Multiplication assignment	a *= b
/=	Division assignment	a /= b
%=	Modulus assignment	a %= b

Comparison operators

Comparison operators are used to compare two values.

Operator	Description	Example
==	Equal	a==b
!=	Not Equal	a!=b
>	Greater than	a>b
>=	Greater than or equal to	a >= b
<< /td>	Less than	a < b
<=< /td>	Less than or equal to	a <= b

Logical operators

Logical operators perform logical operations and return a boolean value.

Operator	Description	Example
&&	Logical AND	a && b
||	Logical OR	a || b
!	Logical NOT	! (a=2 or b=3)

Bitwise operators

Bitwise operators are used to deal with binary operations.

Operator	Description	Example
&	Bitwise AND	a & b
|	Bitwise OR	a | b
~	Bitwise NOT	~a
^	Bitwise XOR	a ^ b

Chapter 3

Java Flow Control

Java if...else Statement

If-Else statements are part of conditional statements.

There are four types of conditional statements in Java:

- The if statement
- The if-else statement

- The if…elif…else Statement
- The nested-if statement

If Statement

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

Syntax:

```
if (condition) {
  // block of code to be executed if the condition is true
}
```

Example:

```
public class Main {
  public static void main(String[] args) {
    if (10 > 5) {
      System.out.println("10 is greater than 5");
    }
  }
}
```

Output:

```
10 is greater than 5
```

If...else statement

The `If...else` statement is used to execute a block of code if a specified condition is true and another block of code if the condition is false.

Syntax:

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

Example:

```
public class Main {
  public static void main(String[] args) {
    int x = 10;
    if (x > 5) {
      System.out.println("x is greater than 5");
    } else {
      System.out.println("x is not greater than 5");
    }
  }
}
```

Output:

```
x is greater than 5
```

if…elif…else Statement

Java `if-elif-else` statement executes a block of code among multiple possibilities.

Syntax:

```
if (condition) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and
condition2 is true
} else {
  // block of code to be executed if the condition1 is false and
condition2 is false
}
```

Example:

```
public class Main {
  public static void main(String[] args) {
    int x = 10;
    if (x > 15) {
      System.out.println("x is greater than 15");
    } else if (x > 10) {
      System.out.println("x is greater than 10 but less than or equal to
15");
    } else {
      System.out.println("x is equal to 10");
    }
  }
}
```

Output:

```
x is equal to 10
```

Java Switch

The `switch` statement is used to select one of many code blocks to be executed.

Syntax:

```
switch (expression) {
  case value1:
    // block of code
    break;
  case value2:
    // block of code
    break;
  default:
    // block of code
}
```

Example:

```
public class Main {
  public static void main(String[] args) {
    int wish = 1;
    switch (wish) {
      case 1:
        System.out.println("Good Morning");
        break;
      case 2:
        System.out.println("Good Day");
        break;
      case 3:
        System.out.println("Good Evening");
        break;
      case 4:
        System.out.println("Good Night");
        break;
    }
  }
}
```

Output:

```
Good Morning
```

Java For Loop

A `for loop` is used to execute a piece of code a specified number of times.

Syntax:

```
for (initialization; testExpression; increment/decrement) {
   // block of code
}
```

Example:

```
public class Main {
  public static void main(String[] args) {
    for (int i = 1; i <= 5; i++) {
      System.out.println("Hello Java");
    }
  }
}
```

Output:

```
Hello Java
Hello Java
Hello Java
Hello Java
Hello Java
```

Java While Loop

The `while loop` is used to execute a block of code as long as a specified condition is true.

Syntax:

```
while (condition) {
  // block of code
}
```

Example:

```
public class Main {
  public static void main(String[] args) {
    int i = 1;
    while (i <= 8) {
      System.out.println(i);
      i++;
    }
  }
}
```

Output:

```
1
2
3
4
5
6
7
8
```

Java Do...While Loop

The `do-while loop` is similar to the while loop. This loop would execute its statements at least once, even if the condition fails for the first time.

Syntax

```
do {
  // block of code
} while (condition);
```

Example:

```
public class Main {
  public static void main(String[] args) {
    int i = 0;
    do {
      System.out.println(i);
      i++;
    } while (i < 5);
  }
}
```

Output:

```
0
1
2
3
4
```

Java Break and Continue

Break Statement

The `break` statement is used to break out of the loop in which it is encountered. The break statement is used inside loops or switch statements in C programming.

Example:

```
public class Main {
  public static void main(String[] args) {
    int i;
    for (i = 0; i < 10; i++) {
      if (i == 6) {
        break;
      }
      System.out.println(i);
    }
  }
}
```

Output:

```
1
2
3
4
5
```

Continue Statement

The `continue` statement skips the loop's current iteration and proceeds to the next one.

Example:

```
public class Main {
  public static void main(String[] args) {
    int i;
    for (i = 1; i < 10; i++) {
      if (i == 3) {
       continue;
      }
      System.out.println(i);
    }
  }
}
```

Output:

```
1
2
4
5
6
7
8
9
```

Chapter 4

Java Arrays

Arrays in Java are used to store multiple values in a single variable.

Syntax:

```
String[] companies = {"Goggle", "Facebook", "Microsoft"};
```

Accessing Elements of an Array

Array elements can be accessed using indexing. Indexing in Java starts from 0.

Example:

```
public class Main {
  public static void main(String[] args) {
    String[] companies = {"Goggle", "Facebook", "Microsoft"};
    System.out.println(companies[1]);
  }
}
```

Output:

```
Facebook
```

Change an Array Element

To change the value of a specific element, use the index number.

Example:

```
public class Main {
  public static void main(String[] args) {
    String[] companies = {"Goggle", "Facebook", "Microsoft"};
    companies[2] = "TCS";
    System.out.println(companies[2]);
  }
}
```

Output:

```
TCS
```

Chapter 5

Java OOP

Java is an object-oriented programming language. The core concept of the object-oriented programming is to break complex problems into smaller objects.

Java Class

A class is a blueprint for creating objects.

Create a Class

To create a class, use the keyword `class`:

```
public class Main {
  int x = 5;
}
```

Java Objects

An object is called an instance of a class.

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

Output:

```
5
```

Java Methods

A method is a block of code that performs a specific task.

In Java, there are two types of methods:

User-defined Methods: We can create our own method based on our requirements.

Standard Library Methods: These are built-in methods in Java that are available to use.

Declaring a Java Method

The syntax to declare a method is:

```
returnType methodName() {
  // method body
}
```

returnType: It specifies what type of value a method returns. For example if a method has an int return type then it returns an integer value.

methodName: It is an identifier that is used to refer to the particular method in a program.

method body: It includes the programming statements that are used to perform some tasks. The method body is enclosed inside the curly braces `{ }`.

Example:

```
int addNumbers() {
// code
}
```

Calling a Method in Java

In the above example, we have declared a method named `addNumbers()`. Now, to use the method, we need to call it.

```
addNumbers();
```

Example:

```
class Main {

  // create a method
  public int addNumbers(int a, int b) {
    int sum = a + b;
    // return value
    return sum;
  }
  public static void main(String[] args) {
    int num1 = 25;
    int num2 = 15;
    // create an object of Main
    Main obj = new Main();
    // calling method
    int result = obj.addNumbers(num1, num2);
    System.out.println("Sum is: " + result);
  }
}
```

Output:

```
Sum is: 45
```

Java Strings

Java strings are a sequence of characters that are enclosed by double quotes.

Example:

```
public class Main {
  public static void main(String[] args) {
    String name = "Messi";
    System.out.println(name);
  }
}
```

Output:

```
Messi
```

String Concatenation

In Java, you can concatenate two strings with the + operator.

Example:

```
public class Main {
  public static void main(String[] args) {
    String firstName = "Lionel";
    String lastName = "Messi";
    System.out.println(firstName + " " + lastName);
  }
}
```

Output:

```
Lionel Messi
```

String Methods

Java has several built-in methods for manipulating strings.

length()

The length method returns the length of a string.

Example:

```
public class Main {
  public static void main(String[] args) {
    String greeting = "Hello Java";
    System.out.println(greeting.length());
  }
}
```

Output:

```
6
```

toUpperCase() and toLowerCase()

The `toUpperCase()` and `toLowerCase()` methods are used to convert a string to uppercase or lowercase letters.

Example:

```
public class Main {
  public static void main(String[] args) {
    String greeting = "Hello Java";
    System.out.println(greeting.toUpperCase());
    System.out.println(greeting.toLowerCase());
  }
}
```

Output:

```
HELLO JAVA
hello java
```

Java Inheritance

Inheritance is one of the key features of OOP that allows us to create a new class from an existing class.

The `extends` keyword is used to perform inheritance in Java.

Example:

```
class Vehicle {

  protected String brand = "Yamaha";

  public void honk() {

    System.out.println("Bhrum, bhruum!");

  }

}

class Bike extends Vehicle {

  private String modelName = "R15";

  public static void main(String[] args) {

    Bike myFastBike = new Bike();

    myFastBike.honk();

    System.out.println(myFastBike.brand + " " +
myFastBike.modelName);

  }

}
```

Output:

```
Bhrum, bhruum!
Yamaha R15
```

Java Polymorphism

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

Example:

```
class Bird {

  public void birdSound() {

    System.out.println("The bird makes a sound");

  }

}

class Owl extends Bird {

  public void birdSound() {

    System.out.println("The owl sound is: hoot");

  }

}

class Peacock extends Bird {

  public void birdSound() {

    System.out.println("The peacock sound is: scream");

  }

}

class Main {

  public static void main(String[] args) {

    Bird myBird = new Bird();

    Bird myOwl = new Owl();

    Bird myPeacock = new Peacock();

    myBird.birdSound();

    myOwl.birdSound();

    myPeacock.birdSound();

  }

}
```

Output:

```
The bird makes a sound
The owl sound is: hoot
The peacock sound is: scream
```