UNIT-1
FUNDAMENTALS OF JAVA PROGRAMMING

1.1 INTRODUCTION TO JAVA

Java is a high-level, object-oriented programming language that is designed to be portable, simple, and secure. It was created in the mid-1990s by James Gosling at Sun Microsystems (which is now owned by Oracle Corporation) and has since become one of the most popular programming languages in use today.

One of the key features of Java is its platform independence. Java programs can run on any computer system that has a Java Virtual Machine (JVM) installed, which means that developers can write code once and run it on any platform. This makes Java an ideal language for developing web applications and mobile apps.

Another important feature of Java is its object-oriented programming (OOP) model. Everything in Java is an object, which makes it easy to create reusable code and maintain large codebases. Java also has strong type checking, which helps to prevent common programming errors.

Java has a vast array of libraries and frameworks that make it easy to write complex programs quickly. It is widely used for developing enterprise-level applications, web applications, mobile apps, games, and much more.

In order to get started with Java programming, you will need to download and install the Java Development Kit (JDK) from Oracle's website. Once you have the JDK installed, you can use an Integrated Development Environment (IDE) such as Eclipse, IntelliJ IDEA, or NetBeans to write and debug your code.

1.2 HISTORY OF JAVA AND FEATURES

Java was created by James Gosling and his team of developers at Sun Microsystems (which was later acquired by Oracle Corporation) in the mid-1990s. The project was initially called "Oak," but was later renamed to Java.

The original goal of the Java project was to create a platform-independent language that could be used to develop software for a wide range of devices, from small embedded systems to large mainframe computers. The language was designed to be simple, secure, and portable, with a strong emphasis on object-oriented programming.

Java was first released to the public in 1995, and quickly gained popularity due to its platform independence and ease of use. Its popularity continued to grow in the late 1990s, as it became the preferred language for developing web applications and applets.

In 2006, Sun Microsystems released Java as open source software under the GNU General Public License. This move helped to further increase the popularity of Java, as it allowed developers to modify and distribute the language freely.

Today, Java is one of the most widely used programming languages in the world, with millions of developers using it to create a wide range of applications, from web applications and mobile apps to enterprise-level software and games. The language continues to evolve, with new features and improvements being added in each new release.

1.2.1 FEATURES OF JAVA:

Java is a powerful and versatile programming language with many features that make it popular among developers. Some of the key features of Java include:

1. **Platform Independence:** Java programs are compiled into platform-independent bytecode that can be run on any platform that has a Java Virtual Machine (JVM) installed. This makes Java programs highly portable.

2. **Object-Oriented Programming:** Java is a pure object-oriented programming language. Everything in Java is an object, which makes it easy to write reusable code and build complex applications.

3. **Robust and Secure:** Java was designed to be a secure and robust language. It has many built-in security features such as a security manager, which ensures that Java programs can run safely even in untrusted environments.

4. **Multithreading:** Java supports multithreading, which allows multiple threads of execution to run concurrently within a single program. This is especially useful for applications that need to perform multiple tasks at the same time.

5. **High Performance:** Java is known for its high performance and scalability. The JVM optimizes Java bytecode at runtime, which helps to improve the performance of Java applications.

6. **Rich Libraries:** Java has a vast array of libraries and frameworks that make it easy to write complex programs quickly. These libraries cover everything from network programming and database connectivity to user interface design and graphics.

7. **Architecture-neutral:** Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed. In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

8. **Dynamic:** Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++. Java supports dynamic compilation and automatic memory management (garbage collection).

9. **Distributed:** Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

10. **Compiled and Interpreted:** Java can be considered both a compiled and an interpreted language because its source code is first compiled into a binary byte-code. This byte-code runs on the Java Virtual Machine (JVM), which is usually a software-based interpreter.

Overall, Java is a versatile, reliable, and powerful programming language that is used to develop a wide range of applications, from small applets to large enterprise systems.

1.3 JAVA VIRTUAL MACHINE AND BYTE CODE

The Java Virtual Machine (JVM) is a key component of the Java platform. It is a software that provides an environment in which Java bytecode can be executed. The JVM interprets the bytecode and translates it into machine language that can be executed by the computer's processor.

Java bytecode is a highly optimized set of instructions that is produced when Java source code is compiled. It is a platform-independent format that can be executed on any system that has a JVM installed. This means that a Java program can be developed on one platform and then run on any other platform without modification.

The Java bytecode is designed to be highly optimized for execution on the JVM. It is a low-level representation of the original Java source code, and includes many optimizations such as constant folding, dead code elimination, and method in-lining. These optimizations help to improve the performance of Java programs.

The JVM is responsible for interpreting and executing the bytecode. It provides a number of important features, including memory management, garbage collection, and security. The JVM also includes a just-in-time (JIT) compiler, which can dynamically compile frequently used bytecode into machine code for faster execution.

Overall, the JVM and bytecode are important components of the Java platform. They provide a platform-independent way to execute Java programs, while also offering a high degree of performance, security, and reliability.

1.4 TYPES OF JAVA PROGRAM

There are several types of Java programs that can be developed, depending on the requirements of the project. Here are some of the most common types of Java programs:

1. **Console Programs:** These are text-based programs that run in a command-line interface (CLI). They are often used for simple tasks such as data input/output, system administration, or automation.

2. **Desktop Applications:** These are graphical user interface (GUI) programs that run on a desktop computer. They can be used for a wide range of purposes, such as word processing, video editing, or image manipulation.

3. **Web Applications:** These are server-side programs that run on a web server and provide services to clients via a web browser. They can be used for a wide range of purposes, such as e-commerce, social media, or content management.

4. **Mobile Applications:** These are programs that run on mobile devices such as smartphones and tablets. They can be used for a wide range of purposes, such as gaming, communication, or productivity.

5. **Applets:** These are small Java programs that run within a web browser. They are often used for interactive web applications, such as games or multimedia presentations.

6. **Enterprise Applications**: These are large-scale programs that are designed to run within a corporate environment. They can be used for a wide range of purposes, such as customer relationship management, supply chain management, or financial management.

1.5 BASIC CONCEPT OF OOP

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data and code to manipulate that data. The basic concept of OOP revolves around four principles, known as the "four pillars of OOP":

- **Encapsulation:** Encapsulation in Java is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.

- **Inheritance:** Inheritance is the process of creating new classes by inheriting properties and methods from existing classes. This allows the new classes to reuse code and behavior from the existing classes, reducing code duplication and improving code maintainability.

- **Polymorphism:** Polymorphism is the ability of objects to take on multiple forms. This allows objects to behave in different ways depending on the context in which they are used. Polymorphism is achieved through method overriding and method overloading.

- **Abstraction:** Abstraction is a process of hiding the implementation details and showing only functionality to the user. Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

Overall, the four pillars of OOP provide a powerful and flexible way of building software systems that are modular, reusable, and easy to maintain. By encapsulating data, using inheritance and polymorphism to reduce code duplication, and abstracting away unnecessary details, OOP provides a powerful set of tools for building complex systems in a clear and efficient way.

1.6 PROCEDURE ORIENTED V/S OBJECT ORIENTED

Procedure-Oriented Programming (POP) and Object-Oriented Programming (OOP) are two programming paradigms that differ in several ways. Here are some of the key differences between the two:

1. **Data and behavior:** In POP, data and behavior are separate concepts, with data stored in global variables and behavior defined in procedures. In OOP, data and behavior are combined into objects, which encapsulate both.

2. **Encapsulation:** Encapsulation is a key feature of OOP that allows objects to hide their internal state and expose well-defined interfaces. This helps to protect the object's data and ensures that the object's behavior remains consistent. POP does not provide the same level of encapsulation.

3. **Inheritance:** Inheritance is a powerful feature of OOP that allows objects to inherit properties and behavior from parent classes. This can help to reduce code duplication and improve code maintainability. POP does not provide a native way to implement inheritance.

4. **Polymorphism:** Polymorphism is another powerful feature of OOP that allows objects to take on multiple forms. This allows objects to behave in different ways depending on the context in which they are used. POP does not provide a native way to implement polymorphism.

5. **Modularity:** OOP provides a more modular approach to programming, with code organized into objects and classes. This makes it easier to reuse code and maintain large programs. In POP, code is organized into functions or procedures, which can be harder to reuse and maintain in large programs.

6. **Scalability:** OOP is generally more scalable than POP, as it allows for code to be easily extended and modified over time. In contrast, POP can become unwieldy and difficult to maintain as the size of a program grows.

Overall, while POP and OOP have their own strengths and weaknesses, OOP is generally considered to be a more powerful and flexible approach to programming, especially for large and complex systems.

1.7 BASIC PROGRAM OF JAVA

```
public class HelloWorld
 {
         public static void main(String[] args)
          {
                 System.out.println("Hello, World!");
         }
}
```

- **public:** This is an access modifier that makes the class accessible to other classes in the same package or other packages.
- **class:** This keyword is used to define a new class.
- **HelloWorld:** This is the name of the class we're defining.
- **{}:** This block of code contains the class definition.
- **public:** This is an access modifier that makes the method accessible to other classes in the same package or other packages.
- **static**: This keyword makes the method a class method that can be called without creating an instance of the class.
- **void:** This keyword specifies that the method doesn't return a value.

- **main:** This is the name of the method. It's the entry point of the program and is executed when the program is run.
- **(String[] args):** This is the method's argument list. In this case, it takes an array of strings as an argument.
- **{}:** This block of code contains the method definition.
- **System.out.println("Hello, World!"); :** This line of code prints "Hello, World!" to the console. System.out is a predefined output stream in Java, and println() is a method that prints a string to the console and adds a newline character at the end.

1.8 BASIC DATA TYPES

Java has several primitive data types that represent the basic building blocks of data in the language. These include:

1. **byte:** A byte is a signed 8-bit integer, with a range of -128 to 127.

   Example:byte b = 100;
2. **short:** A short is a signed 16-bit integer, with a range of -32,768 to 32,767.

   Example:short s = 5000;
3. **int:** An int is a signed 32-bit integer, with a range of -2,147,483,648 to 2,147,483,647.

   Example:int i = 100000;
4. **long:** A long is a signed 64-bit integer, with a range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

   Example:long l = 1234567890L;
5. **float:** A float is a single-precision 32-bit floating-point number, with a range of approximately 1.4E-45 to 3.4E+38, and a precision of about 7 decimal digits.

   Example:float f = 3.14f;
6. **double:** A double is a double-precision 64-bit floating-point number, with a range of approximately 4.9E-324 to 1.8E+308, and a precision of about 15 decimal digits.

   Example:double d = 3.14159;
7. **boolean:** A boolean represents a logical value, either true or false.

   Example:char c = 'A';
8. **char:** A char represents a single character in Unicode, with a range of '\u0000' (or 0) to '\uffff' (or 65,535).

   Example:boolean b1 = true;

   boolean b2 = false;

In addition to these primitive data types, Java also has a rich set of reference data types, such as classes, interfaces, and arrays, which are built using the primitive data types.

Classes: A class is a blueprint or template for creating objects that encapsulate data and behavior. A class can contain fields (variables) to store data, and methods (functions) to perform operations on that data. Objects are instances of a class, and each object has its own set of values for the fields defined in the class. Here's an example of a simple class definition in Java:

```
public class Person
{
 private String name;
  private int age;

 public Person(String name, int age) {
 this.name = name;
this.age = age;
   }

public String getName() {
  return name;
  }

  public int getAge() {
   return age;
  }
}
```

In this example, the Person class has two fields, name and age, and a constructor that takes arguments to initialize those fields. It also has two methods, getName() and getAge(), which return the values of those fields.

Interfaces: An interface is a collection of abstract methods that define a set of behaviors that a class can implement. In Java, interfaces are used to define contracts between different classes or components in a system. A class can implement one or more interfaces, and by doing so, it agrees to provide implementations for all the methods defined in those interfaces. Here's an example of a simple interface definition in Java:

```
public interface Drawable {
 public void draw();
}
```

In this example, the Drawable interface defines a single abstract method, draw(), which has no implementation. A class that implements this interface must provide an implementation for the draw() method.

Arrays: An array is a collection of elements of the same data type, stored in contiguous memory locations. In Java, arrays are used to store and manipulate collections of data. Arrays can be of any data type, including primitive types and objects. Here's an example of how to create and use an array of integers in Java:

```
int[] numbers = {1, 2, 3, 4, 5};
System.out.println(numbers[0]);
System.out.println(numbers[2]);
numbers[3] = 10;
System.out.println(numbers[3]);
```

In this example, we create an array of integers and initialize it with some values. We then access individual elements of the array using their index, and modify the value of one of the elements.

1.9 OPERATORS

In Java, operators are symbols that perform operations on one or more operands (values, variables, or expressions). There are several types of operators in Java, including:

Arithmetic operators: Arithmetic operators are used to perform arithmetic operations on numerical values. The basic arithmetic operators in Java are:

+ (addition)

- (subtraction)

* (multiplication)

/ (division)

% (modulus)

EXAMPLE:

```
int a = 10;
int b = 5;

// addition
```

```
int sum = a + b; // sum = 15

// subtraction
int difference = a - b; // difference = 5

// multiplication
int product = a * b; // product = 50

// division
int quotient = a / b; // quotient = 2

// modulus
int remainder = a % b; // remainder = 0
```

Assignment operators: Assignment operators are used to assign values to variables. The basic assignment operator in Java is =. There are also compound assignment operators that combine arithmetic operations with assignment, such as +=, -=, *=, /=, and %=.

EXAMPLE:

```
int a = 10;
int b = 5;

// basic assignment
a = b; // a = 5

// compound assignment
a += b; // equivalent to a = a + b; a = 10 + 5 = 15
b -= a; // equivalent to b = b - a; b = 5 - 15 = -10
```

Comparison operators: Comparison operators are used to compare two values and return a boolean value indicating the result. The basic comparison operators in Java are:

== (equal to)

!= (not equal to)

< (less than)

> (greater than)

<= (less than or equal to)

>= (greater than or equal to)

EXAMPLE:

```
int a = 10;
int b = 5;

// equal to
boolean isEqual = (a == b); // isEqual = false

// not equal to
boolean notEqual = (a != b); // notEqual = true

// greater than
boolean greaterThan = (a > b); // greaterThan = true

// less than or equal to
boolean lessThanOrEqual = (a <= b); // lessThanOrEqual = false
```

Logical operators: Logical operators are used to combine boolean values and return a boolean result. The basic logical operators in Java are:

&& (logical AND)

|| (logical OR)

! (logical NOT)

EXAMPLE:

```
boolean x = true;
boolean y = false;

// logical AND
boolean andResult = (x && y); // andResult = false

// logical OR
boolean orResult = (x || y); // orResult = true

// logical NOT
boolean notResult = !x; // notResult = false
```

Bitwise operators: Bitwise operators are used to perform operations on the binary representations of numerical values. The basic bitwise operators in Java are:

& (bitwise AND)

| (bitwise OR)

^ (bitwise XOR)

~ (bitwise NOT)

<< (left shift)

>> (right shift)

>>> (unsigned right shift)

EXAMPLE:

```
int a = 60; // 0011 1100
int b = 13; // 0000 1101

// bitwise AND
int andResult = a & b; // andResult = 0000 1100

// bitwise OR
int orResult = a | b; // orResult = 0011 1101

// bitwise XOR
int xorResult = a ^ b; // xorResult = 0011 0001

// bitwise NOT
int notResult = ~a;
// notResult = 1100 0011

// left shift
int leftShiftResult = a << 2;
 // leftShiftResult = 1111 0000

// right shift
int rightShiftResult = a >> 2;
// rightShiftResult = 0000 1111

// unsigned right shift
int unsignedRightShiftResult = a >>> 2;
// unsignedRightShiftResult = 0000 1111
```

Conditional operator: The conditional operator (also known as the ternary operator) is a shorthand way of writing an if-else statement. It has the following syntax:

condition ? value1 : value2

If the condition is true, the operator returns value1, otherwise it returns value2.

EXAMPLE:

```
int a = 10;
int b = 5;

// conditional operator
int max = (a > b) ? a : b; // max = 10
```

Increment or Decrement Operator: The Java increment or decrement operators require only one operand. Increment or decrement operators are used to perform increment and decrement operations.

Example :

```
 int x=10;
x++ // 10 (11)
++x // 12
 x-- //12 (11)
--x //10
```