Introduction Of Java
Unit – 1

➢ Java History

- Java is a general-purpose, object-oriented programming language developed by Sun Microsystems of USA in 1991. Originally called *Oak by James Gosling*. One of the inventors of the language, Java was designed for the development of software for consumer electronic devices like TVs, VCRs, toasters and such other electronic machines. Below lists some important milestones in the development of Java.

Year	Development
1990	Sun Microsystems decided to develop special software that could be used manipulate consumer electronic devices. A team of Sun Microsystems programmers headed by James Gosling was formed to undertake this task.
1991	After exploring the possibility of using the most popular object-oriented language C++, the team announced a new language named "Oak".
1992	The team, known as Green Project team by Sun, demonstrated the application of their new language to control a list of home appliances using a hand-held device with a tiny touch sensitive screen.
1993	The World Wide Web (WWW) appeared on the Internet and transformed the text-based Internet into a graphical-rich environment. The Green Project team came up with the idea of developing Web applets (tiny programs) using the new language that could run on all types of computers connected to Internet.
1994	The team developed a Web browser called "Hot Java" to locate and run applet programs on Internet. Hot Java demonstrated the power of the new language, thus making it instantly popular among the Internet users.
1995	Oak was renamed "Java", due to some legal snags. Java is just a name and is not an acronym. Many popular companies including Netscape and Microsoft announced their support to Java.
1996	Java established itself not only as a leader for Internet programming but also as a general-purpose, object-oriented programming language. Sun releases Java Development Kit 1.0.
1997	Sun releases Java Development Kit 1.1 (JDK 1.1) Features: • JDBC (Java Database Connectivity) • Inner Classes • Java Beans • RMI (Remote Method Invocation) • Reflection (introspection only)
1998	Sun releases the Java 2 with version 1.2 of the Software Development Kit (SDK 1.2) Features: • Collection's framework. • Java String memory map for constants. • Just In Time (JIT) compiler. • Jar Signer for signing Java ARchive (JAR) files.

	• Policy Tool for granting access to system resources. • Java Foundation Classes (JFC) which consists of Swing 1.0, Drag and Drop, and Java 2D class libraries. • Java Plug-in • Scrollable result sets, BLOB, CLOB, batch update, user-defined types in JDBC. • Audio support in Applets.
1999	Sun releases the Java 2 Platform, Standard Edition (J2SE) and Enterprise Edition (J2EE)
2000	J2SE with SDK 1.3 was released. Code name Kestrel. Features: • Java Sound • Jar Indexing • A huge list of enhancements in almost all the java area.
2002	J2SE with SDK 1.4 was released. Code name is Merlin. Features: • XML Processing • Java Print Service • Logging API • Java Web Start • JDBC 3.0 API • Assertions • Preferences API • Chained Exception • IPv6 Support • Regular Expressions • Image I/O API
2004	J2SE with SDK 5.0(instead of JDK 1.5) was released. This is known as J2SE 5.0. Code name is Tiger. Features: • Generics • Enhanced for Loop • Autoboxing/Unboxing • Typesafe Enums • Varargs • Static Import • Metadata (Annotations) • Instrumentation
2006	J2SE with SDK 6.0 was released in December 2006. Code name is Mustang. Features: • Scripting Language Support • JDBC 4.0 API • Java Compiler API • Pluggable Annotations • Native PKI, Java GSS, Kerberos and LDAP support. • Integrated Web Services. • Lot more enhancements.
2011	J2SE with SDK 7.0 was released in July 2011. Code name is Dolphin. Features: • Strings in switch Statement • Type Inference for Generic Instance Creation • Multiple Exception Handling • Support for Dynamic Languages • Try with Resources

	• Java nio Package • Binary Literals, underscore in literals • Diamond Syntax • Automatic null Handling

➢ Features of Java

1. Compiled and Interpreted

- Usually, computer language is either compiled or interpreted while java combines both approaches thus it makes two-stage system. Java compiler translates source code into byte code instruction. Byte code is not machine instruction i.e., therefore the second stage java interpreter generate machine code that can be executed by machine. i.e., running language.

2. Platform-Independent and Portable

- The most significant contribution of java is its portability. Java program easily moved one computer system to another anywhere and anytime. Change & upgrade in OS processors and system resources will not force any change in java program. So, java popular an Internet which interconnect different types of systems worldwide. We can download java applets from remote computer on to our local computer via Internet and execute it locally. Java ensures portability by two ways: (1) Java compiler generates byte code instruction that can be implemented on any machine. (2) The sizes of primitive data types are machine dependents.

3. Object Oriented

- **Java** is an object-oriented programming language where every program has at least one class. Programs are often built from many classes and objects, which are the instances of a class. Java is a true object-oriented language. Almost everything in java is in object. All program code and data reside within objects and classes. Java comes with an extensive set of classes, arranged in packages, that we can use in our programs by inheritance. The object model in java is simple and easy to control.

4. Robust and secured

- Java is a robust language. It provides many safeguards to ensure reliable code. It has strict compile time and run time checking for data types. It is designed as garbage-collected language relieving the programmers virtually all memory management problems. Java also incorporates concepts of exception handling, which ensures series of error and eliminates any risk of crashing the system.
- Security becomes an important issue for language that is used for programming on Internet. Java system not only verifies all memory access but also ensures no viruses are communicated with an applet. The absence of pointer in java ensure that program cannot gain access to memory location without proper authorization.

5. Distributed

- Java is designed as distributed language for creating applications on networks. It has the ability to share both data and program. Java applications can open and access remote object on Internet as easily as they can do in a local system. This enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

6. Simple, Small and Familiar

- According to Sun Microsystem, Java language is a simple programming language because: Java syntax is based on C++ (so easier for programmers to learn it after C++). Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc. Java is a small and simple language. Many features of C & C++ that are either redundant or sources of unreliable code are not part of java. E.g., java does not use pointer, preprocessor header files, goto statement and many others. It also eliminates operator overloading and multiple inheritance.

7. Multithreaded and Interactive

- Multithread means handling multiple tasks simultaneously. Java supports multithreaded programs. We need not wait for application to finish one task before beginning another. For example, we can listen to an audio clip, while scrolling a page and at the same time download an applet from distant computer. This feature greatly improves the interactive performance of graphical applications.

8. High performance

- Java performance is impressive due to use intermediate byte code. Java speeds comparable to the native of c & c++. of dynamically linking new class. Java architecture is also designed to reduce overheads during runtime. Further, the incorporation multithreading enhances the overall execution speed of java programs.

9. Dynamic and Extensible

- Java is dynamic language, capable of dynamically linking new class libraries, methods and objects. Java can also determine the type of class through a query, making it possible to either dynamically link or abort the program, depending on the response. Java program support functions written in other languages such as c & c++. These functions are known as native methods. This facility enables to programmers to use the efficient function available in these languages. Native methods are linked dynamically at runtime.

#Simple program

```
Public class subash{

Public Static void main (String [] arg)  {

System.out.println("hello world");

}
}
```

Fundamental Programming Structure
Unit - 2

2.1 Writing Comments

Comments can be used to explain Java code, and to make it more readable.

❖ Types of Java Comments

There are two types of comments in Java.

1. Single Line Comment: -

- Single-line comments start with two forward slashes (//).
- Any text between // and the end of the line is ignored by Java (will not be executed).
- This example uses a single-line comment before a line of code:
  // This is a comment

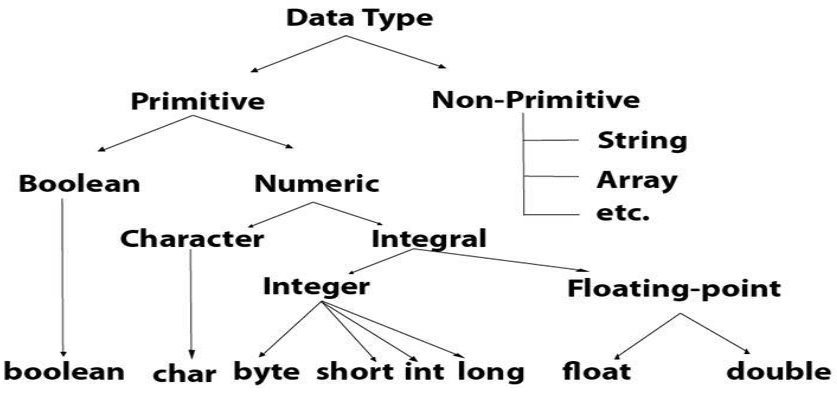
2. Multi Line Comment

- Multi-line comments start with /* and ends with */.
- Any text between /* and */ will be ignored by Java.
- This example uses a multi-line comment:
  /* The code below will print the words Hello World to the screen, and it is amazing */

2.2 Basic Data Types

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java: -

1. **Primitive data types:** - The primitive data types include Boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** - The non-primitive data type include Classes, Interfaces, and Arrays.



Primitive Data Types

- A primitive data type specifies the size and type of variable values, and it has no additional methods.
- There are eight primitive data types in Java:

Data Type	Size	Description
Byte	1 byte	Stores whole numbers from -128 to 127
Short	2 bytes	Stores whole numbers from -32,768 to 32,767
Int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
Long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
Float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
Double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
Boolean	1 bit	Stores true or false values
Char	2 bytes	Stores a single character/letter or ASCII values

Non-Primitive Data Types

- Non-primitive data types are called reference types because they refer to objects.
- Examples of non-primitive types are Strings, Arrays, Classes, Interface, etc.

2.3 Variable and Constants

➢ Variable

- Variables are containers for storing data values.
- In Java, there are different **types** of variables, for example:

- String: - stores text, such as "Hello". String values are surrounded by double quotes
- Int: - stores integers (whole numbers), without decimals, such as 123 or -123
- Float: - stores floating point numbers, with decimals, such as 19.99 or -19.99
- Char: - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes.
- Boolean: - stores values with two states: true or false.

Declaring (Creating) Variables

- To create a variable, you must specify the type and assign it a value: -

Syntax

type variable = value;

Where *type* is one of Java's types (such as int or String), and *variable* is the name of the variable (such as **x** or **name**). The **equal sign** is used to assign values to the variable.

How to declare variables of other types:

Example

```
int myNum = 5;
float myFloatNum = 5.99f;
char myLetter = 'D';
boolean myBool = true;
String myText = "Hello";
```

Display Variables

- The println() method is often used to display variables.

Example

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

➢ Constants

- **Constant** is a value that cannot be changed after assigning it.
- We use constants when we do not want a value to accidentally change.

How to initialize a constant in Java

To initialize a constant with a value, we use the `final` keyword in front of the data type.

Syntax:

```
final data_type CONSTANT_NAME = value;
```

Example:

```
public class Program {

    public static void main(String[] args) {

        // constant initialization
        final float PI = 3.14f;

        System.out.println("Pi: " + PI);
    }
}
```

Output:

Pi: 3.14

2.4 Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + **operator** to add together two values:

Example

```
int x = 100 + 50;
```

Although the + operator is often used to add together two values, like in the example above, it can also be used to add together a variable and a value, or a variable and another variable:

Java divides the operators into the following groups:

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

i. Arithmetic Operators

- Arithmetic operators are used to perform common mathematical operations.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

ii. Assignment Operators

- Assignment operators are used to assign values to variables.

In the example below, we use the **assignment** operator (=) to assign the value **10** to a variable called **x**:

Example

```
public class Main {
  public static void main(String[] args) {
    int x = 10;
    x += 5;
    System.out.println(x);  }}
```

A list of all assignment operators:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

iii. Comparison Operators

- Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

iv. Logical Operators: -

- Logical operators are used to determine the logic between variables or values: -

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

2.5 Type casting: -

Type casting is when you assign a value of one primitive data type to another type.
In Java, there are two types of casting:

- **Widening Casting** (automatically): - converting a smaller type to a larger type size
  byte -> short -> char -> int -> long -> float -> double

Widening casting is done automatically when passing a smaller size type to a larger size type:

Example:

```
public class Main {
  public static void main(String[] args) {
    int myInt = 9;
    double myDouble = myInt; // Automatic casting: int to double
    System.out.println(myInt);
    System.out.println(myDouble);
  }}
```

Output:
9
9.0

- **Narrowing Casting** (manually): - converting a larger type to a smaller size type
  double -> float -> long -> int -> char -> short -> byte

Narrowing casting must be done manually by placing the type in parentheses in front of the value:

Example:

```
public class Main {
  public static void main(String[] args) {
    double myDouble = 9.78d;
    int myInt = (int) myDouble; // Explicit casting: double to int
    System.out.println(myDouble);
    System.out.println(myInt);
  }
}
```

Output;
9.78
9

2.6 Control Flow

Java compiler executes the Java code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements.

Java provides three types of control flow statements.

1. Decision Making statements
2. Loop statements
3. Jump statements

I. Decision-Making statements: -

- Decision-making statements decide which statement to execute and when. Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided. There are two types of decision-making statements in java, i.e., If statement and switch statement.

1) If Statement: -

- the "if" statement is used to evaluate a condition. The condition of the If statement gives a Boolean value, either true or false. In Java, there are four types of if-statements given below.
    1. Simple if statement
    2. if-else statement
    3. if-else-if ladder
    4. Nested if-statement

1. Simple if statement: -

- It is the most basic statement among all control flow statements in Java.

Syntax of if statement is given below.

```
if(condition) {
statement 1; //executes when condition is true
}
```

2. if-else statement: -

- The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

Syntax of if-else statement is given below.

```
if(condition) {
statement 1; //executes when condition is true
```

```
}
else{
 statement 2; //executes when condition is false
 }
```

3. if-else-if ladder: -

- The if-else-if statement contains the if-statement followed by multiple else-if statements.

Syntax of if-else-if statement is given below.

```
if(condition 1) {
statement 1; //executes when condition 1 is true
}
else if(condition 2) {
statement 2; //executes when condition 2 is true
}  else {
statement 2; //executes when all the conditions are false
}
```

4. Nested if-statement: -

- In nested if-statements, the if statement can contain **if** or **if-else** statement inside another if or else-if statement.

Syntax of Nested if-statement is given below.

```
if(condition 1) {
statement 1; //executes when condition 1 is true
if(condition 2) {
statement 2; //executes when condition 2 is true
}
else{
statement 2; //executes when condition 2 is false
}
}
```

Example:

```
class Main {
  public static void main(String[] args) {
    // declaring double type variables
    Double n1 = -1.0, n2 = 4.5, n3 = -5.3, largest;
    // checks if n1 is greater than or equal to n2
    if (n1 >= n2) {
      // if...else statement inside the if block
      // checks if n1 is greater than or equal to n3
      if (n1 >= n3) {
        largest = n1;
```

```
      }

      else {
        largest = n3;
      }
    } else {

      // if..else statement inside else block
      // checks if n2 is greater than or equal to n3
      if (n2 >= n3) {
        largest = n2;
      }

      else {
        largest = n3;
      }
    }

    System.out.println("Largest Number: " + largest);
  }
}
```

Output:

```
Largest Number: 4.5
```

2) Switch Statement: -

- In Java, Switch statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched.

The syntax to use the switch statement is given below.

```
switch (expression){
    case value1:
     statement1;
     break;
    .
    .
    .
    case valueN:
     statementN;
     break;
```

```
    default:
     default statement;
}
```

Example: -

```
public class Student  {
public static void main(String[] args) {
int num = 2;
switch (num){
case 0:
System.out.println("number is 0");
break;
case 1:
System.out.println("number is 1");
break;
default:
System.out.println(num);
}
}
}
```

Output:
2

II. Loop Statements

In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true.

In Java, we have three types of loops that execute similarly. However, there are differences in their syntax and condition checking time.

1. for loop.
2. while loop.
3. do-while loop.

1. for loop: -

- It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code.

The syntax to use the for loop is given below.

```
for(<initialization>, <condition>, <increment/decrement>) {
//block of statements
}
```

Example:

```
public class Calculattion {
```

```
public static void main(String[] args) {
int sum = 0;
for(int j = 1; j<=10; j++) {
sum = sum + j;
}
System.out.println("The sum of first 10 natural numbers is " + sum);
}
}
```

Output
The sum of first 10 natural numbers is 55

2. While loop: -

- The while loop is also used to iterate over the number of statements multiple times.
- It is also known as the entry-controlled loop since the condition is checked at the start of the loop. If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.

The syntax of the while loop is given below.

```
While(<condition>){
//loop statements
}
```

Example: -

```
public class Calculattion {
public static void main(String[] args) {
int i = 0;
System.out.println("Printing the list of first 10 even numbers \n");
while(i<=10) {
System.out.println(i);
i = i + 2;
}}  }
```

Output:

```
Printing the list of first 10 even numbers
0
2
4
6
8
10
```

3. do-while loop: -

- The do-while loop checks the condition at the end of the loop after executing the loop statements
- It is also known as the exit-controlled loop since the condition is not checked in advance.

The syntax of the do-while loop is given below.

```
do
{
//statements
} while
```

Example: -

```
public class Calculattion {
public static void main(String[] args) {
int i = 0;
System.out.println("Printing the list of first 10 even numbers \n");
do {
System.out.println(i);
i = i + 2;
}while(i<=10);
} }
```

Output:

```
Printing the list of first 10 even numbers
0
2
4
6
8
10
```

III. Jump Statements: -

- Jump statements are used to transfer the control of the program to the specific statements. There are two types of jump statements in java, i.e., break and continue.

❖ Break statement: -

- As the name suggests, the break statement is used to break the current flow of the program and transfer the control to the next statement outside the current flow.

The break statement example with loop

```
public class BreakExample {

public static void main(String[] args) {
// TODO Auto-generated method stub
for(int i = 0; i<= 10; i++) {
```

```
System.out.println(i);
if(i==6) {
break;
}
}
}
}
```

Output:

```
0
1
2
3
4
5
6
```

❖ Continue statement: -

- Unlike break statement, the continue statement doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

Example:

```
public class ContinueExample {

public static void main(String[] args) {
// TODO Auto-generated method stub

for(int i = 0; i<= 2; i++) {

for (int j = i; j<=5; j++) {

if(j == 4) {
continue;
}
System.out.println(j);
}
}
}
}
```

Output:

```
0
1
2
3
```

```
5
1
2
3
5
2
3
5
```

2.7 Arrays: -

- Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with **square brackets.**
String[] cars;

Declared a variable that holds an array of strings.

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

Access the Elements of an Array
You access an array element by referring to the index number.

```
public class Main {
  public static void main(String[] args) {
    String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
    System.out.println(cars[0]);
  }
}
```

Output:
Volvo

Array Length: -

To find out how many elements an array has, use the length property:
Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars.length);
// Outputs 4
```

Loop Through an Array
You can loop through the array elements with the for loop, and use the length property to specify how many times the loop should run.

Example: -

```
public class Main {
```

```
  public static void main(String[] args) {
    String[] cars = {“Volvo”, “BMW”, “Ford”, “Mazda”};
    for (int I = 0; I < cars.length; i++) {
      System.out.println(cars[i]);
    }
  }
}
```

Output;
Volvo
BMW
Ford
Mazda

Multidimensional Arrays: -

- A multidimensional array is an array containing one or more arrays.

To create a two-dimensional array, add each array within its own set of **curly braces**:

Example

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
public class Main {
  public static void main(String[] args) {
    int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
    int x = myNumbers[1][2];
    System.out.println(x);
  }
}
```

Output:
7

Objects and Classes
Unit - 3

3.1 An Introduction to Object-Oriented Programming: -

- OOP stands for **Object-Oriented Programming**.
- Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

Object-oriented programming has several advantages over procedural programming: -

- OOP is faster and easier to execute.
- OOP provides a clear structure for the programs.
- OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug.
- OOP makes it possible to create full reusable applications with less code and shorter development time.

OOPs Concepts: -

- Polymorphism
- Inheritance
- Encapsulation
- Abstraction
- Class
- Object
- Method
- Message Passing

Classes and objects are the two main aspects of object-oriented programming.

So, a class is a template for objects, and an object is an instance of a class.

3.2 Using Predefined Class: -

- A Class is like an object constructor, or a "blueprint" for creating objects.
- Pre-defined class name can be used as a class name.
- To create a class, use the keyword **class**:
- The class Number has a main function that displays a message when it is executed. The main function takes string values as arguments.

```
public class Number{
  public static void main (String[] args){
    System.out.println("Pre-defined class name can be used as a class name");
  }
}
```

Predifined class are:

- Scanner
- Console
- System
- String

3.3 Defining Your own Class: -

- Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake. It is also called as user-defined class.
- Class in Java determines how an object will behave and what the object will contain.

Syntax: -

```
class <class_name>{
   field;
   method;
 }
```

Example: -

```
public class Dog {
  String breed;
  int age;
  String color;

  void barking() {
  }

  void hungry() {
  }

  void sleeping() {
  }
}
```

3.4 Static Fields and Methods: -

❖ Static Fields: -

- Static Fields in Java is variable which belongs to the class and initialized only once at the start of the execution. It is a variable which belongs to the class and not to object (instance). Static variables are initialized only once, at the start of the execution. These variables will be initialized first, before the initialization of any instance variables.
- We use the Static keyword to make the field static.
  - A single copy to be shared by all instances of the class

- A static variable can be accessed directly by the class name and doesn't need any object.

Syntax: -

```
<class-name>.<variable-name>
```

Example: -

```
class VariableDemo
{
   static int count=0;
   public void increment()
   {
      count++;
   }
   public static void main(String args[])
   {
      VariableDemo obj1=new VariableDemo();
      VariableDemo obj2=new VariableDemo();
      obj1.increment();
      obj2.increment();
      System.out.println("Obj1: count is="+obj1.count);
      System.out.println("Obj2: count is="+obj2.count);
   }
}
```

Output:

```
Obj1: count is=2
Obj2: count is=2
```

❖ Static Methods: -

- If you apply static keyword with any method, it is known as static method.
  - A static method belongs to the class rather than the object of a class.
  - A static method can be invoked without the need for creating an instance of a class.
  - A static method can access static data member and can change the value of it.

Syntax: -

```
<class-name>.<method-name>
```

Example of static method

```
class SimpleStaticExample
{
   // This is a static method
   static void myMethod()
   {
```

```
        System.out.println("myMethod");
    }

    public static void main(String[] args)
    {
        /* You can see that we are calling this
         * method without creating any object.
         */
         myMethod();
    }
}
```

Output: -
myMethod

3.5 Method Parameters: -

- Information can be passed to methods as parameter. Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.
- The following example has a method that takes a String called **fname** as parameter. When the method is called, we pass along a first name, which is used inside the method to print the full name:

Example: -

Single Parameters

```
public class Main {
  static void myMethod(String fname) {
    System.out.println(fname + " Refsnes");
  }

  public static void main(String[] args) {
    myMethod("Liam");
    myMethod("Jenny");
    myMethod("Anja");
  }
}
// Liam Refsnes
// Jenny Refsnes
// Anja Refsnes
```

❖ Multiple Parameters: -

- **You can have as many parameters as you like.**

Example: -

```
public class Main {
  static void myMethod(String fname, int age) {
    System.out.println(fname + " is " + age);
  }

  public static void main(String[] args) {
    myMethod("Liam", 5);
    myMethod("Jenny", 8);
    myMethod("Anja", 31);
  }
}
// Liam is 5
// Jenny is 8
// Anja is 31
```

3.6 Object Construction: -

- In Java, an object is created from a class. We have already created the class named Main, so now we can use this to create objects.
- To create an object of Main, specify the class name, followed by the object name, and use the keyword **new**:

Example: -

Create an object called "myObj" and print the value of x:

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

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

Output;
5

3.7 Packages: -

- A package in Java is used to group related classes. Think of it as **a folder in a file directory**. We use packages to avoid name conflicts, and to write a better maintainable code. Packages are divided into two categories:
  - Built-in Packages (packages from the Java API)
  - User-defined Packages (create your own packages)

❖ Built-in Packages

- The Java API is a library of prewritten classes, that are free to use, included in the Java Development Environment.
- To use a class or a package from the library, you need to use the import keyword:

Syntax: -

```
import package.name.Class;   // Import a single class
import package.name.*;   // Import the whole package
```

for example, the Scanner class, **which is used to get user input**, write the following code:
import java.util.Scanner;

Example:

```
import java.util.Scanner; // import the Scanner class

class Main {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);
    String userName;

    // Enter username and press Enter
    System.out.println("Enter username");
    userName = myObj.nextLine();

    System.out.println("Username is: " + userName);
  }
}
```

❖ User-defined Packages: -

- **To create a package, use the package keyword:**
- **To create your own package, you need to understand that Java uses a file system directory to store them. Just like folders on your computer:**

Example

```
└── root
  └── mypack
    └── MyPackageClass.java
```

```
package mypack;
class MyPackageClass {
  public static void main(String[] args) {
    System.out.println("This is my package!");
  }
}
```

Output;
This is my package!

Inheritance and Interfaces
Unit - 4

4.1 Classes, Super classes and Subclasses: -

- In Java, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:
  - **Class -**A class is a blueprint from which individual objects are created.
  - **subclass** (child) - the class that inherits from another class.
  - **superclass** (parent) - the class being inherited from.

To inherit from a class, use the extends keyword.
In the example below, the Car class (subclass) inherits the attributes and methods from the Vehicle class (superclass):

Example: -

```
class Vehicle {
  protected String brand = "Ford";
  public void honk() {
    System.out.println("Tuut, tuut!");
  }
}

class Car extends Vehicle {
  private String modelName = "Mustang";
  public static void main(String[] args) {
```

```
    Car myFastCar = new Car();
    myFastCar.honk();
    System.out.println(myFastCar.brand + " " + myFastCar.modelName);
  }
}
```

Output: -
Tuut, tuut!
Ford Mustang

4.2 Polymorphism: -

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

Inheritance lets us inherit attributes and methods from another class. **Polymorphism** uses those methods to perform different tasks. This allows us to perform a single action in different ways.

For example, think of a superclass called Animal that has a method called animalSound(). Subclasses of Animals could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound.

Example: -

```
class Animal {
  public void animalSound() {
    System.out.println("The animal makes a sound");
  }
}

class Pig extends Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
}

class Dog extends Animal {
  public void animalSound() {
    System.out.println("The dog says: bow wow");
  }
}

class Main {
  public static void main(String[] args) {
    Animal myAnimal = new Animal();
    Animal myPig = new Pig();
    Animal myDog = new Dog();
```

```
    myAnimal.animalSound();
    myPig.animalSound();
    myDog.animalSound();
  }
}
```

Output: -
The animal makes a sound
The pig says: wee wee
The dog says: bow wow

4.3 Dynamic Binding: -

- In dynamic binding, the method call is bonded to the method body at runtime. This is also known as late binding.

Method Overriding is a perfect example of dynamic binding as in overriding both parent and child classes have same method and in this case the **type of the object** determines which method is to be executed. The type of object is determined at the run time so this is known as dynamic binding.

In overriding both parent and child classes have same method . Let's see by an example.

```
class Human{
   //Overridden Method
   public void walk()
   {
      System.out.println("Human walks");
   }
}
class Demo extends Human{
   //Overriding Method
   public void walk(){
      System.out.println("Boy walks");
   }
   public static void main( String args[]) {
      /* Reference is of Human type and object is
       * Boy type
       */
      Human obj = new Demo();
      /* Reference is of HUman type and object is
       * of Human type.
       */
      Human obj2 = new Human();
      obj.walk();
      obj2.walk();
   }
}
```

Output: -
Boy walks
Human walks

4.4 Final Classes and Methods: -

❖ Final Classes: -

- When we declare a class as final, then we restrict other classes to inherit or extend it.
- In short, Java final class can't be extended by other classes in the inheritance. If another class attempts to extend the final class, then there will be a compilation error.

Example: -

```
final class Tech {
  // methods and variables of the class Tech
}
class Vidvan extends Tech {
  // COMPILE- TIME error as it extends final class
}
```

❖ Final Methods: -

- We can declare Java methods as Final Method by adding the Final keyword before the method name.
- The Method with Final Keyword cannot be overridden in the subclasses.

Example:

```
class Bike{
  final void run(){System.out.println("running...");}
}
class Honda2 extends Bike{
   public static void main(String args[]){
    new Honda2().run();
   }
}
```

Output: - running...

4.5 Abstract Classes: -

- Data abstraction is the process of hiding certain details and showing only essential information to the user.
- Abstraction can be achieved with either abstract classes or interfaces.
- To access the abstract class, it must be inherited from another class.
- The abstract keyword is a non-access modifier, used for classes and methods:

- **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

Example: -

```
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }}
// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }}
class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Output:

```
The pig says: wee wee
Zzz
```

4.6 Access Specifies: -

- The access Specifier in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access Specifier:

1. **Private: -**The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default:** - The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

3. **Protected: -** The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public: -** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

1) Private: -

Example: -

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}

public class Simple{
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }
}
```

2) Default: -

Example: -

```
//save by A.java
package pack;
class A{
  void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.*;
class B{
  public static void main(String args[]){
   A obj = new A();//Compile Time Error
   obj.msg();//Compile Time Error
  }
}
```

Note: In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

3) Protected: -

Example: -

```
//save by A.java
package pack;
public class A{
protected void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.*;
  class B extends A{
  public static void main(String args[]){
   B obj = new B();
  obj.msg();
 }
}
```

Output:
Hello

4) Public: -

Example: -

```
//save by A.java
package pack;
public class A{
public void msg(){System.out.println("Hello");}
}

//save by B.java

package mypack;
import pack.*;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Output: Hello

4.7 Interfaces: -

- Another way to achieve abstraction in Java, is with interfaces.

- An interface is a completely "**abstract class**" that is used to group related methods with empty bodies.
- To access the interface methods, the interface must be "implemented" (kinda like inherited) by another class with the implements keyword (instead of extends). The body of the interface method is provided by the "implement" class:

Example: -

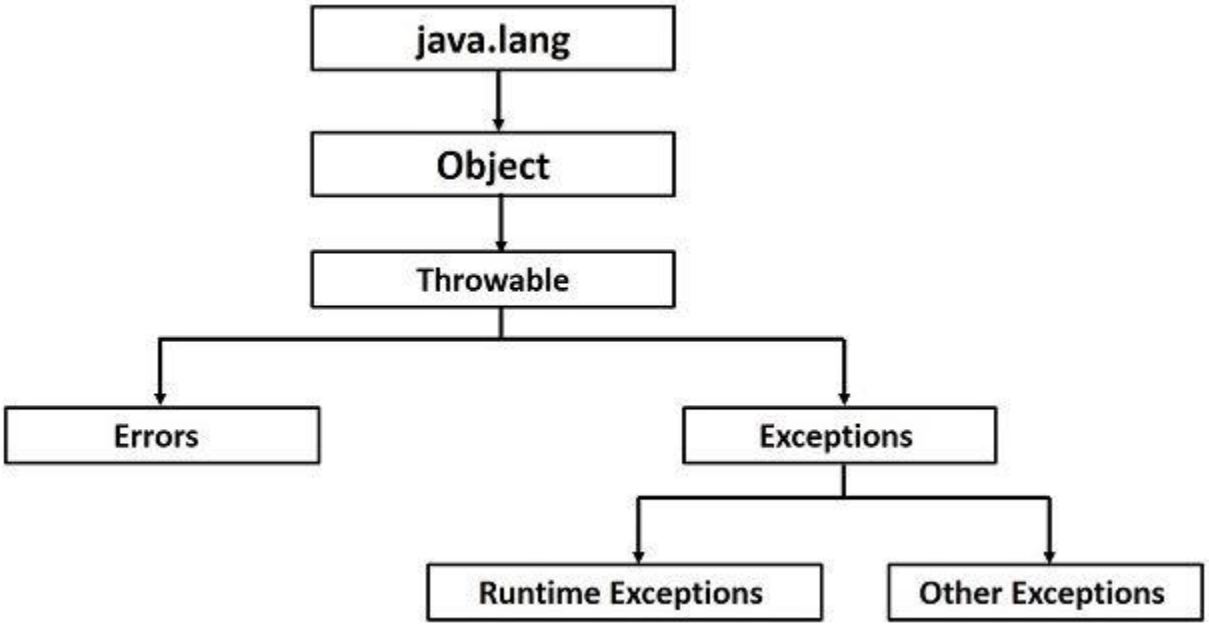
```
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void sleep(); // interface method (does not have a body)
}

class Pig implements Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    System.out.println("Zzz");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig();
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Output;
The pig says: wee wee
Zzz

Exception Handling
Unit - 5

5.1 Dealing with Errors: -

- The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.
- In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.
- Exception Handling is a mechanism to handle runtime errors such as Class Not Found Exception, IO Exception, SQL Exception, Remote Exception, etc.
- The **try-catch** is the simplest method of handling exceptions. Put the code you want to run in the try block, and any Java exceptions that the code throws are caught by one or more catch blocks. This method will catch any type of Java exceptions that get thrown. This is the simplest mechanism for handling exceptions.
- The Exception class has two main subclasses: IO Exception class and Run time Exception Class.



5.2 Catching Exceptions: -

- A method catches an exception using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following –

Syntax: -

```
try {
   // Protected code
```

```
} catch (ExceptionName e1) {
   // Catch block
}
```

Example: -

```
import java.io.*;

public class ExcepTest {

   public static void main(String args[]) {
      try {
         int a[] = new int[2];
         System.out.println("Access element three :" + a[3]);
      } catch (ArrayIndexOutOfBoundsException e) {
         System.out.println("Exception thrown  :" + e);
      }
      System.out.println("Out of the block");
   }
}
```

Output:
Exception thrown :java.lang.ArrayIndexOutOfBoundsException: 3
Out of the block

5.3 Try, Catch, Throw, Throws, and Finally: -

Try: -

The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.

Catch: -

The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.

Throw: -

The "throw" keyword is used to throw an exception.

Throws: -

The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

Finally: -

The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.

Example of try, catch, finally:

```
class Main {
   public static void main(String args[]) {
     //try block
     try{
           System.out.println("::Try block::");
           int num=67/0;
           System.out.println(num);
      }
      //catch block
      catch(ArithmeticException e){
          System.out.println("::Catch block::");
         System.out.println("ArithmeticException::Number divided by zero");
      }
      //finally block
      finally{
          System.out.println("::Finally block::");
      }
      System.out.println("Rest of the code continues...");
   }
}
```

Output;

```
::Try block::
::Catch block::
ArithmeticException::Number divided by zero
::Finally block::
Rest of the code continues...
```

Example of throw and throws:

```
import java.io.*;
class Example_Throw {
    //declare exception using throws in the method signature
  void testMethod(int num) throws IOException, ArithmeticException{
     if(num==1)
        throw new IOException("IOException Occurred");
     else
        throw new ArithmeticException("ArithmeticException");
  }
}class Main{
  public static void main(String args[]){
   try{
```

```
      //this try block calls the above method so handle the exception
     Example_Throw obj=new Example_Throw();
     obj.testMethod(1);
   }catch(Exception ex){
     System.out.println(ex);
    }
  }
}
```

Output;
IOException Occurred