**Module 6- Core Java**

**1.Introduction to Java**

**1) History of Java**

**Ans:-**

🡪before java : “OAK” : 1991 : james goasling

Renamed “java” from “OAK” : 1995 : james goasling

🡪 James Gosling, Mike Sheridan, and Patrick Naughton at Sun Microsystems (now Oracle Corporation) initiated the Java project.

**2) Features of Java (Platform Independent, Object-Oriented, etc.).**

Ans:-

🡪1)simple

🡪2)OO

🡪3)interpreter : JVM : bytecode (classfile) to machine

Code.

🡪4)Robust :powerful

🡪5)secure

🡪6)dynamic

🡪7)high performance :10x

🡪8)multithreading :

-🡪9)platform independent :

🡪10) portable :

**(3) Understanding JVM, JRE, and JDK.**

**Ans:-**

🡪JVM is a software program that runs Java bytecode on a computer.

🡪JVM translates bytecode into machine code, allowing Java programs to run on any device that has a JVM.

🡪JRE is a software package that provides the libraries, frameworks, and tools required to run Java programs.

🡪JRE includes the JVM, Java Class Loader, and other libraries and frameworks.

🡪JDK is a software package that provides the tools and libraries required to develop, test, and run Java programs.

🡪JDK includes the JRE, as well as additional tools and libraries for Java development.

🡪JDK > JRE > JVM

🡪JDK includes JRE, which includes JVM.

**(4) Setting up the Java environment and IDE (e.g., Eclipse, IntelliJ).**

**Ans:-**

🡪Setting up Java Environment

1. Download and install JDK from Oracle's website.

2. Set JAVA\_HOME environment variable to JDK installation path.

3. Add JDK's bin directory to system's PATH variable.

4. Verify Java installation using java -version command.

🡪Setting up Eclipse IDE

1. Download and install Eclipse IDE for Java Developers.

2. Launch Eclipse and create a new Java project.

3. Choose project name, location, and Java runtime (JDK) version.

4. Create a new Java class and start coding.

**(5) Java Program Structure (Packages, Classes, Methods).**

**Ans:-**

🡪Packages

🡪 A package is a collection of related classes, interfaces, and subpackages.

🡪 Packages help organize code, avoid naming conflicts, and control access to classes and members.

🡪 package packageName; (first line of a Java file).

🡪class

🡪is an collection of data member(variables) and member function(methods, process) with its behaviors.

🡪Classes define the structure and behavior of objects.

🡪 public class ClassName { ... }.

🡪method

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

🡪 Methods define the behavior of objects and can be reused.🡪public returnType methodName(parameters) .

**2. Data Types, Variables, and Operators**

**(1)Primitive Data Types in Java (int, float, char, etc.)**

**Ans:-**

🡪Int

🡪 Size: 8 byte

🡪Range: -2,147,483,648 to 2,147,483,647

🡪most commonly used integer type.

🡪 Float

🡪 Size: 4 byte

🡪 Decimal numbers, used for floating-point calculations.

🡪 Char

🡪 Size: 2 byte

🡪 Single characters, used for text processing.

🡪 Double

🡪 Size: 8 byte🡪 Decimal numbers, used for high-precision floating-point calculations

(2) Variable Declaration and Initialization.

Ans:-

🡪variable

🡪means is nothing but to store some value.

🡪initialization

🡪// Implicit initialization

int num;

// Explicit initialization

int num = 10;

// Dynamic initialization

int num = new Scanner(System.in);

**(3) Operators: Arithmetic, Relational, Logical, Assignment, Unary, and Bitwise.**

**Ans:-**

🡪 Arithmetic Operators

🡪 Addition: +

🡪 Subtraction: -

🡪 Multiplication: \*

🡪 Division: /

🡪Modulus: %

🡪. Relational Operators

- Equal to: ==

- Not equal to: !=

- Greater than: >

- Less than: <

- Greater than or equal to: >=

- Less than or equal to: <=

🡪3. Logical Operators

- And: &&

- Or: ||

- Not: !

🡪4. Assignment Operators

- Assign: =

- Add and assign: +=

- Subtract and assign: -=

- Multiply and assign: \*=

- Divide and assign: /=

- Modulus and assign: %=

🡪5. Unary Operators

- Increment: ++

- Decrement: --

- Positive: +

- Negative: -

- Not: !

🡪6. Bitwise Operators

- And: &

- Or: |

- Xor: ^

- Not: ~

- Left shift: <<

- Right shift: >>

- Unsigned right shift: >>>

**(4) Type Conversion and Type Casting.**

**Ans:-**

🡪Type conversion : convert from one data type to another data type mechanism.

🡪:there are mainly 2 types.

🡪1) implicit :automatically : convert from smaller data type in size convert into Bigger Data type.

🡪2) explicit :type casting : convert from Bigger data type in size convert into Smaller Data type.

🡪 The process of changing a value from one data type to another, potentially losing information.

**3. Control Flow Statements**

(1)If-Else Statements.

Ans:-

🡪 if condition is true then your if block will be execute otherwise else block will be executed.

🡪if()

{

Block of code………

}else{

Block of code…..

}

(2) Switch Case Statements.

Ans:-

🡪 A switch case statement is a type of control structure that allows you to execute different blocks of code based on the value of an expression. It's a multi-way decision maker that can replace multiple if-else statements.

🡪 switch (expression) {

case value1:

// code to execute

break;

case value2:

// code to execute

break;

...

default:

// code to execute if no match found

}

(3) Loops (For, While, Do-While)

Ans:-

🡪for loop

🡪 A for loop is a type of loop that allows you to execute a block of code repeatedly for a specified number of iterations.

🡪 for (initialization; condition; increment/decrement)

{

// code to execute

}

🡪while loop

🡪 A while loop is a type of loop that allows you to execute a block of code repeatedly while a certain condition is true.

🡪 while (condition) {

// code to execute

}

🡪Do while

🡪 A do-while loop is a type of loop that allows you to execute a block of code repeatedly while a certain condition is true. The code is executed at least once before the condition is checked.

🡪 do {

// code to execute

} while (condition);

(4) Break and Continue Keywords.

Ans:-

🡪break

🡪 The break keyword is used to exit a loop or a switch statement.

🡪break;

🡪continue

🡪 The continue keyword is used to skip the current iteration of a loop and move on to the next iteration.

🡪continue;

**4. Classes and Objects**

**(1) Defining a Class and Object in Java.**

**Ans:-**

🡪 is an collection of data member(variables) and member function(methods, process) with its behaviors.

🡪 class classname

{

data member

member function

}

🡪 object : is a instancs of an class.

:when you create class variables also called..

:its uses new keyword and class constructor to create object.

:access whole properties of an class except private

🡪classname objectname = new constructor();

(2) Constructors and Overloading.

Ans:-

🡪constructors

🡪 is an special member function because its same name as a class name.

🡪 : does not return any value even void ,can be overloaded, to initialized value of your data members at object creation time,may used access modifire except private, when your class object you create at thtat time to called constructor.

🡪there are mainly 2 types

1) default : no any argument in constructor

2) parameterized: may have one or more argument in constructor.

🡪overloading

🡪 the two or more method name should be same in a single class but its behaviors(data types, arguments) are different.

**(3) Object Creation, Accessing Members of the Class.**

**Ans:-**

🡪 is a instancs of an class.

🡪 Syntax

🡪ClassName objectName = new ClassName();

🡪 **package** com.oops;

**class** student

{

**int** sid;

String sname,city;

**public** **void** setDataValue()

{

sid=1;

sname="Tisha";

city="Ahmedabad";

}

**public** **void** display()

{

System.***out***.println("sid is..."+sid);

System.***out***.println("sname is..."+sname);

System.***out***.println("city is..."+city);

}

}

**public** **class** classDemo

{

**public** **static** **void** main(String[] args)

{

student s1=**new** student();

s1.setDataValue();

s1.display();

}

}

**(4) this Keyword.**

Ans:-

🡪when your class variable name and argument variable names are same at that time to separate your class variable with using this keyword.

**5. Methods in Java**

**(1)Defining Methods**

**Ans:-**

**🡪**

1. Instance Method: operates on instance variables

2. Static Method: belongs to a class, not an instance

3. Abstract Method: declared but not implemented

4.Access Modifier: public, private, protected.

**(2) Method Parameters and Return Types.**

**Ans:-**

**🡪**Method Parameters:

Formal Parameters: Declared in the method signature

Actual Parameters: Passed to the method when called

Parameter Types: Primitive, reference.

🡪Return Types:

Primitive Types: int, double, boolean, etc.

Reference Types: Objects, arrays, etc.

Void: No return value

**(3)Method Overloading(compile time)**

**Ans:-**

**🡪**the two or more method name should be same in a single class but its behaviors(data types, arguments) are different.

**(4) Static Methods and Variables.**

**Ans:-**

**🡪**Static means constant at reference side : call by reference.

🡪Variable means nothing but store some values.

**6. Object-Oriented Programming (OOPs) Concepts.**

**(1)** **Basics of OOP: Encapsulation, Inheritance, Polymorphism, Abstraction.**

**Ans:-**

🡪Encapsulation: wrapping up of data into single unit.

🡪Inheritance: properties of parent class extends into child class.

🡪polymorphism: ability to take one name having many forms or different forms.

🡪abstraction: only essential part should be display rest of the part will be hide.

(**2) Inheritance: Single, Multilevel, Hierarchical.**

**Ans:-**

**🡪**1) single : only one parent having only one child

2) multilevel : single inheritance having one another child

3) hierarchical : one parent having 2 or more child

**(3) Method Overriding and Dynamic Method Dispatch.**

**Ans:-**

**🡪**method overriding(run time) : the whole signature of the method should be same in super class as well as in subclass but its behaviors (body part of the method) are different**.**

**7. Constructors and Destructors**

**(1) Constructor Types (Default, Parameterized)**

**Ans:-**

**🡪**1) default : no any argument in constructor

2) parameterized: may have one or more argument in constructor.

**(2) Copy Constructor (Emulated in Java).**

**Ans:-**

**🡪**Java does not support copy constructors like C++. However, you can emulate the behavior of a copy constructor using various techniques.

**(3) Constructor Overloading.**

**Ans:-**

**🡪**Constructor overloading is a technique in Java where multiple constructors with different parameter lists can be defined for a class.

**(4)** **Object Life Cycle and Garbage Collection.**

**Ans:-**

**🡪**Object Life Cycle:

The object life cycle refers to the series of stages an object goes through, from creation to garbage collection.

Garbage Collection:

Garbage collection is the automatic process by which the Java Virtual Machine (JVM) reclaims memory occupied by unused or unreachable objects.

**8. Arrays and Strings**

**(1)One-Dimensional and Multidimensional Arrays.**

**Ans:-**

**🡪**1) one dimentional : [] : at a time only one loop will be use

2) two or more dimentional : [][] or [][][] : loop with in loop will be used.

**(2) String Handling in Java: String Class, StringBuffer, StringBuilder.**

**Ans:-**

**🡪**1) String : immutable : we can not changed

:its provide so many methods to use it for your program

:set of sequence of characters

2) String Buffer : mutable : we can be changed

2.1) StringBuilder: it is a mutable sequence of characters, similar to a String, but can be modified.

**(3)** **Array of Objects.**

**Ana:-**

**🡪**An array of objects is a collection of objects of the same class or interface, where each element in the array is a reference to an object**.**

**(4)String Methods (length, charAt, substring, etc.)**

**Ans:-**

**🡪**

1. length(): Returns the length of the string.

2. charAt(int index): Returns the character at the specified index.

3. substring(int beginIndex): Returns a new string that is a subset of the original string.

4. concat(String str): Concatenates the specified string to the end of the current string.

5. equals(Object anObject): Compares the current string with the specified object.

6. toUpperCase(): Converts the current string to uppercase.

7. toLowerCase(): Converts the current string to lowercase.

**9. Inheritance and Polymorphism**

**(1)Inheritance Types and Benefits**

**Ans:-**

**🡪**properties of parent class extends into child class

:properties of superclass extends into subclass

:main purpose is : Reusability , extensibility

:to used "extends" keyword through create inheritance

:always called last child class to create object with access the properties of parent class except private

:there are mainly 5 types

1) single : only one parent having only one child

2) multilevel : single inheritance having one another child

3) hierarchical : one parent having 2 or more child

4) multiple : java does not support directly

5) hybrid: java does not support directly

**(2) Method Overriding.**

**Ans:-**

**🡪**method overriding(run time) : the whole signature of the method should be same in super class as well as in subclass but its behaviors (body part of the method) are different**.**

**(3)** **Dynamic Binding (Run-Time Polymorphism)**

**Ans:-**

**🡪**Dynamic Binding (Run-Time Polymorphism): The process of determining the method to be invoked at runtime, rather than at compile time, based on the actual object type.

**(4) Super Keyword and Method Hiding.**

**Ans:-**

**🡪**when your super class variable name and subclass variable name are same then you used superclass variable want to used value into subclass at that time used super keyword with variable.

**10. Interfaces and Abstract Classes**

**(1)Abstract Classes and Methods**

**Ans:-**

🡪only essential part should be display rest of the part will be hide.

🡪using with method

🡪 do no specify body part of the method

: your class must be also abstract

:must override your abstract method into your child class.

**(2) Interfaces: Multiple Inheritance in Java.**

**Ans:-**

**🡪**main purpose is to resolved the problem of multiple inheritance and hybrid.

**(3)** **Implementing Multiple Interfaces.**

**Ans:-**

**🡪**In Java, a class can implement multiple interfaces, which allows it to inherit methods and constants from multiple sources.

**11. Packages and Access Modifiers**

**(1)Java Packages: Built-in and User-Defined Packages.**

**Ans:-**

🡪A package in Java is a collection of related classes, interfaces, and subpackages that are organized together to provide a single unit of functionality.

🡪Java Packages: Built-in packages

(e.g., java.lang, java.util)

are pre-installed with JDK, while user-defined packages are created by developers to organize their classes and interfaces.

**(2) Access Modifiers: Private, Default, Protected, Public**

**Ans;-**

**🡪**

1. Private: Accessible only within the same class.

2. Default (no modifier): Accessible within the same class and package.

3. Protected: Accessible within the same class, package, and subclasses.

4. Public: Accessible from anywhere.

**(3) Importing Packages and Classpath.**

**Ans:-**

🡪 Importing Packages and Classpath: Importing packages uses "import" statements, while Classpath is a sequence of directories and archives that Java uses to search for classes and resources.

**12. Exception Handling**

**(1)Types of Exceptions: Checked and Unchecked.**

**Ans:-**

🡪 Types of Exceptions: Checked exceptions are checked at compile-time (e.g., IOException),

🡪while Unchecked exceptions are not checked at compile-time (e.g., NullPointerException).

**(2)try, catch, finally, throw, throws.**

**Ans:-**

**🡪**

1) try : to find the error from the block

:when you know in my block some line of code having error

:in which line to find the error that line to reamining line in the block are skipped

:whatever error found in try block that error throw to catchblock.

:try followed with catch , finally or both.

2) catch : whatever error thrown by try block that error will handled with appropriate class

:catch can be multiple

3) finally :if error will come or not but my block always perfomed.

4) throw : to create own Exception / custom exception / user defined exception

:to write inside the method

:used with new keyword and our own Exception constructor

:at a time only 1 exception to called

5) throws : to write with method or function signature

:used with userdefined or system defined exception

:called multiple exceptions by write using ",".

**(3) Custom Exception Classes.**

**Ans:-**

🡪 Custom Exception Classes: A user-defined class that extends the Exception class, allowing developers to create specific exceptions for their applications.

**13. Multithreading**

**(1)Introduction to Threads.**

**Ans:-**

🡪 its an light weight process or processor.

🡪 its totally depends on process

🡪its self class

🡪its derived from java.lang package

🡪each and every programm must have a thread i.e. main()

🡪when you start your code for executing.

**(2) Creating Threads by Extending Thread Class or Implementing Runnable Interface.**

**Ans:-**

🡪 Method 1: Extending Thread Class

1. Extend Thread Class: Create a subclass of the Thread class.

2. Override run() Method: Implement the run() method, which contains the code to be executed by the thread.

🡪Method 2: Implementing Runnable Interface

1. Implement Runnable Interface: Create a class that implements the Runnable interface.

2. Implement run() Method: Implement the run() method, which contains the code to be executed by the thread.

3. Create Thread Object: Create a Thread object, passing the Runnable object to its constructor.

**(3) Thread Life Cycle.**

**Ans:-**

🡪 1) newborn state : when you create object

2) runnable state: start()

3) running state: run()

4) blocked state: suspend()=>resume(), sleep(ms),wait()=>notify()

5) dead state: stop().

**(4) Synchronization and Inter-thread Communication.**

**Ans:-**

🡪 Synchronization:

1. Synchronized Methods: Use synchronized keyword to lock objects.

2. Synchronized Blocks: Use synchronized keyword to lock specific blocks of code.

🡪Inter-thread Communication:

1. wait(): Pause thread execution until notified.

2. notify(): Notify waiting threads to resume execution.

3. notifyAll(): Notify all waiting threads to resume execution.

**14. File Handling**

**(1) o Introduction to File I/O in Java (java.io package).**

**Ans:-**

🡪 its derived from java.io.pacakage;

file its self class

:its mechnism like to store and retrive from it

:its mechanism in the file are unstructed

:file its has common exception will be

FileNotFound,

IOException

:work in Stream

:stream means device or medium

:there are mainly 2 stream

:there are mainly 2 exception mostly used

1.FileNotFound

2.IOException

**(2) FileReader and FileWriter Classes.**

**Ans:-**

🡪 1.Reader: to read the data from the file

:FileReader

2.Writer: to write the data into the file

:FileWriter

**(3) BufferedReader and BufferedWriter.**

**Ans:-**

🡪 1. BufferedReader: Reads text from a character-input stream, buffering characters for efficient reading.

2. BufferedWriter: Writes text to a character-output stream, buffering characters for efficient writing.

**(4) Serialization and Deserialization.**

**Ans:-**

🡪 1. Serialization: Converting Java objects to byte stream.

2. Deserialization: Converting byte stream back to Java objects.

**15. Collections Framework**

**(1)Introduction to Collections Framework.**

**Ans:-**

🡪 is a group of object into single object

:its derived from java.util package

:Collection its self interface.

**(2) List, Set, Map, and Queue Interfaces**

**Ans:**

**🡪**1. List Interface: Ordered collection of elements (e.g., ArrayList, LinkedList).

2. Set Interface: Unordered collection of unique elements (e.g., HashSet, TreeSet).

3. Map Interface: Unordered collection of key-value pairs (e.g., HashMap, TreeMap).

4. Queue Interface: Ordered collection of elements for First-In-First-Out (FIFO) access (e.g., LinkedList, PriorityQueue).

**(3)ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap.**

**Ans:-**

**🡪**1) ArrayList : auto implemented List interface

:its represent like dynamic array

:automatically shrink and grow both

:Default size is 0

:duplicate values are allow

:when you add some value in it , so same way display on console

:add() and remove()

:default symbol is "[]"

2) HashSet :auto implemented Set interface

:its represent like dynamic array

:automatically shrink and grow both

:Default size is 0

:duplicate values are not allow

:all the value has an hashkey

:all hashkey convert into hashcode

:all the value display via hashcode wise

:add() and remove()

:default symbol is "[]"

3) HashMap:auto implemented Map interface

:its represent like dynamic array

:automatically shrink and grow both

:Default size is 0

:its work look like in pair<k,v>

:duplicate pair are not allow,if key is same so value override

:all the pair has an Haskey

:all Haskey convert into hashcode

:all the pair display via hashcode wise

:put(),get() and remove()

:default symbol is "{}"

4)TreeSet:

: A set that maintains its elements in a sorted order.

: No duplicate elements allowed.

: Elements are iterated in a sorted order.

: Based on a balanced binary search tree (Red-Black tree).

5)TreeMap:

: A map that maintains its entries in a sorted order based on keys.

: No duplicate keys allowed.

: Entries are iterated in a sorted order based on keys.

: Based on a balanced binary search tree (Red-Black tree).

**(4) Iterators and ListIterators.**

**Ans:-**

**🡪**Iterator:

1. Iterates over a collection in the forward direction only.

2. hasNext(): Checks if there are more elements to iterate.

3. next(): Returns the next element in the iteration.

4. remove(): Removes the last element returned by the iterator.

ListIterator:

1. Bi-Directional: Iterates over a list in both forward and backward directions.

2. has NeXT(): Checks if there are more elements to iterate in the forward direction.

3. next(): Returns the next element in the forward iteration.

4. hasPrevious(): Checks if there are more elements to iterate in the backward direction.

5. previous(): Returns the previous element in the backward iteration.

6. add(): Inserts a new element into the list.

7. remove(): Removes the last element returned by the iterator.

8. set(): Replaces the last element returned by the iterator with a new element.

**16. Java Input/Output (I/O)**

**(1)Streams in Java (InputStream, OutputStream).**

**Ans:-**

**🡪**

1. InputStream: Reads data from a source (e.g., file, network).

2. OutputStream: Writes data to a destination (e.g., file, network).

**(2) Reading and Writing Data Using Streams.**

**Ans:-**

**🡪**Reading:

1. Create InputStream

2. Read data using read()

3. Close InputStream using close()

Writing:

1. Create OutputStream

2. Write data using write()

3. Close OutputStream using close()

**(3) Handling File I/O Operations.**

**Ans:-**

**🡪**Reading Files:

1. FileInputStream: Reads bytes from a file.

2. BufferedReader: Reads text from a file.

3. FileReader: Reads characters from a file.

Writing Files:

1. FileOutputStream: Writes bytes to a file.

2. BufferedWriter: Writes text to a file.

3. FileWriter: Writes characters to a file.