Java– Industry Assignment 2024

1.Introduction to Java

1. History of Java

-> Java is High-level, object-oriented programming language developed by James Gosling at Sun Microsystems in 1995.

-> Originally named Oak, later changed to Java.

-> Acquired by Oracle Corporation in 2010.

-> Designed to be platform-independent with the principle of "Write One, Run Anywhere".

2.Fetures of Java

->1.platform independent-Java programs run on any OS with a JVM.

2.Object-oriented-Based on Concepts like Classes, Objects, Inheritance, Encapsulation, Abstract, Polymorphism.

3.Secure-Provides features like Bytecode Verification and Security APIS.

4.Robust-Exception handling, memory management, and garbage collection.

5.Multithreaded-Allows concurrent execution of multiple tasks.

6.Portable- Code can run on different platforms without modification.

3.Understading JVM,JRE,JDK

->JVM(Java Virtual Machine):Execute Java bytecode and enable platform independence.

->JRE(Java Runtime Environment):It is provided to Internal libraries to run java
applications.

->JDK(Java Development Kit):It is provided to JRE+JVM And also Debugger, Compiler.

4.Setting Up the Java Environment and IDE

5.Java Program Structure

->Packages: Organize related classes.

->Classes: Blueprint of objects.

->Methods: Contain executable code.

2.Data Types, Variables, and Operators

1.Primitive Data Types in Java

->Java provides 8 primitive data types:

byte->	1 byte	-	Stores small integer values (-128 to 127)
short->	2 bytes	-	Stores medium integer values (-32,768 to 32,767)
int->	4 bytes	-	Stores whole numbers (-2^31 to 2^31-1)
long->	8 bytes	-	Stores large integers (-2^63 to 2^63-1)
float->	4 bytes	-	Stores decimal numbers (up to 7 digits precision)
double->	8 bytes	-	Stores large decimal numbers (up to 15 digits precision)
char->	2 bytes	-	Stores a single character (Unicode-based)
Boolean->	1 bit	-	Stores true or false values

2. Variable Declaration and Initialization

->Variables in Java must be declared before use and can be initialized when declared.

->Declaration Syntax:

```
datatype variableName;
 int a;
```

->Initialization Syntax:

```
datatype variableName = value;
String A="Nikhil";
```

3.Operators in Java

->Operators are used to perform operations on variables and values.

a) Arithmetic Operators

Operator	Description	Example
+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b

/	Division	a / b
%	Modulus (Remainder)	a % b

b) Relational (Comparison) Operators (Return boolean values - true/false)

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

c) Logical Operators (Used for boolean logic)

Operator	Description	Example (x=true, y=false)
&&	Logical AND	x && y false
!	Logical NOT	!x

d) Assignment Operators (Used to assign values to variables)

Operator	Example	Equivalent to
=	a = b	a = b
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a=a%b

e) Unary Operators (Operate on a single operand)

Operator	Description	Example (a=5)	Result
+	Positive value	+a	5
-	Negative value	-a	-5
++	Increment	a++	6
--	Decrement	a--	4

f) Bitwise Operators (Operate on binary values)

Operator	Description	Example (a=5, b=3)	Result

&	AND	a & b	1
`	`	OR	`a
^	XOR	a ^ b	6
~	Complement	~a	-6
<<	Left shift	a << 1	10
>>	Right shift	a >> 1	2

4.Type Conversion and Type Casting

->Implicit (Widening) Casting – Automatic conversion from smaller to larger data types.

->Explicit (Narrowing) Casting – Manual conversion from larger to smaller data types.

3.Control Flow Statements in Java

1.If-Else Statements

->The if-else statement allows conditional execution of code blocks

```
ex:-if (condition) {
    //  true
    } else {
    //  false
    }
```

2. Switch-Case Statements

->The switch statement is used when a variable is tested against multiple values

```
ex:-switch (expression) {
    case value1:
       // Code to execute
       break;
    case value2:
       // Code to execute
       break;
    default:
       // Code to execute if no cases match
}
```

3.Loops in Java

->Loops are used to execute a block of code multiple times.

a) For Loop

->The for loop is used when the number of iterations is known.

```
ex:-for (int i = 1; i <= 5; i++) {
    System.out.println("Iteration: " + i);
    }
```

b) While Loop

->The while loop is used when the condition is checked before execution.

```
ex:-int i = 1;
    while (i <= 5) {
    System.out.println("Iteration: " + i);
    i++;
    }
```

c) Do-While Loop

->The do-while loop guarantees at least one execution.

```
ex:-int i = 1;
    do {
    System.out.println("Iteration: " + i);
    i++;
    } while (i <= 5);
```

4.Break and Continue Keywords

->break exits a loop early.

->continue skips the current iteration and moves to the next

```
ex:- (Break)
    for (int i = 1; i <= 5; i++) {
    if (i == 3) break;
    System.out.println("Iteration: " + i);
    }
```

ex:- (Continue)

```
for (int i = 1; i <= 5; i++) {
if (i == 3) continue;
System.out.println("Iteration: " + i);
}
```

4.Classes and Objects in Java

1.Defining a Class and Object in Java

->Class: A blueprint or template for creating objects. It defines variables (attributes) and methods (behavior).

ex-class car{

```
System.ot.println("print");
}
```

->Object: An instance of a class with specific values assigned to its attributes.

ex:-Car c=new Car();

2.Constructors and Overloading

->A constructor is a special method used to initialize objects.

->It has the same name as the class and no return type.

--Types of Constructors:

1.Default Constructor: No parameters, initializes default values.

2.Parameterized Constructor: Takes arguments for initialization.

3.Constructor Overloading: Multiple constructors with different parameters.

3.Object Creation & Accessing Class Members

->Creating an object: ClassName obj = new ClassName();

->Accessing attributes: obj.attributeName;

->Calling methods: obj.methodName();

4.The this Keyword

->this refers to the current object and is used to avoid name conflicts.

->It helps differentiate instance variables from method parameters.

5.Methods in Java

1.Defining Methods in Java

->A method is a block of code that performs a specific task and can be called multiple times.

Syntax:-

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

2.Method Parameters and Return Types

->Methods can take parameters (inputs) and return values (outputs).

->void methods do not return a value.

3.Method Overloading

->Method Overloading allows multiple methods with the same name but different parameters

4.Static Methods and Variables

->Static variables are shared among all instances of a class.

->Static methods belong to the class rather than instances.

6.Object-Oriented Programming (OOPs) Concepts in Java

1.Basics of OOP :Encapsulation, Inheritance, Polymorphism, Abstraction

a)Encapsulation

->Wrapping data (variables) and methods together as a single unit.

->Uses private access modifiers with getters and setters.

b) Inheritance

->Allows a class to acquire properties and behavior from another class.

--Types:

1.Single Inheritance: One class inherits from another.

2.Multilevel Inheritance: A derived class acts as a base for another class.

3.Hierarchical Inheritance: Multiple classes inherit from a single parent class.

c) Polymorphism

->Method Overloading (Compile-time polymorphism)

->Method Overriding (Runtime polymorphism)

d) Abstraction

->Hiding implementation details while showing essential features.

->Implemented using abstract classes or interfaces.

2.Inheritance in Java

a) Single Inheritance

->One class inherits from another class.

```
Base Class
   |
   |
Derived Class
```

b) Multilevel Inheritance

->A derived class becomes the parent for another class.

```
Base Class
   |
   |
Derived Class
   |
   |
Derived Class
```

c) Hierarchical Inheritance

->Multiple child classes inherit from a single parent class.

```
                Base Class
          ------------|-------------
         |                    |
         |                    |
    Derived Class        Derived Class
```

3. Method Overriding and Dynamic Method Dispatch

->Method Overriding: A subclass provides a specific implementation of a method already defined in its parent class.

->Dynamic Method Dispatch (Runtime Polymorphism): A superclass reference variable can point to a subclass object.

7.Constructors and Destructors in Java

1.Constructor Types (Default, Parameterized)

a) Default Constructor

->A constructor without parameters.

->If no constructor is defined, Java provides a default constructor.

b) Parameterized Constructor

->Accepts arguments to initialize object attributes.

2. Copy Constructor (Emulated in Java)

->Java does not have a built-in copy constructor like C++, but we can implement it manually.

3. Constructor Overloading

->Multiple constructors in the same class with different parameters.

4. Object Life Cycle and Garbage Collection

->Object Creation: Done using the new keyword.

->Garbage Collection: Java automatically removes unused objects using the Garbage Collector.

--Finalizer (finalize() method)

->Java does not support destructors like C++.

->The finalize() method is called by the Garbage Collector before destroying an object.

8.Arrays and Strings in Java

1. One-Dimensional and Multidimensional Arrays

a) One-Dimensional Arrays

->A collection of elements of the same type stored in contiguous memory locations.

Syntax:-

```
dataType[] arrayName = new dataType[size];
```

b) Multidimensional Arrays

->Arrays with more than one dimension, like matrices.

Syntax:-

```
dataType[][] arrayName = new dataType[rows][cols];
```

2. StringHandling in Java: String Class, StringBuffer, StringBuilder

->A String in Java is a sequence of characters. Java provides three main ways to handle strings:

1.String Class (Immutable)

->Strings are immutable, meaning they cannot be changed once created

2.StringBuffer Class (Mutable)

->Used when frequent modifications are required in a string.

->More secure in multithreaded environments.

3.StringBuilder Class (Mutable & Faster than StringBuffer)

->Similar to StringBuffer but faster because it's not thread-safe.

3.Array of Objects

->An array that stores objects instead of primitive data types.

4.StringMethods (length, charAt, substring, etc.)

Method	Description	Example
length()	Returns the length of the string	"Java".length() → 4
charAt(index)	Returns the character at the specified index	"Hello".charAt(1) → 'e'
substring(start, end)	Extracts a substring	"Programming".substring(0, 4) → "Prog"
toUpperCase()	Converts to uppercase	"hello".toUpperCase() → "HELLO"
toLowerCase()	Converts to lowercase	"WORLD".toLowerCase() → "world"
replace(old, new)	Replaces characters	"Java".replace('J', 'K') → "Kava"
contains(str)	Checks if a string contains a substring	"Java Programming".contains("Java") → true

9.Inheritance and Polymorphism

1.Inheritance Types and Benefits

->Inheritance is one of the core Object-Oriented Programming (OOP) concepts that allows a class to inherit properties and

behavior (methods) from another class.

--Benefits of Inheritance

* Code Reusability – Reduces code duplication by reusing existing functionality.

* Method Overriding – Allows modifying inherited methods for specific behavior.

* Improved Maintainability – Easier to manage code by organizing it into a hierarchy.

* Extensibility – New functionality can be added without modifying the existing code.

--Types of Inheritance in Java

1.Single Inheritance

2.Multilevel Inheritance

3.Hierarchical Inheritance

4.Multiple Inheritance--Java does not support multiple inheritance with classes but allows it with interfaces.

2.Method Overriding

->Method Overriding allows a subclass to provide a specific implementation of a method that is already defined in its

superclass.

3. Dynamic Binding (Run-Time Polymorphism)

->Polymorphism in Java allows methods to be called dynamically based on the object's runtime type, not the reference type.

4. super Keyword and MethodHiding

->The super keyword in Java is used to refer to the immediate parent class.

--Uses of super

* Access parent class methods

* Access parent class constructors

* Access parent class variables

--Method Hiding:

->Method Hiding occurs when a static method in a subclass has the same name and signature as in its superclass.--

10. Interfaces and Abstract Classes

1. AbstractClasses and Methods

->An abstract class in Java is a class that cannot be instantiated and may contain abstract methods. It serves as a blueprint for subclasses.

--Abstract Class

-Declared using the abstract keyword.

-Can have both abstract and concrete (regular) methods.

-Cannot be instantiated directly.

-Used for common behaviors that subclasses share

--Abstract Methods

-Declared without a body (abstract void methodName();).

-Must be overridden in the subclass.

2.Interfaces: Multiple Inheritance in Java

->An interface is a blueprint of a class that contains only abstract methods and constants.

->Java does not support multiple inheritance in classes but allows it using interfaces.

3.Implementing Multiple Interfaces

ex:-

```
 interface A {
 void methodA();
 }
 interface B {
 void methodB();
 }
 class C implements A, B { // Implements both interfaces
 public void methodA() {
 System.out.println("Method A from Interface A");
}
public void methodB() {
System.out.println("Method B from Interface B");
}
}
class Test {
     public static void main(String[] args) {
          C obj = new C();
            obj.methodA();
            obj.methodB();
         }
}
```

11.Packages and Access Modifiers

1.JavaPackages: Built-in and User-Defined Packages

->A package in Java is a collection of related classes and interfaces. It helps in organizing code, preventing naming

conflicts, and controlling access.

--Types of Packages in Java

1.Built-in Packages

2.User-defined Packages

(1)Built-in Packages

->Java provides several built-in packages that contain commonly used classes and interfaces.

Package	Description
java.lang	Default package (e.g., String, Math, System).
java.util	Contains utility classes (e.g., ArrayList, HashMap).
java.io	Provides classes for file handling (FileReader, BufferedReader).
java.net	Supports networking (Socket, URL).
java.sql	Used for database connectivity (Connection, ResultSet).

(2)User-Defined Packages

->Developers can create their own packages for better project structure.

->Use the package keyword at the beginning of the file.

2.AccessModifiers: Private, Default, Protected, Public

Modifier	Scope	in Class?	Same Package?	Other Packages	Anywhere
public	Anywhere	Yes	Yes	Yes	Yes

private	Only within the class	Yes	Yes	No	No
(Default)	Within the same package	Yes	Yes	Yes	No
protected	Within the same package & subclasses in other packages	Yes	Yes	Yes	No

3.ImportingPackages and Classpath

->import keyword is used to import a package or class.

ex:-

```
  import java.util.Scanner; // Imports only Scanner class
  class ImportExample {
    public static void main(String[] args) {
     Scanner sc = new Scanner(System.in);
     System.out.println("Enter something:");
     String input = sc.nextLine();
     System.out.println("You entered: " + input);
        }
     }
```

(2)Importing an Entire Package

ex:-

```
 import java.util.*; // Imports all classes from java.util
 class ImportAllExample {
   public static void main(String[] args) {
     ArrayList<Integer> list = new ArrayList<>();
     list.add(10);
     System.out.println(list);
      }
  }
```

12. Exception Handling

1.TypesofExceptions: Checked and Unchecked

(1). Checked Exceptions

-Compile-time exceptions (must be handled using try-catch or throws).

-Occurs in scenarios like file handling, database access, network operations

(2) Unchecked Exceptions (Runtime Exceptions)

-Occurs at runtime.

-Not checked by the compiler.

-Common causes: Invalid input, division by zero, null reference, etc.

2.try,catch, finally, throw, throws

(1)try Block

-Code that might cause an exception goes inside the try block.

(2)catch Block

-If an exception occurs, the catch block handles it.

(3)finally Block

-Always executes, whether an exception occurs or not.

-Used for clean-up operations (closing files, database connections, etc.).

(4)throw Keyword

-Used to manually throw an exception.

-Must be used inside a method or block.

(5)throws Keyword

-Used to declare exceptions that a method might throw.

-Used in method signatures.

3.CustomException Classes

->Java allows you to create your own exception classes by extending Exception or RuntimeException

13.Multithreading

1.Introduction to Threads

->A thread is a lightweight process that runs independently.

->Java provides multithreading to perform multiple tasks simultaneously.

->Threads share the same memory space, making them efficient but also prone to synchronization issues.

2.CreatingThreads by Extending Thread Class or Implementing Runnable Interface

->Java provides two ways to create threads:

1. Extending the Thread Class

-Override the run() method.

-Call start() to begin execution.

2. Implementing the Runnable Interface

-More flexible since Java supports single inheritance.

-Implement Runnable and pass an instance to Thread.

3.ThreadLifeCycle

->

State	Description
New	Thread is created but not started (new Thread()).
Runnable	Thread is ready to run, waiting for CPU.
Running	CPU has assigned time to the thread.
Blocked	Waiting for a resource (e.g., file, network).
Waiting	Waiting for another thread to notify (wait()).
Timed Waiting	Paused for a specific time (sleep(1000)).
Terminated	Thread has finished execution.

-->New --> Runnable --> Running --> Blocked --> Waiting --> Time Waiting -->Terminated

4.Synchronization and Inter-thread Communication

->Since multiple threads share memory, race conditions can occur.

->Synchronization ensures that only one thread accesses a shared resource at a time.

--Inter-thread Communication (wait(), notify(), notifyAll())

*wait() → A thread waits for another thread's signal.

*notify() → Wakes up one waiting thread.

*notifyAll() → Wakes up all waiting threads.

14.File Handling

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

->Java uses streams to perform input and output (I/O) operations.

--Two main types of streams:

(1)Byte Streams (InputStream, OutputStream) → for binary data.

(2)Character Streams (Reader, Writer) → for text data.

--Common file operations:

-Reading and writing files.

-Using buffers for efficiency.

-Serializing and deserializing objects.

2. FileReader and FileWriter Classes

->FileReader → Used to read data from a file character by character.

->FileWriter → Used to write data to a file character by character

3. BufferedReader and BufferedWriter

->BufferedReader → Improves performance by reading lines instead of single characters.

->BufferedWriter → Improves performance by writing lines instead of single characters.

4.Serialization and Deserialization

->Serialization → Converting an object into a byte stream to save it in a file.

->Deserialization → Converting a byte stream back into an object.

15. Collections Framework

1.Introduction to Collections Framework

->Java Collections Framework is part of java.util package.

->It provides predefined data structures (like ArrayList, HashSet, HashMap) to store and manipulate data efficiently.

--Benefits:

*Improves code reusability and performance.

*Provides sorting, searching, and iteration utilities.

*Reduces boilerplate code by handling memory management internally.

2.List,Set,Map,and Queue Interfaces

A) List Implementations (Ordered, allows duplicates)

1.ArrayList (Dynamic array)

-Fast random access (O(1))

-Slow insert/delete in the middle (O(n))

ex:-

```
import java.util.*;
class ArrayListExample {
public static void main(String[] args) {
List<String> list = new ArrayList<Strig>();
list.add("Apple");
list.add("Banana");
list.add("Cherry");
System.out.println(list)
    }
}
```

2.LinkedList (Doubly linked list)

-Fast insert/delete in the middle (O(1))

-Slow random access (O(n))

ex-

```
List<Integer> linkedList = new LinkedList<>();
linkedList.add(10);
linkedList.add(20);
linkedList.add(30);
System.out.println(linkedList);
```

B) Set Implementations (Unique elements, unordered)

1.HashSet (Uses HashTable, no order)

-No duplicate values.

-Fast lookup (O(1)) but unordered.

ex-

```
Set<String> hashSet = new HashSet<>();
hashSet.add("Java");
hashSet.add("Python");
hashSet.add("Java");
System.out.println(hashSet);
```

2.TreeSet (Sorted set)

-No duplicate values.

-Stores elements in ascending order.

ex:-

```
Set<Integer> treeSet = new TreeSet<>();
treeSet.add(5);
treeSet.add(1);
treeSet.add(3);
System.out.println(treeSet);
```

C) Map Implementations (Key-Value pairs, unique keys)

1.HashMap (Unordered key-value mapping)

-Fast retrieval (O(1)) but unordered.

ex:-

```
Map<Integer, String> hashMap = new HashMap<>();
hashMap.put(1, "Apple");
hashMap.put(2, "Banana");
System.out.println(hashMap);
```

2.TreeMap (Sorted key-value mapping)

-Stores keys in sorted order.

ex:-

```
Map<Integer, String> treeMap = new TreeMap<>();
treeMap.put(2, "Banana");
treeMap.put(1, "Apple");
System.out.println(treeMap);
```

D) Queue Implementations (FIFO Order)

1.PriorityQueue (Elements processed based on priority)

-Uses heap-based priority ordering.

ex:-

```
Queue<Integer> priorityQueue = new PriorityQueue<>();
priorityQueue.add(5);
priorityQueue.add(1);
priorityQueue.add(3);
System.out.println(priorityQueue);
```

4. Iterators and ListIterators

->Iterator: Used to traverse elements forward-only.

->ListIterator: Allows both forward and backward traversal.

---Using Iterator:-

ex:-

```
 import java.util.*;
 class IteratorExample {
public static void main(String[] args) {
    List<String> list = new ArrayList<>
(Arrays.asList("A", "B", "C"));

    Iterator<String> iterator = list.iterator();
    while (iterator.hasNext()) {
        System.out.println(iterator.next());
      }
    }
 }
```

---Using ListIterator (Bidirectional)

ex:-

```
  import java.util.*;
  class ListIteratorExample {
 public static void main(String[] args) {
    List<Integer> numbers = new ArrayList<>
 (Arrays.asList(1, 2, 3, 4));

    ListIterator<Integer> listIterator =
  numbers.listIterator();



    while (listIterator.hasNext()) {
        System.out.println("Next: " +
   listIterator.next());
    }
    while (listIterator.hasPrevious()) {
        System.out.println("Previous: " +
   listIterator.previous());
          }
        }
   }
```

16.Java Input/Output(I/o)

1.StremsinJava(InputStream, OutputStream)

->Streams in Java are used for input and output (I/O) operations. They handle data as a continuous flow of bytes or

Characters.

->Types of Streams

-Java provides two types of streams:

Byte Streams: Used for handling raw binary data.

IntputStream (for reading data)

OutputStream (for writing data)

Character Streams: Used for handling text data.

Reader (for reading character data)

Writer (for writing character data)

2.Reading And Writing Data Using Streams.

->Reading Data (InputStream)

The InputStream class is used to read data byte by byte from a file, keyboard, or network.

->Writing Data (OutputStream)

The OutputStream class is used to write data byte by byte to a file, network, or console.

3.Handling File I/o Operations.

->Java provides the java.io package to handle file input and output.