Java Assignment - I

Q. 1) What do you know about JVM, JRE and JDK ?

→ JVM :- (Java Virtual Machine) :

JVM is an abstract computing machine that provides a runtime environment for executing Java bytecode. It's responsible for executing compiled java programs.

- When we run a java program using the java command, it's the JVM that interprets the bytecode and execute the program.
- JVM : allows java program to be platform-independent, Provides memory management and Garbage collection.
- Jvm :- Performance overhead due to bytecode interpretation, Limited access to system resources compared to native applications.

→ JRE : (Java Runtime Environment) :

- JRE is a set of tools and libraries that allows java apps to be executed
- It includes JVM, necessary libraries, and other components, required to run Java applications.
- It is the environment required to run complied Java progm. It includes the Jvm and other components that provids the necessary runtime supports.
- When you install java on your system, you're actually installing the JRE. This allow you to run java app
- only provides runtime support ; does not include development tools.

- JDK - Java Development Kit
- JDK is software development environment for building, debugging, and monitoring java applications,
- It includes JRE, development tools, and libraries.

- JDK - comprehensive development package that includes tools for writing, compiling and debugging java programs.
  - It also includes, a JRE for running java applications.
  - Takes up more disk space due to the inclusion of dev tools.

real-life example

- Imagine Car (Java Program):
  - JVM - engine of car - responsible for executing instruction - driving the car,
  - JRE - its like a Car itself, providing all necessary components for a car engine to work eg fuel tank, exhaust system.
  - JDK - is like car manufacturing plant, providing all the tools needed to build and maintain cars.

(Q.2) Is JRE platform dependant or Independent?

- JRE - It is platform-independant, It provides the necessary runtime environment for Java app to execute.
  - It is designed to be platform-independent, meaning you can install and run the same JRE on different OS as long as they support Java.
  - If you have a JRE installed on a windows system, you can run Java app on it. The same JRE can be installed on a Mac or a Linux system to run the same Java app.
- Consider electricity as the JRE, It can power different types of appliances (like computers, television) regardless of their make or model. Similarly, JRE can run Java app on diff platforms.
- Platform Independence
- Dependency on JVM :- JRE relies on platform-specific JVM to execute Java bytecode.

Q.3) Which is ultimate base class in java class hierarchy? List the name of methods of it?

- Ultimate Base Class (Root Class): In Java, the ultimate base class is 'java.lang.Object'. Every class in Java implicity extends this class.
- Object is at the top of the class hierarchy and provides common methods that all java objects inherits.
- If we write public class MyClass {
  // code
  }
- In this case even though 'MyClass' doesn't explicity extend any class, it implicity extends 'java.lang.Object'.

methods of Object class.

- public boolean equals(Object obj)
- public int hashCode()
- public String toString()
- protected Object clone() throws CloneNotSupportedException
- public final native Class<?> getClass()
- protected void finalize() throws Throwable
- public final native void notify()
- public final native void notifyAll()
- public final void wait() throws InterruptedException.
- public final native void wait(long timeout) throws InterruptedException.
- ——— " ——— (long timeout, int nanos)

- Uniformity: Provides a common interface for all java Objects, allowing for consistent behavior across different classes.
- method Overriding: Allows classes to override these methods to provid custom behavior.

Q.4). Which are the reference types in Java?

- In Java, reference types are data types that store reference (memory addresses) to objects. They do not directly contain the data but instead point to the location of the data.

- Concept:
  Reference Type: They allow you to work with complex data structures and objects in Java. They include classes, interfaces, arrays, enums, and annotations.

```
class MyClass {
    int value;
}
MyClass obj = new MyClass(); // obj is a reference variable
```

OR `int[] arr = new int[5]; // arr is ref.`

ex-> Consider a library card. It doesn't contain the actual book but holds a reference to where the books can be found. Similarly, reference types hold pointers to the actual data in java.

Advantage: Memory Efficiency: ref types allow for the efficient handling of complex data structures

Dynamic memory mgt: - support dynamic allocation, deallocation of memory.

Limitation: Indirect Access:- Access data through reference req extra level of indirection, which can introduce a slight performance overhead.

Potential for Null References: Since references can be 'null', there's a possibility of null pointer exceptions if not handled properly.

Q. 5) Explain Narrowing and widening?

- Narrowing (Type Casting): Converting a data of higher size to a data type of lower size).
- Widening (Automatic Type Conversion): Converting a data type of lower size to a data type of higher size.
- Narrowing - this may lead to loss of data or precision as the destination type might not be able to represent all possible value of the source type.
- Widening - this is safe as it doesn't result in a loss of info.

ex double d = 10.5; int x = 5;
int i = (int) d; double y = x;

- ex→ Narrowing is like casting a lead actor into a minor role, where some aspects of the original role may be lost. Widening is like casting a minor actor into a lead role, where there is no loss of information.

Advantage
- Narrowing → allows you to explicitly specify the type conversion, even if data may be lost.
- Widening → Happens automatically, reducing the need for explicit type casting

Limitation :→ Can lead to data loss and possible runtime errors if not handled properly.
:→ No data loss, but potential loss of precision
eg: In floating-point conversions.

Q.5) How will you print "Hello CDAC" statement on screen, without semicolon?

=> public class Main {
    public static void main (String[] args) {
        if (System.out.printf("Hello CDAC") != null) {
        }
    }
}

- Here, the printf method prints "Hello CDAC" to the console. The 'printf' method returns a 'PrintStream" object, which is not null, so the if condition evaluates to true.
- Advantage → Achieves the task of printing without using a semicolon.
  Unconventional: This approach might be confusing for other developers who are used to seeing a semicolon at the end of statements.

Q.6) Can you write java application without main function? if yes, how?

→ yes, you can create a Java application without a 'main' function by using a static block.

public class Main {
    static {
        System.out.println("Hello CDAC");
        System.exit(0); // terminate the program
    }
}

→ Automated Greeting Message :- Consider a system that displays a greeting message when it starts, This message can be displayed automatically without the need for any user input.

Automatic Initialization :- Useful for tasks that need be performed before the program starts.

Limited Use: This approach is not suitable for applications that require user interaction or command-line-args.

Q. 8) What will happen, if we call main method in static Block?

- Calling the main method in a static block is possible, but it is not recommended. It can lead to issues like if infinite recursion and is generally considered bad practice.

```
public class Main {
    static {
        main ( new String[] { } ) ; // calling main method
    }
    psvm (String[] args) {
        System.out.println (" Hellow ") ;
    }
}
```

→ In this we called main from a static block. This can lead to an Infinite loop of method calls, which is undesirable.

→ It is not recommended

→ Infinite Recursion

Q. 9) In System.out.println, Explain meaning of every word?

→ System : This is a class in Java's core library.

→ out : This is a static member of 'System' class of type 'PrintStream'.

- println → method of the PrintStream class used to print a line of text.

Q. 10) How will you pass object to the function by reference?

- In Java, all objects are passed by reference. When an object is passed to a method, the reference to that object is passed, not a copy of the object itself.

```
class Person {
    String name;
}

public class Main {
    public static void modifyName(Person p) {
        p.name = "John Doe";
    }
    psvm() {
        Person per = new Person();
        per.name = "Jane Doe";
        modifyName(per);
        System.out.println(Per.name);
    }
}
```

- Here 'Person' object is created with the name "Jane Doe". the 'modifyName' method takes a Person object as a parameter and change its 'name' attribute to "John Doe".

- ~~Changing~~ *

- Memory Efficiency :- Passing ref instead of making copies is more memory-efficient, especially for large objects.
- Consistency with Object-Oriented Paradigm: It aligns with the concept that objects are manipulated via references.

Limitations :- Potential for Unintended Side Effects: Since you're working with the actual object, changes in one part of the code may affect other parts unexpectedly.

Q 11) Explain Constructor chaining ? How can we achieve it in C++ ?

→ Constructor Chaining : It refers to the process of one constructor calling another constructor in the same class or in its parent class. using this() or super().

→

```
class Parent {
    int x;
    Parent(int x) {
        this.x = x;
    }
}
class Child extends Parent {
    int y;
    Child(int x, int y) {
        super(x);
        this.y = y;
    }
}
```

→ Here the Child class constructor calls the constructor of its parent class, 'Parent' using super(x);

→ Imagine a scenario where a contractor is building a house. The constructor may coordinate with various teams (electricians, plumbers) to ensure the House is constructed properly. Similarly constructors in a class may coordinate to ensure an object is initialized correctly

Advantage :-

- Code Reusability :-
- Allow for reuse of initialization code.
- Ensures Proper Initialization : Helps ensure that an object is properly initialized before it is used.
- Potential for confusion : Complex constructor chains - can be difficult to understand and maintain

Q 12) Which are the rules to overload method in sub class ?

- In Java, when overloading method in a subclass, ye we must follow below rules :

Shrikant

Parameter List: The parameter list of the overloaded method must differ from the original method.

Return Type: The return type can be the same, or a subclass of the original method's return type.

Access Modifier: It can have the same or wider access modifier (e.g. if the original method is protected, the overloaded method can be 'protected' or public but not private

Exception: The overloaded method can declare exceptions from the same hierarchy or subclasses of those declard by the original method.

Q.13) Explain the difference among finalize and

Finalize :-

- It is a method in the 'Object' class. It's called by garbage collector before an object is garbage collected.
- It provides an opportunity for an object to clean up resources (eg. close files) before it is removed from memory.
- Advantages: Can be used for resource cleanup, but it's not recommended due to its unpredictable nature.
- Limitation: Not ~~guar~~ guaranteed to be called, and it's generally recommended explicity cleanup mechanisms like try-with-resources or manual resource management.

dispose :-

- dispose is a method used in GUI toolkits like Swing and AWT. It's used to release resource held by a graphical component.
- It's used to release resources like window handles, fonts, and other system resources.
- Ensures that system resources are promptly released, improving application performance.
- It is specific to GUI components and not applicable to non-GUI comp.

Q.14) Explain the difference among final, finally & finalize?

⇒ final :- - It is a keyword in Java. It can be applied to classes, methods, and variables.

- When applied to a class, it means the class cannot be inherited, when applied to method, it means the methods cannot be overridden, when applied to a variable, it means variable cannot be reassigned.
- Advantages : Provides a way to create constants, Prevents method overriding, enforce immutability.

finally :-

- 'finally' is a block in Java that follows a try-catch block. It's used to ensure that a section of code is always executed, regardless of whether an exception is thrown.
- It is used to cleanup activities like closing files or releasing resources.
- Ensures that critical cleanup code is executed, even if an exception occurs.
- Cannot be used without a preceding try-catch block.

Q.15) Explain the difference between checked and unchecked ex?

→ Checked Exception :-
- Checked at compile-time
- They represent conditions that a well-written application should anticipate and recover from.

eg → 'IOException' 'SQLException'

- Checked exception is like a predictable delay in a project schedule that can be planned for and managed.

→ Unchecked Exception :-
- Unchecked Exceptions are exceptions that are not checked at compile-time.
- They represent conditions that reflect errors in the program logic and are typically fixed.

eg NullPointerException, ArrayIndexOutOfBoundsException

- Unchecked exception is like a sudden, unexcepted hardware failure in a project, which req immediate attention.

Q.16) Explain exception chaining?

- Exception chaining is the process of associating one exception to another, This allow you to capture information about the root cause of an excepture.
- It help maintain a chain of exceptions, providing a detailed history of what went wrong.

```
try { // some code that may throw exception
} catch ( Exception e ) {
        throw new CustomException ( "Info", e );
}
```

In this example, a new CustomException is thrown with the original exception 'e' as its cause.

- Consider a relay race where the baton is passed from one runner to another. If the baton is dropped, it's important to know which runner dropped it and why. Exception chaining helps trace the original cause of an exception

Q.17) Explain the difference between throw and throws ?

⇒ throw :- - Keyword used to explicitly throw an exception.
- It is used to indicate that a specific exception has occured and provides a custom message.

Eg. throw new CustomException ("Error occurred!");

- It's like deliberately causing an issue in a production line to highlight a specific problem.

throws :- It is a keyword used in method signature to indicate that the method may throw a specific type of exception.
- It is declare to inform the caller of the method that it might encounter a particular exception.

Eg- public void myMethod() throws CustomException {
}

- It's like a warning sign on a road indicating that a certain condition (slippery road) might cause an accident.

Q.18) In which case, finally block doesn't execute ?.

- If JVM exits during execution of the 'try' or catch block (system.exit) due to System.exit()
-
- If there is an infinite loop in the try or catch block
- If the thread executing the try or catch block is interrupted or killed.

Q.19> Explain up casting.

→ Up casting is the process of casting a reference variable to a superclass type.

- It allows you to treat an object of a subclass as an object of its superclass.

```
class Animal {
    void sound() {
        System.out.println("generic animal sound");
    }
}
class Dog extends Animal {
    void sound() {
        System.out.println("Bark");
    }
}
public class Main {
    public static void main(String[] args) {
        Animal myPet = new Dog(); // Up Casting
        myPet.sound(); // Bark
    }
}
```

- Here myPet is type of Animal, but it refers to an object of type Dog. The sound() method of Dog is called.

- Consider a scenario where a pet store owner refers to all pets as 'Animals' rather than specifying the specific breed. This allow them to generalize their interactions will all Pets.

Q.20> Dynamic Method Dispatch ? ⇒

- Dynamic method dispatch is a mechanism in Java that allows a subclass to provide a specific implementation of a method defined in its superclass.

- It enables runtime polymorphism, where the method that get called is determined by the type of the actual object at runtime.

```
class Animal {
    void sound() {
        System.out.println
```

for example look for above question of Up casting.

- Imagine a car rental company. Regardless of the make or model, customers expect the "start engine" button to start the car, The specific implementation depends on the actual car they rent.

Q.21) What do you know about final method ?

→ A final method is a method that cannot be overridden by subclasses.

- It provides a way to prevent subclasses from changing the behavior of a method.

```
class Parent {
    final void display() {
        System.out.println("Display from Parent");
    }
}
class Child extends Parent {
    // Error: cannot override the final method
    // void display() {
    //     System.out("Display from Child");
    // }
}
```

- Consider a company policy that states a certain procedure must be followed without exceptions. This is like making a method as final to ensure it cannot be overriden.

Q.22) Fragile Base Class Problem and overcoming it.

→ Fragile Base class Problem :-

This is a situation where changes to a base class can break derived classes.

- It is a common problem with inheritance, which applies to java and any other language which supports inheritance

- Base class

```
class Base {
    protected int x;
    protected void m() {
                x++;
    }
    protected void n() {
                x++;
                m();
    }
}

class Sub extends Base {
    protected void m() {
                n();
    }
}
```

- If we make changes in the body of super class then we must recompile super class as well as all its sub classes. This problem is called as fragile base class problem.

- We can solve fragile base class problem by defining super type as interface.

- Use Abstraction : Hide implementation details from derived classes.

- Avoid Over reliance on inheritance : Prefer composition over inheritance

- Document contracts : Clearly document the expected behavior of base classes and methods.

23) Why Java doesn't support multiple implementation inheritance.

- Multiple implementation inheritance → ability to inherit from more than one class.

Because → ① Diamond Problem :- It can lead to the diamond problem, where a class inherits two methods with the same signature from different classes creating ambiguity.

② Complexity :- It increase the complexity of method dispatch and introduces potential conflicts.

Q.24) Explain marker interface ? List the name of some marker interfaces.

⇒ - A Marker interface is an empty interface (interface with no methods) that serve to provide metadata about the class implementing it.

- It is used to indicate to the compiles ~~to the com~~ or runtime environment that certain objects should be treated in a special way.

→
```
interface Serializable{
}
class MyClass implements Serializable{
    // class definition.
}
```

- Here Serializable is a marker interface,
- Consider a shipping company that marks packages with a special sticker to indicate they need special handling, The sticker itself doesn't do anything but provides important information

* Adv →
- Metadata Identification : way to add metadata to classes.
- Indicates Intent :- Clearly signals that a class has a certain capability or should be treated in a specific way

Limitation → No Additional Behaviour :- Unlike regular interfaces, marker interfaces don't define any methods, so they don't add behaviour to class.

i) Serializable : → class objects can be converted into a byte stream and descerialized back, This is used for I/O operations, such as Ojed persistance or network communication.

ii) Cloneable :- Indicates that the class can be cloneable cloned using the 'Object.clone()' method, Classes that implements cloneable can create copies of their objects.

iii) Remote : In the context of Java RMI (Remote method invocation) this interface is used to indicates that a class can be called remotely

iv) RandomAccess : It indicates that a list (ArrayList) support efficient random access. It doesn't have methods but is used by certain algorithms and collections to optimize operations.

v) Annotation :- All annotation types in Java implicitly extend the 'java.lang.annotation.Annotation' interface, While not additional marker interface, they serve as markers to annotate classes, methods, fields and other program elements with metadata.

→ Marker Interface are a older concept in Java, and moodern Java often use annotations and other mechanisms, to acchieve similar goals,

Q. 25) Explain the significance of marker Interfaces?

- Consider a library where certain books are marked with a "New Arrival" sticker, The sticker doesn't change the content of the book, but it provides info about its status.

- Meta data Identification
- Code flexibility

But marker interfaces don't define any methods, so they don't add behavior to a class.

```
interface Printable {
}
class Document implements Printable {
    private String content;
    public Document (String content)
        this.content = content;
    }
    public void print() {
        System.out.println(content);
    }
public class MarkerInterfaceExample {
    public static void main(String[] args) {
        Document doc = new Document("sample doc");
        // check if the obj implements the printable marker interface
        if (doc instanceof Printable) {
            System.out.println("this is printable");
            doc.print();
        } else {
            System.out.println("this doc is not printable");
        }
    }
}
```

→ Marker Interface Printable are used to add metadata to class But in real scenarios, marker interface can be used by framework and libraries to identify and handles classes with specific capabilities or requirements.