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

⇒ JVM (Java Virtual Machine):-

The JVM is responsible for interpreting the bytecode & translating it into machine code that can be executed by the underlying operating system. It also provides memory management, security, and other runtime services necessary for executing Java applications.

JRE :- rt.jar + java Virtual Machine
- JRE is platform dependend software that we can download from oracle site..
- To run java application on clients machine we must install / deploy JRE

JDK :- (Java developement kit) it contains

Java development Tools + Java Docs + rt.jar + Java virtual Machine

Java Development tools like javac, java, javap, javadoc etc

Java Docs : HTML Pages which cointains help of core java API

rt.jar : It contains core java API

Java virtual Machine :- It is abstract computer which manages execution of bytecode.

JDK is a platform dependent software that we can download from oracle site

- Developes must install JDK to develop java application

Q.2 Is JRE platform dependant or independent?

→ JRE is platform dependent, because configuration of each OS differs. The use of same byte code for all JVMs on all plateform makes java platform independent or java is platform independentden because java does not run directly on operating system. It runs on JVM which you have to install separately.

Q.2. Which is ultimate base class in java class hierarchy ? List name of methods of it ?

object class is super class of all the classes in java

object is a concrete class declared in java.lang package.

It is also called as ultimate base class / super cosmic base class / root of java class hierarchy.

Object class contains only parameterless constructor.

There are 11 methods (5 non final (2 native + 3 non native) + 6 final methods (4 native + 2 non native methods) in java

★ 5 Non final methods of java.lang.Object class
- public String toString();
public boolean equals (Object obj)
public native int hashCode();
protected native object clone();
protected void finalize()

6 final methods of java.lang.Object class

- Public final native class <?> getClass()
- public final void wait()
- public final native void wait(long timeout)

- public final void wait(long timeout, int nanos)
- public final native void notify();
- public final native void notifyAll();

Q.4. Which are the reference types in java?

→ reference type is also called as Not primitive type

These are 4 reference /non primitive type in java

- interface
- class
- enum
- array

- In case of field, variable of reference type by default contains null value.

```
Date {}
class Employee {
    String name;
    Date joinDate; // null
}
```

- To create instance of reference type, we must use new operator

- Instance of reference type get space on Heap Section.

Q.5. Explain narrowing and widening?

→ Widening :- process of converting, value of variable of narrower type into wider type is called as widening.

```
int num1=10;
double num2 = num1;
```

In case of widening conversion, explicit typecasting is optional.

narrowing :-

process of converting value of variable of wider type into narrower type is called as narrowing.

```
double num1 = 10.5d;
int num2 = (int) num1;
```

In case of narrowing, explicit type casting is mandatory.

Q.6. How will you print "Hello CDAC" statement on screen, without semicolon?

→

```
public class Demo {

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

Q.7 Can you write java application without main function? If yes, how?

→ yes, we can execute a java program without a main method by using a static block.

static block in java is a group of statement that gets executed only once when the class is loaded only once when the class is loaded into the memory by java classloader it is also known as static initialization block.

Q.8: What will happen if we call main method in static block

→ In java the main method is the entry point of java application when you execute a java program the jvm starts by invoking the main method of the class specified as the main class.

If you attempt to call the main method from within a static block of the same class or any other class, it will not have any special behaviour compared to calling main in a static block is just invoking a static method and won't change the entry point of your java application

Example 8-

```
public class Mainclass {
    static {
        // This is static block
        System.out.println ("Inside static block");
        main (new String[] {"arg1", "arg2"});
        // calling main method from the static
           block
    }
    public static void main (String[] args) {
        System.out.println ("Inside main method");
        for (String arg : args) {
            System.out.println ("Argument:" + arg);
        }
    }
}
```

Output :
Inside static block
Inside main method
Argument : arg1
Argument : arg2

As you see calling main from the static block is no different from calling any other static method. It doesn't change the entry point of your program and it won't start a new JVM execution thread or anything like that. It's just a regular method call.

The main method in java is typically invoked by the JVM itself when you run your program from the command line or an IDE

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

Ans 1. System :-
System is final class declared in java.lang package

2. out :-
out is reference of java.io.printstream class.

It is declared as public static final field inside system class. printStream is a class declared in java.io.package.

println :

println is a non static method as java.io.printStream class

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

→ In java, objects are passed by value of the refeerence.

2. This means that while the refeerence to the object is passed to the function, modification to the refeerence itself will not affect the original refeerence outside the function.

Howeve modification to the objects properties will be reflected outside the function

Example

```
Class MyObject {
        int value;
}
void modifyobject (MyObject obj) {
        obj.value = 42;
}
```

```
public static void main (String[] args) {
    MyObject myObj = new MyObject();
    MyObject myObj.value = 10;

    modifyObject (myObj);
// myObj.value is now 42 because it was
    passed by value of the reference.
}
```

In this example, the modifyObject function modifies the 'value' property of the 'myObj' object, which is reflected outside the function.

Que 11. Explain constructor chaining? How can we achive it in C++?

Ans Constructor chaining :-

To reuse body of existing constructor we can call constructor from another constructor. It is called as constructor chaining.

for constructor chaining we should use 'this' statement.

'this' statement must be first statement inside constructor.

```
class Date {
    int day;
    int month;
    int year;

    Date() {
        // Explicit call to the constructor
        this(1, 4, 2023); // constructor chaining
    }
    Date(int day, int month, int year) {
        this.day = day;
        this.month = month;
        this.year = year
    }
}
```

q. How can we achive it in c++?

1. In cpp constructor chaining is achived using a feature "member initializer list or "constructor initializer list.

2. These list are used to initialize the data members of class before the body of the constructor executes.

3. By calling one constructor from another using these initialize lists, you can chain constructor together.

```
class Myclass {
  public
    Myclass() {

    }
  Myclass(int value) : Myclass() {
      this→ value = value;
  };
      int main() {
      Myclass obj1;
    Myclass obj2(42);
      return 0;
    }
```

In this example, we have a class 'Myclass' with two constructor : default constructor & a parameterized constructor.

5 The example parameterized constructor uses constructor chaining to call the default constructor before executing it's own code

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

→ According to the oops if implementation of any method is logically same / equivalent then we should give same name to the method.

If we want to give same name to the method then we should follow same rules.

- If we want to give same name to the method & if type of all the parameters are same then number of parameters passed to the method must be different.

```
static void sum (int num1, int num2) {

}
static void sum (int num1, int num2, int num3)
{
}
```

- If we want to give same name to the methods & if number of parameters passed to the method are same then type of at least one parameter must be different.

```
static void sum (int num1, int num2) {

}
static void sum (int num1, double num2) {

}
```

- If we want to give same name to the method and it's number of parameters passed to the method are same then order of type of parameters must be different

```
static void sum (int num1, float num2){
}
static void sum (float num1, int num2){
}
```

only on the basis of different return type, we can not give same name to methods.

When we define multiple methods with same name using above rules then it is called as method overloading.

Q. 13 Explain the difference among finalize & disposed

Feature	Disposed() Method	finalize() method
1. Defined	It is defined in the interface Disposable interface	It is defined in java.lang.Object
2. Basic	It is used to close or release unmanaged resources stored by an object like files or stream.	It is used to clear unmanaged resources owned by the current object before it is distroyed.

Feature	Dispose() method	Finalize() method
3. Syntax	The syntax of dispose() method is public void dispose() { }	The syntax of finalize method is protected void finalize() { }
4. Access specifier	It is declared as public	It is declared as protected
5. Invoked	Invoked by user	Invoked by garbage collector (GC)
6. Speed	veey quickly	It is invoked more slowly than dispose()
7. peeformance	It exeecutes immediate action & has no impact on the peeformance of the site	It has an impact on the performance of the site
8. Implementation	Eveey time whenevee there is a close() function the dispose() method should be implemented.	Unmanaged resources must be implemented using the finalize() method.

Que 14. Explain difference among final, finally, and finalize ?

⇒ ★final :-

It is keyword we can use with local variable field method and class.

If we used final modifier with method local variable & field then it is consider as constant

If we use final modifier with method then we can not overide / redefine method inside sub class.

If we use final modifier with class then we can not extend it.

★finally :-

It is a block that we can use with try/catch.

If we want to close local resources then we should use finally block.

finalize :-

- It is not final method of java.lang.Object class

- If we want to release class level (which is declared as field) resources then we should use finalize method.

Q.15 Explain the difference among checked & unchecked exception?

→ checked exception and unchecked exception are types of exception in java, which are designed for java compiler (Not for JVM)

checked Exception:-

java.lang.Exception is considered as super class of all the checked exception.

java.lang.Exception and all it's subclasses except java.lang.RuntimeException (and it's subclasses) are considered as checked exception.

Example of checked exception.
- java.lang.cloneNotSupportedException
- java.lang.InteruptedException
- java.io.IOException
- java.sql.SQLException

- Compiler force developer to handle or block for checked exception.

- Unchecked Exception :-

java lang.RuntimeException is considered as super class of all the unchecked exception

- java.lang.RuntimeException and all it's subclasses are considered as unchecked exception.

- Example of unchecked exception
  - RuntimeException
  - NumberformatException
  - NullPointerException
  - NegativeArraySizeException
  - ArrayIndexOutOfBoundsException
  - ArrayStoreException
  - ClassCastException
  - ArithmaticException

- compiler do not force developers to handle or to use try catch block for unchecked exception.

Q.16 Explain exception chaining?

→ Process of handling exception by throwing new type of exception is called as exception chaining.

- Chained exceptions are associated such that the previous exception causes each exception in the chain.

- It can help debug, as it can help us track down the root cause of an error.

- We can use exception chaining in situation where another exception. However, it's important to note that chaining can make our code more difficult to read & understand. Therefore we should use exception chaining sparingly and only when necessary.

-If we use chained exceptions, it is a good idea to document the chain in our code. It'll help others understand our code & make it easier to debug if an error occures.

-for example, if an InputStreamthrows an IOException and the InputStreams read() method throws an EOFException, then the EOFException is the cause of the IOException. The following code demonstrates the use of chained exception.

```
import java.io.*;

class ChainedExceptionDemo {
 public static void main (String [] args)
        throws IOException
    { try {
        throw new IOException ("IOException
                Encapintered");
initCause (new EOFException ("root cause is
        EOFException));
    } catch (Throwble e) {

    // Handle the IOException
EOFException eof = (EOFException) e.getCause();
  System.out.println ("The cause is →" + eof);
    }
  }
}
```

In above code, exception chaining allows us to handle eof exception in a single catch block, if we want to access the original exception (the "cause"), we can use the getCause() method

Q. 17 Explain the difference among throw & throws

→ throw :-

- It is keyword in java

- If we want to genesate new exception the we should use throw keyword.

-Only objects that are instances of Throwable class (or one of it's subclasses) are thrown by the jvm or can be thrown by the java throw statement.

Steing ex = new steing ("Divide by zero exception")

throw ex; // No exception of type steing can be thrown: an exception type must be a subclass of Throwable.

- throw statement is a jump statement

2. throws :-

- It is keyword in java.

- If we want to delegate exception from method to the callee method then we should use throws keyword / clause.

```
public class program {

public static void displayRecord() throws
        'InterruptedException'
    {
    for (int count=1; count<=10; ++count)
    }
System.out.println ("Count :" +count);
  Thread.sleep (500);
  }
  }
public static void main (steing[] args)
    {
  try {
    Program : displayRecord ( );
  } catch (InterruptedException e) {
   e.printStackTrace();
  }
  }
  }
```

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

→ The finally Block may not execute if the jvm exits while the try or catch code is being executed.

- The try block of the writeList method that you've been working with here opens a printWriter. The program should close that stream before existing the writeList method.

Q. 19 Explain Upcasting ?

→ Super class reference variable can contain reference of : instance of subclass It is called as upcasting

- Upcasting is the typecasting of a child object to a parent object.

- Upcasting can be done implicitly Upcasting gives us the flexibility to access the parent class members but it is not possible to access all the child members using this feature

- Instead of all the members, we can access some specified members of the child class

- for instance, we can access the overridden methods.

6. Example :-

Lets there will be an animal class. There can be manage different classes of animals. One such class is Fish.

so lets assume that the fish class extends the animal class.

Therefore the two way of inheritance in this case is implemented as

```
Object O1 = new Boolean (true); //upcasting
Number n1 = new Byte (123);//upcasting
Byte B1 = new Byte (123);

object O5 = new Integer (123456);//upcasting
```

Q.20 Explain Dynamic method dispatch

→ Dynamic method dispatch is the mechanism in which a call to an overridden method is resolved at runtime instead of compile time. this is an important concept because of how java implements run-time polymorphism.

- Java uses the principle of a superclass reference variable can refer to subclass object" to resolve cans to overridden method at run time.

- When a superclass reference is used to call an overridden method, java determines which version of the method to execute based on the type of object being referred to at the time call.

-In other words, it is the type of object being referred to that determines which version of an overridden method will be executed.

Advantages of dynamic method dispatch.

1. It allows java to support overridding of methods which are important for run-time polymorphism.

2. It allows a class to define methods that will be shared by all it's derived classes, while also allowing these sub-classes to define their specific implementation of a few or all of those methods.

3. It allows subclasses to incorporate their own methods & define their implementation

Que 21 What do you know about final method?

→ see Que. No 14

Q. 22. Explain fragile base class problem & how can we overcome it.

→ If we make changes in the body super-class then we must recompile superclass as well as all it's subclasses. this problem is called as fragile base class problem.

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

Q.23. Why java does not support multiple implementation inheritance?

→ class A {}
class B {}
class C extends A, B {} // Not ok
// Multiple implementation inheritance

- In c++ combination of two or more than two types of inheritance is called hybrid inheritance.

- If we combine hierarchical inheritance & multiple inheritance then it becomes diamond inheritance which creates some problem.

To avoid diamond problem, java do not support multiple implementation inheritance/ multiple class inheritance.

- conclusion : In java, class can extends only one class.

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

Ans An interface which do not contain any member is called as marker / tagging interface.

- Marker interface are used to generate metadata It helps JVM to perform some operations eg. to do done serializing state of java instance etc.

- Example
  - java.lang.cloneable
  - java.util.EventListener
  - java.util.RandomAccess
  - java.rmi.Remote

Q. 25. Explain the significance of marker interface?

Ans it's significance lies in it's ability to mark or tag a class as having a specific characteristics or behaviours. Without providing any implementation details. Here's a brief explaination of the significance of marker interface.

- Identification : Marker interfaces are used to identify or classify classes based on certain criteria or characteristics. When a class implements a marker interface it indicates that the class possesses specific traits or compabilities associated with that interface

2. Semetic Information :-

They provide semetic information to both programmers and the runtime environment. this can help developers understand the intended use or behaviour can help developees understand the intended use or behaviour of a class without needing to examine its source code

3. Compile-time & Runtime checks :

Marker interfaces are often used in situation where you want to perform compile-time or run-time checks based on a class's characteristics. for example in java the Serialization marker interface is used to indicate that a class can be serialized & this information is used during object serialization & deserialization

4. Refection & Framework :

Marker interfaces are sometimes used in reflection & by frameworks or libraries to discover & work with classes that meet specific criteria. for instance, a framework might search for classes implementation a peeticular marker interface and apply ceetain behaviours or rules tothem.

5. Customization & Extension :

Developers can create their own marker interface to customize or extend the behaviour of their classes within a framework or application. By implementing a specific marker interface, classes can opt into additional functionality or behaviours.

6. Documentation & contracts :-

They serve as a form of documentation or contract between developers. By adhering to marker interfaces, developers signal their intention to follow certain conventions or requirement.

In summary, marker interfaces are a way to provide metadata about a class's capabilities or characteristics aiding in code organization, communication & the application of specific behaviours or rules.

They are a simple but effective mechanism for adding meaning to classes in object oriented programming.