Page No.
Date

1] JVM (Java Virtual machine) - It is an abstract machine which do not have it's physical existence. It is a specification that provides runtime environment in which java bytecode can be executed. JVM can also run those programs written in other languages and compiled to Java bytecode. JVM is an important part of JDK and JRE. Whatever Java program is ~~run~~ using JRE or JDK goes into JVM and JVM executes the program line by line, hence it is also known as interpreter. JVM largely helps in abstraction of inner implementation from programmer who make use of libraries for their programs from JDK.
It is responsible for three activities → ① loading ② linking ③ Initialization.

JDK (Java Development Kit) — It is a software development environment used for developing Java applications and applets. It physically exists and contain JRE, development tools. JDK contains a private JVM, resources like interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc) to complete the development of java application.

JRE (Java Runtime Environment) - It is an installation package that provides an environment to only run the java program (or application) and not to develop one onto machine. The JRE provides minimum requirement for executing a Java application; it consist of JVM, core classes and supporting libraries.

JVM, JRE and JDK are platform dependent because configuration of each OS is different from each other.

Page No.
Date

Q] Is JRE platform independent or dependent?

→ 2] JRE (Java Runtime Environment) is platform dependent as it requires different configuration for different OS. Also it contains platform dependent files which are used to create JVM in the desired environment.

Q] which is ultimate base class in java class Hierarchy? List the name of methods of it?

→ Object class is the ultimate base class in Java. By default it is the parent class of all other classes. Object class is present in java.lang package. If any class doesn't extend any other class then it is a direct child class of object and if it extends another class then it is indirectly derived. So, object class methods are available to all Java classes.

Methods of object class are —

1. toString() method
2. hashCode() method
3. equals(Object obj) method
4. finalize() method
5. getClass() method
6. clone() method
7. wait() method
8. notify() notifyAll() method

Page No.
Date

Q] which are reference types in Java ?

4] → Java provides two types of data types – primitive and Reference.

primitive data types are predefined in Java that serves as fundamental building block while reference data type refers to where data is stored.

Reference contains address (or reference) of dynamically created objects. All reference types are subclass of type java.lang.object.

The reference types in Java are –

Class, interface, String, Arrays.

Class – A class is a set of objects which shares common characteristics / behaviours / attributes. It is a user defined blueprint / template from which objects can be created.

Arrays – Array in Java is non primitive data type that store elements of a similar data type in the memory.

String – String is type of object that can store sequence of character values. A string acts the same as an array of characters in Java.

Interface – An interface is a fully abstract class that includes a group of abstract methods and static constants. Interface is used in Java to achieve abstraction.

Page No.
Date

Q] Explain narrowing and widening ?

5] There are eight primitive datatypes in Java - Byte, short, int, long, float, double, char and boolean.
These data types can be converted into one another implicitly or explicitly. This process is called Type casting.
Implicit typecasting takes place automatically whereas explicit typecasting is to be done by programmer.
There are two types of Type casting →

① Widening Type casting
② Narrowing Type casting

Widening Type casting →

- Converting a lower data type into a higher one is called as widening type casting.
- It is also called as implicit conversion or casting down.
- It takes place when both data types are compatible with each other and the target type is larger than source type.
- It is safe because no chance to lose data.

Example

```
class Test {
    public static void main (String args [ ]) {
        int num1 = 10 ;
        float num2 = 15.5f;
        long num3 = num1 ;
        System.out.println ("Values are : " + num1
                + " " + num3 ) ;

        double num4 = num2 ;
        System.out.println ("Values are : " + num2 + " "
                + num4) ;
    }
}
```

output – Values are : 10 10
Values are : 15.5 15.5

Narrowing typecasting –

- Converting a higher data type into a lower data type is called narrowing type casting.
- It is also known as explicit conversion or casting up.
- When we try to assign higher data type to lower data type without typecasting manually, we get a compile time error stating "incompatible types : possible lossy conversion". To avoid this error, we need to use cast operator explicitly.

```
class Demo {
  public static void main (String args [] ){
    byte b;
      int a = 1234;
    // byte b = a ; // gives error of possible
                    lossy conversion.
      byte b = (byte) a ;

      System.out.println ("Value is : " + b);
  }
}
```

Output – Value is : 1234

Page No.
Date

Q] Print "Hello CDAC" statement on screen without semicolon?

→ 6] Every statement in Java needs to end with a semicolon. But there are few scenarios when we can write a executable program without semicolon.

Way 1 → Using if statement.

If we place statement inside if statement with a blank parenthesis, we don't get error when we end statement without semicolon.

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

        }
    }
}
```

Output – Hello CDAC

Way 2 → Using append () method of StringBuilder class –

```
import java.util.*;

class Demo {
    public static void main (String args [ ]) {

        if (System.out.append ("Hello CDAC") ==
                null) {

        }
    }
}
```

Output – Hello CDAC

Page No.
Date

Way 3 → Using equals method of string class

```
import java.util.*;

class Demo{
    public static void main (String args [])
    {
        if (System.out.append ("Hello CDAC").equals
            (null)) {

        }
    }
}
```

output – Hello CDAC.

Page No.
Date

Q] Can you write java application without main function? How?

A] → Yes, we can run Java application without main method. There are different types of execution models available in Java. Example - Applets, that run on browser don't have main method. Instead they have life cycle methods like init(), start() and stop() that control execution.

Similarly servlet is Java program that works on call back mechanism, it has methods like init(), service(), destroy().

Another is MIDlet which runs on mobile devices. Even they have methods like startApp(), pauseApp() and destroyApp() to start, pause and shutdown mobile applications since this MIDlet is also a Java program, one can say that they can run program without main method.

So, we can run a Java application without main method in a managed environment like Applet, servlet, MIDlet which runs under control of browser, server and mobile device, but can't run a core Java program without public static void main(String args[]) { } method.

Q] What will happen, if we call main method in static block?

→ 8] The main method is the entry point of a Java program and it's called by the Java Virtual Machine (JVM) to start the execution of the program. Static blocks are used for static initialization of a class, and they are executed when the class is loaded into the JVM. Calling main method within a static block won't give any expected outcome. It will end up with some unexpected behaviours or errors.

Example →

```
Class Demo {
    static {
        public static void main (String args [] ) {
            System.out.println ("Hello");
        }
    }
}
```

Output — ERROR.

- Hence it is not recommended to call the main method explicitly within a static block.

Q] In System.out.println explain meaning of every word ?

→ A] In Java, System.out.println() is a statement that prints the arguments passed to it.
It is a method that prints a message to standard output (console) and appends a newline character.

- System is a built-in final class in java.lang package. It cannot be instantiated and provides access to standard input, output and error streams.
- Out is an instance of system class and represents the standard output stream. It is of type PrintStream and is responsible for writing data to output ~~system~~.Stream.
- println() method is an overloaded method of printstream class. It is used when we want to print our result in two separate lines. It throws the cursor to the next line after displaying the results.

Example -

```
class Demo {
    public static void main (String args[]) {
    System.out.println ("Interview Questions
                        Assignment");
    System.out.println ("Assignment completed");
    }
}
```

Output — Interview Questions Assignment
Assignment completed.

Q] How will you pass object to function by reference?

10] → Calling a method by reference means passing a reference (variable's location) as parameter. The non primitive data types (class, objects, array, string and interfaces) are always passed ~~by~~ as references.
Though Java is strictly call by value when we pass the reference of object it creates a copy of reference and then passes it as value to the method.
The copy reference also points to same address, so all the changes also reflect in the main method.

Example-

```
class Demo {
    int num = 10;
  // passing object as parameter
  public static void increment ( Demo d) {
        d.num = d.num + 1;
    System.out.println ("Value in method is :"
                        + d.num);
  }

  public static void main (String args [ ]) {
        Demo d = new Demo ( );
    System.out.println ("Value before method call :"
                        + d.num);
        increment (d);
    System.out.println ("Value after method call :"
                        + d.num);
  }
}
```

output → value before method call : 10
value in method is : 11
value after method call : 11

Page No.
Date

Q] Explain constructor chaining? How can we achieve it in c++?

11] → Constructor chaining is the process of calling one constructor from another constructor with respect to same object that has been currently active.

Constructor chaining process is used when we want to perform multiple tasks in a single constructor rather than creating a code for each task in a single constructor we can create a separate constructor for each task and make their chain which makes the program more readable.

Constructor chaining can be achieved in following ways—

a] Using this() keyword — When we have multiple constructors in same class, one constructor can call other constructor using this() keyword.

b) Using super() keyword — In case of different classes, constructors in child class can call constructor of Base class using super() keyword.

c) Constructor chaining occurs through inheritance. Before child class constructor, superclass / parent class constructor is called first. This ensures that creation of child class object starts with initialization of the data members of parent class.

In case of constructor chaining, there could be any number of constructors and order of constructor calling will be from topmost parent class constructor to the last child class constructor.

Q] What are rules to overload method in subclass?

12] → If a class has multiple methods having same name but different parameters, it is known as Method overloading. Method overloading in Java is also known as compile time polymorphism, static polymorphism or Early binding.

Rules to overload method –

1] When multiple methods have same name, the number of parameters in it should be different.

Ex →

```
class Addition {
    int sum (int a, int b)
    {
        return (a+b);
    }
    int sum (int a, int b, int c)
    {
        return (a+b+c);
    }
}
class Test {
    public static void main (String args []) {
        Addition obj = new Addition ();
        obj.sum (10, 20);
        obj.sum (10, 20, 30);
    }
}
```

2] When both name and number of parameters of multiple methods are same, then the type of parameters should be different.

```
class Addition{
    int sum (int a, int b) {
        return (a+b);
    }
    int sum (int a, float b) {
        return (a+b);
    }
}
class Test {
    public static void main (String args[]) {
        Addition obj = new Addition ();
        obj.sum(10, 20);
        obj.sum (15, 30.5);
    }
}
```

3] If method name, no. of parameter and type of parameter is same, then order of parameters should be different.

```
class Addition {
    int sum (int a , float b ) {
        return (a+b);
    }
    int sum (float a, int b) {
        return (a+b);
    }
}
```

```
class Test {
    public static void main (String args [ ]) {
        Addition obj = new Addition ( );
        obj. sum (10, 20.5);
        Obj. sum (15.2, 13);
    }
}
```

If any of above rule is violated, then a compile time error saying "method already defined" is thrown by compiles.

Q] Explain the difference among finalize and dispose ?

→ 13] The finalize () and dispose () methods are use to liberate the unmanaged resources kept by an object.

finalize () method –

It is described in Object class. When an object's reference is not used for an extended time, the garbage collector calls this function. The garbage collector calls finalize () method shortly before entirely destroying the item. This method has been designated as protected and is not marked as public to prevent other classes from accessing it. This method may reduce program speed because it does not immediately free memory.

dispose () method –

It releases any unmanaged resources owned by a class object. These resources include files, data connections and many others. It is defined in the interface IDisposeable and the class implements it by implementing the interface IDisposable. It is not invoked by default and the programmer must manually implement it when creating a custom class that others will use.

Q] Explain the difference among final, finally and finalize ?

→ ~~18~~ 14] Final, Finally and Finalize are keywords in Java used for exception handling.

Final — It is a keyword and access modifier which is used to apply restrictions on class, method or variable.
A variable once declared final cannot be modified.
A final method cannot be overridden by sub class.
A final class cannot be inherited.

Finally — It is a block in Exception Handling. It executes the important code whether exception occurs or not. This block is always related to try and catch block in Exception handling, so it gets executed as soon as try and catch block executes. It clears up all the resources used in try block.

Finalize()— It is a method of object class used to perform clean up processing just before object is garbage collected. finalize() method is used with objects and it gets executed just before object is destroyed.

Page No.
Date

Q] Explain the difference among checked and unchecked exception?
→15] In Java, Exception is an unwanted or unexpected event that occurs during execution of program.

Checked Exception →
These are exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword.
Example – File Not Found Exception, IO Exception

Unchecked Exception →
These exceptions are not checked at compile time. Since these are unchecked hence it is programmers duty to specify or catch them.
Example – Error and RuntimeException classes, Arithmetic exception.

Q] Explain exception chaining?
→16] Exception chaining occurs when one exception causes another exception. The original exception is the cause of the second exception. We can use chaining of exceptions in situations where they can help debug and track down the root cause of an error. However this can make our code more difficult to understand so it should be used only when necessary.

Example – If an InputStream throws an IOException and InputStream's read() method throws an EOFException, then EOFException is the cause of IOException.

Page No.
Date

Example - Consider a situation in which a method throws an Arithmetic Exception because of an attempt to divide by zero but the actual cause of exception was an I/O error which caused divisor to be zero. The method will throw only ArithmeticException to end user. So, the user will not come to know actual cause of exception.

Q] Explain the difference among throw and throws?

→ [7] The throw and throws is the concept of exception handling in Java.

throw → throw keyword in Java is used to throw an exception explicitly in code, inside function/method or block of code. Using throw keyword, we can only propagate unchecked exception. throw is used within the method and it is followed by an instance of Exception to be thrown. Using throw keyword, we can throw only one exception and not multiple exceptions at a time.

throws → throws keyword is used ~~at~~ in the method signature to declare an exception which might be thrown by the function while execution of code. Using it, we can declare both checked and unchecked exceptions but it can be used to propagate checked exceptions only. The throws keyword is followed by class names of Exceptions to be thrown. We can declare multiple exceptions using it that can be thrown by the method.
Example - main() throws IOException, SQLException.

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

→ 18] The Java finally block is always executed whether an exception is handled or not. It will also get executed if a try block exits by using return, break or continue statements.

However there is a possibility where finally block will not be executed.

- When the System.exit() method is called in try block before execution of finally block, then it will not be executed.

Example →

```
package test;

public class FinallyBlock {
    void m1() {

        try {
    System.out.println("In try block");
    System.exit(0);
    }
    finally {
      System.out.println("In finally block");
      }
      System.out.println("outside Finally block");
    }
}

    public static void main(String args[]){
        FinallyBlock obj = new FinallyBlock();
          obj.m1();
      }
}
```

Output - In try block.

- From the program, when System.exit(0) method is called in try block, finally block and statement after finally block are not executed.

Q] Explain upcasting ?

→ [9] In Java, the object can also be typecasted like the datatypes.
Upcasting is a type of object typecasting in which a child object is typecasted to a parent class object. Upcasting is also called Generalization and widening. It can be performed implicitly or explicitly. Using it, we can access variables and methods of parent class to the child class.

Example

```
class parent {
    void printData() {
        System.out.println("Method 1");
    }
}
class child extends parent {
    void printData() {
        System.out.println("Method 2");
    }
}
class Test {
    public static void main(String args[]) {
        parent obj1 = (parent) new child(); // upcasting
        obj1.printData();
    }
}
```

output → Method 2.

Page No.
Date

Q] Explain dynamic method dispatch?

Ans] → Dynamic method Dispatch in Java is the process by which a call to an overridden method is resolved at runtime (during the code execution).
It is another name given to run time polymorphism.
The basis of dynamic dispatch method starts with inheritance where two or more classes exhibit a parent child relationship. In this, overriden method is called through reference variable of superclass. Determination of method to be called is based on object being referred to by reference variable.

Example →

```
class A {
    public void display () {
        System.out.println ("Hello A");
    }
}
class B extends A {
    public void display () {
        System.out.println ("Hello B");
    }
}
class Test {
    public static void main (String args []) {
        A obj = new B ();
        obj.display ();
    }
}
```

Output – Hello B.

Java allows one to give reference of superclass to object of subclass.

Page No.
Date

When both superclass and subclass have same methods in it, then which object method would be called and displayed is decided by JVM at runtime, hence it is a runtime polymorphism. Here in this case, object of subclass will be prioritised, so display method of B will be shown.

Q] What do you know about the Final method ?

21] → Whenever we want that content of method should not change by any outsider or child class, then we declare the method as final method in Java.
The final keyword before the ~~class~~ name of method indicates that the method cannot be overridden by subclass.

Real life example — Suppose we make a class car and declare a method wheels () which is not final in it.
If someone makes a new company class car and extends the class car and overrides wheels () method and changes number of wheels to something odd (3, 5, 7) etc then actual structure of car is changed.
To restrict this unwanted and improper use, final methods are created.

Example

```
Class Demo1 {
    public final void display () {
        System.out.println ("Final method");
    }
}

public class Demo2 extends Demo1 {
    public final void display () { // Error
        System.out.println ("trying to override");
    }
    public static void main (String args [ ]) {
        Demo2 obj = new Demo2 ();
        obj.display ();
    }
}
```

Output - Error The overridden method is Final.

Q] Explain fragile base class ~~prom~~ problem and how can we overcome it?

→ 22] Fragile base class problem is fundamental architectural problem of object oriented programming system where base classes are considered fragile because seemingly safe modification to base class when inherited by derived class may cause the derived classes to malfunction. The programmer cannot determine whether a base class change is safe simply by examining the methods of base class isolation.

Example

```
class super {
  private int counter = 0;
  void inc1() {
     count++;
  }
  void inc2() {
    count++;
  }
}
class sub extends super {
    @override
    void inc2() {
       inc1();
    }
}
```

Calling dynamically bound method inc2() on instance of sub will correctly increase the field counter by one.

If however the code of the superclass is changed in following way,

```
class Super {
    private int counter = 0;
    void inc1() {
        inc2();
    }
    void inc2() {
        counter++;
    }
}
```

a call to dynamically bound method inc2() on instance of sub class can cause an infinite recursion between itself and the method inc1() of superclass and eventually cause a stack overflow.
This problem could have been avoided by declaring methods in superclass as final which would make it impossible for a subclass to override them. However this is not always desirable or possible. Therefore, it is good practice for super classes to avoid changing call to dynamically bound methods.

Q] Why Java does not support multiple implementation inheritance?

→23] Multiple inheritance is a feature of object oriented concept where a class can inherit properties from more than one parent class.

It is not supported by Java because using classes and handling the complexity that causes due to multiple inheritances is very complex.

The problem occurs when there exists methods with same signature in both parent and child class. on calling the method, the compiler cannot determine which class method to be called and even on calling which class method gets priority.

Example→

```
class first {
    void display () {
        System.out.println (" First ");
    }
}

class second {
    // System.out.println (" Hi ");
    void display () {
        System.out.println (" second ");
    }
}

class Demo extends first, second {
    public static void main (string args [ ]) {
        Demo obj = new Demo ();
        obj.display ();
    }
}
```

Page No.
Date

Output - Compilation error is thrown

From the code, compiler faces ambiguity in which method here display() method to be called, of first class or second class.

Therefore to avoid such complications, Java doesnot support multiple inheritances of classes.

Q] Explain marker interface. List name of some marker interfaces?

=> 2Q] An interface that does not contain methods, fields and constants is known as marker interface. It is also called as marker interface / tag interface / empty interface. Marker interface delivers runtime type information about an object so the compiler and JVM have additional information about it. The declaration of marker interface is same as interface in Java but the interface must be empty.

Syntax →

```
public interface MyMarkerInterface {

}
```

The main purpose behind ideology of marker interfaces is that they are used to convey a message to the JVM that the class implementing this type of interface has some extra functionalities.

Some of marker interfaces in Java include cloneable, Serializable, Remote interface.

a) Cloneable interface – It is used to indicate JVM that clone() method of object class can be used by objects of the class that have implemented cloneable interface.

b) Serializable interface – To save state of an object of class into physical memory or retrieve state already stored in physical memory, class must be made serializable by implementing Serializable interface.

c) Remote interface – It is used for Remote method invocation purposes.

Page No.
Date

Q] Explain the significance of marker interface?

→ 25] Marker interface is an empty interface having no fields, methods or constants.

- Marker interface is used as a tag that inform the Java compiler by a message so that it can add some special behaviour to the class implementing it. As it indicates signal/command to JVM, it is reason that JVM and compiler have additional information about an object.
- If we have some information about the class and that information never changes, in such cases, we use marker interface to represent the same.
- Implementing a marker interface / empty interface, it will tell the compiler to do some operations.
- Marker interface is used to logically divide the code and it is a good way to categorize code.
- It is useful for developing API and in Frameworks like Spring.