2019 / 2020

a) **Bytecode is executed by JVM.**

The output of a Java compiler is bytecode, which leads to the security and portability of the Java code. It is highly developed set of instructions that are designed to be executed by the Java runtime system known as JVM. (Java virtual Machine). The Java program executed by the JVM that makes the code portable & secure. Because JVM prevents the code from generating its side effects. The Java code is portable, as the same byte code run on any platform.

b) x = 9 Y = 5 z = 10

++Z + y - y + z + x++

11 + 5 - 5 + 10 + 9 = 25

c) **Static block, instance block, constructor, method.**

* The static block will execute whenever the class is loaded by JVM.
* Instance block will execute whenever an object is created, & they are invoked before the constructors. If there are two objects, the instance block will execute two times for each object.
* The constructor will execute after the instance block & it also execute every time the object created.
* A method always executed at the end.

d) **nameless object :- An object that has no reference.**

It is basically referred to as anonymous objects. They don't have any names. Also we can say, when an object is initialized but is not assigned to any reference variable.

EX :- new Employee ( ) ;

e) class A has name A

class B has name B

f) Tight coupling

In tight coupling a group of classes & objects are highly dependent on each other. Tight coupling is also used in some cases, like when an object creates some other objects that are going to be used by them.

Cohesion :- It defines how the classes in Java are designed. It is closely related to ensuring that the purpose for which a class is getting created in Java is well-focused & single.

Loose coupling :- In loose coupling, a method or class is almost independent, & they have less depend on each other.

Cohesion vs Coupling : Cohesion is about how well elements within a module belong together & serve a common purpose. Coupling is about how much one module depends or interacts with other modules.

g) A listener is an object that is notified when an event occurs.

h) addKeyListener()

Java KeyListener() is notified whenever you change state of key.

addKistener :-

eventKeyboardListener() :-

i) Primitive data types in Java

byte, short, int, long, float, double, char, boolean

Primitive data type & an object of a wrapper class

j) Primitive data types stored directly on the stack.
when we create a string using new operator, it always created in a heap memory

k) In Java arrays are object references.

l) All. (A letter, underscore, $)

m) { { 9, 2, 1 }, { 1, 4, 8 } }
The first if will change an odd number to an even. The second if will also execute after an odd number has been made even. Both loops start at index 1 so this only changes the items in the second row & second and third column.

row < arr.length (2)

	0	1	2
0	3	2	1
1	1	(2)	(3)

n) The final keyword is a non-access modifier used for classes, attributes and methods, which make them non-changeable (imposible to inherite or override)
we cannot create an instance of an abstract class as it does not have any complete implementation.
An interface can extend other interfaces in Java.
List, set, Map are interfaces in Java.
1, 3

o) The synchronized keyword locks a single thread with the shared data so that no other thread can access it.
The synchronized keyword can only Synchronize a block or a method.

3) Sychronization in Java is the capability to control the acc of multiple threads to any shared resource. In the Multithreading concept, multiple threads try to access the shared resources at a time to produce inconsistent result. The sychronization is nesessary for reliable communication between threads.

4) 4 is correct. It defines an anonymous inner class instance, which also means it creates an instance of that new anonymous class at the same time. The anonymous class is an implementer of the runnable interface, so it must override the run() method of Runnable

1) Inheritance is the ~~fea~~ feature which is used to reduce the use of nested classes.

5) objects can be passed by references. so iii is false.

6) protected access is used to make the members private. But those members can be inherited. This gives both- security & code reuse capability to a program. Thus answer is 3.

2) a) **Abstraction :-**

It is used to hide the unnecessary information or data from the user but shows the essential data that is useful for the user. Abstraction can be achieved with either abstract classes or interfaces.

The abstract keyword is a non access modifier, used for classes & methods.

Abstract class :- is a restricted class that cannot be used to create object.

Abstract method :- can only be used in an abstract class, and it does not have a body.

Example :-

```
// abstract class
    abstract class Animal {
// Abstract method (does not have a body):
        Public abstract void animalSound ();
        // Regular method
        Public void sleep () {
            System.out.println("Zzz");
        }
}
```

b)

Abstract class	Inteface
Abstract class can have abstra -ct & non abstract methods.	Interface can have only abstrac -t method.
Abstract class doesn't support multiple inheritance.	Interface support multiple inheritance.
Abstract class can have final, non-final, static & non-static variables.	Interface has only static & final variables.
The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
An abstract class can ~~have~~ be extended using keyword "extend".	An interface can be imple -mented using keyword "implements".

c). **Encapsulation.**

It is used to bind up the data into a single unit called class. It provides the mechanism which is know as data hiding.

* Declare class variables /attributes as Private.
* Provide public get & set methods to access & update the value of a private variable.

Example :-

```
Public class Main {
    Public static void main (string [] args) {
        Person myObj = new Person ( );
        myObj. setName ("John");
        System. out. println (myObj. getName ( ));
    }
}
```

d) **Benifits of Encapsulation.**

Protect your data :- You can keep your data and codes safe from external inheritance.

Easy to test code :- The code which is encapsulated is simple to debug & easy to test for unit testing.

Flexible :- The encapsulated code is cleaner, flexible & easy to change as per our needs.

Easy to reuse.

03) a) **Exeeption.**

In Java, an exception is an event that disrupts the normal flow of the program. If the exception object is not caught and handled properly, the interpeter will display an error message as shown in the output of the previous program & will terminate the program.

b) checked Exceptions

They occur at compile time. The compiler check for a checked exeption. These exceptions can be handled at the compliation time The JVM requires that the exception be caught and handled.

unchecked Exceptions

They occur at runtime. The compiler doesn't cheack for these kinds of exceptions. These kinds of exceptions can't be caught or handled during compliation! The JVM doesn't require the exception to be caught & handled.

c)

```
class MyException extends Exception {
    Public MyException (String s)
    {
        super (s);
    }
}
Public class main {
    Public static void main (String args [])
    {
        try {
            throw new MyException ("GeeksGeeks");
        }
        catch (MyException ex) {
            System. out. println ("caught");
            System. out. Println (ex. getMessage ());
        }
    }
```

Java provides us the facility to create our own exceptions which are basically derived classes of exception. Creating our own Exception is known as a custom exception or user defined exception. Basically Java custom exceptions are used to customize the exception according to user needs.

d) **static variable :-**

when a variable is declared as static, then a single copy of the variable is created and shared among all objects at the class level. They are essentially global variables.

Example :-

```
Class Test {
    // static variable
       static int a = m1();
    // static block
       static
       {
         System.out.println("Inside static block");
       }
    // static method
       static int m1()
       {
         System.out.println("from m1");
         return 20;
       }
    // static method (main !!)
       public static void main(String[] args)
       {
         System.out.Println("value of a: " + a);

       }
}
```

4) a) **Java package :**

A java package is a group of similar types of classes, interfaces & sub package. Package in Java can be categorized in 2 form, built in package & user-defined package.

uses of Java package

Preventing naming conflicts.

Making searching / locating & usage of classes, interfaces, enumerations & annotations easier.

Providing controlled access.

Packages can be considered as data encapsulation.

b) **Multithreading** is a Java feature that allows concurrent execution of two or more part of a program for maximum utilization of cpu. Each part of such program is called a thread. So, threads are light weight processes within a process.

Advantages :

1) Responsiveness : Multithreading in an interactive application enables a program to continue running even if a section is blocked or executing a lengthy process, increasing user responsiveness.

2) Resource sharing : Process can only share the resources only via techniques such as message passing & shared memory.

3) Economy : Allocating memory & resources for process creation is an expensive procedure because it is a time & space consuming task.

4) Better communication : Thread synchronization functions could be used to improve inter-process communication.

creating a thread

c) Implementing the java.lang.Runnable Interface

Extending the java.lang.Thread class.

①

```
import java.io.*;
class GFG implements Runnable {
  Public static void main (String
  arg[])
  {
    GFG gfg = new GFG();
    Thread t = new Thread (gfg, "gg");
    t.start();
    System.out.println (t.getname());
  }
  @Override public void run()
  {
  System.out.println ("Insido");
  }
}
```

②

```
import java.io.*;
class GFG extends Thread {
  Public void run()
  {
    System.out.println ("welcome");
  }
  public static void main (String [] args)
  {
    GFG g = new GFG();
    g.start();
  }
}
```

d) **Runtime Polymorphism**

* The call is not resolved by the compiler.
* It is also known as Dynamic binding. Late binding & overridding as well

example :-

Complile time polymorphism

* The call is resolved by the compiler.
* It is also known as static binding. Early binding & overloading as well

example :-