Dop (29.08.20)

Question 01 : a) Java is platform independent and portable. - Justify.

b) Explain the key feature of object Oriented Programming.

c) what's the difference between = , = = , and .equals() ?

Answer :

1) a) The meaning of platform-independent is that the java compiled code (byte code) can run on all operating systems. A program is written in a language that is a human-readable language. It may contains words, phrases, etc which the machine does not understan For the source code to be understood by the machine-level language. So, here comes the role of Compiler. The compiler converts the high-level language ~~so, there~~ (human language) into a format understood by the machines.

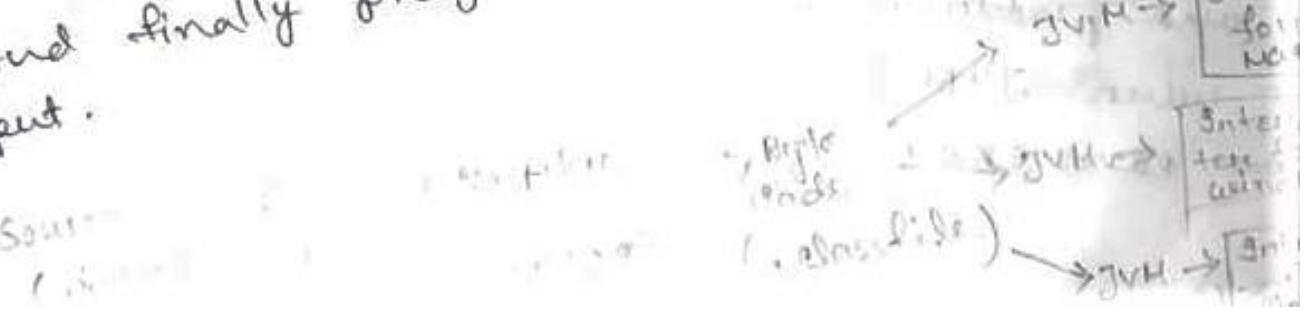
Therefore, a compiler is a program that translate the source code for another program from a programming language into executable code.

This executable code may be a sequence of machine instructions that can be executed by

the CPU directly, or it may be an intermediate representation that is interpreted by a virtual machine. This Intermediate representation in Java is the Java Byte code.

step by step Execution of Java Program.

1. Whenever, a programe is written in JAVA, the Javac compiler is.
2. The result of JAVA compiler is the .class file or the bytecode not the machine native code (unlike C compiler)
3. The bytecode generated is a non-executable code and needs an interpreter to execute on a machine. The interpreter is the JVM and the the Byte code is executed by JVM
4. And finally program runs to give the desire Output.



Java is portable due to following features:

1. Output of Java code is in Bytecode (ie. Non Executable code).
2. Bytecode is highly optimized set of instructions
3. Byte code is executed by machine which is Java runtime machine is also call as JVM.
4. Because of Output of java is in bytecode so its not possible to modify by malicious programs thats why java is secure.
5. JVM is interpreter.
6. JVM takes bytecode as input and execute it.
7. Output of java is in bytecode so we need to setup JVM for other platform which makes Java platfrom Independent.
8. when JVM is installed on any system then we can execute any java program.

1) B) The key feature of Object Oriented Programming

Object: Object means a real world entity such as a pen chair, table, computer, watch etc.

Object: 1) Any entity that has state and behavior is known as an object.

ii) For example a chair, pen, table, keyboard, bike etc. It can be physical or logical.

example: A dog is an Objectthing because it has states color, name, breed, etc as well as behaviours like wagging the tail, barking eating etc.

Class: i) Collection of objects is called class. It is a logical entity.

ii) A class can also be defined as blueprint from which we can create an individual object. class doesn't consume any space.

Inheritance: i) when one object acquires all the properties and behaviors of a parent object, it is known as inheritance.

Polymorphism: i) If one task is performed by different ways, it is known as polymorphism.

ii) for example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle etc.

Abstraction: i) Hiding internal details and showing functionality is known as abstraction.

ii) In Java, we use abstract class and interface to achieve abstraction.

Encapsulation : 1) Binding (or wrapping) code and data together into a single unit are known as encapsulation.

1) c) ~~ab~~ Differences between = , == and .equal

=	==	.equals()
		1. equals is a method define inside the

1(c)

Difference between =, ==, and equals():

Here, '=' is an assignment opeator. And equals() is a method and '==' is an comparison operator.

'=' is used in Java for assign' content, assign a value of a variable to another variable, assign content of other variable as a value returned by a function.

Example:

```
a = fun();  // assign a content
int fun()
{   int x = 70;  // assign a variable
    int b;
    b = x;  // value copy to one
            variable to another

    return b;
}
```

In general, "==" and "equals()" used in Java to compare objects to check equality but there are some differences too:

"==" and equal()

1. Main differences between "==" and "equals()" is "==" is compare operator and 'equals()" is a method.

2. == operator used for reference comparison (if both objects point to the same memory location) and equals() used for the comparison of values in the objects.

Example:

```
public class Test {
public static void main(string [] args)
  { String s1 = new string ("Hello");
    String s2 = new String ("Hello");
System.out.println (s1==s2);
System.out.println (s1.equals(s2));
  }
}
```

Output: false
true

cannot be placed on left hand side

Example : 1 = x;
is invalid.

can be ... left hand side.

Example ; 1 = = 1 is valid and return 1

compare two string to check whether they are equal or not.

Q2(a) Define class. How do classes help us to organize our programs?

Class : A class can be defined as a template/ blueprint that describes the behaviour / state that the object of its type support.

A class, if designed properly, will provide a grouping of related data and common tasks. Rather than having arrays of primitive all over the place, classes allow us to keep various bits of associated information together.
A common example is a generic Person class:

Trying to keep a list of information about people together without classes will consist of several separate arrays of information:

```
String [ ] firstNames;
   "   [ ] lastNames;
int [ ] height;
int [ ] weight
```

Obviously, trying to keep these collections of data organized and synchronized will take a great deal of effort. Compare that using a Person class.

```
class Person {
  String firstName;
  String lastName;
  int height;
  int weight;
}
```

Then we only need to keep a single array up to date:

```
Person [ ] people;
```

2(b) When would private and Protected class members be used in an object oriented programming? Clearly distinguish between them.

Ans

Private: The private access modifier is specified using the keyword private.
The methods or data members declared as private are accessible only within class in which they are declared.
* Any other class of some package will not be able to access these members.
* To. level classes or interface cannot be declared as private because.
→ "Private" means only visible within the enclosing class
→ Protected means only visible within the enclosing class and any subclasses.

Protected: The protected access modifier specific using the keyword protected.
→ The methods or data members declared as protected are accessible within same package or sub classes in different package.

2(e) Write Java code to exchange two integer variables using a method named Swap. The main method will call the Swap method and the changes inside the Swap method must be visible to the main method, you also need to write the main method.

Ans:

```
import java.util.Scanner;
public class SwapTwoNumbers {
    public static void main (String arg[]){
        Sy Scanner se = new Scanner (System.in);
        System.out.println ("Enter first number :: ");
        int num1 = se.nextInt ();
        System.out.println ("Enter Second number :: ");
        int num2 = se.nextInt ();
        int temp = 0;
        temp = num1;
        num1 = num2;
        num2 = temp;
        System.out.println ("After swapping ::");
        System.out.println ("value of first number :: " + num1);
        System.out.println ("  "  "  "  2nd  "  :: " + num2);
    }
}
```

Output:

```
Enter the first number ::
22
Enter the second number ::
33
After swapping ::
value of first number :: 33
value of second number :: 22
```

3(a) write down the requirement of a recursive function. Explain different ways to overload a method.

Ans: In java, the function call mechanism supports the possibility of having a method call itself. This functionality is known as recursion.

There are two main requirements of recursive function:

A Stop condition: the function returns a value when a certain condition is satisfied, without a further recursive call.

The Recursive call: the function calls itself with an input which a step closer to the stop condition.

Example:

```
public static long factorial (int n) {
    if (n == 1) return 1;
    return n * factorial (n-1);
```

Method overloading can be achieved in following three ways —

* By changing the number of parameters in the method
* " " " order " parameter types
* By " " data types of the parameters.

Example:

```
public class Tester {
public static void main (String args [ ]) {
    Tester tester = new Tester ();
System.out.println (tester.add (1, 2));
```

```
System.out.println(tester.add(1,2,3));
System.out.println(tester.add(1.0f, 2,3));
"  "  "  "  (  "  .add(1, 2.0f, 3));
} public int add(int a, int b) {
      return a+b;
} public int add(int a, int b, int c) {
      return a+b+c;
} public float add(~~int~~ float a, ~~float~~ int b, int c) {
      return a+b+c;
} public float add(int a, float b, int c) {
   return a+b+c }
}
```

Output:
```
3
6
6.0
6.0
```

3(b) Describe the role of the final keyword and give a list of guidelines for when it should be used and when it should not be used.

Ans * When a variable is declared with final keyword, its value can't be modified, essentially, a constant.
* This means that we must initialize a final field when it is declared.

Initializing a final variable:

We must initialize a final variable, otherwise compiler will throw compile-time error. A final variable can only be initialized once, either via an initializer or assignment statement. There are two ways to initialize a final variable:

* First, we can initialize a final variable when it is declared. This approach is the most common.
* Second, we can assign it a value within a constructor.

When we use a final variable:

* The only difference between a normal variable and a final variable is that we can re-assign value to a normal variable but we cannot change the value of a final variable once assigned. Hence final variable must be used only for the values that we want to remain constant throughout the execution of program.

4(a) Define constructor. How do invoke constructor in Java?

Ans: Constructor: A constructor initializes an object when it is created. It has the same name as its class and is syntactically similar to a method. However, constructors have no explicit return type.

following the syntax of a constructor —

```
class className {
    className () {
    }
}
```

A constructor is similar to method and it is invoked at the time creating an object of the class, it is generally used to initialize the instance variables of a class. The constructors have some name as their class and have no return type. There is no need to 'invoke constructors explicitly these are automatically invoked at time of 'instantiation.

The this keyword in Java is a reference to the object of the current class. Using it, you can refer a field, method or, construction of a class.

Invoking a constructor from a method:

No you cannot call a constructor from a method. The only place from which you can invoke constructors using "this()" or "super()" is the first line of another constructor. If you try to invoke constructors explicitly else where, a compile time error will be generated.

4(b) Write two different ways to create string in java. Which one is better and why? (4-(b)

Ans: String is a sequence of characters. In java, objects of string are immutable which means a constant and can not be changed once created.

There are two ways to create string in java:

String literal:

* In java, String can be created like this: Assigning a string literal to a string instance:

```
String str1 = "Welcome";
String str2 = "welcome";
```

The problem with this approach:

* string object java.
* We have not created any string object using new keyword above.
* The compiler does that task for us it creates a string object having the string literal (that we have provided, in this case it is "welcome") and assigns it to the provided strings instances.
* But if the object already exists in the memory it ~~to the provided string instances~~ does not create a new object rather it assigns the same old object to the new instance
* that means even though we have two instances above (str1 and str2) only created on string object (having the value "Welcome") and assigned the same to both the instances.

using New keyword:

* As we saw above that when we tried to assign the same string object to two different literals, compiler only created one object and made both of the literals to point the same object.

To Overcome that approach are can create string like this
String Str1 = new String ("welcome");
String str2 = " " ("welcome");
In this case compiler would create two different object in memory having the same string.

4

(4)(e) In Java, method parameters are passed by value - explain what this means and give examples of the consequences.

while working under calling process, arguments is to be passed .. these should be in the same order as their respective parameters in the method specification. Parameters can be passed by value or by reference.

passing parameters by value means calling a method with pass a parameter. Through this, the argument value is passed to the parameter.

~~As a example here shows~~ the following programs shows an example of passing parameters by values.

```
public class swapping Example {
    public static void main (String [ ] args) {
        int a = 30;
        int b = 45;
        System.out.println ("Before swapping, a = "
                            + a" and b = " + b);

// Invoke the swap method
        swapFunction (a, b);
        System.out.println ("\n** Now, Before and After
            swaping values will be same here **;");
```

```
System.out.println("After swapping, a =" + a + "
                    and b is " + b);
}
public static void swapFunction(int a, int b) {
    System.out.println("Before swapping (Inside),
                a = " + a + "b = " + b);
// swap n1 with n2
    int c = a;
    a = b;
    b = c;
System.out.println("After swapping (Inside),
            case a = " + a + " b = " + b);
}
}
```

Output:

Before swapping, a = 30 and b = 45

Before swaping (Inside, a = 30 b = 45

After " (" , a = 45 b = 30

** Now, Before and After swapping values will be same there **:

After swapping, a = 30 and b = 45.

5(a) How multiple inheritance is implemented in Java? can abstract class be final - explain in brief.

Ans: Multiple inheritance in java programming is achieved or implemented using interfaces. Java does not support multiple inheritance using classes.
In simple terms, a class can inherit only one class and multiple interfaces in a java programs. In java terminology, we can say that
"A class can extend only one class but it can implement multiple interfaces."
For example, below inheritance using multi multiple classes is wrong as two classes can't be extended.
Class C is inheriting A and B.

```
class A { }
u   B { }
u   C extends A, B { }
```

Now program example for multiple inheritance in java language is correct as this example is extending only one class A and implementing multiple interfaces i.e. IB and IC

```
class A { }
u   IB { }
u   IC { }
class B extends A implements IB, IC { }
```

Abstract can extend final.
We can't make an abstract class or method final in java because the abstract and final are the mutual exclusive concept. An abstract class is incomplete and can only be instantiated by extending a concreate class and implementing all abstract methods within

(5)

a final class is considered as complete and can't be extended that's why Java compiler throws a compile time error. using both abstract and final modifier with a class in illegal in Java.

5(b) write down a function method "compute_Volume" to compute the volume of a 3D rectangular box with height h, width w and length l. then, write a new function by changing the previous function a little so that it can compute the volume of a rectangular box as well as that of a cube. remember that, a cube can only have only one parameter. The "compute_volume" function should be able to handle the following calls.

1. compute_Volume (30, 20, 10);
2. compute_u (10, 10, 10);
3. compute_volume (10);

Ans

```
public class compute_volume {
    double height;
    double length;
    double width;
    double total_vol;
    public compute_volume() {
    }
    public double compute_volume(double x, double y,
                                 double z)
    {
        height = x;
        length = y;
        width = z;
    }
```

```
Public double compute_volume (double x,
                              double y, double z) {
}       return total_vol = x * y * z;
public double compute_volume (double x) {
return total_vol = Math.pow (x, 3);

package compute_volume;

public class Main { }
public static void main (String [ ] args) {
    Compute_volume compute = new Compute_volume
                                        ();
System.Out.println ( compute. Compute_volume
                                    (30, 20, 10);
Sy  u   u   u ( compute. u . .u. u - u .
                                    (10, 10, 10);
Sy  . . u       u ( compute. e  . . . (10));
} }
```

Output: 6000.0
1000.0
1000.0

Question 6:

a) What are the benefits of packages? Explain Java API package?

⇒ A java package is a group of similar types of classes, interfaces and sub-packeges.

The benefits of java packages are -

1. Java package is used to categorize by the classes and interfaces.
2. It is easy to maintained.
3. Java packages is provide as access protection.
4. It may removes naming collision.
5. We can create our own package or extend already available package.

Java API pages are explained below:

Package	Contents
Java.lang	Language support classes. They include clas for primitive types, string, math functio thread and exceptions.
Java.util	Language utility classes such as vector hash tables, random numbers, data, etc

java.io	Input/output support classes. They provide facilities for the input and output of data.
java.applet	classes for creating and implementing applets.
Java.net	Classes for networking. They include classes for communication with local computers as well as with internet servers.
Java.awt	Set of classes for implementing graphical use interface. They include classes for windows, buttons, lists, menus and so on.

(b) What do you mean by abstract class? What are the restrictions to classes that extend abstract class? Explain in short with code examples.

⇒ **Abstract Class:** An abstract class is a class that is declared abstract - it may or may not include abstract ~~class~~ methods

Abstract classes cannot be instantiated, but they can be subclassed.

An abstract method is a method that is declared without implementation (without braces, and followed by a semicolons) like this:

```
abstract void moveTo(double deltaX, double deltaY);
```

If a class includes abstract methods, then the class itself must be declared abstract, as in:

```
public abstract class GraphicObject {
    // declare fields
    // declare nonabstract methods
    abstract void draw();
}
```

When an abstract class is subclassed, the su
class usually provides implementations for
of the abstract methods in its parents cla
However, it does not, then the el subclas
must also be declared abstract.

Ques-7:

a) What is exception? Explain the syntax of try block and catch block with an example.

⇒ Exception: Exception is a run-time error which arises during the execution of java program. The term exception in java stands for an "exceptional event".

The purpose of exception handling is to detect and report an exception so that proper action can be taken and prevent the program which is automatically terminate or stop the execution because of that exception.

Java exception handling is managed by using five keywords: try, catch, throw, throws and finally.

Try: Piece of code of a program that anyone want to monitor for exceptions are contained within a try block. If an exception occurs with the try block it is thrown.

```
try { // block of code to monitor for errors
}
```

Catch: Catch block can catch this exception and handle it in some logical number.

```
catch (ExceptionType1 e1) {
    // exception handlers for ExceptionType1
}
catch (ExceptionType2 e2) {
    // exception handler for ExceptionType2
}
```

7(b): Write short note on polymorphism. Write a java program that demonstrates the use of polymorphism.

⇒ Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism is OOP occurs when a parent class reference is used to refer to a child class object.

The word "poly" means many and "morphs" means forms. So, polymorphism means many forms.

Two types of polymorphism in java:

1. Compile time polymorphism and 2. Runtime polymorphism.

Polymorphism is one of the OOPs feature that allows us to perform a single action in different ways.

For example, lets say we have a class `Animal` that has a method `sound()`. Since this is a generic class so we can't give it a implementation like: Roar, Meow, Oink etc. We had to give a generic message.

```
public class Animal {
  ...
  public void sound() {
    System.out.println("Animal is making a
                        sound");
  }
}
```

Now lets say we two subclasses of Animal class: `Horse` and `Cat` that extands (see interface) `Animal` class. We can provide the implementation to the same method like this:

```
public class Horse extends Animal {
...
@Override
public void sound() {
  System.out.println("Neigh");
}
```

and

```
public class Cat extends Animal{
...
@Override
public void sound(){
System.out.println("Meow");
}
}
```

As we can see that although we had the common action for all subclasses `sound()` but there were different ways to do the same action. This is a perfect example of polymorphism.

Ans to the Q No 8

(a)

Q. Properties of static va. and methods? why the main () method is declared as static?

sol^n:-

static v. can be used to refer the common properties of all objects. Creats memory only once in class area at the time of class loading. We use the static v. for the property that is common to all obj. for ex - , in class student, all student shares the same college name. We use static method to ~~chang~~ chang static v.

Methods declared as ~~static~~ can directly only call static method and access static data.

main () is declared as static as it is directly call by the JVM without creating an object of the class in which it is declared. when Java runtime starts there

is no object of class present. thats why main method has to be static, so JVM can load the class into memory and call the main method.

(b)

Q. Defn interfaces? How to implement interface.

Soln:- An interface in the JAVA programming language is an abstract type that is used to specify a behaviour that classes must implement. They are similar to protocols. It is declared using the interface keyword and may only contain method signature & constant declarations.

A class uses implement keyword to implement an interface. The " " appears in the class declaration following the extends portion of the declaration

Ans to the Q No 8

(C)

Output:- 30 50

10 20

9(a) Difference between constructor and method:

Constructor	Method
1. Constructor name should be same as class.	1. Method name may or may not be same as class.
2. Constructor never returns any value.	2. Method have return type. so it returns value.
3. 2 type of constructor available in Java.	3. Java supports 6 types of method.
4. Constructor only can be called by new keyword in Java	4. Method can be called by its class name, object or directly.
5. Constructor can't be overridden in Java.	5. Method can be overridden
6. Constructor cannot be decleared as static.	6. Method can be decleared as static.
7. Java provides default constructor.	7. Java never provides any method by default.

Constructor	Method.
8. The purpose of a constructor is to create an instance of a class.	8. The purpose of a method is to execute Java code.
9. They are used to initialize objects that doesn't exist.	9. They perform operations on already created objects.
10. They are not inherited by subclass.	10. They are inherited by subclass.

9(b)

Source Code:

```
public class Shape {
    String shape;
    public string draw() {
        return "This is draw method.";
    }
    public string erase() {
        return "This is erase method.";
    }
}
public class Circle extends Shape {
}
public class Square extends Shape {
}
public class Triangle extends Shape {
}
```

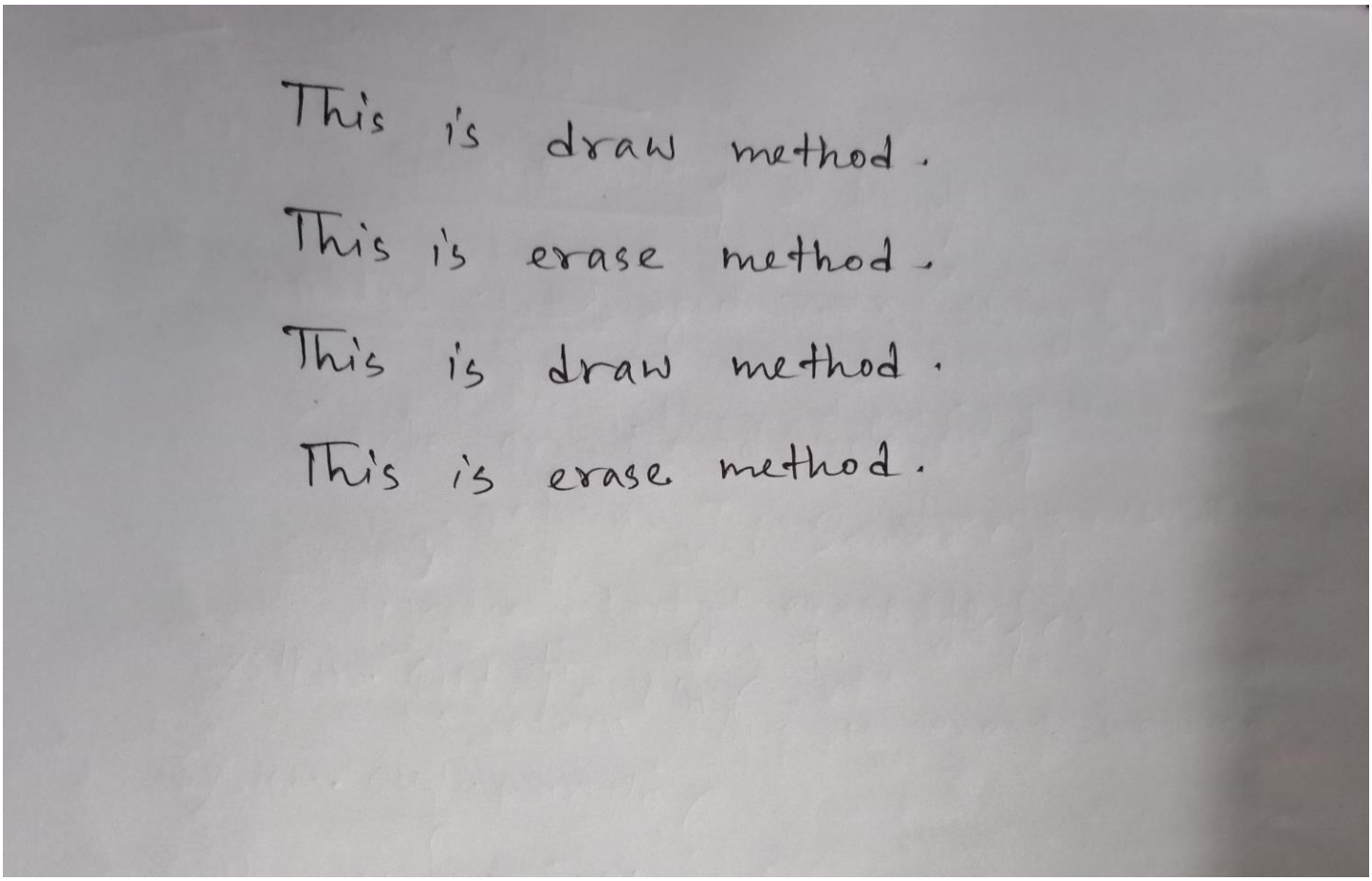
```
public class Main {

    public static void main (String [] args) {

        Circle cr = new Circle ();
        Square sq = new Square ();
        Triangle tr = new Triangle ();
    System. Out. println (cr. draw());
    System. out. println (cr. erase ());
    System. out. println( sq. draw ());
    System. out. println ( sq. erase ());
    System. out. println (tr. draw ());
    System. out. println (tr. erase ());

    }
}
```

Output: This is draw method.
This is erase method.



10(a)

```
interface i1{
   public void f1();
   public void f2();
}
interface i2{
   public void f3();
   public void f4();
}
interface i3{
   public void f5();
   public void f6();
```

```
}
interface i4{
    public void f7();
}
class MyClass implements i1,i2,i3,i4{
    public void f1(){
        System.out.println("It is f1");
    }
    public void f2(){
        System.out.println("It is f2");
    }
    public void f3(){
        System.out.println("It is f3");
    }
    public void f4(){
        System.out.println("It is f4");
    }
    public void f5(){
        System.out.println("It is f5");
    }
    public void f6(){
        System.out.println("It is f6");
    }
     public void f7(){
        System.out.println("It is f7");
```

```
    }
}
public class HelloWorld{
    public static void main(String []args){
        MyClass ob = new MyClass();
            ob.f1();
            ob.f2();
            ob.f3();
            ob.f4();
            ob.f5();
            ob.f6();
            ob.f7();
        }
}
```

10(b) What is the difference between declaring a variable in a class with access modifier rather than no modifier?

Solve: It isn't same.

Both protected and no access modifier variables are accessible in the same package, but protected variables can be accessed by a subclass instance anywhere. If a class has no modifier, it is visible only within its own package. The protected modifier specifies that the member can only accessed within its own package and, in addition, by a subclass of its class in another package.

Modifier	Class	Package	Subclass	World
Public	Y	Y	Y	Y
Protected	Y	Y	Y	N
no modifier	Y	Y	N	N
Private	Y	N	N	N