Code -

G. S. Mandal's

Maharashtra Institute of Technology, Aurangabad

(An Autonomous Institute)

END SEMESTER EXAMINATION

Second Year B.Tech (ECE) – Feb/Mar-2023

Course Code : ECE204 **Course Name : Programming in JAVA**
Duration : 2 Hrs **Max. Marks : 50** **Date :**
Instructions :

i) **All questions are compulsory**
ii) **Assume suitable data wherever necessary and clearly state it**
iii) **Figures to right indicate full marks**

Q. 1	**Answer any five (Marks:10)**	
		Marks
a)	What is the need of object oriented programming? Solution: Object-oriented programming (OOP) is a way of thinking about and organizing code for maximum reusability. With this type of programming, a program comprises objects that can interact with the user, other objects, or other programs. This makes programs more efficient and easier to understand.	2
b)	What is an Operator? Explain with an example. Solution : In mathematics and computer programming, an operator is a character that represents a specific mathematical or logical action or process. For instance, "x" is an arithmetic operator that indicates multiplication, while "&&" is a logical operator representing the logical AND function in programming.	2
c)	Define Classes and object? Solution : **Classes:** A class defines the properties and behaviors for objects .A class : a template, blue print, or contract that defines what an object's data fields and methods will be. **Objects**: An entity that has state and behavior is known as an object **Object Definitions:** An object is *a real-world entity.* An object is *a runtime entity.* The object is *an entity which has state and behavior.* The object is *an instance of a class.*	2

d)	Define Inheritance with its Syntax? State its types?
	Solution :
	Defination : Inheritance is one of the key features of OOP that **allows us to create a new class from an existing class**.
	Syntax :
	public class **name** extends **superclass** {
	public class Lawyer **extends Employee** {
		2

	... } Types : 1. Single 2. Multilevel 3. Hierarchical 4. Multiple 5.Hybrid	
e)	Why to use Interface in JAVA? Solution : There are mainly three reasons to use interface. They are given below. • It is used to achieve abstraction. • By interface, we can support the functionality of multiple inheritance. • It can be used to achieve loose coupling.	2
f)	What is Exception and Exception handling in JAVA? Solution : Exception : In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime. Exception Handling : Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, Etc.	2
g)	Write Life cycle of a Thread (Thread States)? Solution : In Java, a thread always exists in any one of the following states. These states are: 1. New 2. Active 3. Blocked / Waiting 4. Timed Waiting 5. Terminated	2
Q.2	a) Differentiate Constructor and Methods.	4

Solution :

4

Java Constructor	Java Method
Constructor is used to initialize the state of an object.	Method is used to expose behaviour of an object.
Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.
The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in any case.
Constructor name must be same as the class name.	Method name may or may not be same as class name.

b) Define Access Modifiers. Discuss Access Modifiers in JAVA.

Solution :

Definition : **Access modifiers** are object-oriented programming that is used to set the accessibility of classes, constructors, methods, and other members of Java.

Four Types of Access Modifiers

- **Private**: We can access the **private modifier** only within the same class and not from outside the class.
- **Default:** We can access the **default modifier** only within the same package and not from outside the package. And also, if we do not specify any access modifier it will automatically consider it as default.
- Protected: We can access the protected modifier within the same package and also from outside the package with the help of the child class. If we do not make the child class, we cannot access it from outside the package. So inheritance is a must for accessing it from outside the package.
- Public: We can access the public modifier from anywhere. We can access public modifiers from within the class as well as from outside the class and also within the package and outside the package.

Accessibility of Access Modifiers in Java

Access Modifier	Accessible by classes in the same package	Accessible by classes in other packages	Accessible by subclasses in the same package	Accessible by subclasses in other packages
Public	Yes	Yes	Yes	Yes
Protected	Yes	No	Yes	Yes
Package (default)	Yes	No	Yes	No
Private	No	No	No	No

Q.3 Explain **IS-A** relationship in inheritance with example. Write a program to implement single level inheritance assuming suitable data. 8

Solution:

IS-A Relationship:

IS-A Relationship is wholly related to Inheritance. For example – a kiwi is a fruit; a bulb is a device.

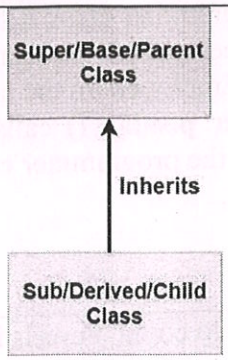
- IS-A relationship can simply be achieved by using extends Keyword.
- IS-A relationship is additionally used for code reusability in Java and to avoid code redundancy.
- IS-A relationship is unidirectional, which means we can say that a bulb is a device, but vice versa; a device is a bulb is not possible since all the devices are not bulbs.
- IS-A relationship is tightly coupled, which means changing one entity will affect another entity.

Advantage of IS-A relationship

- Code Reusability.
- Reduce redundancy.

Examples:
Car is a Vehicle
Orange is a Fruit
Surgeon is a Doctor
Dog is an Animal
means
Car can inherit from Vehicle, Orange can inherit from Fruit,
Just like
Child can inherit from parent

Single inheritance is the simplest type of inheritance in java. In this, a class inherits the properties from a single class. The class which inherits is called the derived class or child class or subclass, while the class from which the derived class inherits is called the base class or superclass or parent class. So, in single inheritance, we have only one derived class and one base class.



Syntax of Single Inheritance in Java

```
class base class
{
    .... methods
}
class derivedClass name extends baseClass
{
   methods ... along with this additional feature
}
```

Java uses the keyword "**extends**" to create a new class(derived class) from the existing class(base class). The term "**extends**" means to increase the functionality as the derived class can reuse the methods and fields of the base class, and along with this, new methods and fields can also be defined in the derived class, hence increasing the functionality.

Program:

```
class Employee {
   void salary() {
      System.out.println("Salary= 200000");
   }
}

class Programmer extends Employee {
// Programmer class inherits from Employee class
   void bonus() {
      System.out.println("Bonus=50000");
   }
}

class single_inheritance {
   public static void main(String args[]) {
      Programmer p = new Programmer();
      p.salary(); // calls method of super class
      p.bonus(); // calls method of sub class
   }
}
```

Output:

```
Salary= 200000
Bonus=50000
```

Explanation

In the above example, we have two classes- The employee class and the Programmer class.

	We see that the Programmer class extends the Employee class, which is an example of single inheritance. The relationship "**is a**" is established here, i.e., a Programmer **is an** Employee. In the main method, we create an object '**p**' of the class Programmer. **p.salary()** calls the method of the Employee class, which is inherited by the programmer class, and p.bonus() calls the method of the programmer class.	

Q.4	Write a Java Program to count Vowels in a String
	
	Solution:
	import java.util.Scanner;
	class VowelsInaString
	{
	public static void main(String[ ] arg)
	{
	String s;
	char ch;
	int i=0;
	Scanner sc=new Scanner(System.in);
	System.out.print("Enter a string : ");
	s=sc.nextLine();
	System.out.println("Vowels in a string are");
	for(int j=0;j<s.length();j++)
	{
	ch=s.charAt(j);
	switch(ch)
	{
	case 'a' :
	case 'e' :
	case 'i' :
	case 'o' :
	case 'u' :
	case 'A' :
	case 'E' :
	case 'I' :
	case 'O' :
	case 'U' :i=1;
	System.out.println(ch);
	}
	}
	if(i==0)
	System.out.println("There are no vowels in a string");
	
	}
	}
		8

Q.5	Explain the following terms with respect to exception handling with program: i) try ii) catch iii) throw iv) finally Solution : An exception is an "unwanted or unexpected event", which occurs during the execution of the program i.e, at run-time, that disrupts the normal flow of the	8

program's instructions. When an exception occurs, the execution of the program gets terminated. An exception can occur due to several reasons like a Network connection problem, Bad input provided by a user, Opening a non-existing file in your program, etc

Blocks & Keywords used for exception handling

1. try: The try block contains a set of statements where an exception can occur.

```
try
{
   // statement(s) that might cause exception
}
```

2. catch: The catch block is used to handle the uncertain condition of a try block. A try block is always followed by a catch block, which handles the exception that occurs in the associated try block.

```
catch
{
  // statement(s) that handle an exception
  // examples, closing a connection, closing
  // file, exiting the process after writing
  // details to a log file.
}
```

3. throw: The throw keyword is used to transfer control from the try block to the catch block.

4. throws: The throws keyword is used for exception handling without try & catch block. It specifies the exceptions that a method can throw to the caller and does not handle itself.

5. finally: It is executed after the catch block. We use it to put some common code (to be executed irrespective of whether an exception has occurred or not) when there are multiple catch blocks.

Example of an exception generated by the system is given below :

```
Exception in thread "main"
java.lang.ArithmeticException: divide
by zero at ExceptionDemo.main(ExceptionDemo.java:5)
ExceptionDemo: The class name
main:The method name
ExceptionDemo.java:The file name
java:5:line number
// Java program to demonstrate working of try,
// catch and finally

class Division {
        public static void main(String[] args)
        {
                int a = 10, b = 5, c = 5, result;
                try {
                        result = a / (b - c);
                        System.out.println("result" + result);
```

```
                }

                catch (ArithmeticException e) {
                        System.out.println("Exception caught:Division by
zero");
                }

                finally {
                        System.out.println("I am in final block");
                }
        }
}
```

Output:
Exception caught:Division by zero
I am in final block

An example of throws keyword:

```
// Java program to demonstrate working of throws
class ThrowsExecp {

        // This method throws an exception
        // to be handled
        // by caller or caller
        // of caller and so on.
        static void fun() throws IllegalAccessException
        {
                System.out.println("Inside fun(). ");
                throw new IllegalAccessException("demo");
        }

        // This is a caller function
        public static void main(String args[])
        {
                try {
                        fun();
                }
                catch (IllegalAccessException e) {
                        System.out.println("caught in main.");
                }
        }
}
```

Output:
Inside fun().
caught in main.

Q.5. **OR** 8

Define Multithreading.

Write a program that creates three tasks and three threads to run them.

- The first task prints the letter "a" 100 times.
- The second task prints the letter "b" 100 times.
- The third task prints the integers 1 through 100

Solution :

Multithreading enables multiple tasks in a program to be executed concurrently.

One of the powerful features of Java is its built-in support for multithreading—the concurrent running of multiple tasks within a program. In many programming languages, you have to invoke system-dependent procedures and functions to implement multithreading.

Program :

```
public class TaskThreadDemo {
 public static void main(String[] args) {
 // Create tasks
  Runnable printA = new PrintChar('a', 100);
  Runnable printB = new PrintChar('b', 100);
  Runnable print100 = new PrintNum(100);
  // Create threads
  Thread thread1 = new Thread(printA);
 Thread thread2 = new Thread(printB);
 Thread thread3 = new Thread(print100);
 // Start threads
 thread1.start();
 thread2.start();
 thread3.start();
 }
 }
 // The task for printing a character a specified number of times
 class PrintChar implements Runnable {
 private char charToPrint; // The character to print
 private int times; // The number of times to repeat
 /** Construct a task with a specified character and number of
 * times to print the character
 */
 public PrintChar(char c, int t) {
 charToPrint = c;
 times = t;
 }
 @Override /** Override the run() method to tell the system
 * what task to perform
 */
 public void run() {
 for (int i = 0; i < times; i++) {
 System.out.print(charToPrint);
 }
 }
 }
 // The task class for printing numbers from 1 to n for a given n
 class PrintNum implements Runnable {
 private int lastNum;
 /** Construct a task for printing 1, 2, ..., n */
```

```
public PrintNum(int n) {
lastNum = n;
}
@Override /** Tell the thread how to run */
public void run() {
for (int i = 1; i <= lastNum; i++) {
System.out.print(" " + i);
}
}
}
```

Q.6 Explain interface ? Elaborate implementation of Interface using one example ? 8

Solution :

An interface is a fully abstract class. It includes a group of abstract methods (methods without a body).

We use the interface keyword to create an interface in Java. For example,

```
interface Language {
 public void getType();

 public void getVersion();
}
```

Here,

- *Language* is an interface.
- It includes abstract methods: getType() and getVersion().

Implementing an Interface

Like abstract classes, we cannot create objects of interfaces.

To use an interface, other classes must implement it. We use the implements keyword to implement an interface.

Example: Java Interface

```
interface Polygon {
 void getArea(int length, int breadth);
}

// implement the Polygon interface
class Rectangle implements Polygon {

 // implementation of abstract method
 public void getArea(int length, int breadth) {
  System.out.println("The area of the rectangle is " + (length * breadth));
 }
}

class Main {
 public static void main(String[] args) {
  Rectangle r1 = new Rectangle();
  r1.getArea(5, 6);
 }
```

}

Implementing Multiple Interfaces

In Java, a class can also implement multiple interfaces. For example,

```
interface A {
  // members of A
}

interface B {
  // members of B
}

class C implements A, B {
  // abstract members of A
  // abstract members of B
}
```

b) How to access packages from another packages? Explain it with any one method in detail.

Solution :

There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname;
3. fully qualified name.

Example:

If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

Example of package that import the packagename.*

//save by A.java

```
package pack;
public class A{
  public void msg()
{
System.out.println("Hello Vinodpuri");
}
}
```

//save by B.java

```
package mypack;
import pack.*;
class B{
  public static void main(String args[]){
  A obj = new A();
  obj.msg();
  }
}
```

Output: Hello Vinodpuri

OR

Q6.

8

Write a program to reverse a number in Java using a while loop and using a Static method

Solution : **Using a While Loop**

```
import java.util.Scanner;
class Rev
{
public static void main(String[] arg)
{
int a,res=0,n;
Scanner sc=new Scanner(System.in);
System.out.println("Enter a number");
n=sc.nextInt();
while(n!=0)
{
a=n%10;
res=(res*10)+a;
n=n/10;
}
System.out.println("reverse of a number is "+res);
}
}
```

Using a Static Method

```
import java.util.Scanner;
class Rev
{
        public static void main(String[] arg)
        {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter a number");
        int x=sc.nextInt();
        int r=reverse(x);
        System.out.println("Reverse of a number is = "+r);
        }
static int reverse(int num)
{
        int rem,res=0;
        while(num!=0)
        {
        rem=num%10;
        res=(res*10)+rem;
        num=num/10;
        }
        return res;
}
```