FAT - 2

Java Programming Sem - 1

Q.1 Explain the polymorphism in Java (function overloading and function overriding) with suitable example.

The word polymorphism means having many forms. In simple words, we can define java polymorphism as the ability of a message to be displayed in more than one form.

Polymorphism allow us to perform single action in different ways.

In other words, polymorphism allow us to define, one interface and have multiple implementations.

The word "poly" means many and "morphs" means forms, so it means many forms.

In Java, Polymorphism is mainly divided into two types -

- Compile time polymorphism.
- Run time polymorphism.

→ Compile-time polymorphism —

It is also known as static polymorphism. This type of polymorphism is achieved by function overloading or operator overloading.

→ function overloading —

When there are multiple functions with the same name but different parameters then these functions are said to be overloaded.

functions can be overloaded by changes in number of arguments or by changing in the type of arguments.

Example —

```
class Helper {

    static int multiply(int a, int b)
    {
        return a * b;
    }
```

```
static double multiply (double a, double b)
{
    return a*b;
}

class Helper A
{
    public static void main (string[]
                                args)
    {
        System.out.println (Helper.multiply
                                (2,4));

        System.out.println (Helper.multiply
                                (5.5, 6.3));

    }
}
```

Output –

```
8
34.65
```

→ Runtime polymorphism –

It is also known as Dynamic method Dispatch.

It is a process in which a function call to the overriden method is resolved at runtime.

This type of polymorphism is achieved by function overriding.

→ function overriding –

function overriding occurs when a derived class has a definition for one of the member functions of the base class.

The base class is said to be overridden.

Example –

```
class Parent
{
    void Print()
    {
        System.out.println("parent class");
    }
}
```

Rainbow

```
class subclass1 extends Parent
{
    void Print()
    {
        System.out.println("subclass 1");
    }
}

class subclass2 extends Parent
{
    void Print()
    {
        System.out.println("subclass 2");
    }
}

class A
{
    public static void main(String[]
                                args)
    {
        Parent a;

        a = new subclass1();
        a.Print();

        a = new subclass2();
        a.Print();
    }
}
```

Rainbow

Output -

```
Subclass 1
Subclass 2
```

Q.2 Explain the Exception Handling mechanism in Java with example (try, catch, throw, throws, finally).

Exception Handling in Java is one of the effective means to handle runtime errors so that regular flow of the application can be preserved.

Java Exception Handling is a mechanism to handle runtime errors so that the regular flow of the application preserved such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

In Java, Exception is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program's instructions.

Exceptions can be caught and handled by the program.

When an exception occurs within a method, it creates an object. This object is called the exception object.

It contains information about the exception, such as the name and description of the exception and the state of the program when the exception occurred.

→ Major reasons why an exception occurs –

- Invalid user input.
- Device failure.
- Loss of network connection.
- Physical limitations.
- code errors.
- Opening an unavailable file.

→ Types of Exceptions –

Exceptions can be categorized in two ways –

- Built-in Exceptions.
- User-Defined Exceptions.

→ Built-in Exceptions —

Built-in exceptions are the exceptions that are available in Java libraries. These exceptions are suitable to explain certain error situations.

It is categorized in two parts —

— checked exceptions -

Checked exceptions are called compile time exceptions because these exceptions are checked at compile time by the compiler.

— unchecked exceptions —

The unchecked exceptions are just opposite to the checked exceptions. The compiler will not check these exceptions at compile time.

In simple words, if a program throws an unchecked exception and even if we didn't handle or declare it,

the program would not give a compilation error.

Example –

```
class Exception
{
   public static void main(String args[])
    {
       try{ //code that may raise exception
          }
            // rest of the program
    }
}
```

→ To handle exceptions, java provides several methods, such as –

- try
- catch
- finally
- throw
- throws

→ Try block –

The try block is a domain that has a list of statements in which exceptions may occur.

Try block cannot be used alone, therefore, it is always accompained by either catch block, finally block or both.

Syntax –

```
try {
    // code that may throw an
       exception
  } catch(exception-class-Name ref)
      {
        //rest of the code
      }
```

→ Catch block –

The catch block is a method that is utilized to grasp exceptional cases.

It always accompany a ~~catch~~ try block.

finally block can accompany a catch

block after it accompanies a try block.

A number of catch block can be linked with a try block.

It can handle many exception cases in all linked blocks.

The particular catch block assigned the code with exception handles the exception.

Syntax –

```
try{ //Statements that give an
       exception
   }
   catch(exception(type)e(object))
   { //error handling code
   }
```

Example –

```
import java.util.*;
public class main
{
  public static void main(String args[])
  {
```

```
ArrayList<String> al = new ArrayList
                        <String> ();
al.add("coding");
al.add("Java");

try{ String wrongAccess = al.get(5);
  } catch(Arithmetic Expression ae)
    {
      System.out.println("wrong
arithmetic expression. Please try again");
  } catch(IndexOutOfBoundsExcep
      tion indxExcep)
    {
      System.out.println("You
tried to access wrong index.
Please check and try again");
  } catch(Exception e)
    { System.out.println("This is
      will handle any exception");
    }
  }
}
```

output -

you tried to access wrong index. Please check and try again!

→ Throw block -

In Exception Handling, the throw keyword explicitly throws an exception from a method or constructor.

We can throw either checked or unchecked exceptions in java by throw keyword.

The "throw" keyword is mainly used to throw a custom exception.

The only object of the throwable class or its subclasses can be thrown.

When a throw statement is encountered, program execution is halted & the nearest catch statement is searched for a matching kind of exception.

Example -

```
public class Throwexample
{
    static void checkEligibility(int stuage,
                                 int stuweight)
    {
        if(stuage<15 && stuweight<45)
        {
            throw new ArithmeticException
            ("Student is not eligible for
              registration");
        }
        else
        {
            System.out.println("Student Entry
              is valid!!");
        }
    }
    public static void main(String args[])
    {
        System.out.println("Student Entry
          is valid")

        System.out.println("welcome to
          the Registration process!!");
```

```
        checkEligibility(10,39);
        System.out.println("Have a nice day");
    }
}
```

Output -

```
Welcome to the Registration process !!
Exception in thread "main" java.lang.
Arithmetic Exception : Student is
  not eligible for registration.
at ThrowExample.checkEligibility(ThrowExam
  ple.java:8)
at ThrowExample.main(ThrowExample.java:
                                  16)
```

→ throws block -

Any method that might cause exceptions must identify all of the exceptions that can occur during its execution.

The programmer calling the method is aware of which exceptions

must be handled. The throws keyword can be used to do this.

Example –

```
import java.io.*;

class ThrowExample
{
  void mymethod(int num) throws IOException, ClassNotFoundException
  {
    if(num==1)
      throw new classNotfoundException
      ("classNot found Exception");
  }
}

public class main
{
  public static void main(String args[])
  {
    try
    {
      ThrowExample Obj=new ThrowExample();
      Obj.mymethod(1);
    }
```

```
        catch (exception ex)
        {
            System.out.println(ex);
        }
    }
}
```

output –

javac /tmp/DowkQM3icb/ThrowExample.java

java.io.IOException: IOException Occurred

→ finally block –

The finally block in java is used to put impt. codes such as clean up code e.g., closing the file or closing the connection.

The finally block executes whether exception handled or not.

A finally contains all the crucial statements regardless of the exception occurs or not.

Example -

```
import java.io.*;

class A
{
  public static void main(String[] args)
  {
    try
    {
      System.out.println("inside try block");
      System.out.println(34/2);
    }
    catch(ArithmeticException e)
    {
      System.out.println("Arithmetic
          Exception");
    }
    finally
    {
      System.out.println("finally :
          i execute always.");
    }
  }
}
```

Output –

inside try block
17
finally : i execute always.