NAGARJUNA COLLEGE OF ENGINEERING AND TECHNOLOGY, BENGALURU

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

Laboratory Programs

Manual

Object Oriented Programming with JAVA

Course Code: 19CSI43

1. Write a Java program that prompts the user for an integer and then prints out all prime numbersup to that integer.
2. Write a Java program that prints the following pattern type

```
*****
*****
****
***
**
*
```

3. Write a java program to calculate gross salary & net salary taking the following data.Input:empno,empname,basic Process: DA=50%of basic HRA=25%of basic CCA=Rs240/-

PF=10%of basic PT=Rs100/-

4. Write a Java program that displays area of different Figures(Rectangle,Square,Triangle) usingthe method overloading.
5. Write a Java program that displays the time in different formats in the form of HH,MM,SS usingconstructor Overloading.
6. Write a Java program that counts the number of objects created by using static variable.
7. Write a java program that implements educational hierarchy using inheritance.
8. Write a java program that implements Array Index out of bound Exception using built-in-Exception.
9. Write a java program that implements bank transactions using user defined exception.
10. Write a java program to identify the significance of finally block in handling exception.

1. Write a Java program that prompts the user for an integer and then prints out all prime numbers up to that integer.

```
//Program to print prime numbers upto given integer
import java.util.*;
class Prime
{
	public static void main(String args[])
	{
		int n,f;
		Scanner scr=new Scanner(System.in);
		System.out.println("\nEnter n value:  ");
		n=scr.nextInt();
		System.out.println("\nPrimenumbers are : ");
		for(int i=2;i<=n;i++)
		{
			f=0;
			for(int j=2;j<=i/2;j++)
			if((i%j)==0)
			{
				f=1;
				break;
			}
		if(f==0)
		System.out.print(i+"   ");
		}
	}
}
```

Output:

```
Enter n value:
10

Prime numbers are :

2   3   5   7
```

Write a Java program that prints the following pattern type

Programs for printing pyramid patterns in Java

a. Simple pyramid pattern

```
import java.io.*;

 // Java code to demonstrate star patterns

public class GeeksForGeeks

{

    // Function to demonstrate printing pattern

    public static void printStars(int n)

    {

        int i, j;

         // outer loop to handle number of rows, n in this case

                for(i=0; i<n; i++)

        {
            //  inner loop to handle number of columns
            //  values changing acc. to outer loop
            for(j=0; j<=i; j++)
            {
                // printing stars
                System.out.print("* ");
            }
            // ending line after each row
            System.out.println();
        }
    }
     // Driver Function
    public static void main(String args[])
    {
        int n = 5;

        printStars(n);

    }

}
```

Output

```
*

* *

* * *

* * * *

* * * * *
```

b. Printing Triangle pattern

```
    *

   * *

  * * *

 * * * *

* * * * *
```

```
import java.io.*;

// Java code to demonstrate star pattern
public class GeeksForGeeks
{
    // Function to demonstrate printing pattern
    public static void printTriagle(int n)
    {
        // outer loop to handle number of rows
        //  n in this case
        for (int i=0; i<n; i++)
        {
             // inner loop to handle number spaces
            // values changing acc. to requirement
            for (int j=n-i; j>1; j--)
            {
                // printing spaces
                System.out.print(" ");
            }

            //  inner loop to handle number of columns
            //  values changing acc. to outer loop
            for (int j=0; j<=i; j++ )
            {
                // printing stars
                System.out.print("* ");
            }
              // ending line after each row
            System.out.println();
        }
    }
        // Driver Function
    public static void main(String args[])
    {
        int n = 5;
        printTriagle(n);
    }
}
```

3. Write a java program to calculate the net salary of an employee by considering the parameters called HRA(House rent allowance),DA(Dearness allowance),GS (Gross salary)and income tax. Let us assume some parameters.

HRA=10% of basic salary

DA=73% of basic salary

GS=basic salary+DA+HRA

Income tax=30% of gross salary

net salary= GS-income tax**Algorithm:**

```
1)Start
2)Take the input from the user as employee name,id and basic salary
3)Calculate the the above parameters DA,HRA,GS,income tax and net salary
4)Display the output
5)Stop
```

Code of Program

```
import java.util.*;
class employee
{
    private String employeid;
    private String empname;
    private int basicsalary,HRA,DA,GS,incometax,netsalary;
    public void read()
    {
       Scanner scan= new Scanner(System.in);
       System.out.println("Enter the employee id");//taking all the inputs from the user
       employeid=scan.next();
       System.out.println("Enter the employee name");
       empname=scan.next();
       System.out.println("Enter the basic salary of an employee");
```

```
        basicsalary=scan.nextInt();
        calculate();
      }
      public void calculate()  //calculating all the parameters
      {
        HRA=(10/100)*basicsalary;
        DA=(73/100)*basicsalary;
        GS=basicsalary+DA+HRA;
        incometax=(30/100)*GS;
        netsalary=GS-incometax;
      }
      public void display()  //displaying the calculating parameters
      {
        System.out.println("Employeeid  :  "+employeid+"n"+"Employename  :
     "+empname+"n"+"Employee basic salary :  "+basicsalary+"n"+"HRA  :
     "+HRA+"n"+"DA  :  "+DA+"n"+"GS  :  "+GS+"n"+"Incometax  :
     "+incometax+"n"+"netsalary  :  "+netsalary);
      }
 }
class main //main function
{
      public static void main(String args[])
      {
         employee employeobj=new employee();
         employeobj.read(); //calling read function
         employeobj.display(); //calling display function
      }
    }
```

4. Java Program to find area of Geometric figures using method Overloading

This program finds the area of square, rectangle and circle using method overloading. In this program we have three methods with same name area(), which means we are overloading area() method. By having three different implementation of area method, we are calculating the area of square, rectangle and circle.

Example: Program to find area of Square, Rectangle and Circle using Method Overloading.

```
class JavaExample
{
   void calculateArea(float x)
   {
      System.out.println("Area of the square: "+x*x+" sq units");
   }
   void calculateArea(float x, float y)
   {
      System.out.println("Area of the rectangle: "+x*y+" sq units");
   }
   void calculateArea(double r)
   {
      double area = 3.14*r*r;
      System.out.println("Area of the circle: "+area+" sq units");
   }
   public static void main(String args[]){
      JavaExample obj = new JavaExample();

      /* This statement will call the first area() method
```

```
         * because we are passing only one argument with
         * the "f" suffix. f is used to denote the float numbers
         *
         */
            obj.calculateArea(6.1f);

            /* This will call the second method because we are passing
             * two arguments and only second method has two arguments
             */
            obj.calculateArea(10,22);

            /* This will call the second method because we have not suffixed
             * the value with "f" when we do not suffix a float value with f
             * then it is considered as type double.
             */
            obj.calculateArea(6.1);
    }
}
```

Output:

```
Area of the square: 37.21 sq units
Area of the rectangle: 220.0 sq units
Area of the circle: 116.8394 sq units
```

5.Write a Java program that displays the time in different formats in the form of HH,MM,SS using constructor Overloading

Problem Description

How to display date in different formats?

Solution

This example displays the names of the weekdays in short form with the help of DateFormatSymbols().get Weekdays() method of DateFormatSymbols class.

```
import java.text.*;
import java.util.*;
public class Main {
  public static void main(String[] args) {
    Date dt = new Date(1000000000000L);
    DateFormat[] dtformat = new DateFormat[6];
    dtformat[0] = DateFormat.getInstance();
    dtformat[1] = DateFormat.getDateInstance();
    dtformat[2] = DateFormat.getDateInstance(DateFormat.MEDIUM);
    dtformat[3] = DateFormat.getDateInstance(DateFormat.FULL);
    dtformat[4] = DateFormat.getDateInstance(DateFormat.LONG);
    dtformat[5] = DateFormat.getDateInstance(DateFormat.SHORT);
    for(DateFormat dateform : dtformat) System.out.println(dateform.format(dt));
  }
}
```

Result

The above code sample will produce the following result.

```
9/9/01 7:16 AM
```

```
Sep 9, 2001
Sep 9, 2001
Sunday, September 9, 2001
September 9, 2001
9/9/01
```

The following is an another sample example of date in another format.

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;

public class SimpleDateFormatExample {
  public static void main(String[] args) {
    Date curDate = new Date();
    SimpleDateFormat format = new SimpleDateFormat("yyyy/MM/dd");
    String DateToStr = format.format(curDate);
    System.out.println(DateToStr);
    format = new SimpleDateFormat("dd-M-yyyy hh:mm:ss");
    DateToStr = format.format(curDate);
    System.out.println(DateToStr);
    format = new SimpleDateFormat("dd MMMM yyyy zzzz", Locale.ENGLISH);
    DateToStr = format.format(curDate);
    System.out.println(DateToStr);
        format = new SimpleDateFormat("E, dd MMM yyyy HH:mm:ss z");
    DateToStr = format.format(curDate);
    System.out.println(DateToStr);
        try {
         Date strToDate = format.parse(DateToStr);
         System.out.println(strToDate);
      } catch (ParseException e) {
         e.printStackTrace();
      }
   }
}
```

The above code sample will produce the following result.

```
2016/11/11
11-11-2016 07:12:27
11 November 2016 Coordinated Universal Time
Fri, 11 Nov 2016 07:12:27 UTC
Fri Nov 11 07:12:27 UTC 2016
```

6. Write a Java program that counts the number of objects created by using static variable

Java static variable

If you declare any variable as static, it is known as a static variable.

- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

Advantages of static variable

It makes your program **memory efficient** (i.e., it saves memory).

Understanding the problem without static variable

```
class Student{
     int rollno;
     String name;
     String college="ITS";
}
```

Suppose there are 500 students in my college, now all instance data members will get memory each time when the object is created. All students have its unique rollno and name, so instance data member is good in such case. Here, "college" refers to the common property of all objects. If we make it static, this field will get the memory only once.

Example of static variable

```
//Java Program to demonstrate the use of static variable
class Student{
   int rollno;//instance variable
   String name;
   static String college ="ITS";//static variable
   //constructor
   Student(int r, String n){
   rollno = r;
   name = n;
   }
   //method to display the values
   void display (){System.out.println(rollno+" "+name+" "+college);}
 }
 //Test class to show the values of objects
 public class TestStaticVariable1{
  public static void main(String args[]){
  Student s1 = new Student(111,"Karan");
  Student s2 = new Student(222,"Aryan");
  //we can change the college of all objects by the single line of code
  //Student.college="BBDIT";
  s1.display();
  s2.display();
  }
 }
```

1. Output:

```
111 Karan ITS
        222yan ITS
```

7.Write a java program that implements educational hierarchy using inheritance.

In this type of inheritance, there are more than one derived classes which get created from one single base class.

Example:

```
class Teacher {

 void teach() {

  System.out.println("Teaching subject");

 }

}

class Student extends Teacher {

 void listen() {

  System.out.println("Listening");

 }

}

class Principal extends Teacher {

 void evaluate() {

  System.out.println("Evaluating");

 }
}

class CheckForInheritance {

 public static void main(String argu[]) {

  Principal p = new Principal();

  p.evaluate();

  p.teach();

  // p.listen(); will produce an error

 }

}
```

8 **ArrayIndexOutOfBounds Exception :** It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

```
// Java program to demonstrate

// ArrayIndexOutOfBoundException

class ArrayIndexOutOfBound_Demo {

public static void main(String args[])

   {

      try {

         int a[] = new int[5];

         a[6] = 9; // accessing 7th element in an array of

         // size 5

      }
```

```
        catch (ArrayIndexOutOfBoundsException e) {

            System.out.println("Array Index is Out Of Bounds");

        }

    }

}
```

Output:

```
Array Index is Out Of Bounds
```

9.Write a java program that implements bank transactions using user defined exception

How to Create User Defined Exception Class In Java

Introduction :

Sometimes it is required to develop meaningful exceptions based on application requirements.For example suppose you have one savings account in any Bank and you have 50 dollars in your account.Suppose you attempted to withdraw 60 dollars from your account. In java you can handle this scenario by using ArithmeticException.But this Exception is not meaningful as a user point of view. You need to display some error message related to insufficient fund.

To have good meaning, we can create our own exception. If an exception like InSufficientFundException exists, it well suits for the problem of insufficient funds.

We can create our own exceptions by extending 'Exception' class.

Steps to implement User defined Exception class :

- The user defined exception class must extend from java.lang.Exception or java.lang.RunTimeException class. Also note that RuntimeException and its sub classes are not checked by the compiler and need not be declared in the method's signature.
- While creating custom exception, prefer to create an unchecked, Runtime exception than a checked exception.
- Apart from providing default no argument constructor on your custom Exception class, consider providing at least two more constructors, one which should accept a failure message and other which can accept another Throwable as cause.
- Every user defined exception class parametrized Constructor must called parametrized Constructor of either java.lang.Exception or java.lang.RunTimeException class by using super(string parameter always).

Here is complete example of how to create a user defined custom Exception in Java.

In this example we have an Account class, which is representing a bank account where you can deposit and withdraw money, but what will happen if you want to withdraw money which exceed your bank balance? You will not be allowed, and this is where user defined exception comes into picture. We have created a custom exception called InSufficientFundException to handle this scenario.

InSufficientFundException.java

This is Custom Exception class.In this class we have created two constructors for displaying error message.

```
package com.jwt.core.java;
 public class InSufficientFundException extends RuntimeException {
    private String message;
    public InSufficientFundException(String message) {
       this.message = message;
   }
    public InSufficientFundException(Throwable cause, String message) {
       super(cause);
       this.message = message;
   }
    public String getMessage() {
       return message;
   }}
```

Account.java

In this class we have created two methods withdraw and deposit.Initial Balance of Account is 3000 dollars.Withdraw method parameter is amount and that amount we are subtracting from initial balance. If amount passes is greater than initial balance, it will throw Custom Exception InSufficientFundException,else it will calculate new balance by subtracting amount from initial balance.

```
package com.jwt.core.java;
 public class Account {
    private int balance = 3000;
    public int balance() {
       return balance;
    }
    public void withdraw(int amount) throws InSufficientFundException {
       if (amount > balance) {
           throw new InSufficientFundException(String.format(
                   "Current balance %d is less than requested amount %d",
                   balance, amount));
       }
       balance = balance - amount;
    }
    public void deposit(int amount)
    {
        if (amount <= 0) {
            throw new IllegalArgumentException(String.format(
                    "Invalid deposit amount %s", amount));
        }
      }
    }
```

Test.java

Now create a test class to test our implemented Custom Exception Class.

```
   package com.jwt.core.java;
    public class Test {
       public static void main(String args[]) {
         Account acct = new Account();
         System.out.println("Current balance : " + acct.balance());
         System.out.println("Withdrawing $200");
         acct.withdraw(200);
         System.out.println("Current balance : " + acct.balance());
         acct.withdraw(3500);
       }
    }
```

Output

```
Current balance : 2800

Exception in thread "main" com.jwt.core.java.InSufficientFundException: Current balance 2800
is less than requested amount 3500

        at com.jwt.core.java.Account.withdraw(Account.java:13)

        at com.jwt.core.java.Test.main(Test.java:11)
```

10. Write a java program to identify the significance of finally block in handling exception.

Java Program to Use finally block for Catching Exceptions

The **finally block** in java is used to put important codes such as clean up code e.g. closing the file or closing the connection. The finally block executes whether exception rise or not and whether exception handled or not. A finally contains all the crucial statements regardless of the exception occurs or not.

There are 3 possible cases where finally block can be used:

Case 1: When an exception does not rise

In this case, the program runs fine without throwing any exception and finally block execute after the try block.

Java

```
// Java program to demonstrate

// finally block in java When

// exception does not rise

  import java.io.*;

  class GFG {

    public static void main(String[] args)

    {
```

```
    try {

        System.out.println("inside try block");

          // Not throw any exception

        System.out.println(34 / 2);

    }

            // Not execute in this case

    catch (ArithmeticException e)

        System.out.println("Arithmetic Exception");

            }

    // Always execute

    finally {

        System.out.println(

            "finally : i execute always.");

             }

}
```

```
}
```

Output

```
inside try block

17

finally: i execute always.
```

Case 2: When the exception rises and handled by the catch block

In this case, the program throws an exception but handled by the catch block, and finally block executes after the catch block.

Java

```
// Java program to demonstrate finally block in java

// When exception rise and handled by catch

  import java.io.*;

  class GFG {

        public static void main(String[] args)

    {

        try {

            System.out.println("inside try block");
            // Throw an Arithmetic exception
          System.out.println(34 / 0);
        }
          // catch an Arithmetic exception
        catch (ArithmeticException e) {
              System.out.println(
                "catch : exception handled.");
        }
          // Always execute
        finally {
          System.out.println("finally : i execute always.");
        }
    }
}
```

Output

```
inside try block
```

catch : exception handled.

finally : i execute always.

Case 3: When exception rise and not handled by the catch block

In this case, the program throws an exception but not handled by catch so finally block execute after the try block and after the execution of finally block program terminate abnormally, But finally block execute fine.

Java

```
// Java program to demonstrate finally block

// When exception rise and not handled by catch

  import java.io.*;

  class GFG {

    public static void main(String[] args)

    {

        try {

            System.out.println("Inside try block");

              // Throw an Arithmetic exception

            System.out.println(34 / 0);
        }

          // Can not accept Arithmetic type exception

        // Only accept Null Pointer type Exception

        catch (NullPointerException e) {

              System.out.println(

                "catch : exception not handled.");

        }

          // Always execute

        finally {

              System.out.println(

                "finally : i will execute always.");

        }

        // This will not execute

        System.out.println("i want to run");

    }
```

```
}
```

Output

```
Inside try block

finally : i will execute always.

Exception in thread "main" java.lang.ArithmeticException: / by zero

    at GFG.main(File.java:10
```

BIBLOGRAPHY

1) **https://www.javatpoint.com**
2) **https://www.geeksforgeeks.org**
3) **https://www.w3schools.com**
4) **http://javawithsuman.blogspot.com/p/applet-architecture.html**
5) **https://www.tutorialspoint.com**
6) **https://www.programiz.com**
7) **https://beginnersbook.com**
8) **https://www.guru99.com**