Name	Ahmad Raza Ansari
Roll No.	05
Department	IT
Batch	A
Year	2nd

Experiment: 01

Problem definition: -

Create a Teacher class and derive Professor/Associate_Professor/Assistant_Professor class from Teacher class. Define appropriate constructor for all the classes. Also define a method to display information of Teacher. Make necessary assumptions as required.

Theory: -

- A Java class constructor initializes instances (objects) of that class. Typically, the constructor initializes the fields of the object that need initialization. Java constructors can also take parameters, so fields can be initialized in the object at creation time.

- A Class is like an object constructor, or a "blueprint" for creating objects.

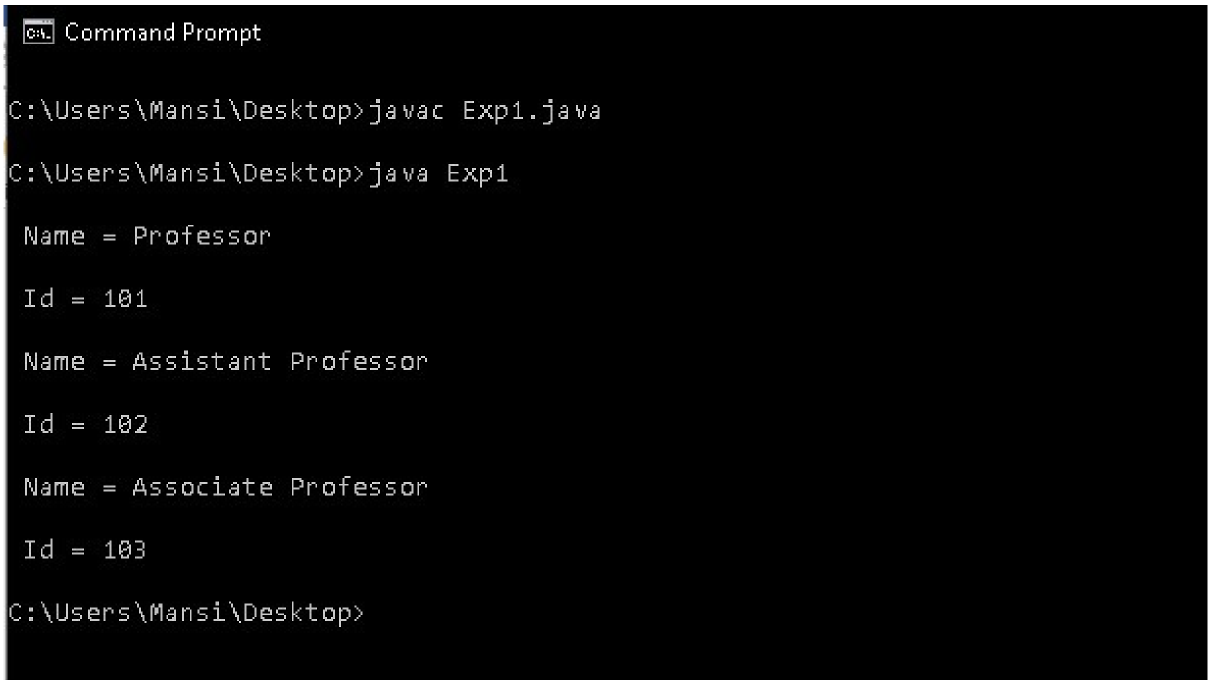
Code: -

Exp1.java

```
class Teacher
 {
    String teacherName;
    int teacherId;

    public Teacher(String teacherName, int teacherId)
    {
       this.teacherName = teacherName;
       this.teacherId = teacherId;
    }

    public void display()
    {
       System.out.println("\n Name = "+teacherName);
       System.out.println("\n Id = "+teacherId);
    }
}
class Professor extends Teacher
{
    String teacherName;
    int teacherId;
    public Professor(String teacherName, int teacherId)
    {
        super(teacherName, teacherId);
        this.teacherName = teacherName;
        this.teacherId = teacherId;
    }
    public void display()
    {
        System.out.println("\n Name = "+teacherName);
        System.out.println("\n Id = "+teacherId);
    }
}
class Associate_Professor extends Teacher
{
    String teacherName;
    int teacherId;
    public Associate_Professor(String teacherName, int teacherId)
    {
        super(teacherName, teacherId);
        this.teacherName = teacherName;
        this.teacherId = teacherId;
    }
    public void display()
    {
        System.out.println("\n Name = "+teacherName);
        System.out.println("\n Id = "+teacherId);
    }
}
class Assistant_Professor extends Teacher
{
    String teacherName;
    int teacherId;
    public Assistant_Professor(String teacherName, int teacherId)
    {
        super(teacherName, teacherId);
        this.teacherName = teacherName;
        this.teacherId = teacherId;
    }

    public void display()
    {
        System.out.println("\n Name = "+teacherName);
        System.out.println("\n Id = "+teacherId);
    }
}
public class Exp1
{
    public static void main(String [] args)
    {
        Professor P1 = new Professor("Professor", 101);
        Assistant_Professor P2 = new Assistant_Professor("Assistant Professor", 102);
        Assistant_Professor P3 = new Assistant_Professor("Associate Professor", 103);
```

```
                P1.display();

                P2.display();

                P3.display();

        }

    }
```

Output: -



```
C:\Users\Mansi\Desktop>javac Exp1.java

C:\Users\Mansi\Desktop>java Exp1

 Name = Professor

 Id = 101

 Name = Assistant Professor

 Id = 102

 Name = Associate Professor

 Id = 103

C:\Users\Mansi\Desktop>
```

Conclusion: -

Hence, we have successfully performed the program of classes and object.

Experiment: 02

Problem Definition : -

An employee works in a particular department of an organization. Every employee has an employee number, name and draws a particular salary. Every department has a name and a head of department. The head of department is an employee. Every year a new head of department takes over. Also, every year an employee is given an annual salary enhancement. Identify and design the classes for the above description with suitable instance variables and methods. The classes should be such that they implement information hiding. You must give logic in support of your design. Also create two objects of each class.

Theory: -

Instance variables in Java are non-static variables which are defined in a class outside any method, constructor or a block. Each instantiated object of the class has a separate copy or instance of that variable. An instance variable belongs to a class. A class can have two types of methods: instance methods and class methods. Instance methods are also called non-static methods. Class methods are also called static methods. An instance method implements behaviour for the instances of the class. An instance method can only beinvoked by an instance of the class. A static method implements the behaviour for the class itself. A class method always executes by a class. Rule It is not allowed to refer to instance variables from inside a static method. A static method can refer to only static variables. A non-static method can refer to both static variables and non-static variables

Algorithm: -

1. Start the Program
2. Create a Class
3. Declare the Various Method
4. Create another class with main method.
5. Call the main method and create object
6. Display the object

7.Stop.

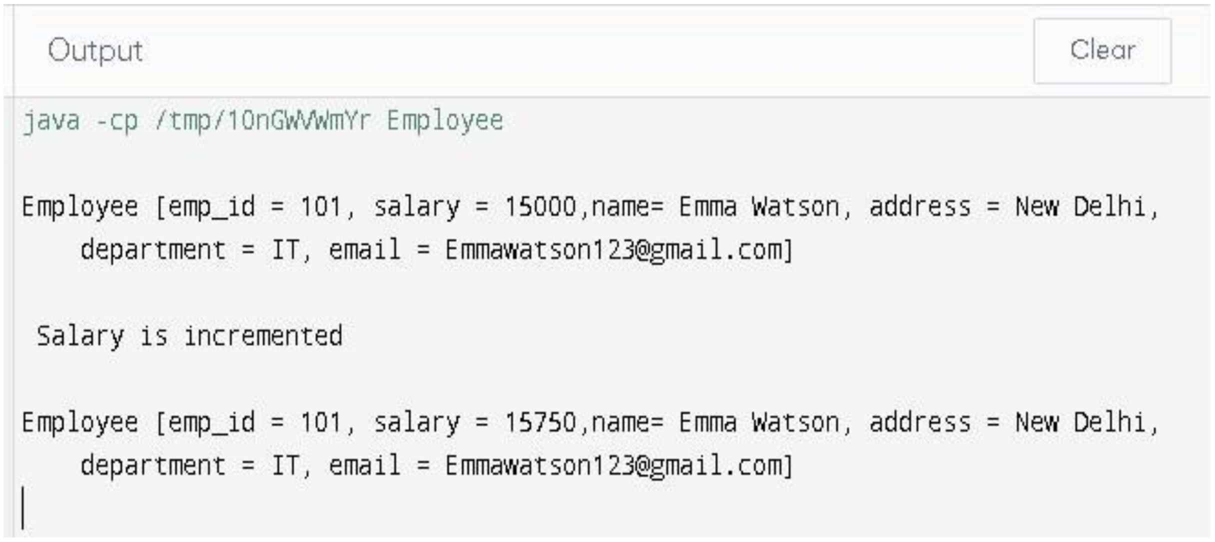
Code: -

```
class EmployeeDetails
```

```
{
    int emp_id, salary;
String name, address, department, email;
public int getEmp_id()
{
 return emp_id;
 }
public void setEmp_id(int emp_id)
{
 this.emp_id = emp_id;
}
public int getSalary() {
 return salary;
}
public void setSalary(int salary)
{
 this.salary = salary;
 }
public String getName() {
 return name;
}
public void setName(String name)
{
 this.name = name;

}
public String getAddress() {
 return address;
 }
```

```
public void setAddress(String address) {
 this.address = address;
 }
public String getDepartment()
{
 return department;
}
public void setDepartment(String department) {
 this.department = department; }
public String getEmail()
{
 return email;

}
public void setEmail(String email)
{
 this.email = email;
 }
@Override
public String toString() {
 return "Employee [emp_id = " + emp_id + ", salary = " + salary + ",name= " + name + ",
address = " +
address
 + ", department = " + department + ", email = " + email + "]"; } }
class Employee
{ public static void main(String args[])
{
 EmployeeDetails emp = new EmployeeDetails();
 emp.setEmp_id(101);
 emp.setName("Emma Watson");
 emp.setDepartment("IT");
 emp.setSalary(15000);
 emp.setAddress("New Delhi");
 emp.setEmail("Emmawatson123@gmail.com");
 System.out.println(emp);
 int sal = emp.getSalary();
 int increment = 0;
 if ((sal >=1000) && (sal <=1500))
 { increment += (sal * 2)/100;
 sal = sal+increment;
 emp.setSalary(sal);
 System.out.println("\n Salary is incremented \n");
 System.out.println(emp);
 }else if ((sal >=1500) && (sal <=20000)){
 increment += (sal * 5)/100;
 sal = sal+increment;
 emp.setSalary(sal);
 System.out.println("\n Salary is incremented \n");
 System.out.println(emp);
 }
 else {
 System.out.println("\n Salary is not incremented \n");
 System.out.println(emp);
 }
}
}
```

Output: -



Output

Clear

```
java -cp /tmp/1OnGWVWmYr Employee

Employee [emp_id = 101, salary = 15000,name= Emma Watson, address = New Delhi,
    department = IT, email = Emmawatson123@gmail.com]

 Salary is incremented

Employee [emp_id = 101, salary = 15750,name= Emma Watson, address = New Delhi,
    department = IT, email = Emmawatson123@gmail.com]
```

Conclusion: -

In the following experiment we see that the annual salary of an employee has been incremented.

Experiment no :- 03

Title: To study Inheritance, Interfaces

Problem Statement: Consider a hierarchy, where a sportsperson can either be an athlete

or a hockey player. Every sportsperson has a unique name. An athlete

is characterized by the event in which he/she participates; whereas a

hockey player is characterised by the number of goals scored by

him/her.

Perform the following tasks using Java :

(i) Create the class hierarchy with suitable instance variables and

methods.

(ii) Create a suitable constructor for each class.

(iii) Create a method named display_all_info with suitable

parameters. This method should display all the information about the

object of a class.

(iv) Write the main method that demonstrates polymorphism.

Theory: Interfaces in Java

Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).

- Interfaces specify what a class must do and not how. It is the blueprint of the class.
- An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to (or must implement) move(). So it specifies a set of methods that the class has to implement.
- If a class implements an interface and does not provide method bodies

for all functions specified in the interface, then the class must be declared abstract.

To declare an interface, use **interface** keyword. It is used to provide total abstraction. That means all the methods in an interface are declared with an empty body and are public and all fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface. To implement interface use **implements** keyword.

Why do we use interface ?

- It is used to achieve total abstraction.
- Since java does not support multiple inheritance in case of class, but by using interface it can achieve multiple inheritance .

Algorithm:

Step 1: Start the program.

Step 2: Declare the Interface Sportperson.

Step 3: Declare the display_all_info member function.

Step 4: Declare te class Hockey and implements thw Interface Sportperson.

Step 5: Declare and define constructor Hockey and derive the fuction and from the Interface and define it.

Step 6: Declsre the class Athletc and implements the interface Sportperson.

Step 7: Declare and define constructor Athletc and derive the fuction and from the Interface and define it.

Step 8: Declare the main class.

Step 9: Declare the derived class object and call the function using object and also invoke the constructor.

Step 10: Stop the Program.

Program: interface Sportsperson

```
{
void display_all_info();
}

class Hockey implements Sportsperson
{
String name;
int age;
int goals;
public Hockey(String n, int a, int g)
{
name = n;
age = a;
goals = g;
}

public void display_all_info()
{
System.out.println("Hockey Player:" + " Name: "+ name + " Age: "+ age +"
Number of goals:" + goals);
}

}

class Athlete implements Sportsperson
```

```
{
String type;
String name;
int age;
public Athlete(String n, int a, String t)
{
name = n;
age = a;
type = t;
}

public void display_all_info()
{
System.out.println("Athlete:" + " Name: "+ name + " Age: "+ age +" Type of
athlete:" + type);
}
}

public class Exp3
{
public static void main(String args[])
{
Hockey h1 = new Hockey("Dhyan Chand",60,100);
Hockey h2 = new Hockey("Manpreet Singh",30,50);
Athlete a1 = new Athlete("P.T.Usha",50,"Running");
```

Athlete a2 = new Athlete("Michael Phelps",35,"Swimming");

h1.display_all_info();

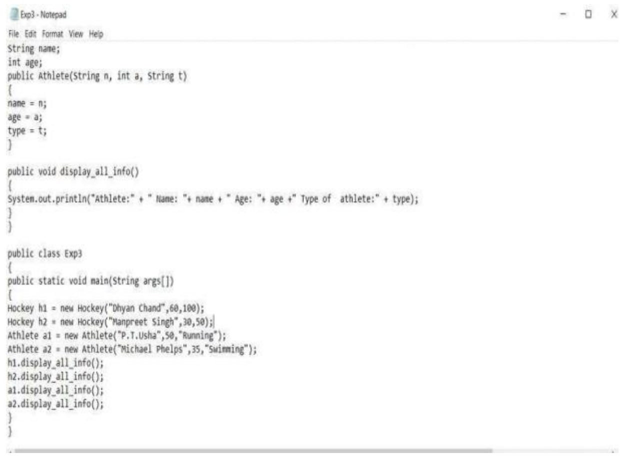
h2.display_all_info();

a1.display_all_info();

a2.display_all_info();

}}

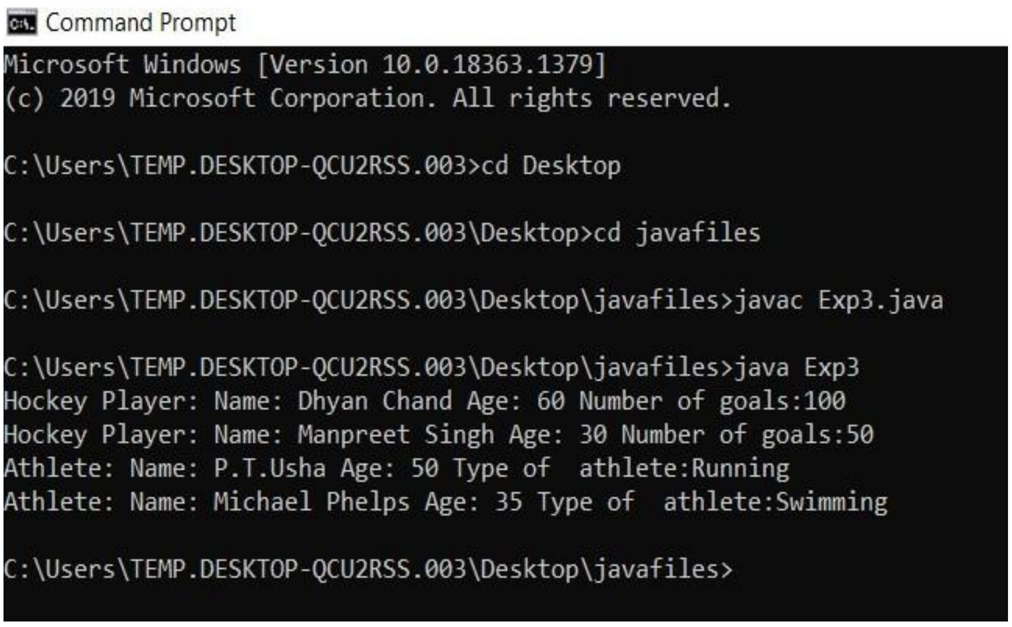
Atual Input/Output:



Exp3 - Notepad

File Edit Format View Help

```
String name;
int age;
public Athlete(String n, int a, String t)
{
name = n;
age = a;
type = t;
}

public void display_all_info()
{
System.out.println("Athlete:" + " Name: "+ name + " Age: "+ age +" Type of  athlete:" + type);
}
}

public class Exp3
{
public static void main(String args[])
{
Hockey h1 = new Hockey("Dhyan Chand",60,100);
Hockey h2 = new Hockey("Manpreet Singh",30,50);
Athlete a1 = new Athlete("P.T.Usha",50,"Running");
Athlete a2 = new Athlete("Michael Phelps",35,"Swimming");
h1.display_all_info();
h2.display_all_info();
a1.display_all_info();
a2.display_all_info();
}
}
```



Command Prompt

```
Microsoft Windows [Version 10.0.18363.1379]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\TEMP.DESKTOP-QCU2RSS.003>cd Desktop

C:\Users\TEMP.DESKTOP-QCU2RSS.003\Desktop>cd javafiles

C:\Users\TEMP.DESKTOP-QCU2RSS.003\Desktop\javafiles>javac Exp3.java

C:\Users\TEMP.DESKTOP-QCU2RSS.003\Desktop\javafiles>java Exp3
Hockey Player: Name: Dhyan Chand Age: 60 Number of goals:100
Hockey Player: Name: Manpreet Singh Age: 30 Number of goals:50
Athlete: Name: P.T.Usha Age: 50 Type of  athlete:Running
Athlete: Name: Michael Phelps Age: 35 Type of  athlete:Swimming

C:\Users\TEMP.DESKTOP-QCU2RSS.003\Desktop\javafiles>
```

bxdjh

EXPERIMENT NO-04

Aim: Create an interface vehicle and classes like bicycle, car, bike etc, having common functionalities and put all the common functionalities in the interface. Classes like Bicycle, Bike, car etc implement all these functionalities in their own class in their own way.

Theory:

Interfaces in java:

Like a class, interfaces in Java can have methods and variables, but the methods declared in interface are by default abstract (only method signature, no body).

- Interfaces specify what a class must do and not how. It is the blueprint of the class.
- An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to (or must implement) move(). So it specifies a set of methods that the class has to implement.
- If a class implements an interface and does not provide method bodies for all functions specified in the interface, then class must be declared abstract.
- A Java library example is, Comparator Interface. If a class implements this interface, then it can be used to sort a collection.

Syntax :

```
interface <interface_name>{


    // declare constant fields
    // declare methods that abstract
    // by default.
}
```

To declare an interface, use **interface** keyword. It is used to provide total abstraction. That means all the methods in interface are declared with empty body and are public and all fields are public, static and final by default. A class that implement interface must implement all the methods declared in the interface. To implement interface use **implements** keyword.

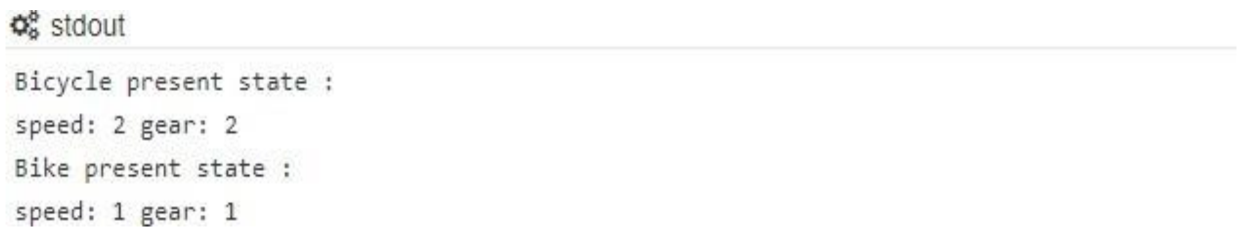
Program:

```
import java.io.*;
interface Vehicle {
        void changeGear(int a);
        void speedUp(int a);
        void applyBrakes(int a);
}

class Bicycle implements Vehicle{

        int speed;
        int gear;
        @Override
        public void changeGear(int newGear){

                gear = newGear;
        }
        @Override
        public void speedUp(int increment){

                speed = speed + increment;
        }
        @Override
        public void applyBrakes(int decrement){

                speed = speed - decrement;
        }

        public void printStates() {
```

```
        System.out.println("speed: " + speed
            + " gear: " + gear);
    }
}

class Bike implements Vehicle {

    int speed;
    int gear;
    @Override
    public void changeGear(int newGear){

        gear = newGear;
    }
    @Override
    public void speedUp(int increment){

        speed = speed + increment;
    }
    @Override
    public void applyBrakes(int decrement){

        speed = speed - decrement;
    }

    public void printStates() {
        System.out.println("speed: " + speed
            + " gear: " + gear);
    }

}
class GFG {

    public static void main (String[] args) {
        Bicycle bicycle = new Bicycle();
```

```
                bicycle.changeGear(2);
                bicycle.speedUp(3);
                bicycle.applyBrakes(1);

                System.out.println("Bicycle present state :");
                bicycle.printStates();
                Bike bike = new Bike();
                bike.changeGear(1);
                bike.speedUp(4);
                bike.applyBrakes(3);

                System.out.println("Bike present state :");
                bike.printStates();
        }
}
```

Output:



```
Bicycle present state :
speed: 2 gear: 2
Bike present state :
speed: 1 gear: 1
```

Conclusion: In this program we have created an interface vehicle and classes like bicycle, car, bike etc, having common functionalities and put all the common functionalities in the interface. Classes like Bicycle, Bike, car etc implement all these functionalities in their own class in their own way.

EXPERIMENT NO.05

Aim: Study of Packages. (Create a class “Amount in Words” within a user defined package to convert the amount into words. (Consider amount not to be more than 100000)

Lab Outcome: 2.ITL304.3

Problem Statement:

Create a class “Amount In Words” within a user defined package to convert the amount into words. (Consider amount not to be more than 100000).

Theory:

Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces. Packages are used for:

- Preventing naming conflicts. For example there can be two classes with name Employee in two packages, college.staff.cse.Employee and college.staff.ee.Employee
- Making searching/locating and usage of classes, interfaces, enumerations and annotations easier
- Providing controlled access: protected and default have package level access control. A protected member is accessible by classes in the same package and its subclasses. A

default member (without any access specifier) is accessible by classes in the same package only.

- Packages can be considered as data encapsulation (or data-hiding).

How Package Workes?

Package names and directory structure are closely related. For example if a package name is college.staff.cse, then there are three directories, college, staff and cse such that cse is present in staff and staff is present college. Also, the directory college is accessible through CLASSPATH variable, i.e., path of parent directory of college is present in CLASSPATH. The idea is to make sure that classes are easy to locate.

Package naming conventions : Packages are named in reverse order of domain names, i.e., org.geeksforgeeks.practice. For example, in a college, the recommended convention is college.tech.cse, college.tech.ee, college.art.history, etc.

Source Code:

Package File

```
package MyPackage;
import java.util.*;
public class MyPackage {
String string;
String st1[] =
{"","one","two","three","four","five","six","seven","eight","n
ine"};
String st2[] = {"hundred","thousand","lakh","crore"};
String st3[] =
{"ten","eleven","twelve","thirteen","fourteen","fifteen","sixt
een","seventeen","eighteen","ninteen"};
String st4[] =
{"twenty","thirty","fourty","fifty","sixty","seventy","eighty"
,"ninty"};
public String convert(int number) {
int n = 1;
int word;
string = "";
while (number != 0) {
switch (n) {
case 1:
word = number % 100;
pass(word);
if (number > 100 && number % 100 != 0) {
show("and ");
}
```

```
number /= 100;
break;
case 2:
word = number % 10;
if (word != 0) {
show(" ");
show(st2[0]);

show(" ");
pass(word);
}
number /= 10;
break;
case 3:
word = number % 100;
if (word != 0) {
show(" ");
show(st2[1]);
show(" ");
pass(word);
}
number /= 100;
break;
case 4:
word = number % 100;
if (word != 0) {
show(" ");
show(st2[2]);
show(" ");
pass(word);
}
```

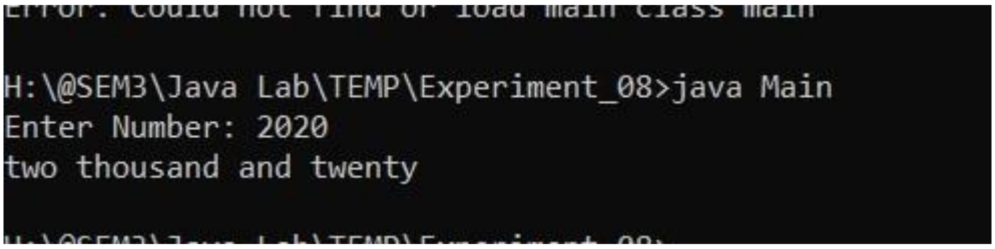
```
number /= 100;
break;
case 5:
word = number % 100;
if (word != 0) {
show(" ");
show(st2[3]);
show(" ");
pass(word);
}

number /= 100;
break;
}
n++;
}
return string;
}
public void pass(int number) {
int word, q;
if (number < 10) {
show(st1[number]);
}
if (number > 9 && number < 20) {
show(st3[number - 10]);
}
if (number > 19) {
word = number % 10;
if (word == 0) {
q = number / 10;
show(st4[q - 2]);
```

```
} else {
q = number / 10;
show(st1[word]);
show(" ");
show(st4[q - 2]);
}
}
}
public void show(String s) {
String st;
st = string;
string = s;
string += st;

}
}
```

MAIN CLASS FILE:

```
import MyPackage.MyPackage;
import java.util.*;
public class Main
{
public static void main(String[] args) {
MyPackage mp = new MyPackage();
Scanner input = new Scanner(System.in);
System.out.print("Enter Number: ");
int num  =  input.nextInt();
String inwords = mp.convert(num);
System.out.println(inwords);
}
}
```

OUTPUT:



```
Error: Could not find or load main class main

H:\@SEM3\Java Lab\TEMP\Experiment_08>java Main
Enter Number: 2020
two thousand and twenty

H:\@SEM3\Java Lab\TEMP\Experiment_08>
```

Conclusion:

Created a package name MyPackage and wrote code for Converting a number into word imported the same package “MyPackage” into Main Class File and asked user to Enter a number and get output as word

Experiment No – 6

Aim : Write java program where user will enter loginid and password as input. The password should be 8 digit containing one digit and one special symbol.
If user enter valid password satisfying above criteria then show "Login Successful Message".
If user enter invalid Password then create InvalidPasswordException stating "Please enter valid password of length 8 containing one digit and one Special Symbol".

Theory:

Whenever at any line exception is generated then Java immediately terminates at the execution of the program. Java displays the error message also related to that exception i.e. it defines the reason of exception but the problem is its highly technical and not user friendly error message especially for an end user. To overcome above two drawbacks programmer will make use of exception handling mechanism.

It's the behaviour of a program after an exception occurs. In simple words, programmer can handle the exceptions as per their needs.

In Java, an exception is an object that wraps an error event that occurred within a method and contains:

- ✓ Information about the error including its type
- ✓ The state of the program when the error occurred

Exception objects can be thrown and caught.

In Java we have five keywords which are used to handle the exceptions in the programmers own way i.e. they are **try, catch, throw, throws** and **finally**.

The core advantage of exception handling is to maintain the normal flow of the application. An exception normally disrupts the normal flow of the application that is why we use exception handling.

try : The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.

catch: The "catch" block is used to handle the exception. It must be preceded by try block whichmeans we can't use catch block alone. It can be followed by finally block later.

throw: The "throw" keyword is used to explicitly throw an exception. We can throw either checked or unchecked exception in java by throw keyword.

throws: The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

finally: The "finally" block is used to execute the important code of the program. It is executedwhether an exception is handled or not.

Creating an Exception Class

If you wish to define your own exception

1. Pick a self-describing Exception class name.

2. Decide if the exception should be checked or unchecked.

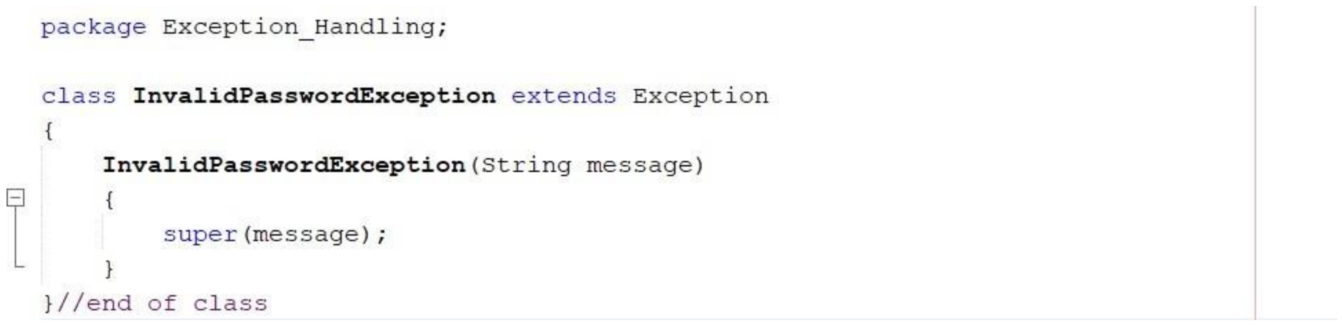
If checked then use extends Exception and if unchecked then use extends RuntimeException and then define constructors that call into super class constructors, taking message and/or cause parameters.

Code:

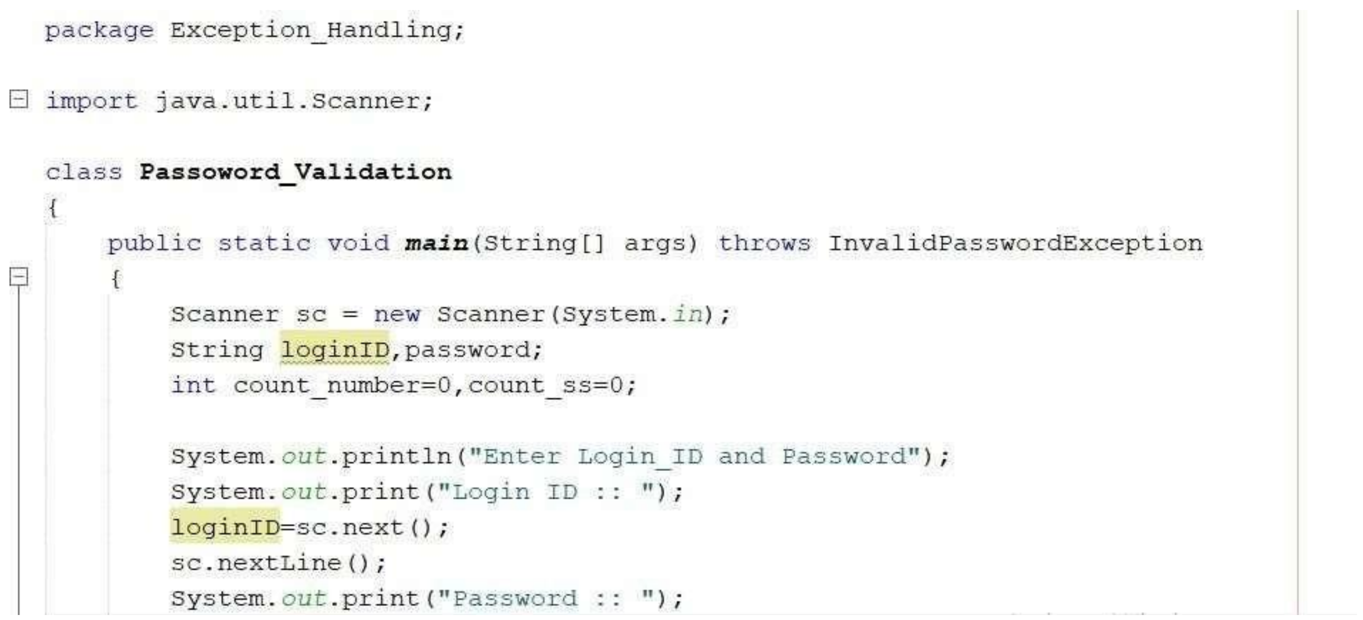
```
package Exception_Handling;
class InvalidPasswordException extends Exception
{
        InvalidPasswordException(String message)
        {
             super(message);
        }
}//end of class
---------------------------------------------------------------------------------------------------------------------------------------------
package Exception_Handling;
import   java.util.Scanner;
class Passoword_Validation
{
       public static void main(String[] args) throws InvalidPasswordException
       {
               Scanner sc = new Scanner(System.in);
```

```
String loginID,password;
int count_number=0,count_ss=0;
System.out.println("Enter Login_ID and Password");
System.out.print("Login ID :: ");
loginID=sc.next();
sc.nextLine();
System.out.print("Password :: ");
password=sc.nextLine();
char c[]=password.toCharArray();

//Digit Validation
for(int i=0;i<password.length();i++)
{
        if(c[i]>='0' && c[i]<='9')
        {
                count_number++;
        }//end of if
}//end of for

//Special Symbol Validation
for(int i=0;i<password.length();i++)
{
        if((c[i]<=47 || c[i]>=58) && (c[i]<=64 || c[i]>=91) && (c[i]<=96 ||
                c[i]>=123))
        {
                count_ss++;
        }//end of if
}//end of for
try
```

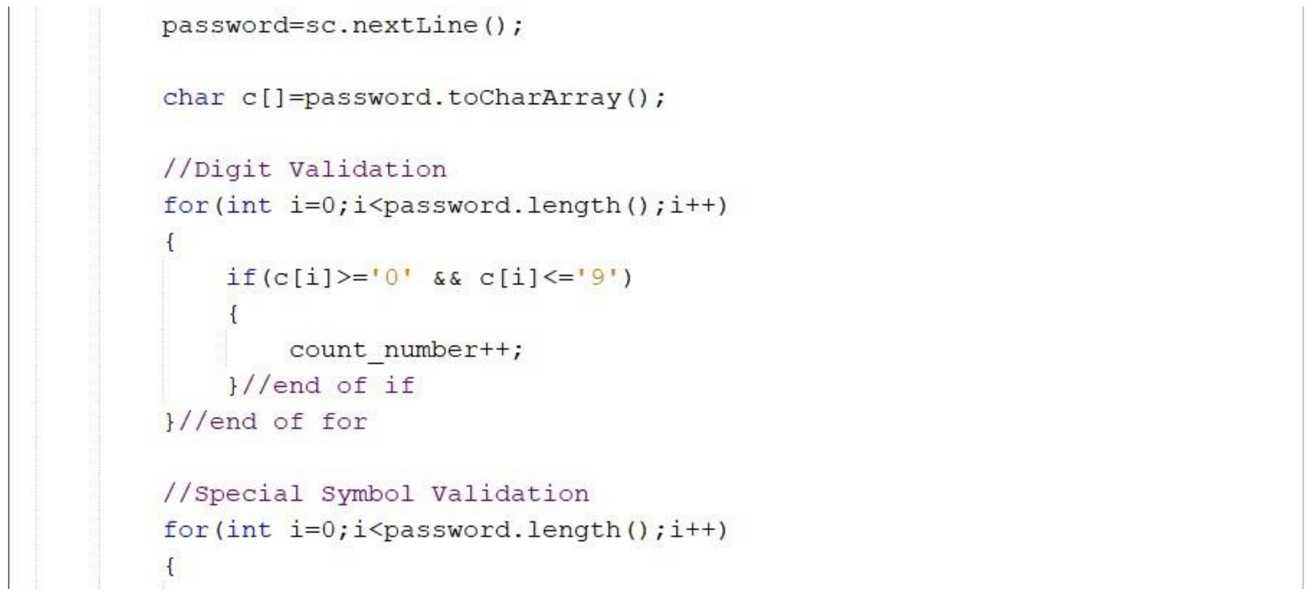
```
		{
			if(password.length()!=8 || count_number!=1 || count_ss!=1)
			{
				InvalidPasswordException ipe;
				ipe = new InvalidPasswordException("Please enter"
						+ " valid password of length 8 containing one digit"
						+ " and one Special Symbol");
				throw ipe;
			}//end of if
			else
			{
				System.out.println("Login Successful");
			}//end of else
		}//end of try
		catch(InvalidPasswordException ipe)
		{
			System.out.println(ipe.getMessage());
		}//end of catch
	}//end of main()
}//end of class
```



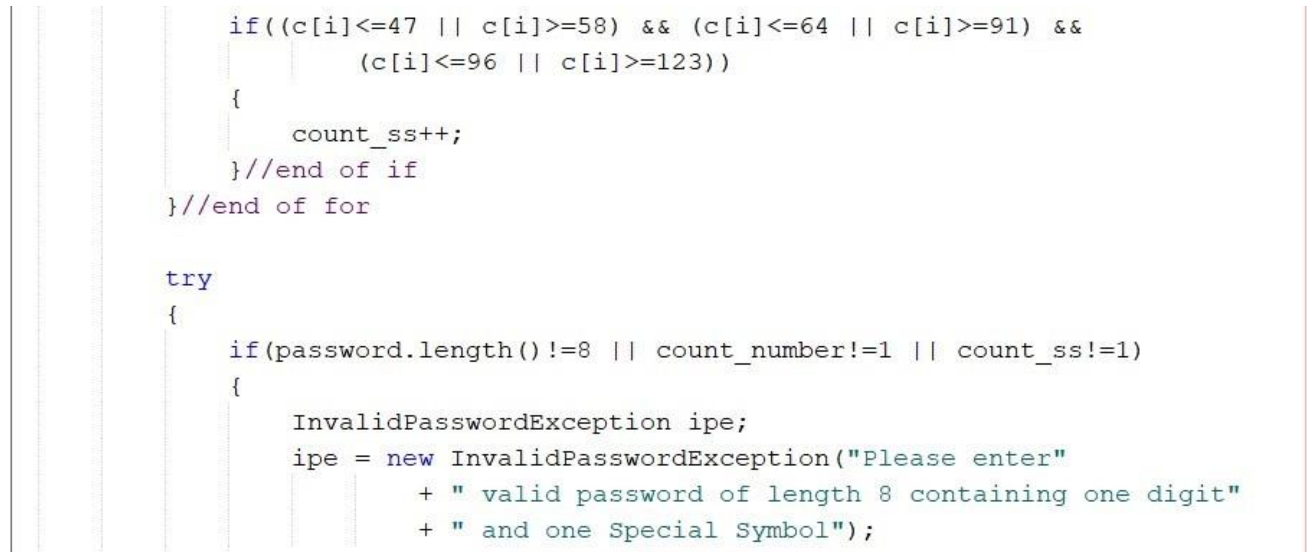
```
package Exception_Handling;

class InvalidPasswordException extends Exception
{
    InvalidPasswordException(String message)
    {
        super(message);
    }
}//end of class
```



```
package Exception_Handling;

import java.util.Scanner;

class Passoword_Validation
{
    public static void main(String[] args) throws InvalidPasswordException
    {
        Scanner sc = new Scanner(System.in);
        String loginID,password;
        int count_number=0,count_ss=0;

        System.out.println("Enter Login_ID and Password");
        System.out.print("Login ID :: ");
        loginID=sc.next();
        sc.nextLine();
        System.out.print("Password :: ");
```

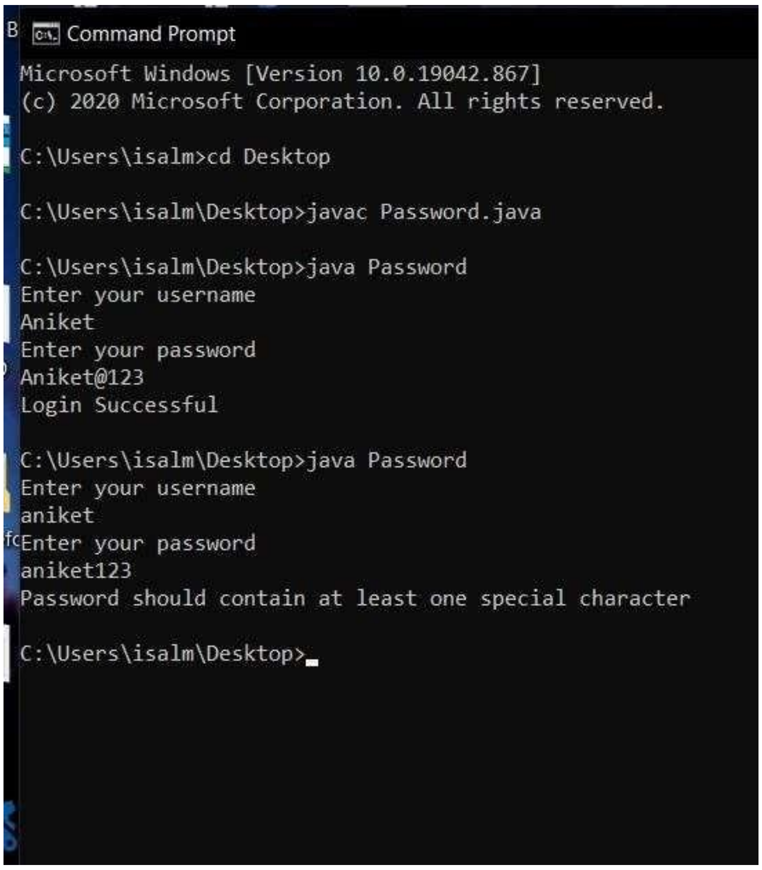


```
        password=sc.nextLine();

        char c[]=password.toCharArray();

        //Digit Validation
        for(int i=0;i<password.length();i++)
        {
            if(c[i]>='0' && c[i]<='9')
            {
                count_number++;
            }//end of if
        }//end of for

        //Special Symbol Validation
        for(int i=0;i<password.length();i++)
        {
```



```
            if((c[i]<=47 || c[i]>=58) && (c[i]<=64 || c[i]>=91) &&
                    (c[i]<=96 || c[i]>=123))
            {
                count_ss++;
            }//end of if
        }//end of for

        try
        {
            if(password.length()!=8 || count_number!=1 || count_ss!=1)
            {
                InvalidPasswordException ipe;
                ipe = new InvalidPasswordException("Please enter"
                        + " valid password of length 8 containing one digit"
                        + " and one Special Symbol");
                throw ipe;
            }//end of if
            else
            {
                System.out.println("Login Successful");
            }//end of else
        }//end of try
        catch(InvalidPasswordException ipe)
        {
            System.out.println(ipe.getMessage());
        }//end of catch
    }//end of main()
}//end of class
```

Output:



```
Microsoft Windows [Version 10.0.19042.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\isalm>cd Desktop

C:\Users\isalm\Desktop>javac Password.java

C:\Users\isalm\Desktop>java Password
Enter your username
Aniket
Enter your password
Aniket@123
Login Successful

C:\Users\isalm\Desktop>java Password
Enter your username
aniket
Enter your password
aniket123
Password should contain at least one special character

C:\Users\isalm\Desktop>
```

Conclusion:

In the above experiment we understood the mechanism of exception handling i.e. the behaviour of program after an exception occurs along with its hierarchy. We also learnt about explicitly throwing an exception using throw keyword and creating our own Exception Class and handling it i.e. handling custom exceptions.

Experiment no :- 07

Title:-

Inheritance And Exceptional Handling

Problem statement:-

Java program to create:-
i. Java program to create with 1000Rs minimum balance, adeposit() method to deposit amount, a withdraw() method to withdraw amount and also throws LessBalanceException if an account holder tries to withdraw money which makes the balance become less than 1000Rs.
ii. A Class called LessBalanceException which returns the statement that says withdraw amount (Rs) is not valid.
iii. A Class which creates 2 accounts, both account deposit money and one account tries to withdraw more money which generates a LessBalanceException take appropriate action for the same.

Theory:-.

Exception handling is one of the most important feature of java programming that allows us to handle the runtime errors caused by exceptions. In this guide, we will learn what is an exception, types of it, exception classes and how to handle exceptions in java with examples.

An Exception is an unwanted event that interrupts the normal flow of the program. When an exception occurs program execution gets terminated. In such cases we get a system generated error message. The good thing about exceptions is that they can be handled in Java. By handling the exceptions we can provide a meaningful message to the user about the issue rather than a system generated message, which may not be understandable to a user.

Algorithm :-

1. The try-catch block is used to handle exceptions in Java. Here's the syntax of try...catch block:

```
try {

  // code

}

catch(Exception e) {
  // code

}
```

2. Here, we have placed the code that might generate an exception inside the try block. Every try block is followed by a catch block.

3. When an exception occurs, it is caught by the catch block. The catch block cannot be used without the try block.

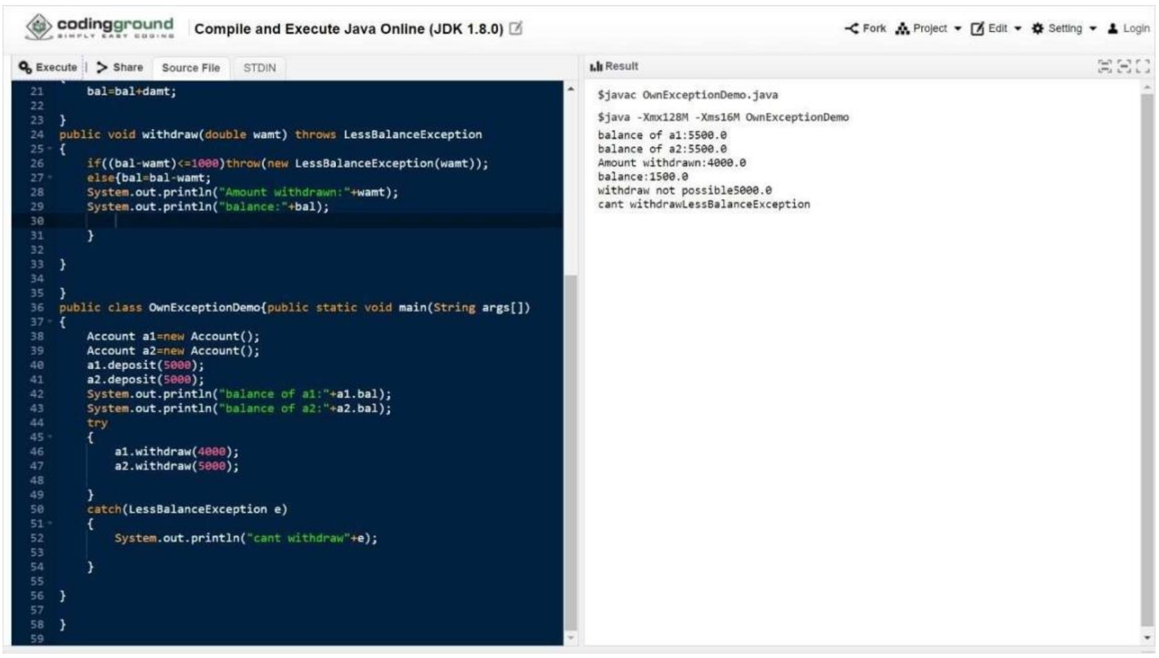
Program:-

```
import java.io.*;

class LessBalanceException extends Exception

{

      double amt;

      LessBalanceException(double wamt)

      {

            amt=wamt;

            System.out.println("withdraw not possible"+amt);

      }

}

class Account

{

      public double bal;

      Account(){bal=500.0;

      }
```

```
public void deposit(double damt)
{
     bal=bal+damt;
}
public void withdraw(double wamt) throws LessBalanceException
{
     if((bal-wamt)<=1000)throw(new LessBalanceException(wamt));
     else{bal=bal-wamt;
     System.out.println("Amount withdrawn:"+wamt);
     System.out.println("balance:"+bal);
     }
}
}
public class OwnExceptionDemo{public static void main(String args[])
{
     Account a1=new Account();
     Account a2=new Account();
     a1.deposit(5000);
     a2.deposit(5000);
     System.out.println("balance of a1:"+a1.bal);
     System.out.println("balance of a2:"+a2.bal);
     try
     {
          a1.withdraw(4000);
          a2.withdraw(5000);
     }
     catch(LessBalanceException e)
```

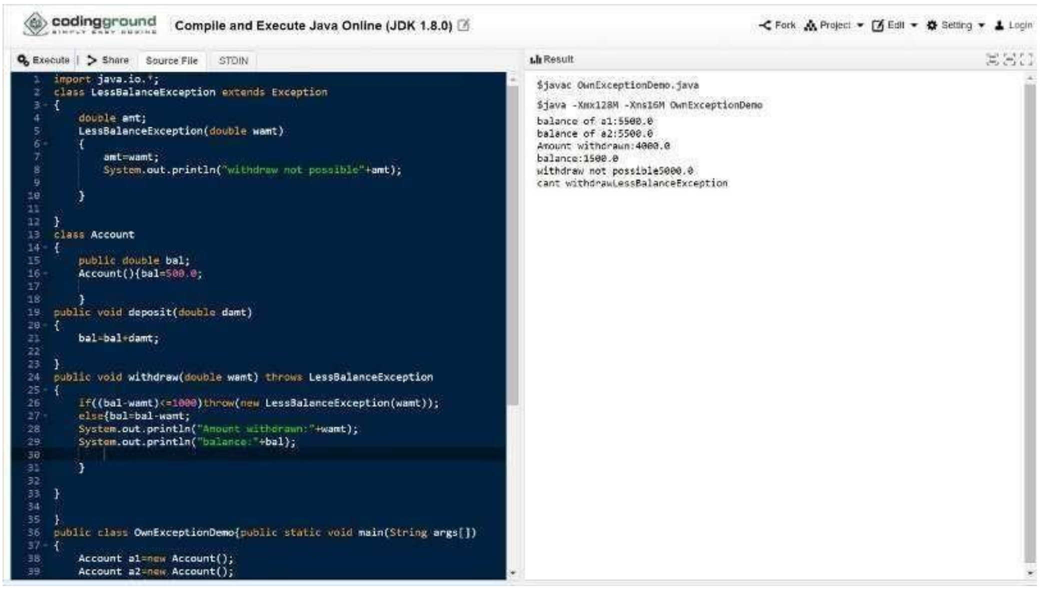
```
        {
                System.out.println("cant withdraw"+e);
        }
    }
}
```

Input and output :



```
    bal=bal+damt;

}
public void withdraw(double wamt) throws LessBalanceException
{
    if((bal-wamt)<=1000)throw(new LessBalanceException(wamt));
    else{bal=bal-wamt;
    System.out.println("Amount withdrawn:"+wamt);
    System.out.println("balance:"+bal);

    }

}

}
public class OwnExceptionDemo{public static void main(String args[])
{
    Account a1=new Account();
    Account a2=new Account();
    a1.deposit(5000);
    a2.deposit(5000);
    System.out.println("balance of a1:"+a1.bal);
    System.out.println("balance of a2:"+a2.bal);
    try
    {
        a1.withdraw(4000);
        a2.withdraw(5000);

    }
    catch(LessBalanceException e)
    {
        System.out.println("cant withdraw"+e);

    }

}

}
```

```
$javac OwnExceptionDemo.java

$java -Xmx128M -Xms16M OwnExceptionDemo
balance of a1:5500.0
balance of a2:5500.0
Amount withdrawn:4000.0
balance:1500.0
withdraw not possible5000.0
cant withdrawLessBalanceException
```



```
import java.io.*;
class LessBalanceException extends Exception
{
    double amt;
    LessBalanceException(double wamt)
    {
        amt=wamt;
        System.out.println("withdraw not possible"+amt);

    }

}
class Account
{
    public double bal;
    Account(){bal=500.0;

    }
public void deposit(double damt)
{
    bal=bal+damt;

}
public void withdraw(double wamt) throws LessBalanceException
{
    if((bal-wamt)<=1000)throw(new LessBalanceException(wamt));
    else{bal=bal-wamt;
    System.out.println("Amount withdrawn:"+wamt);
    System.out.println("balance:"+bal);

    }

}

}
public class OwnExceptionDemo{public static void main(String args[])
{
    Account a1=new Account();
    Account a2=new Account();
```

```
$javac OwnExceptionDemo.java

$java -Xmx128M -Xms16M OwnExceptionDemo
balance of a1:5500.0
balance of a2:5500.0
Amount withdrawn:4000.0
balance:1500.0
withdraw not possible5000.0
cant withdrawLessBalanceException
```

Conclusion:-

Thus we have created account using java program

Experiment no :- 08

Title : Create two threads such that one thread will print even number and another will print odd number in an ordered fashion.

Theory :

Multithreading in Java :

Multithreading in Java is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Advantages of Java Multithreading :

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time**.

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

How to create thread :

There are two ways to create a thread:

1. By extending Thread class : We create a class that extends the **java.lang.Thread** class. This class overrides the run() method available in the Thread class. A thread begins its life inside run() method. We create an object of our new class and call start() method to start the execution of a thread. Start() invokes the run() method on the Thread

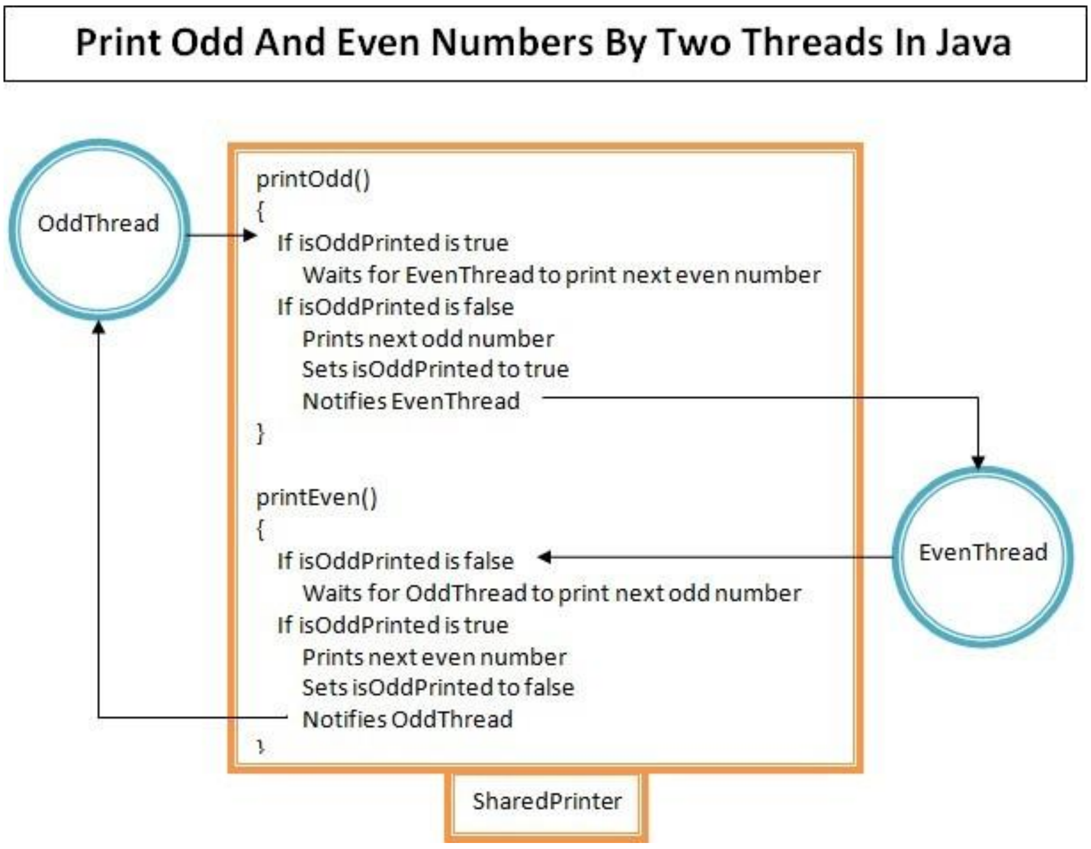
object.
2. By implementing Runnable interface : We create a new class which implements java.lang.Runnable interface and override run() method. Then we instantiate a Thread object and call start() method on this object.

Thread class:

Thread class provide constructors and methods to create and perform operations on a thread
Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

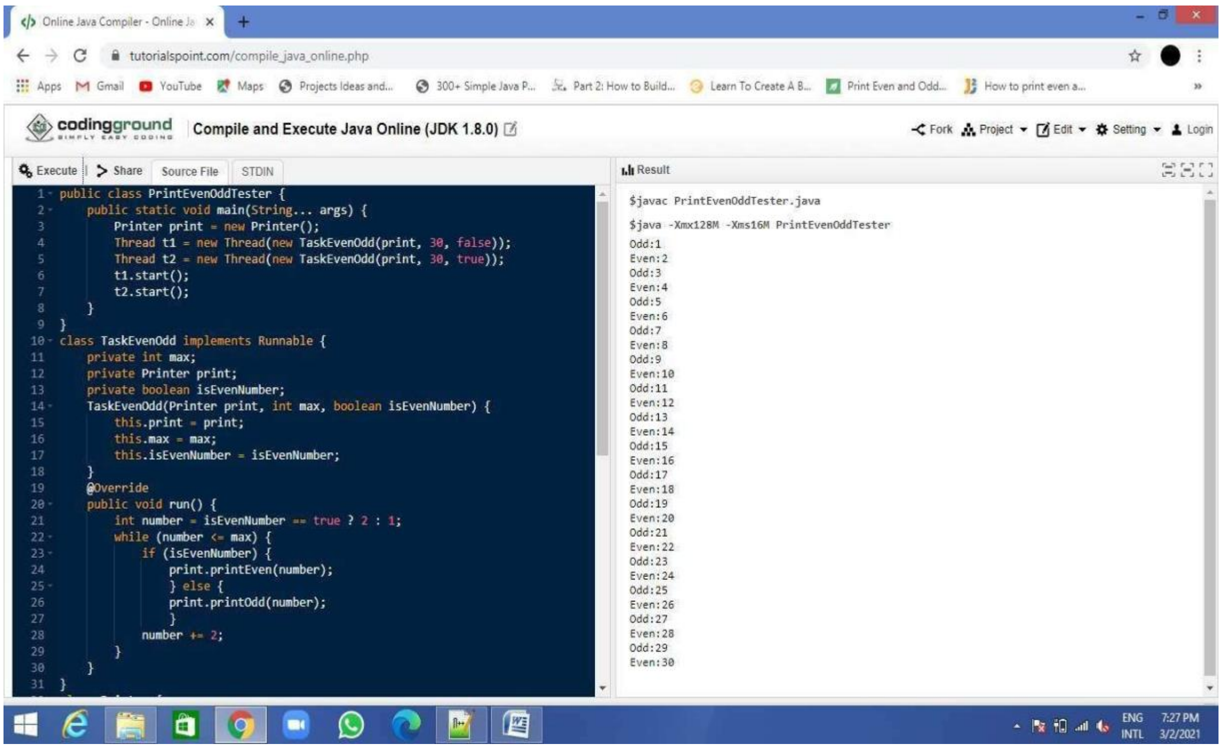


Prog

ram :

```
public class PrintEvenOddTester {

    public static void main(String... args) {
        Printer print = new Printer();
        Thread t1 = new Thread(new TaskEvenOdd(print, 30, false));
        Thread t2 = new Thread(new TaskEvenOdd(print, 30, true));
        t1.start();
        t2.start();  }
}
class TaskEvenOdd implements Runnable {
    private int max;
    private Printer print;
    private boolean isEvenNumber;
    TaskEvenOdd(Printer print, int max, boolean isEvenNumber) {
        this.print = print;
        this.max = max;
        this.isEvenNumber = isEvenNumber;
    }
    @Override
    public void run() {
        int number = isEvenNumber == true ? 2 : 1;
        while (number <= max) {
            if (isEvenNumber) {
                print.printEven(number);
                } else {
                print.printOdd(number);
                }
            number += 2;
        }
    }
}
class Printer {
    boolean isOdd = false;
    synchronized void printEven(int number) {
        while (isOdd == false) {
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("Even:" + number);
        isOdd = false;
        notifyAll();
    }
    synchronized void printOdd(int number) {
        while (isOdd == true) {
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("Odd:" + number);
        isOdd = true;
        notifyAll();
    }
}
```

<u>Input/output :</u>



Online Java Compiler - Online Ja
tutorialspoint.com/compile_java_online.php
codingground
Compile and Execute Java Online (JDK 1.8.0)
Fork
Project
Edit
Setting
Login
Execute
Share
Source File
STDIN
Result

```
public class PrintEvenOddTester {
    public static void main(String... args) {
        Printer print = new Printer();
        Thread t1 = new Thread(new TaskEvenOdd(print, 30, false));
        Thread t2 = new Thread(new TaskEvenOdd(print, 30, true));
        t1.start();
        t2.start();
    }
}
class TaskEvenOdd implements Runnable {
    private int max;
    private Printer print;
    private boolean isEvenNumber;
    TaskEvenOdd(Printer print, int max, boolean isEvenNumber) {
        this.print = print;
        this.max = max;
        this.isEvenNumber = isEvenNumber;
    }
    @Override
    public void run() {
        int number = isEvenNumber == true ? 2 : 1;
        while (number <= max) {
            if (isEvenNumber) {
                print.printEven(number);
                } else {
                print.printOdd(number);
                }
            number += 2;
        }
    }
}
```

```
$javac PrintEvenOddTester.java
$java -Xmx128M -Xms16M PrintEvenOddTester
Odd:1
Even:2
Odd:3
Even:4
Odd:5
Even:6
Odd:7
Even:8
Odd:9
Even:10
Odd:11
Even:12
Odd:13
Even:14
Odd:15
Even:16
Odd:17
Even:18
Odd:19
Even:20
Odd:21
Even:22
Odd:23
Even:24
Odd:25
Even:26
Odd:27
Even:28
Odd:29
Even:30
```

<u>Conclusion :</u> we had a look at how we can print odd and even numbers alternatively using two threads in Java.