**Term Work 1**

Tittle:

Write a program to demonstrate the implementation of a 2-dimension array.

Problem Definition:

Write a Java program to accept IA marks obtained by five students in three subjects. The program should accept marks obtained by each student and display the total marks and the average marks. The average marks are computed using a method as the average of best two marks obtained.

Introduction:

Basics – for 5 subject marks example –
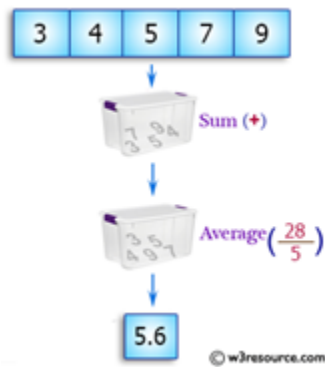
	Column 1	Column 2	Column 3	Column 4
Row 1	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 2	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 3	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Here is how we can initialize a 2-dimensional array in Java.

Algorithm:

1. Repeat For I = 1 to M

2.   Repeat For J = 1 to N

3.   Create a 2 – dimensional array

     [End of Step 2 For Loop]

     [End of Step 1 For Loop]

4. Exit

Implementation Steps:

Step 1: Create 2 – dimensional Array and store marks data and calculate total

Step 2: Compute average function

Step 3: Prompt the outputs to the screen – total and average – for 5 students and 3 subjects –

Observation and Output:

Marks:

	Max. Marks	Marks Scored
Conducting the experiment	5	
Conclusion and Outcome	5	
Viva Voce	5	
	15	
Checked on		
Signature of the faculty		

Term Work 2

Tittle:

Write a program to demonstrate the implementation of class and its member methods.

Problem Definition:

Design a class by name myTriangle to model a triangle geometrical object with three sides. Include functions to:

1. Initialize the three sides of the triangle.
2. Determine the type of triangle represented by the three sides (Equilateral/ Isosceles/ Scalene triangle).
3. Compute and return the area of the triangle.

Note:

When three sides are given we use the following formula:

s = (a+b+c) / 2;

area = sqrt(s*(s-a)*(s-b)*(s-c));

Introduction:

A triangle is a three-sided polygon, which has three vertices. The three sides are connected with each other end to end at a point, which forms the angles of the triangle. The sum of all three angles of the triangle is equal to 180 degrees.

Types of Triangle:

- Scalene Triangles
- Isosceles triangles

- Equilateral triangles

If all the three sides are different in length, then its scalene triangle.

If any two sides are equal in length, then it is an isosceles triangle.

If all three sides are equal in length, then it is an equilateral triangle.

Area is the region occupied by it in a two-dimensional space.

Area = ½ (Product of base and height of a triangle)

Sum of the Lengths of Two Sides of a Triangle:

A polygon is a simple closed figure made up of only line segments. A triangle is the smallest polygon formed by joining three-line segments. These three-line segments are called the sides of the triangle. Any three-line segments cannot form a triangle. A triangle can be drawn only if the sum of any two sides of the triangle is greater than the third side. This is stated as the triangle inequality.

Triangle Inequality
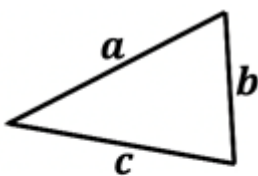
$$a + b > c$$
$$a + c > b$$
$$b + c > a$$

Algorithm:

Step 1: Get inputs – sides of a triangle a, b, c

Step 2: Testing triangle existence – sum of two sides must be greater than the third side.

Step 3: Calculation area of a triangle, Area = ½ X Base X Height

Observation and Output:

Marks:

	Max. Marks	Marks Scored
Conducting the experiment	5	
Conclusion and Outcome	5	
Viva Voce	5	
	15	
Checked on		
Signature of the faculty		

TERMWORK 3

Title: Write a program to demonstrate the implementation of parameterized:

      a.   Methods.

      b.   Constructor.

Problem Statement: A certain small bank intends to automate a few of its banking operations for its customers. Design a class by name mybankAccount to store the customer data having following details:

     1.accountNumber 2. acctType 3. Name 4. Address 5. accountBalance

The class must have both default and parameterized constructors. Write appropriate method to compute interest accrued on accountBalance based on accountType and time in years. Assume 5% for S/B account 6.5% for RD account and 7.65 for FD account. Further, add two methods withdrawAmount/depositAmount with amount as input to withdraw and deposit respectively. The withdrawAmount method must report in-sufficient balance if accountBalance falls below Rs. 1000.

Introduction:

Parameterised Methods:

Information can be passed to methods as parameters. Parameters act as variables inside the method.

Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

The below example has a method that takes a $String$ called fname as parameter. When the method is called, we pass along a first name, which is used inside the method to print the full name:

Example:

```java
public class Main {

  static void myMethod(String fname) {

    System.out.println(fname + " Refsnes");

  }


  public static void main(String[] args) {

    myMethod("Liam");

    myMethod("Jenny");

    myMethod("Anja");

  }
```

Parameterised constructors:

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

A constructor which has a specific number of parameters is called a parameterized constructor.

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

In the below example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
class Student4{
    int id;
    String name;
    //creating a parameterized constructor
    Student4(int i,String n){
    id = i;
    name = n;
    }
    //method to display the values
    void display(){System.out.println(id+" "+name);}
    public static void main(String args[]){
    //creating objects and passing values
```

```
Student4 s1 = new Student4(111,"Karan");

Student4 s2 = new Student4(222,"Aryan");

//calling method to display the values of object

s1.display();

s2.display();

}

}
```

Algorithm:

TASK 1 : Build the class with appropriate member variables, constructors and methods.

TASK 2 : Instantiate three objects of above type and perform different operations on the same.

TASK 3 : Write a function to display all the three customer details in a tabular form with appropriate column headings.

Step 1:Create a Bank account method and collect the name, address and other details from the user

Step 2:Create BankAcc constructor and assign the details

Step 3: Compute Interest by using following functions

Step 4: Calculate deposit and withdraw based on details

Observations and Output:

Marks:

	Max. Marks	Marks Scored
Conducting the experiment	5	
Conclusion and Outcome	5	
Viva Voce	5	
	15	
Checked on		
Signature of the faculty		

TERMWORK- 4

Title: Write a program to demonstrate the implementation of inheritance.

Problem Statement:

A company has two types of employees – Full Time and Par time. The company records for each employee his/her name, age, address, salary and gender. Given the basic salary of the Full Time employee the components of his/her gross salary are: Dearness allowance – 75% of basic salary, HRA – 7.5% of basic salary, IT – 10% of basic. The salary of a Part time employee is dependent on the qualification, experience, number of working hours and the rate per hour, as below:

	Qualification		
Experience	BE	MTech	Ph.D
1-5 years	300 Rs.	500 Rs.	800 Rs.
6-10 years	400 Rs.	700 Rs.	1200 Rs.
>10 years	500 Rs.	1000 Rs.	1500 Rs.

Introduction:

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Terms used in Inheritance

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

When a class inherits another class, it is known as a *single inheritance*. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

```
class Animal{

    void eat(){System.out.println("eating...");}

}
class Dog extends Animal{

void bark(){System.out.println("barking...");}

}
class TestInheritance{

public static void main(String args[]){

Dog d=new Dog();

d.bark();

d.eat();
```

```
}}
```

Algorithm:

Model this as a problem of hierarchical inheritance by:

1) Identifying the super class with its data members and member functions.

2) Identify the sub-class/sub-classes and their associated data members and member functions.

Test the program by creating objects of the classes that are so identified.

Steps to be followed in programming are:

Step 1: Create a class Employee with the following details

Step 2: Create a subclass FTEmployee with Employee as a base class

Step 3: Create one more sublass PTEmployee with the base class Employee

Step 4: Create a main class and create objects using FTEmployee and PTEmployee derived classes

Observations and Output:
Marks:

	Max. Marks	Marks Scored
Conducting the experiment	5	
Conclusion and Outcome	5	
Viva Voce	5	

	15	
Checked on		
Signature of the faculty		

TERMWORK 5a

Title: Write a program to demonstrate the implementation of method overloading.

Problem Definition: Create a Stack class having an integer array say elem and top_of_stack as instance variables. Define three overloaded methods having the following signatures:
a. initStack(int size) to create an array of specified size and initialize the top_of_stack
b. initStack(Stack another) to initialize the Stack object with state of the Stack object "another"
c. initStack(int [] a) to initialize contents of a[] to the instance variable elem.

Write following methods:
a. push(): Pushes the element onto the stack,
b. pop(): Returns the element on the top of the stack, removing it in the process, and
c. peek(): Returns the element on the top of the stack, but does not remove it.

Also write methods that check whether stack is full and stack is empty and return boolean value true or false appropriately.

Introduction:

Method Overloading in Java
Method overloading in java is a feature that allows a class to have more than one method with the same name, but with different parameters.

Java supports method overloading through two mechanisms:
1.By changing the number of parameters
2.By changing the data type of parameters Overloading by changing the number of parameters A method can be overloaded by changing the number of parameters.

What is Method overloading in Java?

"Method overloading is a feature of Java in which a class has more than one method of the same name and their parameters are different."

In other words, we can say that Method overloading is a concept of Java in which we can create multiple methods of the same name in the same class, and all methods work in different ways. When more than one method of the same name is created in a Class, this type of method is called the Overloaded Method.

We can easily understand about method of overloading by the below example:
Suppose we have to write a method to find the square of an integer number. We can write this method as follows:

```
public void intSquare ( int number )
{
 int square = number * number;
 System.out.printIn("Method with Integer Argument Called:"+square);
}
public void doubleSquare(double number)
{
 double square = number * number;
 System.out.printIn("Method with double Argument Called:"+square);
}

public void longSquare(long number)
{
long square = number * number;
System.out.printIn("Method with long Argument Called:"+square);
}
```

If we look carefully, to find the square of a number only, according to the data type of the number, we have to take three names as follows:

```
intSquare()
doubleSquare()
longSquare()
```

If it is possible that a programmer has to take only one name and the program itself decides which method to use for which type of value, then it will be easier for the programmer to get the same. There is no need to memorize the names of more than one method for type work. In Java, we can give the above three methods the same name.

If we provide only the square () name instead of giving different names to the above three methods and write the rest of the description as follows, then Java's Compiler does not generate any error.

```
public void Square ( int number )
{
```

```java
        int square = number * number;
        System.out.println("Method with Integer Argument Called:"+square);
 }
public void Square(double number)
 {
        double square = number * number;
        System.out.println("Method with double Argument Called:"+square);
}
public void Square(long number)
 {
        long square = number * number;
        System.out.println("Method with long Argument Called:"+square);
}
```

If we define these three methods in a class, then these methods can be called Overloaded Methods as they have the same name. Let us develop CalculateSquare Class to understand this, which is as follows:

```java
class CalculateSquare
 {
        public void square()
         {
                System.out.println("No Parameter Method Called");
         }
        public int square( int number )
        {
                int square = number * number;
                System.out.println("Method with Integer Argument Called:"+square);
        }
        public float square( float number ){
                 float square = number * number;
                 System.out.println("Method with float Argument Called:"+square);
        }
        public static void main(String[] args) {
                CalculateSquare obj = new CalculateSquare();
                obj.square();
                obj.square(5);
                obj.square(2.5);
         }
 }
```

Note: We have not provided any argument in the 'parenthesis' of the square() method in our program. In this case, the Compiler Class calls the method in which no Parameter has been defined to achieve the Argument.

Output:

No Parameter Method Called
Method with Integer Argument Called: 25
Method with float Argument Called: 6.25

In this way, we can define more than one Methods of the same name in a class, which is called Method Overloading, and

The Java compiler itself performs the appropriate Method Call for an object, based on the Data Type of the Arguments of the Methods.

Benefits of using Method Overloading

Method overloading increases the readability of the program.
1. This provides flexibility to programmers so that they can call the same method for different types of data.
2. This makes the code look clean.
3. This reduces the execution time because the binding is done in compilation time itself.
4. Method overloading minimizes the complexity of the code.
5. With this, we can use the code again, which saves memory.

Program:

```
class Sample{
    int disp(int x){
         return x;
    }
    double disp(int y){
         return y;
    }
    public static void main(String args[])
    {
    Sample s = new Sample();
    System.out.printIn("Value of x : " + s.disp(5));
    System.out.printIn("Value of y : " + s.disp(6.5));
    }
}
```

Output:
Sample.java:6: error: method disp(int) is already defined in class SampleAlgorithm:

Algorithm:

Step 1: Create a Stack class with
Step 2: Declare 2 member variables: ele(int array) and top(int).
Step 3: Create member functions

Function INITSTACK(int size)
Step 1: Create an int array of length size
Step 2: Initialize top = -1

Function INITSTACK(Stack another)
Step 1: Create an int array(ele) of size "length"(of object another)
Step 2: Initialize top = -1

Step 3: Call PUSH to copy the items of "ele" to the array of "another".

Function INITSTACK(Array a)
Step 1: Create an int array(ele) of length of "a"
Step 2: Initialize top = -1
Step 3: Call PUSH to copy the items of "a" to the array of "another".

Function PUSH(Element) Operation in Stack
Step 1: If Top=Max-1 , Display "Overflow : Stack is full" and Exit
Step 2: Top=Top+1
Step 3: Stack[TOP]=Element

Function POP()
Step 1: If TOP=-1, Display "Underflow: Stack is empty" and Exit
Step 2: Initialize delElement=ele[Top]
Step 3: Top=Top-1
Step 4: Return delElement

Function PEEK()
Step 1: Return ele[Top]

Function Main()
Step 1: Create 3 objects(s1,s2,s3) of the Stack Class
Step 2: Call PUSH to push an item into s1
Step 3: Call INITSTACK(s1) on object s2
Step 4: Create an int array "array" with elements initialized
Step 5: Call INITSTACK(array)
Step 6:Define a integer variable 'top' and initialize with '-1'. (int top = -1)
Step 7:Call pop and peek functions and display the value returned(element on top of stack)

Observations and output:
Marks:

	Max. Marks	Marks Scored
Conducting the experiment	5	
Calculation, Graph, Results, Conclusion and Outcome	5	

Viva Voce	**5**	
	15	
Checked on		
Signature of the faculty		

TERMWORK 5b

Title: Write a program to demonstrate the implementation of method overriding

DeImplement the following class hierarchy. In the Cuboid class, override the method computeArea() and computePerimeter() of Rectangle class to compute the surface area and perimeter of a rectangle cuboid. Add a method computeVolume() in Cuboid class to compute volume of the cuboid. Assuming length, width and height as l, w and h respectively,

formula to find the surface area = 2(lw) + 2(hl) + 2(hw)
formula to find the perimeter = 2l + 2w
formula to find the volume = l x w x h

Output and Observations

Marks:

	Max. Marks	Marks Scored
Conducting the experiment	5	
Conclusion and Outcome	5	
Viva Voce	5	

	15	
Checked on		
Signature of the faculty		

TERMWORK 6

Title: Write a program to demonstrate Run-time Polymorphism.

Problem definition: Design an abstract class Car to have carName, chassiNum, modelName as member variables and add two abstract methods, startCar and operateSteering . Inherit MarutiCar and BmwCar from Car class and override the two abstract methods in their own unique way. Design a driver class to have driver name, gender and age as data members and add a method driveCar with abstract class reference variable as argument and invoke the two basic operations namely, startCar and operateStearing and demonstrate run-time polymorphism.

Introduction:

Runtime polymorphism in Java is also popularly known as Dynamic Binding or Dynamic Method Dispatch. In this process, the call to an overridden method is resolved dynamically at runtime rather than at compile-time. You can achieve Runtime polymorphism via Method Overriding.

Method Overriding is done when a child or a subclass has a method with the same name, parameters, and return type as the parent or the superclass; then that function overrides the function in the superclass. In simpler terms, if the subclass provides its definition to a method already present in the superclass; then that function in the base class is said to be overridden.

Also, it should be noted that runtime polymorphism can only be achieved through functions and not data members.

Overriding is done by using a reference variable of the superclass. The method to be called is determined based on the object which is being referred to by the reference variable. This is also known as Upcasting.

Upcasting takes place when the Parent class's reference variable refers to the object of the child class. For example:

class A{}
class B extends A{}
A a=new B(); //upcasting

Example of run-time polymorphism in java

We will create two classes Car and Innova, Innova class will extend the car class and will override its run() method.

```
class Car
{
        void run()
        {
        System.out.println(" running");
        }
}
class innova extends Car
{
        void run();
        {
                System.out.println(" running fast at 120km");
        }
        public static void main(String args[])
        {
                Car c = new innova();
                c.run();
        }
}
```

Algorithm:

Car class
Step 1:Create abstract class Car
Step 2: In the constructor initialize the name, chassi and model of the car
Step 3: Declare 2 abstract functions(startCar() and operateSteering())
Step 4: Display the name, chassi and model of the car.

MarutiCar class
Step 1: Create class MarutiCar (inherits Car)
Step 2: In the constructor initialize the name, chassi and model of the car by calling the super class
        constructor
Step 3: function startCar() display a message

Step 4: function operateSteering() display a message

BmwCar class
Step 1: Create class BmwCar (inherits Car)
Step 2: In the constructor initialize the name, chassi and model of the car by calling the super class
         constructor
Step 3: function startCar() displays a message
Step 4: function operateSteering() displays a message

Driver class
Step 1:Create class Driver
Step 2: In the constructor initialize the name, age and gender
Step 3: function driveCar(Car obj) displays name, age and gender
Step 4: Call display(), startCar() and operateSteering() with object obj

Main class
Step 1: Create class Driver
Step 2: Create an object "m" of MarutiCar and initialize
Step 3: Create an object "b" of BmwCar and initialize
Step 4: Create an object "dl" of Driver and initialize
Step 5: Call driveCar(object m)
Step 6: Call driveCar(object b)

Output and Observations:

Marks:

	Max. Marks	Marks Scored
Conducting the experiment	5	
Conclusion and Outcome	5	
Viva Voce	5	
	15	

Checked on		
Signature of the faculty		

TERMWORK 7

Title: Write a program to demonstrate the implementation of interfaces.

Problem Definition: Write a Java application to implement the following UML diagram.

- PrimeTester class implements isPrime() method by iterating from 2 to n-1 for a given number n
- ImprPrimeTester class implements isPrime() method by iterating from 2 to n/2
- FasterPrimeTester class implements isPrime() method by iterating from 2 to
- FastestPrimeTester class implements isPrime() method using Fermat's Little theorem.
  - Fermat's Little Theorem:
  - If n is a prime number, then for every a, $1 < a < n-1$,
    - $a^{n-1} \% n = 1$
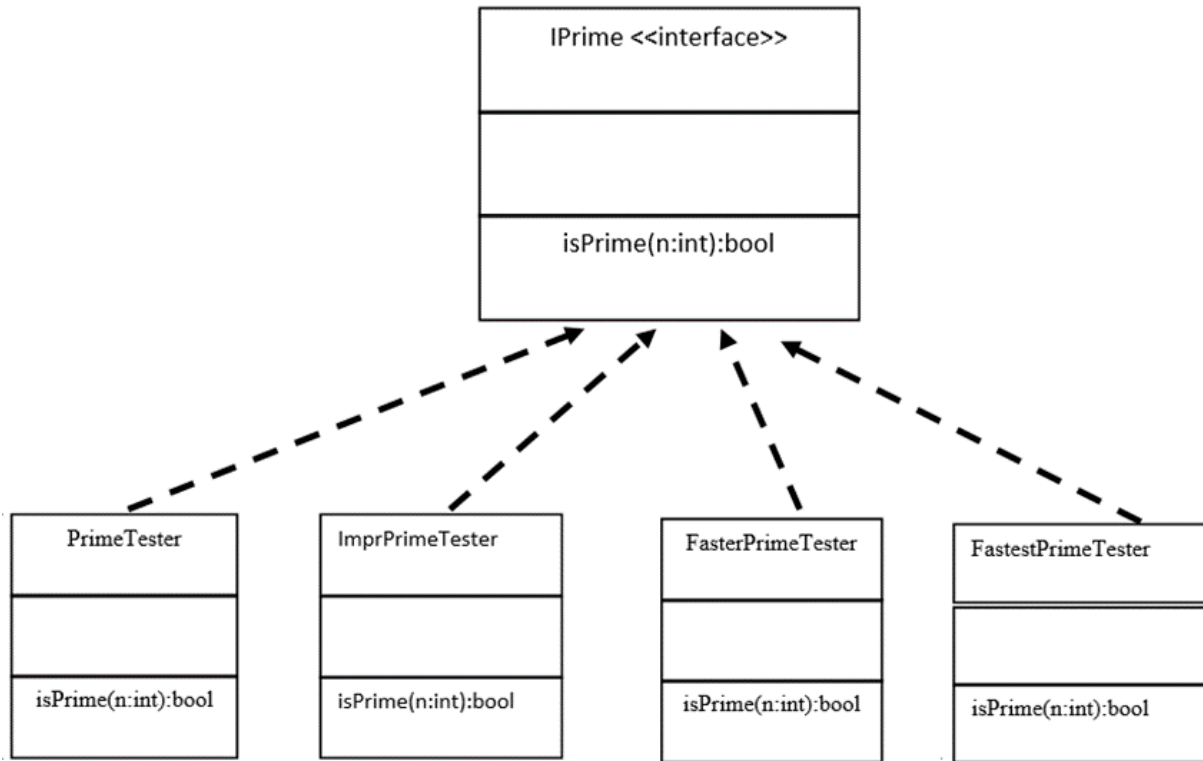
Introduction:

An interface is an abstract "class" that is used to group related methods with "empty" bodies:

To access the interface methods, the interface must be "implemented" (kind of like inherited) by another class with the implements keyword (instead of extends). The body of the interface method is provided by the "implement" class

Example:

The implements keyword is used to implement an interface.

- It **cannot** be used to create objects (in the example above, it is not possible to create an "Animal" object in the MyMainClass)
- Interface methods does not have a body – the body is provided by the "implement" class

- On implementation of an interface, you must override all of its methods
- Interface methods are by default abstract and public
- Interface attributes are by default public, static and final
- An interface cannot contain a constructor (as it cannot be used to create objects)

Algorithm:

1. Create an interface IsPrime and declare isPrime(int n) method without its body

2. Create a class PrimeTester which implements IsPrime interface

3. Override the method isPrime(int n) declared in IsPrime interface and check whether a number is prime or not using the formula given for PrimeTester class

4. Create a class ImprPrimeTester which implements IsPrime interface

5. Override the method isPrime(int n) declared in IsPrime interface and check whether a number is prime or not using the formula given for ImprPrimeTester class

6. Create a class FasterPrimeTester which implements IsPrime interface

7. Override the method isPrime(int n) declared in IsPrime interface and check whether a number is prime or not using the formula given for FasterPrimeTester class

8. Create a class FastestPrimeTester which implements IsPrime interface

9. Override the method isPrime(int n) declared in IsPrime interface and check whether a number is prime or not using the formula given for FastestPrimeTester class

10. Create a driver class and define main method

11. Create objects of the PrimeTester, ImprPrimeTester, FasterPrimeTester and FastestPrimeTester classes to initiate isPrime(int n) method and display the result

Observations and Output:

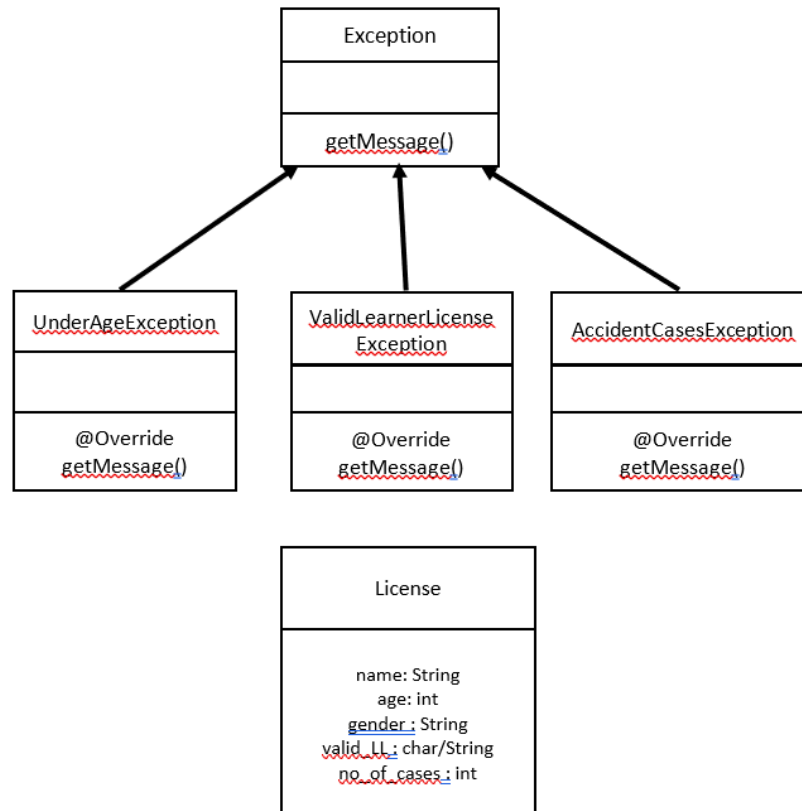**Marks:**

	Max. Marks	Marks Scored
Conducting the experiment	5	
Conclusion and Outcome	5	
Viva Voce	5	
	15	
Checked on		
Signature of the faculty		

TERMWORK 8

Title: Write a program to demonstrate the implementation of customized exception handling.

Problem Definition: Assume that you have received a request from the transport authority for automating the task of issuing the permanent license for two wheelers. The mandatory condition to issue the license are: 1) the applicant must be over 18 years of age and 2) holder of a valid learner's license and 3) no accident cases in the last one year.

Write a Java program that reads user details as required (use the Scanner class). Create user defined exceptions to check for the three conditions imposed by the transport authority. Based on the inputs entered by the user, decide and display whether or not a license has to be issued or an error message as defined by the user exception.

```
           ┌─────────────────────┐
           │      Exception      │
           ├─────────────────────┤
           │                     │
           ├─────────────────────┤
           │    getMessage()     │
           └─────────────────────┘
```

┌──────────────────────┐   ┌──────────────────────┐   ┌──────────────────────┐
│   UnderAgeException   │   │ ValidLearnerLicense │   │ AccidentCasesException│
│   │   │   Exception   │   │   │
├──────────────────────┤   ├──────────────────────┤   ├──────────────────────┤
│   │   │   │   │   │
├──────────────────────┤   ├──────────────────────┤   ├──────────────────────┤
│   @Override   │   │   @Override   │   │   @Override   │
│   getMessage()   │   │   getMessage()   │   │   getMessage()   │
└──────────────────────┘   └──────────────────────┘   └──────────────────────┘

┌──────────────────────┐
│   License   │
├──────────────────────┤
│   │
│   name: String   │
│   age: int   │
│   gender : String   │
│ valid_LL : char/String│
│   no_of_cases : int   │
└──────────────────────┘

Introduction:

In Java, we can create our own exceptions that are derived classes of the Exception class. Creating our own Exception is known as a custom exception or user-defined exception. Basically, Java custom exceptions are used to customize the exception according to user need.

Consider the example 1 in which InvalidAgeException class extends the Exception class.

Using the custom exception, we can have our own exception and message. Here, we have passed a string to the constructor of superclass i.e. Exception class that can be obtained using getMessage() method on the object we have created**.**

```java
// class representing custom exception
class InvalidAgeException  extends Exception
{
    public InvalidAgeException (String str)
    {
        // calling the constructor of parent Exception
        super(str);
    }
}
```

```java
// class that uses custom exception InvalidAgeException
public class TestCustomException1
{

    // method to check the age
    static void validate (int age) throws InvalidAgeException{
        if(age < 18){

        // throw an object of user defined exception
        throw new InvalidAgeException("age is not valid to vote");
    }
        else {
        System.out.println("welcome to vote");
        }
    }
```

```java
// main method
public static void main(String args[])
{
    try
    {
        // calling the method
        validate(13);
    }
    catch (InvalidAgeException ex)
    {
        System.out.println("Caught the exception");

        // printing the message from InvalidAgeException object
        System.out.println("Exception occured: " + ex);
    }

    System.out.println("rest of the code...");
}
}
```

In the above code, constructor of InvalidAgeException takes a string as an argument. This string is passed to the constructor of parent class Exception using the super() method. Also the constructor of Exception class can be called without using a parameter and calling super() method is not mandatory.

Algorithm:

1. Create a License class

2. Read the input for a person's name, age, gender, validLLR, no_of_accidents in past one year

3. Create a driver class and define main method and initiate validateApplicant() method

4. In validateApplicant(applicant) method,

5. Create a class UnderAgeException which extends Exception class

6. Create a class ValidLLRException which extends Exception class

7. Create a class NumAccidentsException which extends Exception class

Observations and Output:

Marks:

	Max. Marks	Marks Scored
Conducting the experiment	5	
Conclusion and Outcome	5	
Viva Voce	5	
	15	
Checked on		
Signature of the faculty		

TERMWORK 9

Title: Write a program to demonstrate the implementation of string handling.

Problem Definition: 9.1) Read a string containing 3_4 words using Scanner class object. Split it into words and for each word check if its palindrome by writing a function isPalindrome(String the myWord, int s, int e) which return true if its palindrome else return false. Where s is start index and e is end index of the input myWord. Print it in uppercase if it is palindrome else reverse the string and print it in lowercase. Use appropriate string functions to implement the above problem statement.

Introduction:

Scanner Class in Java

Scanner is a class in java.util package used for obtaining the input of the primitive types like int, double, etc. and strings. It is the easiest way to read input in a Java program, though not very efficient if you want an input method for scenarios where time is a constraint like in competitive programming.

- To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream. We may pass an object of class File if we want to read input from a file.
- To read numerical values of a certain data type XYZ, the function to use is nextXYZ(). For example, to read a value of type short, we can use nextShort()
- To read strings, we use nextLine().
- To read a single character, we use next().charAt(0). next() function returns the next token/word in the input as a string and charAt(0) function returns the first character in that string.
- The Scanner class reads an entire line and divides the line into tokens. Tokens are small elements that have some meaning to the Java compiler. For example, suppose there is an input string: How are you In this case, the scanner object will read the entire line and divides the string into tokens: "How", "are" and "you". The object then iterates over each token and reads each token using its different methods.

  How to use scanner class for strings?

Java's Scanner String input method

1. Import java.util.*; to make Java's Scanner class available.
2. Use the new keyword to create an instance of the Scanner class.
3. Pass the static System.in object to the Scanner's constructor.

4. Use Scanner's next() method to take input one String at a time.

isLowerCase( ): The method determines whether the specified char value is lowercase.

Syntax : boolean isLowerCase(char ch)

isUpperCase() Method: This method determines whether the specified char value is uppercase.

Syntax: boolean isUpperCase(char ch)

Palindrome String algorithm

1. Start
2. Get the String from user.
3. Hold the String in temporary variable.
4. Reverse the String.
5. Compare the temporary String with reversed String.
6. If both Strings are same
7. Print it in uppercase (if it is palindrome) else reverse the string and print it in lowercase. (make use of inbuilt function for uppercase and lowercase conversion)
8. Stop

Output and Observations:

Marks:

	Max. Marks	Marks Scored
Conducting the experiment	5	
Conclusion and Outcome	5	

Viva Voce	**5**	
	15	
Checked on		
Signature of the faculty		

<div align="center">**TERMWORK10**</div>

Title: 10. Write a program to demonstrate the implementation of JAVA swings.

Problem Definition: Design and develop a GUI application as shown below. Assume the two numbers to be integers. The application must check for invalid division condition and throw an appropriate exception.
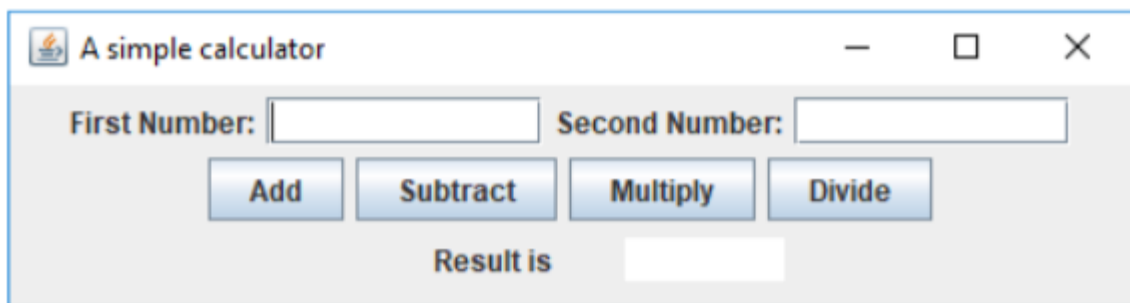
Introduction:

Swing in Java is a Graphical User Interface (GUI) toolkit that includes the GUI components. Swing provides a rich set of widgets and packages to make sophisticated GUI components for Java applications. Swing is a part of Java Foundation Classes(JFC), which is an API for Java GUI programming that provides GUI.

The Java Swing library is built on top of the Java Abstract Widget Toolkit (**AWT**), an older, platform dependent GUI toolkit. You can use the Java simple GUI programming components like button, textbox, etc., from the library and do not have to create the components from scratch.

GUI (Graphical User Interface) in Java is an easy-to-use visual experience builder for Java applications. It is mainly made of graphical components like buttons, labels, windows, etc. through which the user can interact with an application. GUI plays an important role to build easy interfaces for Java applications.

Algorithm

1. Start

2. Read two numbers

3. *Divide* first *number* by *second number*

4. Inside the try block check the condition if(number1/number2) produce valid   positi
then print it otherwise throw the exception

5. Catch the exception and display the appropriate message.

6. Stop

Output and Observations:

Marks:

	Max. Marks	Marks Scored
Conducting the experiment	5	
Conclusion and Outcome	5	
Viva Voce	5	
	15	
Checked on		
Signature of the faculty		