Unit 2

Experiment Number : 8 **Date: 20/02/2026**

Implementation of Packages in Java

Aim:

Write a Java program to implement built-in, user-defined packages and accessing all classes in a package.

PRE LAB EXERCISE

QUESTIONS

1. What is java.util package and what collection framework does it contain?
   The java.util package provides utility classes for data structures, date/time, and more.
   It contains the Java Collection Framework (JCF) for storing and manipulating data.
   JCF includes interfaces like List, Set, Queue and classes like ArrayList, HashMap.
   It makes programming easier by offering reusable data structures and algorithms.
2. What are the two types of packages in Java?

   Java has built-in packages provided by the language, like java.util and java.io.
   It also has user-defined packages created by programmers for organizing code.
   Packages help avoid naming conflicts and improve code modularity.
   They make code easier to maintain and reuse across programs.

IN LAB EXERCISE

Objective

To understand and implement the concepts of built-in, user-defined packages and accessing all classes in a package in Java.

Built-in Packages comprise a large number of classes that are part of the Java API. Some of the commonly used built-in packages are:

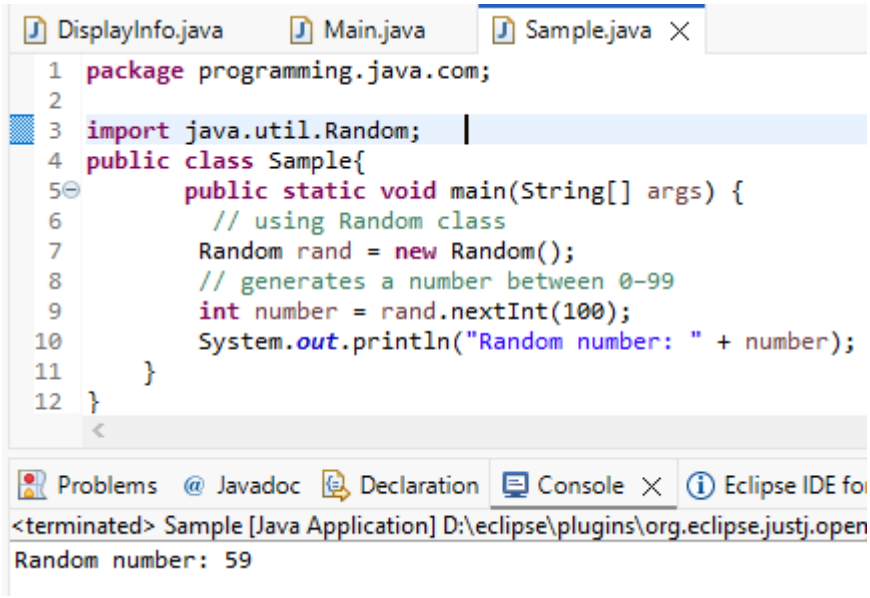
- java.lang: Contains language support classes(e.g, classes that define primitive data types, math operations). This package is automatically imported.
- java.io: Contains classes for supporting input/output operations.
- java.util: Contains utility classes that implement data structures such as Linked Lists and Dictionaries, as well as support for date and time operations.
- java.applet: Contains classes for creating Applets.
- java.awt: Contains classes for implementing the components for graphical user interfaces (like buttons, menus, etc).

Source Code

```
import java.util.Random;   // built-in package
public class Sample{
      public static void main(String[] args) {
       // using Random class
       Random rand = new Random();
       // generates a number between 0–99
       int number = rand.nextInt(100);
       System.out.println("Random number: " + number);
   }
}
```

Output

Random number: 59



User-defined Packages are the packages that are defined by the user.

Source code

```
package com.myapp;
public class Helper {
   public static void show() {
       System.out.println("Hello from Helper!");
```

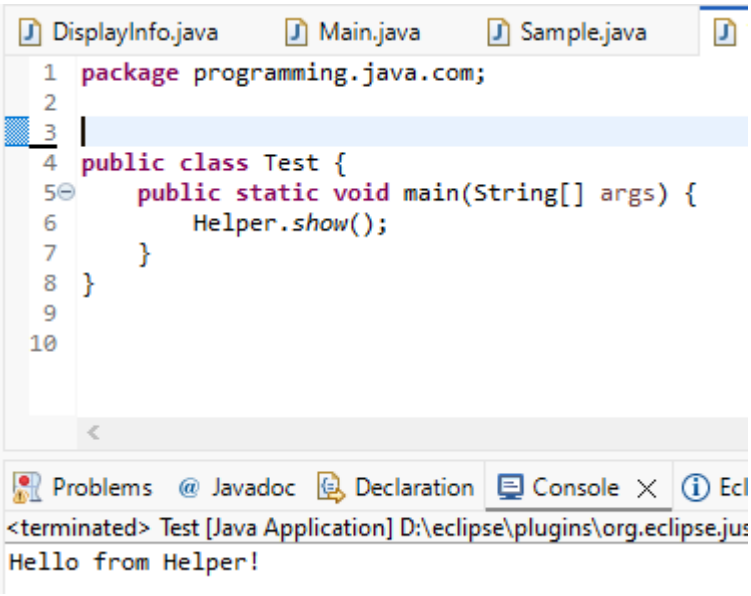
```
    }
}
```

==To use this in another class==

```
import com.myapp.Helper;
public class Test {
    public static void main(String[] args) {
        Helper.show();
    }
}
```

Output:

Hello from Helper!



//Importing all classes from a package.

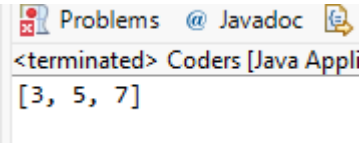
Source code

```
import java.util.Vector;
public class Coders {
    public Coders() {
        // java.util.Vector is imported, We are able to access it directly in our code.
      Vector v = new Vector();
      java.util.ArrayList l = new java.util.ArrayList();
```

```
        l.add(3);
        l.add(5);
        l.add(7);
        System.out.println(l);
    }
    public static void main(String[] args) {
      new Coders();
    }
}
```

Output

[3,5,7]



POST LAB EXERCISE

1. What will happen if two classes in different packages have the same name and are imported in a Java file?
   The compiler will get confused if both classes are imported using * and have the same name.This will cause a naming conflict and a compilation error.
   To resolve it, you must use the fully qualified class name with the package.
   Example: java.util.Date d; java.sql.Date s;

2. What is the purpose of using packages in Java?
   Packages organize classes and interfaces into namespaces for better structure.
   They avoid naming conflicts between classes with the same name.
   Packages make code modular, reusable, and easier to maintain.
   They also provide access control with public, private, and protected modifiers.

3. Which built-in Java package would you use if you want to create a GUI window and display a message?
   A. java.util
   B. java.sql
   C. java.awt
   D. java.net

   C. java.awt
   The java.awt package provides classes for creating GUI components like windows, buttons, and dialogs.

ASSESSMENT

Description	Max Marks	Marks Awarded
Pre Lab Exercise	**5**	
In Lab Exercise	**10**	
Post Lab Exercise	**5**	
Viva	**10**	
Total	**30**	
Faculty Signature		

Experiment Number : 9 **Date:23/02/2026**

Implementation of a Java Program to import packages using different methods

Aim:

Write a Java program to import packages using different methods for different use cases.

PRE LAB EXERCISE

QUESTIONS

1. How to import a single class and multiple classes from a package in Java?

To import a single class:

import packageName.ClassName;

To import multiple classes from a package:

import packageName.*;

Using * imports all classes in the package but not sub-packages.

2. Which package is always imported by default in every Java class?

- The java.lang package is imported by default in every Java program.
- It contains core classes like String, Math, Object, Integer, etc.
- No explicit import statement is needed for java.lang.

IN LAB EXERCISE

Objective

To understand and implement the Java packages using different methods and import them.

Problem

Define a package named ‘useFul’ with a class names ‘UseMe’ having following methods:

1) area()- To calculate the area of given shape.

2) salary()- To calculate the salary given basic Salary,da,hRA.

3) percentage()-To calculate the percentage given total marks and marks obtained.

4) Develop a program named ‘Package Use’ to import the above package ‘useFul’ and use the method area().

5) Develop a program named ‘manager’

Source Code

```
//Package Creation:

package useFull;

import java.util.*;

public class UseMe

{

        Scanner obj=new Scanner(System.in);

public static void area()

      {

          class method{

           void aos(int a)

           {

             System.out.print("\nArea of square with length "+a+" is "+(a*a));

           }

          void aor(int a,int b)

          {

             System.out.print("\nArea of reactangle with dimensions "+a+" & "+b+" is
"+(a*b));

           }

           void aoc(int r)

          {

            double a=3.14*r*r;

           }

          System.out.print("\nArea of circle with radius "+r+" is "+a);

        }

        void aot(int a,int b)

         {

          float ar=(a*b)/2;
```

```
        System.out.print("\nArea of triangle with dimensions "+a+" &"+b+" is "+ar);
      } }
    Scanner obj=new Scanner(System.in);
    method m=new method();
    System.out.print("\n1.Square\n2.Rectangle\n3.Circle\n4.Triangle\nSelect the shape\n");
    int ch=obj.nextInt();
    UseMe u=new UseMe();
    switch(ch)
            {
            case 1:System.out.print("\nEnter the length of side of square : ");
                    int s=obj.nextInt();m.aos(s);
                    break;
            case 2:System.out.print("\nEnter the dimensions of rectangle : ");
                    int l=obj.nextInt();
                    int b=obj.nextInt();
                    m.aor(l,b);
                    break;
            case 3:System.out.print("\nEnter the radius of circle : ");
                    int r=obj.nextInt();
                    m.aoc(r);
                    break;
            case 4:System.out.print("\nEnter the dimensions of triangle : ");
                    int ba=obj.nextInt();
                    int w=obj.nextInt();
                    m.aot(ba,w);
                    break; } }
public void salary()
  {
    int ba,da,hra;
    System.out.print("\nEnter the basic salary : ");
```

```
        ba=obj.nextInt();
        System.out.print("\nEnter the dearness allowance :");
        da=obj.nextInt();
        System.out.print("\nEnter the house rent allowance : ");
        hra=obj.nextInt();
        System.out.print("\nThe total Gross salary of employee is : "+(ba+da+hra));
    }
public void percentage()
    {
        int n,sum=0;
        float p;
        System.out.print("\nEnter the total number of subjects : ");
        n=obj.nextInt();
        int m[]=new int[n];
        System.out.print("\nEnter the marks of "+n+" subjects : ");
        for(int i=0;i<n;i++)
        {
            m[n]=obj.nextInt();
        }
        for(int i=0;i<n;i++)
        {
             sum=sum+m[i];
        }
        p=sum/n;
        {
                System.out.print("\nPercentahe of student : "+p);
        }
     }
}
//Package Implementation-1:
```

```
import useFull.UseMe;
class packageUse
{
        public static void main(String args[])
        {
        UseMe o=new UseMe();o.area();
        }
}
```

Output

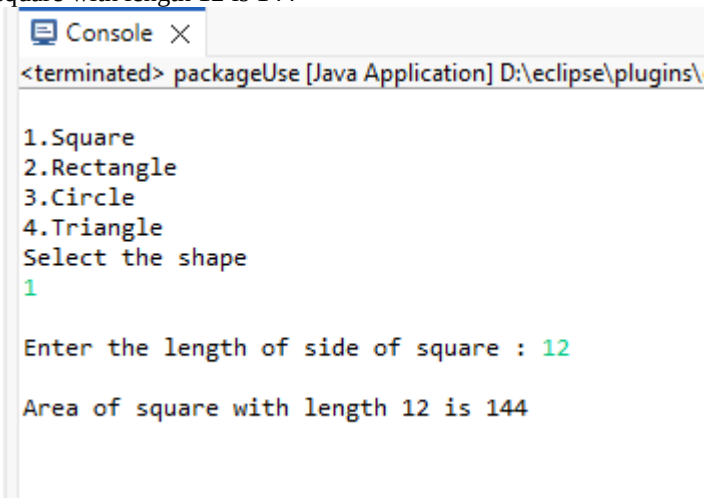
javac packageUse.java

java packageUse

```
1.Square
2.Rectangle
3.Circle
4.Triangle
Select the shape
1

Enter the length of side of square : 12

Area of square with length 12 is 144
```



//Package Implementation-2:

```
import useFull.UseMe;
class manager
{
   public static void main(String args[])
```

```
    {
     UseMe obj=new UseMe();obj.salary();
     }
}
```

Output

javac manager.java

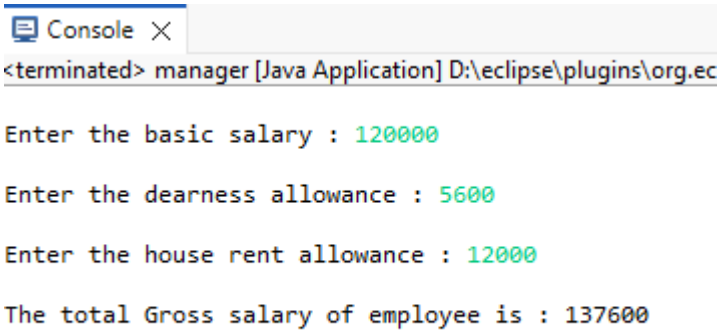
java manager

Enter the basic salary: 120000

Enter the dearness allowance: 5600

Enter the house rent allowance: 12000

The total Gross salary of employee is: 137600



POST LAB EXERCISE

1. Find the key differences between java.util and java.lang packages.
   The java.lang package is automatically imported in every Java program, while java.util must be imported manually. java.lang contains fundamental classes like String, Math, Object, and Integer. On the other hand, java.util provides utility classes such as ArrayList, HashMap, Scanner, and date/time classes. java.lang supports core language features, whereas java.util is mainly used for collections, data structures, and utility functions.

2. List some of the subpackages of java.util.
   The java.util package has several subpackages, including java.util.concurrent for multithreading and concurrency, java.util.regex for regular expressions, java.util.zip for data compression, java.util.spi for service provider interfaces, and java.util.logging for logging purposes.

ASSESSMENT

Description	Max Marks	Marks Awarded
Pre Lab Exercise	**5**	
In Lab Exercise	**10**	
Post Lab Exercise	**5**	
Viva	**10**	
Total	**30**	
Faculty Signature		

Experiment Number : 10 **Date: 25/02/2026**

Implementation of Access Modifiers in Java

Aim:

Write a Java program to implement different access modifiers.

PRE LAB EXERCISE QUESTIONS

1. What are the 4 access modifiers available in Java?
   - **Public**
   - **Private**
   - **Protected**
   - **Default**

2. Which access modifiers can be used in Java to control access to class members?
   The **public** modifier allows members to be accessed from anywhere in the program.
   The **private** modifier restricts access to within the same class only.
   The **protected** modifier allows access within the same package and to subclasses in other packages.
   The **default** modifier (when no keyword is specified) allows access only within the same package.

IN LAB EXERCISE

Objective

To demonstrate different access modifiers such as Default, Private, Protected and Public using Java programs.

	Default	Private	Protected	Public
Same Class	Yes	Yes	Yes	Yes
Same Package Subclass	Yes	No	Yes	Yes
Same Package Non-Subclass	Yes	No	Yes	Yes
Different Package Subclass	No	No	Yes	Yes
Different Package Non-Subclass	No	No	No	Yes

Fig: Comparison table of Access Modifiers in Java

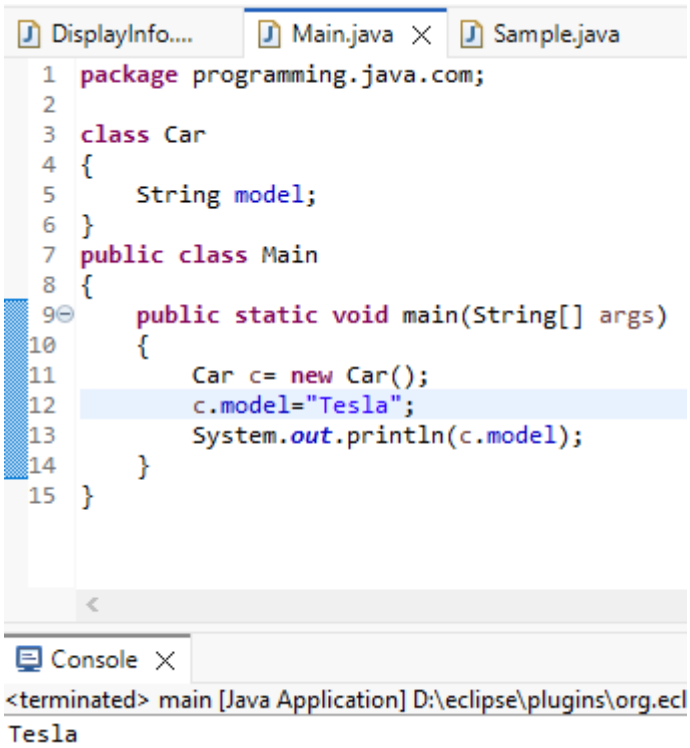
Source Code

// Default Access modifier

```
class Car {
    String model; // default access
}
public class Main {
       public static void main(String[] args){
        Car c = new Car();
        c.model = "Tesla"; // accessible within the same package
        System.out.println(c.model);
    }
}
```

Output:

Tesla



//Private Access Modifier

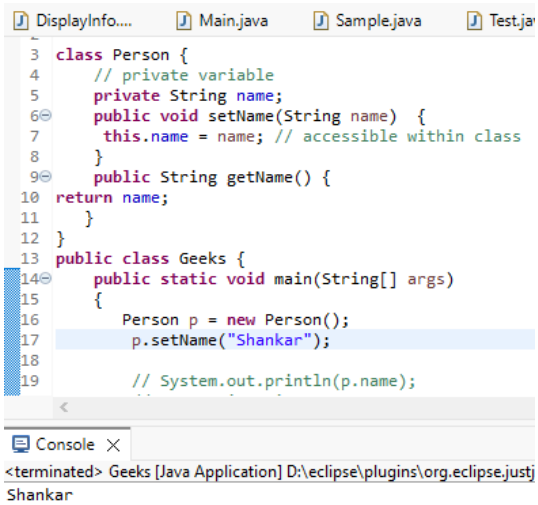
```
class Person {
    // private variable
    private String name;
```

```
    public void setName(String name)  {
     this.name = name; // accessible within class
    }
    public String getName() {
            return name;
   }
}
public class Geeks {
    public static void main(String[] args)
    {
       Person p = new Person();
        p.setName("Alice");

        // System.out.println(p.name);
        // Error: 'name'
        // has private access
        System.out.println(p.getName());
    }
}
```

Output

Shankar

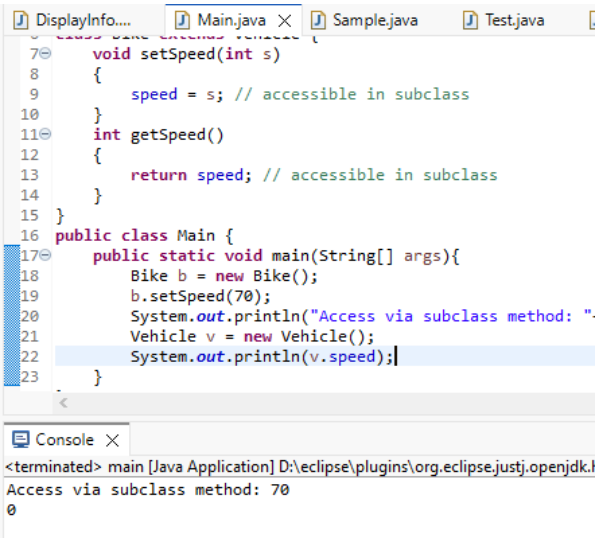


//Protected Access Modifier

```
class Vehicle {
    protected int speed; // protected member
}
class Bike extends Vehicle {
    void setSpeed(int s)
    {
        speed = s; // accessible in subclass
    }
    int getSpeed()
    {
        return speed; // accessible in subclass
    }
}
public class Main {
    public static void main(String[] args){
        Bike b = new Bike();
        b.setSpeed(100);
        System.out.println("Access via subclass method: "+ b.getSpeed());
        Vehicle v = new Vehicle();
        System.out.println(v.speed);
    }
}
```

Output

```
Access via subclass method: 70
0
```

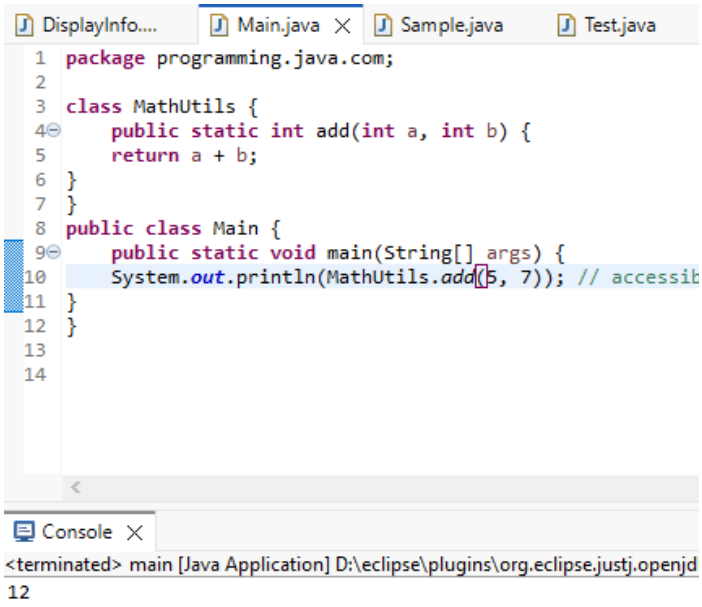


//Public Access Modifier

```
class MathUtils {
        public static int add(int a, int b) {
        return a + b;
    }
}
public class Main {
        public static void main(String[] args) {
        System.out.println(MathUtils.add(5, 10)); // accessible anywhere
    }
}
```

Output

12



POST LAB EXERCISE

1. Write a Java program to implement Default access modifier which has a default class within the same package and a default class from a different package.

```
class Print {
    void disp () {
        System.out.println("Default access inside the same package");
    }
}

public class Main {
    public static void main(String[] args) {
        Print p = new Print();
        p.disp ();
    }
}
```

2. Which access modifier provides the highest level of access?

   Public is the access modifier that provides the highest level of access.

ASSESSMENT

Description	Max Marks	Marks Awarded
Pre Lab Exercise	**5**	
In Lab Exercise	**10**	
Post Lab Exercise	**5**	
Viva	**10**	
Total	**30**	
Faculty Signature		

Experiment Number :11 **Date: 25/02/2026**

Implementation of multiple inheritance in Java using Interface

Aim:

Write a Java program to implement multiple inheritance using Interface.

PRE LAB EXERCISE QUESTIONS

1. Why Java does not support multiple inheritance using classes like that of C?
   Java does not support multiple inheritance using classes to avoid ambiguity problems like the **Diamond Problem**.
   If two parent classes have the same method, the child class would not know which one to inherit.
   To keep the design simple and avoid confusion, Java allows multiple inheritance only through interfaces.

2. What is the primary purpose of an interface in Java?

   The primary purpose of an interface in Java is to define a contract that classes must follow.
   It specifies what methods a class should implement without providing full implementation.
   Interfaces support abstraction and allow multiple inheritance in Java.

   To define a contract of methods a class must implement

IN LAB EXERCISE

Objective

To demonstrate how an interface in Java defines constants and abstract methods, which are implemented by a class.

Source Code

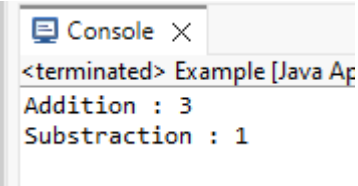
```
import java.io.*;
  // Add interface
interface Add{
    int add(int a,int b);
  }
// Sub interface
interface Sub{
        int sub(int a,int b);
  }
// Calculator class implementing Add and Sub
class Cal implements Add , Sub
  {
// Method to add two numbers
        public int add(int a,int b){
        return a+b;
    }
        // Method to sub two numbers
        public int sub(int a,int b){
        return a-b;
    }
}
class Example{
    // Main Method
    public static void main (String[] args){

        // instance of Cal class
        Cal x = new Cal();
```

```
        System.out.println("Addition : " + x.add(2,1));
        System.out.println("Substraction : " + x.sub(2,1));
    }
}
```

Outputs

Addition : 3
Subtraction : 1



POST LAB EXERCISE

3. Can a functional interface extend another interface?

   Yes, a functional interface can extend another interface if it still has only one abstract method.
   If more than one abstract method exists after extending, it is no longer a functional interface.

4. Which feature was introduced in interfaces starting from Java 8?

   Starting from Java 8, **default methods and static methods** were introduced in interfaces.
   These allow interfaces to have method implementations without breaking existing classes.

ASSESSMENT

Description	Max Marks	Marks Awarded
Pre Lab Exercise	**5**	
In Lab Exercise	**10**	
Post Lab Exercise	**5**	
Viva	**10**	
Total	**30**	
Faculty Signature		

Experiment Number : 12 **Date: 25/02/2026**

Implementation of exception handling in Java

Aim:

Write a java program to implement exception handling.

PRE LAB EXERCISE QUESTIONS

1. What is exception handling in Java?
   Exception handling in Java is a mechanism used to handle runtime errors so that the program does not crash. It allows a program to detect, catch, and manage exceptions using keywords like try, catch, throw, and finally. This helps maintain the normal flow of the program even when errors occur.

2. What is the purpose of using try-catch blocks in exception handling?
   The try-catch block is used to handle exceptions that may occur during program execution. The try block contains code that might cause an exception, and the catch block handles the error. It prevents the program from crashing and allows it to continue running normally.

IN LAB EXERCISE

Objective

To understand and implement exception handling through try-catch and finally blocks.

Source Code

```
import java.util.InputMismatchException;

import java.util.Scanner;

public class ExceptionHandlingExample {

 public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

         try {

        System.out.print("Enter an integer: ");

        int num = scanner.nextInt();

         int result = 10 / num;

        System.out.println("Result: " + result);

        } catch (InputMismatchException e) {
```

```
        System.out.println("Invalid input! Please enter an integer.");
        } catch (ArithmeticException e) {
        System.out.println("Cannot divide by zero!");
        } finally {
        scanner.close();
        System.out.println("Program execution completed.");
        } }
}
```

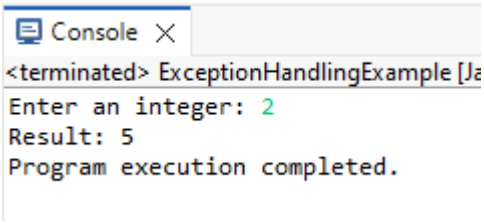
Outputs

Output 1 (Valid Input - Non-zero Integer):

Enter an integer: 2

Result: 5

Program execution completed.

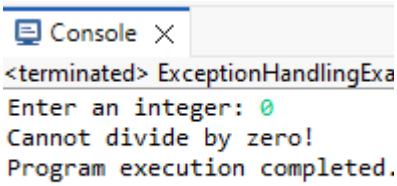


Output 2 (Valid Input - Zero):

Enter an integer: 0

Cannot divide by zero!

Program execution completed.

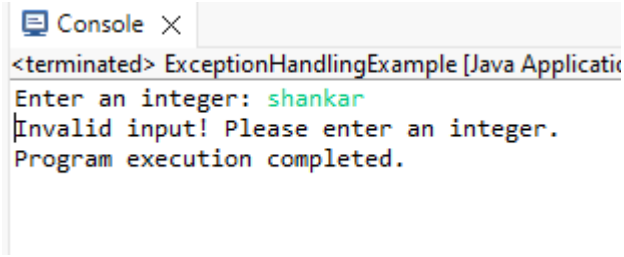


Output 3 (Invalid Input - Non-integer):

Enter an integer: shankar

Invalid input! Please enter an integer.

Program execution completed.



POST LAB EXERCISE

1. What is the purpose of the Scanner object in the Java program?
   The Scanner object is used to take input from the user at runtime.
   It reads data such as integers, strings, and other types from the keyboard.

2. What exceptions are expected to be thrown in the code within the try block? Why?
   An InputMismatchException may be thrown if the user enters data of the wrong type (e.g., text instead of an integer).
   ArithmeticException may also occur if there is an illegal mathematical operation like division by zero.

ASSESSMENT

Description	Max Marks	Marks Awarded
Pre Lab Exercise	**5**	
In Lab Exercise	**10**	
Post Lab Exercise	**5**	
Viva	**10**	
Total	**30**	
Faculty Signature		

Experiment Number : 13 **Date: 26/02/2026**

Implementation of User-defined Exception in Java

Aim:

Write a java program to implement User-defined Exception.

PRE LAB EXERCISE QUESTIONS

1. What is Java custom exception?
   A custom exception in Java is a user-defined exception created by extending the Exception class or RuntimeException class.
   It is used to handle application-specific errors according to program requirements.

2. What are the two types of custom exceptions in Java?
   The two types of custom exceptions are checked exceptions (extend Exception) and unchecked exceptions (extend RuntimeException).

IN LAB EXERCISE

Objective

To understand and implement Checked and Unchecked custom exceptions.

Source Code

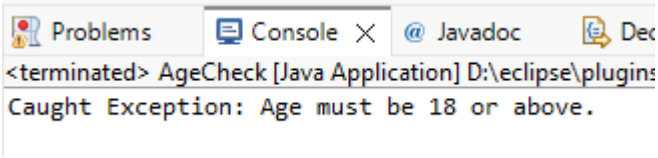
Ex 1: // Custom Checked Exception

```
class InvalidAgeException extends Exception {
   public InvalidAgeException(String m) {
      super(m);
   }
}
// Using the Custom Exception
public class AgeCheck{
   public static void validate(int age)
     throws InvalidAgeException {
       if (age < 18) {
          throw new InvalidAgeException("Age must be 18 or above.");
```

```
        }
        System.out.println("Valid age: " + age);
    }
public static void main(String[] args) {
  try {
            validate(12);
        } catch (InvalidAgeException e) {
            System.out.println("Caught Exception: " + e.getMessage());
        }
    }
}
```

Output 1

Caught Exception: Age must be 18 or above.



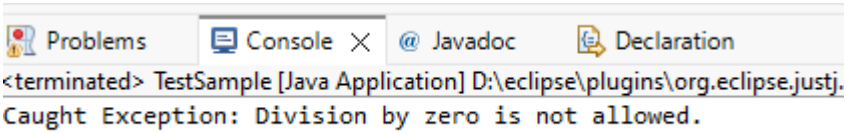
Ex 2: // Custom Unchecked Exception

```
class DivideByZeroException extends RuntimeException {
    public DivideByZeroException(String m) {
        super(m);
    }
}
public class TestSample {
    public static void divide(int a, int b) {
        if (b == 0) {
            throw new DivideByZeroException("Division by zero is not allowed.");
        }
        System.out.println("Result: " + (a / b));
    }
```

```
public static void main(String[] args) {
  try {
        divide(10, 0);
     } catch (DivideByZeroException e) {
        System.out.println("Caught Exception: " + e.getMessage());
     }
   }
}
```

Output 2

Caught Exception: Division by zero is not allowed.



POST LAB EXERCISE

1. What is required to create a custom exception in Java?
   To create a custom exception in Java, you must create a new class that extends either Exception (for checked) or RuntimeException (for unchecked).
   You can also define constructors to pass error messages to the superclass.

2. List some use cases for the checked and unchecked exceptions.

   - **Checked exceptions:** File handling, database connections, network operations where recovery is possible.

   - **Unchecked exceptions:** Programming errors like division by zero, null references, or invalid input.

ASSESSMENT

Description	Max Marks	Marks Awarded
Pre Lab Exercise	**5**	
In Lab Exercise	**10**	
Post Lab Exercise	**5**	
Viva	**10**	
Total	**30**	
Faculty Signature		