Experiment Number : 8 **Date: 25.02.2026**

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 is a built-in package in Java that provides a collection of utility classes and interfaces used for data manipulation and general programming tasks. It includes classes for handling data structures, date and time operations, random number generation, string tokenizing, and user input through the Scanner class. One of the most important parts of the java.util package is the Java Collection Framework, which provides a standardized architecture to store and manage groups of objects. The Collection Framework includes interfaces such as List, Set, Queue, and Map, along with their implementation classes like ArrayList, LinkedList, Vector, HashSet, TreeSet, HashMap, and TreeMap. These classes help programmers efficiently store, retrieve, and manipulate data.

2. What are the two types of packages in Java?

There are two types of packages in Java: built-in packages and user-defined packages. Built-in packages are predefined packages that are provided by Java to support various functionalities such as input/output operations, networking, utilities, and data handling. Examples include java.lang, java.util, and java.io. User-defined packages are packages created by programmers to organize their own classes and interfaces. They help in structuring large programs, avoiding name conflicts, and improving code reusability and maintainability.

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: 34

```
Random number: 34
```

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!

```
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?

If two classes with the same name from different packages are imported into the same Java file, it will cause a compile-time error due to ambiguity. The compiler will not be able to determine which class is being referred to when the class name is used. To resolve this issue, the programmer must use the fully qualified class name (including the package name) instead of importing both classes. This removes the ambiguity and clearly specifies which class should be used.

2. What is the purpose of using packages in Java?

The purpose of using packages in Java is to organize related classes and interfaces into a structured hierarchy. Packages help avoid name conflicts between classes, improve code readability, and make large applications easier to manage. They also provide access protection and better maintainability by grouping related functionalities together. By using packages, programmers can create modular and reusable code.

3. Which built-in Java package would you use if you want to create a GUI window and display a message?

- java.util
- java.sql
- java.awt
- java.net

The correct answer is **java.awt**. The java.awt package contains classes for creating graphical user interface (GUI) components such as windows, buttons, labels, and text fields. It provides the basic tools needed to design and display GUI applications in Java.

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: 25.02.2026**

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 from a package in Java, you use the import keyword followed by the full package name and the class name. For example, if you want to import only the Scanner class from the java.util package, you write import java.util.Scanner;. This allows you to use the Scanner class directly in your program without writing the full package name every time. To import multiple classes from the same package, you can use the wildcard symbol *. For example, import java.util.*; imports all classes available in the java.util package, so you can use classes like ArrayList, Vector, Scanner, etc., directly in your code.

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

The package that is always imported by default in every Java class is java.lang. You do not need to explicitly import it because the Java compiler automatically makes it available. Classes such as String, System, Math, Integer, and Objectbelong to java.lang, which is why you can use them without writing an import statement

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

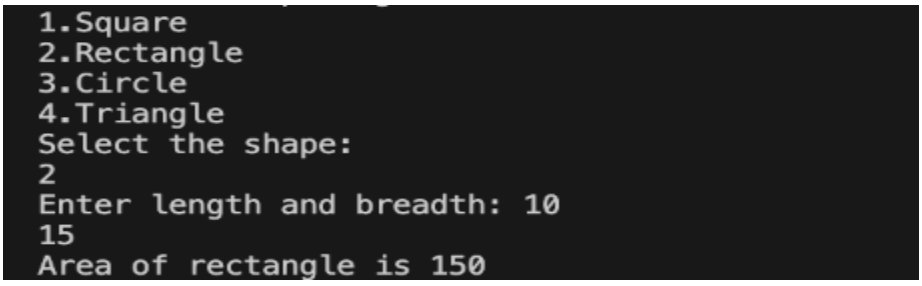
1. Square
2. Rectangle
3. Circle
4. Triangle

Select the shape

2

Enter the dimensions of the rectangle: 10 15

Area of the rectangle with dimensions 10&15 is 150



//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: 100000

Enter the dearness allowance: 5000

Enter the house rent allowance: 2000

The total Gross salary of employee is: 107000

```
Enter the basic salary: 100000
Enter the dearness allowance: 5000
Enter the house rent allowance: 2000
The total Gross salary of employee is: 107000
```

POST LAB EXERCISE

1. Find the key differences between java.util and java.lang packages.

The main difference between the java.util and java.lang packages is their purpose and usage. The java.lang package contains fundamental classes that are essential for basic programming in Java, such as String, System, Math, Object, Integer, and Thread. This package is automatically imported by default in every Java program, so there is no need to write an import statement for it. On the other hand, the java.util package provides utility classes that support data structures, collections, date and time handling, random number generation, and user input. It includes important classes and interfaces such as ArrayList, LinkedList, HashMap, HashSet, Vector, Scanner, and Date. Unlike java.lang, the java.utilpackage must be imported explicitly if its classes are used.

2. List some of the subpackages of java.util

Some of the subpackages of java.util include java.util.concurrent, which provides support for multithreaded programming; java.util.function, which contains functional interfaces used in lambda expressions; java.util.logging, which supports logging operations; java.util.regex, which provides classes for regular expression handling; and java.util.stream, which supports stream processing introduced in Java 8.

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: 26.02.2026**

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?

Java has **four access modifiers**:

1. **private**
2. **default** (no keyword)
3. **protected**

4. **public**

2. Which access modifiers can be used in Java to control access to class members?

To control access to **class members** (variables, methods, constructors), Java uses:

- **private**
- **default**
- **protected**
- **public**

All four modifiers can be used for class members.

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

```
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

Alice

```
Alice
```

//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: 100

0

```
Access via subclass method: 100
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

15

```
15
```

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.

Default access means:

- No keyword is used.
- Accessible only **within the same package**.
- Not accessible from different package.

Package 1:

File TestDefault.java

```
package package1;
```

```
class DefaultClass {     // default class

   void display() {     // default method
      System.out.println("Default class inside same package");
   }
}

public class TestDefault {
   public static void main(String[] args) {

      DefaultClass obj = new DefaultClass();
      obj.display();   // Accessible (same package)
   }
}
```

Package 2:

File: TestOutside.java

```
package package2;

import package1.DefaultClass;

public class TestOutside {
   public static void main(String[] args) {

      DefaultClass obj = new DefaultClass();
      obj.display();
   }
}
```

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

public provides the highest level of access.

Because public members are accessible:

- Inside same class
- Inside same package
- In subclasses
- In different packages

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: 26.02.2026**

Implementation of multiple inheritance in Java using Interface

Aim:

Write a Java program to implement multiple inheritance using Interface.

PRE LAB EXERCISE

QUESTIONS

Ruddhida R P – 24BCS230

1. Why Java does not support multiple inheritance using classes like that of C?

Java does not support multiple inheritance using classes to avoid the **Diamond Problem**.

In multiple inheritance, if two parent classes have the same method, the child class cannot decide which method to inherit. This creates ambiguity and confusion.

To avoid this complexity and maintain simplicity, Java allows:

- Multiple inheritance using **interfaces**
- Not using multiple classes

2. What is the primary purpose of an interface in Java?
   A. To store data using instance variables
   B. To define a contract of methods a class must implement
   C. To allow creation of objects
   D. To improve runtime performance

Correct Answer:

To define a contract of methods a class must implement.

An interface specifies method declarations that a class must implement.

It ensures that different classes follow the same structure.

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

```
Addition : 3
Subtraction : 1
```

POST LAB EXERCISE

1. Can a functional interface extend another interface?

Yes, a functional interface **can extend another interface**,

but it must still have **only one abstract method** in total.

If extending another interface results in more than one abstract method,

then it will no longer be a functional interface.

2. Which feature was introduced in interfaces starting from Java 8?
   A. Constructors
   B. Private methods
   C. Default and static methods
   D. Final classes

Correct Answer:

Default and static methods

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: 26.02.2026**

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 normal flow of the program is not interrupted.

An **exception** is an unexpected event that occurs during program execution, such as:

- Dividing a number by zero
- Accessing an invalid array index
- Trying to open a file that does not exist

In Java, exceptions are handled using keywords like:

- try
- catch
- finally
- throw
- throws

All exception classes are part of the java.lang package and are derived from the class **Exception**.

2. What is the purpose of using try-catch blocks in exception handling?

The **purpose of try-catch blocks** is to:

Prevent Program Crash

If an error occurs, the program does not stop suddenly.

Handle Errors Gracefully

You can display a meaningful message instead of a system error.

Maintain Normal Program Flow

After handling the exception, the program continues execution.

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.");
        } }
}
```

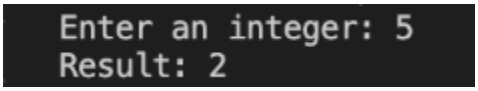
Outputs

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

Enter an integer: 5

Result: 2

Program execution completed.

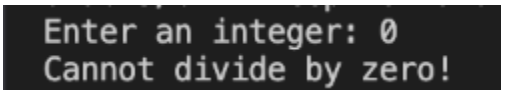


Output 2 (Valid Input - Zero):

Enter an integer: 0

Cannot divide by zero!

Program execution completed.



Output 3 (Invalid Input - Non-integer):

Enter an integer: abc

Invalid input! Please enter an integer.

Program execution completed.

```
Enter an integer: abc
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 belongs to the **Scanner** class, which is part of the java.util package.

Purpose:

- Read user input from keyboard
- Read different data types like:

int → nextInt()

double → nextDouble()

String → nextLine()

- Parse input into required data type

2. What exceptions are expected to be thrown in the code within the try block? Why?

- Scanner object is used to take user input from the keyboard.
- Expected exceptions:

ArithmeticException -when dividing by zero

InputMismatchException - when wrong input type is entered

- These occur due to invalid mathematical operations or incorrect user input.

NumberFormatException - Converting string to number fails

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 or RuntimeException class.

It is used to define application-specific errors that are not covered by built-in Java exceptions.

2. What are the two types of custom exceptions in Java?

A custom exception is a user-defined exception created by extending Exception or RuntimeException to handle application-specific errors.

The two types of custom exceptions are:

1. Checked custom exception
2. Unchecked custom exception

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);
    }
}
// Using the Custom Exception
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?
   A. Extend the Object class
   B. Extend the Throwable class
   C. Extend Exception or RuntimeException
   D. Implement the Serializable interface

Extend Exception or Runtime Exception

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

To create a custom exception in Java, we must extend Exception or RuntimeException.

Checked exceptions are used for recoverable errors like file handling and database operations, while unchecked exceptions are used for programming errors such as division by zero and null pointer 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	