Experiment Number : 8 **Date: 12-2-2026**

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?
   java.util is a built-in Java package that provides utility classes for handling data and common operations.
   It contains the **Collection Framework**, which includes interfaces like **List, Set, Queue, Map** and classes such as **ArrayList, LinkedList, HashSet, HashMap, TreeMap**, etc., used to store and manage groups of objects.
2. What are the two types of packages in Java?

- **Built-in Packages** – Predefined packages provided by Java (e.g., `java.lang`, `java.util`, `java.io`).
- **User-defined Packages** – Packages created by programmers using the `package` keyword to organize classes.

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:

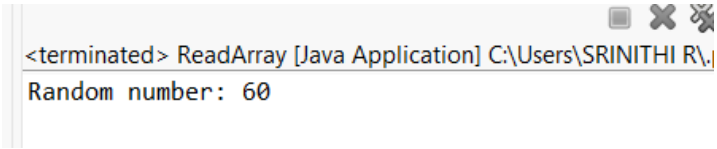
- java.lang: Contains language support classes(e.g, classes that define primitive data types, math operations). This package is automatically imported.
- java.io: Contains classes for supporting input/output operations.
- java.util: Contains utility classes that implement data structures such as Linked 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



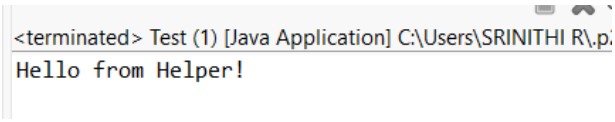
User-defined Packages are the packages that are defined by the user.

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:

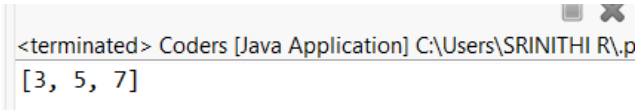


//Importing all classes from a package.

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



POST LAB EXERCISE

1. What will happen if two classes in different packages have the same name and are imported in a Java file?
   It creates a **compile-time ambiguity error** because the compiler cannot decide which class to use.
   To resolve this, you must use the **fully qualified name** (e.g., packageName.ClassName) instead of importing both.

2. What is the purpose of using packages in Java?
   Packages are used to:

- Organize related classes and interfaces into groups
- Avoid naming conflicts
- Provide better access control
- Improve code management and reusability

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
   It contains classes like Frame, Label, and Button used to build GUI applications.

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: 12-2-26**

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?
   Import a single class:
   import java.util.ArrayList;
   Import multiple classes (entire package):
   import java.util.*;

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

   `java.lang`

   It is automatically imported and contains commonly used classes like `String`, `System`, `Math`, `Integer`, etc.

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

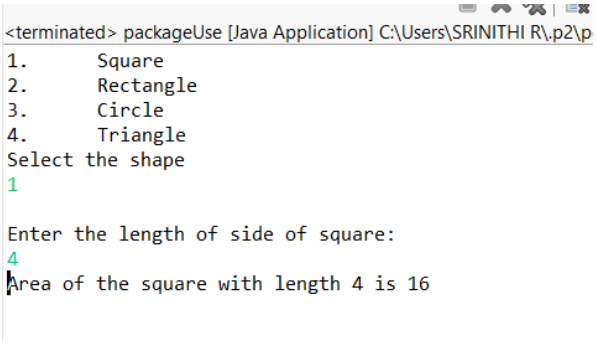
```
//Package Creation:
package useFull;
import java.util.*;
public class UseMe
{
        Scanner obj=new Scanner(System.in);
public static void area()
      {
          class method{
           void aos(int a)
           {
             System.out.print("\nArea of square with length "+a+" is "+(a*a));
           }
          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();
        }
    }
```

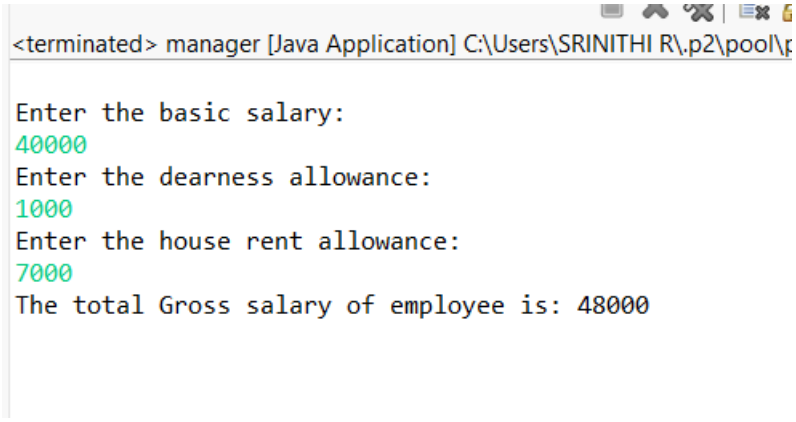
Output



//Package Implementation-2:

```
    import useFull.UseMe;
    class manager
    {
       public static void main(String args[])
       {
        UseMe obj=new UseMe();obj.salary();
        }
    }
```

Output



POST LAB EXERCISE

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

Feature	java.lang	java.util
Import	Automatically imported	Must be imported manually
Purpose	Basic language support	Utility classes & data structures
Contains	String, System, Math, Wrapper classes	ArrayList, HashMap, Scanner, Collections
Usage	Core Java features	Collection Framework, date, random, etc.

2. List some of the subpackages of java.util

- `java.util.concurrent`
- `java.util.function`
- `java.util.logging`
- `java.util.regex`
- `java.util.stream`

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?
   The four access modifiers in Java are:
   - **public** – Accessible from anywhere (any class, any package).
   - **protected** – Accessible within the same package and by subclasses (even in different packages).
   - **default (no modifier)** – Accessible only within the same package.
   - **private** – Accessible only within the same class.

2. Which access modifiers can be used in Java to control access to class members?
   For **class members** (variables, methods, constructors), all four access modifiers can be used:
   - public
   - protected
   - default (no modifier)
   - private

   Example:

```
public class Example {
   public int a;       // accessible everywhere
   protected int b;    // same package + subclass
   int c;              // default (same package only)
   private int d;      // same class only
}
```

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:

```
<terminated> Main [Java Application] C:\Users\SRINITHI R\
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

```
<terminated> Geeks [Java Application] C:\Users\SRINITHI R\.
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

```
<terminated> Main1 [Java Application] C:\Users\SRINITHI R\.
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

```
<terminated> Main1 [Java Application] C:\Users\SRINITHI R\.p
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.
   In Java, **default access** allows access only within the same package.
   Package 1:

```
package package1;

class DefaultClass {   // default class
    void show() {
        System.out.println("Same package access");
    }
}
```

```
package package1;

public class TestSame {
    public static void main(String[] args) {
        DefaultClass obj = new DefaultClass();
        obj.show();   // ✓ Accessible (same package)
    }
}
```

   Package 2:

```
package package2;

import package1.DefaultClass;  // ❌ Error

public class TestDifferent {
    public static void main(String[] args) {
        DefaultClass obj = new DefaultClass(); // Not accessible
    }
}
```

   Thus, default class is accessible **only within the same package**, not from a different package.

2. Which access modifier provides the highest level of access?
   public provides the **highest level of access** because it allows access:

- Within the same class
- Within the same package
- From different packages
- From any other class

   So, **public is the most accessible modifier** 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 :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 the **Diamond Problem** (ambiguity when two parent classes have the same method).
This prevents:

- Method conflict ambiguity
- Complexity in inheritance hierarchy

Instead, Java supports multiple inheritance using **interfaces**.

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

D. To allow creation of objects

D. To improve runtime performance

Correct Answer: B. To define a contract of methods a class must implement

An interface defines:

- A set of abstract methods
- Rules that implementing classes must follow

It provides **100% abstraction (conceptually)** and supports multiple inheritance through interfaces.

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

```
<terminated> Example [Java Application] C:\Users\SRINITHI R\.
Addition : 3
Substraction : 1
```

POST LAB EXERCISE

1. Can a functional interface extend another interface?

    - Yes, a **functional interface** in Java **can extend another interface**, **only if** it still has **exactly one abstract method** after inheritance.
    - If it has more than one abstract method, it is **not** 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

C. Default and static methods

Interfaces can contain:

- default methods
- static methods

These were introduced to support **lambda expressions** and improve backward compatibility.

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?
   In Java, **exception handling** is a mechanism used to handle runtime errors so that the program does not crash and continues execution normally.
   It helps to:
   - Detect runtime errors
   - Handle them gracefully
   - Maintain normal program flow

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

   The **try-catch** block is used to:

   - Place risky code inside the try block
   - Catch and handle errors inside the catch block

   Purpose:

   - Prevent program termination
   - Provide error message
   - Allow the program to continue 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):

```
<terminated> ExceptionHandlingExample [Java Application] C:\Users\SRINITHI R
Enter an integer: 5
Result: 2
Program execution completed.
```

Output 2 (Valid Input - Zero):

```
<terminated> ExceptionHandlingExample [Java Application] C:\Users\SRINITHI R\.
Enter an integer: 0
Cannot divide by zero!
Program execution completed.
```

Output 3 (Invalid Input - Non-integer):

```
<terminated> ExceptionHandlingExample [Java Application] C:\Users\SRINITHI R\
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?
   In Java, the **Scanner** object is used to:

- Take **input from the user**
- Read different data types like int, double, String, etc.
- Read input from sources like keyboard (System.in), files, etc.

2. What exceptions are expected to be thrown in the code within the try block? Why?
   Common exceptions (depending on typical input programs):

- **ArithmeticException**
  - Occurs when dividing by zero.
  - Example: int result = a / 0;
- **InputMismatchException**
  - Occurs when the user enters a wrong data type.
  - Example: Entering text instead of an integer.
- **NumberFormatException** (if parsing is used)
  - Occurs when converting invalid string to number.

These exceptions occur due to **invalid input or illegal operations** inside the try block.

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?
   In Java, a **custom exception** is a user-defined exception created by extending the existing exception classes.
   It is used to:
   - Define application-specific errors
   - Provide meaningful error messages
   - Handle special business logic conditions

   Example: Creating InvalidAgeException for age less than 18.

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

   Checked Custom Exception:

   - Extend Exception
   - Must be handled using try-catch or throws

   Unchecked Custom Exception:

   - Extend RuntimeException
   - Not mandatory to handle at compile time

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

```
<terminated> AgeCheck (1) [Java Application] C:\Users\SRINITHI F
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

```
<terminated> TestSample [Java Application] C:\Users\SRINITHI R\.p2\pool
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

C. Extend Exception or RuntimeException

- Extend Exception → for **checked exception**
- Extend RuntimeException → for **unchecked exception**

2. List some use cases for the checked and unchecked exceptions.
   Checked Exceptions (compile-time checked):
   Must be handled using try-catch or throws.

Use cases:

- File handling (IOException)
- Database operations (SQLException)
- Network operations
- Situations where recovery is possible

Unchecked Exceptions (runtime exceptions):

Occur at runtime and are not checked at compile time.

Use cases:

- Programming errors (NullPointerException)
- Arithmetic errors (ArithmeticException)
- Array index errors (ArrayIndexOutOfBoundsException)
- Invalid arguments (IllegalArgumentException)

ASSESSMENT

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