Page No:
Date:

Name : UDHAYARAJAN. J.
Reg NO : 2202561
Sub : JAVA PROGRAMMING
Date : 11/11/2025
class : MCA-I.
Type : CCE2 (Open book test - unit 2)

Q1). What is a package? Explain the steps to create a user defined package with examples.

Package:

A package in Java is a mechanism to organize a group related classes, interfaces and sub-packages.

It helps in avoiding name conflicts, maintaing modularity and controlling accost to classes.

types of packages in Java:

1. Build-in packages - provided by Java. (java.util, java.io, java.lang).
2. User-defined package - Created By programmer to structure their code.

steps to create user defined. packages:

1. Declare a package
   package MyMathFunction;
2. Save in same-named Folder
   MyMathFunction / MathUtils.java.
3. Compile package class.
   Javac MyMathFunction/MathUtils.java.
4. import package.
   import MyMathFunction.MathUtils;
5. Run the program.
   Java MainApp.java.

(Ref java practical.pdf).

example:

MyMathFunctions / MathUtils.java.
(create the package).

```
package MyMathFunctions;
public class MathUtile {
    public static double findSqrt(double number) {
        return Math.sqrt(number);
    }
    public static double findpower(double base, double exponent) {
        return Math.pow(base, exponent);
    }
}
```

MainApp.java.
(importing)

```
import MyMathFunctions.MathUtils;
public class MainApp {
    public static void main(String[]args) {
        double num = 25.0;
        double base = 2.0;
        double exponent = 3.0;

        double sqrtResult = MathUtils.findSqrt(num);
        double powerResult = MathUtils.findpower(base, exponent);
        System.out.println("the Squaae root of "+num+" is:"
                +sqrtResult);
        System.out.println("base "+base+" raised to the power of "+
                exponent+" is: "+powerresult);
    }
}
```

output:
The square root of 25 is 5
2.0 raised to the power of 3 is: 9.

Page No:
Date:

Q2) Explain import and static import with an example:

import :

The import statement in java is used to access classes from other packages with using their fully qualified names.

syntax:

```
import package_name.class_name ; // import specific class
import package_name.* ; // import all classes from the package.
```

example:

```
import java.util.Scanner ; // importing Scanner class from java.util. package.

public class ImportExample {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter your name");
        String name = sc.nextLine();
        System.out.println("Hello, " + name);
    }
}
```

output:

Enter your name : Udhayarajan J.
Hello, Udhayarajan J.

Static import:

The static import statement is used to import static members (fields and methods) of a class so they can used directly without class name.

Syntax:

```
import static package-name.class_name.static_member;
import static package-name.class_name.*;
```

example:

```
import static java.lang.Math.*; // import all static
                                   members of Math class
public class StaticImportExample {
    public static void main(String[] args) {
        double result = sqrt(16) + pow(2, 3);
        System.out.println("result" + result);
    }
}
```

output:

result : 13.

Q3) Exception handling in details with example.

Exception:

An Exception is an unexpected event that occurs during execution of a program and disrupts the normal flow of instruction.

ex:

1. Divided a number by zero.
2. Accessing invalid array index
3. opening non-existing file.

Page No:
Date:

Exception Handling:

Exception Handling is mechanism in Java to detect and handle runtime errors so that the normal flow of the program is maintained.

Java provides a powerful exception handling framework using try, catch, throw, throws and finally.

try - contains code that might throw an exception.
catch - Handles the exception that occurs in the try block.
finally - executes code whether an exception occur or not (used for cleanup).
throw - Used to manually throw an exception.
throws - Declare exceptions that can be thrown by a method.

Types of Exception:

checked Exception:
- checked at compile time.
- ex: IOException, SQLException, FileNotFound.

Unchecked Exception:
- checked at Runtime.
- ex: Arithmetic Exception, Array IndexOutOfBounds exception, Nullpointer Exception.

Errors:
- Serious issues beyond the program's control
- ex: OutOfMemoryError, StackOverflowError.

General Syntax:

```
try {
    // code that may cause an exception.
}
catch (ExceptionType e) {
    // code to handle the exception
}
finally {
    // code that will always execute
}
```

example:

Handling Divided by zero.

```
public class ExceptionExample {
    public static void main(String[] args) {
        try {
            int a=10;
            int b=0;
            int result = a/b;
            System.out.println("result:"+ result);
        }
        catch (ArithmeticException e) {
            System.out.println("Error : Divided by zero");
        }
        finally {
            System.out.println("Execution Completed");
        }
    }
}
```

output:

Error : Divided by zero.
Execution Completed.

Page No:
Date.

Multiple catch blocks:

```
public class MultiCatchExample {
    public static void main(String[] args) {
        try {
            int[] arr = {1, 2, 3};
            System.out.println(arr[5]);
        }
        catch (ArithMaticException e) {
            System.out.println("Arithmetic error occured");
        }
        catch (ArrayIndexOutOfRangeException e) {
            System.out.println("Array index is invalid");
        }
        catch (Exception e) {
            System.out.println("some other error occured");
        }
    }
}
```

output:
Array index is invalid.

Throws and Throw.

```
public class ThrowExample {
    static void checkAge(int age) throws ArithMaticException {
        if (age > 18) {
            System.out.println("Access granted");
        }
        else {
            throw new ArithmeticException("
                Access denied - you must above
                18 years old");
        }
    }
```

```
public static void main(String[] args) {
    try {
        checkAge(15);
    }
    catch(ArithMeticException e) {
        System.out.println(e.getMessage());
    }
}
```

output

Access denied. you must above 18 years old.

Q4) write code for a user defined exception for age that is valid or not for voting and a driving licence.

Sol:

```
public class InvalidAgeException extends Exception {
    public InvalidAgeException(String message) {
        super(message);
    }
}

public class AgeValidation {
    static void checkVotingAge(int age) throws
                                InvalidAgeException {
        if (age < 18) {
            throw new InvalidAgeException("Not
                eligible for voting age must be
                18 or above");
        }
        else {
            System.out.println("Eligible for voting");
        }
    }
```

```
static void checkDrivingAge(int age) throws
                                InvalidAgeException {
    if (age < 18) {
        throw new InvalidAgeException("Not
                eligible for a driving licence
                Age must 18 or above");
    }
    else {
        System.out.println("Eligible for driving
                                licence");
    }
}
public static void main(String[] args) {
    try {
        int age1 = 16;
        checkVotingAge(age);
        checkDrivingAge(age);
    }
    catch(InvalidAgeException e) {
        system.out.println("Exception caught:"
                        +e.getMessage());
    }
    finally {
        System.out.println("Age validation
                        completed");
    }
}
```

Output:

(For age = 16)

Exception caught Not eligible for voting! Age must 18 or above.

Age validation completed.

(for age = 20)
Eligible for voting
Eligible for driving licence.
Age validation Completed.

Q5). Explain threading with an Example.

Thread:

A thread in java is a light weight sub-process or smallest unit of execution that runs independly within a program.

Thread allows a program to perform multiple tasks simultaneously - this is called multi threading.

Multithreading:

Multithreading is a feature of java that allows two or more threads to run concurrently

It helps in better performance and efficient cpu utilization.

Advantages:

- Efficient CPU use.
- Faster Execution.
- Simplifies program design.
- independent execution.

ways to Create Thread in Java.

- By extending the thread class.
- By implementing the runnable interface.

example:

(Part: practical program)

implement a Thread - 1 Extending the thread class.

```
class cookingTask extends Thread {
    private string task;
    cookingTask(String Task) {
        this.task = Task;
    }
    public void run() {
        System.out.println(task + " is being prepared
                by " + Thread.currentThread
                .getName());
    }
}

public class Restaurant {
    public static void main(String[] args) {
        Thread t1 = new CookingTask("pasta");
        Thread t2 = new CookingTask("Salad");
        Thread t3 = new CookingTask("Dessert");
        Thread t4 = new CookingTask("Rice");

        t1.start();
        t2.start();
        t3.start();
        t4.start();
    }
}
```

output:

pasta being prepared by Thread - 0
Rice being prepared by Thread - 3
Dessert being prepared by Thread - 2
Salad being prepared by Thread - 1

Page No:
Date:

(Q6) Explain Runnable interface with an Example.

Runnable interface:

In Java the Runnable interface is used to create and run threads. It's part of the java.lang package and is implemented by any class whose instances are intended to be executed by a thread.

Advantages:

- Multiple inheritance
- Shared resource.
- clean structure
- flexibility.

Steps ~~ways~~ to create a Thread using Runnable interface.

1. create a class that implements the Runnable interface.
2. override the run() method.
3. create an instance of the class.
4. pass it to a thread object.
5. Start the thread using start() method.

example:

implement thread using the Runnable interface.

(for practical program)

```
public class MyRunnableTask implements Runnable {
    public void run() {
        System.out.println("Thread" + Thread.currentThread().
                getName() + "is running");
        for(int i=0; i<5; i++) {
            System.out.println(Thread.currentThread().
                getName() + " : " + i);
```

```
            try {
                Thread.sleep(100);
            }
            catch(InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}

public class RunnableExample {
    public static void main(String[] args) {
        MyRunnableTask obj = new MyRunnableTask();
        Thread t1 = new Thread(obj, "A");
        Thread t2 = new Thread(obj, "B");

        t1.start();
        t2.start();
    }
}
```

output:

Thread A is running.
A : 1
A : 2
Thread B is running.
B : 1
A : 3.
B : 2.
.....
.....
.....