1. Define package. Explain the steps involved in creating a user-defined package with an example

- **Ans:** A **package** in Java is a **namespace** that groups related **classes, interfaces, and sub-packages** together.
- Packages help in:
  - Organizing large programs
  - Avoiding **name conflicts**
  - Providing **access protection**
  - Improving **code reusability and maintenance**

Stes to Create a User-Defined Package (with Example)

Step 1: Write a Java program with `package` keyword

- Use the `package` statement as the **first line** of the program.
- Syntax:
- `package packagename;`

Example:

```
package mypackage;

public class Demo
{
    public void display()
    {
        System.out.println("This is a user-defined package");
    }
}
```

Step 2: Save the program

- Save the file with the **class name**.
- Example:
- `Demo.java`

Step 3: Compile the program

- Use `javac` with `-d` option to create the package directory.
- Command:
- `javac -d . Demo.java`
- This creates a folder named **mypackage** containing `Demo.class`.

Step 4: Use the package in another program

- Use the `import` statement to access the package.

Example:

```
import mypackage.Demo;

class Test
{
    public static void main(String args[])
    {
        Demo d = new Demo();
        d.display();
    }
}
```

Step 5: Compile and run the program

- Compile:
- `javac Test.java`
- Run:
- `java Test`

---

2. How do you create your own Exception class? Explain with a sample program with output

- **Ans:** A **user-defined exception** is a custom exception created by the programmer to handle **application-specific errors**.
- It is created by **extending the `Exception` class** (for checked exceptions) or **`RuntimeException` class** (for unchecked exceptions).

Steps to Create Your Own Exception Class

1. **Create a new class** that extends `Exception`
2. **Define a constructor** to pass an error message
3. **Use `throw` keyword** to raise the exception
4. **Handle the exception** using `try-catch`

Example:

```
class MyException extends Exception

{

   MyException(String msg)

   {

      super(msg);
```

```
    }
}


class Test
{
    public static void main(String args[])
    {
        try
        {
            throw new MyException("Custom Exception Occurred");
        }
        catch(MyException e)
        {
            System.out.println(e.getMessage());
        }
    }
}
```

Output:

Custom Exception Occurred

---

3. **Define an Exception. What are the key turns used in Exception Handling?Explain**

Ans: An **exception** is an **abnormal condition or runtime error** that occurs during the execution of a program and **disrupts the normal flow** of instructions.
Examples: division by zero, array index out of bounds, file not found, etc.

Exception Handling in Java

Exception handling is a mechanism used to **detect, handle, and recover from runtime errors**, ensuring **normal program execution**.

Key Keywords Used in Exception Handling

1. try

- Used to **enclose the code** that may generate an exception.
- Must be followed by **at least one `catch` or `finally` block**.
- Syntax:

```
try
{
    // risky code
}
```

2. catch

- Used to **handle the exception** thrown in the `try` block.
- Prevents abnormal termination of the program.
- Multiple `catch` blocks can be used.
- Syntax:

```
catch(ExceptionType e)
{
    // handling code
}
```

3. finally

- Executes **always**, whether an exception occurs or not.
- Used for **resource cleanup** (closing files, database connections, etc.).
- Syntax:

```
finally
{
    // cleanup code
}
```

4. throw

- Used to **explicitly generate an exception**.
- Can throw **only one exception at a time**.
- Syntax:

```
throw new ExceptionType("Error message");
```

5. throws

- Used to **declare exceptions** that a method may pass to the calling method.
- Mainly used for **checked exceptions**.
- Syntax:

```
methodName() throws ExceptionType
```

---

4. **Defiine Thread Demonstrate Creation of Multiple Threads with program.**

Ans: A **thread** is a **light-weight sub-process** that represents an **independent path of execution** within a program.
Multiple threads can run **concurrently**, sharing the same memory, to improve **CPU utilization and performance**.

Multithreading in Java

- Java supports multithreading to execute **two or more threads simultaneously**.
- Each thread has its own:
  - Program counter
  - Stack
- But shares:
  - Memory
  - Resources of the process

Ways to Create Threads in Java

1. By **extending `Thread` class**
2. By **implementing `Runnable` interface**

Creation of Multiple Threads (Using Thread Class)

Program

```
class MyThread extends Thread
{
    public void run()
    {
        System.out.println("Thread running: " +
Thread.currentThread().getName());
    }
}

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

```
        MyThread t1 = new MyThread();
        MyThread t2 = new MyThread();
        MyThread t3 = new MyThread();

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

Output

(Order may vary)

```
Thread running: Thread-0
Thread running: Thread-1
Thread running: Thread-2
```

---

5. **What is Enumeration? Explain the methods values() and valuesof()**

Ans: An **Enumeration (enum)** in Java is a **special data type** used to define a **fixed set of named constants**.
Enums improve **type safety**, **readability**, and **maintainability** compared to traditional constants.

Example: Days of a week, Directions, Colors, States, etc.

Methods Used with Enum

1. `values()` Method

- `values()` is a **static method** provided by the Java compiler.
- It returns an **array of enum constants** in the order they are declared.
- Mainly used to **iterate through enum values**.

Syntax:

```
EnumName.values();
```

2. `valueOf()` Method

- `valueOf()` returns the **enum constant** whose name matches the given string.
- If the string does not match, it throws **IllegalArgumentException**.
- Case-sensitive.

Syntax:

```
EnumName.valueOf("CONSTANT");
```

Example Program Demonstrating `values()` and `valueOf()`

```
enum Day
{
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY
}

class EnumDemo
{
    public static void main(String args[])
    {
        // Using values() method
        for(Day d : Day.values())
        {
            System.out.println(d);
        }

        // Using valueOf() method
        Day day = Day.valueOf("MONDAY");
        System.out.println("Selected day: " + day);
    }
}
```

Output

```
MONDAY
TUESDAY
WEDNESDAY
THURSDAY
FRIDAY
Selected day: MONDAY
```