Unit 3

- Defining a Package
- Access Protection
- Importing Packages

1. Introduction to Packages

- A **package** in Java is a collection of related classes, interfaces, and sub-packages.
- Think of it as a **folder/directory** in your computer that organizes files.
- Provides **modularity, reusability, and access protection**.
- Packages prevent **name conflicts** (e.g., java.util.Date vs. java.sql.Date).

👉 Two types of packages:

1. **Built-in Packages** → Already available in Java (e.g., java.lang, java.util, java.io).
2. **User-defined Packages** → Created by programmers.

---

2. Defining a Package

- To create a package, use the package keyword.
- The statement should be the **first line** of the Java program.

📌 **Example: Defining a Package**

```
// File: MyPackageClass.java

package mypackage;   // defining package


public class MyPackageClass {

    public void displayMessage() {

        System.out.println("Hello from MyPackageClass!");

    }

}
```

- Save this file inside a folder named mypackage.
- Compile using:

- javac -d . MyPackageClass.java

(-d . tells compiler to put class file in proper directory structure)

---

3. Access Protection in Packages

- Java provides **access modifiers** to control accessibility of classes, methods, and variables.

Modifier	**Same Class**	**Same Package**	**Subclass (diff package)**	**Other package**
public	✓	✓	✓	✓
protected	✓	✓	✓	✕
(default)	✓	✓	✕	✕
private	✓	✕	✕	✕

📌 **Example: Access Control**

```
// File: mypackage/Student.java
package mypackage;


public class Student {
    public String name = "John";        // accessible everywhere
    protected int age = 20;             // accessible within package + subclasses
    String course = "Java";             // default access (package only)
    private String password = "1234";   // private to this class


    public void showDetails() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Course: " + course);
        System.out.println("Password: " + password);
    }
}
```

4. Importing Packages

- To use classes from another package, we **import** them.
- Syntax:

```
import package_name.class_name;    // imports specific class
import package_name.*;             // imports all classes
```

📌 **Example: Importing User-Defined Package**

```
// File: TestPackage.java
import mypackage.MyPackageClass;  // importing specific class


public class TestPackage {
    public static void main(String[] args) {
        MyPackageClass obj = new MyPackageClass();
        obj.displayMessage();
    }
}
```

👉 Compile & Run:

```
javac TestPackage.java
java TestPackage
```

5. Using Built-in Packages

📌 Example: Using java.util package

```
import java.util.Date;


public class BuiltInPackageDemo {
    public static void main(String[] args) {
        Date today = new Date();
        System.out.println("Current date & time: " + today);
    }
}
```

---

6. Advantages of Packages

- Provides **modularity** (organized code).
- Avoids **naming conflicts**.
- Provides **access control** (via modifiers).
- Supports **reusability**.
- Makes project development easier in teams.

✅ Summary:

- package keyword defines a package.
- Use **access modifiers** to control visibility.
- import keyword helps in reusing classes from other packages.
- Java has **built-in** as well as **user-defined** packages.

Exception Handling:

- Defining Exceptions
- Errors
- Hierarchy of Exception Class
- Built-in Exceptions

---

1. What is an Exception?

- **Exception** → An **unwanted or unexpected event** that occurs during program execution and disrupts the normal flow of instructions.
- Examples:
  - Dividing a number by zero
  - Accessing an array element out of bounds
  - Invalid type casting
  - File not found

👉 In Java, exceptions are **objects** that represent runtime errors.

---

2. Difference Between Errors and Exceptions

- **Errors**: Serious problems that occur **beyond the control of the programmer**.
  - Example: OutOfMemoryError, StackOverflowError
  - Not recoverable in most cases.
- **Exceptions**: Problems that occur due to **programming mistakes or external factors**.
  - Example: NullPointerException, ArithmeticException
  - Can be handled using **try-catch** blocks.

📌 Example: Error vs Exception

```
public class ErrorVsException {

    public static void main(String[] args) {

        // Exception Example

        try {

            int result = 10 / 0;  // ArithmeticException

        } catch (Exception e) {

            System.out.println("Exception caught: " + e);

        }


        // Error Example (Uncomment to see - may crash program)

        // recursiveMethod();   // causes StackOverflowError

    }


    static void recursiveMethod() {

        recursiveMethod();  // infinite recursion

    }

}
```

3. Hierarchy of Exception Class

All exceptions in Java are subclasses of **Throwable**.

📌 **Exception Hierarchy:**

```
          Object
            |
         Throwable
        /        \
     Error       Exception
       /                    \
    (Checked Exceptions)  (Unchecked Exceptions)
```

- **Throwable** → Root class of all exceptions and errors.
- **Error** → Represents serious system problems (e.g., OutOfMemoryError).
- **Exception** → Represents runtime problems that can be handled.
    - **Checked Exceptions**: Must be handled at compile-time. (e.g., IOException, SQLException)
    - **Unchecked Exceptions (Runtime Exceptions)**: Occur at runtime, not checked by compiler. (e.g., ArithmeticException, NullPointerException)

4. Built-in Exceptions in Java

Java provides many predefined exceptions.

✅ **Checked Exceptions (Compile-time)**

- IOException → Error in input/output operations.
- SQLException → Database access error.
- ClassNotFoundException → Class not found when loading dynamically.
- FileNotFoundException → File not available.

✅ **Unchecked Exceptions (Runtime)**

- ArithmeticException → Divide by zero.
- NullPointerException → Accessing methods/variables of null object.
- ArrayIndexOutOfBoundsException → Invalid array index access.
- NumberFormatException → Converting invalid string to number.

5. Examples of Exceptions

📌 Example 1: **ArithmeticException**

```
public class ArithmeticDemo {
    public static void main(String[] args) {
        try {
            int result = 100 / 0;   // Division by zero
        } catch (ArithmeticException e) {
            System.out.println("Exception caught: " + e);
        }
    }
}
```

📌 Example 2: **NullPointerException**

```
public class NullPointerDemo {
    public static void main(String[] args) {
        try {
            String str = null;
            System.out.println(str.length());  // NullPointerException
        } catch (NullPointerException e) {
            System.out.println("Exception caught: " + e);
        }
    }
}
```

📌 Example 3: **ArrayIndexOutOfBoundsException**

```
public class ArrayExceptionDemo {
    public static void main(String[] args) {
        try {
            int[] arr = {10, 20, 30};
            System.out.println(arr[5]);  // invalid index
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Exception caught: " + e);
        }
    }
}
```

📌 Example 4: **Checked Exception (FileNotFoundException)**

```
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class CheckedExceptionDemo {
    public static void main(String[] args) {
        try {
            FileReader file = new FileReader("data.txt"); // may not exist
        } catch (IOException e) {
            System.out.println("Checked Exception caught: " + e);
        }
    }
}
```

---

6. Key Points

- **Errors** → Serious, unrecoverable.
- **Exceptions** → Recoverable runtime problems.
- **Throwable** → Parent class of both.
- **Checked exceptions** → Must be handled (compiler checks).
- **Unchecked exceptions** → Runtime only.

✅ **Summary for Students**

- Exceptions help handle unexpected situations in programs.
- Errors are system failures, exceptions are program failures.
- Use **try-catch-finally** to handle exceptions.
- Java provides many **built-in exceptions** for common problems.