➢ ***What is a Package in Java:***

Java packages are a key organizational tool that allows you to group related classes, interfaces, and sub-packages together to structure your code efficiently. They help manage large projects, avoid naming conflicts, and control access. Let's dive into the details step-by-step:

1. What Are Java Packages?

A **package** in Java is essentially a namespace that organizes classes, interfaces, and sub-packages into logical groups. It acts like a folder structure for your Java code.

Benefits of Packages:

1. **Avoid Name Conflicts**: Two classes with the same name can coexist in different packages.
2. **Organize Code**: Group related classes and interfaces together for better readability.
3. **Access Control**: Packages help define access levels with `protected` and `default` visibility.
4. **Reusability**: Reuse classes and interfaces in multiple programs.

Types of Packages:

1. **Built-in Packages**: Provided by Java (e.g., `java.util`, `java.io`, `java.lang`).
2. **User-defined Packages**: Created by the programmer.

2. How to Create a Package?

1. Syntax for Declaring a Package:

At the top of your Java file, use the `package` keyword to declare the package:

`java code:`

- `package mypackage;`

2. Steps to Create a Package:

1. **Create a Directory**:
   - The directory name should match the package name.
   - For example, if your package is `mypackage`, create a folder named `mypackage`.
1. **Write Your Class**:
   - Place your Java file in the directory.
   - Declare the package at the top of the file:

- ***Java code for creating the java package:***

- package mypackage;

```
public class MyClass {
        public void displayMessage() {
                System.out.println("Hello from MyClass in mypackage!");
        }
}
```

3. Compile the Class:

- Use the javac command with the -d option to specify the base directory for the package:

  CODE:

  - javac -d . MyClass.java

3. How to Import Packages?

Importing an Entire Package:

To use all classes in a package, use the `import` keyword followed by the package name:

`java` code:

- import mypackage.*;

Importing a Specific Class:

To import only one class:

Java code:

- import mypackage.MyClass;

Using Fully Qualified Names (Without Import):

You can use a class from a package without importing it by specifying its fully qualified name:

- mypackage.MyClass obj = new mypackage.MyClass();

4. How to Use Functions or Methods from Packages?

Example:

1. **Create a Package**:

- Java code:

```
package utilities;


public class MathUtils {
    public static int add(int a, int b) {
        return a + b;
    }
}
```

2. Use a Method in Another Class:

- `Java code:`

```
import utilities.MathUtils;


public class Main {
    public static void main(String[] args) {
        int result = MathUtils.add(5, 3);
        System.out.println("Result: " + result);
    }
}
```

5. Package Hierarchy

Packages can have a hierarchy, like a directory structure.

Example:

If you have the package `com.example.myapp`:

- **Root Package**: `com`
- **Sub-Package**: `example`
- **Sub-Sub-Package**: `myapp`

Directory Structure:

com/

example/

myapp/

MyAppClass.java

- Defining a Class in a Hierarchical Package:

```
package com.example.myapp;


public class MyAppClass {
    public void display() {
        System.out.println("Hello from MyAppClass!");
    }
}
```

6. Access Control in Packages

Access Modifiers in Relation to Packages:

Modifier	Same Class	Same Package	Subclass (Other Package)	World (Other Packages)
public	**Yes**	**Yes**	**Yes**	**Yes**
protected	**Yes**	**Yes**	**Yes**	**No**
default	**Yes**	**Yes**	**No**	**No**
private	**Yes**	**No**	**No**	**No**

7. Built-In Packages

Some commonly used built-in packages:

1. **`java.lang`**: Contains classes like `String`, `Math`, `Integer`, etc. (Automatically imported).
2. **`java.util`**: Contains utility classes like `ArrayList`, `HashMap`, `Scanner`, etc.
3. **`java.io`**: Contains classes for input and output, like `File`, `BufferedReader`, etc.
4. **`java.net`**: Contains classes for networking, like `Socket`, `URL`, etc.
5. **`java.sql`**: Contains classes for database access.

8. Interfaces and Classes in Packages

Classes in Packages:

A package can contain one or more classes. Classes within a package are related in functionality.

Interfaces in Packages:

An interface can also belong to a package and is used for defining a contract that implementing classes must follow.

Example:

- Code:

```
package mypackage;


public interface MyInterface {
        void sayHello();
}
```

Implementation in Another Class:

- **`Code:`**

```
package mypackage;


public class MyImplementation implements MyInterface {
   @Override
   public void sayHello() {
      System.out.println("Hello from MyImplementation!");
   }}
```