Step-by-Step Process for Java Packages

A **package** in Java is used to organize classes and interfaces systematically to avoid name conflicts and enhance modularity. Here's how to **create and use Java packages** step by step.

Step 1: Create a Package

1. Open your **Java project** in VS Code or any other IDE.
2. Inside the `src` folder (or any directory), create a new **package**.
    - The folder structure should match the package name.
    - Example: If your package is named `mypackage`, create a folder named `mypackage`.
3. Inside the package, create a new Java file, e.g., `MyClass.java`.
4. At the **top** of the file, declare the package:

```
package mypackage; // Declaring the package

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

Step 2: Compile the Package

To compile the package, open the **command prompt (terminal)** and navigate to the parent directory of `mypackage`.

Run the following command:

```
javac -d . MyClass.java
```

- The `-d .` flag tells the compiler to place the compiled `.class` file in the correct package directory.
- The package folder (`mypackage`) will now contain `MyClass.class`.

Step 3: Use the Package in Another Class

1. Create another Java file **outside** the package, e.g., `Main.java`.
2. Import the package at the top using `import mypackage.MyClass;`.
3. Use the class in your program:

```
import mypackage.MyClass; // Importing the package

public class Main {
    public static void main(String[] args) {
        MyClass obj = new MyClass();
        obj.displayMessage(); // Calling the method
    }
}
```

Step 4: Compile and Run the Program

1. Compile both Java files:
2. `javac Main.java`
3. Run the program:
4. `java Main`

   Output:

   ```
   Hello from MyClass in mypackage!
   ```

Step 5: Using `import` vs. Fully Qualified Name

- **Using `import` statement** (recommended):
- `import mypackage.MyClass;`
- **Using fully qualified name** (without `import`):
- `public class Main {`
- `    public static void main(String[] args) {`
- `        mypackage.MyClass obj = new mypackage.MyClass();`
- `        obj.displayMessage();`
- `    }`
- `}`

Step 6: Creating Subpackages (Optional)

You can create **subpackages** by nesting folders:

```
src/
 ├── mypackage/
 │    ├── MyClass.java
 │    ├── subpackage/
 │         ├── SubClass.java
```

To declare a **sub-package**, use:

```
package mypackage.subpackage;
```

To **import** and use it:

```
import mypackage.subpackage.SubClass;
```

✓ Packages help in code organization and avoiding name conflicts.
✓ Use `import` to access classes easily.
✓ Compile using `-d .` to maintain package structure.

```java
package mypackage; // Declaring the package

public class My_Package {
    public static float  i;
    public void displayMessage() {
            float j=(i*9.5f)/100;
            i=i+j;
        System.out.println("Hello from MyClass in mypackage!"+i);
    }
}
```

```java
import java.util.Scanner;
import mypackage.My_Package;
public class hello {
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        My_Package m=new My_Package();
        System.err.println("Enter Ammount");
         m.i=sc.nextFloat();

        m.displayMessage();




    }
}
```