Understanding Packages and Importing Interfaces in Java

1. What is a Package in Java?

A **package** in Java is a way to organize related classes and interfaces together. It acts as a namespace to avoid class name conflicts and improve code maintainability. Java provides built-in packages (like `java.util`, `java.io`), but developers can also create their own.

Advantages of Using Packages

- Avoids **name conflicts** by grouping related classes.
- Provides **access control** using different access modifiers.
- Makes code **modular**, improving maintainability and reusability.
- Supports **better organization** of large projects.

2. Defining a Package in Java

To create a package, use the `package` keyword at the beginning of a Java file.

Example: Creating a Package (`mypackage`)

```
package mypackage;  // Declaring a package

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

Notes:

- The file **MyClass.java** must be saved in a directory named **mypackage**.
- The package declaration **must be the first statement** in the file.

Compiling and Running a Java Package

1. Compile the Java file

Navigate to the directory containing the package and run:

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

This creates a folder `mypackage` and stores `MyClass.class` inside it.

2. Running the Java file

Move to the parent directory of `mypackage` and run:

```
java mypackage.MyClass
```

3. Importing Packages in Java

To use a class from a different package, you need to **import** it using:

1. **Import a specific class** → `import package_name.ClassName;`
2. **Import all classes from a package** → `import package_name.*;`
3. **Fully qualified name** (without `import` statement) → `package_name.ClassName obj = new package_name.ClassName();`

Example: Importing a Class

File: `mypackage/MyClass.java`

```
package mypackage;

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

File: `Main.java` (in another package)

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

public class Main {
    public static void main(String[] args) {
        MyClass obj = new MyClass();
        obj.showMessage();
    }
}
```

4. Interfaces in Java Packages

An **interface** in Java defines a contract with abstract methods that must be implemented by a class.

Example: Defining and Importing an Interface

File: `mypackage/MyInterface.java`

```
package mypackage;

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

File: `mypackage/ImplementingClass.java`

```
package mypackage;

public class ImplementingClass implements MyInterface {
    public void greet() {
        System.out.println("Hello from ImplementingClass!");
    }
}
```

File: `Main.java` (Using the Interface in Another Package)

```
import mypackage.MyInterface;
import mypackage.ImplementingClass;

public class Main {
    public static void main(String[] args) {
        MyInterface obj = new ImplementingClass();
        obj.greet();
    }
}
```

5. Importing Static Members (`static import`)

Java allows importing **static methods or variables** from a class using `import static`.

Example: Using `static import`

```
import static java.lang.Math.*;   // Importing all static methods of Math
class

public class Main {
    public static void main(String[] args) {
        System.out.println("Square root of 25: " + sqrt(25)); // No need to
write Math.sqrt()
    }
}
```

6. Summary

- **Packages** group related classes and interfaces.
- Use `import package_name.ClassName;` or `import package_name.*;` to access classes.
- **Interfaces** define contracts and are implemented by classes.
- **Static imports** allow using static members without class names.

Understanding Package Accessibility in Java

Overview of Java Packages

A **package** in Java is a namespace that organizes related classes and interfaces. It helps in avoiding name conflicts and improves code maintainability. Java provides built-in packages (`java.util`, `java.io`, etc.), but developers can also create their own packages.

Access Modifiers and Their Scope

Access modifiers in Java determine the visibility of classes, methods, and variables across different packages. There are **four types of access modifiers**:

Modifier	**Same Class**	**Same Package**	**Subclass (Different Package)**	**Non-Subclass (Different Package)**
private	□ Yes	□ No	□ No	□ No
default	□ Yes	□ Yes	□ No	□ No
protected	□ Yes	□ Yes	□ Yes (only via subclassing)	□ No
public	□ Yes	□ Yes	□ Yes	□ Yes

Understanding Each Access Modifier in Detail

1. `private` (Most Restrictive)

- The **private** modifier allows access only within the same class.
- **Not accessible in the same package, subclass, or any other package**.
- **Use Case**: Used for encapsulation to hide implementation details.

Example:

```
package packageA;

public class Example {
    private void privateMethod() {
        System.out.println("This is a private method.");
    }

    public void test() {
        privateMethod(); // Accessible within the same class
    }
}
```

□ Not Accessible in Another Class:

```
package packageA;

public class Test {
    public static void main(String[] args) {
        Example obj = new Example();
        // obj.privateMethod(); // Compilation error
    }
}
```

---

2. `default` (Package-Private)

- If no access modifier is specified, it defaults to **package-private**.
- **Accessible within the same package** but **not outside the package**.
- **Use Case**: Useful when classes are closely related and shouldn't be accessed externally.

Example:

```
package packageA;

class DefaultExample {
    void defaultMethod() {
        System.out.println("This is a default method.");
    }
}
```

Accessible in the Same Package:

```
package packageA;

public class Test {
    public static void main(String[] args) {
        DefaultExample obj = new DefaultExample();
        obj.defaultMethod(); // □  Works because it's in the same package
    }
}
```

□ Not Accessible in Another Package:

```
package packageB;

import packageA.DefaultExample; // □  Compilation error

public class Test {
    public static void main(String[] args) {
        DefaultExample obj = new DefaultExample();
        // obj.defaultMethod(); // □  Not accessible outside package
    }
}
```

---

3. `protected` (Package + Inherited)

- **Accessible within the same package**.
- **Accessible in a subclass even if it's in a different package**.
- **Not accessible in non-subclasses from another package**.
- **Use Case**: Useful when you want to provide access to subclasses but hide details from unrelated classes.

Example in Parent Class (`packageA`)

```
package packageA;

public class Parent {
    protected void protectedMethod() {
        System.out.println("This is a protected method.");
    }
}
```

Accessing from Subclass in Another Package (`packageB`)

```
package packageB;

import packageA.Parent;

public class Child extends Parent {
    public void test() {
        protectedMethod(); // □ Works because it's inherited
    }
}
```

□ Not Accessible from Non-Subclass in Another Package

```
package packageB;

import packageA.Parent;

public class Test {
    public static void main(String[] args) {
        Parent obj = new Parent();
        // obj.protectedMethod(); // □ Compilation error (Not accessible)
    }
}
```

---

4. `public` (Least Restrictive)

- **Accessible from anywhere**.
- **No restrictions across packages or classes**.
- **Use Case**: Used for methods, classes, or variables that should be accessible globally.

Example:

```
package packageA;

public class PublicExample {
    public void publicMethod() {
        System.out.println("This is a public method.");
    }
}
```

Accessible in Any Package

```
package packageB;

import packageA.PublicExample;

public class Test {
    public static void main(String[] args) {
        PublicExample obj = new PublicExample();
        obj.publicMethod(); // □  Works from any package
    }
}
```

Practical Implementation: Package Accessibility Example

Step 1: Create a Base Class (`packageA.BaseClass`)

```
package packageA;

public class BaseClass {
    private void privateMethod() {
        System.out.println("Private  Method  -  Only  accessible  within  this
class");
    }

    void defaultMethod() {
        System.out.println("Default  Method  -  Accessible  within  the  same
package");
    }

    protected void protectedMethod() {
        System.out.println("Protected  Method  -  Accessible  within  the  same
package and by subclasses");
    }

    public void publicMethod() {
        System.out.println("Public Method - Accessible everywhere");
    }
}
```

Step 2: Create a Subclass in Another Package (`packageB.SubClass`)

```
package packageB;

import packageA.BaseClass;

public class SubClass extends BaseClass {
    public void accessMethods() {
        // privateMethod(); // □ Not accessible
        // defaultMethod(); // □ Not accessible (package-private)
        protectedMethod(); // □ Accessible via subclassing
        publicMethod();    // □ Accessible
    }
}
```

Step 3: Create a Non-Subclass in Another Package (`packageB.NonSubClass`)

```
package packageB;

import packageA.BaseClass;

public class NonSubClass {
    public void accessMethods() {
        BaseClass obj = new BaseClass();

        // obj.privateMethod(); // □ Not accessible
        // obj.defaultMethod(); // □ Not accessible
        // obj.protectedMethod(); // □ Not accessible (not a subclass)
        obj.publicMethod(); // □ Accessible
    }
}
```

Step 4: Main Class to Run the Program (`packageB.MainClass`)

```
package packageB;

public class MainClass {
    public static void main(String[] args) {
        // Testing access in subclass
        SubClass subObj = new SubClass();
        subObj.accessMethods();

        // Testing access in non-subclass
        NonSubClass nonSubObj = new NonSubClass();
        nonSubObj.accessMethods();
    }
}
```

Expected Output:

```
Trying to access methods from subclass in a different package:
✔ Public Method is accessible.
✔ Protected Method is accessible via subclassing.
✘ Default Method is NOT accessible outside the package.
✘ Private Method is NOT accessible anywhere except its own class.

Trying to access methods from a non-subclass in a different package:
✔ Public Method is accessible.
✘ Protected Method is NOT accessible (unless subclassed).
✘ Default Method is NOT accessible.
✘ Private Method is NOT accessible.
```

Key Takeaways

1. **Private members** are not accessible outside the class.
2. **Default members** are only accessible within the same package.
3. **Protected members** are accessible within the same package and through inheritance in other packages.
4. **Public members** are accessible from anywhere.