What is a Package?

A **package** in Java is a **container for classes, interfaces, and sub-packages**.
It helps:

- Organize code logically
- Prevent **name conflicts**
- Control **visibility/access** to classes

🔹 Why Use Packages?

1. **Namespace Management** – Avoid class name clashes (e.g., your List class vs. Java's java.util.List)
2. **Access Protection** – Classes, methods, and variables can have controlled visibility using public, private, protected, or default.
3. **Code Organization** – Makes large projects maintainable and modular.

🔹 Declaring a Package

At the very **top of the Java file**, use:

```
package MyPackage;
```

- This tells the compiler that this class belongs to the MyPackage package.
- The file must be saved inside a folder named MyPackage (case-sensitive).

🔹 Example – Package Declaration

Directory Structure:

```
MyPackage/

└── Hello.java
```

Hello.java

```
package MyPackage;
public class Hello {
    public void greet() {
        System.out.println("Hello from package!");
    }
}
```

🔹 Importing a Package

To use the Hello class in another class:

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

✅ Only public classes and members from the package can be accessed this way.

🔹 Package = Naming + Visibility Mechanism

- **Naming**: Helps uniquely identify classes (e.g., java.util.Scanner)
- **Visibility**: Only public members are accessible outside the package

---

🔹 Package Hierarchy

Packages can be nested:

package java.awt.image;

- Must be stored in matching folder structure:

arduino

CopyEdit

java/

└── awt/

  └── image/

    └── YourClass.class

❗ Folder names must **exactly match** the package names (case-sensitive).

🔹 How Java Finds Your Package

1. **Default Search Location**: Current working directory
2. **CLASSPATH Variable**: Set a global path to include external packages

3. **-classpath Option**: Pass path explicitly during compilation/run

```
javac -classpath . MyProgram.java
```

```
java -classpath . MyProgram
```

🔹 Access Control in Packages

Access Modifier	Same Class	Same Package	Subclass (other pkg)	Other Classes
private	✅	❌	❌	❌
default	✅	✅	❌	❌
protected	✅	✅	✅	❌
public	✅	✅	✅	✅

🔑 When a package is **imported**, only the public items are available to **non-subclasses** in **other packages**.

🔹 What is static in Java?

The static keyword is used to define members (variables, methods, blocks, and nested classes) that belong to the class rather than instances (objects).

🔸 Key Characteristics of static Members:

- Belong to the class, not to any object.
- Can be accessed without creating an object.
- main() method is static so that the JVM can call it without creating an object of the class.
- Cannot use this or super keywords inside a static method.

🔹 Static Variables (Class Variables)

- Declared using static inside a class but outside any method.
- Shared among **all instances** of the class.
- Initialized only once when the class is loaded.

- Can be accessed via class name or object.

```
class Counter {
    static int count = 0;
    Counter() {
        count++;
        System.out.println(count);
    }
}
Output:
CopyEdit
1
2
3
```

◆ **Static Methods**

- Can be called using the class name: ClassName.methodName()
- Cannot access **non-static variables** or **methods** directly
- Cannot use this or super keywords

```
class Human {
    String message = "Hello";

    public static void display(Human human) {
        System.out.println(human.message);  // access through
object
    }

    public static void main(String[] args) {
        Human h = new Human();
        h.message = "Kunal's message";
        display(h);
    }
}
```

◆ **Static Block**

- Used to initialize static variables.
- Runs **once** when the class is first loaded.

```
class UseStatic {
  static int a = 3;
  static int b;

  static {
        System.out.println("Static block initialized.");
        b = a * 4;
    }

    static void meth(int x) {
        System.out.println("x = " + x);
        System.out.println("a = " + a);
        System.out.println("b = " + b);
    }

    public static void main(String[] args) {
        meth(42);
    }
}
```

Output:

```
makefile
CopyEdit
Static block initialized.
x = 42
a = 3
b = 12
```

◆ **Static Classes**

- Only **nested classes** (inner classes) can be declared static
- A static nested class **does not** need a reference to its outer class

```
public class Outer {
    static class Inner {
        String name;

        Inner(String name) {
            this.name = name;
        }
    }

    public static void main(String[] args) {
        Inner a = new Inner("Kunal");
        Inner b = new Inner("Rahul");

        System.out.println(a.name); // Kunal
        System.out.println(b.name); // Rahul
    }
}
```

◆ **Key Rules of static**

Rule	**Explanation**
Static methods can only call other static methods	Cannot access instance (non-static) members
Cannot use this or super	Because static is not tied to any object
Static variables are shared	Common storage across all instances
Static block executes once	On class loading
Static inner classes are allowed	Acts like a top-level class