02 (OOP Internals)

1. Java Packages

→ A package in Java is a named container (or folder) that groups together related classes, interfaces, sub-packages, and other types.

A package acts like a namespace. It helps avoid name conflicts and gives a structured way to organize your code.

(i) why use packages?

a. Avoid Naming conflicts → Two classes named Student can exist in different packages without error.

b. Code organization → You can group classes by purpose: models, services, Controllers, etc.

c. Access Control → ~~You can group classes by purpose~~ You can control what classes and methods are visible to others.

d. Reusability & Modularity → Easy to use/import specific parts of your project in other apps or modules.

(ii) Types of packages.

(1) Built-in packges

⇒ provided by Java (eg. java.lang, java.util, java.io, java.net, etc).

Date
Page No.

(2) user-defined packages
- Created by developers to structure projects.

(iii) import Statement
In java, the import statement is used to bring other classes, interfaces, or entire packages into scope, so that you can use them without writing their full name (fully qualified name) every time.

Example ⇒ import java.util.Scanner; // ⇒ imports Scanner
~~File : Message.java~~ // class from java.util package.

2. Static in Java
→ Static means belonging to the class, not to any specific object.
• It can be used with -
(a) Static variable → Shared among all instances (objects) of class.
(b) Static Method → Can be called without creating an object. Can only access static stuff.
(c) Static Block → Used to initialize static data. Executes only once when class is loaded.
(d) Static class → Nested class that can be accessed without creating an object of outer class.

Example

Date
Page No.

```
Class Example {
• Static int number;   // static variable belongs to class, shared by all objects.
•   Example (int number){
Example.number = number;   // Assign static variable using Classname (not 'this')
  ~~static~~ }
• Static void showNumber( ) {
  Sout(number);   // static ~~variable~~ method can access Static variable
   }
 }
      public class Main {
       psum (String[] args) {
•        Sout(Example.number) //Access static variable using Class Name.
.       new Example (10);   //output → 10
•      Example.showNumber( );  // call static method with class name
                               // (no Object needed)
      new Example (25);
      Example.showNumber;  output → 25
      }
    }
```

(i) Non-Static Member Inside Static

⇒ Static methods cannot access non-static variables or methods directly.

. Non-static member belong to objects. so, must create an object inside the static method

Example

```
public class Main {
   int x = 10;   // Non-Static variable
psvm (String[] args) {
XX   sout(x);   X// Error : Can't access not-static variable
                 // directly inside a static method

•   Main obj = new Main();  // need to create object,
•    sout(obj.x);      // to acces non-static through object.
  }
}
```

(ii) Static Inside Non-Static

=> we can access static members from non-static method - even without creating an object.

Example ->

```
public class Main{
   Static String msg = "Hello";  //static variable
 public void printmsg() {
•   sout(msg);  // Direct access to static variable
 }
 psvm(String[] args){
•  Main obj = new Main();  // Create object to call
                            // non-static method
• obj.printMsg();  // Non-static method can access static variable

 }
}
```

(iii) this keyword Inside static

-> this refers to the current object, but static methods do not belong to any object, so this cannot be used inside static methods.

Example ->

```
class Sample {
        int num = 100;
        static void display() {
XX  Sout(this.num) //Error : Cannot use 'this' in static method
XX   // Explanatation -> No object exists in static context,
                        // so 'this' is meaningless.
        }
    }
```

(iv) Initialization of static variable

-> static variables are shared by all objects of a class and belong to the class itself, not to any object. They can be initialized in two main ways.

(a) Direct Initialization

-> ~~static~~ static int x = 10;

(b) Using a static block

```
static {
      //Initialization code
}
```

Example →

```
        Class Config{
static int version;          //static variable declaration(default value = 0)
   static {                  // static block to initialize static values
version = 10;                // This block runs only once the class is
   }                         //loaded
 }

     public class Main {
       psvm(String[] args){
       sout(Config.version);   //Accessing static variable via
       }                       // class name(no object needed)
 }
```

(3) Inner Classes

⇒ An inner class is a class defined inside another Class. It helps group classes that logically belong together and can access the outer class's members.

• Types of Inner Classes

(i) Member (Non-Static) → Normal class inside another class (needs outer class object).
(ii) Static Nested Class → Inner Class with static keyword. (no outer class object)
(iii) Local Inner Class → Defined inside a method. (need outer class object)
(iv) Anonymous Inner Class → Class without a name (usually for Interface). (No outer class object needed)

// Examples will be done later // Available in practice folder lateron.

Date
Page No.

4. ~~Singleton Class~~ Singleton Class

→ A singleton class is a class that allows only one object to be created in the entire JVM.

• Rules to Create Singleton Class

(i) Make the Constructor private
(ii) Create a static object of the class inside the class
(iii) provide a public static method to return the object.

Example

It is in Github practice folder