SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NAAC – UGC with 'A' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

23CSB101

OBJECT ORIENTED PROGRAMMING

UNIT II

INHERITANCE, PACKAGES AND INTERFACES

STATIC, NESTED and INNER CLASSES

By

M.Kanchana

Assistant Professor/CSE

STATIC NESTED CLASSES

- A static nested class is a class inside another class, but it is declared as **static.**
- A static nested class does not depend on an instance of the outer class.
- It can be accessed without creating an object of the outer class.

A Car User Manual is tied to a specific car model, not a single car.

Toyota Camry Manual applies to all Toyota Camry cars not just one.

STATIC NESTED CLASSES

- It can access static data members of outer class including private.
- Static nested class cannot access non-static (instance) data member or method.

```
class OuterClass {
   static class SNC {
 void display() {
        System.out.println("Inside Static Nested Class");}}}

   public class Main {
   public static void main(String[] args) {
      // Creating an object of StaticNestedClass
      OuterClass.SNC obj = new OuterClass.SNC();
      obj.display();
      // Output: Inside Static Nested Class
   }
}
```

INNER CLASSES (Non-static nested class)

- An inner class is a class that is defined inside another class.
- In Java, an **inner class** is used when a class is **strongly associated** with another class.

Types

1. Member inner class
2. Anonymous inner class
3. Local inner class

INNER CLASSES (Non-static nested class)

Member inner class:

- A regular inner class that is a member of the outer class.
- A regular inner class that is a **permanent part of** the outer class.
- It can access all members (including private ones) of the outer class.
- An Engine is always part of a Car, so it makes sense to define Engine inside the Car class.
- An Engine is always a part of a Car. Without an engine, the car cannot function.

INNER CLASSES

Syntax: For declaring Inner classes

```
class Outer
{
//code
class Inner
{
//code
}
}
```

INNER CLASSES

Instantiating an Inner Class:

Two Methods:

Instantiating an Inner class from outside the outer class:

Syntax:

OuterClass.InnerClass objectName=OuterObj.new InnerClass();

Instantiating an Inner Class from Within Code in the Outer Class:

Syntax:

InnerClassName obj=new InnerClassName();

INNER CLASSES

```
class Car {
    private String model;
    Car(String model) {
        this.model = model;
    }
    class Engine {
        void start() {
            System.out.println(model + "'s engine is starting...");
        }}}

   public class Main {
       public static void main(String[] args) {
           Car myCar = new Car("Tesla Model S");
           Car.Engine engine = myCar.new Engine();
                   engine.start();  }}
```

INNER CLASSES

2. Anonymous inner class:

- A class **without a name**, declared and instantiated **only once** for immediate use.

- A Mechanic is called to fix the car but does not become a permanent part of the car. The mechanic is created when needed and disappears afterward.

INNER CLASSES

```
class Car {
    void repair() {
        System.out.println("Car needs repairs.");}}

public class Main {
    public static void main(String[] args) {
Car mechanic = new Car() {
            @Override
            void repair() {
                System.out.println("Mechanic is repairing the car.");
            }
        };
        mechanic.repair();  //  Output: Mechanic is repairing the car.
    }
}
```

INNER CLASSES

3. Local inner class:

- A class declared inside a method, making it local to that method. It exists only when the method is called.
- It is like a **temporary helper class** that exists only when needed.
- Car has **a spare tire**, but you **only use it during emergencies**.A Spare Tire is used only when needed (if a tire is punctured) and then removed.

INNER CLASSES

```
class Car {
    void breakdown() {
        class SpareTire {
          void use() {
             System.out.println("Using the spare tire to continue driving.");}}

        SpareTire tire = new SpareTire();
       tire.use(); // Output: Using the spare tire to continue driving.}}

        public class Main {
            public static void main(String[] args) {
                Car myCar = new Car();
                myCar.breakdown();  // Calls the method where the local class is used
            }
       }
```

INNER CLASSES

Advantages :

Logical Grouping : An Engine cannot exist without a Car, so it makes sense to keep it inside Car.

Encapsulation: The Engine class can directly access Car's private members (e.g., model).

Better Code Organization: It keeps related functionality together instead of creating multiple separate classes.

THANK YOU