07_2 Nested Class

Object-Oriented Programming

Kind of Nested Class

- **Outer class** includes the nested class
- **Nested class** is defined within outer class (Static nested, Inner, Local Inner)

```java
public class OuterClass {
    static class SNClass { } // Static nested class
    class InnerClass { }     // (non-static) Inner class
    void someMethod() {
        class LIClass { }    // Local Inner Class
        LIClass liObject = new LIClass(); // only used within the method
    }
}
public class OuterClassDemo {
    public static void main(String[] args) {
        OuterClass.SNClass snObject = new OuterClass.SNClass();
        OuterClass outObject = new OuterClass(); // create outer object first
        OuterClass.InnerClass inObject = outObject.new InnerClass();
        outObject.someMethod();
    }
}
```

Example) Nested Class (1/3)

```java
public class OuterClass {

    static class SNClass { // Static nested class
        void display() {
            System.out.println("Inside static nested class");
        }
    }

    class InnerClass { // Inner class (member inner class)
        void display() {
            System.out.println("Inside inner class");
        }
    }
```

Example) Nested Class (2/3)

```java
public class OuterClass {

    . . .

    void myMethod() {
        class LIClass { // Local Inner Class
            void display() {
                System.out.println("Inside local inner class");
            }
        }
        LIClass liObject = new LIClass(); // should be used within the method
        liObject.display();
    }
}
```

Example) Nested Class (3/3)

```java
public class OuterClassDemo {
    public static void main(String[] args) {

        // directly create OuterClass.SNClass
        OuterClass.SNClass snObject = new OuterClass.SNClass();
        snObject.display(); // OUTPUT: “Inside static nested class”

        OuterClass outObject = new OuterClass(); // create outer object first

        // using outObject.new to create innerClass's object
        OuterClass.InnerClass inObject = outObject.new InnerClass();
        inObject.display(); // OUTPUT: “Inside inner class”

        outObject.myMethod(); // OUTPUT: “Inside local inner class”
    }
}
```

Example) (Non-static) Inner Class

```
public class AClass {
    public class BClass {
        public class CClass {  }
    }
    public static void main(String[] args) {
        AClass aObject = new AClass();
        AClass.BClass bObject = aObject.new BClass();
        AClass.BClass.CClass cObject = bObject.new CClass();
    }
}
```

Rules for Nested Class

- **Name of an inner class**
  - cannot be reused inside the outer class
- **Private inner class**
  - cannot be accessed by name outside the the outer class
  - should be accessed through the public (package) method
- **Private variables and methods** of inner and outer classes
  - can access of each other by name

Example) Private Inner Class

```
class OuterClass {
    private class InnerClass { }
    void createInnerObject() {
        InnerClass inner = new InnerClass();
    }
}

public class Main {
    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        outer.createInnerObject();  // OK
        // OuterClass.InnerClass inner = outer.new InnerClass();
        // compile error!! cannot access from outside directly
    }
}
```

Example) Outer, Inner Private Members

```
class OuterClass {
    private String outerPrivateVar = "Outer Private Variable";
    class InnerClass {
        private String innerPrivateVar = "Inner Private Variable";
        void accessOuterClass() {
            System.out.println(outerPrivateVar); // Outer’s private access, OK
        }
    }
    void accessInnerClass() {
        InnerClass inner = new InnerClass();
        System.out.println(inner.innerPrivateVar);// Inner’s private access, OK
    }
}
```

Anonymous Class

- Only need to implement an interface once
  - implemented class is **not reused** elsewhere
- Easier to understand
  - class implementation is just near the variable
- Implement callback method
  - used in GUI applications such as button click

Example) Anonymous Class

```
interface Computer {
    void compute();
}
public class AnonymousClassDemo {
    public static void main(String[] argc) {
        Computer computer1 = new Computer() { // anonymous class
                                  public void compute() {
                                    System.out.println("This is the computer1");
                                  }
                                };
        Computer computer2 = new Computer() {
                                  public void compute() {
                                    System.out.println("This is the computer2");
                                  }
                                };
        computer1.compute(); // OUTPUT: “This is the computer1”
        computer2.compute(); // OUTPUT: “This is the computer2”
    }
}
```