UNIT-3 (partial)

Prepared By:
Dr. Deepak Kumar Sharma
Asst. Professor (SG)
SoCS UPES Dehradun

UNIT-III

Unit III: Nested Classes, Exceptions, Multithreading & IO Streams

Nested Classes, Types of Nested Classes, Exception Handling, Exception Handlers, Concurrent Programming, The Thread Class and Runnable Interface, Thread Priorities, Synchronization, Java’s I/O Streams, Byte Streams and Character Streams, FileWriter, FileReader.

Java Nested and Inner Class

- Nested Class: Defining a class within another class.

```
class OuterClass {
    // ...
    class NestedClass {
        // ...
    }
}
```

Two types of nested classes can be created in Java:

•Non-static nested class (inner class)
- Member inner class
- Anonymous inner class
- Local inner class

•Static nested class

Non-Static Nested Class (Inner Class)

- Non-static nested classes are known as inner classes.
- It has access to members of the enclosing class (outer class).
- *Must instantiate the outer class first, in order to instantiate the inner class.*

Need of inner class:

- Sometimes users need to program a class in such a way so that no other class can access it. Therefore, it would be better if you include it within other classes.

Advantage of Java inner classes

- Nested classes represent a particular type of relationship that is **it can access all the members (data members and methods) of the outer class,** including private.
- Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.
- **Code Optimization**: It requires less code to write.

Java Member Inner class

- A non-static class that is created inside a class but outside a method is called **member inner class**.
- It is also known as a **regular inner class**.
- It can be declared with access modifiers like public, default, private, and protected.

Syntax:

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

```
class TestMemberOuter{
 private int data=30;
 class Inner{
  void msg(){System.out.println("data is "+data);}
 }
 public static void main(String args[]){
  TestMemberOuter obj=new TestMemberOuter();
//to create an object of the member inner class
//OuterClassReference.new MemberInnerClassConstructor();
TestMemberOuter.Inner in=obj.new Inner();
  in.msg();
 }
}
```

Output: data is 30

Java Anonymous inner class

- Java anonymous inner class is an inner class without a name and for which only a single object is created.
- An anonymous inner class can be useful when making an instance of an object with certain "extras" such as overloading methods of a class or interface, *without having to actually subclass a class*.
- Java Anonymous inner class can be created in two ways:
  - Class (may be abstract or concrete).
  - Interface

Example- Java Anonymous inner class

```
class Animal {
     void makeSound(){}
}

public class TestAnonymousInner {
    public static void main(String args[]) {
      //Anonymous Inner Class
        Animal animal = new Animal() {
            void makeSound() {
                System.out.println(“Check Animal ");
            }
        };

        animal.makeSound();
    }
}
```

```
abstract class Person{
  abstract void eat();
}
class TestAnonymousInner{
 public static void main(String args[]){
//Anonymous Inner Class
  Person p=new Person(){
  void eat(){System.out.println("nice fruits");}
  };
  p.eat();
 }
}
```

Internal Working:

- A class is created, but its name is decided by the compiler, which extends the Person class and provides the implementation of the eat() method.
- An object of the Anonymous class is created that is referred to by 'p,' a reference variable of Person type.

Java anonymous inner class example using interface

```
interface Eatable{
 void eat();
}
class TestAnnonymousInner1{
 public static void main(String args[]){
 Eatable e=new Eatable(){
  public void eat(){System.out.println("nice fruits");}
 };
 e.eat();
 }
}
```

Output: nice fruits

Java Local inner class

- A class i.e., created inside a method, is called local inner class in java.
- Local Inner Classes are the inner classes that are defined inside a block. Generally, this block is a method body.
- Sometimes this block can be a for loop, or an if clause.
- Local Inner classes are not a member of any enclosing classes.
- They belong to the block they are defined within, due to which local inner classes cannot have any access modifiers associated with them.
- However, they can be marked as final or abstract.
- These classes have access to the fields of the class enclosing it.
- If you want to invoke the methods of the local inner class, you must instantiate this class inside the method.

Java local inner class example

```
public class localInner{
 private int data=30;//instance variable
 void display(){
  class Local{
   void msg(){System.out.println(data);}
  }
  Local l=new Local();
  l.msg();
 }
 public static void main(String args[]){
  localInner obj=new localInner();
  obj.display();
 }
}
```

output:

```
30
```

Example- Inner Class

```
class CPU {
    double price;
    // nested class
    class Processor{
        // members of nested class
        double cores;
        String manufacturer;
        double getCache(){
            return 4.3;
        }
    }
    // nested protected class
    protected class RAM{
        // members of protected nested class
        double memory;
        String manufacturer;
        double getClockSpeed(){
            return 5.5;
        }
    }
}
```

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

        // create object of Outer class CPU
        CPU cpu = new CPU();

       // create an object of inner class Processor using outer class
        CPU.Processor processor = cpu.new Processor();

        // create an object of inner class RAM using outer class CPU
        CPU.RAM ram = cpu.new RAM();
        System.out.println("Processor Cache = " + processor.getCache());
        System.out.println("Ram Clock speed = " + ram.getClockSpeed());
    }
}
```

Output:

```
Processor Cache = 4.3
Ram Clock speed = 5.5
```

Accessing Members of Outer Class within Inner Class

```
class Car {
    String carName;
    String carType;
    // assign values using constructor
    public Car(String name, String type) {
        this.carName = name;
        this.carType = type;
    }
    // private method
    private String getCarName() {
        return this.carName;
    }
// inner class
    class Engine {
        String engineType;
        void setEngine() {
           // Accessing the carType property of Car
            if(Car.this.carType.equals("4WD")){
                // Invoking method getCarName() of Car
                if(Car.this.getCarName().equals("Crysler")) {
                    this.engineType = "Smaller";
                } else {
                    this.engineType = "Bigger";
                }
            }else{
                this.engineType = "Bigger";
            }
        }
        String getEngineType(){
            return this.engineType;
        }
    }
}
```

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

// create an object of the outer class Car
        Car car1 = new Car("Mazda", "8WD");
        // create an object of inner class using the outer class
        Car.Engine engine = car1.new Engine();
        engine.setEngine();
        System.out.println("Engine Type for 8WD= " + engine.getEngineType());

        Car car2 = new Car("Crysler", "4WD");
        Car.Engine c2engine = car2.new Engine();
        c2engine.setEngine();
        System.out.println("Engine Type for 4WD = " + c2engine.getEngineType());
    }
}
```

- can access the members of the outer class by using **"this"** keyword.

Static Nested Class

- A static class inside another class.
- Static nested classes are not called static inner classes.
- Unlike inner class, a static nested class cannot access the member variables of the outer class. It is because the **static nested class** doesn't require you to create an instance of the outer class.
- It cannot access non-static data members and methods.
- It can access static data members of the outer class, including private.
- Static nested classes can include both static and non-static fields and methods.
- To access the static nested class, we don't need objects of the outer class.

Note: In Java, only nested classes are allowed to be static.

Static Nested class

```
class TestOuter1{
 static int data=30;

  static class Inner{
  void msg(){System.out.println("data is "+data);}
 }

public static void main(String args[]){

TestOuter1.Inner obj=new TestOuter1.Inner();

 obj.msg();
 }
}
```

Output:

```
data is 30
```

Static Nested Class

```
class Animal {

// inner class
  class Reptile {
    public void displayInfo() {
      System.out.println("I am a reptile.");
    }
  }

// static class
  static class Mammal {
    public void displayInfo() {
      System.out.println("I am a mammal.");
    }
  }
}
```

```
class Main {
  public static void main(String[] args) {
    // object creation of the outer class
    Animal animal = new Animal();

    // object creation of the non-static class
    Animal.Reptile reptile = animal.new Reptile();
    reptile.displayInfo();

    // object creation of the static nested class
    Animal.Mammal mammal = new Animal.Mammal();
    mammal.displayInfo();

  }
}
```

Output

```
I am a reptile.
I am a mammal.
```

Java static nested class example with a **static method**

```
public class TestOuter{
  static int data=30;
  static class Inner{
   static void msg(){System.out.println("data is "+data);}
  }
  public static void main(String args[]){
  TestOuter.Inner.msg();//no need to create the instance of static nested class
  }
}
```

CheckPoint: Accessing Non-static members

```
class Animal {
 static class Mammal {
  public void displayInfo() {
    System.out.println("I am a mammal.");
  }
 }

 class Reptile {
  public void displayInfo() {
    System.out.println("I am a reptile.");
  }
 }

 public void eat() {
  System.out.println("I eat food.");
 }
}
```

```
class Main {
 public static void main(String[] args) {
  Animal animal = new Animal();
  Animal.Reptile reptile = animal.new Reptile();
  reptile.displayInfo();

  Animal.Mammal mammal = new Animal.Mammal();
  mammal.displayInfo();
  mammal.eat();
 }
}
```

OUTPUT:

```
Main.java:28: error: cannot find symbol
    mammal.eat();
          ^
  symbol:   method eat()
  location: variable mammal of type Mammal
1 error
compiler exit status 1
```

Note: static nested classes can only access the class members (static fields and methods) of the outer class.

CheckPoint?

```
static class Animal {
 public static void displayInfo() {
   System.out.println("I am an animal");
 }
}

class Main {
 public static void main(String[] args) {
   Animal.displayInfo();
 }
}
```

Output

```
Main.java:1: error: modifier static not allowed here
static class Animal {
       ^
1 error
compiler exit status 1
```

Java Nested Interface

- An interface, i.e., declared within another interface or class, is known as a nested interface.
- The nested interfaces are used to group related interfaces so that they can be easy to maintain.
- The nested interface must be referred to by the outer interface or class. It can't be accessed directly.

Points to remember for nested interfaces

- The nested interface must be public if it is declared inside the interface, but it can have any access modifier if declared within the class.
- Nested interfaces are declared static

Syntax of nested interface

```
interface interface_name{
 ...
 interface nested_interface_name{
  ...
 }
}
```

← **within the interface**

```
class class_name{
 ...
 interface nested_interface_name{
  ...
 }
}
```

← **within the class**

Example of nested interface which is declared within the interface

```
public interface OuterInterface {
    void outerMethod();

    // Nested interface declared within OuterInterface
    interface NestedInterface {
        void nestedMethod();
    }
}

public class MyClass implements OuterInterface.NestedInterface {
    @Override
    public void nestedMethod() {
        System.out.println("Nested interface method implementation");
    }

    public static void main(String[] args) {
        MyClass myClass = new MyClass();
        myClass.nestedMethod();
    }
}
```

Output:

```
Nested interface method implementation
```

Example of nested interface which is declared within the class

```
class OuterClass {
    interface NestedInterface {
        void nestedMethod();
    }
}

class ImplementNestedInterface implements OuterClass.NestedInterface {
    public void nestedMethod() {
        System.out.println("Nested method implementation");
    }
}

public class Main {
    public static void main(String[] args) {
       ImplementNestedInterface instance = new ImplementNestedInterface();
       instance.nestedMethod();
    }
}
```

Output:

```
Nested interface method implementation
```

Can we define a class inside the interface?

```
interface M{
 class A{}
}
```

- Yes, if we define a class inside the interface, the Java compiler creates a **static nested class.**

```
interface MyInterface {
    void myMethod();

    static class MyStaticClass {
        void staticMethod() {
            System.out.println("This is a static nested class method.");
        }
    }
}

public class Main {
    public static void main(String[] args) {
        MyInterface.MyStaticClass staticObj = new MyInterface.MyStaticClass();
        staticObj.staticMethod(); // Outputs: This is a static nested class method.
    }
}
```