Nested Class

Nested Classes

- A nested class is a class within another class

```
class OuterClass {
    ...
    class NestedClass {
        ...
    }
}
public class OuterClassTest { }
```

top-level class

top-level class

- Nested classes are divided into two categories
  - non-static
    - ***inner classes***
  - static
    - ***static nested classes***

```
class OuterClass {
    ...
    class InnerClass {
        ...
    }

    static class StaticNestedClass {
        ...
    }
}
```

Reasons for using nested classes

- logically grouping classes that are only used in one place
- increasing encapsulation
- more readable and maintainable code

```
class OuterClass {
    ...
    private class NestedClass {
        ...
    }
}
```

Logical grouping of classes—If a class is useful to only one other class, then it is logical to embed it in that class and keep the two together.

encapsulation—inner class can be hidden from the outside world.

Association with Outer Class

- As with instance methods and variables, an **inner class** is associated with an instance of its enclosing class
- As with class methods and variables, a **static nested class** is associated with its outer class

```
class OuterClass {
    private String name ;
    public static int count =0 ;
    public String toString() {
        String msg = "Instance " + name ;
        return msg ;
    }
    public static int next() {
        return count++ ;
    }
}
```

```
class OuterClass {
    private class InnerClass { }
    public static class StaticNestedClass { }
    public String toString() {
        class LocalClass { }
        return (new LocalClass()).toString() ;
    }
    public static int next() {
        return StaticNestedClass.COUNT ;
    }
}
```

Inner Class

- An instance of InnerClass has direct access to the methods and fields of its enclosing instance.
- it cannot define any static members itself
- There are two special kinds of inner classes: **local classes** and **anonymous classes**

```
class OuterClass {
    class InnerClass {
        int inner = 2 ;
        // static int count = 0 ; //error
    }
    public String toString() {
        class LocalClass {
            int local = 0 ;
            final static int COUNT = 0 ; //constant
        }
    }
```

```
    InnerClass ic = new InnerClass() ;
    public static void main (String[] args) {
        // InnerClass i = new InnerClass() ; //error
        OuterClass oc = new OuterClass();
        InnerClass i = oc.new InnerClass() ;
        System.out.prinln(i.inner);  //2
    }
    public int getHash () {
        // LocalClass lc = new LocalClass() ; //error
        return (new InnerClass()).hashCode() ;
    }
}
```

Static Nested Class

- StaticNestedclass cannot refer directly to instance variables or methods defined in its enclosing class
- it can use them only through an object reference

```
class OuterClass {
    private String name ;

    static class StaticNestedClass {
        int id = 0 ;
        static int COUNT = 5 ;
    }
    public String toString() {
            String msg = "Instance " + name ;
            return msg ;
    }
```

```
    StaticNestedClass snc =  new StaticNestedClass() ;
    public static void main (String[] args) {
        StaticNestedClass s = new StaticNestedClass() ;
        System.out.println(s.COUNT); //5
    }
    public int getCount () {
        StaticNestedClass s = new StaticNestedClass() ;
        return s.COUNT ;
    }
}
```

Inner Class and Nested Static Class : An Example (1/3)

```
public class OuterClass {

    String outerField = "Outer field";
    static String staticOuterField = "Static outer field";

    class InnerClass {
        void accessMembers() {
            System.out.println(outerField);
            System.out.println(staticOuterField);
        }
    }

    static class StaticNestedClass {
        void accessMembers(OuterClass outer) {
            // Compiler error: Cannot make a static reference to the non-static
            //     field outerField
            // System.out.println(outerField);
            System.out.println(outer.outerField);
            System.out.println(staticOuterField);
        }
    }
```

Inner Class and Nested Static Class : An Example (2/3)

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

        System.out.println("Inner class:");
        System.out.println("------------");
        OuterClass outerObject = new OuterClass();
        OuterClass.InnerClass innerObject = outerObject.new InnerClass();
        innerObject.accessMembers();


        System.out.println("₩nStatic nested class:");
        System.out.println("--------------------");
        StaticNestedClass staticNestedObject = new StaticNestedClass();
        staticNestedObject.accessMembers(outerObject);


        System.out.println("₩nTop-level class:");
        System.out.println("--------------------");
        TopLevelClass topLevelObject = new TopLevelClass();
        topLevelObject.accessMembers(outerObject);
    }
}
```

```
Inner class:
------------
Outer field
Static outer field


Static nested class:
--------------------
Outer field
Static outer field
```

```
Top-level class:
--------------------
Outer field
Static outer field
```

Inner Class and Nested Static Class : An Example (3/3)

```
public class TopLevelClass {

    void accessMembers(OuterClass outer) {
        // Compiler error: Cannot make a static reference to the non-static
        //     field OuterClass.outerField
        // System.out.println(OuterClass.outerField);
        System.out.println(outer.outerField);
        System.out.println(OuterClass.staticOuterField);
    }
}
```

```java
public class Person {
    // 필수 매개변수
    private final String firstName;  2 usages
    private final String lastName;  2 usages

    // 선택 매개변수
    private final int age;  2 usages
    private final String address;  2 usages
    private final String phoneNumber;  2 usages

    // Person의 private 생성자
    private Person(Builder builder) {  1 usage
        this.firstName = builder.firstName;
        this.lastName = builder.lastName;
        this.age = builder.age;
        this.address = builder.address;
        this.phoneNumber = builder.phoneNumber;
    }

    @Override
    public String toString() {
        return "Person{" +
                "firstName='" + firstName + '\'' +
                ", lastName='" + lastName + '\'' +
                ", age=" + age +
                ", address='" + address + '\'' +
                ", phoneNumber='" + phoneNumber + '\'' +
                '}';
    }
```

```java
    // static 내부 클래스 Builder
    public static class Builder {  5 usages
        // 필수 매개변수
        private final String firstName;  2 usages
        private final String lastName;  2 usages

        // 선택 매개변수 - 기본값으로 초기화
        private int age = 0;  2 usages
        private String address = "";  2 usages
        private String phoneNumber = "";  2 usages

        // 필수 매개변수를 받는 생성자
        public Builder(String firstName, String lastName) {  1 usage
            this.firstName = firstName;
            this.lastName = lastName;
        }

        // 선택 매개변수에 대한 메서드들
        public Builder age(int age) {  1 usage
            this.age = age;
            return this;
        }

        public Builder address(String address) {  1 usage
            this.address = address;
            return this;
        }

        public Builder phoneNumber(String phoneNumber) {  1 usage
            this.phoneNumber = phoneNumber;
            return this;
        }

        // 최종적으로 Person 객체를 생성하는 build 메서드
        public Person build() {  1 usage
            return new Person( builder: this);
        }
    }
```

```java
    // 예제 실행
    public static void main(String[] args) {
        Person person = new Person.Builder("John", "Doe")
                .age(30)
                .address("123 Main St")
                .phoneNumber("123-456-7890")
                .build();

        System.out.println(person);
    }
}
```

Shadowing

- If a declaration of a type in a particular scope has **the same name** as another declaration in the enclosing scope, then the declaration *shadows* the declaration of the enclosing scope.

```
public class ShadowTest {
    public int x = 0;

    class FirstLevel {
        public int x = 1;

        void methodInFirstLevel(int x) {
            System.out.println("x = " + x);
            System.out.println("this.x = " + this.x);
            System.out.println("ShadowTest.this.x = " + ShadowTest.this.x);
        }
    }

    public static void main(String... args) {
        ShadowTest st = new ShadowTest();
        ShadowTest.FirstLevel fl = st.new FirstLevel();
        fl.methodInFirstLevel(23);
    }
}
```

```
x = 23
this.x = 1
ShadowTest.this.x = 0
```

Local Classes (1/2)

- You can define a class locally **inside a single method.**
- A local inner class can access the fields of their outer classes.
- In addition, a local inner class can access local variables, but they must be ***final*** or ***effectively final*** (there is never changed after they are initialized since Java 8).

```
public class LocalClassExample {
    static String regularExpression = "[^0-9]";

    public static void validatePhoneNumber(
        String number1, String number2) {
        final int numberLength = 10;
        // Valid in JDK 8 and later: (effectively final)
        // int numberLength = 10; //never changed
```

```
class PhoneNumber {        // local class
  String formattedNumber = null;
  PhoneNumber(String number){
    String currentNumber = number.replaceAll(
              regularExpression, "");
    if (currentNumber.length() == numberLength)
        formattedNumber = currentNumber;
    else
        formattedNumber = null;
  }
  public String getNumber() {
    return formattedNumber;
  }
}
```

Local Classes (2/2)

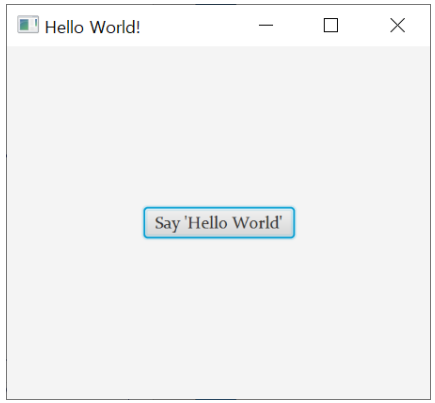
```
        PhoneNumber myNumber1 = new PhoneNumber(number1);
        PhoneNumber myNumber2 = new PhoneNumber(number2);

        if (myNumber1.getNumber() == null)
            System.out.println("First number is invalid");
        else
            System.out.println("First number is " + myNumber1.getNumber());
        if (myNumber2.getNumber() == null)
            System.out.println("Second number is invalid");
        else
            System.out.println("Second number is " + myNumber2.getNumber());

    } // end of validatePhoneNumber

    public static void main(String... args) {
        validatePhoneNumber("123-456-7890", "456-7890");
    }
}
```

```
First number is 1234567890
Second number is invalid
```

Anonymous Inner Classes

- You can define a inner class without name
- **<u>You can declare and instantiate a class at the same time</u>**
- It cannot have a constructor because it does not have a name
- They are like **<u>local classes</u>** except that they do not have a name (final)

```
public class HelloWorld extends Application {
    public static void main(String[] args) {
        launch(args);
    }
    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Hello World!");
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        primaryStage.setScene(new Scene(root, 300, 250));
        primaryStage.show();
    }
}
```



When to Use Nested Classes

- Nested class
  - logically grouping classes (e.g., Map, Entry)
  - to provide security for the important code
  - heler classes (e.g., builder pattern)

- Local class
  - nested method is required

- Anonymous class
  - just for instant use
  - only one object of the class is required
  - **listener interfaces or event handlers in GUI based programming**

Q&A