Object Oriented Programming

Topics to be covered today

- Static Class Members
- Inner Classes
- String Handling

Static Class Members

- Normally, the members of a class (its variables and methods) may be only used through the objects of this class.
- Static members are independent of the objects:
  - Variables
  - Methods
  - initialization block
- All declared with the static keyword.

Static Variable

- Static variable:

  static int a;

- Essentially, it a global variable shared by all instances of the class.
- It cannot be used within a non-static method.

Static Methods

- Static method:
  - static void meth() { … }
- Several restrictions apply:
  - can only call static methods
  - must only access static variables
  - cannot refer to this

Static Block

- Static block:
  - static { … }
- This is where the static variables are initialized.
- The block is executed exactly once, when the class is first loaded.

Example: Static

```
class UseStatic {
  static int a = 3;
  static int b;
  static void meth(int x) {
     System.out.print("x = " + x + " a = " + a);
     System.out.println(" b = " + b);
  }
  static {
     System.out.println("Static block initialized.");
     b = a * 4;
  }
  public static void main(String args[]) {
     meth(42);
  }
}
```

```
Static block initialized.
x = 42
a = 3
b = 12
```

Static Member Usage

- How to use static members outside their class?
- Consider this class:

```
class StaticDemo {
  static int a = 42;
  static int b = 99;
  static void callme() {
     System.out.println("a = " + a);
  }
}
```

Static Member Usage

- Static variables/method are used through the class name:

```
StaticDemo.a
StaticDemo.callme()
```

- Example

```
class StaticByName {
  public static void main(String args[]) {
     StaticDemo.callme();
     System.out.println("b = " + StaticDemo.b);
  }
}
```

Nested Classes

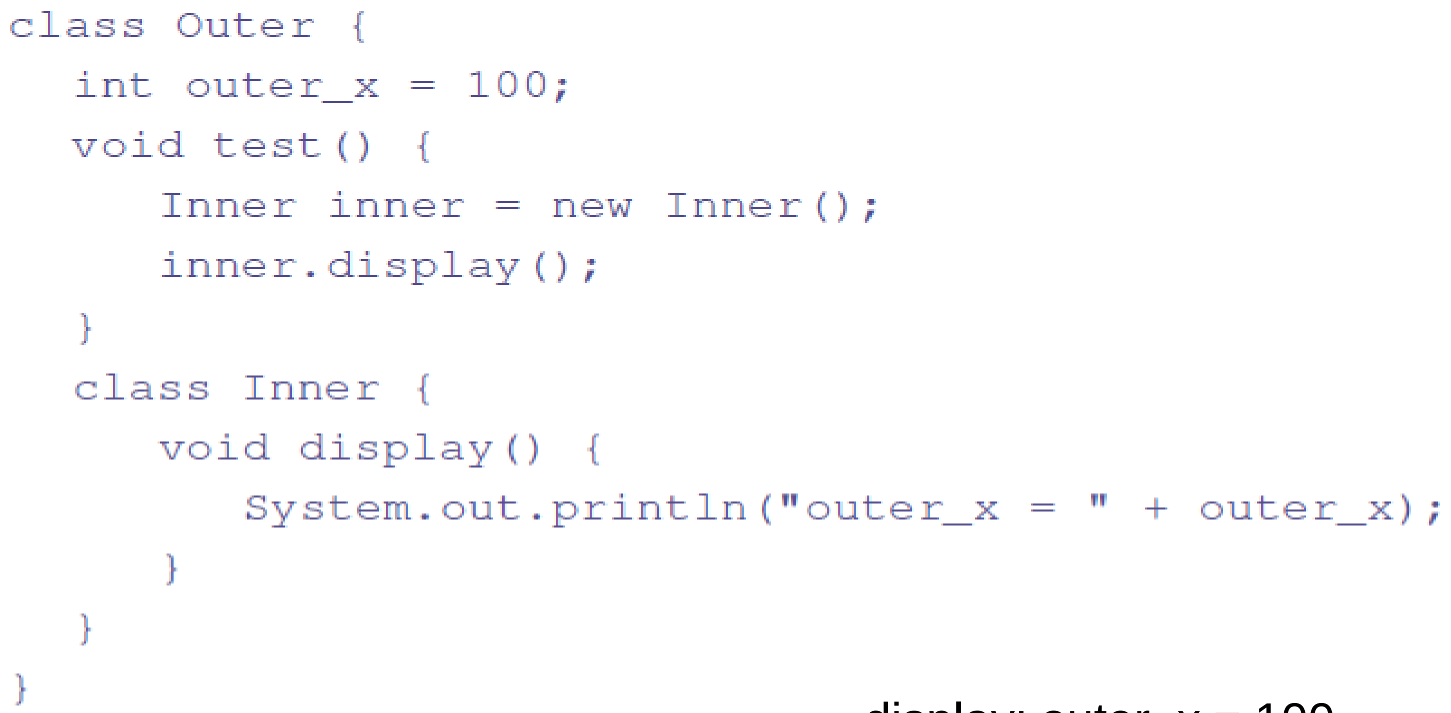
- It is possible to define a class within a class – nested class.
- The scope of the nested class is its enclosing class: if class B is defined within class A then B is known to A but not outside.
- Access rights:
  - a nested class has access to all members of its enclosing class, including its private members
  - the enclosing class does not have access to the members of the nested class without object creation

Types of Nested Classes

- There are two types of nested classes:
  - static – cannot access the members of its enclosing class directly, but through an object; defined with the static keyword
  - non-static – has direct access to all members of the enclosing class in the same way as other non-static member of this class so
- A static nested class is seldom used.
- A non-static nested class is also called an inner class.

Example: Inner Classes

- Outer has a variable out_x, an inner class Inner and a method test which creates an object of the Inner class and calls its display method:



```
class Outer {
   int outer_x = 100;
   void test() {
       Inner inner = new Inner();
       inner.display();
   }
   class Inner {
       void display() {
           System.out.println("outer_x = " + outer_x);
       }
   }
}
```

display: outer_x = 100

Example: Inner Classes

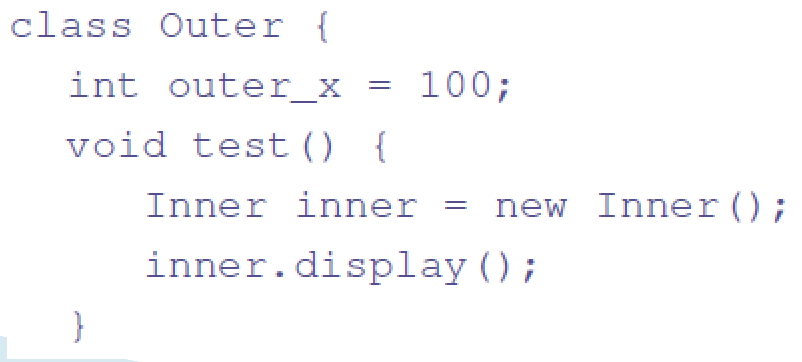
- A demonstration class to create an object of the Outer class and invoke the test method on this object:

```
class InnerClassDemo {
  public static void main(String args[]) {
     Outer outer = new Outer();
     outer.test();
  }
}
```

- The Inner class is only known within the Outer class. Any reference to Inner outside Outer will create a compile-time error.

Inner Members Visibility

- Inner class has access to all member of the outer class.
- The reverse is not true: members of the inner class are known only within the scope of the inner class and may not be used by the outer class.
- This is the Outer class with a variable, two methods and Inner class.
- The first method refers to the Inner class correctly through an object:



```
class Outer {
  int outer_x = 100;
  void test() {
      Inner inner = new Inner();
      inner.display();
  }
```

Inner Members Visibility

- Inner class declares variable y and refers to the Outer class variable:

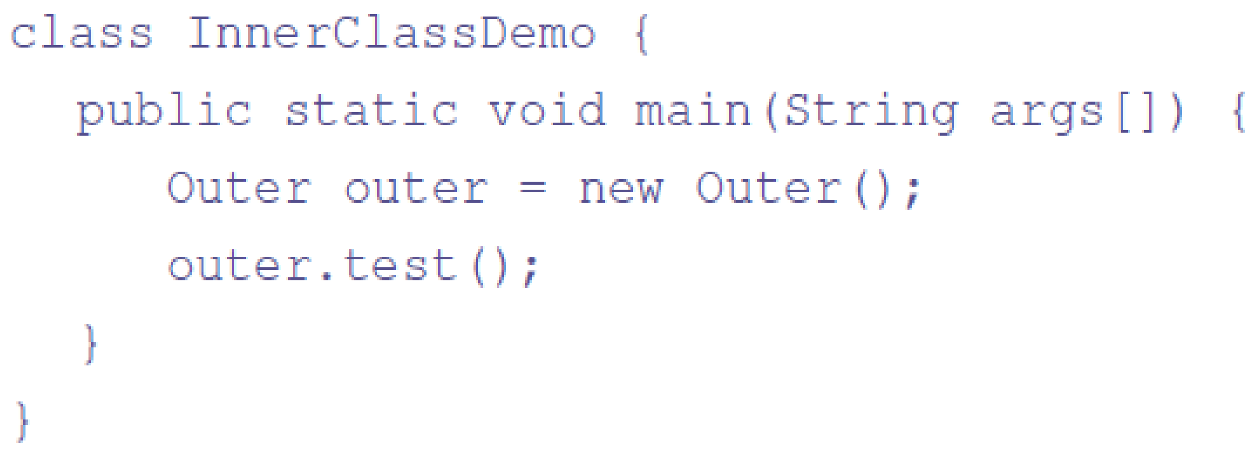
```
class Inner {
    int y = 10;
    void display() {
        System.out.println("outer_x = " + outer_x);
    }
}
```

- Showy method refers incorrectly to the Inner class’s y variable:

```
void showy() {
    System.out.println(y);
  }
}
```

Inner Members Visibility

- As a result, this program will not compile:



```
class InnerClassDemo {
  public static void main(String args[]) {
     Outer outer = new Outer();
     outer.test();
  }
}
```

Inner Class Declaration

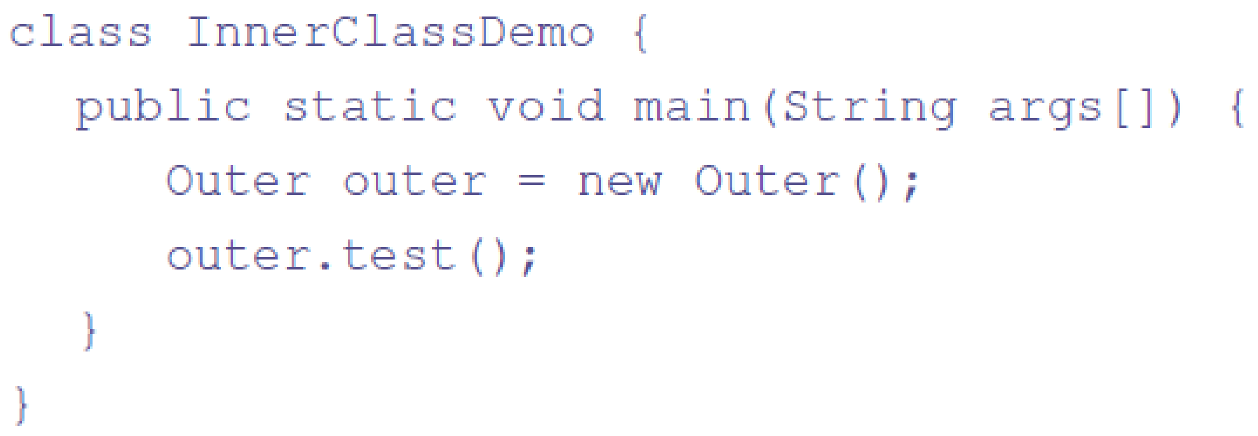
- So far, all inner classes were defined within the outer class scope.
- In fact, an inner class may be defined within any block scope.
- The following is an example of an inner class define within a for loop.

Inner Class Declaration Example

```
class Outer {
   int outer_x = 100;
   void test() {
      for (int i=0; i<10; i++) {
         class Inner {
            void display() {
               System.out.println("outer_x= " + outer_x);
            }
         }
         Inner inner = new Inner();
         inner.display();
      }
   }
}
```

Inner Class Declaration Example

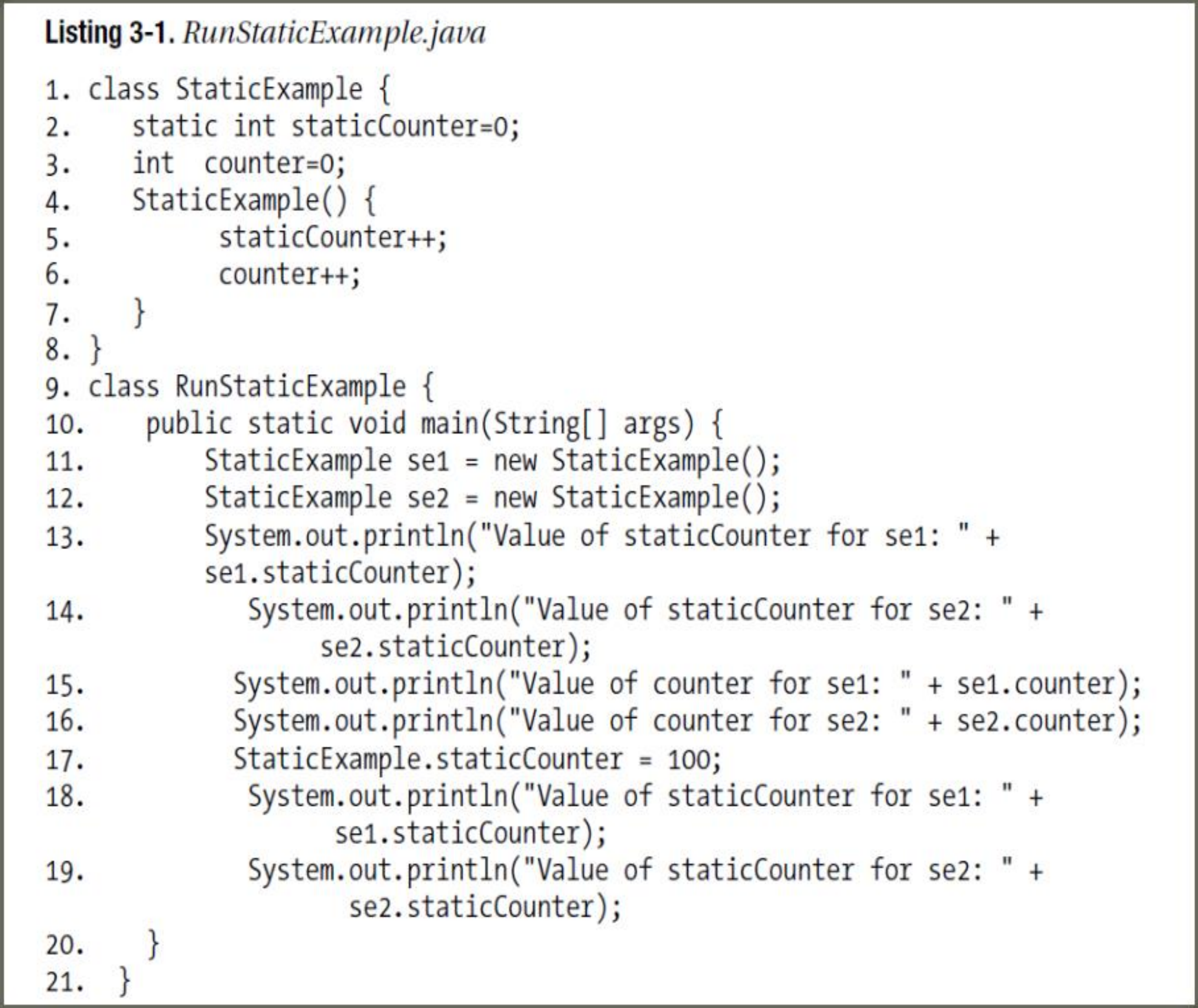
- A demonstration creates an Outer object and invokes a test method on it:



```
class InnerClassDemo {
  public static void main(String args[]) {
     Outer outer = new Outer();
     outer.test();
  }
}
```

The Static Method and Variable

- The static methods and variables are shared by all the instances of a class
 The static modifier may be applied to a variable, a method, and a block of code inside a method
 Because a static element of a class is visible to all the instances of the class, if one instance makes a change to it, all the instances see that change



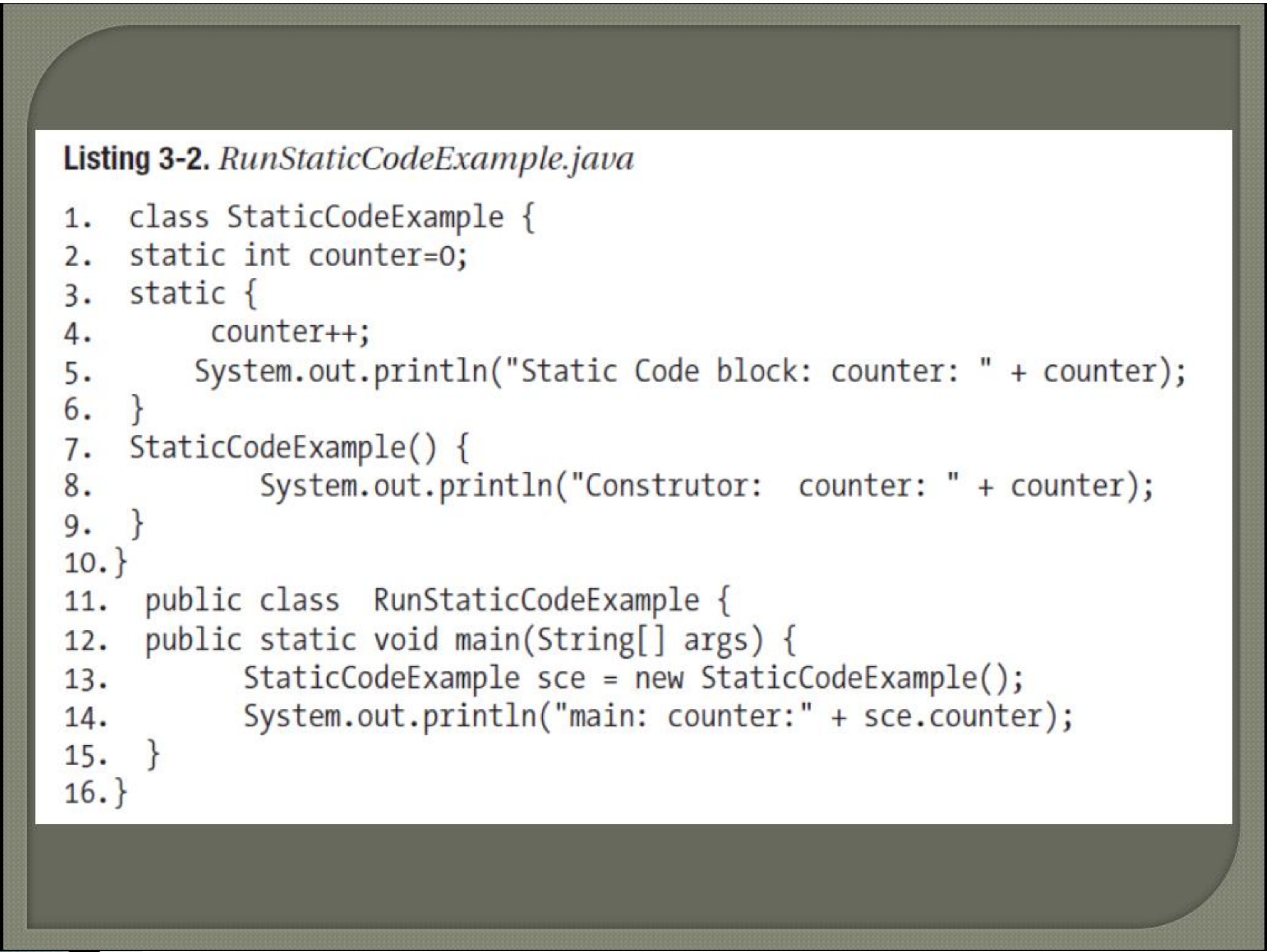
Listing 3-1. *RunStaticExample.java*

```
class StaticExample {
   static int staticCounter=0;
   int  counter=0;
   StaticExample() {
         staticCounter++;
         counter++;
   }
}
class RunStaticExample {
    public static void main(String[] args) {
        StaticExample se1 = new StaticExample();
        StaticExample se2 = new StaticExample();
        System.out.println("Value of staticCounter for se1: " +
        se1.staticCounter);
           System.out.println("Value of staticCounter for se2: " +
                se2.staticCounter);
          System.out.println("Value of counter for se1: " + se1.counter);
          System.out.println("Value of counter for se2: " + se2.counter);
          StaticExample.staticCounter = 100;
           System.out.println("Value of staticCounter for se1: " +
                 se1.staticCounter);
           System.out.println("Value of staticCounter for se2: " +
                  se2.staticCounter);
    }
}
```

- A static variable is initialized when a class is loaded, whereas an instance variable is initialized when an instance of the class is created
  A static method also belongs to the class. It can be called even before a single instance of the class exists
  A static method can only access the static members of the class

Static Code Block

- A class can also have a static code block outside of any method
  □The code block does not belong to any method, but only to the class
  □executed before the class is instantiated, or even before the method main() is called

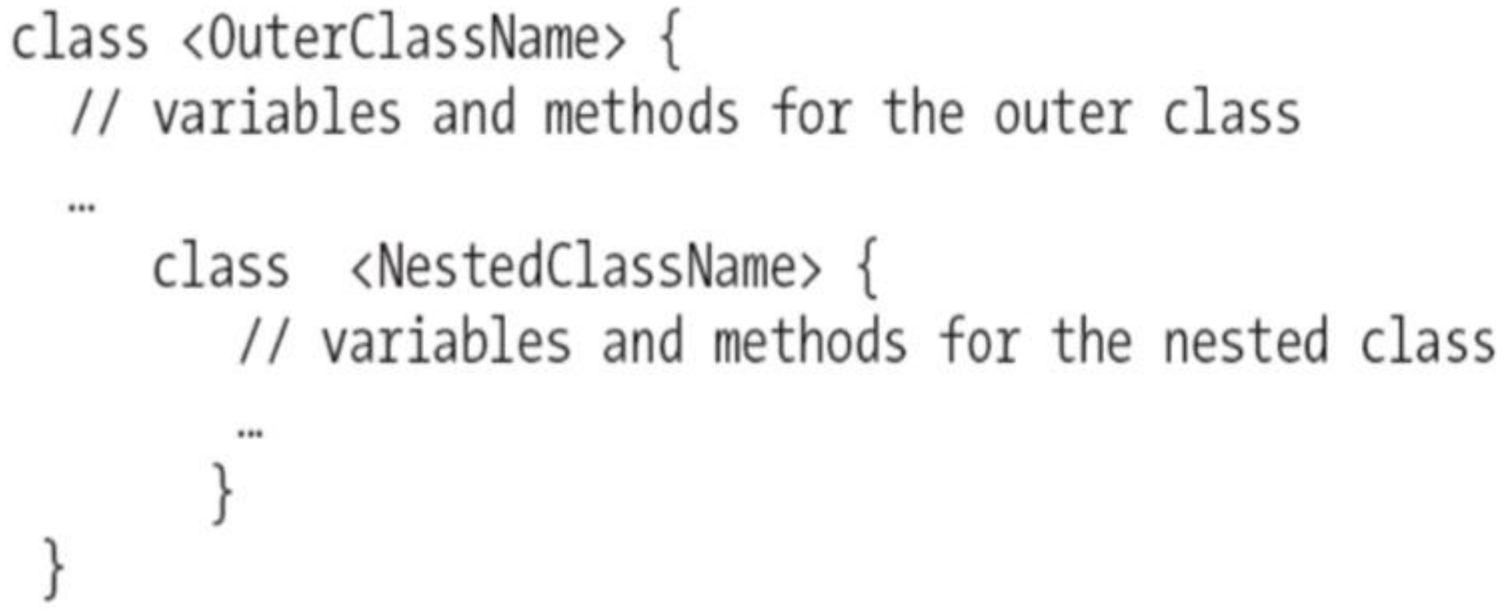


Listing 3-2. *RunStaticCodeExample.java*

```
class StaticCodeExample {
static int counter=0;
static {
     counter++;
    System.out.println("Static Code block: counter: " + counter);
 }
StaticCodeExample() {
        System.out.println("Construtor:  counter: " + counter);
 }
}
 public class  RunStaticCodeExample {
 public static void main(String[] args) {
       StaticCodeExample sce = new StaticCodeExample();
       System.out.println("main: counter:" + sce.counter);
 }
}
```

Nested Class

- allows you to define a class (like a variable or a method) inside a top-level class (outer class or enclosing class)



```
class <OuterClassName> {
  // variables and methods for the outer class
  ...
      class  <NestedClassName> {
        // variables and methods for the nested class
        ...
      }
}
```

- an instance of an inner class can only exist within an instance of its outer class

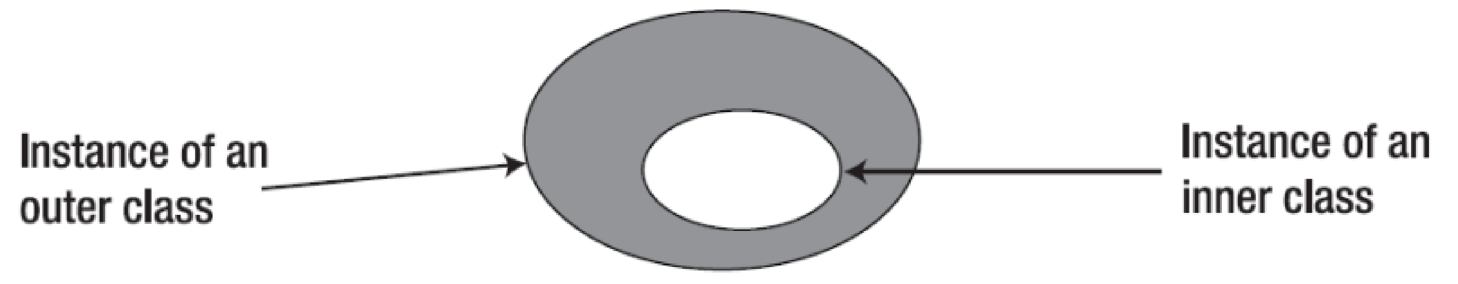


Figure 3-1. *The instance of an inner class has direct access to the instance variables and methods of an instance of the outer class.*

Listing 3-5. *TestNested.java*

```java
class TestNested {
 public static void main(String[] args) {
   String ext = "From external class";
   MyTopLevel mt = new MyTopLevel();
   mt.createNested();
   MyTopLevel.MyInner inner = mt.new MyInner();
   inner.accessInner(ext);
 }
}
  class MyTopLevel{
    private String top = "From Top level class";
     MyInner minn = new MyInner();
     public void createNested() {
          minn.accessInner(top);
     }
     class MyInner {
         public void accessInner(String st) {
            System.out.println(st);
         }
      }
  }
```

Questions?