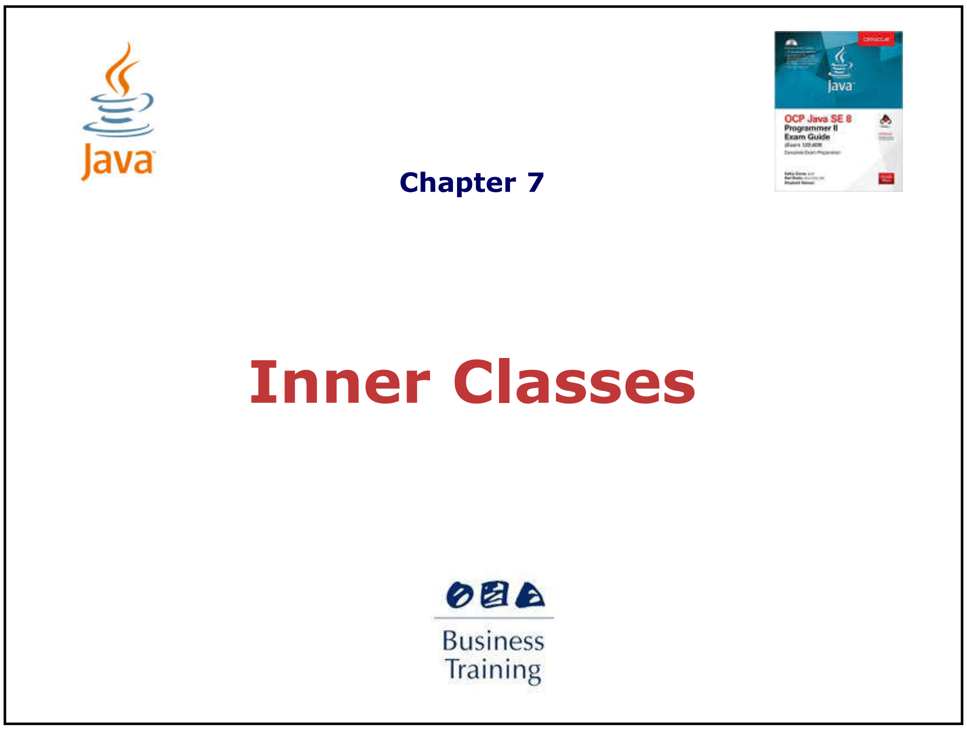
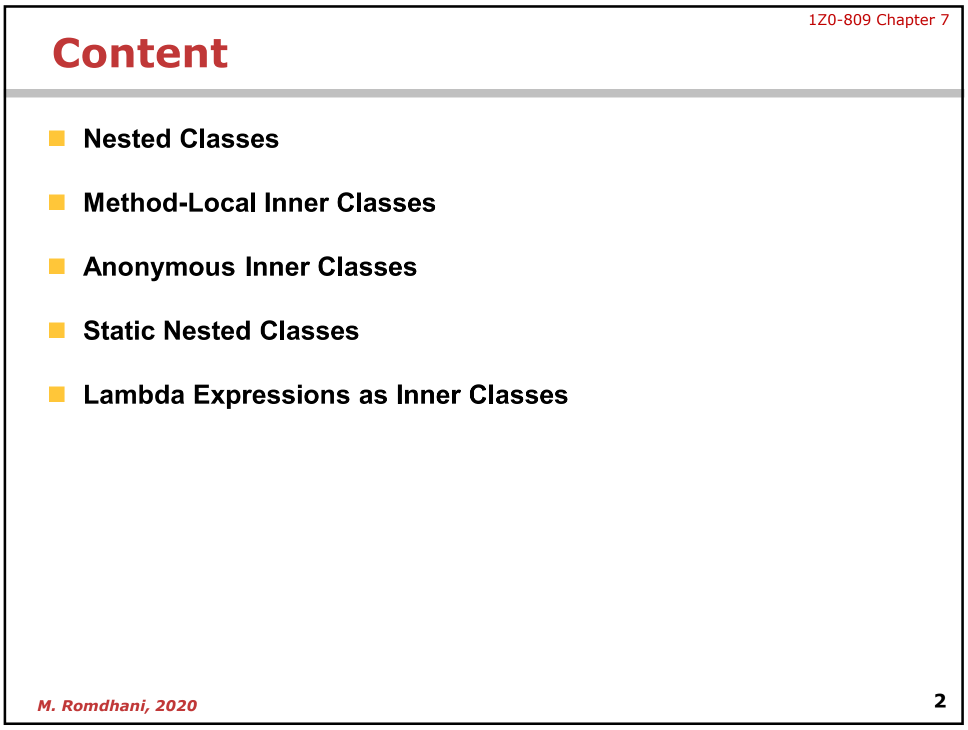


Chapter 7

Inner Classes

Business Training

1



1Z0-809 Chapter 7

Content

- Nested Classes
- Method-Local Inner Classes
- Anonymous Inner Classes
- Static Nested Classes
- Lambda Expressions as Inner Classes

M. Romdhani, 2020

2

2

Nested Classes

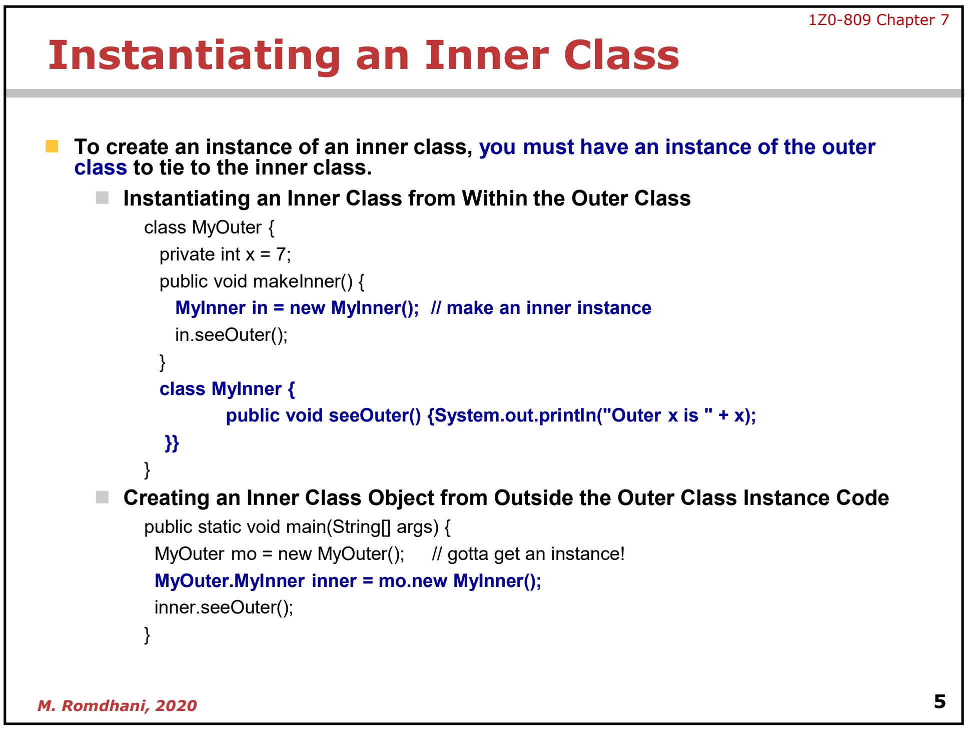
3

Coding "Regular" Inner Class

- **We use the term regular here to represent inner classes that are not:**
  1. Static
  2. Method-local
  3. Anonymous
- **Example**

```
class MyOuter {
     private int x = 7; // inner class definition
     class MyInner {
          public void seeOuter() {
                    System.out.println("Outer x is " + x);
          }
     } // close inner class definition
} // close outer class
```

4



1Z0-809 Chapter 7

Instantiating an Inner Class

- **To create an instance of an inner class, you must have an instance of the outer class to tie to the inner class.**
  - **Instantiating an Inner Class from Within the Outer Class**

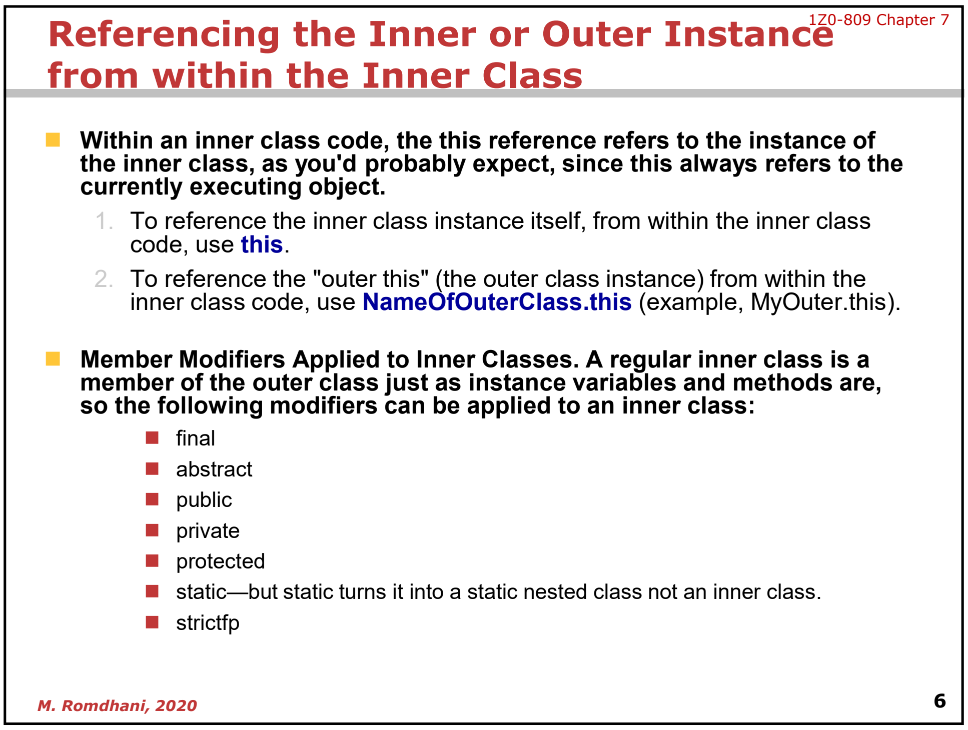
```
class MyOuter {
  private int x = 7;
  public void makeInner() {
    MyInner in = new MyInner();  // make an inner instance
    in.seeOuter();
  }
  class MyInner {
            public void seeOuter() {System.out.println("Outer x is " + x);
   }}
}
```

  - **Creating an Inner Class Object from Outside the Outer Class Instance Code**

```
public static void main(String[] args) {
  MyOuter mo = new MyOuter();     // gotta get an instance!
  MyOuter.MyInner inner = mo.new MyInner();
  inner.seeOuter();
}
```

M. Romdhani, 2020 **5**

5



1Z0-809 Chapter 7

Referencing the Inner or Outer Instance from within the Inner Class

- **Within an inner class code, the this reference refers to the instance of the inner class, as you'd probably expect, since this always refers to the currently executing object.**
  1. To reference the inner class instance itself, from within the inner class code, use **this**.
  2. To reference the "outer this" (the outer class instance) from within the inner class code, use **NameOfOuterClass.this** (example, MyOuter.this).
- **Member Modifiers Applied to Inner Classes. A regular inner class is a member of the outer class just as instance variables and methods are, so the following modifiers can be applied to an inner class:**
  - final
  - abstract
  - public
  - private
  - protected
  - static—but static turns it into a static nested class not an inner class.
  - strictfp

M. Romdhani, 2020 **6**

6

Method-Local Inner Classes

7

1Z0-809 Chapter 7

Method-Local Inner Classes

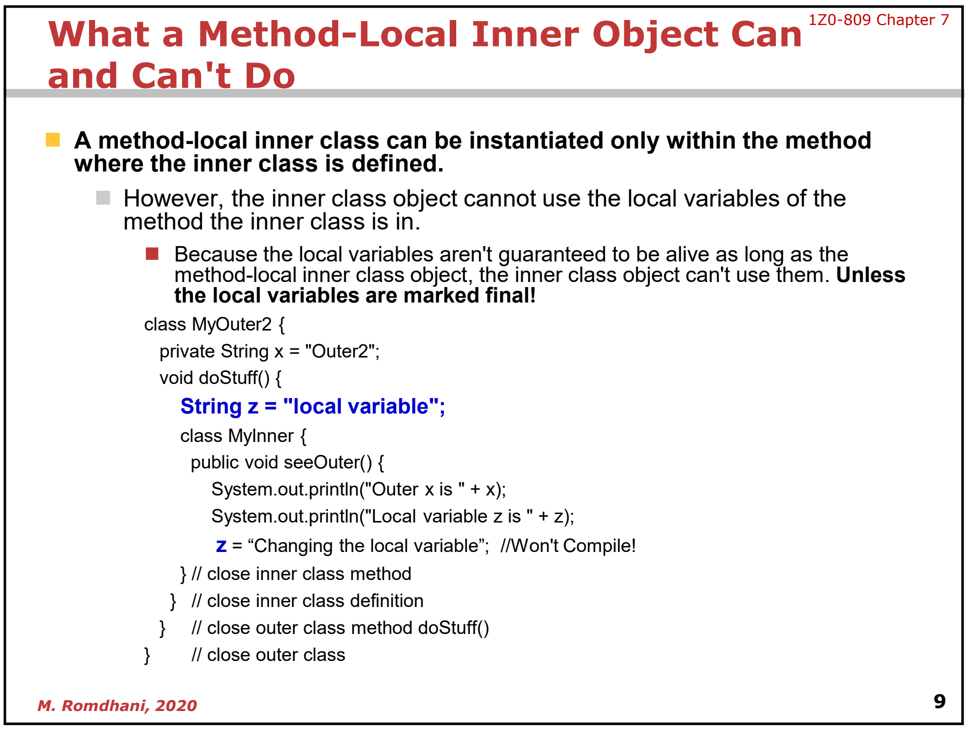
- **You can also define an inner class within a method. These inner classes are referred to as :**

```
class MyOuter2 {
    private String x = "Outer2";
    void doStuff() {
      class MyInner {
        public void seeOuter() {
          System.out.println("Outer x is " + x);
        } // close inner class method
      } // close inner class definition
    } // close outer class method doStuff()
} // close outer class
```

  - **So if you want to actually use the inner class (say, to invoke its methods), then you must make an instance of it somewhere within the method but below the inner class definition.**

M. Romdhani, 2020 **8**

8



1Z0-809 Chapter 7

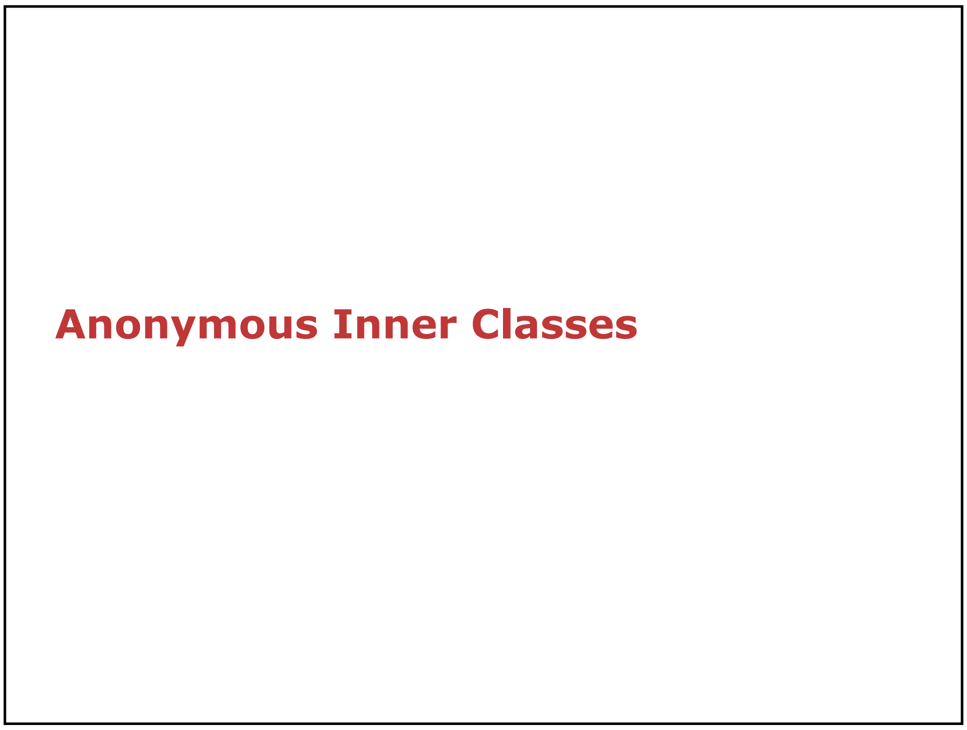
What a Method-Local Inner Object Can and Can't Do

- **A method-local inner class can be instantiated only within the method where the inner class is defined.**
  - However, the inner class object cannot use the local variables of the method the inner class is in.
    - Because the local variables aren't guaranteed to be alive as long as the method-local inner class object, the inner class object can't use them. **Unless the local variables are marked final!**

```
class MyOuter2 {
  private String x = "Outer2";
  void doStuff() {
     String z = "local variable";
     class MyInner {
       public void seeOuter() {
          System.out.println("Outer x is " + x);
          System.out.println("Local variable z is " + z);
          z = “Changing the local variable”;  //Won't Compile!
     } // close inner class method
   }  // close inner class definition
  }   // close outer class method doStuff()
}     // close outer class
```

M. Romdhani, 2020 9

9



Anonymous Inner Classes

10

Plain-Old Anonymous Inner Classes, Flavor One

- **We're going to look at the most unusual syntax you might ever see in Java; inner classes declared without any class name at all**

```
class Popcorn {
  public void pop() {
    System.out.println("popcorn");
   }
}
class Food {
  Popcorn p = new Popcorn() {   public void pop() {
                                    System.out.println("anonymous popcorn");
                                 }
                               };
}
```

  - The Popcorn reference variable refers not to an instance of Popcorn, but to an instance of an anonymous (unnamed) subclass of Popcorn.

Plain-Old Anonymous Inner Classes, Flavor One

- **The compiler will complain if you try to invoke any method on an anonymous inner class reference that is not in the superclass class definition.**

```
class Popcorn {
  public void pop() {
    System.out.println("popcorn");   }
}
class Food {
   Popcorn p = new Popcorn() {
                    public void sizzle () {
                         System.out.println("anonymous sizzling popcorn");     }
                    public void pop() {
                         System.out.println("anonymous popcorn");     }
               };
   public void popIt() {
       p.pop();    // OK, Popcorn has a pop() method
       p.sizzle(); // Not Legal! Popcorn does not have sizzle()
}}
```

Plain-Old Anonymous Inner Classes, Flavor Two

- **The only difference between flavor one and flavor two is that flavor one creates an anonymous subclass of the specified class type, whereas flavor two creates an anonymous implementor of the specified interface type.**

```
interface Cookable {  public void cook();}
class Food {
   Cookable c = new Cookable() {
                          public void cook() {
                              System.out.println("anonymous cookable implementer");
                      } };
 }
```

- **Argument-Defined Anonymous Inner Classes**

```
 class MyWonderfulClass {
  void go() {
   Bar b = new Bar();
   b.doStuff(new Foo() {
    public void foof() {
     System.out.println("foofy");
    } // end foof method
   }); // end inner class def, arg, and b.doStuff stmt.
  } // end go()
 } // end class
interface Foo {void foof();}
class Bar {void doStuff(Foo f) {} }
```

Static Inner Classes

Static Nested Classes

- **A static nested class is simply a class that's a static member of the enclosing class:**

```
class BigOuter {
  static class Nested { }
}
```

- **Instantiating and Using Static Nested Classes**
  - You use standard syntax to access a static nested class from its enclosing class.

```
class BigOuter {
  static class Nest {void go() ( System.out.println("hi"); } }
}
class Broom {
    static class B2 {void goB2() { System.out.println("hi 2"); } }
    public static void main(String[] args) {
            BigOuter.Nest n = new BigOuter.Nest();   // both class names
            n.go();
            B2 b2 = new B2();     // access the enclosed class
            b2.goB2();
     }
}
```

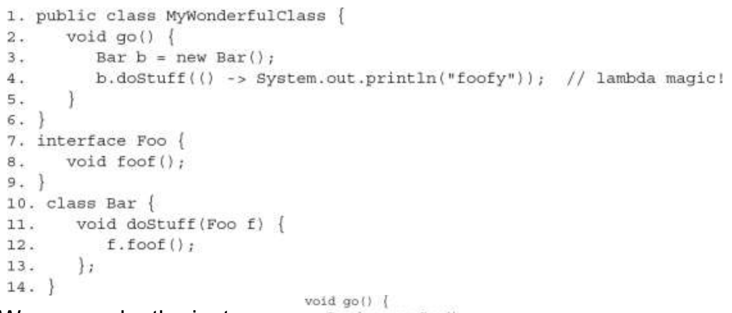
Which produces :

```
hi
hi 2
```

Lambda Expressions as Inner Classes

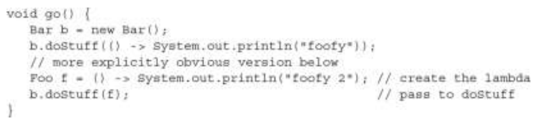
Create and use Lambda expressions

- **Inner classes are often used for short, quick implementations of a class**
  - When implementing an interface with an anonymous inner class, you'll often have opportunities to use lambda expressions to make your code more **concise**.
  - Now, let's see how to use this lambda expression:



```
public class MyWonderfulClass {
   void go() {
      Bar b = new Bar();
      b.doStuff(() -> System.out.println("foofy"));  // lambda magic!
   }
}
interface Foo {
   void foof();
}
class Bar {
    void doStuff(Foo f) {
       f.foof();
    };
}
```

  - We can make the instance object—the instance of a class that implements the Foo interface—a bit more explicit by creating it separately and then passing the instance to doStuff():



```
void go() {
  Bar b = new Bar();
  b.doStuff(() -> System.out.println("foofy"));
  // more explicitly obvious version below
  Foo f = () -> System.out.println("foofy 2"); // create the lambda
  b.doStuff(f);                                // pass to doStuff
}
```