JAVA PROGRAMMING BASICS

Module 2: Java Object-oriented Programming

Training program

1. Classes and Instances
2. The Methods
3. The Constructors
4. Static elements
5. Initialization sections
6. Package
7. Inheritance and Polymorphism
8. Abstract classes and interfaces
9. String processing
10. Exceptions and Assertions
11. Nested classes
12. Enums
13. Wrapper classes for primitive types
14. Generics
15. Collections
16. Method overload resolution
17. Multithreads
18. Core Java Classes
19. Object Oriented Design

Module contents

- Nested classes
  - The Nested classes classification
  - Creating instances of static and non-static nested classes
  - Inner class access to outer class fields
  - The Nested classes and JVM
  - Local inner classes
  - Anonymous inner classes
  - Using Anonymous class for array sorting. Interface Comparator
  - Inner classes and multiple inheritance

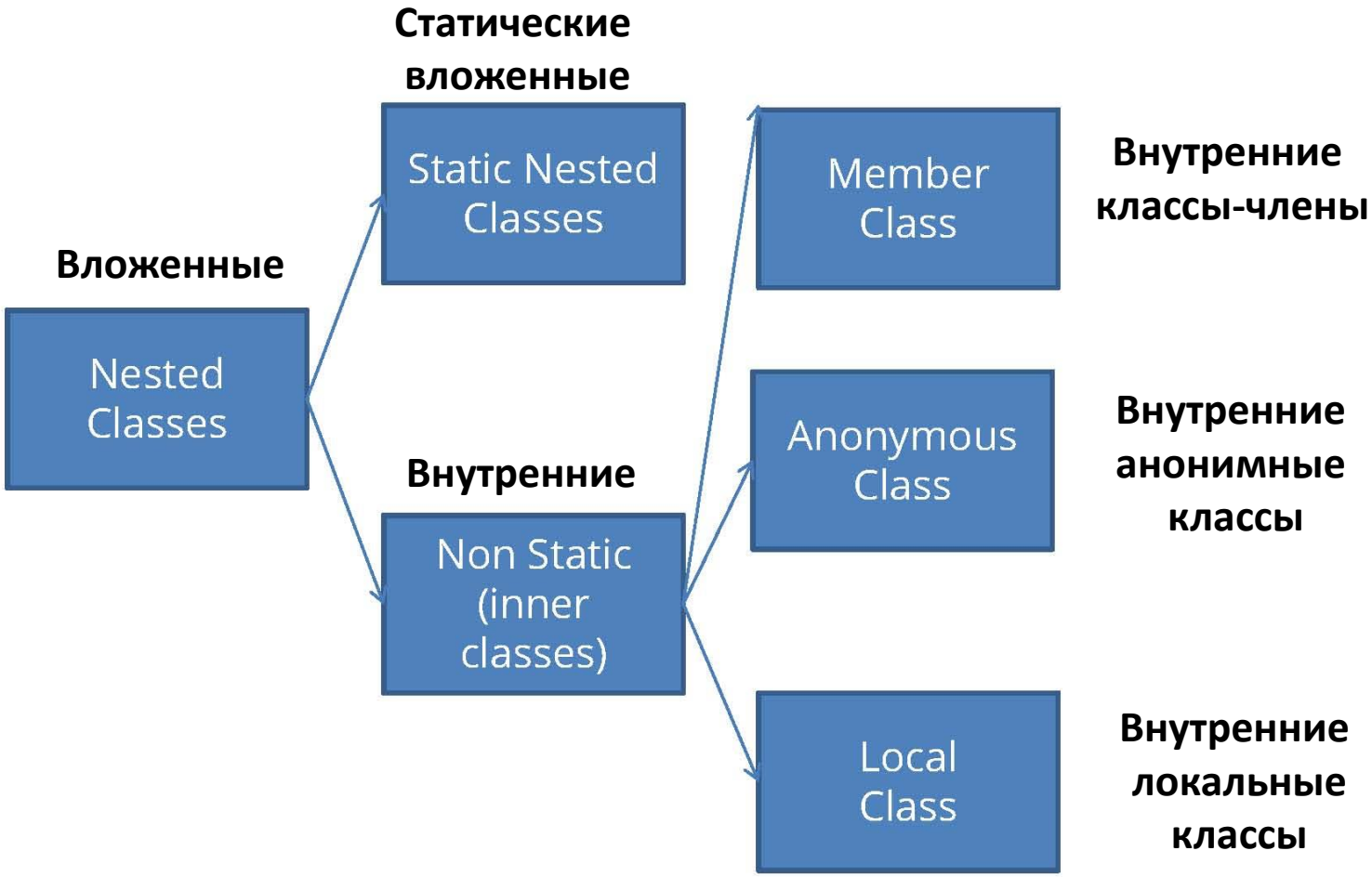
Module contents

- Nested classes
  - The Nested classes classification
  - Creating instances of static and non-static nested classes
  - Inner class access to outer class fields
  - The Nested classes and JVM
  - Local inner classes
  - Anonymous inner classes
  - Using Anonymous class for array sorting. Interface Comparator
  - Inner classes and multiple inheritance

The Nested classes classification 1/4

- The Java programming language allows you to define a class within another class. Such a class is called a *nested class*

The Nested classes classification 2/4



The Nested classes classification 3/4

- Why Use Nested Classes?
- It is a way of logically grouping classes that are only used in one place
- It increases encapsulation
- It can lead to more readable and maintainable code

The Nested classes classification 4/4

- The static nested and inner classes

```
1. public class OuterClass {
2.     //...
3.     static class StaticNestedClass {
4.         // ...
5.     }
6.     class InnerClass {
7.         // ...
8.     }
9. }
```

Module contents

- Nested classes
  - The Nested classes classification
  - Creating instances of static and non-static nested classes
  - Inner class access to outer class fields
  - The Nested classes and JVM
  - Local inner classes
  - Anonymous inner classes
  - Using Anonymous class for array sorting. Interface Comparator
  - Inner classes and multiple inheritance

Creating instances of static and non-static nested classes 1/3

- Static nested classes are accessed using the enclosing class name:

```
1. OuterClass.StaticNestedClass nestedObject =
2.         new OuterClass.StaticNestedClass();
```

Creating instances of static and non-static nested classes 2/3

- To instantiate an inner class, you must first instantiate the outer class.
- Then, create the inner object within the outer object with this syntax:

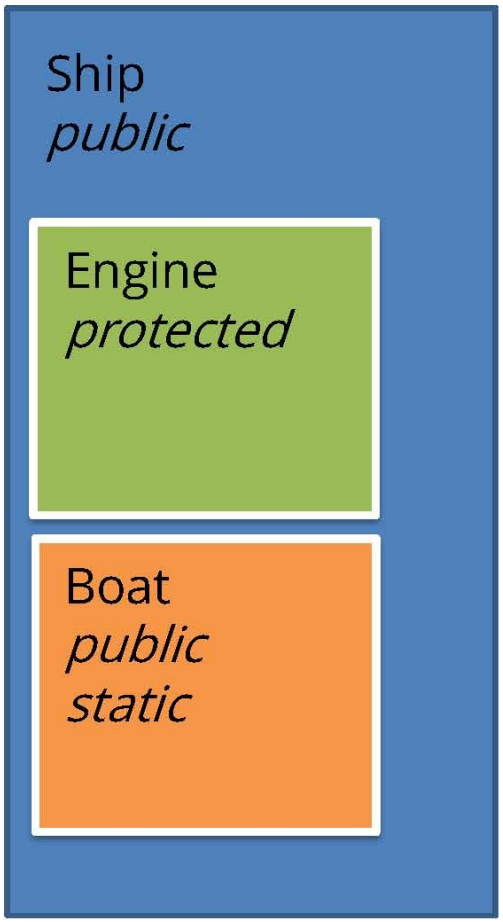
1. OuterClass outerObject = **new** OuterClass();
2. OuterClass.InnerClass innerObject =
3. outerObject.**new** InnerClass();

Creating instances of static and non-static nested classes 3/3

```
1.  class Ship{
2.      protected class Engine{
3.      }
4.      public static class Boat{
5.      }
6.  }
```

- //...

```
1. Ship.Boat boat1 = new Ship.Boat();
2. Ship ship1 = new Ship();
3. Ship.Engine engine = ship1.new Engine();
```



Module contents

- Nested classes
  - The Nested classes classification
  - Creating instances of static and non-static nested classes
  - Nested class access to outer class fields and methods
  - The Nested classes and JVM
  - Local inner classes
  - Anonymous inner classes
  - Using Anonymous class for array sorting. Interface Comparator
  - Inner classes and multiple inheritance

Nested class access to outer class fields 1/6

- The static nested class cannot refer directly to instance variables or methods defined in its enclosing class

```
1. class Ship{
2.     public int x=10;
3.     public static class Boat {
4.         public void test() {
5.             x = 20;
6.         }
7.     }
8. }
```

Nested class access to outer class fields 2/6

- Instance variables or methods defined in its enclosing class can use them only through an object reference

```
1. class Ship{
2.     private int x=10;
3.     public static class Boat {
4.         public void test() {
5.             Ship sh = new Ship();
6.             sh.x = 20;
7.         }
8.     }
9. }
```

Nested class access to outer class fields 3/6

- The static nested class can refer directly to static variables or methods defined in its enclosing class

```
1. class Ship{
2.     private static int x=10;
3.     public static class Boat {
4.         public void test() {
5.             x = 20;
6.         }
7.     }
8. }
```

Nested class access to outer class fields 4/6

- The Inner classes can refer directly to instance variables or methods defined in its enclosing class

```
1. class Ship{
2.     private int x=10;
3.     protected class Engine{
4.         public void test() {
5.             x = 20;
6.         }
7.     }
8. }
```

Nested class access to outer class fields 5/6

- The Inner classes can refer directly to instance variables or methods defined in its enclosing class via the qualified OuterClass.**this**.

```
1.  class Ship{
2.      private int x=10;
3.      protected class Engine{
4.          public void test() {
5.              int x = 100;
6.              Ship.this.x = 20;
7.          }
8.      }
9.  }
```

Nested class access to outer class fields 6/6

```
class Ship{
    void doJob() { System.out.println("Ship"); }
    protected class Engine{
        void doJob() { System.out.println("Engine"); }
        public void test() {
            Ship.this.doJob();//prints Ship
        }
    }
}
```

Module contents

- Nested classes
  - The Nested classes classification
  - Creating instances of static and non-static nested classes
  - Nested class access to outer class fields and methods
  - The Nested classes and JVM
  - Local inner classes
  - Anonymous inner classes
  - Using Anonymous class for array sorting. Interface Comparator
  - Inner classes and multiple inheritance

Nested Classes and the JVM 1/3

- There is no concept of nested classes in bytecode or in the JVM. The compiler generates a separate .class file for each class
- Since access modifiers are enforced in the JVM, the compiler sometimes has to work hard to split nested classes into independent classes

Nested Classes and the JVM 2/3

- The compiler generates a separate .class file for each class

```
public class Outer {
       class Inner {}
}
```

```
> javac Outer.java
> ls
Outer.java
Outer.class
Outer$Inner.class
```

Access methods in nested classes 1/4

```
public class SynthAccessMethodTest {
   public int extrenalClassField = 20;

   public static class Inner {
      public void accessToExternal() {
         SynthAccessMethodTest outer = new SynthAccessMethodTest();
         System.out.println(outer.extrenalClassField);
      }
   }
}
```

javap SynthAccessMethodTest.class

```
Compiled from "SynthAccessMethodTest.java"
public class innerclassespkg.SynthAccessMethodTest {
  public int extrenalClassField;
  public innerclassespkg.SynthAccessMethodTest();
}
```

Access methods in nested classes 2/4

```
public class SynthAccessMethodTest {
   private int extrenalClassField = 20;

   public static class Inner{
      public void accessToExternal() {
         SynthAccessMethodTest outer = new SynthAccessMethodTest();
         System.out.println(outer.extrenalClassField);
      }
   }
}
```

javap SynthAccessMethodTest.class

```
Compiled from "SynthAccessMethodTest.java"
public class innerclassespkg.SynthAccessMethodTest {
  public innerclassespkg.SynthAccessMethodTest();
  static int access$000(innerclassespkg.SynthAccessMethodTest);
}
```

Access methods in nested classes 3/4

```
public class SynthAccessMethodTest {
   private int extrenalClassField = 20;

   public static class Inner{
      public void accessToExternal() {
         SynthAccessMethodTest outer = new SynthAccessMethodTest();
         System.out.println(outer.extrenalClassField);
      }
   }
}
```

javap SynthAccessMethodTest$Inner.class

```
Compiled from "SynthAccessMethodTest.java"
public class innerclassespkg.SynthAccessMethodTest$Inner {
   final innerclassespkg.SynthAccessMethodTest this$0;
   public innerclassespkg.SynthAccessMethodTest$Inner(
                                 innerclassespkg.SynthAccessMethodTest);
   public void accessToExternal();  }
```

Access methods in nested classes 3/3

```
public class Main {

    public static void main(String[] args) {
        SynthAccessMethodTest accessMethodTest = new
                                                    SynthAccessMethodTest();
        SynthAccessMethodTest.Inner inner = accessMethodTest.new
                                                     Inner();
        inner.accessToExternal();
    }
}
```

Доступ к полю внешнего класса методом внутреннего возможен даже если поле приватное

Nested Classes and the JVM 3/3

actually compiles to something like

```
public class Outer {
        class Inner {}
}
```

```
public class Outer {
        static class Inner {
                Outer $this;
                Inner(Outer $this) {
                        this.$this = $this;
                }
        }
}
```

Module contents

- Nested classes
  - The Nested classes classification
  - Creating instances of static and non-static nested classes
  - Nested class access to outer class fields and methods
  - The Nested classes and JVM
  - Local inner classes
  - Anonymous inner classes
  - Using Anonymous class for array sorting. Interface Comparator
  - Inner classes and multiple inheritance

Local inner classes 1/4

- Local classes are classes that are defined in a *block*, which is a group of zero or more statements between balanced braces. You typically find local classes defined in the body of a method.

Local inner classes 2/4

- You can define a local class inside any block, for example, you can define a local class in a method body

```
1. class Ship {
2.  void doJob() {
3.      class LocalClass {
4.      }
5.    }
6. }
```

Local inner classes 3/4

- In addition, a local class has access to final local variables

```
1.  class Ship {
2.      private int x = 10;
3.      void doJob() {
4.          final int y = 10;
5.          class LocalClass {
6.              public void test() {
7.                  x = 20;
8.                  System.out.println(x + " " + y);
9.              }
10.         }
11.     }
12. }
```

Local inner classes 4/4

```
1.  class Ship {
2.      private int x = 10;
3.      void doJob() {
4.          int y = 10;
5.          class LocalClass {
6.              public void test() {
7.                  x = 20;
8.                  System.out.println(x + " " + y);
9.              }
10.         }
11.         y++;
12.     }
13. }
```

Local variable must be final or *effectively final (in Java 8)*

Module contents

- Nested classes
  - The Nested classes classification
  - Creating instances of static and non-static nested classes
  - Nested class access to outer class fields and methods
  - The Nested classes and JVM
  - Local inner classes
  - Anonymous inner classes
  - Using Anonymous class for array sorting. Interface Comparator
  - Inner classes and multiple inheritance

Anonymous inner classes 1/4

- Anonymous classes enable you to declare and instantiate a class at the same time.
- Anonymous classes like local classes except that they do not have a name.
- Use them if you need to use a local class only once.

1. new <parent class / interface>(<arguments>) {
2. <anonymous class body>
3. }

Anonymous inner classes 2/4

```
1.  interface MyTest{
2.      void test();
3.  }
4.  class Ship {
5.      void doJob() {
6.          MyTest tst = new MyTest(){
7.              public void test(){
8.                  System.out.print("TEST");
9.              }
10.         };
11.         tst.test();
12.     }
13. }
```

Anonymous inner classes 3/4

- Anonymous class has access to the members of its enclosing class

```
1.  class Ship {
2.      private int x=10;
3.      void doJob() {
4.          final int y=20;
5.          MyTest tst = new MyTest(){
6.              public void test(){
7.                  System.out.print(x+y);
8.              }
9.          };
10.         tst.test();
11.     }
12. }
```

Anonymous inner classes 4/4

- You cannot declare constructors in an anonymous class

```
1.  class Ship {
2.      void doJob() {
3.          MyTest tst = new MyTest(){
4.              private int z = 10;
5.              {
6.                 System.out.print("Init block");
7.              }
8.              public void test(){
9.                  System.out.print(z);
10.             }
11.         };
12.         tst.test();
13.     }
14. }
```

Module contents

- Nested classes
  - The Nested classes classification
  - Creating instances of static and non-static nested classes
  - Nested class access to outer class fields and methods
  - The Nested classes and JVM
  - Local inner classes
  - Anonymous inner classes
  - Using Anonymous class for array sorting. Interface Comparator
  - Inner classes and multiple inheritance

Using Anonymous class for array sorting. Interface Comparator 1/3

- **Comparator** interface is used to order the objects of user-defined class.
- It provides multiple sorting sequence i.e. you can sort the elements based on any data member. For instance it may be on rollno, name, age or anything else.

Using Anonymous class for array sorting. Interface Comparator 2/3

```
1. String[] arr = {"java", "se", "course","one"};
2. Arrays.sort(arr, new Comparator<String>() {
3.         @Override
4.         public int compare(String arg0, String arg1) {
5.            return arg0.length() - arg1.length();
6.         }
7.      });
8. System.out.println(Arrays.toString(arr));
```

Console output

[se, one, java, course]

Using Anonymous class for array sorting. Interface Comparator 3/3

```
1. String[] arr = {"java", "se", "course","one"};
2. Arrays.sort(arr, new Comparator<String>() {
3.         @Override
4.         public int compare(String arg0, String arg1) {
5.            return arg1.length() – arg0.length();
6.         }
7.      });
8. System.out.println(Arrays.toString(arr));
```

Console output

[course, java, one, se]

Null-safe comparator

```
String[] arr = {"java", null, null};
Arrays.sort(arr, new Comparator<String>() {
        @Override
        public int compare(String o1, String o2) {
            if (o1 == null ^ o2 == null) {
                return (o1 == null) ? 1 : -1;
            }
            if (o1 == null && o2 == null) {
                return 0;
            }
            return o1.compareTo(o2);
        }
    });
System.out.println(Arrays.toString(arr));
```

[java, null, null]

Module contents

- Nested classes
  - The Nested classes classification
  - Creating instances of static and non-static nested classes
  - Nested class access to outer class fields and methods
  - The Nested classes and JVM
  - Local inner classes
  - Anonymous inner classes
  - Using Anonymous class for array sorting. Interface Comparator
  - Inner classes and multiple inheritance

Inner classes and multiple inheritance 1/3

- It's possible to inherit from an inner class, but only as long as the inheriting class can be associated with a subtype of the original outer class

```
1. class BaseOuter {
2.     class BaseInner {}
3.     // Legal because BaseOuter is a subtype of BaseOuter:
4.     class BaseInner2 extends BaseInner {}
5. }
6. class DerivedOut extends BaseOuter {
7.   // Legal because DerivedOuter is a subtype of BaseOuter:
8.     class DerivedInner extends BaseInner {}
9. }
```

Inner classes and multiple inheritance 2/3

```
1.  class MyClass1 {}
2.  abstract class MyClass2 {}
3.  class MyClass3 extends MyClass1{
4.      MyClass2 makeMyClass2() {
5.          return new MyClass2() {};
6.      }
7.  }
8.  class MultiImplementation {
9.      static void takesMyClass1(MyClass1 d) {}
10.     static void takesMyClass2(MyClass2 e) {}
11.     public static void main(String[] args) {
12.         MyClass3 z = new MyClass3();
13.         takesMyClass1(z);
14.         takesMyClass2(z.makeMyClass2());
15.     }
16. }
```

Inner classes and multiple inheritance 3/3

Type	Inner (associated with outer instance)	Definition point	Visibility
Static nested class	No	As a member of another class.	Depends on access modifier.
Non-static member class (regular inner class)	Yes	As a member of another class.	Depends on access modifier.
Local class	Yes (if defined in non-static method)	Inside a method.	From the point it is defined to the end of the method.
Anonymous class	Yes (if defined in non-static method)	As an expression (since defining it also returns the instance).	None.

Inner class vs Nested class example

```
public class Ship {
    private String name;
    private int tonnage;
    private Engine engine;
    private Set<Boat> boats = new HashSet();
    /*Двигатель создаётся вместе с кораблём*/
    public Ship(String name, int tonnage, int power) {
        this.name = name;
        this.tonnage = tonnage;
        this.engine = this.new Engine(power);
    }
    /* Внутренний нестатический класс-член, каждый
     * экземпляр Engine связан с экземпляром Ship*/
    private class Engine {
        private int power;
        …
```

Inner class vs Nested class example

```
…
    public Engine(int power) {
        this.power = power;
    }
    /*Методы внутреннего класса могут иметь доступ к переменным
      внешнего класса, даже приватным*/
    public void launch() {
        System.out.println("Engine of " + name + " launched");
    }
    @Override
    public String toString() {
        return "Engine{" + "power=" + power + '}';
    }
}
…
```

Inner class vs Nested class example

```
…
/* Вложенный статический класс, Boat не привязан к конкретному экзе
   и может использоваться разными экземплярами Ship*/
    static class Boat {
        private int number;
        private Ship ship;
        private int numOfSeats;
        public Boat() {
        }
        public Boat(int number, int numOfSeats) {
            this.number = number;
            this.numOfSeats = numOfSeats;
        }
        public void addBoat(Ship ship) {
            ship.boats.add(this);
            this.ship = ship;
        } …
```

Inner class vs Nested class example

```
…
    public void transferBoat(Ship newShip) {
        this.ship.boats.remove(this);
        addBoat(newShip);
    }
    @Override
    public String toString() {
        return "Boat{" + "number=" + number + ", numOfSeats="
                        + numOfSeats + '}';
    }
}

@Override
public String toString() {
    return "Ship{" + "name=" + name + ", tonnage=" + tonnage + ",
            engine=" + engine + ", boats=" + boats + '}';
}
}
```

Inner class vs Nested class example

```
/**
 * Использование класса Ship с внутренним и вложенным классом в н
*/
public class Main {
    public static void main(String[] args) {
        Ship dryCargoShip = new Ship("T.Shevchenko", 30, 800);
        Ship passengerShip = new Ship("Bukovina", 15, 400);
       /*Доступ к приватному внутреннему классу отсутствует,
        Если бы он был публичным, то это было бы:*/
//        Ship.Engine engine = dryCargoShip.new Engine(500);
        /*Доступ к публичному вложенному классу*/
        Ship.Boat boat1 = new Ship.Boat(123, 28);
        boat1.addBoat(dryCargoShip);
        System.out.println(dryCargoShip);
        System.out.println(passengerShip);
        ...
```

Inner class vs Nested class example

```
…
        boat1.transferBoat(passengerShip);
        System.out.println(dryCargoShip);
        System.out.println(passengerShip);
    }
}
```

Output:

```
Ship{name=T.Shevchenko, tonnage=30, engine=Engine{power=800},
boats=[Boat{number=123, numOfSeats=28}]}
Ship{name=Bukovina, tonnage=15, engine=Engine{power=400},
boats=[]}
Ship{name=T.Shevchenko, tonnage=30, engine=Engine{power=800},
boats=[]}
Ship{name=Bukovina, tonnage=15, engine=Engine{power=400},
boats=[Boat{number=123, numOfSeats=28}]}
```