CS6308- Java Programming

V P Jayachitra
Assistant Professor
Department of Computer Technology
MIT Campus
Anna University

MODULE II JAVA OBJECTS -1	L	T	P	EL
	3	0	4	3
Classes and Objects, Constructor, Destructor, Static instances, this, constants, Thinking in Objects, String class, Text I/O				
SUGGESTED ACTIVITIES : • Flipped classroom • Practical - Implementation of Java programs – using String class, Creating Classes and objects • EL – Thinking in Objects				
SUGGESTED EVALUATION METHODS: • Assignment problems • Quizzes				

Static

- **Why static member?**
  - A class member must be accessed only in conjunction with an object of its class.
  - Is it possible to create a member that can be used by itself, without reference to a specific instance?
  - Yes, static class member will be used independently of any object of that class.

What is static member?

- A member that is declared static can be accessed before any objects of its class are created, and without reference to any object.
- To create such a member, precede its declaration with the keyword static.
- Both methods and variables can be declared to be static.
- Example : main()method
  - main( ) is declared as static
  - main() should be called before any objects exist.

Static variables

- Instance variables declared as static are, essentially, **global variables**.
- When objects of its class are declared, all instances of the class share the same static variable.
- Static variables are used independently of any object using class name followed by the dot operator.
  - classname.staticVariable

Static methods

- Methods declared as static are, essentially, **global methods**.
- Methods declared as static have several restrictions:
  - They can **only directly call other static methods** of their class.
  - They can **only directly access static variables** of their class.
  - **They cannot refer to this or super** in any way. (super relates to inheritance).
  - They are used **independently of any object** using class name followed by the dot operator.
    - classname.method( )

Static block

- Static block is used to initialize your static variables.

- Static block gets executed exactly once, when the class is first loaded.

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

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

```
a = 42
b = 99
```

```
public class StaticBlockExample {
    // Static block
    static {
        System.out.println("Static block is executed.");
    }

    // Constructor
    public StaticBlockExample() {
        System.out.println("Constructor is executed.");
    }

    // Static variable
    static int staticVariable = initializeStaticVariable();

    // Static method
    static int initializeStaticVariable() {
        System.out.println("Static variable initializer is executed.");
        return 10;
    }
public static void main(String[] args) {
        System.out.println("Main method is executed.");
        StaticBlockExample obj = new StaticBlockExample();
    }
}
```

```
Static block is executed.
Static variable initializer is executed.
Main method is executed.
Constructor is executed.
```

- The **static block** is used to initialize static variables and runs only once when the class is first loaded.
- Static variables are initialized after the static block.
- The main method is executed after the static initialization phase, and the **constructor** is executed

```java
public class Final {
    public void final_varaible_method(){
        final int a=1;
        System.out.println(a++);
    }

    public static void main(String []argh){
        Final f=new Final();
        f.final_varaible_method();
    }
}
```

java: cannot assign a value to final variable a

```java
// Demonstrate static variables, methods, and blocks.
class UseStatic {
  //static variables
    static int a = 3;
    static int b;
 //static method
    static void meth(int x) {
        System.out.println("x = " + x);
        System.out.println("a = " + a);
        System.out.println("b = " + b);
    }
//static block
    static {
        System.out.println("Static block initialized.");
        b = a * 4;
    }
    public static void main(String args[]) {
        meth(42);
    }
}
```

```
a = 3
b = 12
```

Introducing Nested and Inner Classes

- **Define a class within another class and such classes are known as nested classes.**
- The scope of a nested class is bounded by the scope of its enclosing class.
  - class B defined within class A does not exist independently of A.
- A nested class has access to the members, including private members, of the class in which it is nested.
- The **enclosing class does not have access to the members of the nested class.**
- A nested class that is declared directly within its enclosing class scope is a member of its enclosing class.
- A nested class also be declared that is local to a block.

```
public class EnclosingClass {
  . . .
  public class NestedClass {
  . . .  }
}
```

Introducing Nested and Inner Classes

```
class OuterClass {
    ...
    class InnerClass {
        ...
    }
    static class StaticNestedClass {
        ...
    }
}
```

- A **nested class** is a member of its enclosing class.
- Non-static nested classes (**inner classes**) have access to other members of the enclosing class, even if they are declared private.
- **Static nested classes** do not have access to other members of the enclosing class.
- As a member of the OuterClass, a nested class can be declared private, public, protected

Why nested class?

- A nested class is a class that’s tightly coupled with the class in which it’s defined.

- A **nested class has access to the private data** within its enclosing class

Why Use Nested Classes?

- Compelling reasons for using nested classes include the following:
  - It is a way of logically grouping classes that are only used in one place:
    - 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
  - It increases encapsulation:
    - Consider two top-level classes, A and B, where B needs access to members of A that would otherwise be declared private.
    - By hiding class B within class A, A's members can be declared private and B can access them. In addition, B itself can be hidden from the outside world.
  - It can lead to more readable and maintainable code:
    - Nesting small classes within top-level classes places the code closer to where it is used.

Nested classes-static nested class

- There are two types of nested classes:
  - **Static**
  - **non-static.**

- A static nested class is one that has the static modifier applied.
- Static class must access the non-static members of its enclosing class through an object.
- Static class cannot refer to non-static members of its enclosing class directly.
  - Because of this restriction, static nested classes are seldom used
- **Note:** Top level class can not be declared as static. Only nested classes can be declared as static.

Nested classes-Inner class

- A non-static nested class is an inner class.
- Inner class has access to all of the variables and methods of its outer class.
- Inner class refer variables and methods directly
- in the same way that other non-static members of the outer class do.

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

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

OuterClass outerObject = new OuterClass();
OuterClass.InnerClass innerObject = outerObject.new InnerClass();

```
public class Manager extends Employee {
  private DirectReports directReports;
  public Manager() {
    this.directReports = new DirectReports();
  }
  . . .
  private class DirectReports {
  . . .
  }
}
```

Creating DirectReports instances: First attempt

Creating DirectReports instances: Second attempt

```
public class Manager extends Employee {
  public Manager() {
  }
  . . .
  public class DirectReports {
  . . .
  }
  }
public static void main(String[] args) {
  Manager manager = new Manager();
  Manager.DirectReports dr = manager.new DirectReports();
}
```

```
public class Manager extends Employee {
  public Manager() {
  }
  . . .
  public class DirectReports {
  . . .
  }
}
public static void main(String[] args) {
  Manager.DirectReports dr = new Manager.DirectReports();// This won't
work!
```

```
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);
        }
    }
```

```
 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
```

Shadowing

```
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
```

1.What is a nested class?

1. A class that provides some utility to other classes in the application
2. A class that is defined within another class
3. A class where one or more interface references are passed into its constructor
4. None of the above

1.What is a nested class?

1. A class that provides some utility to other classes in the application
2. **A class that is defined within another class**
3. A class where one or more interface references are passed into its constructor
4. None of the above

1.Why might you use a nested class?

1. When you need to define a class that is tightly coupled (functionally) with another class
2. When you want to write a class that makes use of a large number of interfaces from the JDK
3. When one class needs to access to another class private data, but you have exceeded the maximum number of allowable classes for your application
4. For a class has more than 20 methods defined on it and should be refactored

1.Why might you use a nested class?

1. **When you need to define a class that is tightly coupled (functionally) with another class**
2. When you want to write a class that makes use of a large number of interfaces from the JDK
3. When one class needs to access to another class private data, but you have exceeded the maximum number of allowable classes for your application
4. For a class has more than 20 methods defined on it and should be refactored

```
public class Outer {

 private static final Logger log = Logger.getLogger(Outer.class.getName());

 public void setInner(Inner inner) {
 this.inner = inner;
 }
 private Inner inner;

 public Inner getInner() {
 return inner;
 }

 private class Inner {
 }
  public static void main(String[] args) {
 Outer outer = new Outer();
 Inner inner = new Outer.Inner();
 outer.setInner(inner);
 log.info("Outer/Inner: " + outer.hashCode() + "/" + outer.getInner().hashCode());
 }

}
```

1.The class name `Outer` is not a legal Java class name.
2.The class name `Inner` is confusing.
3.The main method defined in class `Outer` has the wrong method signature.
4.The `log.info()` call in main() has too many parameters.
5.The class body for class `Inner` is empty.
6.None of the above. The line Inner inner = new Outer.Inner(); is in error. This is not the correct syntax for instantiating a nested class using an enclosing class reference. Rather, it should be: Inner inner = outer.new Inner();

```
public class Outer {

 private static final Logger log = Logger.getLogger(Outer.class.getName());

 public void setInner(Inner inner) {
 this.inner = inner;
 }
 private Inner inner;

 public Inner getInner() {
 return inner;
 }

 private class Inner {
 }
  public static void main(String[] args) {
 Outer outer = new Outer();
 Inner inner = new Outer.Inner();
 outer.setInner(inner);
 log.info("Outer/Inner: " + outer.hashCode() + "/" + outer.getInner().hashCode());
 }

}
```

1. The class name `Outer` is not a legal Java class name.
2. The class name `Inner` is confusing.
3. The main method defined in class `Outer` has the wrong method signature.
4. The `log.info()` call in main() has too many parameters.
5. The class body for class `Inner` is empty.
6. None of the above. The line Inner inner = new Outer.Inner(); is in error. This is not the correct syntax for instantiating a nested class using an enclosing class reference. Rather, it should be: Inner inner = outer.new Inner();

1.Is it possible to write an inner class that can be instantiated by any class in your application (regardless of what package in which it resides) without an enclosing instance of the outer class? Explain your answer.

1.Is it possible to write an inner class that can be instantiated by any class in your application (regardless of what package in which it resides) without an enclosing instance of the outer class? Explain your answer. **Yes, it is. The inner class should be declared with the static modifier, along with public visibility. Then any class in your application can instantiate the inner class without an enclosing outer instance of the class.**

1.What’s the difference between a nested class an an inner class?

1. An inner class is one that is defined using private access only, whereas a nested class is declared static.
2. The two terms are synonymous and can be used interchangeably.
3. A nested class must reside inside the enclosing class and be declared before any of the enclosing class’s variables.
4. A nested class is defined in main(), whereas an inner class can be defined anywhere.
5. There is no such thing as a nested class.
6. None of the above.

1.What’s the difference between a nested class an an inner class?

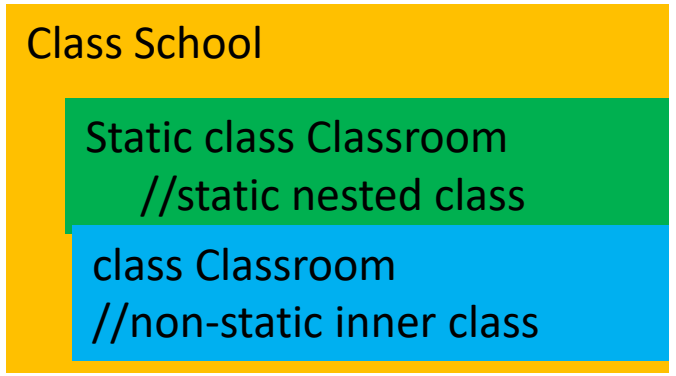
1. An inner class is one that is defined using private access only, whereas a nested class is declared static.
2. **The two terms are synonymous and can be used interchangeably.**
3. A nested class must reside inside the enclosing class and be declared before any of the enclosing class’s variables.
4. A nested class is defined in main(), whereas an inner class can be defined anywhere.
5. There is no such thing as a nested class.
6. None of the above.

1. Which of these statements is true regarding inner classes?
    1. An inner class can access any private data variables of its enclosing class unless it is declared `static`.
    2. An inner class must be declared `public` to be instantiated by any other class than its enclosing class.
    3. A static inner class is not allowed except under special circumstances.
    4. An inner class is completely invisible to its enclosing class.
    5. None of the above.

1. Which of these statements is true regarding inner classes?
    1. **An inner class can access any private data variables of its enclosing class unless it is declared `static`.**
    2. An inner class must be declared `public` to be instantiated by any other class than its enclosing class.
    3. A static inner class is not allowed except under special circumstances.
    4. An inner class is completely invisible to its enclosing class.
    5. None of the above.

```
public class School {
    private String schoolName;
    private Classroom[] classrooms;
    private Student[] students;
    private int classroomCount = 0;
    private int studentCount = 0;
    public School(String schoolName, int maxClassrooms, int maxStudents) {
        this.schoolName = schoolName;
        this.classrooms = new Classroom[maxClassrooms];
        this.students = new Student[maxStudents];
    }

// Static nested class                    // print student details
    public static class Classroom {
        private String roomNumber;
        private String subject;
         public Classroom(String roomNumber, String subject) {
            this.roomNumber = roomNumber;
            this.subject = subject;                                    }
        public String getRoomNumber() {
            return roomNumber;        }
        public String getSubject() {
            return subject;                }
    }
```



```
// Non-static inner class
    public class Student {
        private String name;
        private String grade;

     public Student(String name, String grade) {
            this.name = name;
            this.grade = grade;    }

    public String getName() {
            return name;       }

     public String getGrade() {
            return grade;        }

     public void printStudentDetails() {// print student details
            System.out.println("Student Name: " + name + ", Grade: " + grade + ", School Name: " + schoolName);          }       }

     public void addClassroom(Classroom classroom) {// add a classroom
       if (classroomCount < classrooms.length) {
            classrooms[classroomCount++] = classroom;
       } else {
            System.out.println("No more space to add classrooms.");
       }     }
```

```
// add a student
   public void addStudent(Student student) {
      if (studentCount < students.length) {
         students[studentCount++] = student;
      } else {
         System.out.println("No more space to add students.");
      }
   }
   // print all classrooms
   public void printAllClassrooms() {
      System.out.println("Classrooms:");
      for (int i = 0; i < classroomCount; i++) {
         Classroom classroom = classrooms[i];
         System.out.println("Room Number: " +
classroom.getRoomNumber() + ", Subject: " +
classroom.getSubject());
      }
   }
// print all students
   public void printAllStudents() {
      System.out.println("Students:");
      for (int i = 0; i < studentCount; i++) {
         students[i].printStudentDetails();
      }    }
```

```
   public static void main(String[] args) {
       School school = new School(“MIT", 5, 10);
      // Add classrooms
  school.addClassroom(new School.Classroom("101","Maths"));
school.addClassroom(new School.Classroom("102", "Science"));

      // Add students
      School.Student st1 = school.new Student("ARUN", "A");
      School.Student st2 = school.new Student(“BANU", "B");
      school.addStudent(st1);
      school.addStudent(st2);

      // Print all classrooms
      school.printAllClassrooms();

      // Print all students
      school.printAllStudents();
   } }
```

What is a static nested class and how is it different from an inner class?

```
class OuterClass {

static class StaticNestedClass {
  void display() {
        System.out.println("This is a static nested class");
    }
 }

 class InnerClass {
   void display() {
      System.out.println("This is an inner class");
    }
 }

 }
```

```java
class Calculator {
    static int baseValue = 10;

    static class Adder {
        int add(int x) {
            return x + baseValue;  // Directly accessing static member of outer class
        }
    }
}
```

```java
class Container {
    static class StaticNested {
        static int staticVar = 5;
        static void staticMethod() {
            System.out.println("Static method in static nested class");
        }
    }
}
```

```java
class Outer {
    class Inner {
        void innerMethod() {
            System.out.println("Inner class method");
        }
    }
}
```

```java
Outer outer = new Outer();
Outer.Inner inner = outer.new Inner();
inner.innerMethod();
```

```java
class ShadowExample {
    private int x = 10;

    class InnerShadow {
        private int x = 20;

        void printBoth() {
            System.out.println("Inner x: " + x);
            System.out.println("Outer x: " + ShadowExample.this.x);
        }
    }
}
```