

Monsoon 2019

Nested Classes

O b j e c t O r i e n t e d

P r o g r a m m i n g

by

Dr. Rajendra Prasath

Indian Institute of Information Technology

Sri City – 517 646, Andhra Pradesh, India



Recap: Objects in JAVA ?

- ✧ An entity that has **state** and **behaviour** is known as an object
 - ✧ **Examples:** Chair, bike, marker, pen, table, car etc
 - ✧ It can be physical or logical
- ✧ An object has three characteristics:
 - ✧ **State:** represents data (value) of an object
 - ✧ **Behaviour:** represents the behaviour (functionality) of an object such as deposit, withdraw and so on
 - ✧ **Identity (Internally used):**
 - ✧ Signature (unique) of the object
 - ✧ Object identity is typically implemented via a unique ID
 - ✧ The value of the ID is not visible to the external user
 - ✧ But, Internally by JVM to identify each object uniquely



Recap: this keyword in java

✧ Why use this keyword in JAVA?

- ✧ Main purpose: To differentiate the formal parameter and data members of class
- ✧ Why?
 - ✧ whenever the formal parameter and data members of the class are similar then jvm gets ambiguity (no clarity between formal parameter and member of the class)
 - ✧ So to differentiate between formal parameter and data member of the class, the data member of the class must be preceded by "this".
- ✧ **"this"** keyword can be use in two ways.
 - ✧ **this . (this dot)**
 - ✧ **this() (this off)**



this – invoke a Method: Recap

✧ Example:

```
Class P {  
    Person obj;  
    P (Person obj) {  
        this.obj = obj;  
    }  
    void display() {  
        System.out.println("Name: "  
            + obj.name);  
    }  
}
```

```
Class Person {  
    String name = "Alfred De Souza";  
    Person() {  
        P p = new P(this);  
        p.display();  
    }  
    public static void main(String  
args[]) {  
        Person p = new Person();  
    }  
}
```

Output: **Alfred De Souza**



Recap: static – Create a counter

```
public class Counter {  
    static int count = 0;  
  
    public Counter() {  
        count++;  
        System.out.println("Counter: " + count);  
    }  
  
    public static void main(String[] args) {  
        Counter c;  
        c = new Counter();  
        c = new Counter();  
        c = new Counter();  
        c = new Counter();  
    }  
}
```



Nested Classes

- ✧ **What is a nested class?**
- ✧ A class within another class is known as **Nested class**
- ✧ The scope of the nested is bounded by the scope of its enclosing class.
- ✧ The class that holds the inner class is called the outer class.
- ✧ Nested Classes are Divided into two Types:
 - ✧ **Non-Static Nested Classes** – These are the non-static members of a class.
 - ✧ **Static Nested Classes** – These are the static members of a class.



Nested Classes - Advantages

- ✧ 3 advantages of inner classes in java
 - ✧ Nested classes **represent a special type of relationship** that is it can access all the members (data members and methods) of outer class including private.
 - ✧ Nested classes are used to develop **more readable and maintainable code** because it logically group classes and interfaces in one place only.
 - ✧ **Code Optimization:** It requires less code to write.



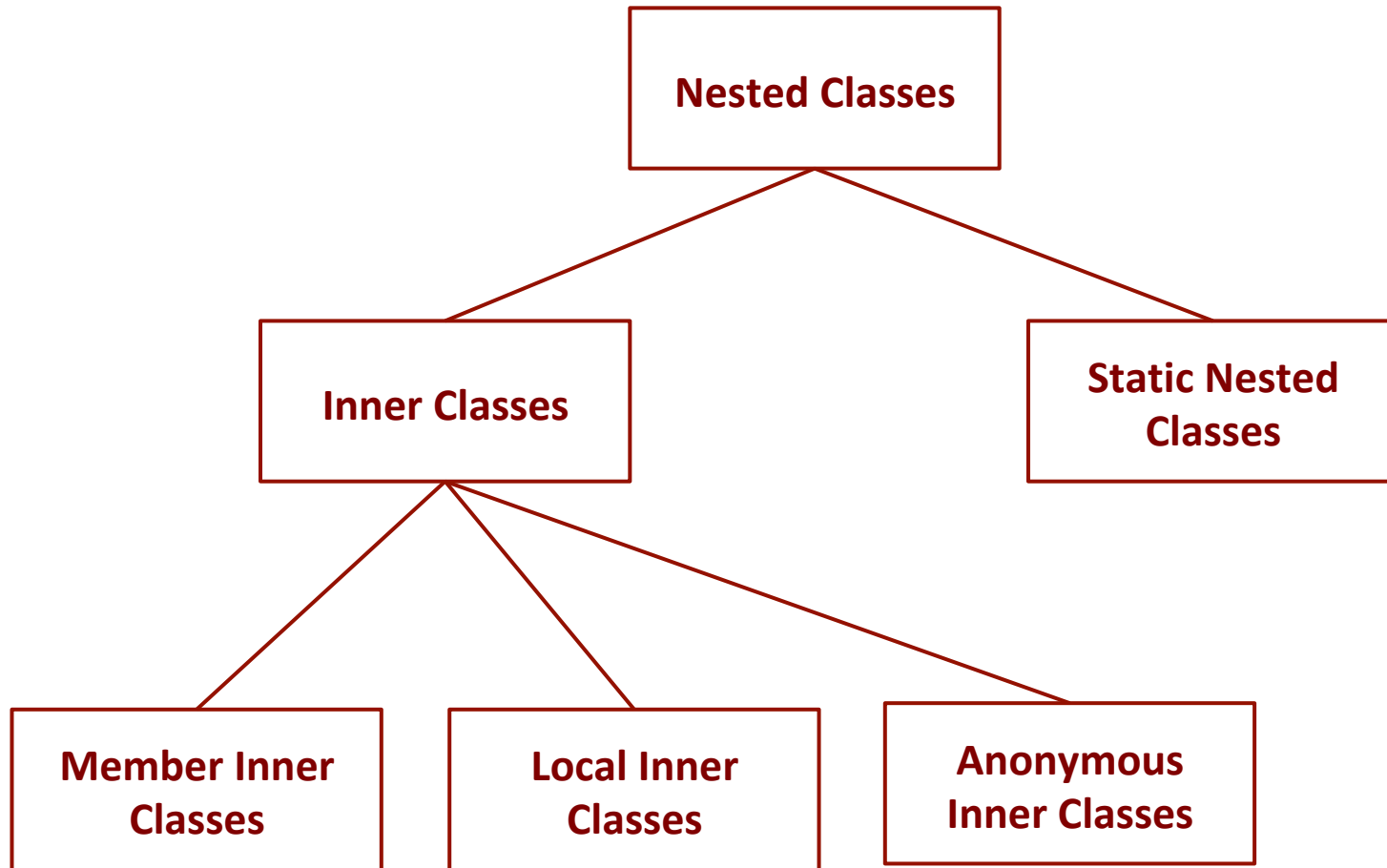
Nested Classes - Advantages

- ✧ **Member Inner Class:** A class created within class and outside method
- ✧ **Anonymous Inner Class:** A class created for implementing interface or extending class
 - ✧ Its name is decided by the java compiler
- ✧ **Local Inner Class:** A class created within method
- ✧ **Static Nested Class:** A static class created within class



Nested Classes - Hierarchy

✧ Various Types of Nested Classes:



Static Nested Classes in JAVA

- ✧ **What is a static nested class?**
- ✧ A static nested class is the one that has static modifier applied.
- ✧ Since it is static, it can not be applied to non-static members of its enclosing class directly.

- ✧ **Syntax:**

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

- ✧ **this** – refers to the currently executing inner class
- ✧ To get this for outer class, use “**Outer.this**”



Java Static Nested Class

- ✧ A static class created inside a class is called static nested class in java
- ✧ It cannot access non- static data members and methods
- ✧ It can be accessed by outer class name
- ✧ It can access static data members of outer class including private
- ✧ Static nested class cannot access non-static (instance) data member or method
- ✧ **Syntax:**
Outer.Inner objectName = **new** Outer.Inner();



Static Nested Class - Example

```
class Outer {  
    static int id=11;  
    static String name="";  
    static class Inner {  
        void display() {  
            System.out.println("ID is: "+id);  
            System.out.println("Name is: "+name);  
        }  
    }  
    public static void main(String args[] {  
        Outer.Inner obj=new Outer.Inner();  
        obj.display();  
    }  
}
```



Static Nested Class - Variation

```
class Outer {  
    static int id=11;  
    static String name="";  
    static class Inner {  
        static void display() {  
            System.out.println("ID is: "+id);  
            System.out.println("Name is: "+name);  
        }  
    }  
    public static void main(String args[] {  
        Outer.Inner.dispaly();  
    }  
}
```



One More Static Nested Class

```
class Outer {  
    private static int age=24;  
    static class Inner {  
        int getAge() {  
            return age;  
        }  
    }  
    public static void main(String args[] {  
        Outer.Inner inn = new Outer.Inner();  
        System.out.println("Age = " + inn.getAge() );  
    }  
}
```



One More Static Nested Class

```
class Outer {  
    private static int age=28;  
    static class Inner {  
        int age = 21;  
        int getAge() {  
            return age;  
        }  
    }  
    public static void main(String args[] {  
        Outer.Inner n1 = new Outer.Inner();  
        System.out.println("Age = " + n1.getAge() );  
    }  
}
```

**Member with
the same name !!**

Inner member will shadow the
outer class member



Static Nested Class

- ✧ Can **access only static members** of outer class inside a static nested class and **Can not access the non-static members** of outer class inside a static nested class

```
class Outer {  
    static int i;        //static field of Outer  
    int j;              //Non-static field of Outer  
    void methodOne() { // Non-static method of Outer  
    }  
    static void methodTwo() { // static method of Outer  
    }  
    static class Nested {  
        void methodOfInner() {  
            System.out.println(i); // static field can be accessed  
            System.out.println(j); // Compile time error - Non-static field  
  
            methodTwo(); // can access static method  
            methodOne(); // Compile time error: non-static field  
        }  
    }  
}
```



Static Nested Class - Access

- ✧ Non-static Members Of Outer Class Cannot Access Inside a Static Nested Class

```
class Outer {  
    int age = 21; //Non-static field of Outer Class  
    static class Inner{  
        int yrs = age;  
        int getAge() {  
            return yrs;  
        }  
    }  
}
```

Compiler Error !!

This error is due to the assignment of non-static data field of outer class to non-static data field.

```
public static void main(String[] args) {  
    Outer.Inner m1 = new Outer.Inner();  
    System.out.println("Sr. Number :" + m1.getAge());  
}
```



Static Nested Class - Access

- ✧ Non-static Members Of Outer Class Cannot Access Inside a Static Nested Class

```
class Outer {  
    static int age = 21; //Non-static field of Outer Class  
    static class Inner{  
        int yrs = age;  
        int getAge() {  
            return yrs;  
        }  
    }  
}
```

No Compiler Error

Because assignment of static data field of outer class to non-static data field.

```
public static void main(String[] args) {  
    Outer.Inner m1 = new Outer.Inner();  
    System.out.println("Sr. Number :" + m1.getAge());  
}
```



More than one static nested class

```
class Outer {  
    int i = 10;           // Non-static Field of Outer  
    static void methodOne() {  
        System.out.println("Static method of OuterClass");  
    }  
    static class NestedClassOne {  
        int i = 20;       // Non-static Field of NestedClassOne  
        static void methodOne() {  
            System.out.println("Static method of NestedClassOne");  
        }  
    }  
    static class NestedClassTwo {  
        int i = 30;       // Non-static Field of NestedClassTwo  
        static void methodOne() {  
            System.out.println("static method of NestedClassTwo");  
        }  
    }  
}
```



More than one ... (contd.)

Main() Method

```
public static void main(String[] args) {  
    // static member can be accessed directly through class name  
    Outer.methodOne();  
  
    Outer outer = new Outer();  
    //Instance member must be accessed through object ref  
    System.out.println(outer.i);  
  
    Outer.NestedClassOne.methodOne();  
    Outer.NestedClassOne nestedOne = new Outer.NestedClassOne();  
    System.out.println(nestedOne.i);  
  
    Outer.NestedClassTwo.methodOne();  
    Outer.NestedClassTwo nestedTwo = new Outer.NestedClassTwo();  
    System.out.println(nestedTwo.i);  
}
```



Overloading – Methods / Constructors

```
class Outer {  
    static class Nested {  
        Nested () {  
            //First constructor  
        }  
        Nested (int i) {  
            //Second Constructor with one Parameter  
        }  
        Nested (int i, int j) {  
            //Third Constructor with two Parameters  
        }  
        void methodOne () {  
            //Overloaded method (without Parameter)  
        }  
        void methodOne (int i) {  
            //Overloaded method (with one Parameter)  
        }  
        void methodOne (int i, int j) {  
            //Overloaded method (Two Parameters)  
        }  
    }  
}
```



Inner Classes in JAVA

- ✧ A non-static class that is created inside a class but outside a method is called member inner class
- ✧ A Member Inner Class is a class whose definition is directly enclosed by another class or interface declaration
- ✧ Instance of a **Member Inner Class** can only be created if we have a reference to an instance of outer class
- ✧ Purpose:
 - ✧ Provide More Security - make inner class Properties specific to ONLY outer classes
 - ✧ Make more than one property of classes private



Member Inner Class (Within class)

```
class Outer {  
    private static int age=28;  
    class Inner {  
        int getAge() {  
            return age;  
        }  
    }  
    void displayAge() {  
        Inner in = new Inner();  
        System.out.println("Age = " + in.getAge() );  
    }  
    public static void main(String args[] {  
        Outer out = new Outer();  
        out.displayAge();  
    }  
}
```



Member Inner Class (outside class)

```
class Outer {  
    private static int age = 28;  
    class Inner {  
        int getAge() {  
            return age;  
        }  
    }  
    public static void main(String args[] {  
        Outer out = new Outer();  
        Outer.Inner oin = out.new Inner();  
        System.out.println("Age = " + oin.getAge() );  
    }  
}
```



Member Inner Class – Key Points

- ✧ **Member inner class must contain only non-static members**
- ✧ **Static Members are not allowed inside inner classes**

```
class Outer {
```

```
    //Member Inner Class : Class As a Non-Static Member
```

```
    class Inner {
```

```
        int i;                // can contain non-static field
```

```
        static int j = 10;    // It gives compile time error
```

```
        // Should not contain static field
```

```
        void methodOne() {
```

```
            // can have non-static method
```

```
        }
```

```
        static void methodTwo(){
```

```
            //Compile time error - should not contain static method
```

```
        }
```

```
    }
```

```
}
```



Key Points of Member Inner Class

- ✧ Both static and non-static members of outer class can access from inside the member inner class.

```
class Outer {  
    int i;  
    static int j;  
  
    void m1 () {  
        System.out.println("Non-static Method!");  
    }  
    static void m2() {  
        System.out.println("static Method!");  
    }  
    class Inner {  
        void methodOfInner() {  
            m1();  
            m2();  
        }  
    }  
}
```

From Next Class



Member Inner Class–Abstract / Final

✧ Member inner class

```
class Outer {  
    abstract class InnerOne {  
        // abstract inner class  
    }  
    final class InnerTwo {  
        // final inner class  
    }  
}
```



Local Inner Class

- ✧ A class created inside a method is called local inner class in java
- ✧ If we want to invoke the methods of local inner class, we must instantiate this class inside the method
- ✧ Local Inner Class is not a member class of any other class
- ✧ Its declaration is not directly within the declaration of the outer class
- ✧ Local Inner Class can be declared inside particular block of code and its scope is within the block



Local Inner Class – with local variable

```
class Outer{  
    private int data=30; //instance variable  
    void display(){  
        int value=50; //local variable  
        class Local {  
            void msg(){  
                System.out.println("Value is: " + value);  
            }  
        }  
        Local l = new Local();  
        l.msg();  
    }  
    public static void main(String args[]){  
        Outer obj = new Outer();  
        obj.display();  
    }  
}
```



Anonymous Inner Classes

- ✧ A class which does not have name is called as anonymous inner class. Anonymous class can be created using two methods :
 - ✧ By using the interface
 - ✧ By using the abstract class
- ✧ Anonymous Inner Classes are used for writing an interface implementation
- ✧ **Properties:**
 - ✧ Has no name
 - ✧ Can be instantiated only once
 - ✧ Is usually declared inside a method or a code block, a curly braces ending with semicolon.
 - ✧ Is accessible only at the point where it is defined.
 - ✧ Does not have a constructor simply because it does not have a name
 - ✧ Cannot be static



Anonymous IC - Example

```
interface Writable {  
    void write(String msg);  
}  
  
public class CodeTester {  
    public static void main(String[] args) {  
        Writable w1 = new Writable() {  
            public void write(String msg){  
                System.out.println(msg);  
            }  
        };  
        // Note the semicolon  
        w1.write("Writing Diary");  
    }  
}
```



Anonymous IC as argument

```
Interface Message{
    String welcome();
}

public class CodeTester {
    public void showMessage(Message m) {
        System.out.println(m.welcome());
    }
    Public static void main(String args[]) {
        CodeTester ct = new CodeTester();
        ct.showMessage(new Message() {
            public String welcome() {
                return "Welcome Folks!!";
            }
        });
    }
}
```



Returning Object from Method

```
Class Employee {  
    double salary, incentive;  
    Employee (double salary) {  
        this.salary = salary;  
    }  
    Employee updateSalary(double incentive) {  
        Employee emp = new Employee(this.salary + incentive);  
        return emp;  
    }  
    double getSalary() {  
        return this.salary;  
    }  
}
```



Main method

Class **Checker** {

public static void main (String args[]) {

Employee ram = new Employee(24,856.38);

Employee anand;

anand = ram.updateSalary(6259.67);

System.out.print("Ram's Salary = " + ram.getSalary());

System.out.print("Anand's Salary = " +
anand.getSalary());

}

}



Create Object using Inner Class

```
class Circle {  
    double x, y, r;  
}  
class CreateCircle {  
    Circle createCicle() {  
        return new Circle();  
    }  
}  
class CodeTester {  
    public static void main(String[] args) {  
        CreateCircle cc = new CreateCircle();  
        Circle c1 = cc.createCicle();  
        c1.x = 3.0; c1.y = 5.0; c1.r = 5.0;  
        System.out.println("x = " + c1.x + ", y = " + c1.y  
                            + ", r = " + c1.r);  
    }  
}
```



Exercise - 6

- ✧ Apply the Concepts of Nested Classes for
 - ✧ Solving Complex Problems
 - ✧ Try examples in each category of inner classes
 - ✧ Create interfaces and implement the same with your own logics for a specific problem
 - ✧ For example, create **Printable** Interface and implement it with your logics that could print the output in user preferred colors of RGB coordinate.



Assignments / Penalties



- ✧ Every Student is expected to complete the assignments and strictly follow a fair Academic Code of Conduct to avoid severe penalties
- ✧ Penalties would be heavy for those who involve in:
 - ✧ **Copy and Pasting** the code
 - ✧ **Plagiarism** (copied from your neighbor or friend – in this case, both will get “0” marks for that specific take home assignments)
 - ✧ If the candidate is **unable to explain his own solution**, it would be considered as a “copied case” !!
 - ✧ **Any other unfair means** of completing the assignments

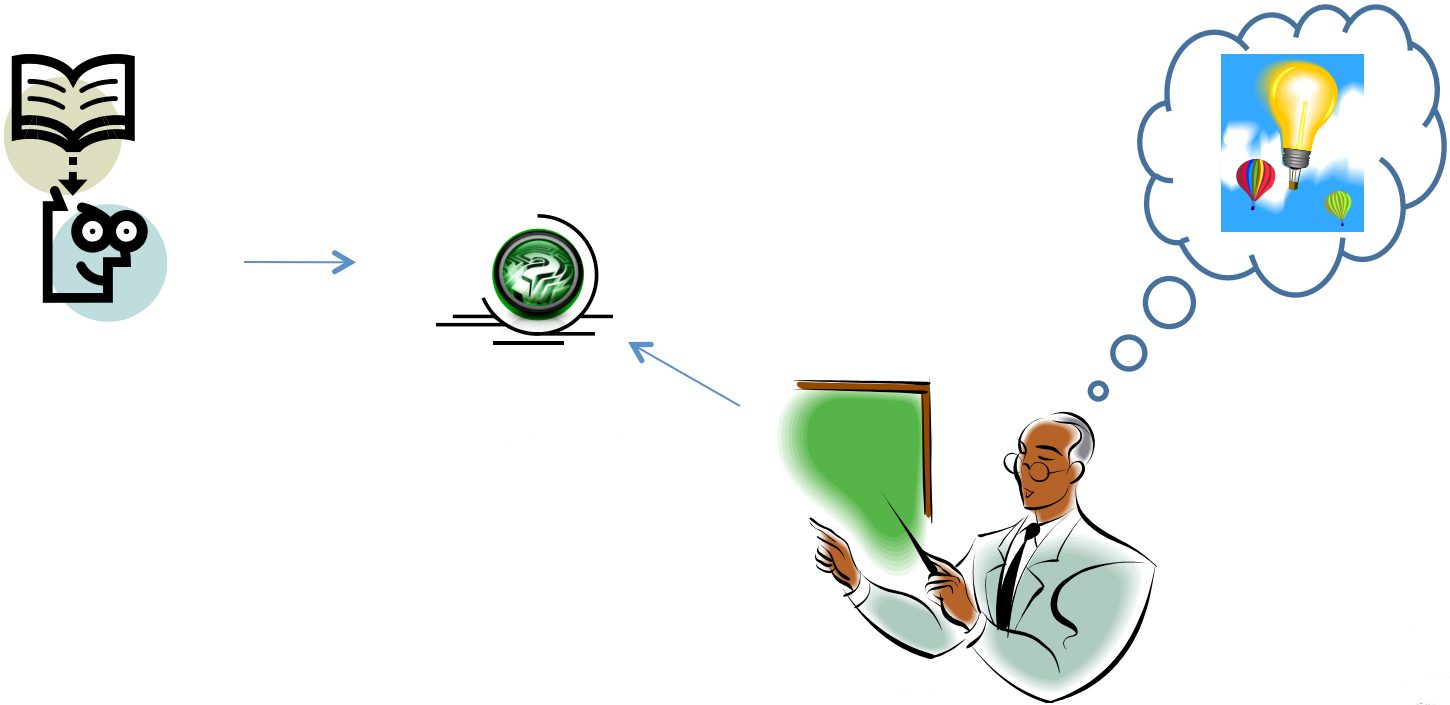


Assistance

- ✧ You may post your questions to me at any time
- ✧ You may meet me in person on available time or with an appointment
- ✧ You may leave me an email any time (email is the best way to reach me faster)



Thanks ...



... Questions ???