Objects & Classes

Fresher Learning Program
January, 2013

People matter, results count.

Objectives of Objects & Classes

- **Purpose:**
  - To understand Classes, its members, object instantiation, constructors, method calling, passing arguments to a method, sub class and super class, overriding and overloading

- **Product:**
  - Understand how to instantiate an object, what is constructor
  - Understand how to call a method and pass arguments
  - Understanding of Static and non static members
  - To know what is sub-class and super class
  - Understand overriding and overloading

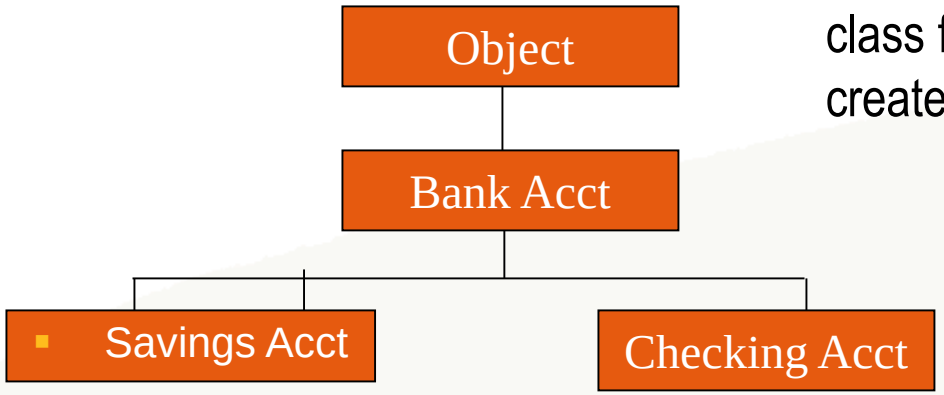
- **Process**:
  - Theory Sessions along with relevant assignments
  - A recap at the end of the session in the form of Quiz.

Table of Contents

- Objects and Classes
  - Concepts of class
  - Class (Static) and instance (non static) variable
  - Method (static and non static methods)
  - Constructor
  - Overloading and Overriding
  - Inner Classes
  - Nested Classes

Concepts : Objects and Classes

- Singly rooted hierarchy



Object class is parent class for all the classes created in Java.

Concepts : Objects and Classes

- Order of code execution (when creating an object)
- static variables initialization.
- static initializer block execution. (in the order of declaration, if multiple blocks found)
- constructor header (super or this – implicit or explicit)
- instance variables initialization / instance initializer block(s) execution
- rest of the code in the constructor

Introduction : Classes

- Class is the logical construct upon which the entire Java language is built because it defines the shape and nature of an Object.

```
public class StudentRecord {
        //we'll add more code here later
}
```

- public - means that our class is accessible to other classes outside the package
- class - this is the keyword used to create a class in Java
- StudentRecord - a unique identifier that describes our class

Instance Variables vs.Static Variables

- Instance Variables
  – Belongs to an object instance
  – Value of variable of an object instance is different from the ones of other object instances

- Class Variables (also called static member variables)
  – variables that belong to the whole class.
  – This means that they have the same value for all the object instances in the same class.

Class Variables Example

Car Class		Object Car A	Object Car B
Instance Variables	Plate Number	ABC 111	XYZ 123
	Color	Blue	Red
	Manufacturer	Mitsubishi	Toyota
	Current Speed	50 km/h	100 km/h
Class Variable	Count = 2		
Instance Methods	Accelerate Method		
	Turn Method		
	Brake Method		

Instance Variables

```
public class StudentRecord {
   // Instance variables
   private String name;
   private String address;
   private int age;
   //we'll add more code here later
}
```

private - here means that the variables are only accessible within the class.

Other objects cannot access these variables directly.

We will cover more about accessibility later.

Static Variables

```
public class StudentRecord {
   //static variables we have declared
   private static int studentCount;
   //we'll add more code here later
}
```

– we use the keyword static to indicate that a variable is a static variable.

Declaring Methods

Defines a particular functionality(behavior) using the properties of a class.

```
Ex:     class A {
                int a,b;
                void AnMethod() {
                        int c=a+b;
                }/
                / “AnMethod “ method defining an addition

                behaviour(Functionality)
                //using the property a & b of class A.
        }
```

Static methods

```
public class StudentRecord {
        private static int studentCount;
        public static int getStudentCount(){
                return studentCount;
        }
}
```

Public - means that the method can be called from objects outside the class

Static - means that the method is static and should be called typing,[ClassName].[methodName].

Declaring Methods

For example, in this case,

we call the method StudentRecord.getStudentCount()

int- is the return type of the method. This means that the method should return a value of type int

getStudentCount- the name of the method

()- this means that our method does not have any parameters

Constructors

- Constructors are important in instantiating an object. It is a method where all the initializations are placed.

- The following are the properties of a constructor:
  – Constructors have the same name as the class
  – A constructor is just like an ordinary method, however only the following information can be placed in the header of the constructor,
  – scope or accessibility identifier (like public...), constructor's name and parameters if it has any.
  – Constructors does not have any return value
  – You cannot call a constructor directly, it can only be called by using the new operator during class instantiation.

Constructors

To declare a constructor, we write,

<modifier> <className> (<parameter>*) {

<statement>*

}

The **default constructor (no-arg constructor)**

– is the constructor without any parameters.

– If the class does not specify any constructors, then an implicit default constructor is created.

```
public StudentRecord() {
        //some code here
}
```

Overloading Constructor Methods

```
public StudentRecord() {
        //some initialization code here
}

public StudentRecord(String temp) {
        this.name = temp;
}

public StudentRecord(String name, String address) {
        this.name = name;
        this.address = address;
}
```

this

“this()” constructor call

- Constructor calls can be chained, meaning, you can call another constructor from inside another constructor.
- We use the this() call for this
- There are a few things to remember when using the this() constructor call:
- When using the this constructor call, IT MUST OCCUR AS THE FIRST STATEMENT in a constructor
- It can ONLY BE USED IN A CONSTRUCTOR DEFINITION. The this call can then be followed by any other relevant statements.

Example

```
public StudentRecord() {
        this("some string");
 }

public StudentRecord(String temp) {
        this.name = temp;
 }

 public static void main( String[] args) {
        StudentRecord annaRecord = new StudentRecord();
 }
```

A sample Hello World Program

```
class A  {
        int i;
        void show(){
                System.out.println("Hello World");
        }
}

public class B {
        public static void main(String args[])       {
                A  obj = new  A();
                obj.show();
        }
}
```

***Output**: Hello World*

Overloading and Overriding :

- Overriding methods :
- the throws clause of the overriding method may only include exceptions that can be thrown by the super class method, including it's subclasses
- the actual method called depends on the object being passed to the method
- Java uses **late-binding** to support polymorphism; which means the decision as to which of the many methods should be used is deferred until runtime

Overloading Methods

- Method overloading
  – allows a method with the same name but different parameters, to have different implementations and return values of different types

  – can be used when the same operation has different implementations.

- Always remember that overloaded methods have the following properties:
  – the same method name
  – different parameters or different number of parameters
  – return types can be different or the same

Example

```
public void print( String temp ){
    System.out.println("Name:" + name);
    System.out.println("Address:" + address);
    System.out.println("Age:" + age);
}

public void print(double eGrade, double mGrade, double sGrade)
    System.out.println("Name:" + name);
    System.out.println("Math Grade:" + mGrade);
    System.out.println("English Grade:" + eGrade);
    System.out.println("Science Grade:" + sGrade);
}
```

Overriding :

- Method Signature - The signature of a method uniquely identifies a method

- The signature consists of the name of the method and the number, type, and order of the arguments required of the method

- The signature of a method is:

  access-modifier return-value-type method-name(parameter-list){ // body }.

  Examples:

  static void TestMethod1(){ }
  public void TestMethod2("Bill") { }
  private void TestMethod3("Bill", 55) { }

Inner Classes : Objects and Classes

Inner classes are non-static classes defined within other classes.

```
class Outer {
        class Inner {
                // code inside inner class
        }
}
```

the compiled class files for the above are: Outer.class and Outer$Inner.class

the Inner class type is: Outer.Inner

Inner Classes : Objects and Classes

Instances of inner classes can be created in a number of ways

Create an Outer class object:
Outer o1 = new Outer();

Then create an Inner class object:
Outer.Inner i1 = o1.new Inner();

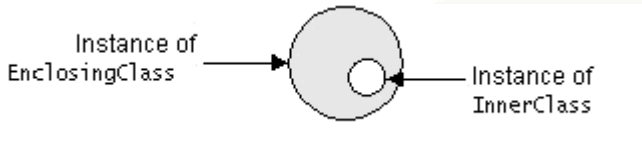
Or

Create the inner class directly:
Outer.Inner i2 = new Outer().new Inner();

Or,

create one from within the outer class constructor

```
class Outer {
        Outer() {
                new Inner();
        }
}
```



Inner Classes : Objects and Classes

Inner classes may be declared **default** (friendly), **public, protected, private, abstract, static** or **final**

e.g. :

```
class Outer {
        class Inner {}
        public class PublicInner{}
        protected class ProtectedInner {}
        private class PrivateInner{}
        abstract class AbstractInner {}
        final class FinalInner {}
        static class StaticInner {}
}
```

Inner Classes : Objects and Classes

- Each instance of a non-static inner class is associated with an instance of their outer class. static inner classes are a special case called as Nested Classes.

- inner classes may not declare static initializers or static members unless they are compile time constants i.e. static final var = value;

- you cannot declare an interface as a member of an inner class; interfaces are never inner.

Inner Classes : Objects and Classes

- inner classes may inherit static members.
- the inner class can access the variables and methods declared in the outer class
- to refer to a field or method in the outer class instance from within the inner class, use Outer.this.fldname

Nested Classes : Objects and Classes

- To make objects of a inner class type independent of objects of the enclosing class, you can declare the inner class as static. Such static inner class are called as Nested Classes

- a **static inner class** behaves like any **top-level** class except that its name and accessibility are defined by its enclosing class i.e. use new Outer.Inner() when calling from another class

- formally called **top-level nested classes**

Nested Classes : Objects and Classes

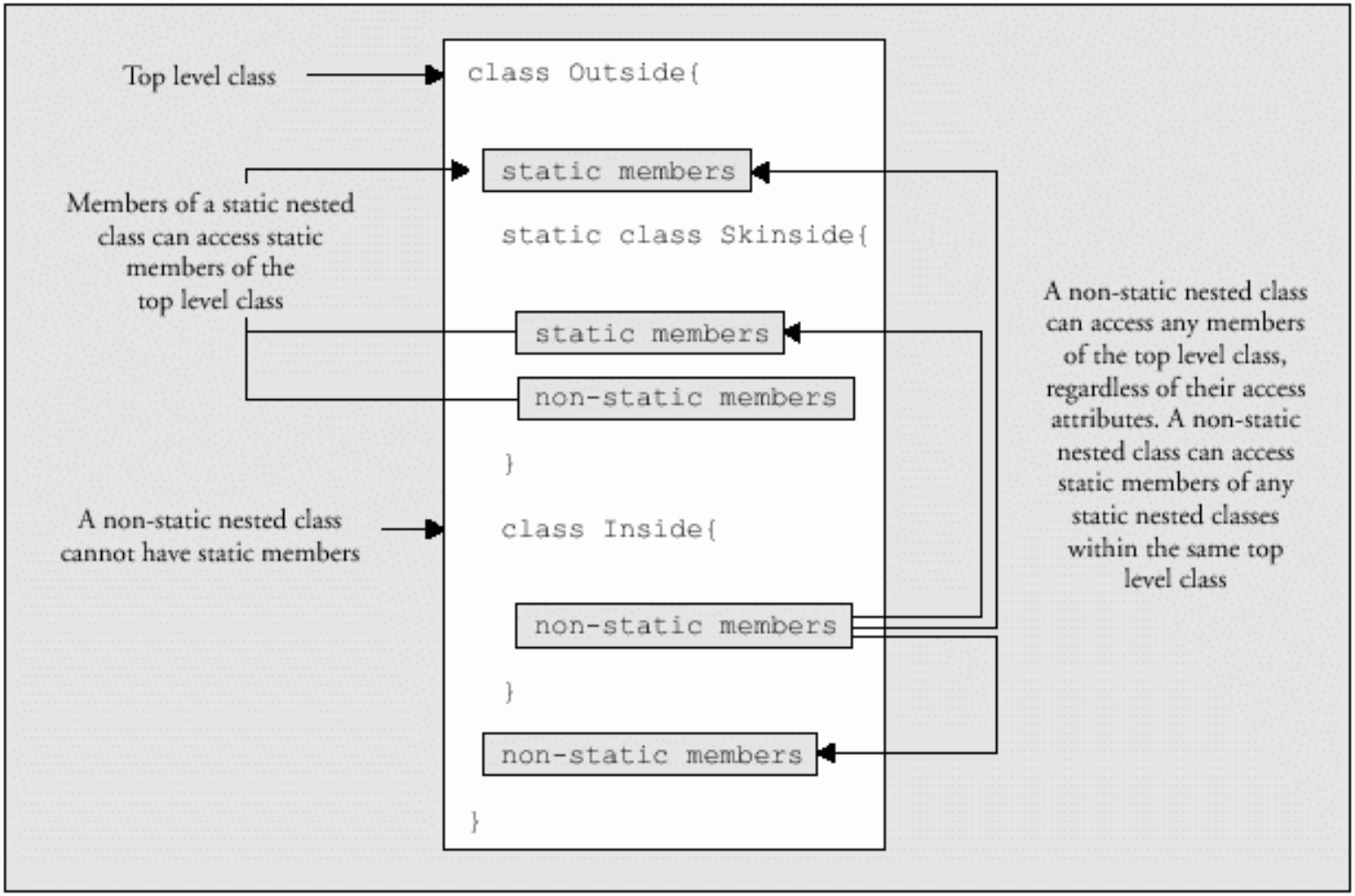
```
class Outer {
        public static void main (String[] args) {
                int x = Inner.value;
        }

        static class Inner {
                static int value = 100;
        }
}
```

they are not associated with an instance of their outer class i.e. you can create an Inner class object from within the Outer class using new Inner(); you do not need to create an Outer class object first as is required with non-static inner classes

static inner classes can directly access static fields of the outer class but must use an instance of the outer class to access the outer classes instance fields

Inner Classes / Nested Classes : Objects and Classes



Recap (Important Keywords)

Overriding

this

Constructor

Inner class

Nested class

Sub class

Static

Overloading

Thank You For Your Time

People matter, results count.