Session 04
More on Classes and Nested Classes

(http://docs.oracle.com/javase/tutorial/java/javaOO/more.html)

Objectives

- In this lesson you will learn:
  - Controlling Access to Members of a Class using modifiers
  - Overriding methods in sub-classes
  - Nested Classes

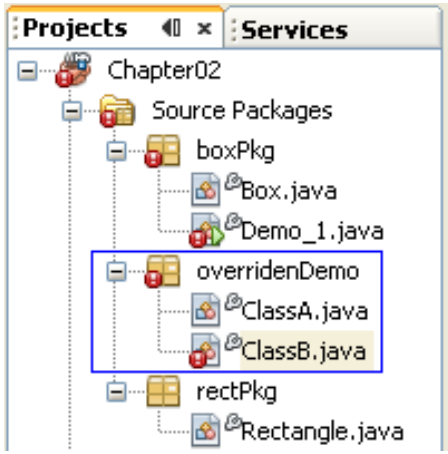
Review: Access Level

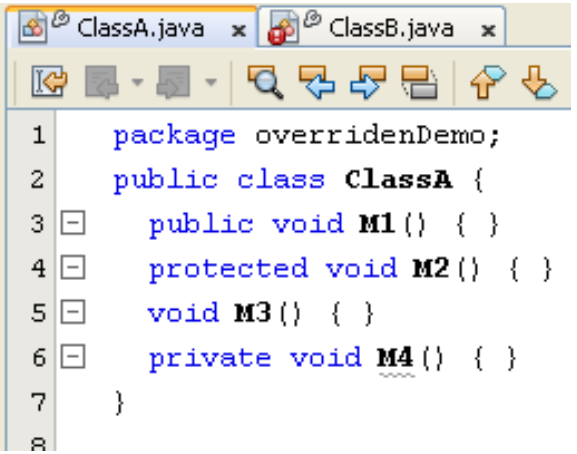
Modifier	Class	Same Package	Subclass-Outside package	World
private	Y	N	N	N
No (default)	Y	Y	N	N
protected	Y	Y	Y	N
public	Y	Y	Y	Y

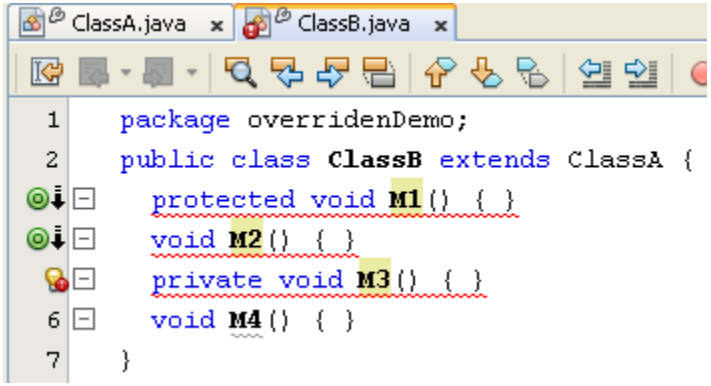
Tips on Choosing an Access Level

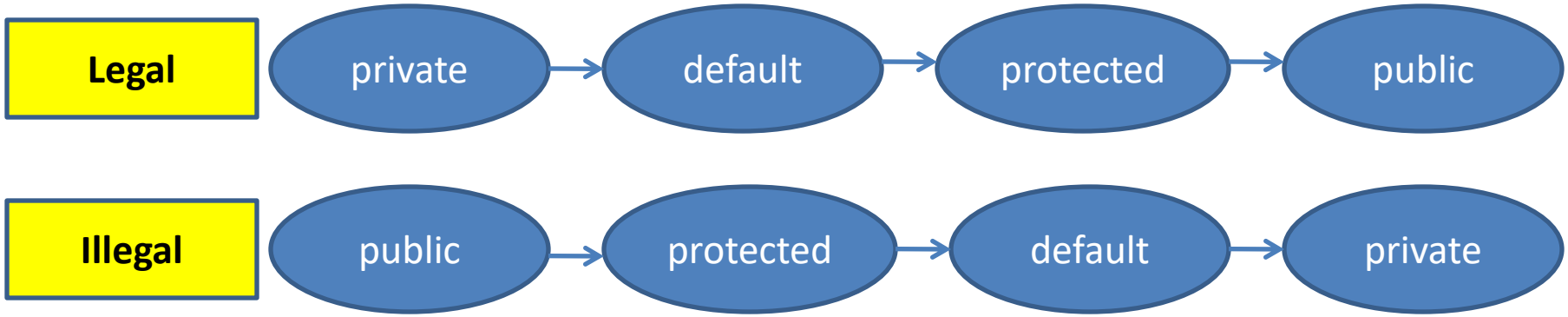
- Use the most restrictive access level that makes sense for a particular member. Use private unless you have a good reason not to.
- Avoid public fields except for constants. Public fields tend to link you to a particular implementation and limit your flexibility in changing your code.

Access Modifier Overridden









The sub-class must be more opened than it's father

Modifier *final*

- Final class: Class can not have sub-class
- Final data is a constant.
- Final method: a method can not be overridden.

OtherModifierDemo.java *

```
1
2   public class OtherModifierDemo extends java.lang.Math {
3   }
```

```
2   public class OtherModifierDemo{
3       final public int MAXN = 5;
4       public static void main(String[] args)
5       { OtherModifierDemo obj= new OtherModifierDemo();
6         obj.MAXN = 1000;
7         final int N=7;
8         N=10;
9       }
10  }
```

```
1   class A{
2       final void M() { System.out.println("MA");
3   }
4   class B extends A {
5     void M() { System.out.println("MB");
6   }
7   public class FinalMethodDemo {
8
9   }
```

Modifier *static*
Class variable/ Object variable

- Object variable: Variable of each object
- Class Variable: A variable is shared in all objects of class. It is stored separately. It is declared with the modifier ***static***
- Class variable is stored separately. So, it can be accessed as:

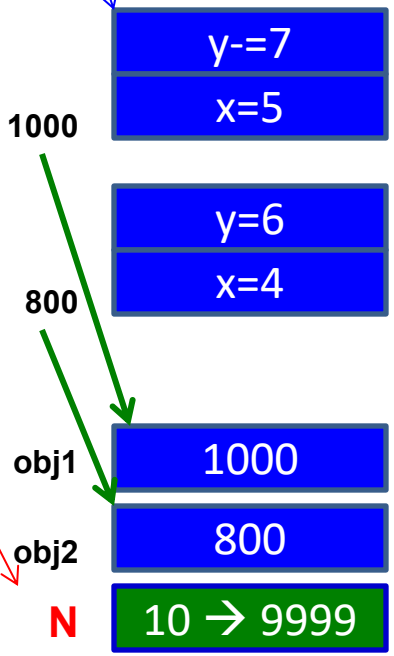
  object.staticVar

  ClassName.staticVar

Modifier *static*: Class variable/ Object variable

```
public class StaticVarDemo {
    static int N=10; // class variable
    int x, y; // object variable
    public StaticVarDemo(int xx, int yy){
        x= xx; y=yy;
    }
    public void setN( int nn){
        N= nn;
    }
    public void output(){
      System.out.println(N + "," + x + "," + y);
    }
}
```

```
public class StaticVarDemoUse {
    public static void main(String args[]){
        StaticVarDemo obj1= new StaticVarDemo(5,7);
        StaticVarDemo obj2= new StaticVarDemo(4,6);
        obj1.output();
        obj2.output();
        obj1.setN(9999);
        obj1.output();
        obj2.output();
        System.out.println(StaticVarDemo.N);
    }
}
```




```
Output - FirstPrj (run)
run:
10,5,7
10,4,6
9999,5,7
9999,4,6
9999
```

Modifier *static*: static code – Free Floating Block

```
public class StaticCodeDemo {
    public static int N=10;
    int x=5, y=7;
    static {
        System.out.println("Static code:" + N);
    }
    int sum(){
        return x+y;
    }
    static {
        System.out.println("Static code: Hello");
    }
}
```

All static code run only one time when the first time the class containing them is accesses

```
public class StaticCodeDemoUse {
    public static void main(String args[]){
        System.out.println(StaticCodeDemo.N);
        StaticCodeDemo obj= new StaticCodeDemo();
        System.out.println(obj.sum());
    }
}
```

The second access

```
Output - FirstPrj (run)
run:
Static code:10
Static code: Hello
10
12
```

Modifier *static*: Static method

- It is called as class method/global method and it is called although no object of this class is created.
- Entry point of Java program is a static method
- Syntax for calling it: ClassName.staticMethod(args)
- ***Static methods***:
  - can access class variables and class methods directly **only**.
  - ***cannot*** access instance variables or instance methods directly—they must use an object reference.
  - cannot use the this keyword as there is no instance for this to refer to.

Modifier *static*: Static code/ Free-floating block

What can free-floating block contain?

- Initialize class (static) variable
- Calling **static methods** with parameter

Modifier *static*: What should be static in a class?

- Constants:
  - The static modifier, in combination with the final modifier, is also used to define constants. The final modifier indicates that the value of this field cannot change.

    static final double PI = 3.141592653589793;

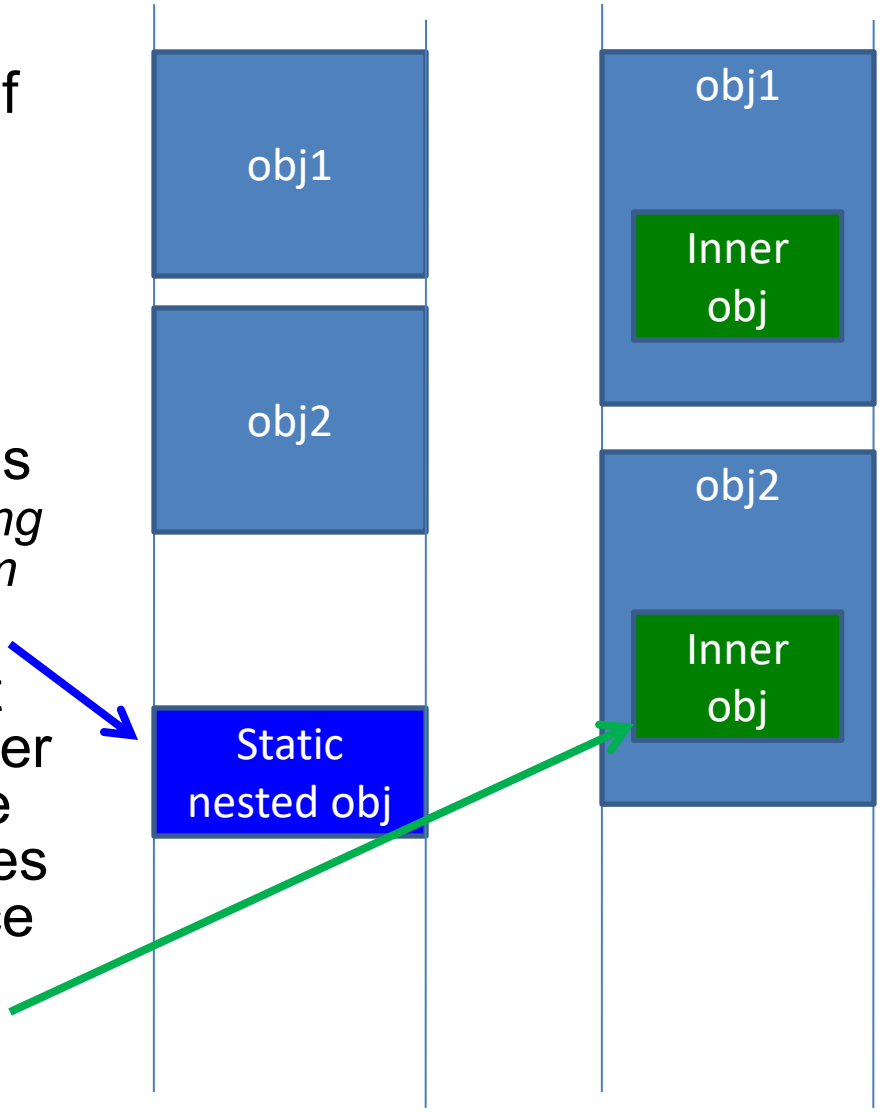
Initializing Fields (3)

- Initializing Instance Members
  - Will be copied initializer blocks into every constructor.
  - Can be used to share a block of code between multiple constructors.
  - Initializer blocks for instance variables look just like static initializer blocks, but without the static keyword:

```
{
  // whatever code is needed for initialization goes here
}
```

Nested classes

- Class is declared inside some other class or inside the body of other class's method.
- 2 types of nested class: Static Nested classes, Inner classes (non-static)
- **Static nested class** → All objects belong to the outer class share one nested object(*cặp song sinh dính nhau, phần dính nhau nằm ở bên ngoài*).
- **Inner class**: Each outer object has it's own inner object → Outer object must contain an instance of the inner object then accesses members of this nested instance (*trái tim nằm bên trong thân người)*



Nested classes…

A nested class violates the recommendation “low coupling” in class design. Why are nested classes used?

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

Static Nested Classes Demo…

- Class-level nested class.
- Because the static nested object is stored separately from enclosing instances, static nested object can be initiated without enclosing objects.

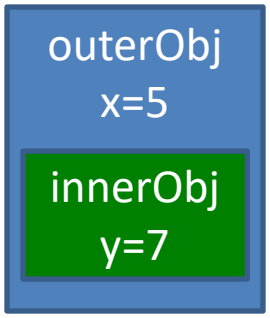
```
1    public class OuterClass3 {
2      int x;
3      public static class MyInner {
4          int n=10;
5          void print() { System.out.println(n +x); }
6      }
7    }
```

```
public class OuterClass3 {
  int x = 1;
  static class MyInner {
      int n=10;
      void print() { System.out.println(n);}
  }
  public static void main(String args[]){
      OuterClass3.MyInner obj= new OuterClass3.MyInner();
      obj.print();
  }
}
```

```
Output
Debugger Console x  Chapter06 (run) x
run:
10
```

Methods of static nested class can not access data of enclosing instance because it can created even when enclosing objects do not exist.

Inner classes demo…



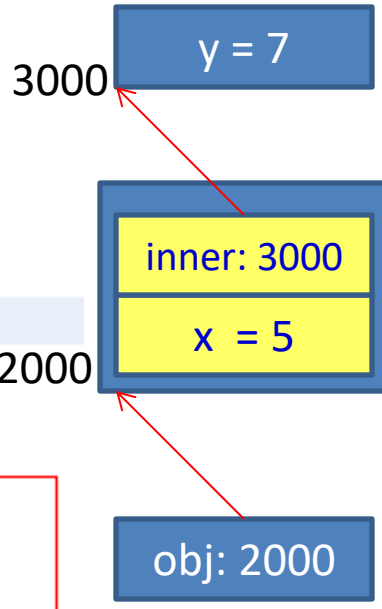
A method of nested class can access all members of its enclosing class

```
public class OuterClass {
    int x=5;
    class Inner1 {
        int y= 7;
        void M_Inner(){
            System.out.print(x+y);
        }
    }
    void M_Outer(){
        System.out.print(x+y);
    }
}
```

Error: inner obj do not created yet. Modified

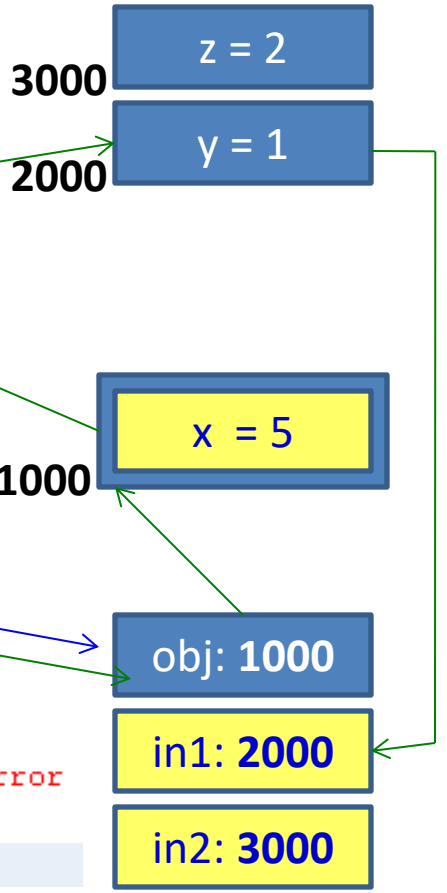
```
1  public class OuterClass {
2      int x=5;
3      class Inner1 {
4          int y= 7;
5          Inner1(){
6              System.out.println(2*x+y);
7          }
8          void printInner(){
9              System.out.println(y);
10         }
11     }
12     Inner1 inner= new Inner1();
13     void M_Outer(){
14         System.out.println(x + inner.y);
15     }
16     public static void main(String[] args){
17         OuterClass obj= new OuterClass();
18         obj.M_Outer();
19     }
20 }
```

```
Output - Chapter06 (run)
run:
17
12
```



Creating Inner Class instance through enclosing instance

```
1   public class OuterClass2 {
2     int x = 5;
3     class Inner1 {
4       int y = 1;
5       public void print() { System.out.println(y);}
6     }
7     public class Inner2 {
8       int z = 2;
9       public void print() { System.out.println(z);}
10    }
11    public static void main(String[] args){
12      OuterClass2 obj = new OuterClass2();
13      OuterClass2.Inner1 in1 = obj.new Inner1();
14      OuterClass2.Inner2 in2 = obj.new Inner2();
15      in1.print();
16      in2.print();
17      // OuterClass2.Inner1 in11 = new OuterClass2.Inner1(); // error
18    }
19  }
public class Use_OuterClass2 {
 public static void main(String[] args){
    OuterClass2.Inner1 in1 = new OuterClass2().new Inner1();
    in1.print();
  }
}
```




```
Output - Chapter06 (run)
run:
1
```

Inner Classes
Defined Inside Methods

```
1   public class OuterClass3 {
2     int x = 1;
3     public void M() {
4         final int t = 2;// inner class accesses final local variable only
5         class Inner {
6             int y = 3;
7             void print() { System.out.println(y + x + t);}
8         }
9         Inner objInner= new Inner();
10        objInner.print();
11    }
12    public static void main(String args[]){
13        OuterClass3 obj= new OuterClass3();
14        obj.M();
15    }
16  }
```

Local class

Inner-method class can not access normal local variables of the containing method.

```
Output - Chapter06 (run)
run:
6
BUILD SUCCESSFUL (total time: 0 seconds)
```

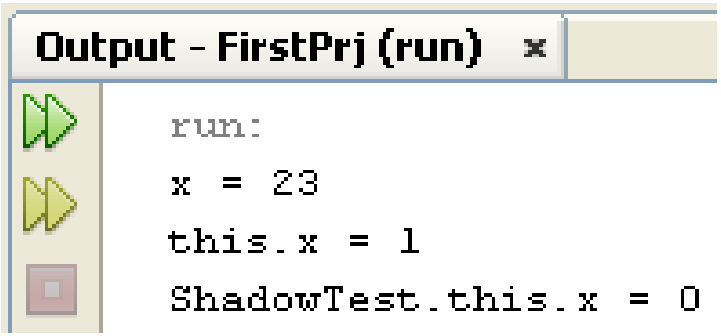
Shadowing (1)

- Declaration of a type in a particular scope has the same name as another declaration in the enclosing scope, then the declaration ***shadows*** the declaration of the enclosing scope.

Shadowing Variable (2)

```
public class ShadowTest {
    public int x = 0;
    class FirstLevel {
        public int x = 1; // shadowing

        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);
    }
}
```



Mechanism: Local data is treated first

Summary

- Overriding methods in sub-classes
- Controlling Access to Members of a Class using modifiers
- Nested Classes