Review of Basic Concepts

Topics

- Object-Oriented Concepts
- Declaring a class
- Declaring properties in a class
- Declaring methods in a class
- Declaring constructor methods
- Creating object instances
- Accessing object members (properties, methods)
- Access modifiers

Topics

- Object-Oriented Concepts In Java program
  - encapsulation
  - inheritance
- Overriding methods (of a super class) by the ones in sub-class
- Abstract class and abstract methods
- Interface
- “this”, “super”, “static”, “final” keywords
- Inner classes

Object Oriented (OO) Concepts

Object-Oriented Concepts

- Object-Oriented Design
  - Focuses on object and classes based on real world scenarios
  - Emphasizes state, behavior and interaction of objects
  - Advantages:
    - Faster development
    - Increased quality
    - Easier maintenance
    - Enhanced modifiability
    - Increase software reuse
- Class
  - Allows you to define new data types
  - Considered as a blueprint or template to create objects

Object-Oriented Concepts

- Object (instance)
  - An entity that has a state, behavior and identity with a well-defined role in problem space
  - An actual instance of a class
  - Created every time you instantiate a class using the *new* keyword.
- Attribute (property, field)
  - Data element of an object
  - Stores information about the object
  - A.K.A. Data member, instance variable, property, data field

Object-Oriented Concepts

- Method
  - Describes the behavior of an object
  - Also called a function or a procedure

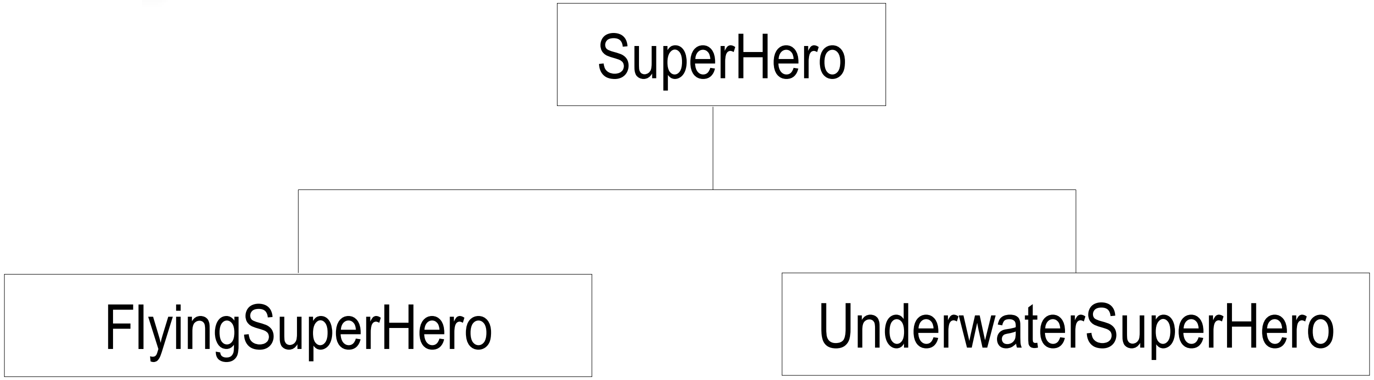
- Constructor (method)
  - Method-like
  - For creating and initializing a new object

Object-Oriented Concepts

- Package
  - Grouping of classes and/or sub-packages
  - Analogous to a directory of a file system
- Encapsulation
  - Principle of hiding the implementation details not relevant to the user of the class
- Abstraction
  - Ignoring aspects of a subject not relevant to the current purpose to focus on those that are

Object-Oriented Concepts

- Inheritance
  - Relationship between classes wherein one class is the superclass or the parent class of another
  - Refers to the properties and behaviors received from an ancestor
  - Also know as a "is-a" relationship



Object-Oriented Concepts

- Polymorphism
  - "poly" means many while "morph" means form
  - Ability of an object to assume may different forms

- Interface
  - Contract in the form of a collection of method and constant declarations
  - Implementing class promises to follow the contract.

Declaring a Class

Java Program Structure: Declaring Classes

- Syntax

```
<classDeclaration> ::=
   <modifier> class <name> {
      <attributeDeclaration>*
      <constructorDeclaration>*
      <methodDeclaration>*
   }
```

where

- Modifier
  - Refers to an access modifier or other types of modifiers
- Name
  - Is an identifier for the name of the class

Java Program Structure: Declaring Classes

```
class SuperHero {
   String superPowers[];
   void setSuperPowers(String superPowers[])  {
      this.superPowers = superPowers;
   }
   void printSuperPowers() {
      for (int i = 0; i < superPowers.length; i++) {
         System.out.println(superPowers[i]);
      }
   }
}
```

Declaring Properties (Attributes)

Java Program Structure: Declaring Attributes

- Syntax:

```
<attributeDeclaration> ::=
   <modifier> <type> <name> [=
  <default_value>];


<type> ::=
   byte | short | int | long | char | float
  | double  | boolean | <class>
```

Java Program Structure: Declaring Attributes

```
public class AttributeDemo {
   private String studNum;
   public boolean graduating = false;
   protected float unitsTaken = 0.0f;
   String college;
}
```

Declaring Methods

Java Program Structure: Declaring Methods

- Syntax:

```
<methodDeclaration> ::=
   <modifier> <returnType> <name>
  (<parameter>*) {
      <statement>*
   }
<parameter> ::=
   <parameter_type> <parameter_name>[,]
```

Java Program Structure: Declaring Methods

```
class MethodDemo {
   int data;
   int getData() {
      return data;
   }
   void setData(int data) {
      this.data = data;
   }
   void setMaxData(int data1, int data2) {
      data = (data1>data2)? data1 : data2;
   }
}
```

Declaring Constructors

Java Program Structure: Declaring a Constructor

- Syntax:

```
<constructorDeclaration> ::=
   <modifier> <className> (<parameter>*) {
      <statement>*
   }
```

where

- Modifier
  - Can be any access modifier but not other types of modifiers.

- Default constructor (no-arg constructor)
  - No arguments

Java Program Structure: Declaring a Constructor

```
class ConstructorDemo {
   private int data;
   public ConstructorDemo() {
      data = 100;
   }
   ConstructorDemo(int data) {
      this.data = data;
   }
}
```

Creating an Object Instance

Java Program Structure: Instantiating a Class

- Syntax:

```
new <constructorName>(<parameters>)
```

- Example:

```
class ConstructObj {
   int data;
   ConstructObj() {
      /* initialize data */
   }
   public static void main(String args[]) {
      ConstructObj obj = new ConstructObj();
  }
}
```

Accessing Object Members

Java Program Structure: Accessing Object Members

- Dot notation:

```
<object>.<member>
```

- Some examples:

```
String myString = new String(“My String”);
//Access length method
System.out.println(“Length: ” + myString.length());

int intArr = {1, 2, 3, 4, 5};
//Access length attribute
System.out.println(“Length: ” + intArr.length);
```

Java Program Structure: Accessing Object Members

```
class ConstructObj {
    int data;
    ConstructObj() {
        /* initialize data */
    }
    void setData(int data) {
        this.data = data;
    }
    public static void main(String args[]) {
        ConstructObj obj = new ConstructObj();
        obj.setData = 10;     //access setData()
        System.out.println(obj.data); //access data
    }
}
```

Packages

Java Program Structure: Packages

- Syntax for indicating that the code belongs to a package:

```
<packageDeclaration> ::=

   package <packageName>;
```

- Syntax for importing other packages:

```
<importDeclaration> ::=

   import <packageName.elementAccessed>;
```

- Source code format:

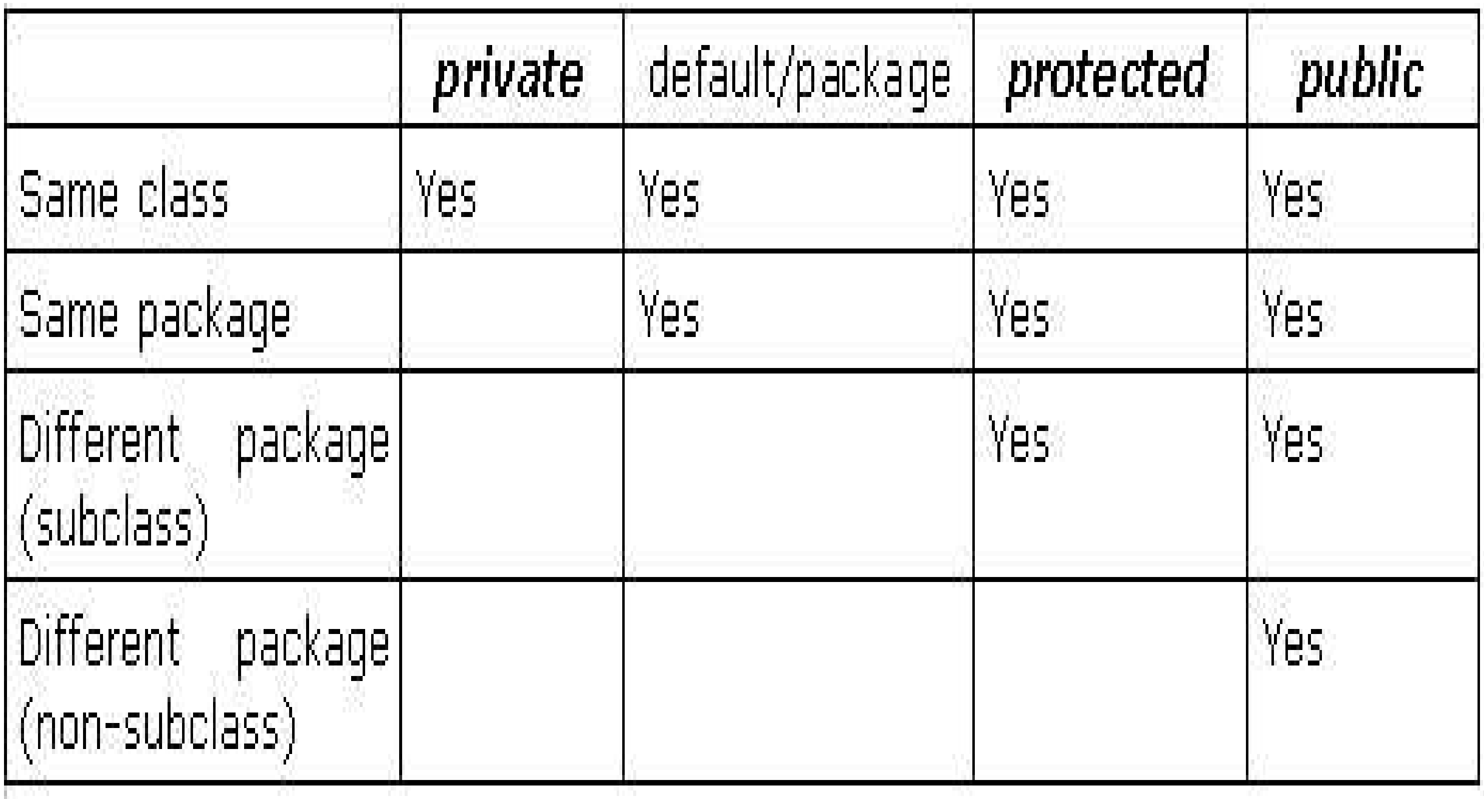
```
[<packageDeclaration>]

<importDeclaration>*

<classDeclaration>+
```

Java Program Structure: Packages

```
package registration.reports;
import registration.processing.*;
import java.util.List;
import java.lang.*;  //imported by default
class MyClass {
   /* details of MyClass */
}
```

Access Modifiers

Java Program Structure: The Access Modifiers

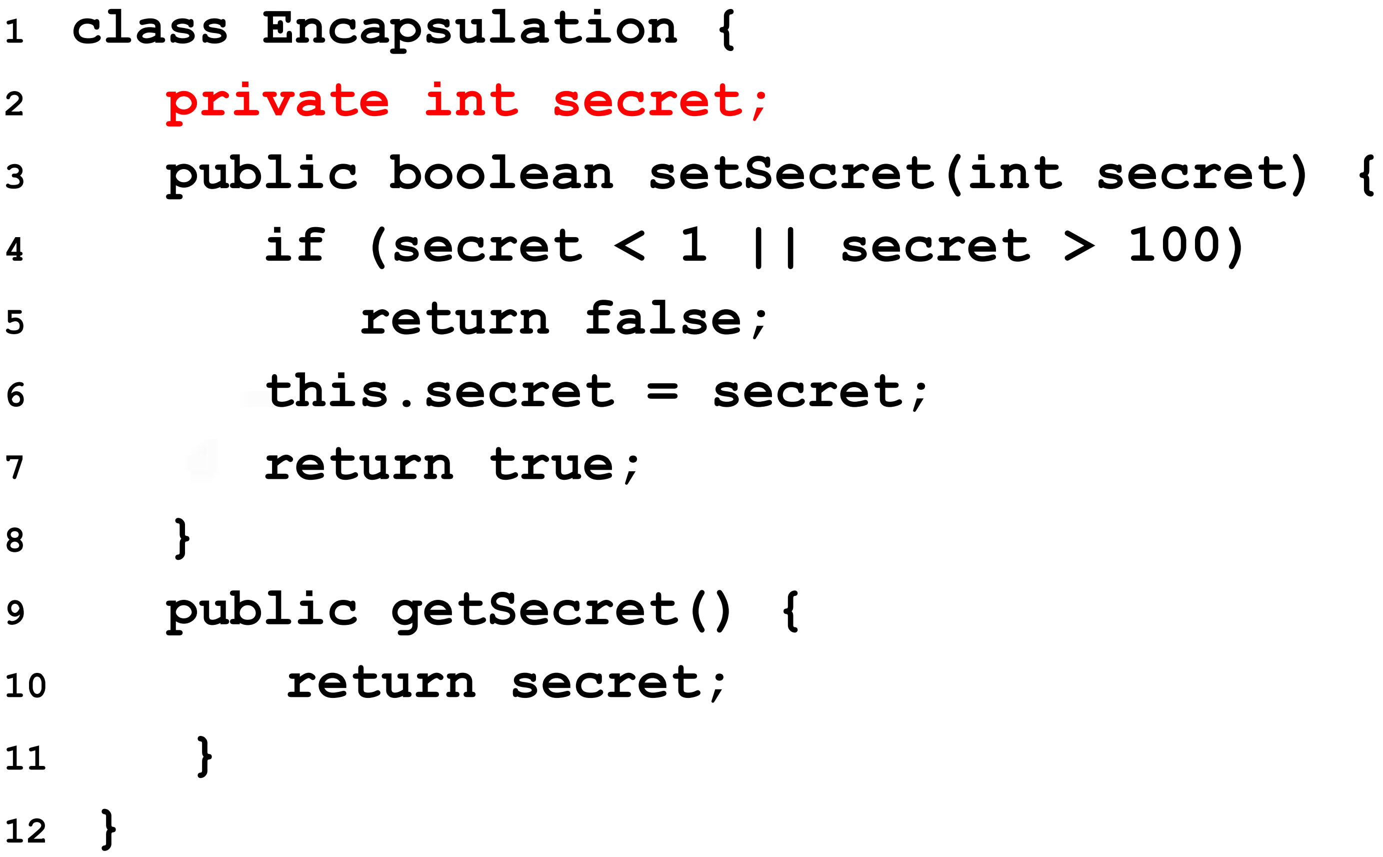


	private	default/package	*protected*	*public*
Same class	Yes	Yes	Yes	Yes
Same package		Yes	Yes	Yes
Different package (subclass)			Yes	Yes
Different package (non-subclass)				Yes

OO Concepts in Java program: Encapsulation, Inheritance

Java Program Structure: Encapsulation

- Hide members by making them *private*
- Example



```
class Encapsulation {
   private int secret;
   public boolean setSecret(int secret) {
      if (secret < 1 || secret > 100)
         return false;
      this.secret = secret;
      return true;
   }
   public getSecret() {
       return secret;
    }
 }
```

Java Program Structure: Inheritance

- Creating a child class or a subclass:
  - Use *extends* in declaring the class
  - Syntax:

```
class <childClassName> extends <parentClassName>
```

- A class can only extend one parent class

Java Program Structure: Inheritance

```
import java.awt.*;

class Point {
    int x;
    int y;
}

class ColoredPoint extends Point {
    Color color;
}
```

Overriding Methods (by Sub-Class)

Java Program Structure: Overriding Methods

- Subclass defines a method whose signature is identical to a method in the superclass

- Signature of a method
  - Information found in the method header definition
    - Return type
    - Method name
    - Parameter list of the method

- Different from method overloading!

Java Program Structure: Overriding Methods

```
class Superclass {
   void display(int n) {
      System.out.println("super: " + n);
   }
}

class Subclass extends Superclass {
   void display(int k) {
      System.out.println("sub: " + k);
    }
 }

 // continued...
```

Java Program Structure: Overriding Methods

```
class OverrideDemo {
    public static void main(String args[]) {
        Subclass SubObj = new Subclass();
        Superclass SuperObj = SubObj;
        SubObj.display(3);
        ((Superclass)SubObj).display(4);
    }
}
```

Java Program Structure: Overriding Methods

- Version of method called
  - Based on actual type of the object that invoked the method
- Access modifier for the methods need not be the same
  - Access modifier of the overriding method
    - Same access modifier as that of the overridden method
    - Less restrictive access modifier

Java Program Structure: Overriding Methods

```
class Superclass {
   void overriddenMethod() {
   }
}
class Subclass1 extends Superclass {
   public void overriddenMethod() {
   }
}
class Subclass2 extends Superclass {
    void overriddenMethod() {
    }
 }
 //continued...
```

Java Program Structure: Overriding Methods

```
/* class Superclass {
   void overriddenMethod() {
   }
} */
class Subclass3 extends Superclass {
   protected void overriddenMethod() {
   }
}
class Subclass4 extends Superclass {
   private void overriddenMethod() {
   }
}
```

Abstract Class & Abstract Methods

Java Program Structure: Abstract Classes and Methods

- Syntax:

```
abstract <modifier> <returnType> <name>
  (<parameter>*);
```

- Class containing an *abstract* method should be declared *abstract*
  - `abstract class <name> {`
  - `   /* constructors, fields and methods */`
  - `}`

Java Program Structure: Abstract Classes and Methods

- *abstract* keyword is not for:
  - Constructor
  - static method

- *abstract* classes cannot be instantiated

- Classes that extends an abstract class:
  - Should implement all *abstract* methods
  - Otherwise, the subclass itself should be declared *abstract*

Java Program Structure: Abstract Classes and Methods

```
abstract class SuperHero {
   String superPowers[];
   void setSuperPowers(String superPowers[])  {
      this.superPowers = superPowers;
   }
   void printSuperPowers() {
      for (int i = 0; i < superPowers.length; i++) {
         System.out.println(superPowers[i]);
      }
    }
    abstract void displayPower();
 }
 //continued...
```

Java Program Structure: Abstract Classes and Methods

```
class FlyingSuperHero extends SuperHero {
   void displayPower() {
      System.out.println("Fly...");
   }
}

class Spiderman extends SuperHero {
   void displayPower() {
      System.out.println("Communicate with sea" +
                          " creatures...");
      System.out.println("Fast swimming ability...");
    }
 }
```

Interface

Java Program Structure: Interface

- Syntax:

```
<interfaceDeclaration> ::=
    <modifier> interface <name> {
        <attributeDeclaration>*
        [<modifier> <returnType> <name>
                                   (<parameter>*);]*
    }
```

- Members are *public* when the interface is declared *public*

Java Program Structure: Interface

- Interface attributes:
  - Implicitly *static* and *final*
  - Must be initialized

- Modifiers:
  - Access modifiers: *public*, package
  - Must be initialized

- Implementing an interface:
  - Use *implements* keyword
  - Should implement all the interface's methods
  - A class can implement several interfaces

Java Program Structure: Interface

```
interface MyInterface {
    void iMethod();
}

class MyClass1 implements MyInterface {
    public void iMethod() {
        System.out.println("Interface method.");
    }
    void myMethod() {
         System.out.println("Another method.");
     }
 }
 //continued...
```

Java Program Structure: Interface

```
class MyClass2 implements MyInterface {
    public void iMethod() {
        System.out.println("Another implementation.");
    }
}
class InterfaceDemo {
    public static void main(String args[]) {
        MyClass1 mc1 = new MyClass1();
        MyClass2 mc2 = new MyClass2();
        mc1.iMethod();
        mc1.myMethod();
        mc2.iMethod();
    }
}
```

“this” keyword

Java Program Structure: The *this* Keyword

- Why *this*?
  1. Disambiguate local attribute from a local variable
  2. Refer to the object that invoked the non-static method
  3. Refer to other constructors

Java Program Structure: The *this* Keyword

- Disambiguate local attribute from a local variable

```
class ThisDemo1 {
   int data;
   void method(int data) {
      this.data = data;
      /*
         this.data refers to the attribute
         while data refers to the local variable
      */
   }
}
```

Java Program Structure: The *this* Keyword

- Refer to the object that invoked the non-static method

```
class ThisDemo2 {
   int data;
   void method() {
      System.out.println(data);  //this.data
   }
   void method2() {
      method();   //this.method();
   }
}
```

Java Program Structure: The *this* Keyword

- Method Overloading
  - Different methods within a class sharing the same name
  - Parameter lists should differ
    - Number of parameters
    - Type of parameters
  - Constructors can also be overloaded
  - An example:

```
class MyClass {
    void myMeth() {}
    void myMeth(int i) {}
    void myMeth(int i, int j) {}
}
```

Java Program Structure: The *this* Keyword

- Refer to other constructors

```
class ThisDemo3 {
   int data;
   ThisDemo3() {
      this(100);
   }
   ThisDemo3(int data) {
      this.data = data;
   }
}
```

- Call to *this()* should be the first statement in constructor

“super” keyword

Java Program Structure: The *super* Keyword

- Related to inheritance
  - Invoke superclass constructors
  - Can be used like the *this* keyword to refer to members of the superclass

- Calling superclass constructors

```
class Person {
   String firstName;
   String lastName;
   Person(String fname, String lname) {
      firstName = fname;
      lastName = lname;
   }
}
```

Java Program Structure: The *super* Keyword

```
//continuation...
 class Student extends Person {
  String studNum;
  Student(String fname, String lname, String sNum)  {
       super(fname, lname);
       studNum = sNum;
    }
 }
```

- *super()*
  - Refers to the immediate superclass
  - Should be first statement in the subclass's constructor

Java Program Structure: The *super* Keyword

- Referring to superclass members

```
class Superclass{
   int a;
   void display_a(){
      System.out.println("a = " + a);
   }
}

//continued...
```

Java Program Structure: The *super* Keyword

```
class Subclass extends Superclass {
    int a;
    void display_a(){
        System.out.println("a = " + a);
    }
    void set_super_a(int n){
        super.a = n;
    }
    void display_super_a(){
        super.display_a();
    }
}
```

Java Program Structure: The *super* Keyword

```
class SuperDemo {
   public static void main(String args[]){
      Superclass SuperObj = new Superclass();
      Subclass SubObj = new Subclass();
      SuperObj.a = 1;
      SubObj.a = 2;
      SubObj.set_super_a(3);
      SuperObj.display_a();
      SubObj.display_a();
      SubObj.display_super_a();
      System.out.println(SubObj.a);
   }
}
```

“static” keyword

Java Program Structure: The *static* Keyword

- Applied to members of a class:
  - Attributes
  - Methods
  - Inner classes
- Allows accessing of static or class members without instantiation
- Class variables
  - Behave like a global variable
  - Can be accessed by all instances of the class

Java Program Structure: The *static* Keyword

- Class methods
  - May be invoked without creating an object of its class
  - Can only access static members of the class
  - Cannot refer to *this* or *super*

- *static* blocks
  - Executed only once, when the class is loaded
  - For initializing class variables

Java Program Structure: The *static* Keyword

```
class Demo {
    static int a = 0;
    static void staticMethod(int i) {
        System.out.println(i);
    }
    static {   //static block
        System.out.println("static block");
        a += 1;
    }
}

//continued...
```

Java Program Structure: The *static* Keyword

```
class StaticDemo {
   public static void main(String args[]) {
      System.out.println(Demo.a);
      Demo.staticMethod(5);
      Demo d = new Demo();
      System.out.println(d.a);
      d.staticMethod(0);
      Demo e = new Demo();
      System.out.println(e.a);
      d.a += 3;
      System.out.println(Demo.a+", "+d.a+", "+e.a);
   }
}
```

Java Program Structure: The *final* Keyword

- Applied to variables, methods and classes
- Restricts what we can do with the variables, methods and classes
- *final* variable
  - Cannot be modified once its value has been set
  - Example:
    - `final int data = 10;`
    - `data++;`

“final” keyword

Java Program Structure: The *final* Keyword

- *final* method
  - Cannot be overridden
  - Example:

```
final void myMethod() {   //in a parent class
}
void myMethod() {   //in a child class
}
```

- *final* class
  - Cannot be inherited
  - Example:
    - `final public class MyClass {}`
    - `class WrongClass extends MyClass {}`

Java Program Structure: The *final* Keyword

- Keyword may be placed before after other modifiers

```
public final static void meth() {} or
final public static void meth() {} or ...
//order of modifiers is not important
```

Inner Classes

Java Program Structure: Inner Classes

- Class declared within another class

- Accessing the members of the inner class:
  - Need to instatiate an inner class member first
  - Example:

```
innerObj.innerMember = 5;
//innerObj is an instance of the inner class
//innerMember is a member of the inner class
```

Java Program Structure: Inner Classes

- Methods of the inner class can directly access members of the outer class
  - Example:

```
class Out {
   int OutData;
   class In {
      void inMeth() {
         OutData = 10;
      }
   }
}
```

Java Program Structure: Inner Classes

```
class OuterClass {
   int data = 5;
   class InnerClass {
      int data2 = 10;
      void method() {
         System.out.println(data);
         System.out.println(data2);
      }
   }

//continued...
```

Java Program Structure:

```
  public static void main(String args[]) {
      OuterClass oc = new OuterClass();
      InnerClass ic = oc.new InnerClass();
      System.out.println(oc.data);
      System.out.println(ic.data2);
      ic.method();
   }
}
```

Summary

- Object-Oriented Concepts
  - Object-Oriented Design
  - Class
  - Object
  - Attribute
  - Method
  - Constructor
  - Package
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism
  - Interface

Summary

- Java Program Structure
  - Declaring Java Classes
  - Declaring Attributes
  - Declaring Methods
  - Declaring a Constructor
  - Instantiating a Class
  - Accessing Object Members
  - Packages
  - The Access Modifiers
  - Inheritance
  - Overriding Methods
  - Abstract Classes and Methods
  - Interface
  - The *this* Keyword
  - The *super* Keyword
  - The *static* Keyword
  - The *final* Keyword
  - Inner Classes