COP 2250 – Java Programming I - Chapter 9 - Objects and Classes

# Objects as Models

Java is an object-oriented language. Java programs can model the real world, using objects in code to represent actual physical objects or intangible objects. A Java programmer working for AmTrack would create and work with boxcar objects. A Java programmer working for Coca Cola would use objects for bottles, cans, cases, and delivery trucks.

Objects:

1. Have state

* This means objects contain one or more items of data with assigned values.
* These items of data are called attributes, fields, instance variables, or properties.

1. Have behaviors

* This means that objects respond to events to perform operations.
* These operations are called instance methods.
* Methods often change the state of an object by changing the value of a property.

|  |  |
| --- | --- |
| Example: an FM Radio Object | FMRadio Class in Java Code |
| 1. Instance variables  * boolean isOn * double station * int volume  1. Instance methods  * setIsOn * setStation * setVolume | **public** **class** FMRadio {  **private** **boolean** isOn;  **private** **double** station;  **private** **int** volume;    **public** FMRadio() {  isOn = **false**;  station = 89.7;  volume = 6;  }    **public** **void** setOn(**boolean** isOn) {  **this**.isOn = isOn;  }  **public** **void** setStation(**double** station) {  **this**.station = station;  }  **public** **void** setVolume(**int** volume) {  **this**.volume = volume;  }  } |

Classes of the type discussed here commonly don’t have a main method, so they are not executable. **They are for data**, the Model in the MVC paradigm.

Class Declarations

* The instance variables and instance methods that define an object are defined by a class.
* A class is like a template that defines the properties and methods of objects of that class type.
* The class declaration only defines a template or blueprint for a class.
* An object of a class is called an instance of the class.

Analogy: think of the class declaration as a cookie cutter for creating cookie objects, and the cookies it creates as instances of the cookie class.

* A class defines the properties and methods for an object of that class.
* There can be only one public class in a file, and it must have the same name as the source code filename. The general form of a class:

public class SomeName {

instance variable declarations

constructor method declaration(s)

instance method declarations

}

* Figure 9.2 shows a class named Circle.
* Figure 9.3 shows how Circle might be coded in Java.
* Note there is no main method in this class.

Introduction to UML (Unified Modeling Language)

* Unified Modeling Language™ (UML) is an industry-standard language for specifying, visualizing, constructing, and documenting the parts of software systems. It helps in the design of Java classes. Liang has many in our book.
* This object-oriented system of notation has evolved from the work of Grady Booch, James Rumbaugh, Ivar Jacobson, and the Rational Software Corporation.
* Today, UML is managed by the Object Management Group (OMG) as the standard for modeling object oriented programs. It simplifies the complex process of OO software design, making a "blueprint" for construction.
* UML defines nine types of diagrams: class (package), object, use case, sequence, collaboration, statechart, activity, component, and deployment.
* UML class diagrams offer a concise way to describe the composition of a Java class.

**Employee**

-name : String

-salary : double

+Employee( nm : String )

+toString( ) : String

+getSalary( ) : double

+setSalary( sal : double )

Class name

public class Employee {

private String name;

attributes

(properties)

private double salary;

public Employee ( String nm ) { … }

operations

(methods)

public String toString( ) { … }

public double getSalary( ) { … }

public void setSalary( double sal ) { … }

}

* See Figure 9.4 on page 324 for the UML diagram for the Circle class described earlier.

Examine class SimpleCircle on page 326-327, then try TestSimpleCircle

* Java programmers don’t commonly put a main method in data class, although it is possible as SimpleCircle demonstrates. Instead, programmers typically code data classes and main classes in separate files.
* The next TV example illustrates this practice.

Examine class TV on page 327-328. Note the UML, too. Then try TestTV

Constructing Objects Using Constructors (See page 329)

* Constructors are special methods used to create objects of a class.
* One or more special methods called constructors (abbreviation ctors) are also normally included in the definition of a class to assign initial values to the instance variables.
* **Ctors** are **public** and have no return-type. Ctors must have the same name as the class.
* Ctors with parameters set the properties of an object when it is instantiated with new in a program.
* A no-argument ctor can be added so an instance can be created with default values.
* A default ctor is added automatically by Java if no ctor is defined.

Accessing Objects via Reference Variables (page 330)

* The name of a class is used to create a reference variable for an object of that class type.

Circle circ;

* The new operator can then create a new object of the class and assign it to the reference variable.

circ = new Circle(); // in this case, this is a no-args ctor

* The two steps above can be combined.

Circle circ = new Circle( ) ;

Accessing an Object’s Data and Methods

* The dot operator is appended to the reference variable to accomplish these tasks.
* You have seen this many times already in this course.

Exercise

Write a data class named **Rectangle.java** with double properties for the **length** and **width**. Add a no arg ctor that sets both properties to zero and a parameterized ctor. Include two methods to return the perimeter and area of a rectangle, respectively. Add a main method to Rectangle.java and use it to instantiate several rectangles and display their areas and perimeters. In a separate file, create a program named **TestMyRectangle.java** that instantiates several rectangles and displays their areas and perimeters.

#### Difference Between Variables of Primitive Types and Reference Types (page 332)

* The variable of primitive types such as ints, doubles and chars stores the actual value of the variable. Object types are stored differently.
* When an object is instantiated, the object variable doesn’t hold the object, it holds a reference to the object. A reference stores a value indicating the address of the instance.
* Any object variable can be assigned the value null,a Java reserved word.
* A null is used to initialize an object variable without creating an object.
* An instance method cannot be accessed from an object with a null value.
* More than one object variable can refer to the same object.

Introduction to the Date Class (page 334-335)

* The Date class is available in the **java.util** package.
* View the skeletal UML diagram for the Date class on page 334.

The Random Class

* View the UML diagram for this class on page 334.
* Note the point about **seeding** the ctor.
* The default no-args ctor uses the time as a seed, should generate differently every time.

### The Point2D Class

* This class in the Java API was developed to represent a point in a two-dimensional graph.
* The UML diagram on page 336 shows some of its methods.

Try TestPoint2D

### Static (aka class) Methods and Properties, Constants

Static Methods

* A class, or static, method is one that can be called without creating an instance of the class.
* A static method belongs to the class itself, not to any instance of the class.
* The Math class with methods like Math.sqrt( ) and Math.max() is an example.
* In fact, all of the methods of the Math class are static methods.
* The static modifier is required to create a static method.

**public static int getCalories ( ) {**

**return calories;**

**}**

* A static method cannot access instance variables of the class, only static variables (covered next) and its own parameters.
* A static method can take parameters and return a value.
* If a static method is declared public, the method can be called with the class name and the dot operator.
* The main( ) method (which is static) cannot invoke a non-static method.
* You cannot create an instance of the Math class, you can only execute its static methods such as:

**Math.sqrt() Math.round() Math.pow()**

* The main method in an executable class is a static method.
* A static method intended for the internal use of a class should be private.
* To make static methods accessible to other classes, declare them as public.

### Static (class) Variables

* Every new object of a class gets its own set of instance variables.
* Static variables belong to the class, but not to any one instance.
* Static variables are stored only once in a program, and are accessible to all instances.
* As with static methods, the static keyword is used to create static variables.
* A public static variable can be accessed by the **class name** and the dot operator.

System.out.print( Merchandise.numInStock );

* static variables are used for:
  + Variables accessible to all methods in all classes (if public).
  + Storing data for the use of static methods.
  + For constants. To do so, include the keyword final in the declaration:

public static final double SALES\_TAX\_RATE = 0.07;

By convention, constants are named in upper case with underscores as shown.

Examine CircleWithStaticMembers on page 338.  
Then try TestCircleWithStaticMembers

### Visibility (aka Access) Modifiers (page 342-344) Package Access

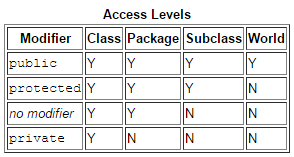
### When no visibility modifier is used for a property or method, it is accessible by any class in the same package.

The declaration of an instance variable, method, or ctor normally starts with an access modifier:

public - means can be accessed by all classes.

private - limits access to only the current class and its methods.

protected - limits access to the current class and subclasses of it (more later).



* Commonly, instance variables are declared as private and instance methods and constructors are declared as public. Declaring instance variables as private is a security measure termed information hiding, or data field encapsulation. This safeguards your object data from other classes.
* A method used to retrieve the value of an instance variable is called an accessor, or getter.
* A getter has the name pattern **getPropertyName()** and returns the desired property.
* A method that sets the value of a property is called a mutator, or setter.
* A setter always has the name pattern **setPropertyName(propertyType, propertyValue)**.
* A setter is always of return-type void and always needs a parameter for the assignment.

Examine CircleWithPrivateDataFields on page 345-346.  
Then try TestCircleWithPrivateDataFields

Example:

Needs parameter for the assignment

**public class SomeClass {**

**private int quantity;**

**public void setQuantity( int qty ) {quantity = qty;} //mutator**

**public int getQuantity() {return quantity;} //accessor**

Overloading Constructors

* Recall that a class may have multiple methods with the same name.
* This explains why a class can have two or more ctors.
* This overloading is legal as long as:
  + - The number of parameters is different.
    - The types of the parameters are different.

NOTE: The return type of a method is **not** sufficient for the compiler to distinguish between several methods with the same name.

### Objects as Method Parameters (page 347-351)

* Primitive arguments (ints, doubles, chars etc.) in methods are passed by value.
* This means that a method receives copies of the arguments for use within the method.
* If the method changes the value stored in the copy, this does NOT change the original value outside the method.
* This is not the case for methods that take objects as arguments. Object instances and arrays contain a reference (address of the object). They are NOT passed by value. They are passed by reference. The address is passed to the method.
* This means that when an object or array passed to a method is changed, this change will be evident after the method has finished executing.

Then try TestPassObject

Arrays of objects

* You can create arrays of objects in the same way as you create arrays of primitive types.
* All elements in the array must be of the same object type.

Then try TotalArea

### Immutable Objects

* These are objects that can be instantiated, but then not changed. The String class is immutable.
* To make your own immutable objects, make all properties private and don’t provide any setters.

The this Reference

* The this keyword, when used in a class definition, refers to the object itself.
* Note its usage on pages 356-357, especially when invoking a constructor.

### Choosing Method Names

* Start with a verb, since methods perform some action. For a method that computes a worker’s pay try:

public double **compute**( ) { … }

* If the verb doesn’t suffice, add a noun ( usually the target of the action )

public double **computePay**( ) { … }

* If the use of the method is still unclear, add adjectives for the noun

public double **computeWeeklyPay**( ) { … }