Object Orientation
Objects and Classes

References: Beginning Java Objects by Jacquie Barker;
Designing Object-Oriented Software by Rebecca Wirfs- Brock; Object-oriented Analysis & Design by Grady Booch;
Sara Stoecklin

Overview

• Object Oriented Programming defined
• Object orientation: Objects and Classes
• Defining classes in Java
• Composition: Objects as Attributes

Object-Oriented Programming

• “Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships.” --Grady Booch
• Main points in this definition:
  – Objects (not algorithms) are fundamental logical building blocks
  – each object is an instance of some class
  – classes are related via inheritance (“is-a” relationship)
• Programming without inheritance is doing object-based programming
Objects

• Objects are based on a “blueprint,” “prototype,” “model”

• Example “blueprints”:

- Floor plan
- Design for an auto

- Schematic of a VCR

Objects

• Object: “something material that may be perceived by the senses; something mental or physical toward which thought, feeling, or action is directed.” (Merriam Webster dictionary)

• Physical objects that may comprise a student registration system:

- professors
- students
- classrooms

Objects

• Conceptual objects that may be found in the system:

- courses
- departments
- degrees

• Software object: “a software construct that bundles together data (state) and functions (behavior) which, taken together, represent an abstraction of a ‘real-world’ (physical or conceptual) object.” -- Barker
Objects

- Objects retain certain information and know how to perform certain operations.
- The “bundling” of state and behavior of an object is known as **encapsulation**.
- An object has a public interface and a private representation. This is known as information hiding.
  - You can change the internal representation of an object or implement a new algorithm for a specific operation without changing the object's abstract, public interface.

Classes

- "A class is an abstraction describing the common features of all members in a group of similar objects.” — Barker
- Classes are models of real-world entities.
- Objects are examples (instances) of these classes.

You define the model (house plan) one time and use that model to create many instances (houses).

Object-Orientedness

- An object is an **instance** of a **class**
  
<table>
<thead>
<tr>
<th>Class</th>
<th>Instance (object)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>John Douglas ID #23434</td>
</tr>
</tbody>
</table>

  The Student is the class and uniquely identified John Douglas ID#23434 is an object or instance of that class.
Object-Orientation

• An object is an instance of a class

<table>
<thead>
<tr>
<th>Class</th>
<th>Instance, (object)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>John Douglas ID #23434</td>
</tr>
<tr>
<td>Television</td>
<td>Zenith serial #3568755</td>
</tr>
</tbody>
</table>

A Television is the class and uniquely identified Zenith with a serial #3568755 is an instance of that class.

Object-Orientation

• An object has
  state - characteristics (attributes)
  behavior - what an object does

<table>
<thead>
<tr>
<th>Object</th>
<th>State(s)</th>
<th>Behaviors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zenith</td>
<td>Channel, Size</td>
<td>Turn off, Increase Volume</td>
</tr>
</tbody>
</table>

A TV object Zenith #4353 has states of the size of the TV and the Channel displayed on the TV. It also has behaviors such as turn off the TV or increase the volume on Zenith #4353 TV.

Objects

• Data/State/Attributes associated with a student
  – student name
  – student ID and/or SSN
  – student birth date
  – student address
  – student major, if declared
  – student GPA
  – student’s advisor
  – student’s current course load
  – student’s transcript
Objects

• Data/State/Attributes associated with a course
  – course number
  – course name
  – course’s prerequisites
  – course credit hours
  – course instructors
• Attributes are data elements used to describe an object
• Collectively, attributes define object state

Objects

• Behavior/Operations/Methods associated with a student object may include:
  – enroll in a course
  – drop a course
  – choose a major
  – select a faculty advisor
  – reveal GPA when asked
  – reveal whether a particular course has been taken and all information about the course (when taken, who the instructor was, what grade earned)

Objects

• Behavior/Operations/Methods associated with a course:
  – permit student to register for
  – determine if student already in the course
  – reveal how many students registered thus far or how many seats remain
  – reveal course prerequisites
  – reveal course credit hours
  – reveal who is teaching the course
Objects

- Object behavior includes accessing its own attributes and modifying/maintaining its attributes
- Operations (methods) are also thought of as services that an object performs for its clients.

Defining New Classes in Java

- A Java class is a “blueprint” for an object.

Class “blueprint”

```java
class Student {
    int id;
    double gpa;
    String advisor;
    void printId ()
        Console.println(Id )
    // end printID
} // end
```

Memory Instance

For Class Student

```
<table>
<thead>
<tr>
<th>id</th>
<th>gpa</th>
<th>advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Defining New Classes in Java

```java
Student myStudent;  
```

```
<table>
<thead>
<tr>
<th>id</th>
<th>gpa</th>
<th>advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
myStudent = new Student();  
```

```
<table>
<thead>
<tr>
<th>id</th>
<th>gpa</th>
<th>advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
myStudent.printId();  
```

```
<table>
<thead>
<tr>
<th>id</th>
<th>gpa</th>
<th>advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
myStudent.id = 123;  
```

```
<table>
<thead>
<tr>
<th>id</th>
<th>gpa</th>
<th>advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
myStudent.gpa = 3.5;  
```

```
<table>
<thead>
<tr>
<th>id</th>
<th>gpa</th>
<th>advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>3.5</td>
<td></td>
</tr>
</tbody>
</table>
```

```
myStudent.advisor = "Smith";  
```

```
<table>
<thead>
<tr>
<th>id</th>
<th>gpa</th>
<th>advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>3.5</td>
<td>Smith</td>
</tr>
</tbody>
</table>
```

```
myStudent.printId();  
```

```
<table>
<thead>
<tr>
<th>id</th>
<th>gpa</th>
<th>advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>3.5</td>
<td>Smith</td>
</tr>
</tbody>
</table>
```
Defining New Classes in Java

```java
public class StudentTest {
    public static void main(String args[]) {
        Student S1;
        S1 = new Student();
        S1.id = 1234;
        S1.gpa = 3.5;
        S1.printId();
    }
}
```

```java
class Student {
    int id;
    double gpa;
    String advisor;
    void printId() {
        System.out.println(id);
    }
}
```

To use the class or "data type", you must first instantiate an object using `new`.

S1

id :    1234

gpa :  3.5

advisor :

printId ()
Defining New Classes in Java

```java
public class StudentTest {
    public static void main(String args[]) {
        Student S1;
        S1 = new Student();
        S1.id = 1234;
        S1.gpa = 3.5;
        S1.printId();
    }
}

class Student {
    int id;
    double gpa;
    String advisor;
    void printId() {
        System.out.println(id);
    }
}
```

S1
id : 1234
gpa : 3.5
advisor :
printId()

Thus the variable referenced as S1.id has the value 1234 in its memory location.

And the variable referenced as S1.gpa has the value 3.5 in its memory location.

Objects as Attributes

- One class can use another class object to define an attribute. This makes the class a 'composite' class.
- For example, we can define the ‘advisor’ attribute of the student to be of Professor type rather than simple String type.
- There is no limit to the number of levels to which objects can be “bundled” inside of one another.
Objects as Attributes

- Advantages to designing composite classes:
  - Professor attributes are encapsulated
  - Avoids data redundancy and potential loss of data integrity
    - If professor name changes, it only changes in the Professor class. If String had been used for advisor, all objects with that advisor would have to change the name as well as the Professor class.
  - Student object can request other services of the Professor object (e.g. where the professor’s office is located, what other classes the professor is teaching, etc.)

Themes in OO development

- Abstraction
  - Postpone implementation decisions. Creates fewer dependencies.
- Encapsulation
  - Information hiding
    - Keep external aspects (interface) separate from internal implementation details.
- Combining data and behavior (ADT)
  - Behavior is associated with the data. Behavior of a request depends on the type or class of the data.
Themes in OO development

- Emphasis on object structure, not procedure structure
  - Emphasis is on what an object is rather than how it is used. The use of an object changes more often than the behavior of an object. Hence, the system is more stable.

- Sharing
  - Inheritance allows sharing of data and behavior. Easier to build libraries that can be shared across projects.

- Synergy (cooperation)
  - There is synergy between OO features (Abstraction, encapsulation, classification, inheritance, polymorphism)