Why Object Oriented Programming

A Typical N-Tiered Architecture

From "Embrace and Extend with Java" (Information Week); 27 Aug 2001; http://www.informationweek.com/thisweek/story/IWK20010823S0007
Administrative Updates

- Email
  - Class announcements / discussion - cs332-02@cs.gmu.edu
  - Check email / use MEMO / change forwarding
    - MEMO Email Information [http://www.cs.gmu.edu/Email.html](http://www.cs.gmu.edu/Email.html)
  - Homework - [mailto:cs332@cs.gmu.edu](mailto:cs332@cs.gmu.edu)
- TA Office hours: Wednesdays, 2-4 PM, ST2 Room 365
- Honor policy
JVM

Java Virtual Machine (JVM) - The environment in which Java code runs: the program that executes Java code.

Java and C Execution Compared

![Diagram showing the execution comparison between Java and C]
Homework due today

What are some possible solutions?

Some things to avoid

- Stick to the requirements
- Don’t wait until too late to start
Sample Homework Solution

Me.java

class Me {

    public static void main(String[] args) {
        Me me = new Me("Jonathan Doughty",
                        "xxx-xx-xxxx",
                        "jdoughty@cs.gmu.edu");
        System.out.println(me);
        System.out.println(new java.util.Date());
        System.out.println(System.getProperty("java.version") + " " +
                            System.getProperty("os.name");
    }

    /** Instance variables for Me objects */
    String name;
    String gmuid;
    String email;

    /** Constructor */
    Me(String name, String id, String addr) {
        this.name = name;
        gmuid = id;
        email = addr;
    }

    /** Method to identify objects of type Me.  Java will call
    ** toString() whenever Me objects need to be converted to
    ** String objects.
    **/
    public String toString() {
        String result = "name:" + name + " GMUID:" + gmuid + " email:" + email;
    return result;
    }
}

Key aspects of Object Oriented Programming

- Abstraction - just the essentials, *ignoring all the unessential details*
- Composition - building from parts
- Separation - what vs. how
- Generalization - finding the common elements
Abstraction

- *Simplifying to its essentials the description of a real-world entity*
- Related to Encapsulation
- Properties of a Good Abstraction:
  - Well named
  - Coherent - attributes and behavior that are related and make sense (also called Cohesion)
  - Accurate - attributes and behavior match the entity being modeled
  - Minimal - nothing extraneous
  - Complete - all the attributes and behavior that are necessary
  - Consistency - Operations should be consistent with each other and with respect to names, arguments, return values, and behavior.
  - Decoupled - Different abstractions should not be inter-dependent
Classes

- A class is a blueprint, or prototype, that defines the variables and the methods common to all objects of a certain kind. - Java Tutorial
- In Java, everything is defined in some class
- Classes are either defined by the Application Program Interface (API) or are programmer defined abstract data types.
- Standard Java API classes are organized in packages:
  - java.lang
  - java.util
  - ...
- There will always be one file for each class
- At least to start, you will find it easiest if you maintain one .java source file for each class
Methods

• A method is a named sequence of instructions
  ○ an action you can request an object to take
  ○ a message you can send to an object
• Methods have a signature: a name and zero or more arguments
• Methods declare a data type (primitive or object or void) they return
• Guideline: Methods should generally be fairly short: roughly 10 - 20 lines of code.
• Guideline: Methods should be well named: the action they accomplish or the message they send.
• Constructors - a "method" to initialize objects
• Constructors and methods can be over-loaded
• Constructors and methods can have accessibility modifiers
Constructors

Constructors are the mechanism by which new instances of a class are created from the blueprint of the class definition.

- The purpose of a constructor is to initialize a new object.
- Constructors look something like method definitions except
  - They always have the same name as the class
  - They never return any type of value
- You "call" a constructor using the new operator and supplying any needed constructor arguments.
- Every Java class has, by default, a constructor:
  ```java
  public ClassName() {
  }
  ```
  that takes no arguments and does no special initialization.
- If you don’t define a constructor, Java will create a default, no-arg constructor.
- If you define any constructor, with or without arguments, Java assumes you know what you are doing and defines no default constructor.
Constructor Guidelines

- The purpose of a constructor is to put a newly created object into a known, initial state.
- Constructors should not do a lot of processing.
- Guideline: Separate object initialization from object behavior.
Objects

- An object is a software bundle of variables and related methods. - Java Tutorial
- Everything is either a primitive data type or an object reference
- Objects are things:
  - Models of real world, physical things, like Students, Presidents, Banks
  - Conceptual things, like Courses, Elections, FinancialTransactions
Java Variables

- In Java, declaring a variable to have a primitive data type reserves space for that primitive data type. E.g.,

  ```java
  int courseNumber;
  ```

- Declaring a variable to have an object type reserves space to hold a reference to an object of that type (or any derived type). E.g.,

  ```java
  CSStudent student;
  ```

- Declaring a variable to hold an object does not create the object
- The memory for an object instance is dynamically allocated using the `new` operator
Variables (Fields)

Local Variables

Variables you declare and use only within a method or a smaller

{ /* block of code */ }

Instance variables

Variables declared outside any method but still within a class

- Methods can use these instance methods without any other qualifiers or declarations.
- Instance variables retain their value across method calls, until all references to their object go out of scope
- Each object instance will have its own copy of these fields
- In a good object oriented Java class, just about all variables ought to be instance (or local) variables.
Objects as Data Types

A Java object should

- Be a bundle of closely related variables and methods
- Have the properties of a good abstraction
- Be focused on one concept: be coherent

Guideline: Define new object types liberally
Classes, Fields, Methods, Objects

Class: CSStudent

Fields:
name
labPartner

Methods:
getName()
assignPartner()
toString()

Fred,
Wilma,
Barney, null
Encapsulation

- Bundling together of **state** and **behavior**
- Objects have state: **attributes / data** that the objects maintains
- Objects have behavior: public **methods** that they expose for other objects to use - send "messages" to the object.
- Classes should expose to the outside (make public) **only** the methods needed to make the object do the things it is designed to do - provide the required behavior
- Exposed methods become a contract the object must keep
Information Hiding

- Private variables and methods of the object are implementation details unexposed to other objects.
- Implementation details, if hidden, can be changed at any time without affecting other objects.
- **None** of the internal details related to the implementation of the class should be visible to the outside.
- supports **modularity** - an object can be written and maintained independently of other objects.
Composition

- "An organization of components interacting to achieve a coherent, common behavior"
- *Composition* extends the responsibilities of an object by delegating work to additional objects.

*Composition* is the major mechanism for extending the **responsibilities** of an object. Nearly every object in an object model is composed of, knows of, or works with other objects.

"**Java Design**" Peter Coad and Mark Mayfield, Prentice Hall, 1996

- Create object **types** liberally
- The "has-a" relationship
CSStudent.java

/** The CSStudent class will enable objects to be created, each of which encapsulates information about an individual student. **
* @author Jonathan Doughty
**/
public class CSStudent {

    // Fields associated with an individual CSStudent objects.
    private String name;
    private CSStudent labPartner = null;

    // Constructor for CSStudent objects
    public CSStudent(String newStudentName) {
        name = newStudentName;
    }

    // Methods that can be called on individual CSStudent objects, i.e., messages that can be sent to individual CSCourse objects.
    public String getName() {
        return name;
    }

    /** Make the student remember who her lab partner is **/
    public void assignPartner(CSStudent partner) {
        labPartner = partner;
    }

    public String toString() {
        String result = "Student name: " + name;
        if (labPartner != null)
result += " Lab partner: " + labPartner.getName();
return result;
}
/** A Java class that models information kept about a course and students in the course.
**
** @author Jonathan Doughty
**/ public class CSCourse {

private Integer courseNum;

/** Dynamic collection of references to student objects enrolled in the class */
private java.util.ArrayList students = new java.util.ArrayList();

/** Creates a new CSCourse object with the indicated course number. */
public CSCourse(int number) {
    courseNum = new Integer(number);
}

/** Enroll the student whose name is provided into this course. */
public void enroll(String name) {
    CSStudent current = new CSStudent(name);
    students.add(current);
}

/** Assign pairs of students as lab partners for this course. */
public void assignLabPartners() {

    // Assign every other pair of students as lab partners
    int next = 0;
    int pairs = students.size() / 2;

    while (pairs > 0) {

CSStudent student = (CSStudent) students.get(next);
CSStudent partner = (CSStudent) students.get(next+1);

student.assignPartner(partner);
partner.assignPartner(student);
next += 2;
pairs--;
}
}

/** List the roster of students enrolled in this course to ** the PrintStream provided. **/ public void listRoster(java.io.PrintStream out) {
  for (int i = students.size() - 1; i >= 0; i--) {
    // Print out information about each student. In this case,
    // simply request each student object reference to identify
    // itself.
    out.println(students.get(i).toString());
  }
}

public String toString() {
  StringBuffer buf = new StringBuffer();
  String endl = System.getProperty("line.separator");
  buf.append("Course number: ");
  buf.append(courseNum);
  for (int i = students.size() - 1; i >= 0; i--) {
    buf.append(endl);
    buf.append(students.get(i));
  }
  return buf.toString();
}
Putting it together - 3

CSTest.java

/** A Java class to test CSCourse objects 
 **
 ** @author Jonathan Doughty
 ** @version 1.0
 ***/

public class CSTest {

/** The starting point for this application 
 ***/

public static void main(String[] args) { } 

}
Homework 2

Reading

- Chapter 3 - Objects and Classes

Programming

In this week’s assignment you’ll be creating several Java classes to model a very simple airline.

This assignment has two goals: 1) is to get objects defined in separate classes to work together. 2) To create several classes each focused on a single abstraction.

- Create a class named **Airplane**. Airplane objects should have the following characteristics:
  - An identifying number (ID). Use a String object for the airplane ID.
  - A total number of seats.
  - A number of currently available seats.
  - A cost per seat. On these airplanes all seats will cost the same. **You decide what data type to use for the seat cost field.** See below for some considerations.
  - A method or methods to change the cost per seat.
  - An implementation of the `public String toString()` method that will return a String containing the airplane ID and the current cost per seat.

- Create a class named **Airline**. Airline objects should have the following characteristics:
  - A name
  - A collection of Airplane objects
  - A method named `addAirplane` that will cause a new Airplane to be added to the collection of airplanes.
  - An implementation of the `public String toString()` method
that will return a String containing the airline name and other information identifying the current state of the airline.

- Create also a test class that will be used to:
  - Create a new Airline with some name
  - Request the Airline to add several new Airplanes; at least five but no more than ten. You can use whatever id number and initial cost per seat information you would like
  - Cause at least one of the Airplane objects to change its cost per seat value
  - Cause the Airline object and the Airplane objects to identify themselves and, in the case of the Airplanes, their current cost per seat.

**Things to consider**

- Whose responsibility should it be to create Airplane objects?
- Whose responsibility should it be to assign Airplane identifiers?
- Whose responsibility should it be to manipulate the Airplane cost per seat?
- Should the data type used for the cost per seat be a primitive or an object type? What are the potential advantages and disadvantages of the choices?

Submit before the start of next week’s lecture a [jar archive](mailto:cs332@cs.gmu.edu) with your source code to [cs332@cs.gmu.edu](mailto:cs332@cs.gmu.edu). Name the jar archive [your_gmu_id].jar. More guidelines to follow on creating jar archives.
Generalization

- Identifying, and possibly organizing, common elements among different entities
- Hierarchy - inheritance
- Polymorphism
- Patterns
Homework 2

Reading

- Chapter 3 - Objects and Classes

Programming

In this week’s assignment you’ll be creating several Java classes to model a very simple airline:

- Create a class named **Airplane**. Airplane objects should have the following characteristics:
  
  - An identifying number. Use a String object for the airplane number.
  
  - A total number of seats.
  
  - A number of currently available seats.
  
  - A cost per seat. On these airplanes all seats will cost the same. **You decide what data type to use for the seat cost field.** See below for some considerations.
  
  - A method or methods to change the cost per seat.
  
  - An implementation of the **public String toString()** method that will return a String containing the airplane ID and the current cost per seat.

- Create a class named **Airline**. Airline objects should have the following characteristics:
○ A name

○ A collection of Airplane objects

○ A method named **createAirplane** that will cause a new Airplane to be created and added to the collection of airplanes.

- Create a class named TestAirline that will be used to:
  
  ○ Create a new Airline with some name
  
  ○ Request the Airline to add several new Airplanes; at least five. You can use whatever id number and initial cost per seat information you would like.

- Some considerations when creating your Airplane and Airline classes:
  
  ○ Whose responsibility should it be to assign Airplane identifiers?
  
  ○ Whose responsibility should it be to manipulate the Airplane cost per seat?
  
  ○ Should the data type used for the cost per seat be a primitive or an object type? What are the potential advantages and disadvantages of the choices?

- Submit before the start of next week’s lecture a **jar archive** with you source code to mailto:cs332@cs.gmu.edu.