Highlights of Previous Lecture

- Moving common capabilities to abstract bases classes
- Using abstract methods to support base class behavior in derived classes
- How design choices can lead to brittle behavior and thwart polymorphism
- Type safe enumerations
  - Why are they a good things?
  - How do they support polymorphism?
Final Project

The class final project will be to develop an airline reservation system for a small airline. Requests to book reservations on flights between various destinations will be made by users.

- You will be provided with information on starting and destination airports and departure times.
- You will be provided with a sequence of reservation requests.
Finding classes

- From the specification, what are the likely candidates for objects used in the system.
- Consider data / events that must be recorded and remembered
- Think of organizational / administrative units that need to be modeled
- Consider other systems that communicate or interact with the system being developed
Naming Objects and Methods

"Choosing good names for things is an important part of the analysis process. In an object-oriented design, classes, objects, attributes, methods, and source code files are among the things that need names. The names you select should reflect the semantics of the item. Just as a good writer uses the English language carefully and effectively, a good programmer will carefully and effectively choose names."

- Classes should be named with common noun phrases, such as Color or Sensor.
- Objects should be named to indicate they are specific instances with identity, such as theDoorSensor, foregroundColor, or listOfSensors.
- Methods that modify the state of an object, or cause it to do something should be named with active verb phrases, such as Draw or setColor.
- Methods that return state information should indicate a result or use a form of the verb “to be,” such as getColor or isClosed.

- Wampler - "The Essence of Object-Oriented Programming"

Naming Conventions are also discussed in the Java Language Specification and in the Code Conventions for the Java Programming Language document.
Final project - Goals

- Set up collections of flights
- Maintain information about seat availability on flights
- Respond to reservation requests
- "Sell" as many seats as possible on scheduled flights
- Create solutions that demonstrate Object Oriented programming skills
Final project - some possible static / data aspects

- Reservations
- Airports
- Itineraries
- DepartureTimes
- Airplanes
- Seats
- SeatCategories
- Tickets
- Prices
Finding class responsibilities / collaborators

Imagine yourself as one of the objects in the system. Ask yourself questions:

- Who do I interact with?
- How do I respond to a message from some other object?
- What is my job?
- What do I do?
- What do I contribute to the system? What do I need to remember?
Final Project - Overview

Requests to book reservations on flights between various destinations will be made by potential passengers.

- The final project will consist of a number of Java classes and interfaces.
- You will create some of these classes and I will create others.
- I will provide you with working code with which you will integrate your implementations.
  - I will provide a jar file(s) containing class files and data
  - I won’t necessarily provide source code
  - For my final evaluation of your code I may use different classes and data
- Your code will contain the `main` method to control reservation system operations
- You will be required to call some of my code to get inputs:
  - Flight Itinerary and departure times
  - Reservation requests
- The means by which your code and mine will communicate is through implementations of Java `interfaces`.
High Level Final Project Sequence Diagram

Your code:

• request Flight initialization
• initialize Flight information
  • request Reservations

My code:

• Reservation requests
• request Flight information
  • Flight information
Initial Final Project Interfaces

Consider these three to start:

reservations/Flight.java

```java
package reservations;

import java.util.Date;
import java.util.Iterator;
import reservations.Itinerary;

public interface Flight {
    public Date getLeavingTime();
    public Itinerary getItinerary();
    public Iterator getReservationIterator();
}
```

You will be implementing the Flight interface, among others

reservations/Itinerary.java

```java
package reservations;

import java.util.Date;
import reservations.Airport;
import reservations.Flight;

public interface Itinerary {
    public Airport getStartingAirport();
    public Airport getDestinationAirport();
}
```
I will implement the Itinerary interface, as well as the Airport interface:

reservations/Airport.java

```java
package reservations;

public interface Airport {
  public String getCode();
  public String getCity();
  public String getState();
}
```
Homework

Reading

- Chapter 11 - Modeling the Dynamic/Behavioral Aspects of the System

Homework

The goals of this homework are

- To get practice implementing an interface
- To start thinking about the final project
- To work through the details of working with the jar archives I will provide in your development environment

Write a class that implements the `Flight` interface

- Your class should include a `public` constructor that takes a single argument

```java
public ClassName( reservations.Itinerary i)
```

- To to insure that your classes don’t have access to aspects of mine that they should not, and accommodate differences in development environments, all of the interfaces and my code will reside in the `reservations` class.
- You don’t have to be concerned with the details of the getReservationIterator method as yet.
- I will provide you with a jar file that contains code and data to generate sample Itinerary objects and documentation on how to use it.
- Your classes can reside in another package or in the default, unnamed package. **Do not place your classes in the reservations package.**
This will be due next 19 November.