Overview

- Homework 3
- Project/Paper
- Object Oriented Design
Goals

- Learn Object-Oriented Design Methodologies
Assignments

- Brookshear: Ch 5.5, Ch 6.3/6.4, Ch 7 (especially 7.7) (Read)
- Read linked documents on these slides (slides will be posted in courseworks)
Objectives:

- Review the main OOP concepts:
  - inheritance
  - abstraction
  - encapsulation
  - polymorphism

- Get an appreciation for the complexity of object-oriented design.
What are OOP’s claims to fame?

- Better suited for team development
- Facilitates utilizing and creating reusable software components
- Easier GUI programming
- Easier program maintenance
OOP in a Nutshell:

- A program models a world of interacting objects.

- Objects create other objects and “send messages” to each other (in Java, call each other’s methods).

- Each object belongs to a class; a class defines properties of its objects. The data type of an object is its class.

- Programmers write classes (and reuse existing classes).
Main OOP Concepts:

- Inheritance
- Abstraction
- Encapsulation
- Polymorphism
- Event-driven computations
Inheritance

- A class can **extend** another class, inheriting all its data members and methods while redefining some of them and/or adding its own.

- A class can **implement** an interface, implementing all the specified methods.

- Inheritance implements the “is a” relationship between objects.
Inheritance (cont’d)

subclass or derived class extends or superclass
subinterface extends superinterface
class implements interface
Inheritance (cont’d)

- In Java, a subclass can extend only one superclass.
- In Java, a subinterface can extend one superinterface
- In Java, a class can implement several interfaces — this is Java’s form of *multiple inheritance.*
Inheritance (cont’d)

■ An abstract class can have code for some of its methods; other methods are declared abstract and left with no code.

■ An interface only lists methods but does not have any code.

■ A concrete class may extend an abstract class and/or implement one or several interfaces, supplying the code for all the methods.
Inheritance (cont’d)

- Inheritance plays a dual role:
  - A subclass reuses the code from the superclass.
  - A subclass (or a class that implements an interface) inherits the data type of the superclass (or the interface) as its own secondary type.
Inheritance (cont’d)

- Inheritance leads to a hierarchy of classes and/or interfaces in an application:

```
Game

Solitaire  GameFor2

BoardGame

Chess  Backgammon
```
Inheritance (cont’d)

- An object of a class at the bottom of a hierarchy inherits all the methods of all the classes above.

- It also inherits the data types of all the classes and interfaces above.

- Inheritance is also used to extend hierarchies of library classes, reusing the library code and inheriting library data types.
Inheritance (cont’d)

- Inheritance implements the “is a” relationship.
- Not to be confused with embedding (an object has another object as a part), which represents the “has a” relationship:

  A sailboat is a boat

  [Images of different types of boats]

  A sailboat has a sail
Quiz

- True or False? Inheritance is helpful for the following:

  - □ Team development __________
  - □ Reusable software __________
  - □ GUI programming __________
  - □ Easier program maintenance __________
Answer

True or False? Inheritance is helpful for the following:

☐ Team development
☑ Reusable software
☑ GUI programming
☐ Easier program maintenance
Abstraction

- Abstraction means ignoring irrelevant features, properties, or functions and emphasizing the relevant ones...

  “Relevant” to what?

- ... relevant to the given project (with an eye to future reuse in similar projects).
Abstraction (cont’d)

- Example from javax.swing:
  public abstract class AbstractButton

Fields:
  protected ButtonModel model
  etc.

Methods:
  void addActionListener (ActionListener l);
  String getActionCommand();
  String getText();
  etc.

The data model that determines the button’s state

Apply to any button: “regular” button, a checkbox, a toggle button, etc.
Abstraction (cont’d)

java.lang.Object
  |
  +--java.awt.Component
  |
  +--java.awt.Container
  |
  +--javax.swing.JComponent
  |
  +--javax.swing.AbstractButton

Extends features of other abstract and concrete classes
Encapsulation

- Encapsulation means that all data members (*fields*) of a class are declared *private*. Some methods may be private, too.

- The class interacts with other classes (called the *clients* of this class) only through the class’s constructors and public methods.

- Constructors and public methods of a class serve as the *interface* to class’s clients.
Encapsulation (cont’d)

- Ensures that structural changes remain local:
  - Usually, the structure of a class (as defined by its fields) changes more often than the class’s constructors and methods.
  - Encapsulation ensures that when fields change, no changes are needed in other classes (a principle known as “locality”).
Quiz

- True or False? Abstraction and encapsulation are helpful for the following:
  - Team development _________
  - Reusable software _________
  - GUI programming _________
  - Easier program maintenance _________
Answer

- True or False? Abstraction and encapsulation are helpful for the following:

☑ Team development
☑ Reusable software
☐ GUI programming
☑ Easier program maintenance
Polymorphism

- We often want to refer to an object by its primary, most specific, data type.
- This is necessary when we call methods specific to this particular type of object:

```java
ComputerPlayer player1 = new ComputerPlayer();
HumanPlayer player2 = new HumanPlayer("Nancy", 8);
...
if (player2.getAge() < 10)
    player1.setStrategy(new Level1Strategy());
```
Polymorphism (cont’d)

- But sometimes we want to refer to an object by its inherited, more generic type:

```java
Player players[] = new Player[2];
players[0] = new ComputerPlayer();
players[1] = new HumanPlayer("Nancy", 8);

game.addPlayer(players[0]);
game.addPlayer(players[1]);
```

Both ComputerPlayer and HumanPlayer implement Player
Polymorphism (cont’d)

- Why disguise an object as a more generic type?
  - To mix different related types in the same collection
  - To pass it to a method that expects a parameter of a more generic type
  - To declare a more generic field (especially in an abstract class) which will be initialized and “specialized” later.
Polymorphism (cont’d)

- Polymorphism ensures that the appropriate method is called for an object of a specific type when the object is disguised as a more generic type:

```java
while ( game.notDone() )
{
    players[k].makeMove();
    k = (k + 1) % numPlayers;
}
```

The appropriate `makeMove` method is called for all players (e.g., for a `HumanPlayer` and a `ComputerPlayer`).
Polymorphism (cont’d)

- Good news: polymorphism is already supported in Java — all you have to do is use it properly.

- Polymorphism is implemented using a technique called \textit{late} (or \textit{dynamic}) \textit{method binding}: which exact method to call is determined at run time.
OO Software Design

- Designing a good OOP application is a daunting task.
- It is largely an art: there are no precise rules for identifying classes, objects, and methods.
- Many considerations determine which classes should be defined and their responsibilities.
- A bad design can nullify all the potential OOP benefits.
OO Design (cont’d)

- A few considerations that determine which classes are defined and their responsibilities:
  - Manageable size
  - Clear limited functionality
  - Potential reuse
  - Support for multiple objects
  - The need to derive from a library class
  - The need to make a listener or to implement a particular interface
  - The need to collect a few data elements in one entity
Review:

- Name the main software development concerns that are believed to be addressed by OOP.
- Explain the dual role of inheritance.
- Can an interface extend another interface? If so, what does it mean?
- Can an interface extend a class? If so, what does it mean?
- Why do you think Java does not allow a class to extend several classes?
Review (cont’d):

- What is abstraction?
- Explain how encapsulation helps in software maintenance.
- Why sometimes objects end up disguised as objects of more generic types?
- What is polymorphism?