CSCI0320
Introduction to Software Engineering

Lecture 2
Interface & Class Design

Course Reminders

- SOLAR design is this week
- FINAL PROJECT
  - Teams should be determined by Thursday
    - Talk after class today as needed

Interface and Class Design

- Principles to consider
  - Simplicity
    - Ease of coding
    - Ease of use
  - Abstraction
    - Hide as much as possible
  - Correctness
    - Make sure it can/will work
- Design methodology
  - Considering alternatives
  - Evaluating possibilities
  - Exploring the solution space
- Work in terms of interfaces

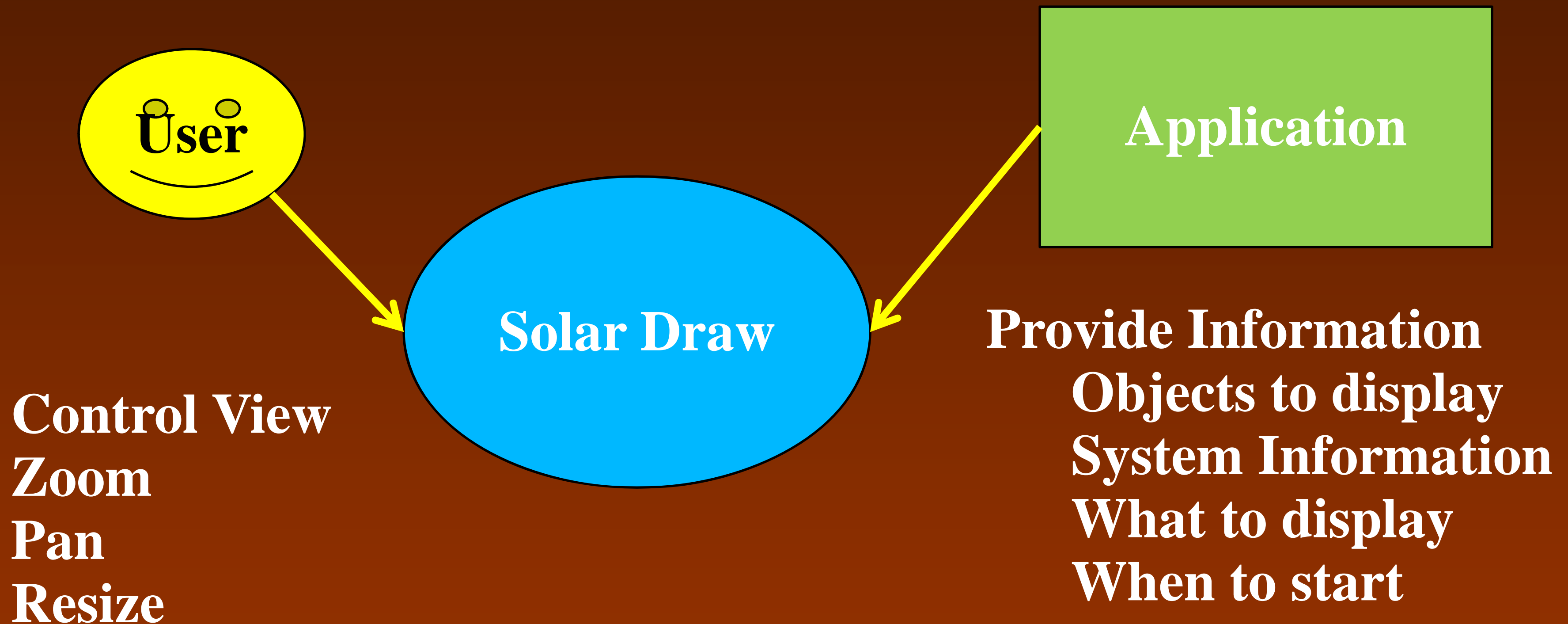
SolarDraw Library

- Demo of Galaxy running
- Objectives (Responsibilities)
  - Display what is happening
    - As the program computes
  - Minimally intrusive
  - Display information about the computation

What are the Classes

- What information do we need
  - To do the display
  - From the user
    - Interface only, not implementation
- Objects
  - Mass, position, velocity, size, name
  - Do you need all this information for display?
- System information
  - Current time
- Display
  - Control of the graphics

How are the Classes Used



Asynchronous Display

SolarDraw.java

- General
  - Header; SVN/CVS support; Package name
  - Imports; Javadoc class comment
  - Public interface; Note comments at end of file
- Display Object
  - What is the simplest interface
  - How to unregister objects
  - When to start the display
- Planetoid Object (Object)
  - Need to get information from the application
  - Alternatives. Is asynchrony safe? Units?
  - Note callbacks. Note that Object is a bad name
- Control Object
  - Same alternatives
  - Should there be feedback to the program
- Factory Object
  - Alternatives for creation
  - Different kinds of factories

Review

- Design first in terms of interfaces
- Keep the interfaces abstract
- Keep the interfaces simple
- Think about different approaches
  - There is no “right” or “wrong” answer
  - Some are better than others
    - Simpler, more abstract, easier to use
    - More flexible in handling evolution

Tic Tac Toe

You are to write a program that will play a standard game of tic-tac-toe. The input and output from the program will be textual. The user will be given the choice of going first or second. The program should number the squares and, when it is the user's turn, allow the user to enter the number of the square of his/her move or to resign or claim victory. After inputting the user's move or determining that the computer should go first, the computer should find its move, print out a version on the board, and then prompt the user for input. The program should be able to play at a variety of levels, some of which allow the user to win, some of which do not. The program should keep track of the number of games won or lost in a given session.

Finding Classes

- Candidate classes
  - Everything you can think of
- Sources
  - Objects in the problem
  - Objects in the potential solution
  - Algorithms and data structures
  - Interfaces to the user, other systems, etc.
- Suggestions

Proposed Classes

Finding Classes: English

- You are to write a program that will play a standard game of tic-tac-toe. The input and output from the program will be textual. The user will be given the choice of going first or second. The program should number the squares and, when it is the user's turn, allow the user to enter the number of the square of his/her move or to resign or claim victory. After inputting the user's move or determining that the computer should go first, the computer should find its move, print out a version on the board, and then prompt the user for input. The program should be able to play at a variety of levels, some of which allow the user to win, some of which do not. The program should keep track of the number of games won or lost in a given session.

Select Classes

- What is the right set of classes
  - What classes imply others
  - What classes contain others
- Want a set of 5-10 classes at the top level
  - Too many creates problems
  - Too few doesn't help the design
- Annotate each class
  - Responsibilities (potential methods)
  - Requirements (potential needs)

Simple Classes

- **Who**: X, O, or Empty
  - Contents of a square
  - Indication of whose turn it is
- **Move**
  - Legal move (by user or computer)
  - Squares 1 … 9
  - PASS, WIN, LOSE, TIE
  - NEW GAME, QUIT
- **Game State**
  - INITIAL, PLAYING
  - DRAW, X WINS, O WINS

Classes

- Board
  - Contains the current board
  - Methods to make a move
  - Methods to access a square
  - Common tic-tac-toe logic
    - Check for a win or draw
    - Check for a winning move
- Player
  - User or computer player
    - User: needs a user interface
    - Computer: needs a way of finding move
  - Find the next move

Classes

- **AI**
  - Represents the different levels
  - Algorithm for finding a move
  - Several implementations
    - Easy, Medium, Hard
- **Score**
  - Class for recording the score
- **Game**
  - Main program
  - Controls the game

What Next

- Start with the main program
  - Implement this
  - Add classes as needed
  - This works for small systems
- Start with interfaces
  - Create interfaces for principle classes
    - Denoting the methods they provide others
  - Purpose
    - Better define the class
    - Provide the tools needed for implementation
    - Abstraction: hide the details

How to Do Interfaces

- C/C++
  - Define the .H file(s)
    - Typically start with <proj_local.H>
    - Provide constant definitions
    - Define enumerations
    - Define abstract classes
  - Single .H file for a package
    - Local .H files provide implementation details
- Java
  - Define an interface class
    - Generally have a common interface for package

TicTac Interfaces

- TicTac.java
  - Smart enumerations
  - Interfaces for other components

Moving to Implementations

- Start with the interfaces
- Then work on the classes
  - Start with the main program
    - This gives you a working system from start
    - Lets you correct the interfaces as you go
  - Provide stub implementations of interfaces
    - To allow early testing and debugging
- Alternatives
  - Start with the "riskiest" class
  - Start with unit tests for each interface

Class Implementation

- TicTacGame.java
- TicTacComputer.java
- TicTacBoard.java
- TicTacUser.java
- TicTacMedium.java
- TicTacHard.java

Review

- Keep the Code Simple
  - Well documented
  - Use the interfaces provided
    - Hide information as much as possible
- Continually rethink the design
  - Especially at the early stages
  - Can you simplify the interfaces
    - Reduce the number of methods
    - Reduce the number of parameters
    - Make them easier to use
  - Can you hide any more details

Review

- Coding style is important
  - Spaces
  - Comments and documentation
  - Make the code readable
    - Internal spacing and indentation
    - Variable names
    - Internal comments where needed
  - Consistent organization
    - Ordering of elements
    - Naming conventions

Good Design

- Emphasize
  - Simplicity of the design
  - Simplicity for the user of the code
  - Abstraction (Interfaces) (information hiding)
  - Safety
  - Functionality
  - Consistency
- Lots of little decisions
  - That affect each other
  - That have to be looked at globally
  - Weltanschauung
- Lots of tradeoffs