CSPP 511-01:
Introduction to Object-Oriented Programming

Harri Hakula

Ryerson 256, tel. 773-702-8584
hhakula@cs.uchicago.edu

August 14, 2000
Outline

- Project Status
- PPPad – The Implementation
- InputManager
- Calculator
Project Status

- Design documents have been reviewed.
- Alpha previews: Thursday-Sunday. Timetable to be announced on Wednesday.
- Model implementation is at version 0.1a.

The best way to use the model implementation is to copy ideas, not lines of code.

Why? 1) It is a moving target, 2) There are bugs . . .

Do not waste time trying to understand every nuance when you could easily brew your own.
PPPad – The Implementation

Statistics: (Version 0.1a)

• 1380 lines including comments.
• 19 classes and interfaces.
• 835 excluding SavitchIn.
• 47 lines per class on the average, excluding SavitchIn.
PPPad – The Implementation Cntd.

Missing features: Memo handling, scrolling.

Pros: Clean separation of functionality (Calculator does its own analysis, though.)

Cons:

- list implementation,
- managers should be derived from base class (menus),
- scrolling has not been planned for,
- error handling is lacking.
What is the problem that InputManager solves? (Or tries to . . . )

Ultimately we want to read from keyboard with SavitchIn. However, input has different meaning depending on context: commands and text. In our program, we always know the context. Thus, if we expect a command, we can let InputManager to verify that the next string is indeed a valid command.
InputManager Cntd.

Q: Since we always know the context, why cannot I just use SavitchIn and process the input at the point where I use it?
A: You can, but then you have to duplicate your code over all managers (or command handling classes).

Q: Should I then analyze every input token within the InputManager?
A: Yes and no. The calculator has its own, specific commands (arithmetic operations), perhaps they should be handled locally and not within the InputManager. On the other hand, every point where input is being processed makes debugging more difficult.
Calculator

Design goal: Easiest way to convert my old integer-based implementation into one that can be used in my PPPad.

The key issue: Where is input processed? At the same time I want to interpret the standard commands such as H and Q, and the arithmetic commands.

Two options: Either modify the InputManager or keep the analysis inside the Calculator.

I decided to go for the latter.
Problem: How can I detect a standard command if all processing is local?
The solution is simple: If InputManager.nextCommand() does not recognize the input string, that string is put back to the list! This means that at every point where InputManager.nextCommand() fails, the top string must be removed separately.

Even in my very small project, this lead to three modifications in different classes.
Double Trouble

The calculator needs a stack of doubles. We have very nice generic stacks, can we really use them? I cheat by wrapping the doubles and just using my utterly horrible List.

public class DoubleStack {
    public DoubleStack() {
        store = new List();
    }
    public void push(double value) {
        store.insert(new Double(value));
    }
    public double pop() {
        Double value = (Double)store.pop();
        return value.doubleValue();
    }
    private List store;
}
Questions?