Object Oriented Programming and Design in Java

Session 3
Instructor: Bert Huang
Announcements

• Next Monday's class canceled for Distinguished Lecture: Feb 1, 11 AM Davis Auditorium.

• Course survey due

• Homework 1 will be posted soon, "officially" out Feb. 3rd
Review

- basic syntax, javadoc, primitive types, references, importing packages, exceptions, input, Arrays, ArrayLists, declaration keywords, code style
- CUNIX and Eclipse demo
Today’s Plan

- Turning ideas into a program
- Use cases
- Identifying classes
- UML diagrams: class diagram, sequence diagram, state diagram
- Example: todo list manager
Ideas to Programs

Analysis (common sense)

Design (object-oriented)

Implementation (actual programming)
Phase 1: Analysis

- Ideas or description of final product may be inadequate
- Specifically describe requirements to be considered a completed program
- Decide on exact functionality
- Limit ambition, but don’t think too much about design and implementation
Use Cases

• Use cases specifically describe the operation of the program

• Narrows down exactly what you want your program to do

• Useful as test cases

• Implementation and design don’t matter
Phase 2: Design

- More explicit about object interactions
- Define classes of objects
- Decide responsibilities of classes
- Define attributes and methods of classes
Identifying Classes

• Good first step: look for **tangible nouns** in use cases. Then...

• **Agents** - objects that perform tasks

• **Events** - store information about events

• **Systems, interfaces** - run the program, talk to user or other programs

• **Foundational classes** - String, Date, etc.
Identifying Responsibilities

• Good first step: look for verbs, actions in use cases

• These actions may directly describe responsibilities, or

• may depend on other responsibilities
CRC “Cards”

- Class - Responsibility - Collaborators
- Brainstorming tool for setting up classes and responsibilities
- Collaborators loosely define class relationships; we get more precise later

<table>
<thead>
<tr>
<th>ClassName</th>
</tr>
</thead>
<tbody>
<tr>
<td>responsibility 1</td>
</tr>
<tr>
<td>responsibility 2</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Walkthroughs with CRC

- Play out (partial) use cases using CRC
- Who does what during the use case?
- Do some objects have too much responsibility?
  - Create helper objects or agents
- Are some classes never used?
Universal Modeling Language

• Standard formatting rules and syntax for modeling software

• More precise than CRC, but still looser than javadoc or actual code skeleton

• Start to name methods based on established responsibilities
UML Class Diagrams

- Each class is a rectangle

- Connect classes by their relationship

![Diagram showing class relationships]

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Attributes : Type</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Class Relationships

- **Dependency** - any time one class needs the other
- **Aggregation** - one class contains elements of the other class
- **Association** - other relationship
- **Inheritance**
- **Interface Implementation**
Sequence Diagrams

- Draw objects as they interact over time
- UML: underline to indicate instances
- Each object has dotted life-line
- **Activation bars** indicate object running
- Arrows indicate method calls

```plaintext
ObjectName : Class

/*activation bar*/
ObjectName : Class

doSomething()

/*activation bar*/
other : Class
```
State Diagrams

- Useful for visualizing how an object changes over time
- Rounded rectangles represent states
- Arrows and text describe triggers for state changes
Checkpoint

- You should have a tractable design
- Manageable class complexity
- Clean encapsulation
- You can write the code skeleton and javadoc now
- Then Phase 3: Implement
Example: Console Todo List Manager

- Most programs start with a vague idea:
- Hey, <your name>, make me a program that like helps keep track of stuff I have to do. Or whatever. And it should sort by due date.
Use Case 1

- User starts the program
- Display saved items numbered and sorted by due date.
- User enters “add laundry” at prompt
- User is prompted for a due date
- User enters date
- list updated and displayed with laundry in its correct sorted position
Use Case 2

- User has previously entered todo items, including “laundry”
- User starts program
- To do list is displayed
- User enters “finished laundry”
- Laundry is removed from the list, remaining items displayed
Classes and Responsibilities

- Nouns: date, item, prompt, list
- Verbs: display, enters, add, delete, update, sort
- Agents: file manager (saving + loading)
- Our classes: TodoItem, TodoPrompt, TodoList, TodoFileManager
CRC

<table>
<thead>
<tr>
<th>TodoItem</th>
<th>TodoList</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store name, date</td>
<td>TodoList</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TodoPrompt</th>
<th>TodoList</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display list</td>
<td>TodoList</td>
</tr>
<tr>
<td>get commands</td>
<td>TodoList</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TodoList</th>
<th>TodoItem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store list of items</td>
<td>TodoItem</td>
</tr>
<tr>
<td>Sort items</td>
<td>TodoPrompt</td>
</tr>
<tr>
<td>Add and remove</td>
<td>TodoFileManager</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TodoFileManager</th>
<th>TodoList</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load list from file</td>
<td>TodoList</td>
</tr>
<tr>
<td>Save list to file</td>
<td>TodoList</td>
</tr>
</tbody>
</table>
## CRC Walkthrough

<table>
<thead>
<tr>
<th>TodoItem</th>
<th>TodoPrompt</th>
<th>TodoFileManager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store name, date</td>
<td>Display list</td>
<td>Load list from file</td>
</tr>
<tr>
<td></td>
<td>get commands</td>
<td>Saved list to file</td>
</tr>
<tr>
<td></td>
<td>add/delete TodoItem</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TodoList</td>
<td>TodoList</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TodoList</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Store list of items</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sort items</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add and remove</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- User starts the program
- Display saved items numbered and sorted by due date.
- User enters “add laundry” at prompt
- User is prompted for a due date
- User enters date
- list updated and displayed with laundry in its correct sorted position
Class Diagram

```
TodoFileManager
- savedFile : File
- loadFile()
- saveFile()

TodoItem
- name : String
- date : Date
- getName()
- getDate()

TodoPrompt
- executeCommand(String s)
- displayList()

TodoList
- list<TodoItem> : ArrayList
- sort()
- addItem()
- deleteItem()
- getItem(int index)
```
Sequence Diagram 1

: Todo
myList : TodoList
manager : TodoFileManager
prompt : TodoPrompt

create
create
add
start
get
add "laundry"
get
return name

load from disk
Display list to user
User adds laundry
Display list to user
Violet

- I used Horstmann’s Violet to draw the UML diagrams on last few slides
- http://horstmann.com/violet
Reading

- Horstmann Ch. 2
- Look at his VoiceMail example