Software Development Study

Fawzi Emad
Chau-Wen Tseng
Department of Computer Science
University of Maryland, College Park

Covered So Far

- Software life cycle
- Object-oriented design & programming
- Unified Modeling Language (UML)
Assume You Already Know

- Object-oriented programming in Java
- Class definitions
  - State
  - Behavior
  - Identity
- Inheritance
  - Extending a class
  - Abstract classes
  - Interfaces
  - Polymorphism
  - Class object

If you need a refresher, read Chapter 3 in book

Today

- Case study in OOP
- Testing
Object-Oriented Design Case Study

Problem Specification

Software design
1. Identifying classes
2. State and behavior
3. Inheritance and interfaces
4. UML diagrams

Testing
1. Unit test
2. Integration test
3. Acceptance test

Problem Specification

Specification document
- Build a heating simulation that models behavior of
  - Living room
  - Thermostat
  - Furnace
  - Environment outside room
- Advance simulation clock every 60 seconds
- Calculate temperature using formula (in book)
- Turn furnace on/off depending on room temperature
- Output temperature until simulation length reached
Software Design

1. Identifying classes
2. State and behavior
3. Inheritance and interfaces
4. UML diagrams

Design – Identifying Classes

- Find nouns in specification
  - Simulation
  - Room
  - Thermostat
  - Furnace
  - Environment
  - Clock
  - Temperature
Design – Identifying Classes

- Find nouns in specification
  - Simulation ⇒ Entity outside program
  - Room ⇒ Entity in program
  - Thermostat ⇒ Entity in program
  - Furnace ⇒ Entity in program
  - Environment ⇒ Entity in program
  - Clock ⇒ Entity in program
  - Temperature ⇒ State of entity

Design – State and Behavior

- Find state for each class
  - Nouns in specification (not representing classes)
  - State required for function
  - Add to class as instance variables

- Find behavior for each class
  - Verbs in specification
  - Interactions between classes
  - Constructors & destructors
  - Add to class as methods
State – Instance Variables

- **Environment**
  - Temperature

- **Furnace**
  - On / off state
  - Capacity, efficiency (from formula)

- **Room**
  - Temperature
  - Area (from formula)
  - Reference to furnace
  - Reference to environment

---

State – Instance Variables

- **Thermostat**
  - Desired temperature setting
  - Amount of overheating (from formula)
  - Reference to furnace it controls
  - Reference to room it is in

- **Clock**
  - Current time
  - Interval between ticks
Behavior

Find verbs in specification

- Models (behavior)
- Advance (clock)
- Calculate (temperature)
- Turn on/off (furnace)
- Output (temperature)

⇒ outside scope of class
⇒ mutator for Clock
⇒ mutator for Room
⇒ mutator for Furnace
⇒ mutator for Room
Behavior – Class Methods

- **Environment**
  - Double getTemperature()
  - Void setTemperature(double t)

- **Furnace**
  - Boolean isHeating();
  - Void setHeating(boolean onOff)

- **Room**
  - Double getFloorArea()
  - Furnace getFurnace()
  - Environment getEnvironment()
  - Void determineTemperatureChange()

Behavior – Class Methods

- **Thermostat**
  - Room getRoom()
  - Furnace getFurnace()
  - Double getSetting()
  - Void setSetting(double newSetting)
  - Double overHeat()
  - Void determineStateChange()

- **Clock**
  - Clock(int tickInterval)
Design – Inheritance and Interfaces

Select inheritance and interfaces
- Specialize existing classes
- Allow future sharing of state & behavior

Inheritance

Add GasFurnace
- Add pilot light
- Specialize existing Furnace

Example
- GasFurnace extends Furnace
- New state
  - pilotLight
- New behavior
  - Boolean isPilotOn()
  - Void setPilot(boolean onOff)
Interfaces

- **Add ClockListener**
  - Useful for simulations
  - Allow objects to update their state based on clock

- **Example**
  - Interface ClockListener
    - Void preEvent(double timeInterval)
    - Void event()
  - Room implements ClockListener
  - Thermostat implements ClockListener

---

**Design – UML Diagrams**

![UML Diagram]
Design – UML Diagrams

Testing

- Goal
  - Detect and eliminate errors in program
  - Feedback to improve software
    - Specification changes
    - Add new functionality
  - Extremely important for success!
Testing

Techniques

- **Clear box testing**
  - Allowed to examine code
  - Attempt to improve thoroughness of tests

- **Black box testing**
  - Treat program as "black box"
  - Test behavior in response to inputs

Testing

Stages

- **Alpha test**
  - Test components during development
  - Usually clear box test

- **Beta test**
  - Test in real user environment
  - Always black box test

- **Acceptance**
Testing

- Empirical testing
  - Test software with selected test cases
  - More scalable than verification
  - Not guaranteed to detect all errors

- Steps
  1. Unit test
  2. Integration test
  3. Acceptance test

Unit Test

- Test individual units extensively
  - Classes
  - Methods

- Central part of “eXtreme Programming” (XP)
  - Extensive unit testing during development
  - Design unit tests along with specification

- Approach
  - Test each method of class
  - Test every possible flow path through method
Flow Path

- Unique execution sequence through program

Example

```java
S1
while (B1) {
    if (B2)
        S2
    else
        S3
}
```

Flows

- S1
- S1, S2
- S1, S3
- S1, S2, S2
- S1, S2, S3
- S1, S3, S2
- S1, S3, S3
- ...

Unit Test

- Not possible to test all flow paths
  - Many paths by combining conditionals, switches
  - Infinite number of paths for loops
  - New paths caused by exceptions

Test coverage

- Alternative to flow path
- Ensure each line of code tested
- Does not capture all possible combinations
Integration Test

- Test interaction between units
  - Possible units fail when combined
  - May find problems in specifications

- Approach
  - Test units together
  - Proceed bottom up, in increasing size

- Example test sequence
  1. AB, AC, AD, CD, CE
  2. ACD
  3. ABCDE

Acceptance Test

- Test entire software

- Approach
  - Place software in user environment
  - Test software with
    - Real-world data
    - Real users
    - Typical operating conditions
  - Test cases selected by users
  - Ensure software meets specifications
Testing – Heating Simulation

Unit tests
- Constructors for each class
- Methods for each class

Integration tests
- Test Room / Thermostat with Furnace
- Test Room / Thermostat with ClockListener

Acceptance tests
- Run simulations with different parameters
- Check program produces correct results
- Ensure program terminates