CMPS 115 Winter 04

Class #15 (2004/02/24)

- Changes/Review
- interfaces, Java and more tips
- <break>
- Configuration Management
- Presentation Expectations
Changes & Lecture 14 Takeaway

- Changes/Notices/Comments
  - See new syllabus; interfaces/CM today; evolution Thurs
  - Th: presentations + shorter COTS discussion
  - Exam 2: due 3/9; hand out 3/2 or 3/4?

- Takeaway lecture 14
  - OO Testing
    - State-based metrics (transition coverage, event coverage)
    - Mock Objects (static - easy; dynamic - flexible, use tools)
  - Performance Design issues
    - network latency usually dominates distributed apps; shared resources and control dominates multi-thread apps; algorithms & coding dominates compute-intensive apps
    - select 3 - 5 issues, work off one at a time before selecting next 3 - 5
    - for each: observe, hypothesize, test (include warm-up), repair, regression
    - performance is data and context sensitive
Where Do We Stand?

- **Points**
  - 70% done
  - 180 rem proj
  - 120 rem lect
  - EC: 3 lecture opportunities left, plus papers

- **Time**
  - 75% done
  - CM, evolution lectures
  - Newer paradigms: COTS, frameworks, aspects
Types, Abstract Classes, Interfaces, oh my ..
Types Review

- A “type” is a collection of operations
- Java supports two kinds of types: *primitive* and *reference*
  - primitive types are int, boolean, String, etc: naming == using, special syntactic sugar, fixed operations
  - reference types (“objects”) offer:
    - a level of indirection: naming == finding, then message == using
    - a Class object (holds behavior, *static* methods, etc)
    - inheritance, partial implementation, meta-data
- Operations in type contract can be
  - overloaded (select by types in signature: \( \text{foo(int I)} \) \text{ vs. } \text{foo(String S), int + \{int/byte/String/…\}})
  - overriden in reference type subclasses (replaced by inheritance; polymorphism)
    - *final* prevents overriding
    - *abstract* demands overriding
Abstract Classes vs Interfaces

- **Interfaces**
  - hold constants (pub static final) and method sig
  - allow multiple inheritance of specification
  - later initialization possible (first-use)
  - good for composition, flexibility

- **Abstract classes**
  - can have partial implementation (some methods)
  - can provide class constants
  - good for library/base code
  - single-inheritance limitation
Motors Example

Abstract Classes and Interfaces
Abstract Class Design

see http://www.flashline.com/content/Meyer/tmeyer1.jsp

Fig 1

Motor
  (abstract)
  getHorsePower() : int

BatteryPoweredMotor
  (abstract)
  getTimeToRecharge() : int
  getLumensToOperate() : int

SolarPoweredMotor
  (abstract)
  getTimeToRecharge() : int
  getLumensToOperate() : int

Fig 2

modeling just behavior

<<interface>>
Motor
getHorsePower() : int

<<interface>>
BatteryPoweredMotor
getHorsePower() : int
getTimeToRecharge() : int
getLumensToOperate() : int

<<interface>>
SolarPoweredMotor
getHorsePower() : int
getTimeToRecharge() : int
getLumensToOperate() : int

Fig 4, factored

<<interface>>
DualPoweredMotor
getHorsePower() : int
getTimeToRecharge() : int
getLumensToOperate() : int

<<interface>>
BatteryPoweredMotor
getHorsePower() : int
getTimeToRecharge() : int
getLumensToOperate() : int

<<interface>>
SolarPoweredMotor
getHorsePower() : int
getTimeToRecharge() : int
getLumensToOperate() : int

Fig 3

inherit behavior, not impl
Factory Design

Draw a UML structure diagram for an AbstractFactory design for getting an object which implements a “Connection” interface. The Connection interface has four methods on it:

- `connect(String destination)` which returns a boolean,
- `sendMessage(String s)` which transmits a message,
- `getMessage()` which returns a `String`, and
- `disconnect()` which has a void return.

There are two concrete Connections you might serve up: a `TcpConnection`, and a `TelephoneConnection`.

To get you started: the client will get the `AbstractFactory` by calling `MyConnectionFactory.getInstance()`. This method (which you do not have to code!) looks at a property value and returns either a `TcpConnectionFactory` or a `TelephoneConnectionFactory`.
HW2 P3

MyConnectionFactory
+getInstance():Connection
+createConnection():Connection

client

interface Connection
+connect(destination: String)
+sendMessage(msg: String): void
+getMessage(): String
+disconnect(): void

creates

TcpConnectionFactory
+CreateConnection(): Connection

TcpConnection
+connect(destination: String):
+sendMessage(msg: String): void
+getMessage(): String
+disconnect(): void

creates

TelephoneConnectionFactory
+CreateConnection(): Connection

TelephoneConnection

creates
Convert inheritance to composition/delegation

Composition: inseparable

interface-driven
interface ICustomer
* +problem:String=(
+getProblem():String
+getIdentity():SSN

interface IEmployeeRole

+getIdentity():SSN

* works for

1

interface IManager

interface IClerk

+getProblemKeywords(custID:SSN):String

1

Customer
-SSN
-famName:String
+getProblem():String

Person
-famName:String

Employee
-SSN
-famName:String
-myCustomers:HashMap=null

Manager

Clerk

SSN

0..1

identity

0..1

identity

Manager

Clerk

Customer

Person

Employee

IClerk

IManager

ICustomer

IEmployeeRole

interface
Another Composition Example

```java
// BankAccount class
class BankAccount {
    int balance;
    public int deposit(int amount) { balance += amount; }
}

// BankAccountFactory class
class BankAccountFactory {
    static BankAccount getInstance() {
        return new BankAccount();
    }
}

// Loggable interface
interface Loggable {
    void setLogStream(Stream s);
}

// Now, create a LoggableBankAccount...
```
Design with Composition, Inheritance, Interface

- Use classes to say "What objects are."
- Use class extension to model IS-A-KIND-OF relationships.
- Use composition to enlist the help of other objects.
- Understand the difference between inheritance and composition.
- Understand the significance of the interface.
- Use interfaces to say "What objects can do," or occasionally, "What can be done to an object."
- Use interface extension to model INCLUDES-A relationships.
**W/E 24 Feb:**
A: 1 Snake for Brian’s syllabus idea
A: 2 Astrix early sched inspection
A: 3 Astrix team performance on HW2 P3, P4
W: 1 Astrix for Melissa’s syllabus idea
W: 2 Astrix Horia’s J2ME writeup
W: 1 Snake Niraj’s suggestion to move up E2

Week ends Tuesday at class start

<table>
<thead>
<tr>
<th></th>
<th>27-Jan</th>
<th>3-Feb</th>
<th>10-Feb</th>
<th>17-Feb</th>
<th>24-Feb</th>
<th>2-Mar</th>
<th>9-Mar</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Astrix</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td></td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>BattleSnake</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>Caesar's</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>CLASS TOT</td>
<td>9</td>
<td>17</td>
<td>30</td>
<td>43</td>
<td>53</td>
<td>53</td>
<td>53</td>
<td>53</td>
</tr>
</tbody>
</table>

Current week:
Instructor awarded 4/6
TA awarded 6/9

Averaging (53/5)=10.6/wk * 7 weeks = 76 BP … need 80 BP
Need 14/wk rate now

**NO FREE PIZZA AT THIS RATE**

… but it’s very close … guest lecturers give BP for Q&A
Configuration Management

Change is inevitable

Artwork taken from The Subversion Book
Client/Serve CM

Repository

write
Sally

read
Harry
read
Ira
Client/Serve CM Problem

Two users read the same file.

They both begin to edit their copies.

Harry publishes his version first.

Sally accidentally overwrites Harry’s version!
Lock/Mod/Unlock Approach

Repository

Harry
A

LOCK
read

Sally
A

Harry “locks” file A, then copies it for editing.

While Harry edits, Sally’s lock attempt fails.

Repository

A'

write
UNLOCK

Harry
A'

Sally

Harry writes his version, then releases his lock.

Repository

A'

LOCK
read

Sally
A'

A'

Now Sally can lock, read, and edit the latest version.
Lock/Mod/Unlock Drawbacks

- Restrictive
  - Administrative problems (Harry locks, goes on vacation…)
  - Unnecessary serialization (what if they’re working on separate parts of the file?)

- False sense of security
  - Pretend that Harry locks and edits file A, while Sally simultaneously locks and edits file B. But suppose that A and B depend on one another, and the changes made to each are semantically incompatible.

  Suddenly A and B don't work together anymore.
Copy/Mod/Merge Approach

Repository

A

Harry

A

read

Sally

A

read

Two users copy the same file.

Repository

A

Harry

A'

write

Sally

A''

They both begin to edit their copies.

Repository

A''

Harry

A'

write

Sally

A''

Sally publishes her version first.

Repository

A''

Harry

A'

Harry gets an “out-of-date” error!
Copy/Mod/Merge Approach - con’t

Harry compares the latest version to his own.

A new merged version is created.

The merged version is published.

Now both users have each others’ changes.
What If Changes Overlap?

- Physically overlap
  - caught by merge process, rejected as *merge conflict*
  - depends on differencing algorithm (text is easiest)
- Semantically overlap
  - caught by unit tests
  - avoided by management scheduling one person to work on one semantic issue; supported by clear separation of design areas in code
- Practically, not a big problem
See …

- The Subversion Book (practical, easy) at
  http://subervsion.tigris.org

- Version Management with CVS by Per
  Cederqvist et al is the "official" manual for CVS.
  Commonly known as "the Cederqvist"
  http://www.cvshome.org/docs/manual/

- CM Concepts reading (S. Dart SEI Report)
  http://www.sei.cmu.edu/legacy/scm/abstracts/abscm_concepts.html

- Impact of CM report in DForge Docs area
ALL DONE FOR TODAY
Branching

Original line of development

1st branch

3rd branch

2nd branch

trunk, also main branch

time
Creating a Branch

Subversion branch is a lazy copy: just links until file change.
Revision, Tag, Release

- Revision - a state of a file or file system
  - in Subversion, an entire filesystem tree, updated on each atomic commit (may change multiple files at once); in CVS, each file get new version number on commit

- Tag - a name for a revision or group of revisions
  - in Subversion, you “tag:” by copying; in CVS, tag is entry in files

- Release - has process-specific meaning, usually a tag for a tested/doc’ed build
Presentation Expectations: Thursday

- 15 minutes each
- Status report, not demo
  - Summary of app (2 minutes, 1 diagram)
    - what’s the allocation: what is a client, a server, how do they connect
    - what’s the platform (J2ME/J2SE, assumed packages)
    - general functional condition of app; what’s running now?
  - Physical modularization (package structure, dependencies, deployment units)
  - How are you doing against your time/risk estimates?
  - Prognosis?
- Start at eight, don’t be late - it’s as rude for us to enter late as for next class to enter early :)
Design With Types

- Make the types of your method parameters and return values as precise as possible.
- Prefer dynamic binding to `instanceof` and downcasting.
- Make sure subtypes fulfill the semantic contract of their supertypes. (Contract includes docs)
- Use `instanceof` and downcasting to ask "Can you do something for me?"
- Avoid optional clauses in semantic contracts.
- Prefer compile-time type information to run-time class information.