Comp-303 : Programming Techniques

Lecture 1

Alexandre Denault
Computer Science
McGill University
Winter 2004

Course Description

As found on Minerva . . .

Software architecture, design patterns, object-oriented programming concepts, profiling and optimization. Students will implement a significant programming project.

Course Content

Comp-303 is . . .

- How does *Java* object orientated programming work and why is it useful?
- How do I build *Java* code that can be easily extended?
- How do I build *Java* code that is easy to understand?
- How do I manage a large *Java* project (large amounts of code)?
- How do I deal with problem code or problems in code?
- How do I gather project requirements and properly design my applications?
- Is there any proven techniques I use when designing software?
- How do I use *Java* object orientated programming to make reusable components?

Course Content (cont.)

Comp-303 is *not* . . .

- How do I improve my running time by 14% ?
- How do I profile my program?
- How do I build a GUI in Java?
- How do I use Java feature X ?
- How do I implement program X ?
- How do I program in C++ ?
- How do I sort a list in O(n log n) time?

Instructor and Teacher's Assistant

- Instructor:
  Alexandre Denault - alexandre.denault@adinfo.qc.ca
  - Office: McConnell 322 (cubicle in the back)
  - Office Hours:
    - Tuesday & Thursday 1h00 - 2h30
    - or send me an email ...
- Teacher's Assistant:
  Sokhom Pheng
  - Office: TBD
  - Office Hours: TBD

Lecture Schedule and Prerequisites

- Lectures:
  - Tuesday and Thursday, 4h05-5h25
  - Trottier Building 0070
- Prerequisites:
  - COMP 206
  - COMP 251
  - COMP 302

Restriction Note: Open only to students registered in a Core Group* or Mathematics Group* program, or the Minor in Computer Science. * as defined in the SOCS section, Undergraduate Programs Calendar.

Workload and Grade Distribution

- This course has a very heavy workload (it's a 4 credit course).
- You will be require to put in practice the material learned in class, both in the assignments and in the final project.
- Warning: Do not take more than 2 classes requiring you to complete a large project per term.
- Grade Distribution
  - Homework Assignments (3) : 30%
  - Midterm (2) : 20%
  - Project : 50%
    - Design Doc : 3%
    - Status Meeting: 2%
    - Final Product: 45%

Assignments

- Allows you to practice the material seen in class.
- Allows me to evaluate what you have learned.
- Each assignment is worth 10% of your grade.
- Tentative dates:
  - Assignment 1 : January 20th - February 3rd
  - Assignment 2 : February 10th - March 2nd
  - Assignment 3 : March 9th - March 23rd
- You have a buffer of 3 late days (to use as you wish)
- If you want to use a late day, simply mention it in your readme file.
- Assignments will be handed in paper format (in class) and on WebCT.
- The T.A. will correct the assignments.

Midterm

- Short midterms to allow me to see if you understand the material.
- If we didn't see it in class, it's not in the midterm.
- Tentative dates:
  - Midterm 1 : Thursday, February 19th
  - Midterm 2 : Thursday, April 8th

Project

- Non-trivial project that allow you to use the material seen in class.
- The project must be completed in teams of 3 or 4.
- The project must have a high level of complexity (+/- 20 classes per student).
- Games (i.e. board games) have always been a popular topic.
- Milestones and Deadlines:
  - Requirement and Specification Doc. : Thursday, January 29th
  - Interview with T.A. : Week of March 1st
  - Final product (with some documentation) : Thursday, April 8th
  - Interview with Teacher/T.A. : Week of April 19th

Textbook

Required Textbook:

- Program Development in Java: Abstraction, Specification, and Object-Oriented Design
  by Barbara Liskov and John Guttag, Addison Wesley 2001

Other good textbooks:

- Design Patterns Explained: A New perspective on Object-Oriented Design
  by Alan Shalloway and James R. Trott, Addison Wesley 2002
- Java Design Patterns: A Tutorial
  by James W. Cooper, Addison Wesley 2000

Slides

- Why use slides?
  - Because my handwriting is horrible on the board.
  - Because it help me to *not forget* material.
- Why do the slides look weird?
  - Because I'm learning to use LaTeX.
  - Because learning LaTeXis as intuitive as learning to skate by yourself.

Decomposition

- Divide a large tasks in smaller components.
- Easier to complete smaller components individually.
- When programming, divide a project in smaller modules with little interaction.
  - Different people can implement different modules independently.
  - Maintain and modify in a controlled manner with limited effect (no spaghetti code)
- Dividing into subproblems
  - Subproblems approximately same level of detail.
  - Subproblems can be solved independently.
  - Solutions to subproblems can be combined to solve the whole problem.

Non-CS Example to Decomposition

- Renovating an old house can be a daunting project.
- Many different aspects of the house may need repairs.
- The project will be easier to complete if the tasks are divided:
  - Fix electric wiring
  - Check plumbing and replace leaky pipes
  - Fix holes in wall
  - Refinish wooden floors
  - etc ...
- Or the project can be decomposed another way ...
  - Renovate Kitchen
  - Renovate Bathroom
  - etc ...
- The important thing is not to tackle the whole project at once.

A CS Example to Decomposition

- An Instant Messaging application can be a challenging project.
- Fortunately, it is easy to decompose:
  - Design Communication Protocol
  - Build authentification engine
  - Build connection tracking component
  - Build messaging component
  - Build chat component
  - Build message transfer component
  - Build audio component
  - Build video component
  - etc ...

Art of Decomposition

- It is easy to solve subproblems independently.
- The hard part is to combined them.

- Problem: Write a play using n writers.
- Nave decomposition: Each writer takes a character and goes off to write the character's dialog lines independent from other writers.
  - incoherent nonsensical result that is counter-productive decomposition

Abstraction

- Decompose by changing the level of detail to be considered.
- It allows us to forget information and consequently to treat things that are different as if they were the same.
- For example, on you harddisk, you will find hundreds of different types of files (Spreadsheet, Binary, Text, etc).
- However, a file manager takes abstraction of this and treats all file equally (move, copy, erase, etc).
- Another common example would be programming languages and loops.
- When programming in C, we use *while* and *for* instructions to build loops of all kinds.
- This is an abstraction to the dozen of machine code instruction used to create loops.

Non CS example to Abstraction

- Abstraction can be done at many different levels:

- Fish
  - Shark
  - Salmon
- Reptile
  - Frog
  - Snake
- Mammal
  - Rodent
  - Cetacean
  - Primate
    - Chimpanzee
    - Human

Abstraction in Programming

- As mentioned previously, abstraction is used in programming languages.
- In high-level programming languages, constructs are provided to programmer. (For example, set operations)

```
Set a;
if (a.isIn(e)) {
   z = a.indexOf(e);
}
```

- It is impossible to predict all the abstraction that could be needed.
- That is why programming languages provide tools for abstraction.

Abstraction by parameterization

- Abstract from the identity of data by replacing instances by parameters.
- *Generalizes* modules to be used in more situations.
- For example . . .

  ```
  x * x + w * w;
  ```

- . . . could be replaced by . . .

  ```
  sumsquares(x,w);
  ```

- . . . where *sumsquares* is a function that sums the square of both of it's *parameters*.
- Functions can be used to describe an infinite number of computations.
- This is easy to realize in current programming languages.

Abstraction by specification

- Abstract from the computation described by a procedure to the end that procedure was designed to accomplish.
- For example, my specification documentation describes a function that returns an approximation of the square root of X by . . .
- An abstract to this description would be:

```
float ans = x /2.0;
int i = 1;
while (i < 7) {
   ans = (ans + coef / ans ) / 2.0;
   i++;
}
return ans;
```

Kinds of abstractions

Abstraction by parameterization and abstraction by specification are tools to construct different kinds of abstraction:

- Procedural abstraction
- Data abstraction
- Iteration abstraction
- Type hierarchy

Procedural abstraction

- Procedural abstraction introduces new operations
- Adds functionality to the machine defined by a high-level language
- Useful if a problem can be decomposed into independent functional units.
- Uses both parameterization and specification

Data abstraction

- Data abstraction introduces new *data types*.
- Data objects are expressed as sets of operations that are meaningful for those objects:
  - create objects
  - get information
  - modify objects
- For example, MultiSets are sets that can store more than one instance of the same element:
  - insert
  - delete
  - numberOf
  - size

Iteration abstraction

- Iteration abstraction allows us to iterate over items in a collection without revealing details of how the items are obtained.

```
i = s.iteration();
while (i.hasMoreElements()) {
   e = i.nextElement();
   e.doSomething();
}
```

- The order in which the elements are visited is abstracted.

Type hierarchies

- Type hierarchies allow us to abstract from individual types to families of related types.
- The common operations are defined in a supertype.
- Sub types define extra operations (and can themselves be ancestors to a family of subtypes).
- Example: the following types can be read from ...

```
Stream
   File
      BinaryFile
      TextFile
   Keyboard
      Socket
```

Summary

- Decomposition and abstraction are techniques to construct large programs that are easy to understand, maintain and modify.
- Abstraction allows us to ignore details and treat different objects as thought they were the same.
- Parameterization generalizes to wider applicability
- Four kinds of abstraction:
  - Procedural abstraction
  - Data abstraction
  - Iteration abstraction
  - Type hierarchy