Crash course in UML and the Project

Comp-303 : Programming Techniques
Lecture 4

Alexandre Denault
Computer Science
McGill University
Winter 2004
Announcements

- Assignment 1 will be handed out next class.
Last lecture . . .

• Variables, Objects and Primitives
• Object variables are called references
• Objects can be mutable or immutable
• Java is strongly typed and type-safe
• Java provides automatic garbage collection
• All objects are subtypes of Object and understand toString() and equals()
• Primitive types are converted to other types
• All types can be cast to other types (no conversion)
• Packages provide encapsulation and naming scope
• java.util provides Vector
• Executions starts at main() method
• Tool of the day: CVS
What is UML?

- UML is a visual language used to create models of programs or modules.
Why learn UML?

- To communicate with others.
- To give a graphical representations to your ideas.
- To avoid confusion in design/communication
- Because you might need it for the project
- Because it looks really cool in your C.V.
Why use UML?

• UML has guidelines for drawing several types of diagrams:
  – Use Case Diagrams
  – Activity Diagrams
  – Interaction Diagrams
  – Sequence Diagrams
  – Class Diagrams
  – State Diagrams
  – Deployment Diagrams

• Don’t worry, we won’t need all of these.

• We will mostly work with Class Diagrams, State Diagrams and Sequence Diagrams.
Good references on UML

- Fundamentals of Object-Oriented Design in UML; Meilir Page-Jones; Addison-Wesley; 2000
• Dia is a gtk+ based diagram creation program released under the GPL license.

http://www.lysator.liu.se/~alla/dia/

• Dia is a free program that allows you to draw your UML diagrams.

• All the UML diagrams in these notes were created with Dia.

• Dia is available on Linux and Windows.

• Dia2code allows you to generate code from your UML class diagrams.
Class Diagrams

• This is the type of diagram we will use most often in class.

• It allows you illustrate both the structure and the relationships of your classes.

• There are 3 kinds of relationships between classes:
  – Hierarchy
  – Contains
  – Uses
Classes in a class diagram

Three elements can be represented in a class box:

- name
- members / properties
- methods / functions

<table>
<thead>
<tr>
<th>Book</th>
</tr>
</thead>
<tbody>
<tr>
<td>-chapterList: Vector</td>
</tr>
<tr>
<td>+title: String</td>
</tr>
<tr>
<td>+author: String</td>
</tr>
<tr>
<td>+read(): String</td>
</tr>
<tr>
<td>+write(chapter:int,text:String): void</td>
</tr>
</tbody>
</table>

- The ”+” symbol indicates a public member
- The ”-” symbol indicates a private member
- If the ”#” symbol was used, it would indicate a protected member
Hierarchy Relation

• The first type of relationship is hierarchy, also known as inheritance.

![Diagram](image_url)

• An arrow is used to illustrate the inheritance relationship.

• In this diagram, Manual and Art Book are a subtype of Book.

• If the name of the class Book were written in italic, this would indicate the class is abstract.
The second type of relationship is *contains*, also known as composition or aggregation.

- A full diamond indicates a composition relation. This means Class book cannot exist without class Page.
- An empty (white) diamond indicates an aggregation relation. This means class Book can be contained in class Store. However, class Store does not need class Book to exist.
• The last type of relationship is *uses*.

![Student to Book Use Relationship Diagram]

• This diagram shows that the class Student uses the class Book.

• The class Student does not define itself with the class Book, it just uses it.

• Notice that the arrow seems to be doing in the wrong direction.
Cardinality of a Relation

• Sometimes, it becomes important to define the cardinality of a relation.

• In other words, we want to count the number of object that can be found on each side of the relation.

  ![Diagram]

  • In this example, we can have either no book or one book ($0..1$).
  • We also define that our scenario requires at least one page ($1..*$.)
  • We can’t define a scenario with zero pages because that would violate the composition rule with class Book.
Adding Notes

- UML is not flexible enough to describe all scenarios.
- To describe complicated scenarios, designers can use notes to annotate their designs.

In this example, the design must add a constraint forcing the number of pages to be even.

- UML provides no primitive or easy solution.
- However, the designer can simply add the constraint as a note in the diagram.
Combining all the previous elements

- We can combine all the previous elements to produce complete UML diagrams.
State diagrams are used to describe the various state a program can reach.

This diagram shows the various states for a typical lamp.
Sequence Diagram

- Sequence Diagrams are used to show the order in which objects are instantiated and used (object’s life).

- This example shows a program finding a file and opening it.

```
Main

Finder

Reader

find file ()

file location

read file ()

file content
```
Hints for the project

- You should have a team by now. If you don’t come and see me after class.
- Choose a project leader (not a dictator).
- You’re team should be talking \textit{at least} a hour each week.
- Have some technique to control your sources (CVS ?).
- Assign modules/packages depending on people’s skill.
- Start early. There is not penalty for finishing a project early.
• Before attempting any large scale project, it is imperative to spend some time writing out the Requirements & Specifications.

• Requirements can be described as requests for a particular piece of software.

• Specifications can be described as the proposed solutions for the requirements.

• Validating the Requirements & Specifications with the client before starting a project is always a smart idea.

• Requirement Analysis and Specifications are covered latter in the class, so I’ll be very brief on this topic.
Requirements

• As mentioned before, requirements are *the requests* for a particular piece of software.

• For the project, the main requirements are defined by the professor (programming language, size of project, etc.).

• However, the main requirements are too loss to properly define your project.

• Once you have chosen your project, it is up to your team to define your requirements / goals for this project.
Specifications

• Specifications are proposed solutions to clearly stated and approved requirements.

• For the projects, your initial specifications should be a general overview of how your project will work and how you will design your module.

• Your specifications should (at a minimum) explain the four following concepts:
  – Concept : What does it do?
  – Appearance : What does it look like?
  – Controls : How can I use it?
  – Behavior : How does it work?
Specifications (cont.)

- Your specification should include a few class diagrams (doesn’t need to have properties and method).

- Specifications are not binding:
  - It is normal to sometime deviate from the initial specifications.
  - However, when deviating from the initial specifications, it is important to document the changes.
  - Too many deviations from the initial specifications is a clear indication of a design flaw in the specification.
  - In a business context, variation on the initial specification is subject to client approval.
Example of Requirement.&Spec. Document

This example show how you could format your document.

• Front page
• Table of content
• Requirements
• Specifications
  – Concept
  – Appearance
  – Control
  – Behavior
    – Package A (class diag?)
    – Package B (class diag?)
    – Package C (class diag?)
More hints for the project

• Each student is expected to produce his/her package.

• Try to find a project idea that you can easily split into 3 or 4 packages.

• Simplify your project using modularity.
  – Your team only needs to agree on the public members of your package.
  – Everything private to your package is only your concern.
Summary

• What is UML and why learn it?

• References and Tools for UML

• Class diagrams
  – Elements of a class
  – Relations (Hierarchy, Contains, Uses)
  – Cardinality
  – Notes
  – Putting it all together

• State diagrams

• Sequence diagrams

• Requirements & Specifications
Tool of the day: Labyrinth

- Java reproduction of the Labyrinth game.
- Has the following features:
  - Client/Server model
  - Gui Interface
  - Multiplayer over Internet
  - Computer playing with 8 A.I. personality
- Too complex for 1 student to complete.
- Ideal for a group of 4 students.