Unified Modeling Language (UML) Diagrams for Classes, Objects, and Sequences

CS 370 SE Practicum, Cengiz Günay

(Some slides courtesy of Eugene Agichtein and the Internets)
Make-up lecture:

- You’ve already made a data model, but here’re some more details
- Unified Modeling Language (UML) Diagrams for Classes, Objects, and Sequences
Exit survey: Entrepreneurship, User Stories, Legal Protection

- From the two startups you heard about, what was the hardest part about starting a company in your opinion?
- What’re benefits of describing project requirements in terms of user stories?
- What legal protection do you prefer to seek for your software? Copyright, trademark, or patents?

Entry survey: UML diagrams

- Briefly tell me about an object-oriented feature that you used in your previous programming experiences.
- In data model diagrams, what else do you think would be useful to depict in addition to data attributes and relationships?

Warning: Submit on Blackboard, but you won’t get points unless you also answer the exit survey at the end.
Data model: Entity-relationship (ER) diagrams

- Entities
- Attributes
- Relationships (aggregation, multiplicity, ...)

What’s missing?
Data model: Entity-relationship (ER) diagrams

- Entities
- Attributes
- Relationships (aggregation, multiplicity, ...)

What’s missing?

- Methods/functions
Classes: Encapsulate Data + Methods

Object-oriented (OO) programming principles:

- Used by all mainstream software today
- Not have a good grasp for it? Watch this lecture:

  https://www.youtube.com/watch?v=DdUSzJ8taMs
Classes: Encapsulate Data + Methods

Object-oriented (OO) programming principles:

- Used by all mainstream software today
- Not have a good grasp for it? Watch this lecture:

  https://www.youtube.com/watch?v=DdUSzJ8taMs

- Now we’ll learn how to depict classes in diagrams with UML
UML class diagrams

• What is a UML class diagram? What does it represent?
  – A picture of the classes in an OO system, their fields and methods, and connections between the classes that interact or inherit from each other

• What are some things not represented in a class diagram?
  – details of how the classes interact
  – algorithmic details; how particular behavior is implemented
  – trivial methods (get/set)
  – classes that come from libraries (ArrayList, etc.)
Diagram of one class

- **class name** in top of box
  - write `<interface>` above interfaces' names
  - use *italics* for an *abstract class* name
- **attributes**
  - should include all fields of the object
  - also includes derived “properties”
- **operations / methods**
  - may omit trivial (get/set) methods
  - should not include inherited methods
Class attributes

• **attributes** (fields, instance variables)
  – visibility:  
    + public
    # protected
    – private
    ~ package (default)
    / derived

  – underline **static attributes**

• **derived attribute**: not stored, but can be computed from other attribute values
Class operations / methods

- operations / methods
  - visibility name (parameters) : returnType
  - underline static methods
  - parameter types listed as (name: type)
  - omit returnType on constructors and when return is void
Comments

- represented as a folded note, attached to the appropriate class/method/etc by a dashed line

```
ArrayList

interface Cloneable

```

Cloneable is a "tagging" interface with no methods. The clone() method is defined in the Object class.
Relationships between classes

- **generalization**: an inheritance (is-a) relationship
  - inheritance between classes
  - interface implementation

- **association**: a usage (is-part-of) relationship
  - dependency
  - aggregation
  - composition
Generalization relationships

- Hierarchies drawn top-down with arrows pointing upward to parent
- Line/arrow styles differ based on parent
  - class: solid, black arrow
  - abstract class: solid, white arrow
  - interface: dashed, white arrow
- Trivial / obvious relationships, such as drawing the class `Object` as a parent, are generally omitted
Associational relationships

1. multiplicity   (how many are used)
   - *  \(\Rightarrow\) 0, 1, or more
   - 1  \(\Rightarrow\) 1 exactly
   - 2..4  \(\Rightarrow\) between 2 and 4, inclusive
   - 3..*  \(\Rightarrow\) 3 or more

2. name   (what relationship the objects have)

3. navigability   (direction)
Multiplicity

• one-to-one
  – Ex: each student must have exactly one ID card

• one-to-many
  – a RectangleList can contain 0, 1, 2, ... rectangles
Association types

• **aggregation**: "is part of"
  – clear white diamond

• **composition**: "is entirely made of"
  – stronger version of aggregation
  – the parts live and die with the whole
  – black diamond

• **dependency**: "uses temporarily"
  – dotted line or arrow
  – often is an implementation
The Voting Program

VoterAuthentication
- voterPersonalInfo: VoterPersonalInformation
- voterID: String
- voterPassword: securePW

VoterPersonalIdentification
- voterLastName: String
- voterFirstName: String
- voterMiddleName: String
- voterSSN: String
- voterAddress1: String
- voterAddress2: String
- voterCity: String
- voterState: String
- voterZIP: String

BallotCreation
- ballotName: String
- candidates: String []
- displayBallot(): void
- createBallot(): void

+ validateZipCode(voterZIP: String): String
+ validateState(parameter0VoterState: String): String

securePW
- PWEntered: JPasswordField
- securePW(PW: securePW): securePW

this is only a small subset of the actual package ...
Class

Abstract Class

Generalization

Multiplicity

Simple Aggregation

Composition

Simple Association

Rental Item

Customer

Rental Invoice

Checkout Screen

DVD Movie

VHS Movie

Video Game

1..*

1

1

0..1
Tools for creating UML

• Violet (free)
  – http://sourceforge.net/projects/violet/

• Rational Rose
  – http://www.rational.com/

• Visual Paradigm UML Suite (trial)
  – http://www.visual-paradigm.com/
  – (nearly) direct download link:

• (there are many others, but many are commercial and cost money)
Other free UML tools

- Dia (Gnome): https://wiki.gnome.org/Apps/Dia
- Umbrello (KDE): http://umbrello.kde.org/
- Ting Ting used: http://creately.com/
• Class diagrams are good for
  – discovering related data and attributes
  – getting a quick picture of the important entities in a system
  – seeing whether you have too few/many classes
  – seeing whether the relationships between objects are too complex, too many in number, simple enough, etc.
  – spotting dependencies between one class/object and another
• Not so great for
  – discovering algorithmic (not data-driven) behavior
  – finding the flow of steps for objects to solve a given problem
  – understanding the app's overall control flow (event-driven? web-based? sequential? etc.)
From classes to individuals (objects)
Related: Object diagrams

- shows an individual object, rather than entire class
  - `objectName : type`
  - `attribute = value`
  - objects can be connected by lines that state the reason the two objects are talking
What about time?
UML sequence diagrams

- sequence diagram: an “interaction diagram” that models a single scenario executing in the system
  – perhaps second most used UML diagram (behind class diagram)
Sequence diagram key parts

• **participant**: object or entity that acts in the diagram
  – diagram starts with an unattached "found message" arrow

• **message**: communication between participant objects

• The axes in a sequence diagram
  – horizontal: which object/participant is acting
  – vertical: time (down -> forward in time)
Basic Course

The Customer specifies an author on the Search Page and then presses the Search button.

The system validates the Customer’s search criteria.

The system searches the Catalog for books associated with the specified author.

When the search is complete, the system displays the search results on the Search Results Page.

Alternate Course

If the Customer did not enter the name of an author before pressing the Search button, the system displays an error message to that effect and prompts the Customer to re-enter an author name.
Representing objects

• Squares with object type, optionally preceded by "name :“ (if it clarifies diagram)

• object's "life line" represented by dashed vertical line
Messages between objects

- messages (method calls) indicated by arrow to other object
  - write message name and arguments above arrow

```
message name
Admit (patientID, roomType)
```

```
arguments

:Hospital
```
Messages

• messages (method calls) indicated by arrow to other object
  – dashed arrow back indicates return
  – different arrowheads for normal / concurrent (asynchronous) calls
Lifetime of objects

- **creation**: arrow with 'new' written above it
  - notice that an object created after the start of the scenario appears lower than the others

- **deletion**: an X at bottom of object's lifeline
  - Java doesn't explicitly delete objects; they fall out of scope and are garbage-collected
Indicating method calls

- **activation**: thick box over object's life line; drawn when object's method is on the stack
  - either that object is running its code, or it is on the stack waiting for another object's method to finish
  - nest activations to indicate recursion
Selection and loops

- **frame**: box around part of diagram to indicate **if** or loop
  - **if** -> (opt) **[condition]**
  - **if/else** -> (alt) **[condition]**, separated by horizontal dashed line
  - **Loop** -> (loop) **[condition or items to loop over]**
Forms of system control

- What can you say about the control flow of each system?
  - Is it centralized?
  - Is it distributed?
WHAT'S THE POINT?
Why not just code it?

• Sequence diagrams can be somewhat close to the code level

• So why not just code up that algorithm rather than drawing it as a sequence diagram?
  - a good sequence diagram is still a bit above the level of the real code (not all code is drawn on diagram)
  - sequence diagrams are language-agnostic (can be implemented in many different languages)
  - non-coders can do sequence diagrams
  - easier to do sequence diagrams as a team
  - can see many objects/classes at a time on same page (visual bandwidth)
The trouble comes when people feel compelled to convey the whole model or design through UML. A lot of object model diagrams are too complete and, simultaneously, leave too much out. ... Nor is UML a very satisfying programming language.
Eric Evans, 2003, Domain-Driven Design: Tackling Complexity in the Heart of Software

The vocabulary and rules of a language such as UML tell you how to create and read well-formed models, but they don't tell you what models you should build and when you should create them. That's the role of the software development process.

The fundamental reason to use UML involves communication. ... Natural language is too imprecise and gets tangled when it comes to complex concepts. Code is precise but too detailed. So I use UML when I want a certain amount of precision but I don't want to get lost in the details.
So...

- UML allows you to say some of the things that languages don’t allow you to say explicitly about software systems.
- It can be used effectively; it can be used horribly.
  - Flon’s Law: Good programs can be written in any language; and bad programs can be written in any language.
- Knowing the basics is important – it’s a common lingo (and it sometimes shows up in interviews).
Exit survey: UML Diagrams

- When do you need an object diagram as opposed to a class diagram?
- Give an example from a program that you previously wrote where making a sequence diagram would have been meaningful.

**Warning:** Submit on Blackboard together with exit/entry surveys at the beginning.
Next class:

- Version Control Systems