Department of Computer Science
Undergraduate Events

Events this week

**How to Prepare for the Tech Career Fair**
Date: Wed. Jan 6
Time: 5 – 6:30 pm
Location: DMP 110

**Resume Writing Workshop (for non-coop students)**
Date: Thurs. Jan 7
Time: 12:30 – 2 pm
Location: DMP 201

**CSSS Movie Night**
Date: Thurs. Jan 7
Time: 6 – 10 pm
Location: DMP 310
Movies: “Up” & “The Hangover” (Free Popcorn & Pop)

**Drop-In Resume Edition Session**
Date: Mon. Jan 11
Time: 11 am – 2 pm
Location: Rm 255, ICICS/CS Bldg

**Industry Panel**
Speakers: Managers from Google, IBM, Microsoft, TELUS, etc.
Date: Tues. Jan 12
Location: Panel: DMP 110; Networking: X-wing Undergrad Lounge

**Tech Career Fair**
Date: Wed. Jan 13
Time: 10 am – 4 pm
Location: SUB Ballroom

---

**Administrivia**

- Labs start next Monday (January 11th)
- The midterm date and time is set:
  - Wednesday March 10th, 6:00 – 7:15pm
  - Add it to your calendars and let me know right away if you have an insurmountable conflict.
- The WebCT site appears to be up:
  - http://www.vista.ubc.ca/
  - Or the Vista link on the cs211 home page
- We are using WebCT for grades and the bulletin board
- Read the bulletin board regularly
Class Design I
Class Attributes and Methods

You will be expected to:

- determine some appropriate attributes for a class given a general description of the class
- determine some appropriate methods for a class given a general description of the class
- assess whether a given class description is cohesive and robust

Reading:
2nd edition
Chapter: 9,
Sections: 9.1-9.4, 9.6-9.9

3rd and 4th edition
Chapter: 8,
Sections: 8.1-8.4, 8.6-8.9

Some ideas come from:
- “Object-Oriented Software Development Using Java”, Xiaoping Jia, Addison Wesley, 2002

Design

What is design? What makes something a design problem? It’s where you stand with a foot in two worlds --- the world of technology and the world of people and human purposes --- and you try to bring the two together.


A concrete example… Technology?

Human purpose?
Design

A software example...

Many different aspects of design

Software program

Architecture

Human-computer interface
Software design

Based on a description of what the system should do (requirements), we need to identify and define:

- classes
- attributes of each class
- behaviour of each class
- relations between classes

During design, the focus is on how the system will work, not on the implementation (precise) details.

Design is guided by principles and heuristics, not definitive rules.

Example: A music system for a phone

- What a music system for a phone should be able to do…
  - Multiple formats
  - Play, FF, rew, Stop, Pause
  - artwork
  - sort/sdel in edit
  - sync
  - playlists
  - phone/player
  - add/delete
  - shuffle
  - loop
  - volume

- Let’s identify some classes…
  - Graphics
  - Player
  - Playlist
  - Phone
  - Sync
  - File
  - Song
  - Album
  - Setting
Class design (aka low-level design)

- Our focus now is on how to design a single class. We’ll assume that we know which class(es) we need; designing classes and their relationships will be a topic later this term.
- For each class we are designing, we need to define:
  - the data (attributes or fields) associated with the class’ concept
  - the behaviour (responsibilities, public services) associated with the class’ concept; this includes:
    - public methods
    - the class invariants
- We will ignore for now…
  - private methods
  - the data structures used to implement collections of data

Designing for one class: Identifying attributes

- **Objective**: identify and name all data that a class needs to support the behaviour of objects of that class.
- **Goal**: each class should have high cohesion
  - each class must represent a single concept
  - all data and operations must be highly related to each other
- **Initial heuristic**: consult the requirements (problem description), looking for adjectives and possessive phrases related to objects of the class of interest to discover what information the objects of the class will need.
- **Review**: eliminate any false attributes
  - attributes whose value depends on the context
    - e.g., Consider a Person class. Such a class is unlikely to have an employee_id attribute because a person may have zero, one, or more jobs.
  - attributes that are unrelated to the rest
    - either these attributes do not belong or the class should be split.
Designing for one class:
Designing each attribute

- For each attribute, must distinguish:
  - Kind of attribute
    - **instance attribute**: value of attribute depends on the object
    - **class attribute**: one value per class
  - Visibility
    - private, protected, package, public
  - Kind of value (type)
    - primitive values (int, double)
    - references to objects
  - Whether it is a constant attribute
    - in Java will be declared as final static

Designing for one class:
Identifying class behaviour

- **Objective**: identify and name all operations a class needs to provide/support
- **Initial heuristic**: Consult the requirements (problem description), look for verbs related to objects of the class of interest to discover the likely responsibilities of the class
- **Review**: check for problem specific methods needed to
  - **maintain** the state (attributes) of the object
  - perform **calculations** the class is responsible for
  - **monitor** what objects of the class are responsible for detecting and responding to
  - respond to **queries** that return information without modifying an object of the class
- It is often helpful to identify and go over some user scenarios to ensure as complete behaviour as possible is designed
Designing for one class:  
Designing each method

- For each method, need to distinguish:
  - Kind
    - instance methods are associated with an object
    - class methods are applied to a class and are independent of any object
      - declared as static and can only access static attributes (not instance attributes)
  - Visibility
    - private, protected, package, public
  - Signature (= method name + parameter types)
    - (a class cannot have two methods with the same signature)

- Notes…
  - final methods cannot be overridden in any subclass
  - overloaded method = method name with more than one signature

Designing for one class:  
Additional class design guidelines

- Ensure each class has
  - a “good”—useful for clients—set of constructors
  - appropriate accessor for certain attributes (getter methods)
  - appropriate mutator for some attributes (setter methods)
  - a destructor if necessary (in Java this is done by defining the finalize() method in the class; use very sparingly, if at all)
  - equality method — equals()
  - string representation method (good for debugging) — toString()

- May need to define methods for
  - cloning: for creating copies — clone() or copy constructor
  - hash code: returns an integer code that “represents” the object - hashCode()

- We’ll talk more about cloning, hashCode, etc., later in term. See “Effective Java” book by Joshua Bloch if interested in class design.