CPSC 211
Introduction to Software Development

Summer 2010

Term 1, Part 2

Instructor

- Name: Gabriel Murray
- Office: CS/ICICS 123
- Email: gabrielm@cs.ubc.ca
- Office Hours: see web page (or by appointment)
- Research interests: Natural Language Processing, Artificial Intelligence, Machine Learning, Semantic Web, Ontologies, Linguistics

Course Objectives

- When you complete this course, you will be able to:
  - move from personal software development methodologies to professional standards and practices
    - design software following standard principles and formalisms
    - create programs that interact with their environment (files etc.) and human users according to standard professional norms
    - develop effective software testing skills
  - given an API, write code that conforms to the API to perform a given task
  - identify and evaluate trade-offs in design and implementation decisions for systems of an intermediate size
  - read and write programs in Java using advanced features
    - collections, exceptions, etc.
  - extend your mental model of computation from that developed in CPSC111
    - recursion, concurrency, etc.
  - work with an existing codebase, including reading and understanding given code, and augment its functionality [in assignments]

Components & Evaluation

- Your grade in this course will be based on the following activities:
  - lab participation (5%)
  - in-class exercises/participation (5%)
  - four assignments (25%)
  - a midterm examination (20%)
  - a final examination (45%)
- To pass this course, you must obtain a 50% overall mark and, in addition, you must:
  - **pass the assignments AND**
  - **pass the final examination.**
- **Students who fail the assignments or the final exam will be assigned, as final grade in the course, the minimum of 45% and the grade computed using the above formula.** The instructors reserve the right to modify the course grading scheme at any time

Administration

- Main web sites for the course:
  http://www.cs.ubc.ca/~gabrielm/211/
  - contains most course material (notes, labs, assignments, etc.)
- Vista site for the course
  - contains bulletin board and grades
- Carefully read the course information at
  http://www.cs.ubc.ca/~gabrielm/211/courseInformation.html
- Labs start Wednesday

Labs

- Mondays & Wednesdays
- Starting this Wednesday, the 23rd
- Room ICICS 005
- You registered for a lab section when you registered for the course
- Do the pre-reading and pre-exercise **before** attending the lab

Lab 1

- Introduction to Unix and Eclipse

Assignments

- There are 4 assignments in 6 weeks
- Assignment 1 is being released within the next day
- Check Vista
- Music library application

Exams

- The final exam will be on the last day of class, normal time and location
- Midterm date to be announced soon

Getting Help

- Vista
  - Fastest way to get a quick response to a Java question
- Learning Centre (hours posted soon)
- Email TAs directly (check webpage)
- Email me directly (gabrielm@cs.ubc.ca)
- Labs

Summer Term

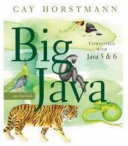
- Intense six-week course
- When one assignment is due, the next is immediately released
- Two exams
- Labs twice a week
- We strongly suggest you **do not** try to take two summer courses simultaneously

Typical Class Structure

- 9:00-9:10 any business, misc.
- 9:10-9:45 lecture
- 9:45-10:15 in-class exercise
- 10:15-10:30 tea break
- 10:30-11:00 lecture
- 11:00-11:30 in-class exercise

Textbook

- Big Java, Cay Horstmann
  - 3rd E
  - 2nd E okay



Intro. to Software Development

- Thinking back to CPSC 111...

Review: Classes, Objects, References

- A typical object oriented program consists of
  - a set of class definitions
  - a set of objects that interact with each other
- Class *methods* define the object's behaviour (i.e. what an object can do)
- *References* provide a way to distinguish and access the objects
  - a reference holds the address of an object
- Computation is performed by applying methods to objects

Review: Example

```
public class Account
{
   private Customer owner;
   private double balance;
   public Account( )  { balance = 0; }
   public void setOwner(Customer c)
   { ... }
   ....
}
public class Customer
{
   private String name;
   public Customer() { ... }
   public setName(String n) { ... }
   .....
}
```

```
public class AccountTest
{
   Account janeAcc =
            new Account();
   Customer jane =
            new Customer();

   jane.setName("Jane
   Black")
   janeAcc.setOwner(jane);
   janeAcc.deposit( 100.00);
   .....
}
```

Find the classes, objects and references shown on this page

Review: Example

```
public class Account
{
   private Customer owner;
   private double balance;
   public Account( )  { balance = 0; }
   public void setOwner(Customer c)
   { ... }
   ....
}
public class Customer
{
   private String name;
   public Customer() { ... }
   public setName(String n) { ... }
   .....
}
```

```
public class AccountTest
{
   Account janeAcc =
            new Account();
   Customer jane =
            new Customer();

   jane.setName("Jane
   Black")
   janeAcc.setOwner(jane);
   janeAcc.deposit(100.00);
   .....
}
```

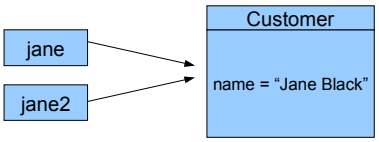
What are the methods? Which are the constructors? What are the instance fields?

Object reference

- A variable like jane *does not contain the object*
- Rather, it refers to the object's memory location
- You can have two object variables refer to the same object, e.g.

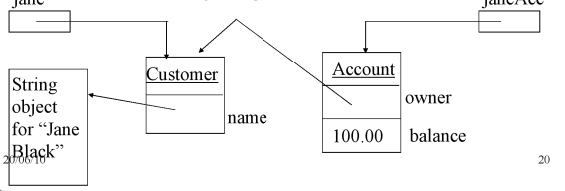
```
Customer jane2 = jane;
```


Object reference



Review: Memory Diagrams

- Show how objects and references are stored in the computer
- Show the relationship between them.
- Are informal and used for pedagogy
- Example: A memory diagram for Jane, and her account:



Review Question 1

How many b's will this code print to the screen?

```
for (int i = 1; i <= 5; i++)
  for (int j = 0; j < 4; j=j+2)
     System.out.println("b");
```


Review Question 1

How many b's will this code print to the screen?

```
for (int i = 1; i <= 5; i++)
  for (int j = 0; j < 4; j=j+2)
     System.out.println("b");
```

What are each of these components of the for-loop doing?

Review Question 1

How many b's will this code print to the screen?

```
for (int i = 1; i <= 5; i++)
  for (int j = 0; j < 4; j=j+2)
     System.out.println("b");
```

Recall that these are the same:

i++;
i = i+1;

Review Question 2

What does the following code print to the screen?

```
int a = 4;
if (a < 4)
   if (a < 1)
     System.out.println("good");
   else
     System.out.println("bad");
```

Review Question 2

What does the following code print to the screen?

```
int a = 4;
if (a < 4)
   if (a < 1)
     System.out.println("good");
   else
     System.out.println("bad");
```

The last else belongs to the last if

Review Question 3

Assume that Cat and Dog are subclasses of Mammal. Which of the following statements are valid?

```
a) Cat montana = new Cat();
b) Cat tuxedo = new Mammal();
c) Cat silas = new Dog();
d) Mammal animal = new Cat();
e) Mammal fluffyAnimal = new Dog();
f) animal = montana;
g) montana = fluffyAnimal;
```


Review Question 4

Consider the Counter class on the right. What is printed out by the following code?

```
Counter c1 = new Counter();
Counter c2 = new Counter();
c1.addOne();
c2.subtractOne();
c2.addOne();
c1.addOne();
System.out.println(
          c1.getCount());
System.out.println(
          c2.getCount());
```

What is we remove "static" from the declaration of count?

```
public class Counter {

   private static int count = 0;

   public void addOne() {
    count++;
   }
   public void subtractOne() {
    count--;
   }
   public int getCount() {
    return count;
   }
}
```


static

- Remember that a static instance field does not belong to any particular object
- If a static instance field is declared in a class definition, all objects of that class share one copy of the instance field
- In contrast, when we remove static then each object has its own copy

In-Class Exercise I

- Write a static method `sumArray` that takes an array of ints as its only parameter and returns the sum of the values in the array.
- For example, if `sampleArray` was defined as

```
int[] sampleArray = {2, 3, 2};
```

and passed as a parameter, the method would return 7.

In-Class Exercise I

Recall that the for-each loop (or enhanced for loop) is useful when you just want to iterate through a sequence of elements.

for (Type variable: collection)
  statement

Tea break!

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 edition
Chapter: 8,
Sections: 8.1-8.4, 8.6-8.9

Some ideas come from:

- *"Practical Object-Oriented Development with UML and Java" R. Lee, W. Tepfenhart, Prentice Hall, 2002.*
- *"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.

- Mitchell Kapor, A Software Design Manifesto (1991)

A concrete example…

Technology?

Human purpose?

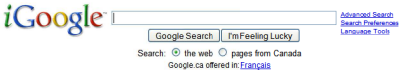
Design

A software example…

Many different aspects of design

Software program

Architecture



Human-computer interface

Software design

Software program

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, focus is on how the system will work, not on 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…
- Let's identify some classes…

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 **accessors** for certain attributes (getter methods)
  - appropriate **mutators** 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.*

Designing for one class: Additional class design guidelines… Minimize side effects

- A side effect of a method is any modification that is observable outside the method
- Some side effects are necessary; some are acceptable; others are wrong
- Some guidelines:
  - Accessor methods should not have any side effects
  - Mutator methods should change only the implicit argument
  - Avoid designing methods that change their explicit arguments, if it is possible
  - Avoid designing methods that change another object i.e. in class Account:
    - bad design: method printBalance that prints balance on System.out
    - good design: method getBalance that returns balance

bad:couples Account w/ System and PrintStream

Bank account example

- Problem Description
  - The bank wants a software system to maintain customer accounts. Each account belongs to a single customer and is identified by a unique id assigned by the bank. The owner and the id of an account can never change. A customer is identified by their name and can open an account, deposit and withdraw money into it and check the account balance, which must never be negative.

  . . .
- Suppose we design a class Account to represents a single account. What would be the attributes (data components) for the Account class?
- Would be correct to add the customer address and phone number as components to Account class?

Bank account example

- What should be the operations?

Representing class design: UML

- When designing software, we need to focus on how the design works, not all of the details of expressing the design in a programming language
- Software developers sometimes use UML (Unified Modelling Language) to express a design
- UML's graphical modelling notation lets developers focus on
  - classes and their important attributes and methods
  - relationships between classes

And to see that information in a condensed form

- UML has many different diagram types, we'll consider only class diagrams in this course

Representing a class in UML class diagram

- Use a rectangle with 3 compartments showing
  - the class name
  - the class data components (or attributes or data fields)
  - the class methods
- Example:

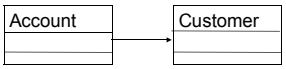
Account
- owner: Customer - balance: double
+ Account(s : Item) : void + setOwner(c : Customer) : void

…

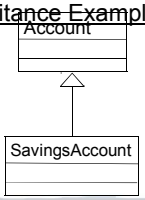
Representing class relationships

- Relationships are shown by arrows
- We'll consider just two types of relationship (for now):
  - Association : one class contains one or more references to another class
  - Inheritance : one class extends another class

Association Example



Inheritance Example



Is this enough?

- We have seen how to
  - identify attributes for a class
  - identify methods (the behavior) of a class
- We need a way to specify the behavior of each method
  - specification must be independent of programming language
  - must balance between
    - the important aspects that need to be captured by any implementation
    - give an implementor the freedom to decide on the rest
- Next class we'll discuss **class contracts** help specify method behaviour

In-Class Exercise II

- Given the following project description, identify the classes, attributes and methods

Review learning goals

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 edition
Chapter: 8,
Sections: 8.1-8.4, 8.6-8.9

Some ideas come from:

- *"Practical Object-Oriented Development with UML and Java" R. Lee, W. Tepfenhart, Prentice Hall, 2002.*
- *"Object-Oriented Software Development Using Java", Xiaoping Jia, Addison Wesley, 2002*