Department of Computer Science

Undergraduate Events

Events this week

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

**Industry Panel**
- **Speakers:** Managers from IBM, Microsoft, SAP, TELUS, Radical…
- **Date:** Tues. Jan 12
- **Time:** Panel: 5:15 – 6:15 pm
  Networking: 6:15 – 7:15 pm
- **Location:** DMP 110 for panel,
  X-wing ugrad lounge for networking

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

**Google Tech Talk**
- **Date:** Wed, Jan 13
- **Time:** 4 – 5 pm
- **Location:** DMP 110

**IBM Info Session**
- **Date:** Wed., Jan 13
- **Time:** 5:30 – 7 pm
- **Location:** Wesbrook 100

**Administrivia**

- Lecture slides (day by day) are on the web:
- Reminder: Labs start Monday (January 11th)
- Reminder: 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.
Where are we?

Designing a single class

• Think about attributes
  • Adjectives and possessive phrases in the specification
• Think about methods
  • Verbs in the specification
• Round out the class interface
  • Constructors, clone(), equals(), toString()

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 (this)
  • Avoid designing methods that change their explicit arguments, if it is possible
  • Avoid designing methods that change another object
    • o bad design: method printName() that prints name on System.out
    • o good design: method getName() that returns name
**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 and by a descriptive name. 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 represent a single account. What would be the attributes (data components) for the Account class?
  - id: int
  - name: String
  - balance: double
  - owner: Customer

- Would be correct to add the customer address and phone number as components to Account class? No.

**Bank account example**

- What should the operations be?
  - getBalance(): double
  - deposit(double): double
  - withdraw(double): double
  - Account(Customer, String, double, int)
  - getId(): int
  - getName(): String
  - transfer(double, Account)
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:

<table>
<thead>
<tr>
<th>Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>- owner: Customer</td>
</tr>
<tr>
<td>- name : String</td>
</tr>
<tr>
<td>- balance: double</td>
</tr>
<tr>
<td>+ Account(n : String) : void</td>
</tr>
<tr>
<td>+ setOwner(c : Customer) : void</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Representing class relationships

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

**Association Example**

```
Account
---------
|        |
|        |
|        |

Customer
---------
|        |
|        |
|        |
```

**Inheritance Example**

```
Account
--------
|        |
|        |
|        |

SavingsAccount
--------------
|              |
```

Is this enough?

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