CS210
Intermediate Computing with Data Structures (Java)
Bob Wilson
bobw@cs.umb.edu
S-3-176
Prerequisites

• We expect that you already know the Java programming language at an introductory level (usually gained by taking our CS110 or CS 114L/115L)

• You should be capable of writing a program with at least two Java classes from scratch

• If you can not, you should NOT take this course. See your advisor for other options such as retaking CS110 or taking CS119
Welcome to CS210

• Textbook is Lewis and Chase, *Java Software Structures, 4th Ed.*

• The course syllabus is on CS210 website:
  http://www.cs.umb.edu/~bobw/CS210/index.html

• Go to the UNIX/PC lab (Science Bldg 3rd floor)
  – “Apply” for a UNIX/PC account and/or add CS210
  – You need that account to upload your project files

• We’ll go through the syllabus now
Course Procedures

• Two lecture sessions each week
  – Held in this classroom
  – Slides and demonstrations, but no hands-on

• I recommend that you print copies of my lecture notes and bring them to each lecture

• One lab session each week
  – In one of the Healey Library General Use Labs
  – Hands-on work associated with the lectures
  – Lab report must be turned in the following week
Homework Assignments

• Reading for this class: L&C 1.1, App A
• Reading for next class: L&C App B
  – I will briefly review this material next class
  – I assume that you are already familiar with the Java programming language
  – You are responsible for it in homework or exams
  – If you are not familiar with it, please see me!
• Lab 1: Read assignment for next week
Software Development Methods

- Classical “Waterfall” Development Steps

1. Edit and save source code
2. Build source code to create program
3. Run program and evaluate results

Errors may occur at any step and need to be corrected before proceeding.
Errors

• A program can have three types of errors:

• The IDE editor and/or compiler will find syntax errors and other basic problems (*compile-time errors*)
  
  – If compile-time errors exist, an executable version of the program is not created

• A problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (*run-time errors*)

• A program may run, but produce incorrect results, perhaps using an incorrect formula (*logical errors*)
Unified Modeling Language (UML)

• UML is a graphical tool to visualize and analyze the requirements and do design of an object-oriented solution to a problem

• Three basic types of diagrams:
  – Use Case Diagram
  – Class Diagram
  – Interaction Diagram

• A good reference is *UML Distilled, 3rd Ed.*, Martin Fowler, Addison-Wesley/Pearson
Unified Modeling Language (UML)

• Advantage of UML – It is graphical
  – Allows you to visualize the problem / solution
  – Organizes your detailed information

• Disadvantage of UML – It is graphical
  – Can be done with pencil and paper – tedious!
  – Commercial UML S/W tools are expensive!
    • Example: Rational ROSE (IBM acquired Rational)
  – We have UMLPAD which is a simple design tool to aid in drawing UML diagrams
Use Case Diagrams

• Typically the first diagrams drawn
• Helpful for visualizing the requirements
• Icons on the Use Case Diagram
  – Actors: Users or other external systems
  – Objects: Potential classes in the solution
  – Scenarios: Sequences of interactions between Actors and Objects that are typical for the solution to the problem (Both success cases and error cases should be included)
Example: Use Case Diagram

- **Customer**
  - **Buy with Cash**
  - **Buy On Credit**
- **Sales Person**
  - **Process Cash Sale**
  - **Process Credit Sale**
- **Bartender**
  - **Cry over lost commission**
  - **Stiff Drink**
  - **Cash Register**
  - **Bank**
  - **Card Swipe Machine**
- **Product**
- **“No Sale”**
  - <<includes>>
Class Diagrams

• Classify the Objects in the Use Cases
• Define name of each class
• Define each class’s attributes
  – Constants
  – Variables
• Define each class’s behaviors
  – Methods
• Show relationships between classes
  – Depends on, Inherits, etc.
Example: Class Diagram

**Credit Card**
- myCardData : CardData

+ read( ) : CardData

**CardData**
- cardType : enum {Visa, …}
- myBank : Bank
- myAcctNumber : int

+ getCardType( ) : enum
+ getBank( ) : Bank
+ getAcctNumber( ) : int

**Bank**
- name : String
- address : String
- accounts [ ] : AcctData

+ processCharge (thisCardData : CardData
amt : double,
storeName : String,
storeBank : Bank,
storeAcctNumber : int
) : boolean

**AcctData**
- acctLimit : double
- acctBalance : double
...
Interaction Diagrams

• Shows the time relationship of the events in a scenario between actors and objects
  – UML Sequence Diagram
  – Sometimes called a “ladder diagram”
• A vertical line represents an actor or object
• A horizontal line represents an interaction
  – E.G. a call to a method of another object
• Progress of time is shown down the page
Example: Interaction Diagram

Process Credit Sale

Time

Sales Person

Card Swipe Machine

Credit Card

Bank

swipeCard( )

enterAmt(…)

readResponse( )

read( )

return CardData

return boolean

processCharge( … )

return “OK”
Software Development Tools

- Using Sun Java SDK alone

![Diagram showing the development process using Sun Java SDK](Image)
Using Sun Java SDK Alone

- Example DOS Commands and Parameters
  C:\> edit HelloWorld.java
  (Create/edit “source file” in an external window)
  C:\> javac HelloWorld.java (creates .class file)
  C:\> java -classpath … HelloWorld
  Hello World
  C:\> exit
Software Development Tools

- We will use a combination of the Dr Java IDE and the Sun Java SDK
On-line Demonstration: Dr Java
Software Development Tools

• Download/install the software development tools on your own PC if you wish to do your project assignments at home
  – Sun Software Development Kit (SDK)
  – Dr Java Integrated Development Environment (IDE)

• Use the PCs in the Healey Library labs as they already have these tools installed