Announcements

- Adds
  - Enrolled in lab ⇒ Permission number for class
  - Ordinary student and prereqs ⇒ 8 seats
  - Open University and prereqs ⇒ Next Monday (if seats - very likely)

- Syllabus
  - First Project (trivial) due Friday by 8pm
  - First Pop Quiz (JAVA) soon
  - How to be successful
  - Questions

Comments

- Must pass 2 (3) lab exams to pass course
- Lowest quiz dropped
- Compute percentage for each type
  (Proj, Quiz, Lab Exam, Midterm, Final)
- Class% = Proj% * 20 + Quiz% * 10 + LabExam% * 10 + Midterm%*30 + Final% * 30
- Lates: Max 1 week - Max 1 project
- Read web page from time to time
  www.csun.edu/~jnoga/comp182
- WebCT

Suggested Purchases

- 5ish Scantrons Form 882
- Data Structures Reference (one fine)
  - Data Structures & Algorithms
    Goodrich & Tamassia
  - Data Abstraction & Problem Solving
    Carrano & Pritchard
  - Data Structures & Abstractions
    Carrano & Savitch
- Java Reference (one fine)
  - Intro to Java Programming - Liang
  - Thinking in Java - Eckel
  - java.sun.com
Java Classes

- Basic building block of Java programs
- Template for an object
- Consists of variables & methods
- Variables (normally should be private)
- Constructor (instantiates object)
- Accessors (get private/protected fields)
- Mutators (change private/protected fields)
- public static void main(String args[])
- Normal Methods (public/private)

Example: Basketball Player

```java
public class MyBasketBallPlayer {
    private char position;
    private byte uniformNumber;
    private short seasonPoints;
    private int lifetimePoints;
    private long heightInPicoMeters;
    private boolean injured;
    private float yearSalary;
    private double lifetimeSalary;
    private String name;
    private static int ptsForFreeThrow;
}
```

Constructor

```java
public MyBasketBallPlayer(char p, byte num) {
    position = p;
    uniformNumber = num;
    seasonPoints = 0;
    lifetimePoints = 0;
    heightInPicoMeters = -1;
    injured = false;
    yearSalary = 0f;
    lifetimeSalary = 0;
}
```

Accessors & Mutators

```java
public boolean getInjuryStatus() {
    return injured;
}
public void setInjured() {
    injured = true;
}
public void setSalaryIncrease(float raise) {
    yearSalary += raise;
    lifetimeSalary += raise;
}
```
Main Method

```java
javac MyBasketBallPlayer.java; java MyBasketBallPlayer
public static void main(String args[]) {
    MyBasketBallPlayer dufus;
    dufus = new MyBasketBallPlayer('C', 3);
    System.out.println("I want more money!");
    dufus.setMorePoints(dufus.shoot(2));
}
```

Normal Methods

```java
private int shoot(int kind) {
    double luck = Math.random();
    if (luck > .7 && kind == 3)
        return kind;
    if (luck > .6 && kind == 2)
        return kind;
    if (luck > .4 && kind == 1)
        return kind;
    return 0;
}
```

Loops

```java
public void exercise(int num) {
    for(int i=0; i < num; i++)
        System.out.println("Running a lap.");
}

public void exercise2(int num) {
    while(num > 0) {
        System.out.println("Stair Climbing.");
        num--;
    }
}
```

Loops Continued

```java
public void exercise3(int num) {
    do {
        System.out.println("Hid from coach.");
        System.out.println("Stair Climbing.");
        num--;
    } while (num>0);
}
```
import java.util.Scanner;

public class MyAlarmClock {
    private int hour, minute;
    private int alarmHour, alarmMinute;
    private boolean alarmOn, am, alarmAM;

    public MyAlarmClock() {
        hour = 12; minute = 0;
        alarmHour = 12; alarmMinute = 0;
        am = true;
        alarmAM = true;
        alarmOn = false;
    }

    public MyAlarmClock(int hr, int mn) {
        if (hr<1 || hr>12 || mn<0 || mn>=60) {
            hr = 12; mn = 0;
        } else {
            hour = hr; minute = mn;
        }
        alarmHour = 12; alarmMinute = 0;
        alarmOn = false; am = true;
        alarmAM = true;
    }

    public String getTime() {
        String spacer= "";
        if (minute < 10)
            spacer = "0";
        if (am)
            return hour + ":" + spacer + minute + "am";
        return hour + ":" + spacer + minute + "pm";
    }

    public String getTime(String which) {
        String spacer="";
        if (minute < 10)
            spacer="0";
        if (which.equalsIgnoreCase("military"))
            if(am)
                return hour%12 + ":" + spacer + minute;
        else
            return (hour%12)+12 + ":" + spacer+minute;
        return getTime();
    }
}
public void advanceHr(int number) {
    if (number < 0) return;
    hour = (hour + number) % 24;
    if (hour == 0)
        hour = 24;
    if (hour > 12) {
        hour -= 12;
        am = !am;
    }
}

public void advanceMin(int number) {
    if (number < 0) return;
    minute = minute + number;
    advanceHr(minute/60);
    minute = minute % 60;
}

public static void main(String args[]) {
    MyAlarmClock clock1, clock2, clock3;
    clock1 = new MyAlarmClock(3,2);
    clock2 = new MyAlarmClock();
    clock3 = clock1;
    clock1.advanceHr(49);
    clock1.advanceMin(72);
    clock1.switchAlarmStatus();
    clock2.switchAlarmStatus();
    System.out.println("Cl 1 "+clock1.getTime());
    System.out.println("Cl 2 "+clock2.getTime("MiLiTaRy");
    System.out.println("Cl 3 "
        +clock3.getTime("mIlItArY");
}

public void switchAlarmStatus() {
    alarmOn = !alarmOn;
}

public boolean ringAlarm() {
    if (hour == alarmHour && minute == alarmMinute &&
        am == alarmAM && alarmOn) {
        System.out.println("Ring, Ring, Ring");
        return true;
    }
    return false;
}

System.out.println("Cl 1 "
    +clock1.getTime());
System.out.println("Cl 2 "
    +clock2.getTime("MiLiTaRy");
System.out.println("Cl 3 "
    +clock3.getTime("mIlItArY");
    System.out.println("Check alarm on clock 1");
    clock1.ringAlarm();
    System.out.println("Check alarm on clock 2");
    clock2.ringAlarm();
    System.out.println("Check alarm on clock 3");
    clock3.ringAlarm();
}
public void evaluationOfAlarmClock() {
    Scanner in = new Scanner(System.in);
    int i;
    String str;
    System.out.println("How good is my alarm clock?");
    str = in.next();
    if (str.equals("Fantastic!"))
        System.out.println("Right, it is fantastic!\n");
    else
        System.out.println("Right, it is great!\n");
    System.out.println("Buy how many?");
    i = in.nextInt();
    System.out.println("Wow, you want " + (i+100));
}

public void evalOfAlarmClock2 (String fileName) {
    Scanner fin = new Scanner(new File(fileName));
    String str;
    int fan = 0, gr = 0;
    do {
        str = fin.next();
        if(str.equals("Fantastic!"))
            fan++;
        else
            gr++;
    }
    while(str != null);
    System.out.println(fan+"Fantastic\n"+gr+"Great");
}

public void fakingAlarmEvals(String fileName) {
    PrintWriter fout;
    fout = new PrintWriter(new FileWriter(fileName));
    for(int i=0; i<100; i++) {
        fout.println("Great");
        fout.println("Fantastic");
    }
    fout.close();
}

public void splittingThingsUp(String manyWords) {
    String[] strings;
    strings = manyWords.split(" ");
    for (int i=0; i<strings.length; i++)
        System.out.println("Mr. " + strings[i]);
}

public void isNumber (String unknown) {
    int i=0;
    try {
        i=Integer.parseInt(unknown);
    } catch (NumberFormatException e) {
        System.out.println("That’s not an integer");
        return;
    }
    System.out.println("Now this is an integer: "+i);
}
What is an abstract data type?

- Collection of info and operations on that info
- High level (mathematical?) definition
- Not tied to actual implementation

Pseudo-example: Grocery List
- Items are (instances of) food
- Operations
  - Add to list
  - Remove from list
  - Get ith item
  - Show list

What is a data structure?

- Data = Information
- Structure = Pattern of Organization
- Data Structure = Organized Information
- Actual implementation of an ADT

Pseudo-example: Grocery list actually implemented
- Yellow paper
- Special notebook
- Pencil
- Pen
- Bottom / erase / scan
- Top / scratch out / numbered

(absolute of terminology)

Why study data structures?

- Appear frequently in diverse applications
- Proper choice unbelievably better performance
- Quickly see “standard” answer
- Know if standard answer is insufficient
- Prepares a way of thinking

Course Goals

- For a given data structure
  - Basic Understanding of Idea
  - Implement in Java
  - Analyze & Measure performance
  - Understand trade-offs
- Strengthen programming ability
- Sorting/Searching
Basic Piece of Data / Record

```java
public class OurThing {
    private String name;
    private int key, priority;
    public OurThing(String nm, int ky, int pr) {
        name = nm;
        key = ky;
        priority = pr;
    }
    public String getName() {
        return name;
    }
    public void setName(String nm) {
        name = nm;
    }
}
```

Interfaces

Quick easy way to describe what is needed/required. Basically just a set of signature lines. Class that implements must have every method listed. Implementing (2 related meanings)

```java
public interface MyDS{
    public boolean insert(OurThing ot);
    public boolean delete(int ky);
    public OurThing find(int ky);
    public String toString();
}
```

“Real” example of an ADT - LIST

- Info are any object
- Operations: insert at position, delete by position, find, is empty, length, display

```java
public interface List {
    public boolean insert(OurThing ot, int pos);
    public boolean delete(int pos);
    public int find(OurThing ot);
    public boolean isEmpty();
    public int length();
    public String toString();
}
```

Arrays (simplest data structure)

- Linear set of “cells” each can hold one object
- Each cell has an index with “random access”
- The number of cells is fixed

```
0 1 2 3 ...... capacity−1
```

Supports fast access by position.
Partial implementation using an array of OurThings

```java
public class OurArray implements List{
    private OurThing[] theThings;
    private int howMany, capacity = 10;
    public OurArray() {
        theThings= new OurThing[capacity];
        howMany=0;
    }
    public boolean insert(OurThing ot, int pos) {
        if (howMany >= capacity) return false;
        for (int i = howMany; i>pos; i--)
            theThings[i] = theThings[i-1];
        theThings[pos] = ot;
        howMany++;
        return true;
    }
    public boolean delete(int pos) {
        if (pos < 0 || pos > howMany-1)
            return false;
        theThings[pos] = theThings[howMany-1];
        howMany--;
        return true;
    }
    public int find(OurThing ot) {
        for (int i = 0 ; i < howMany; i++)
            if (ot == theThings[i]) //==, .equals, field
                return i;
        return -1;
    }
    public boolean isEmpty(){...}
    public int length(){...}
    public String toString(){...}
}
```

Array Continued

```java
public int find(OurThing ot) {
    for (int i = 0 ; i < howMany; i++)
        if (ot == theThings[i]) //==, .equals, field
            return i;
    return -1;
}
public boolean isEmpty(){...}
public int length(){...}
public String toString(){...}
```

Steal from End / Shift Over

```java
public boolean delete(int pos) {
    if (pos < 0 || pos > howMany-1)
        return false;
    theThings[pos] = theThings[howMany-1];
    howMany--;
    return true;
}
```

Analysis (thinking)

Which of these delete methods is better? Depends?
- Always deleting last - no real difference
- Always deleting first - shifting much slower
- Keeping order important - shift is better

What is good/bad about arrays?
- Simple
- Fast (if we know the position)
- Fixed size (can fill up)
- Slow (if we don’t know position)
- Anything else?
Wouldn’t it be nice if arrays could grow?

**Vectors (dynamically allocated array)**

**Terminology**
- size is the number of cells in use
- capacity is the current number of cells in the actual array

**Idea:**
- If not full, works same
- If full, modify insert
  - Allocate larger array
  - Copy items into larger array
  - Reset array reference
  - Adjust capacity

---

**Partial Implementation**

```java
public class OurVector implements List {
    private OurThing[] theThings;
    private int size, capacity;
    public OurVector() {
        theThings = new OurThing[10];
        size = 0; capacity = 10;
    }
    public boolean insert(OurThing ot, int pos) {
        if (size == capacity) {
            OurThing tmp[] = new OurThing[capacity+10];
            for (int i=0; i<capacity; i++)
                tmp[i] = theThings[i];
            theThings = tmp;
            capacity = capacity +10;
        } // now there is room - do just like before
    }
}
```

---

**Ordered LIST ADT**

Just like LIST ADT except items in “increasing” order

Notice that insert, delete, find can be done using bisection search

```java
public OurThing find(int ky) {
    int from = 0, to=n-1;
    while (from <= to) {
        if (theThings[(from+to)/2].getKey() == ky)
            return theThings[(from+to)/2];
        if (theThings[(from+to)/2].getKey() < ky)
            from = (from+to)/2 + 1;
        if (theThings[(from+to)/2].getKey() > ky)
            to = (from+to)/2 - 1;
    }
    return null;
}
```