Odds and Ends

- Wed night dinner (next Wed)
- Student research symposium,
  - Thur Sept 9, 4-6pm, Founders
CS 30: Introduction to Computer Science II

- Wayne Iba
- Email: iba@westmont.edu; Phone: x6799
- Office: New Math/CS Modular building
- Office hours: To be determined
- Web page: http://homepage.westmont.edu/iba
- Assignments due on Fridays, 4pm
Course Overview

• Required textbooks
  – *Classic Data Structures in Java*, Timothy Budd. Addison Wesley.
• Potentially Useful
  – *Thinking in Java*, Bruce Eckel
  – Any one of many Java tutorials
• Java development environment
  – BlueJ
  – Forte, Code Warrior, ...
Grading

- Programming assignments: 50%
- Class presentation: 10%
- Surprise quizzes: 10%
- Mid-term exam: 15%
- Final exam: 15%
Course Overview

- Introduction to Computer Science II
  - Data structures
  - Abstractions
  - Problem solving
  - Design
Goals & Expectations

- Have fun!
- Learn about Data Structures
  - Think abstractly about problem models
  - Similarly for problem processes
- Program design
  - Tackle any medium-sized programming problem
  - (design, documentation, implementation, testing)
- Workload
- Quality
Tentative Schedule

1. Intro to Java and managing complexity (ch 1)
2. Abstract data types (ch 2)
3. Algorithms (ch 3)
4. Run-time and correctness (chs 4 & 5)
5. Vectors (ch 6)
6. Sorting and linked-lists (chs 7 & 8)
7. Mid-term (Oct 14)
8. .....

Java Big Picture

• Everything is an object
• Objects organized in Classes
• Classes are related via inheritance
• More specialized classes (more functionality) are subclasses of more general classes
  – Most general class is java.lang.Object
  – Functionality created by new classes
• Programs consist of collection of classes/objects
Intro to Java

- Program structure:
  - Packages, interfaces, classes, and methods
- Statements
  - Declarations, assignments, control-flow
- Expressions
  - Value-bearing statements
Program Structure

- Packages
- Import declarations
- Class declarations
- Interface declarations
- Method declarations
- Constructors
- Data field declarations
Packages

- A means to bundle collection of related classes
  - Create library of classes
  - `public` keyword exposes specific classes
  - Unless otherwise indicated classes live in unnamed package

```
Package my.util;
...
...
public class PrintInt ...
...
```
Import Declarations

- Identify names from another package to be used here
  - import java.io.*;
- What is the analogous C++ construct?
Class declarations

- Each class lives in its own file with the same name as the name of the class (contrast w/ C++)
- Modifiers: public, abstract, and final
- Optional: extends and implements

```java
public class HelloWorld {
    public static void main (String [] args) {
        System.out.println( "Hello World!" );
    }
}

public class Dog extends Mammal { .... }

class Robin extends Bird implements FlyingObject { ... }
```
Interface Declarations

- Provides the sketch of a class (or set of classes)
- Use “interface” instead of “class” in declaration
- Provide stubs for intended methods but not details

```java
public interface Bag extends Collection {
    public void addElement(Object newElement);
    public boolean containsElement(Object testElement);
    public Object findElement(Object testElement);
    public void removeElement(Object oldElement);
}
```
Method Declarations

• A method implements a function or procedure within a class
  – What is the analogous C++ construct?

```java
Public class SpaceShip {
    private int xPos;
    private int yPos;
    private float xVel;
    private float yVel;
    ...
    public void updatePos(){xPos += xVel; yPos += yVel;}
    ...
}
```
Method Modifiers

- Public: method visible wherever instances live
- Protected: visible in package or subclasses
- Private: only within methods of same class
- Final: may not be overridden in sub-class
- Abstract: must be overridden
- Static: may be used without a class instance
- Synchronized: only one thread may use at a time
Constructors

- Special method that creates class instances
- No type specified (creates instance of Class type)

```java
public class SpaceShip{
    ....

    public SpaceShip(int x, int y){
        xPos = x; yPos = y;
        xVel = 0.0; yVel = 0.0;
    }
}
```
Data Field Declarations

- Declaring variable name: type and name
- Optional modifiers
  - Static data fields shared by all instances of class
- Optional initialization

```java
Public class SpaceShip {
    private static float gravity = -32.0;
    private float xVel = 0.0;
    private float yVel = 0.0;
    ...
    public void updatePos(){xPos += xVel; yPos += yVel;}
    ...
}
```
Statements

- Declaration statement
- Assignment statement
- Procedure call
- “if” statement
- “switch” statement
- “while” statement
- “for” statement
- “return” statement
- “throw” statement
- “try” statement
• **Declarations**
  - `public static final double pi = 3.14159;`
  - `SpaceShip myShip;`

• **Assignments**
  - `myShip = new SpaceShip();`
  - `x = 27;`

• **Procedure calls**
  - `myShip.updatePos();`
Conditional Statements

- The “if” and “switch” statements just like C++
  - Want to store the smallest of a and b into smallest
  - ....
  -
  - Want to print the word for a single-digit integer
  - ....
  -
Looping Statements

- (just like C++)
  - Want to update ship's position as long as not crashing
  - ....
  -
  - Want to add the numbers from 15 to 72
  - ....
  -
Odds and Ends

• Need “note-taker” -- call Michelle at x6159
• Fri 9/3, 3:30 Hieronymous
  - *Crisis and subjectivity: Doing fieldwork in baathist Iraq.*
• Wed night dinner (next Wed)
• [ladies only] Thurs evening Sept 9, with Patti Scofield
  - Kihlstroms, 7:30pm
  - [optional] bring suit for hot tub
• Student research symposium,
  - Thur Sept 9, 4-6pm, Founders
Other Statements

- The `return` statement signals termination of method and passes a value back to the caller (like C++).
Throw and Catch (try) Statements

- The **throw** statement signals an error (exception)
  - throw new ArithmeticException();
  - Control passes to nearest “catch” block
- The **try** statement lets the program catch exceptions

```java
try {
    URL address = new URL(args[0]);
    ...
} catch (MalformedURLException e) {
    System.out.println("URL exception “ + e);
} catch (IOException e) {
    System.out.println("I/O exeption “ + e);
}
```
Expressions

- Literals
- Variables
- Data field and method access
- Operators
- Object creation
- Arrays
• Literals: much like C++
  - 7, 3.14159, true, 'a', "a string"
  - null is the unassigned object value
  - Escape characters: \b, \n, \r, \t

• Variables: also
  - What is the variable?
  - private static boolean double;
  - Data fields accessed by simple naming
  - Pseudovariables
    • this
    • super
Data Field and Method Access

- Access public fields by:
  - `objectName.fieldName`
  - `myShip.xPos`
  - For static, may accessed by `ClassName.staticField`

- Methods similar but need parens
  - `myShip.updatePos()`;
  - `marmotPopulation.reproduce(0.7)`;
Operators, Objects, & Arrays

- Operators just like C++
- **Object creation using** new operator
  - Population batPop = new Population("bat", 50, localEnv);
- Arrays are contiguous memory of same-type values
  - **Declared**
  - Card [] myDeck;
  - **Allocated**
  - myDeck = new Card[52];
  - **Accessed**
  - myDeck[38].suit();
  - myDeck[38].rank();
Files

- Streams – flow of data (bytes)
- Java standard streams
  - System.out
  - System.in
  - System.err
- Varieties of streams
  - FileStreams
  - DataInputStream and ObjectInputStream
Managing Complexity

Chapter 1
Programming in “Small” vs “Large”

- Individual programs
  - Entire system can be understood by one mind
  - Main problem is designing/implementing algorithms

- Team projects
  - System too large for anyone to understand
  - Different people involved at different phases
  - Main problem is management
    - interacting details of system behavior
    - Communication (both human and computer)

- Object-Oriented Programming to the rescue
Control of Complexity

- What techniques are available to manage complexity
  - Abstraction
  - Division into parts
  - Composition
  - Layers of specialization
  - Multiple views
  - Patterns
Abstraction, Information Hiding and Layering

- Variable inclusion/exclusion of details
- Multiple but inter-connected (and confused) terms
- Abstraction
  - Ignoring some details in order to focus on others
  - *How does an automobile work?*
- Information hiding
  - Some details irrelevant to a situation
  - Different levels of detailed understanding
- Layering
  - Apply *divide-and-conquer* to complexity
  - *Consider what you know about cars wrt transports*
Division into Parts

- Encapsulation and interchangeability
  - Divide large project into *independent* sub-projects
  - *Encapsulate* details of each sub-project
  - Enables interchange variations to overall project

- Interface and Implementation
  - Encapsulation allows distinguish
  - The *what* vs the *how*
Division into Parts

• The Service View
  – Think of services provided by components in a large system of many various components
  – The services describe what the component does

• Repetition and Recursion
  – Break into parts where parts are nearly identical
  – Summing the number from 25 to 73 with for loop
  – Inductive proof: sum of number from 1 to n is n \( \frac{(n+1)}{2} \)
Recursive Methods

- Inductive proof method works for *recursive methods*
  - Base case
  - Inductive (or recursive) case
- Print multi-digit integer one digit at a time

```java
public void printInt(OutputStream out, int val) {
    // pre: val is greater than or equal to zero
    // post: will print textual representation of val on out
    if (n < 10)
        out.write(digitChar(val)); // base case, print one digit
    else {
        printInt(n/10); // recursive call, print all but last digit
        out.write(digitChar(n%10) // then print last digit
    }
}
```
Updated Version

- Variable “val” -> “n”
- Add paren and semicolon
- Address possible IOException

```java
public void printInt(OutputStream out, int n) throws IOException {
    // pre: val is greater than or equal to zero
    // post: will print textual representation of val on out
    if (n < 10)
        out.write(digitChar(n)); // base case, print one digit
    else {
        printInt(out, n/10); // recursive call, print all but last
        out.write(digitChar(n%10)); // then print last digit
    }
}
```
Composition

- Equally powerful to divide-and-conquer
- Given few simple parts
  - Compose them into more complex part
  - Which in turn can be composed into still more....
- Regular expressions
  - Concatenation (“composition”)
  - Disjunction (“alternation”)
  - Kleene star
    - zero or more concatenations of given regular expression
- Compositional value is ability to mix all of these
- Part-of or “has-a” relations
Layers of Specialization

- Taxonomies (e.g., animals, mammals, reptiles, ....)
- *Is-a* relations
- Exceptions
  - Generalizations always have exceptions
  - Phyl the platypus
- In perspective
  - Levels of descriptive abstraction for the same thing
  - Part-of decomposition/composition
  - *Is-a* class hierarchies
Patterns

- Avoid reinventing the wheel
- Solve new problems based on previous solutions to similar problems
  - *Case-based Reasoning*
- Abstract similar cases into *pattern*
- Pattern example: Adapter
  - Client needs service in particular format
  - Service-provider doesn't support format
  - Adapter supports client's required interface
Exercise 1.4

- Prove, by mathematical induction, that the sum of powers of 2 is one less than the next-higher power. That is, for any nonnegative integer \( n \):

\[
\sum_{i=0}^{n} 2^i = 2^{n+1} - 1
\]