Objectives of CS 230

- Teach main ideas of programming
  - Data abstraction
  - Modularity
  - Performance analysis
  - Basic abstract data types (ADTs)
- Make you a more competent programmer...
  - designer, tester, analyzer, debugger
  - team member
- Help you develop a project worth showing off
- Have fun in the process

Why ADTs?

- Allow you to write complex programs easier
  - To keep mental track of complex data interaction
  - To reuse code (yes!)
  - To improve code performance
- Allows modularity of large projects
  - Easier to understand large chunks of code
  - Easier to collaborate with large teams
- To combat Sergeant Spaghetti Code

Some basic ADTs:

- Collections
- Linked list
- Stack
- Queue
- Table
- Priority queue

Not so basic:

- Tree
- Set
- Graph

Java “portability”
1.5 – Object-Oriented Programming

- Java is an object-oriented programming language
- An object is a fundamental entity in a Java program
- Objects can be used to represent real-world entities
- For instance, a Photo Object might represent a particular photo in a photo management program (e.g., iPhoto)
- Each Photo Object handles the data (pixel files) processing methods (e.g., red eye reduction) and data management (e.g., copy photo) related to that photograph

1.5 – Classes define Objects

- A class is the blueprint of an object
- The class uses methods to define the behaviors of the object
- The class that contains the main method of a Java program represents the entire program
- A class represents a concept, and an object represents the embodiment of that concept
- Multiple objects can be created from the same class

1.5 – Objects and Classes

A class (the concept)

- Bank Account

An object (the realization)

- John’s Bank Account
  - Balance: $5,257
- Jason’s Bank Account
  - Balance: $1,245,069
- Mary’s Bank Account
  - Balance: $16,833

Multiple objects from the same class

1.5 – Inheritance

- One class can be used to derive another via inheritance
- Classes can be organized into hierarchies
3.1 – Objects...

- Generally, we use the `new` operator to create an object.

```java
name1 = new String("Steve Jobs");
```

This calls the String constructor, which is a special method that sets up the object.

Creating an object is called **instantiation**.

An object is an **instance** of a particular class.

3.1 – Understanding the Assignment

<table>
<thead>
<tr>
<th>Before:</th>
<th>After:</th>
</tr>
</thead>
<tbody>
<tr>
<td>num1 = 38</td>
<td>num1 = 38</td>
</tr>
<tr>
<td>num2 = 96</td>
<td>num2 = 38</td>
</tr>
</tbody>
</table>

- **For primitive types**, assignment takes a copy of a value and stores it in a variable.
- **For object references**, assignment copies the address.

```
name2 = name1;
```

Two or more references that refer to the same object are called **aliases**.

Alises can be useful, but dangerous: should be managed **carefully**.

Changing an object through one reference changes it for all of its aliases, because there is really only one object (but this cannot happen on immutable Strings).

3.2 – StringChange.java

```java
//****************************************************************
//  StringChange.java
//  Demonstrates the use of the String class and its methods.
//****************************************************************
public class StringChange {
    public static void main (String[] args) {
        String phrase = "Hi, I love you not ";
        String m1 = phrase.concat ("...");
        String m2 = m1.toUpperCase();
        String m3 = m2.replace('I', 'U');
        String m4 = m3.substring(6, 10);
        System.out.println (m4 + " is the answer");
        System.out.println ("Changed length: "+ m4.length());
    }
}
```
3.3 – Class Libraries

- A class library is a collection of classes that we can use when developing programs.
- The Java standard class library is part of any Java development environment:
  http://docs.oracle.com/javase/6/docs/api/
- Its classes are not part of the Java language per se, but we rely on them heavily.
- Various classes we've already seen (System, Scanner, String) are part of the Java standard class library.
- Other class libraries can be obtained through third party vendors, or you can create your own!

3.3 – Packages

- The classes of the Java standard class library are organized into packages.
- Some of the packages in the standard class library are:
  - java.lang: General support
  - java.applet: Creating applets for the web
  - java.awt: Graphics and graphical user interfaces
  - javax.swing: Additional graphics capabilities
  - java.net: Network communication
  - java.util: Utilities

3.3 – The import Declaration

- To use a class from a package, you could use its fully qualified name:
  `java.util.Scanner scan = new java.util.Scanner (System.in);`
- Or you can import the class, `import java.util.Scanner;` and then use just the class name:
  `Scanner scan = new Scanner (System.in);`
- To import all classes in a package, you can use the wildcard character:
  `import java.util.*;`
- No need to import java.lang (it is imported automatically into all programs).
  - That's why we don’t have to import the System or String classes explicitly.
- The Scanner class, on the other hand, is part of the java.util package, and therefore must be imported.

3.4 – Very useful: The Random Class

- The Random class is part of the java.util package.
- It provides methods that generate pseudorandom numbers.
- A Random object performs complicated calculations based on a seed value to produce a stream of seemingly random values.
- Useful also in testing main() programs by creating random input.
3.4 – RandomNumbers.java

```java
import java.util.Random;

public class RandomNumbers {
    public static void main(String[] args) {
        Random generator = new Random();
        int num1;     float num2;
        System.out.println("A random integer: "+ generator.nextInt());
        System.out.println("From 0 to 9: "+ generator.nextInt(10));
        System.out.println("From -10 to 9: "+ generator.nextInt(20) - 10);
        num2 = generator.nextFloat();
        System.out.println("A random float (between 0-1): "+ num2);
        num2 = generator.nextFloat() * 6;
        // 0.0 to 5.999999
        num1 = (int)num2 + 1;
        System.out.println("From 1 to 6: "+ num1);
    }
}
```

3.5 – Very useful: The Math Class

- The Math class is part of the java.lang package
- Math functions include
  - absolute value: Math.abs()
  - square root: Math.sqrt()
  - Exponentiation: Math.pow() Math.exp()
  - trigonometric functions: Math.sin(), Math.tan()
- The methods of the Math class are static methods (aka class methods)
- Static methods can be invoked through the class name – no new object of the Math class is needed

```
// Write code that computes the roots of a quadratic equation

discriminant = Math.pow(b, 2) - (4 * a * c);
root1 = ((-1 * b) + Math.sqrt(discriminant)) / (2 * a);
root2 = ((-1 * b) - Math.sqrt(discriminant)) / (2 * a);
```

4.2 – Java’s Conditional Operator

- Its syntax is
  ```java
  condition ? expression1 : expression2
  ```
- If the condition is true, expression1 is evaluated; if it is false, expression2 is evaluated
- The value of the entire conditional operator is the value of the selected expression

```
System.out.println("Your change is " +
        count + ((count == 1) ? "Dime" : "Dimes") +
        count + "s, ");
```

- Your change is 1 Dollar, 5 Dimes
- Your change is 3 Dollars, 1 Dime
### 2.3 – Primitive Data

- There are eight primitive data types in Java
  - Four of them represent integers
    - byte, short, int, long
  - Two of them represent floating point numbers
    - float, double
  - One of them represents characters
    - char
  - And one of them represents logical values
    - boolean

#### 2.3 – Numeric Primitive Data

- The difference between the various numeric primitive types is their size, and therefore the values they can store:

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Min Value</th>
<th>Max Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bits</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>int</td>
<td>32 bits</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>&lt; -9 x 10^18</td>
<td>&gt; 9 x 10^18</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td>+/- 3.4 x 10^38 with 7 significant digits</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>+/- 1.7 x 10^308 with 15 significant digits</td>
<td></td>
</tr>
</tbody>
</table>

What happens when you mix different primitive types in a calculation?

#### 2.5 – Assignment Conversion

- Assignment conversion occurs when a value of one type is assigned to a variable of another
  - 
  ```
  float money;
  int dollars;
  
  money = dollars;
  ```
  - The following converts the value in dollars to a float
    ```
    money = (float) dollars;
    ```
  - Note that the value or type of dollars did not change
  - Only widening conversions can happen via assignment
    ```
    dollars = money;
    ```
  - In this case you cast:
    ```
    dollars = (int) money;
    ```
2.5 – Casting

- *Casting* is the most powerful (and dangerous) technique for conversion.
- Both widening and narrowing conversions can be accomplished by casting a value.
- To cast, put type in parentheses in front of the value being converted.
- For example, if `total` and `count` are integers, but we want a floating point `result` when multiply them, we can cast:

```c
result = (float) total * count;
```

2.5 – Promotion

- *Promotion* happens automatically when operators in expressions convert their operands.
- `float total;`
  `int count;`
- The value of `count` is converted to a floating point value to perform the following calculation:

```c
result = total / count;
```