Computer Science II

Data Structures and Algorithms: Spring, 2015

Instructor: Peter Lemieszewski
pelemieszewski@vassar.edu

Lecture 1 notes
1. Course wiki:
   https://www.cs.vassar.edu/courses/cs102-201501-52/top
   
   - Labs, lectures, schedules and general information
   - Check frequently

2. Contact:
   - Office: SP 104.1 Hours: M,W 8-9
   - Email: pelemieszewski@vassar.edu
     - Also to schedule more other office hours

3. Questions:
   - Ask them!

4. Coaches:
   - Use them!

5. Inclement weather or other disruptions?
   - I will email you if class must be cancelled.
6. Texts – two of them!:

David J. Barnes & Michael Kölling

*Objects First with Java A Practical Introduction using BlueJ*

5th edition,

Pearson Education, 2012

Janet Prichard & Frank M. Carrano

*Data Abstraction and Problem Solving with Java: Walls and Mirrors*

3rd Edition,

Addison-Wesley, 2010
Course Goals

- Become more proficient at reading, writing, and understanding software, by:
- Using object oriented programming principles,
- Making use of appropriate abstract data structures:
  - i.e. To make a programming solution as simple as can be*,
- And using Java as our programming language.

* [QUOTEINVESTIGATOR.COM/2011/05/13/EINSTEIN-SIMPLE/]
Before Java

- All programming languages are abstractions of... something.
  - Assembly language is an abstraction of the host computer (Assembler)
  - Imperative programming languages are abstractions of assembly language (Compiler + Assembler)

- A Problem, and its solution requires one to:
  - Think in terms of the computer architecture

- Costs for a solution: incredibly high!
  - LOC productivity – low
  - Bugs, bugs - and more bugs
Enter OOP

To improve error prone programmer productivity, create another abstraction layer:

- Model the problem you want to solve
  - Not the machine where the solution will run

- Object Oriented Programming, OOP:
  - Provides tools to represent various elements of a problem

- General representations
  - “objects” represent elements of the problem in a “solution space”
Enter Java

Complexity for distributed software solutions grew:

- Because the internet!
- The same software has to run on different platforms
- But, software still had to be built for particular machine architectures

- Java, an OOPL, became popular by providing platform independence,
- Another level of abstraction:
- Compile java program into generic bytecode instructions
- Run compiled program in a JVM Java Virtual Machine handles machine dependencies.
A Class:

- Describes a set of objects that have identical characteristics and behaviors.

Less formally, a class is:

- A *conceptual* component in a particular problem space:
  - Think of it as a blueprint, MacGuffin, or a new (abstract) data type.

An object is:

- An instance of the class.
- Not a representation, but some *thing*.
How does one get an object to do something? Via Methods.

A method is a request you can make of an object

- The different methods (requests) available for an object are defined by the interface.
- Parameters are additional pieces of information needed by an object to fulfill some requests

The set of available methods defines the class interface.
Many *instances* can be created from a single class.

- An object has *attributes*: values stored in *fields*.

- The class defines what fields an object has, but each object stores its own set of values
  - i.e. the *state* of the object.
OOP: Return Values

“The courtesy of a reply is requested”

A method may return a result, or it may not.

- A return type is associated with every method
- No result? The type is *void*.
- (more to come…)
An Example Class

Type Name: Light

Interface:

The interface defines the set of requests one can make for a particular object.

Source code:

In Java, we can create a class instance:

```java
Light fixture = new Light();
```

And call a method:

```java
fixture.on();
```