cs2340: Objects and Design
Learning Goals

Primary Learning Goals

This class covers a number of topics closely related to objects and design. After this class, you should have a *thorough* understanding of the following:

Object-Oriented Analysis You will use Class Responsibility Collaboration (CRC) Cards and Scenarios to help you understand a domain in terms of its objects. Proper use of CRC cards put you on the right track for creating a good object-oriented design.

Object-Oriented Programming There is an art and a science to object-oriented programming. Just being able to program with an object-oriented language (such as Java or C++) is not enough. This class will help you move from a "programming with objects" model to a truly object-oriented model of programming. To make this transition, you will learn a new programming language, Smalltalk. Smalltalk is the quintessential object-oriented language and many of the important ideas in OO programming are made accessible in Smalltalk:

- **Responsibility-Driven Design** In object-oriented programming, you *make the objects do the work*. In Smalltalk, everything is an object. That allows you to write object-oriented code that makes the objects responsible for doing the work. If you want to know what 5 factorial is, simply send the factorial message to 5.
- **Use High-Level Tools** Smalltalk is a high-level language with high-level tools. It is late-bound and has a persistent memory. You can play with code in a workspace. You can inspect objects to understand them. You use a browser (not just a text editor) to write and organize code. You use change-sets to distribute code to others. You can program in the debugger. In the Morphic environment, you can poke at the GUI to see what is going on underneath the surface. OO programmers use these high-level tools to create OO code.
- **Message Passing** Message passing is fundamental to most OO languages. Almost everything that happens in Smalltalk is that a message (perhaps with some arguments) gets sent to an object and an object is returned. This allows for some fairly sophisticated and elegant OO code.
- **Instance/Class** The canonical example of a class-based language is Smalltalk—its model of what is a class and what is an instance is definitive. To create object-oriented code, these concepts must be used appropriately.
- **Unit Testing** Designing and maintain robust code in larger projects is tough. Unit testing is one of the best ways to solve this problem. The definitive framework for unit tests, known as xUnit, was first developed for Smalltalk.
- **Recognize Bad Code** There are many symptoms of bad OO code: god classes, non-noun classes, too much implemented on the class side, not enough reuse, lengthy methods, etc. A good programmer must be able to recognize and address these problems.

User Interfaces Most applications today involve some kind of a graphical user interface (GUI). There is an art and a science to creating good user interfaces.

Programming GUIs This class will examine GUIs ranging from the rudimentary to the sophisticated. Principles for creating OO GUIs will be studied. Often, a good design will separate the model (the state of the application) from the view (what is seen of that state). Some GUI paradigms, such as MVC (Model View Controller), explicitly enforce this separation; others, such as Morphic, make it optional. It is important to understand when, why, and how the model should be separated from the view.

Usability Just because a user interface is well designed from a programming standpoint does not mean that it is usable by others. Creating software that others can use is challenging. Yet, there are some principles and techniques that can help us to create better software for others.

Secondary Topics

In addition to the primary topics listed above, we will cover several secondary topic. After this class, you should have a good understanding of the following:

Teamwork You will work with others on a large class project. Learning to work well in a team is one of the most critical job skills you can have.

Extreme Programming (XP) Extreme programming is a popular method of software engineering. This class will introduce some of the elements (pair-programming, unit testing) essential to XP.

Design Patterns Design patterns help you solve common programming problems in a standard way.

Virtual Machines Most OO languages use a virtual machine with garbage collection. There are several reasons for this.

OO Language Features There are many mature object-oriented languages available: Java, Smalltalk, C++, Objective-C, C#, Eiffel, Ruby, CLOS, etc. All have their strengths and weaknesses. You should understand how these languages differ, based on language features.

History of OO Programming is a young field that is still evolving. It is important to understand the past to anticipate the future.