Objects

- The term object is not easily defined
- According to Webster:
  - Object: a visible or tangible thing of relative stable form; A thing that may be apprehended intellectually; A thing to which thought or action is directed
- In this class, we will use the following definition:
  - An object has state, behavior, and identity (Booch)

State

- The state of an object encompasses all of the (static) properties of the object plus the current (dynamic) values of each of these properties
- A property is an inherent or distinctive characteristic, trait, quality, or feature that contribute to making an object uniquely that object
- We will use the word attribute, or data member, to refer to the state of an object

Behavior

- Behavior is how an object acts and reacts, in terms of state changes and interactions with other objects.
- An operation is some action that one object performs upon another in order to elicit a reaction.
- We will use the word method to describe object behavior.
- Invoking a method causes the behavior to take place.
Object-Oriented Design

• Traditional design mechanisms are control based
  – Flow charts!!
• OOD is object-based
  – Identify the objects that make up the system
  – Define the state and behavior of these objects
  – Determine the relationships between these objects

Objects and Classes

• A class is an abstract definition of an object.
  – It defines the structure and behavior of an instance (object) in the class.
  – It serves as a template for creating objects.
• When doing OOP you define classes, from which objects are instantiated.
• When doing OOD, although we focus on objects, we are really interested in finding classes.

Finding Classes

• A class should capture one and only one key abstraction.
  – Student class which represents a student and their schedule for the current quarter.
  – Student and schedule classes.
    • The student class would include a reference to an instance of a schedule as part of its state.
• A class should represent something as opposed to doing something.
Naming Classes

• A class name is typically a single noun that best characterizes the abstraction.
• Difficulty in naming a class may be an indication of a poorly defined abstraction.
• Names should come directly from the vocabulary of the domain.
  – Student, Schedule, Course, College, …

Style Guidelines

• A style guide should dictate naming conventions for classes.
• Sample style guide.
  – Classes are named using one or two nouns.
  – Class names start with an upper case letter.
  – Underscores are not used.
    • Names consisting of multiple words are pushed together and the first letter of each additional word is capitalized.

Define Class Semantics

• After naming a class, a brief concise description of the class should be made.
  – Focus on the purpose of the class not on the implementation.
• The class name and description form the basis for a model of the system.
• Look for the what and ignore the how.
Finding Classes

- Classes are often identified by examining the nouns and noun phrases in the description of a system.
- Nouns found may be:
  - Classes/objects.
  - Descriptions of state of an object.
  - External entities (more on this later).
  - None of the above.

Filtering Nouns

- When identifying nouns, be aware that:
  - Several terms may refer to the same object.
  - One term may refer to more than one object.
  - Natural language is ambiguous.
- This approach can identify many unimportant or non-objects:
  - The list must be filtered.

Looking at Nouns

- The following requirement was written for a banking system.
  - “Legal requirements shall be taken into account.”
- What will be the result if only nouns are considered?
- Each noun must be considered in the context of the problem domain.
  - Nouns cannot stand by themselves.
RULE #1

• You must test every piece of code that you write for this course
  – If you want help from me or any of the TAs you must show test code first

RULE #2

• Don’t be afraid to write throw away code
  – When testing you will probably write code that will never “see the light of day”
  – You should write lots of small programs to test and that allow you to experiment with programming

RULE #3

• Always take baby steps
  – Start a task by picking the smallest easiest portion to do first
  – Test what you write before you move on to tackle another task
  – Always try to build on the simple parts that you know work correctly
RULE #4

• Don’t re-invent the wheel
  – Before writing anything – check to see if it has already been written (by you or someone else)
  – Feel free to re-use code (anything I post is yours to use) BUT MAKE SURE YOU CREDIT THE SOURCE

UML

• Now that we have a taste of OOD, the next question to ask is “how do you document a design?”
• The Unified Modeling Language (UML) is a notation (mostly graphical) that is used to express designs.
  – Developed by Grady Booch, Jim Rumbaugh, and Iver Jacobson.
• UML is a nonproprietary industrial standard and open to all.

UML Views

• A design describes several aspects of a system.
  – Functional aspects describe the static structure and the dynamic interactions between components.
  – Non-functional aspects include items such as timing requirements, reliability, or deployment strategies.
• UML provides five different views that document the various aspects of a system.
• A view is an abstraction consisting of a number of diagrams that highlight a particular aspect of the system.
UML Views

<table>
<thead>
<tr>
<th>View</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use-Case</td>
<td>Describes the functionality that the system should deliver as perceived by external actors. Used to document the requirements and specifications of a system.</td>
</tr>
<tr>
<td>Logical</td>
<td>Illustrates how the functionality of the system will be implemented in terms of the system’s static structure and dynamic behavior.</td>
</tr>
<tr>
<td>Component</td>
<td>Shows the organization of the code components.</td>
</tr>
<tr>
<td>Deployment</td>
<td>Illustrates the deployment of the system into physical architecture with computers and devices called nodes.</td>
</tr>
</tbody>
</table>

UML Diagram Types

- UML defines nine different diagram types that describe specifics aspect of the system.
- A single diagram cannot possibly capture all the information required to describe a system.
  - Several diagrams will be used to describe the design.
- It may not be possible, or even desirable, to use a single class diagram to describe a complete system.
  - Concentrate on key areas of the design and describe them using different class diagrams.

UML Class Diagrams

- A class diagram in UML describes the static structure of a system in terms of its classes and the relationships among those classes.
- Class diagrams are the most common way to describe the design of an object-oriented system.
- Class diagrams provide ways to describe even the subtlest aspects of a class.
  - In order to avoid becoming lost in the details, we will only look at the more commonly used features of class diagrams.
UML Class Diagrams

- A class, in a class diagram, is drawn as a rectangle that can be divided into three compartments.
  - The name of the class appears in bold text centered in the compartment at the top of the rectangle.
  - The compartments that describe the state and behavior are optional.
- Type information for the methods that make up the behavior of the class is optional.
  - Method names that require parameters must be followed by an open and closed parentheses.

Class Diagrams

```
Clock

secs : int
mins : int
hours : int

setTime()
adjustTime()
reset()
```

Class Diagrams

- Include only as much information in a class diagram that is required for a reader to understand your design.
- Don’t overwhelm a reader with trivial details.
  - Do not include contain obvious behaviors in your diagrams.
  - Classes whose behaviors are well known, such as classes provided in a system library, will either be drawn as a single rectangle with no compartments, or omitted from the diagram altogether.
Class Relationships

• Simply describing the classes in a system is not enough to describe how it works.
  – For example, if you were asked to describe a family it would not be enough to say, “A family consists of parents and children.”
• A relationship exists between two classes if one class “knows” about the other.
  – Typically a relationship exists between two classes if an instance of one class invokes a method or accesses the state of an instance of the other class.

UML Class Relationships

• UML defines several different types of relationships that can be used to describe the way in which two or more classes are related in a class diagram.
• In this class we will use three different types of relationships: associations, dependencies, and generalizations.

Associations

• An association is a relationship that ties two or more classes together, and represents a structural relationship between instances of a class.
• An association indicates that objects of one class are connected to objects of another.
  – The connection is permanent and makes up part of the state of one of the associated classes.
• From a programming perspective two classes are associated if the state of one class contains a reference to an instance of the other class.
Navigability

- **Navigability information** can be included in a UML class diagram to clarify the nature of the relationship between two classes.
- An arrow is added to the solid line that represents an association to indicate the direction of a relationship.
Multiplicity Information

- In addition to navigability, multiplicity can be used to describe the nature of a relationship between classes.
- Multiplicity information is added to an UML diagram by placing a '*' or a number next to one end of an association.
- The '*' indicates that there may be zero, one, or more instances of the class.

Multiplicity

![ATM Bank Multiplicity Diagram]

Dependency

- A dependency is a using relationship that specifies that a change in one class may affect another class.
  - A dependency is inherently a one-way relationship where one class is dependent on the other.
- Consider how associations an dependency are typically implemented in a program.
  - Associations are usually implemented as part of the state of one of the classes.
  - A dependency typically takes the form of a local variable, parameter, or return value.
Dependency

Car
- running
- lock
- accelerate

Engine
- curRPM
- running

Driver