Today’s topic:

- Last time, we talked about objects and classes.

- Today, we will cover the rest of Chapter 1, and move on to Chapter 2, plus we’ll learn a new term along the way.
Inheritance

- The **classes** we write allow for multiple **object instances** to be created.

- The class is a **classification** of objects.

- Just as we **classify** real world objects, we can have **subclasses** in our OO designs as well.

---

**Inheritance**

- Inheritance can be viewed as an *is a* relationship.
  - A **Dog** is a Mammal.
  - A **Cat** is a Mammal.
Composition

- Composition means simply that some objects are composed of other objects.

- We will often build objects out of other objects.
  - For example a DeckOfCards object may be composed of a collection of PlayingCard objects.

Composition

- Composition can be viewed as an has a relationship.
  - A house has a door.
  - A cat has a tail.
  - A deck of cards has a collection of cards.
  - A game board has a collection of spaces.
What’s the point?

- We utilize classification systems as a way to enable us to think abstractly about them.

- For example, the taxonomy system in biology.
  - In this example we group organisms into classifications based on shared traits.

- In OO objects that share similarities, can provide similar behaviors.

Definition time

- How would you define abstraction?
Abstraction

- “Abstraction arises from a recognition of similarities between certain objects, situations, or processes in the real world, and the decision to concentrate upon these similarities and to ignore for the time being the differences.”

- “An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer.”

Abstraction

- “A model of a complex system that includes only the details essential to perspective of the viewer of the system.”
  http://guruzon.com/oop-concepts/abstraction/what-is-abstraction-example-tutorial-characteristic-how
Abstraction

- **Abstraction** allows us to describe things at a high level without getting distracted by the details.
  - For example, we can talk about an object and what it does, without worrying about the implementation details of how it actually does it.

- **Encapsulation** puts the data and functions that operate on that data together as a form of data hiding.
  - Details that are not important to use the object are hidden from other objects that interact with it.
Abstraction with Software Development

- Every *program* you write is an *abstraction*.

- Your program *models* some process or system in a way to avoid *unimportant* details and focuses on the *important* ones.

Chapter 02:

“How to think in Terms of Objects”
Determining the Users

Who are the users?
- Typically are users can be thought of as the end users or customers of our software.
- Often our users are other developers who wish to reuse our classes.

How to Think In Terms of Objects

Three keys to the OO thought process:
- Knowing the difference between the interface and implementation
- Thinking more abstractly
- Giving the user the minimal interface possible
Interface vs. Implementation

When designing a class:

- *Identify what the user needs to know and what the user does not need to know.*

- *Encapsulation is the means by which nonessential data is hidden from the user.*

Interface vs. Implementation

When designing the interface, think about how the object is used and not how it is built.

- *The interface consists of the behaviors of the objects.*

- *Identify the purpose of each object from the user’s perspective.*

- *Don’t limit yourself early on, many of the initial ideas will not necessarily make the final cut, interface design is an iterative process.*
Interface vs. Implementation

The implementation is the internal workings of an object.
- A change to the implementation should not require a change to the user’s code.
- Because the interface does not change, the user does not care if the implementation is changed.

Interfaces vs Implementation

Anything that is not part of the public interface can be considered part of the implementation.
- This means that the user will never see any of the instance variables or methods that are part of the implementation.
- The implementation defines how the class meets the interface requirements.
Abstraction with OO

- Thinking *abstractly* about a problem can *simplify* our solution.

- Abstraction allows us to *generalize* and develop classes that can be *reused* and applied to different situations.
  - Reusable classes tend to have interfaces that are more *abstract* than *concrete*.
  - Concrete interfaces tend to be very *specific*.
  - Abstract interfaces are more *general*.

The Minimal User Interface

Give the users *only* what they *absolutely* need.

- It is better to have to *add* interfaces because users really need it than to give the users more interfaces than they need.

- It is vital to design classes from a user’s perspective and not from an information systems viewpoint.

- *End users* (or customers) should be included in the design process as well as *developers*.