

# *Introduction to Programming II*

## *More Objects*

Chris Brooks

Department of Computer Science  
University of San Francisco

## 5-2: More practice with objects

---

- Let's use our Point class to build some additional objects
  - Let's build a Student class.
  - Students have firstName, lastName, major, ID.
  - Add setters, getters, constructors.

## 5-3: More practice with objects

---

- Now add:
  - `toString()`. Print the student's name. Test this with `println`.
  - `compareTo()`. Compare last names.

## 5-4: More practice with objects

---

- Now create a Course class.
  - Should have name, room, students.
  - students should be an ArrayList.
  - Add addStudent(Student) and isEnrolled(Student) methods.

## 5-5: Designing a Program

---

- Knowing the syntax of how to build a class is only the beginning.
- The bigger challenge is figuring out how to fit the pieces together.
- The software development process consists of the following steps:
  - Establishing requirements
  - Creating a design
  - Implementing the design
  - Testing
- Usually, this is an iterative, repeated process.

## 5-6: Requirements

---

- Requirements indicate what a program is supposed to do.
- Expressed as a *functional specification*.
  - What is the input like?
  - What must the output look like?
  - Are there other programs it must interact with?
  - How quickly must the program run?
- Often, this comes from a client.
- Usually not as precise as you'd like.

## 5-7: Design

---

- This is the 'how' part of the program.
- Specifies the classes that are needed and how they interact.
- What methods are called, what data is returned.
- Skipping this step can lead to serious, unpleasant bugs.
- A good design should make implementation straightforward.

## 5-8: Structure Charts

---

- A structure chart is a helpful tool for doing design.
- Helps to *divide and conquer*
- Divide the problem into smaller pieces until you reach pieces that can be tackled directly.
- This is called top-down design.

## 5-9: Example: student database

---

- Let's say we've been hired by USF to build an application for tracking students.
- It should be able to do the following:
  - Add new students to the database
  - Delete a student from the database
  - Enter a student's test scores
  - Print out the average of a student's scores
- These are our *requirements*

## 5-10: Top-down structure chart

---

- main is at the top of the chart
- Nodes below this represent portions of the program that are called.
- Arrows indicate input and output of data.
- Nodes at the next level are then decomposed in the same way.
- Eventually, we get to methods we know how to implement.

## 5-11: Top-down structure chart

---

- The structure chart indicates the calling sequence of the program.
- Serves as a template for the actual program.
- Annotate parameters as in, out, or in-out
- Objects called with methods on them are parameters too!

## 5-12: Choosing Objects

---

- Each object should have one well-defined responsibility.
- A common mistake is to “cram too much” into a single object or method.
- Methods are typically 1-2 screens of code.
- Before making a new class, does what you need already exist?

## 5-13: Bottom-up-design

---

- To implement the program in our structure chart, start at the leaves and work upward.
- Bottom-level methods can be implemented directly.
- Methods at the next level up are implemented in terms of those methods.
- Eventually, you make it back up to main.

## 5-14: Unit testing

---

- As you code each method, you should test it in isolation.
- Write a small program that calls this method with all expected inputs.
- Testing each method seems tedious, but it's very important.
  - A simple bug missed at a lower level can be very difficult to find later on.
  - Good programming practice can help limit the amount of time you spend debugging.