Intro to Programming II
Structure Charts

Chris Brooks

Department of Computer Science
University of San Francisco
3-0: Anatomy of an Object

- Recall that an object consists of:
  - Instance data
  - Methods

- Methods consist of:
  - Parameters
  - Local data
  - Method body
  - Return statement
3-1: Constructors

6 A constructor is a method that is called when an object is first created.
6 Its responsibility is to initialize an object’s instance variables.
6 It must have the same name as the class in constructs.
6 It has no return type.
6 It is called when ’new’ is invoked.
public class Point {
    public int xval;
    public int yval;

    Point(int x, int y) {
        xval = x;
        yval = y;
    }
}
Point p = new Point(3,4);
Multiple Constructors

It’s often helpful to be able to specify some instance variables, but not all.

For example, let’s say our circle class has a default radius of 1.

If the radius is something else, users can specify it.

Point p1 = new Point(3,3); Circle c = new Circle(p); creates a circle with radius 1 at (3,3).

Point p1 = new Point(3,3); Circle c = new Circle(p,5); creates a circle with radius 5 at (3,3).
public class Circle {
    private Point center;
    private int radius;

    public Circle(Point c) {
        center = c;
        radius = 1;
    }

    public Circle (Point c, int r) {
        center = c
        radius = r;
    }
}
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.
3-6: 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.
3-7: 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.
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 to-down design.
3-9: 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
3-10: 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.
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!
Choosing Objects

1. Each object should have one well-defined responsibility.
2. A common mistake is to “cram too much” into a single object or method.
3. Methods are typically 1-2 screens of code.
4. Before making a new class, does what you need already exist?
3-13: *Bottom-up-design*

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