

Introduction to Programming II

Constructors

Chris Brooks

Department of Computer Science
University of San Francisco

4-2: Anatomy of an Object

- Recall that an object consists of:
 - Instance data
 - Methods
- Methods consist of:
 - Parameters
 - Local data
 - Method body
 - Return statement

4-3: Constructors

- A *constructor* is a method that is called when an object is first created.
- Its responsibility is to initialize an object's instance variables.
- It must have the same name as the class in constructs.
- It has no return type.
- It is called when 'new' is invoked.

4-4: Example

```
public class Point {
    public int xval;
    public int yval;

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

4-5: 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).

4-6: Multiple Constructors

```
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;
    }
}
```

4-7: Exercise

- Modify the 'book' class from last week's lecture.
- Add three constructors:
 - Default: takes no arguments.
 - 1 argument: title
 - 2 arguments: author and title
 - author should be an instance of 'name'
- Add a constructor to 'name' that takes two arguments: first name and last name.

4-8: 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.

4-9: 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.

4-10: 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.

4-11: 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.

4-12: 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*

4-13: 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.

4-14: 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!

4-15: 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?

4-16: 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.

4-17: 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.