Course Introduction

Principles of Software System Construction
Prof. Jonathan Aldrich
Fall 2011
Scaling Up: From Programs to Systems

• You’ve written small- to medium-size programs in 15-211
• This course is about managing **software complexity**
  – What does that mean?

• Some aspects of software complexity
  – Scale of code: KLOC -> MLOC
  – Software infrastructure: libraries, frameworks, components
  – Worldly environment: external I/O, network, asynchrony
  – Software evolution: change over time

  – Contrast: algorithmic complexity
    • Not an emphasis in this course
From Programs to Systems

- Algorithms and data structures written from scratch
- Functions with inputs and outputs
- Sequential, local computation
- Formal functional specifications

- Reuse of libraries, frameworks
- Asynchronous and reactive
- Parallel and distributed computation
- Composable, scalable, targeted models

Our goal: understanding both the building blocks and also the principles for construction of software systems at scale
The Four Course Themes

• **Threads and Concurrency**
  – Multicore processors → performance requires **parallelism**
  – Concurrency is also a crucial **system abstraction**
    • Compute in the background while maintaining responsiveness to users
  – Focus: application-level concurrency
    • Contrast functional parallelism (150, 210) and low-level concurrency (213)

• **Object-oriented programming**
  – Excels at creating **flexible designs and reusable code**
  – A primary paradigm in industry
  – Focus: Java
    • Used in industry, upper-division courses

• **Analysis and Modeling**
  – Practical **specification techniques and verification tools**
  – Address challenges of threading, correct library usage, etc.

• **Design**
  – Proposing and evaluating **alternatives**
  – **Modularity**, information hiding, and planning for change
Course Preconditions and Postconditions

Preconditions

• 15-122 or equivalent
  – 2 semesters of programming, knowledge of C-like languages
  – Basic reasoning about programs; basic algorithms and data structures

Postconditions

• Java and OO development skills
  – Use of development, testing, and analysis tools
  – Use of frameworks and libraries
• Understanding large-scale software
  – Frameworks, ecosystems, libraries, components
  – Design patterns and practices
• Concurrent and distributed systems
  – Scaling and performance
  – Safe programming practices
Motivating Example: GraphLayout

Source code: [http://java.sun.com/applets/jdk/1.4/demo/applets/GraphLayout/example1.html](http://java.sun.com/applets/jdk/1.4/demo/applets/GraphLayout/example1.html)

Discussion Points from Class

• What does the design of GraphLayout look like, conceptually?
  – Graph representation
  – Layout algorithm
  – GUI for displaying, responding to user input

• What is most important about the design?
  – Encapsulation
    • To enhance reuse
    • To protect data structures from undesired interference
    • To make the system more robust to change

• How should the GUI be organized
  – Events: need to react to external input, update on regular clock ticks

• How to avoid a “freezing display”
  – Compute, display in different threads \(\rightarrow\) raises coordination challenges
Motivating Example: Virtual Worlds
Discussion Points from Class

• How can we get a virtual world to scale to thousands of users?
  – Offload graphics to the client
  – Take advantage of threads on multicore processors
  – Use a farm of servers, each hosting part of the world
• How can we organize the system to easily add new kinds of virtual objects
  – Need some way of associating each object with its behavior, e.g. using function pointers (or objects as we will see)
• How can we take advantage of similarities in the behavior of similar objects (e.g. different kinds of swords in WOW)
  – A: inheritance (to be discussed...)
Object Background

• Background: simulation → Simula 67, first OO language
• Object-oriented programming: A way of organizing code around data structures rather than operations
• Bottom-up rather than top-down design has benefits:
  – Easier to reuse concepts in new programs
  – Easier to extend the program with new concepts
    • E.g. variations on old concepts
  – Easier to modify the program if a concept changes
    • code changes localized to code implementing the concept
Objects

- An object is a package of state and behavior
  - Fields hold data values (this part is like a struct value)
  - Methods perform operations on that data
    - Functions embedded within the object, which have access to its fields
  - Methods also control access to the fields
    - Usually don’t want to read the fields from outside—make them `private`
Example: Points and Rectangles

class Point {
    int x, y;
    int getX() { return x; } // a method; getY() is similar
    Point(int px, int py) { x = px; y = py; } // constructor for creating the object
}

class Rectangle {
    Point origin;
    int width, height;
    Point getOrigin() { return origin; }
    int getWidth() { return width; }
    void draw() {
        drawLine(origin.getX(), origin.getY(), origin.getX()+width, origin.getY());
        ... // more lines here
    }
    Rectangle(Point o, int w, int h) {
        origin = o; width = w; height = h;
    }
}
Example: Points and Rectangles

```java
class Point {
    int x, y;
    int getX() { return x; } // a method; getY() is similar
    Point(int px, int py) { x = px; y = py; } // constructor for creating the object
}

class Rectangle {
    Point origin;
    int width, height;
    Point getOrigin() { return origin; }
    int getWidth() { return width; }
    void draw() {
        drawLine(origin.getX(), origin.getY(), // first line
                    origin.getX()+width, origin.getY());
        // more lines here
    }
    Rectangle(Point o, int w, int h) {
        origin = o; width = w; height = h;
    }
}

Some Client Code

Point o = new Point(0, 10); // allocates memory, calls ctor
Rectangle r = new Rectangle(o, 5, 10);
r.draw();
int rightEnd = r.getOrigin.getX() + r.getWidth(); // 15
```
Bureaucracy

• TA: Andrew Chang
• Section: bring your laptop tomorrow
• Textbooks (see web)
• Assignments and Evaluation (see web)
  – First assignment out tomorrow: Java warm-up
• Course Schedule (see web)
• Policies and Expectations (see web)
Toad’s Take-home Messages

• 214’s focus: managing complexity, from programs to systems
  – Threads and concurrency
  – Object-oriented programming
  – Analysis and modeling
  – Design

• Graphlayout and virtual worlds illustrate some challenges

• Object-oriented programming organizes code around **concepts**
  – Methods capture behavior, fields capture state
  – As we will see, this organization allows
    • Greater reuse of concepts
    • Better support for change when concepts vary