Introduction

CSE 132
Instructional Staff

• Instructors – Ron Cytron & Roger Chamberlain
  – Offices: Bryan 525 & Bryan 509
  – Email: cytron & roger @wustl.edu

• Head TA – Andrew Buckley
  – Use piazza for posting problems
  – You should have an email invite to join our piazza

• TAs – there are many! See the web page
Course Web Page

• Google cse132 cytron
• Contains calendar (expanding, it’s short now)
• Contains studio and lab assignments
• Documents grading, collaboration, and late policies
  – Note use of WUTexter for participation credit
• Contains documentation on language (Java) and tools (Eclipse, Subversion)
Comparison to CSE 131

• 131 was redesigned in 2012
  – Bottom-up approach
  – Students write code from scratch
• 132 has not been similarly redesigned
  – More “top down”
  – You are placed in the middle of a lot of code
    • And asked to understand it, change it
• I will try to make the course more 131-like for you this semester
Comparison to CSE 131

• Getting labs to work is insufficient to earn an A grade in this course
  – The labs are noticeably more difficult
  – The TAs are being instructed to help, not solve
  – Code style will be graded

• The material is more like the challenges most programmers face daily
  – User interfaces
  – Persistent data
  – Concurrency
  – Network programming
Typical Week

• Monday morning
  – Lecture in Louderman 458 at 10am and 11:30am

• Monday afternoon
  – Studio exercises (and perhaps quizzes) in Urbauer and Whitaker labs (attendance is required!)

• Wednesday afternoon
  – Lab, demos, quizzes (perhaps)

• Additional help: TBD, probably Sun. and Tue.
Topics

• Lab 1 – MVC, a simple user interfacing w/ Swing
• Lab 2 – persistent data, data representation, additional UI work (using GUI builder)
• Lab 3 – concurrency (multiple threads at the same time, all using a common memory system)
• Lab 4 – distributed computing (via the network)
• Lab 5 – multi-person game
CSE 102 Prototype Section

- Our department needs an Intro to Computer Engineering course
- It will likely become our CSE II course, replacing CSE 132 next year
- We need 16 students from the current 132 to try this out. Criteria:
  - Interest, reasonable 131 grade
  - Able to make all sessions
    - Meets 2:30 to 4 PM M W F
  - Willing to serve as TA Fall 2015 for the new course
CSE 102 Prototype Section

• Similar concepts
  – How are integers represented?
  – Demystification
  – User interface
  – Communication between computers
  – Touching the real world
  – Doing more than one thing at a time

• Some new emphases
  – Performance, what goes where
  – Hardware, something you can hold in your hand, stick wires into, power with a 9V battery and carry around
CSE 102 Prototype Section

• Different platform for study
  – Arduino boards
• Currently a “work in progress”
• Survey link will be sent to you today
  – Only complete if you really can make the meetings
  – And you can, without question, serve as a TA Fall 2015
Lecture Next

• Ron thanks Roger profusely for these slides
  – But any mistakes are Ron’s now
Swing

• Set of classes for providing Graphical User Interfaces (GUIs) in Java

• GUIs are constructed using components, e.g.:
  – Buttons
  – Sliders
  – Check Boxes

• Components must be contained within a container class, e.g., JFrame

• Uses Model/View/Controller (MVC) paradigm
Let’s Play

• JFrame
  – What do we see?
  – Put one thing in it (JLabel), what do we see?
  – Must we resize it to see stuff?
  – Put a second thing in it, what do we see?
    • JFrame can have only one thing in it
• Swing containers and components
  – JPanel is both!
• Let’s add a bunch of things
Swing

- In CSE 131, Sedgewick *wrapped* swing with StdDraw
  - You didn’t have to mess with visible etc.
  - There is just one Frame though!
  - And no buttons, sliders, etc.

- Now we play with swing and create GUIs by hand. This is part of Module 1 of CSE 132

- But it’s tedious
  - So we will soon use WindowBuilder
  - Built into eclipse, so you already have it
Model/View/Controller

• Model
  – Data that represents the “thing” that is the subject of the computation
  – Abstract information, does not care how it is viewed or altered

• View
  – Presentation to user of model

• Controller
  – Manages user interaction with model
Model/View/Controller

Model
- Encapsulates application state
- Responds to state queries
- Exposes application functionality
- Notifies views of changes

View
- Renders the models
- Requests updates from models
- Sends user gestures to controller
- Allows controller to select view

Controller
- Defines application behavior
- Maps user actions to model updates
- Selects view for response
- One for each functionality

State Query
Change Notification
State Change

View Selection
User Gestures
Method Invocations
Events
Why MVC?

• Separation of concerns / Modularity
  – Developer concerned with one thing at a time
  – When we write the code, we don’t want to know
    • What will control
    • What views will be present

• Orthogonality
  – Mix and match views and models

• Consistency across several views
  – Model is the only object keeping track of the value
Let’s try it without MVC

• A year
• Some classes that care about the year when it changes
• Every time a new view comes along, pervasive changes to the code
  – Constructor
  – Instance vars
  – Mutator(s)
With MVC

• Model allows for any number of
  – Observers – interested parties
  – Subscribers (same idea, different word)
• When something in the model changes
  – It notifies its subscribers
  – By publishing the change
  – This is often called pub-sub
• Example: eBay auction
  – Many interested parties
  – Don’t want to continually check for latest bid
  – So eBay publishes changes to its subscribers
DefaultBoundedRangeModel
(simple model for integers)

• Characterized by 4 parameters:
  minimum ≤ value ≤ value+extent ≤ maximum

• View could be
  – slider (JSlider) or
  – text (JTextField)

• Each of these could also change the model’s value
  – But they do so by changing the model
  – And then all views are listening for such changes
Year revisited

• Do you recall `extends` in Java?
  – The “isa” gesture
  – A Year is-a `DefaultBoundedRangeModel`
  – Demo

• Temperature

• Price of gas
Combining pub-sub with MVC

• Events – button push, hit return
• Objects that want to know about events “subscribe” as Listeners
• Swing components “publish” to subscribing listeners when user events happen
  – Invoke actionPerformed() for ActionListener
  – Invoke stateChanged() for ChangeListener
Exceptions

• Deviations from the normal flow of control
• “Old style” error checking:
  
  ```java
  if (i < 0 || i >= A.length) {
      // handle out of range index
  }
  else {
      // access array element A[i]
  }
  ```

• Exceptions allow us to be a bit more general
Try/Catch Block

try {
    // arbitrary code that might throw an exception when something goes wrong
}
catch (Exception e) {
    // handle the thrown exception
}
Unchecked / Checked

The class "Throwable" and some of its subclasses.
Persistent Data and I/O

• How do we retain data between invocations of our program? Save it on disk!

• How do we do that? Read and write files.
  – Open file
  – Read or write data (in various formats)
  – Close file

• Generalize – instead of a file, maybe:
  – to/from a network (e.g., a Java program running on another machine), or
  – console window
Java uses Stream Concept

• Upstream writer, downstream reader

• Source writes to stream

• Destination reads from stream

• Either endpoint might be a file or some other input/output device, e.g.,
  – Dest. could be open window on display screen
  – Source could be a temperature sensor
Stream Conventions

• FIFO ordering (First-In-First-Out)
• Protocol must be same at both ends of stream for effective communication to take place
  – Stream of bytes? Chars? Integers?
• Properties supported by streams that “wrap” other streams, e.g.,
  File f = new File(filename);
  FileInputStream stream = new FileInputStream(f);
  DataInputStream dataIn = new DataInputStream(stream);
Wrapping Streams

• A stream can take another stream as a parameter to its constructor
• The outer stream delegates to the wrapped one
• E.g.,

```java
DataOutputStream out =
    new DataOutputStream(  
        new BufferedOutputStream(    
            new FileOutputStream(...)
        )
    );
```
Studio Today

• Make sure you can login
• Form groups (of 2-3 in Whitaker, 4 in Urbauer)
• Do the exercise
• Get signed out by a TA
Lab on Wednesday

• Make sure you can use your personal repo
• Start, and finish, Lab 1
• Get signed out by a TA
Studio Next Week

• We don’t meet on ML King Jr. holiday
• Studio will be on Wednesday
• Get studio signed out by a TA
• Start Lab 2, which is not due until the following Wednesday