CS1013 – Programming Project

Dr. Gavin Doherty

ORI LG.19

Gavin.Doherty@cs.tcd.ie

Trinity College Dublin

DSG
Distributed Systems Group

Team Work

- Set up a group communication mechanism for your team.
- SVN server will be set up for you.
- Swap contact details.
- Arrange time **outside labs to work**
- You get back what you give to your team.
- Agree on work ethic / programming approach
- Keep records of who does not work/attend!

Project

- Let me know of any problems ASAP.
- Non-attendance is unfair to your team-mates and will result in individual failure – as will not doing any work!
- Programming in labs in pairs.
- Assessment at labs and weekly submission.
- Lectures to cover additional topics as required.
- First part: read in data and create an outline of your program.

Pair programming

- 1 person “drives” – typing the code at the keyboard.
- 1 person reviews, points out problems, suggests changes, is generally constructive. Doesn’t dictate the code.
- Agree on what to work on.
- Change roles at least every half hour.
- This is a fairly common practice.

Structure

To start with, we want to read in the data from file.

We'll need to put the data somewhere: what should you use to store the data?

Do we need 1 class or many classes?

Where should this code go?

Setup would be a reasonable place.

Extracting data

- We typically only want to look at a small portion of the data at once.
- So we want to define a number of queries on the data.
- Some queries will just select a smaller part of the dataset.
- Other queries will generate new information – eg. the totals or averages for a subset of the data. You will have to decide how to pass this information around within your program.
- Where will the queries be called?
- Generally in response to user input, but may have initial (default) displayed.

Rendering

- The data has been loaded in, the information we want has been extracted from it, now we want to draw it to the screen.
- We've already looked at drawing images, text, shapes.
- Where does this code go?
- Lots of different ways of displaying the information - we could animate it for example.

User Input

- In the outline program, you could just use `keyPressed()` and use the keyboard to invoke the different queries.
- After this could use something like Widget and widgetList classes from Week 6 to handle input.
- Could interact with the rendering itself, but this is more challenging.
- Various GUI libraries for processing are available. Might not be worth the effort for just a few buttons.

Main Program (Pseudo-code)

- Setup

```
read_in_the_file();
result = default_query();
current_query = query3;// whatever type of query is default
```

- Draw

```
switch(current_query){
    case query1:
        render_query1(results);
        break;
    case query2:
        Etc…..
    }
render_controls();
```

Main Program (Pseudo-code)

- mousePressed()
  - Work out which button pressed

```
switch(event)
    case button 1:
        current_query = query1;
        results=query1();
        break;
    case button 2:
        current_query = query2;
        results=query2();
    Etc.
```

  - You may need several “results” variables for different types of data returned by different queries.

Assessment

- The assessment will focus on:
  - the demonstration of working code
  - a check on code authorship
  - a check on quality of code and documentation
  - progress towards the overall goal of the project
  - the features implemented.

Source/Version/Revision Control

- Need some form of revision control to keep track of the program source code.
- We have a *repository* or server where the code is stored.
- Team members use a *client* to download the current version.
- They work on it, add files, make edits etc.
- Upload the changes so that other team members can work on the new version.
- Weekly code snapshot to be uploaded every week by Friday 4pm. THERE ARE MARKS FOR PROGRESS BASED ON THIS CODE.

Parallel edits

- Need to be careful not to overwrite other people's work.
- Let's say Jane and Sean both download (checkout) the current version at 6pm, and start editing.
- At 7.30pm Jane finishes, and uploads her changes.
- At 7.35pm Sean finishes, and not realising that Jane has made changes, uploads his changes, overwriting Jane's changes.
- Bad.

Merging: different files

- Instead, imagine Jane and Sean are using a revision control system.
- Case 1: Sean and Jane have worked on different files in the project. When Sean uploads (commits) his changes he is told about the changes made by Jane, but since there is no clash, he can download the changed files (update), check that the merged version still compiles and works, then upload his changes.

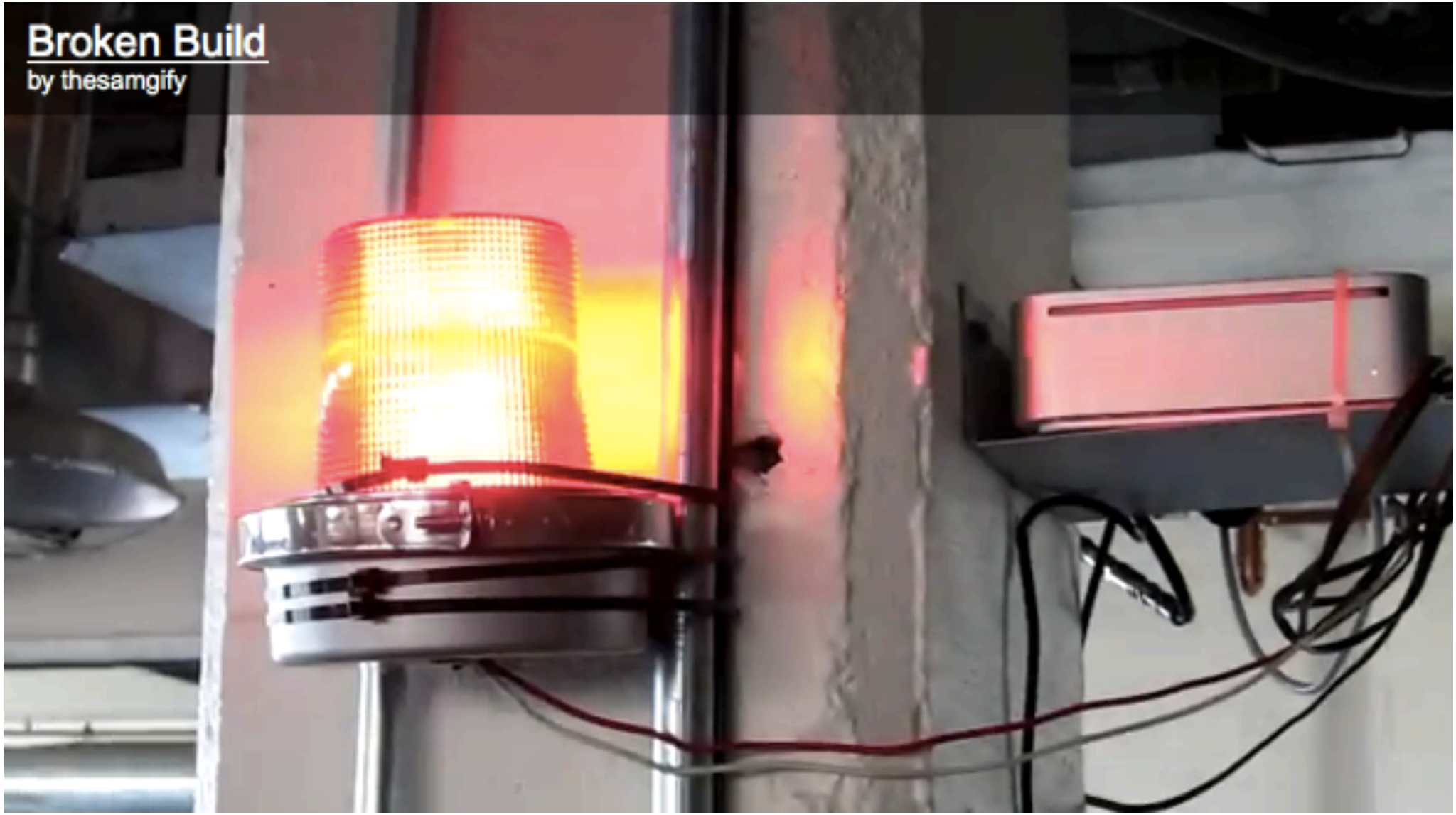
Merging: same file

- Case 2: Sean and Jane have worked on the same file. In this case, Sean is warned about the conflict when he tries to upload (commit) his changes. He can use the special merge editor which allows him to integrate both his own and Jane's changes, compile and run them, before uploading (committing) a new version.
- This approach to version control is called copy-modify-merge.

More features

- You can retrieve old versions of the project using the version control system – everything is stored.
- “Blame” command to find who was the last person to change the code and what they changed.
- Many companies have a machine which continually builds the code and runs tests. If the screen changes it means someone has committed code without compiling and testing it (Bad).
- Can get log of all the changes that have been made, and who made them, at any point in the project.

CS1013 Programming Project



Getting started with Subversion

- Check out your repository (right click on desktop).
- Enter URL (as below), username and password.
- Add files to directory
- Right click on files->TortoiseSVN->add
- Commit your files when done editing. (Right click on file or directory->SVN commit).
- Use SVN update if you just want to check for changes made by others.

- https://subversion.scss.tcd.ie/CS1013-1718-1/
  - for group 1
- https://subversion.scss.tcd.ie/CS1013-1718-2/
  - for group 2

This week's lab

- Create some test data using a text editor or excel (or download from course website).
- Call a method in setup which reads in the data.
- Initially just use `println()` to verify each line is being read in properly.
- Create a class to store the datapoints – each instance storing the data from one line of the file.
- Create an `ArrayList` of instances of this class.
- Create a loop which prints out all of the instances using `println()`
- Print out the instances on the screen in a nice font using `text()`

Demonstration goals

- Show the demonstrator your test data
- Show the code that loads the data- e.g. using `loadStrings()`, that puts the data from each line into instances of a class where each piece of data (rating, business name etc.) is in a different variable.
  1. Print out an instance of the class which has been read in from the file, using `println()`.
  2. Print out all of the instances of the class which have been read in from the file, using `println()`.
  3. Do the same as above but using `text()` and a font.
  4. Commit a version to your SVN repository.

Demonstration goals

- Tell your demonstrator:
  - How your group are organising yourselves.
  - When you are meeting to work on the project.
  - Who is doing what over the next week.
  - Your first group meeting should happen before the lab.
- Talk to your demonstrator about the design for your project. What will it look like?
  - Sketch out some designs on paper, discuss at your group meeting and bring them to the lab.

loadStrings()

Reads the contents of a file or url and creates a String array of its individual lines. If a file is specified, it must be located in the sketch's "data" directory/folder.

```
String lines[] = loadStrings("list.txt");
println("there are " + lines.length + " lines");
for (int i=0; i < lines.length; i++) {
  println(lines[i]);
}
```

(From processing.org)

You will need String.charAt() to go through the characters in the string.