CS1013 – Programming Project

Dr. Gavin Doherty
ORI LG.19
Gavin.Doherty@cs.tcd.ie

Trinity College Dublin

DSG Distributed Systems Group

Debugging

- Golden rule: change ***one*** thing at a time.
- Understand the errors being reported.
- Use `println()` to print out values of important variables. Print out the fact that you have reached certain points in the program.
- Always check whether an object returned from a method is *`null`* before using it – program defensively.
- When accessing an array, always make sure that the index you are using is within the size of the array.
- Get to know the debugging tools when you’ve got a fully functioned IDE.

Coding conventions

- Code conventions are important to programmers for a number of reasons:
- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
- If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

From “Code conventions for the Java programming language”

See also: https://google.github.io/styleguide/javaguide.html

Documenting code

- Many coding standards out there, Oracle have a set for Java. Below are some recommendations from Scott Ambler.
- If your program isn’t worth documenting, it probably isn’t worth running.
- Good comments will come back to help you!
- Document *why* something is being done, not just what.
- Member functions – strong, active verb
- is/get/set for access/update.

Variable names

- In Java, variable names, values which change should be short but meaningful:
  - Example: int testScore = 80;
- Use a full English descriptor:
  - **Which is best? testScore = 12; OR tsc = 12; OR t = 12;**
- In Java, constants, values that do not change, are typically implemented as *static final fields of classes*, *CAPITALISED with spaces:*
  - Example: static final int MONTHS_IN_YEAR = 12;
- For names of components (interface widgets) you can use a full english descriptor postfixed by the widget type:
  - Example: okButton !button456, cancelButton, applyButton

Naming Collections (Arrays)

Should be given a plural name representing the types of objects stored by the array.

- The name should be a full English descriptor with the first letter of all non-initial words capitalized.

- –Examples: customers, orderItems, myCircles, theAliens.
- Try to only access fields of objects using get/set.

Function header

- What and why the member function does what it does.
- What a member function must be passed as parameters.
- What a member function returns.
- Known bugs.
- Any exceptions that a member function throws.
- Include a history of any code changes.
- Examples of how to invoke the member function if appropriate.
- Use whitespace in your code.

Writing files

- Declare a PrintWriter variable:

```
PrintWriter output;
```

- Create a PrintWriter object

```
output = createWriter("delays.csv");
```

- Write to the file

```
output.println(“some text”);
```

- Flush the output (make sure everything is actually written to the disk).

```
output.flush();
```

- Close the file.

```
output.close();
```

Demonstration points for this week

- Should be able to select all of the different query types using the UI and have the query appear.
- Should be able to change the parameters to the query, eg.
  - type the name of the business or select it from a list.
  - select a date range
  - view the highest rated businesses or users.
- Demonstrate 3 types of user interaction.

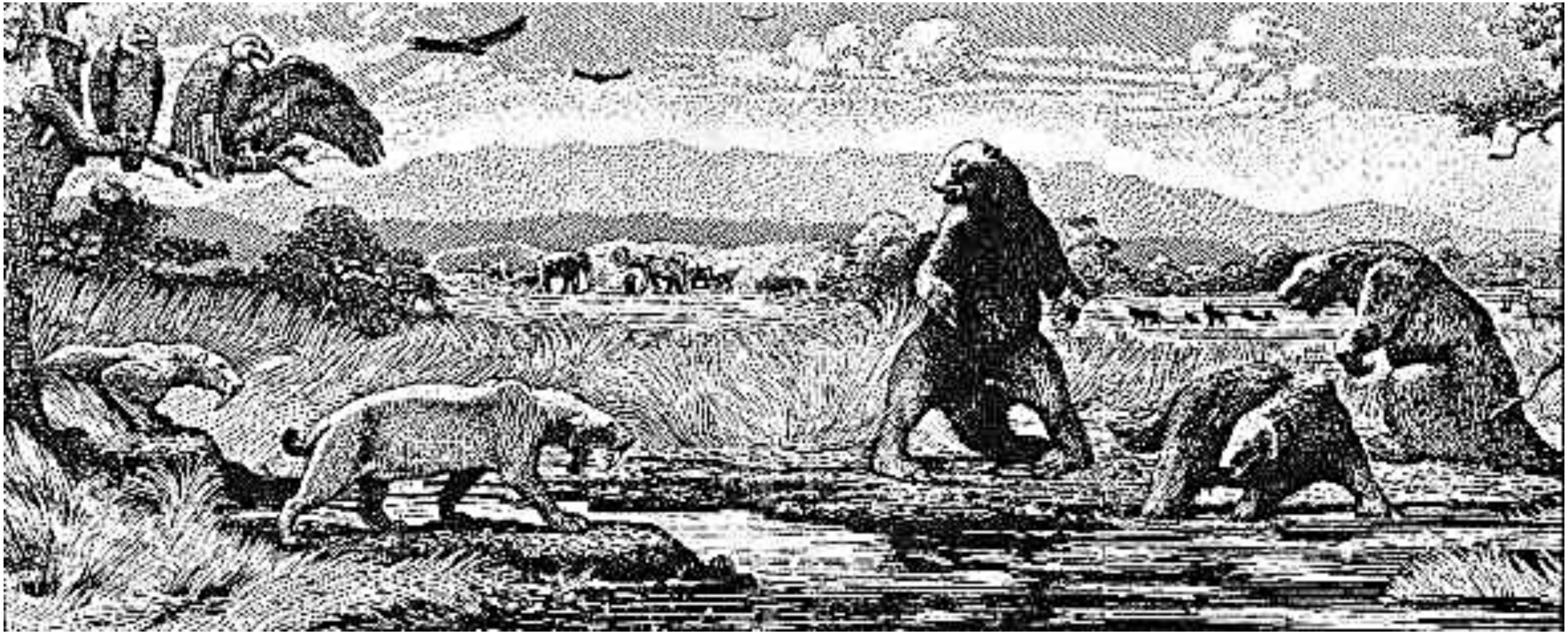
Review - direct

- How to produce output on the screen and have it change.
- Dealing with lots of objects using arrays and lists.
- Structuring larger programs using a number of classes.
- How to take input from the user and use it to change what appears on the screen (updating variables, invoking methods).
- Structure of visualisation programs: load in data, handle user input, run queries, display results, navigate within results.

Review - indirect

- Important to produce readable and comprehensible code, commenting code, using consistent naming conventions.
- Using a revision control system to manage group software projects.
- Working as a team on software projects is a skill in itself.
- Planning in advance makes teamwork on software projects run a lot smoother.

The tar pit...



The mythical man month

- Fred P. Brooks. 1979, 1995.
- Based on IBM OS 360 experiences
  - 5000 person-years of effort
  - Introduced 1963, completed in 1968
- No body of knowledge, no professionals, no mass market, no high level languages.
- Large system development is a tar pit
- A multitude of small problems slow you to a crawl.
- Q: How does a system get to be a year late?
  - *A: One day at a time*

Myths and fallacies

- Poor estimation
  - Assumes nothing will go wrong
  - Hard to know all in advance
  - Probability of success in ***every*** step is small.
  - Most measures confuse effort with progress.
- Person-month
  - Throwing more people at a task which is behind schedule will often make progress slower.
  - Communication and training.
- Not planning to test
  - Many projects on schedule until testing phase. Not budgeted.
- Gutless estimating.
  - Need to learn how to give bad news
  - Need to learn when to tell the client “no”

Programming teams

- Cost does indeed vary as the product of the number of people and the number of months. Progress does not!
- The unit of the person-month implies that people and months are interchangeable
- However, this is only true when a task can be partitioned among many workers with no communication among them!
- When a task is sequential, more effort does not necessarily improve schedule.
- Many tasks in software engineering have sequential constraints.

Programming in teams

- Most applications are much too big to tackle alone
  - Too complex to analyse, too big to design, too much programming
  - One person doesn't have the monopoly on good ideas – however talented they are
- Increasing scope for confusion
  - Decisions aren't fully shared, people aren't notified of changes, …
  - Not everyone understands the issues or ramifications of a decision
  - Difficult to achieve unanimity of design or coding styles
  - "Experts" will disagree on the "right" approach

Communication

- “How, then, shall teams communicate with one another? In as many ways as possible”.
- Informally
- Meetings
- Logs & Tools

Project - Code

- Comments
- Indicate authorship and changes at the top of every source file.
- Our project: Must have everything needed to run. Check this by checking out the repository to a lab machine and trying to run it.

Project presentation

- Demonstrations will be 4pm-7pm Thursday 5th in Regent House. BE THERE ON TIME.
- You have 5 minutes to present the features of your program (what you did, what is good about your design) in the demonstration.
- The final version for marking will be downloaded from subversion at 5pm Friday.
- You should have your presentation and demo on a USB key as a precaution.
- If your group does not have a laptop to present with TALK TO ME AFTER THIS LECTURE.
- As your slot approaches please have the demonstration machine ready to run, and make sure you know how to connect it to a projector – I suggest doing this after this lecture.

Project report

- Outline of design
- How you split up the work and organised the team
- Features implemented
- Problems encountered
- Your report should be in PDF or DOC (word) format, 5 pages MAXIMUM, and should be uploaded to subversion as
  CS1013-report-x.pdf
  - where [x] is your group number
- Have until Friday 6th at 4pm to submit report.

Demonstration

- We have a VERY tight schedule, so your demo must take no more than 5 minutes total.
- Move to the front in advance of your presentation.
- Decide in advance who is speaking + demonstrating.
- Have your laptop ready and make sure it can connect to the projector beforehand.
- If your program takes a while to start, please start it running when you begin your talk.