Introduction and Overview

Software Design and Programming

KLM

Department of Computer Science and Information Systems
Birkbeck, University of London

keith@dcs.bbk.ac.uk

Spring term 2016

Welcome to the module

Course materials available on
`http://moodle.bbk.ac.uk`

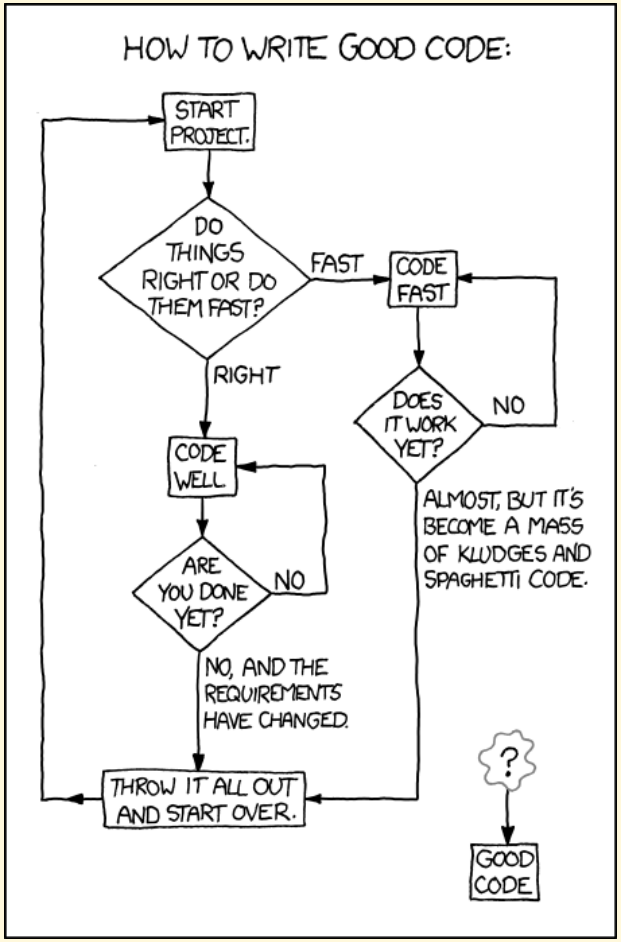
What is this course about?

A combination of

- Software engineering techniques:
  - Design Patterns,
  - Modelling notation and process (UML & UP)
- Object Oriented Programming,
- Functional Programming, and
- *futures* — bleeding-edge programming features

How to write good code? I

This class teaches a style of software design that can help you reach the box labelled *Good Code*



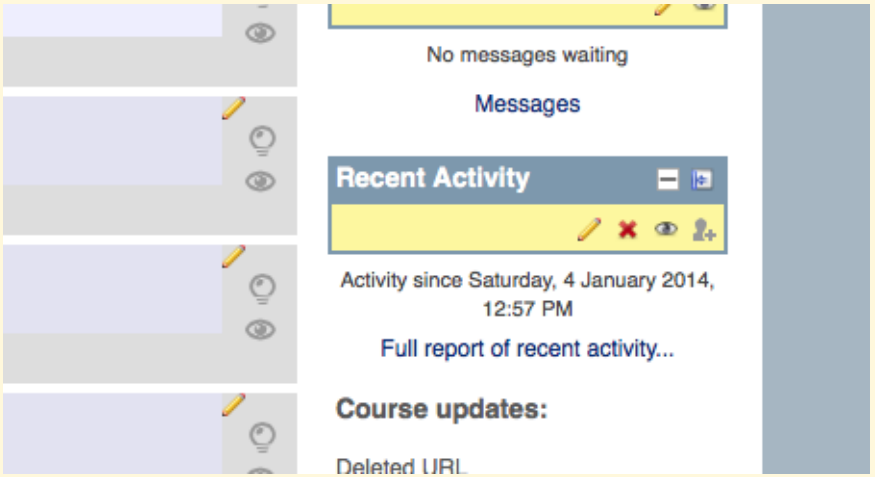
From the web comic, **xkcd**: `http://xkcd.com/844/`

How to write good code? II

Software Design is not completely a black art. . . there are design techniques that lead to better results when applied in support of creative expression

Check the website regularly. . .

- To make it easy for you to track updates there is a side panel which indicates the recent changes to the site. . .



- The Moodle website is your source for the class schedule, homework assignments, announcements, etc.

Teaching Philosophy I

- We want you to participate!
  - Feel free to interrupt me when you have a question
  - Feel free to tell me to slow down if I'm speaking too fast, muttering, or just plain confusing
- I **will not** learn your name because my mind doesn't work that way — Sorry!

Teaching Philosophy II

- *Learning by Doing* (the PiJ philosophy)
  - Most sessions have a video lecture to view prior to the session and some notes/links to follow.
  - We will try to create discussion points for each lecture and will also insert in-class activities where appropriate
  - The exercises and homework will ask you to apply techniques learned in class

Goals of the Class

- Provide students with knowledge and skills in both Object-Oriented and Functional:
  - concepts
  - analysis, design and implementation techniques
  - design methods (software life cycles)
- Students should view software development as a software engineering process that has well-defined stages with each stage requiring specific tools and techniques

Course Structure (Tentative)

Topics
The object model and how it is realised in various object-oriented languages (e.g., Java, Groovy, and C++)
The use of an Integrated Development Environment (IDE) for software development: e.g., editing, debugging, compilation, etc. Modularity, versioning and packaging for the JVM
Further development of the ideas of inheritance and polymorphism (including a revision of parametric polymorphism)
Graphical User Interface programming through JavaFX
Data persistence using Java
Java language features: inner classes, closures, etc. — is there a better language on the JVM?
The SOLID (Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversion) approach to object oriented programming
Meta-protocols in Groovy (and Ruby)
An introduction to Design Patterns and Anti-Patterns and their application to object-oriented software design
An introduction to functional and polyglot programming through the Scala programming language
Code refactoring and analysis
More Scala and distributed computing through actors in Akka

Discussion I

- How many people have used an object-oriented programming language before?
  - Java?
  - C#?
  - C++?
  - Objective-C?
  - Scala?
  - Python?
  - Ruby?
  - Others?
- What features make a language *object-based*?
- What features make a language *prototype-based*?
- What features make a language *object-oriented*?
- What about functional?

Discussion II

How many people are comfortable starting from scratch and creating:

- a script?
- a desktop application?
- a web service?
- a mobile application?
- a system of systems? (i.e. desktop plus web service)
- a database-backed application?

Discussion III

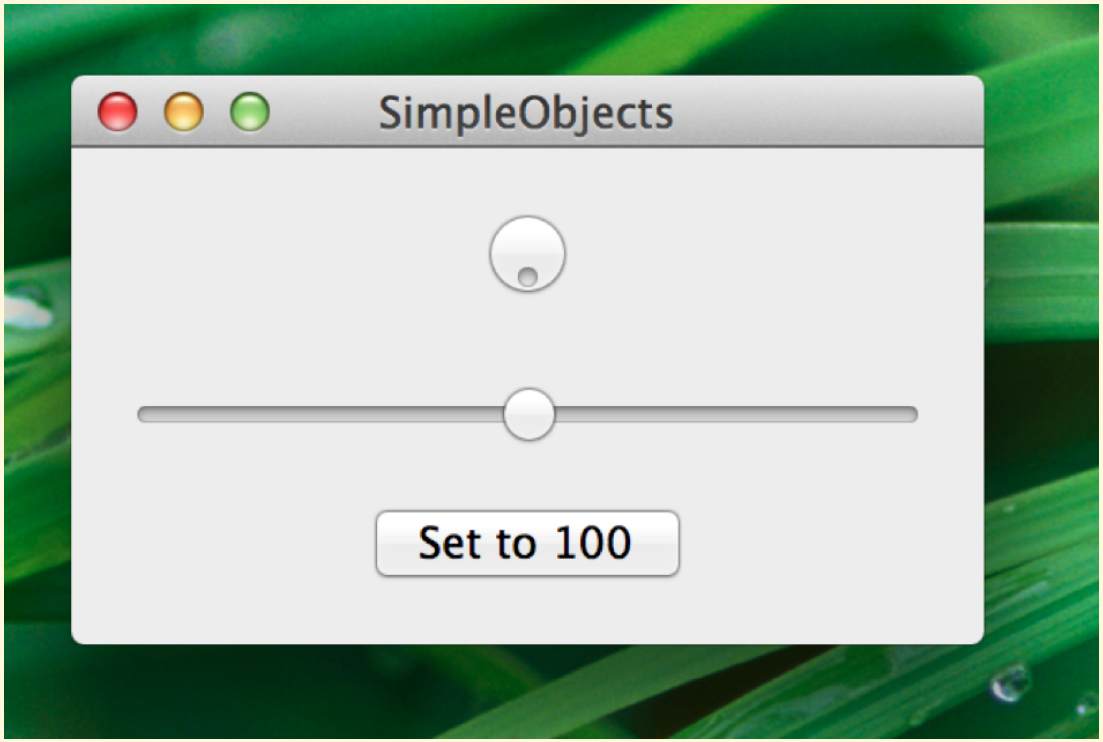
- When you create a program from scratch:
  - do you use OO techniques?
  - OO design heuristics?
  - design patterns?
- If not, what style of software design do you use?
- What styles of software design are you aware of?

Discussion IV

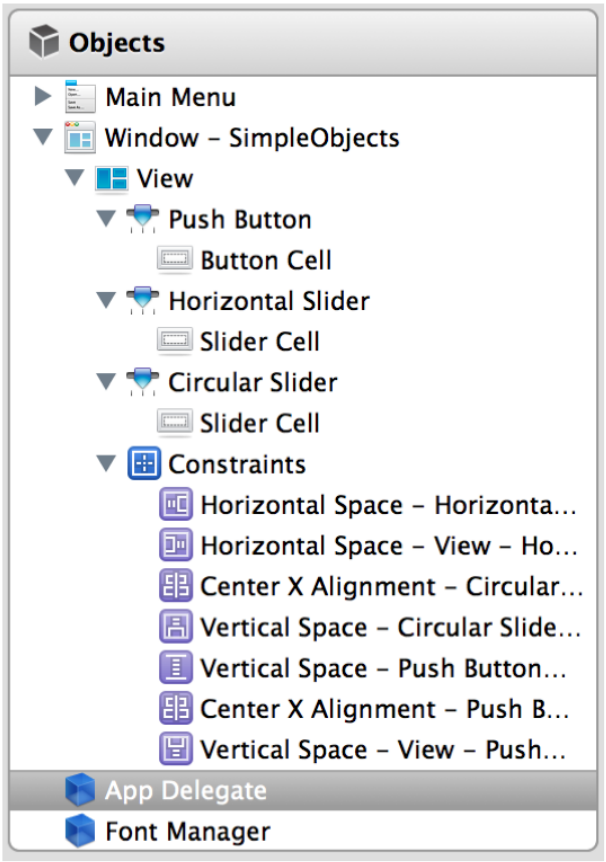
- What is design?
- What comes before design?
- What comes after design?
- Do these questions make sense in software development?
- What would make the process of software design object-oriented?

Discussion V

How many objects do you think are working together to create the application shown on the right?



. . . a lot!



19 objects + AppDelegate = 20 objects

(Let's ignore what's hiding in the Main Menu object...)

A bit of history about this module...

The module used to behave like two sub-courses

- Object-Oriented Programming
  11 lectures – first part of term
- Object-Oriented Design
  11 lectures second part of term
- with shared examination and coursework

but thanks to student feedback we have improved the integration of the materials — end result— hopefully a better user experience

Has the Module changed since last year?

Simple answer — **YES**

- We are always updating our material and syllabus to include changes that happen in industry and academia.

A (Very) Brief History of Object-Oriented everything ☺ I

- All the major concepts were developed in the 1960s as part of a language called Simula
- Alan Kay and his group developed a programming language named Smalltalk in the late 1960s and early 1970s
- In the early 1980s Bjarne Stroustrup developed an extension to the C language that eventually evolved to the language C++
- Explosion of the research in object-oriented programming techniques began

A (Very) Brief History of Object-Oriented everything ☺ II

- In the first major conference on object-oriented programming, in 1986, there were dozens of languages
- These included Eiffel, **Objective-C**, Actor, Object Pascal, and various Lisp dialects
- In the 1990s Object-oriented programming became mainstream and then Java happened
- Then even Microsoft caught the bug (C#)
- . . . and now we are now quite so sure. . . *Polyglot* languages

What are you getting yourself into?

- Programming, and software engineering, is intellectually challenging
  - It can be tremendous fun!
    If you like that sort of thing...
- Lifelong learning is essential
  - The technology is constantly changing
  - We cannot teach you all you need to know
  - We can point you in the right direction and give you a good, hard push — but the rest is up to you!

Elegance in programming I

Consider the following problem:

- You are given a stack of cards, supposedly containing the numbers 1 through 100...
- ...but there are only 99 cards
- How do you determine which card is missing?

Elegance in programming II

One possible solution:

- Go through all the cards looking for 1, then do it again looking for 2, etc.
- But this is inefficient!
- Is there a *better* way?

That is the type of question we wish to address in this course

Changes in computer science

- Computer science is only about 60 years old
- Change is rapid and accelerating
- Dominant language of the 1990s: C++
- Dominant language of early 2000s: Java
- Dominant company: IBM to Microsoft to Google to Apple to ...
- First GUI: Macintosh, 1984
- First web browser: Mosaic, 1992
- Web pages: HTML to DHTML to XML to ...
- ...

Topics we will try to cover. . . I

- The software development process
- Principles of programming and programming languages
- The object model and how it is realised in various object-oriented languages
  (e.g., Java, Groovy, Ruby, Scala, C++, . . .)
- Further development of the ideas of inheritance and polymorphism (including a revision of Generics)

Topics we will try to cover... II

- Consolidation of Java language features: inner classes, annotations, closures, etc. Project Lombok, Java 9...
- The rise of *Functional Programming* languages
- The use of an Integrated Development Environment (IDE): e.g., editing, debugging, compilation, etc.
- Modularity, versioning and packaging (e.g., OSGi, Project Jigsaw)

Topics we will try to cover... III

- An introduction to Design Patterns and Anti-Patterns
- Interface separation and Dependency Injection
- Code refactoring and code analysis
- process based programming (e.g., *actors* as an alternative to *threads*)
- Graphical User Interfaces in Java
  (e.g., Swing and JavaFX)
- Persistence in object-oriented languages

Is programming all there is?

No — programming isn't just about language features and topics it is also about

- tooling,
- infrastructure, and
- networking
- . . . (amongst other topics)

Small projects

You can build a kennel in a few hours

- You don't need a blueprint
- The materials don't cost much
- A little knowledge of tools is enough
- Imperfections are no big deal

Medium-sized projects

You can build a house in a year or so (maybe less, maybe more)

- You really do need blueprints (design patterns?)
- Excess materials mean wasted money

- House building requires more skills: plumbing, bricklaying, electrical work, carpentry, etc.
- Imperfections matter: you don't want a leaky roof!
- It's easier if you aren't doing it all by yourself

Large projects

You cannot build a skyscraper by yourself

- It's just too much work for one person
- You don't have the money
- You don't have all the skills
- Imperfections could be costly or even fatal

Such systems can only be built by a team

- Communication is essential
- A "paper trail" is essential

What does that mean for you in this course?

- What can we ask you to build for this module?
- What will be expected of you in industry?
- We teach skyscraper-level skills, but
  - we ask you to apply those skills to kennel sized problems
  - it's silly, but what alternative do we have?
- It's up to you: When you leave here,
  - will you be able to build skyscrapers?
  - or will you just be very good at building kennels?
  - I know what I'd prefer!

Pre-requisites

Java, Java, Java, Java, ...

Java as a Programming Language

The Java programming language

- has been a huge success but it is showing its age
- has had a stuttering development cycle (had not evolved significantly since JDK5 (2004) although Java 8 has significantly changed that perspective)
- is verbose
- lacks modern language features
  - closures, meta-programming, DSLs, fully functional-programming constructs encouraging *no side-effect* programming, operator overloading, regular expressions as a *first class citizen*, a single type system, etc.
  - JDK9 is being delayed again ...

A brief history of Java

- Originally called **Oak** and was developed in 1991 by James Gosling at Sun MicroSystems
- Intended as a language for use in embedded customer electronic applications
- This determined the characteristics of the language
- Two of the most important features *size* and *reliability*
- Processors in embedded systems are very small, possessing small memory, thus the language must be able to translate into very concise encoding
- Embedded systems should almost never fail and should respond to exceptional and erroneous conditions

Java as a Platform (JVM)

- The JVM is proven to be a great run-time platform
  - Secure, highly performing, mature, etc.
- We need a better programming language for the JVM (or maybe a combination of languages like .NET)
  - Runs on the JVM
  - More productive, more fun, less verbose syntax
  - With modern language features
  - Seamless interoperability with Java programs
- Viable choices
  - Groovy
  - Scala
  - JRuby
  - Clojure

Textbooks - do you really want one?

See the list on the Moodle *shared resources* site

Software. . .

- The Java Development Kit (JDK) — preferably version 1.8 (although we may also venture into Java 9 as well)
- The documentation for the JDK
- The Integrated Development Environment (IDE) of your choice
  - you can use the command line if you really want to
  - we're not recommending one as *one size* ***does not*** *fit all*!
- we will discuss other software as, and when, required (e.g., Scala and Groovy)

Practical work

Everyone is expected to attend lab, but...

- If you are already a programmer, and you understand the material on the worksheet, then you probably do not need to attend
- No new material will be presented in the lab sessions; some demos and tooling, but no new language features
- The "lab sheets" can be done at home — but need to be completed before the next session

Course Evaluation I

This depends on whether you are taking the undergraduate or postgraduate version of the module

Postgraduate

- By two hour written examination (80%)
- By practical coursework (20%)

Undergraduate

- By two hour written examination (75%)
- By practical coursework (25%)

Course Evaluation II

The practical coursework consists of a *portfolio* of work comprising several programming assignments submitted as one portfolio and the exercises for the module

- an individual piece of work
- one carried out as a pair
- one carried out as a group
- the exercises for the module

As always, see Moodle for further details.

Course Evaluation III

There are also supplemental exercises which you should also attempt

- which are not assessed,
- for which we provide outline solutions

so therefore you can judge your own progress!

Course Evaluation IV

oh, and unless I forget...

- Submission will be via Moodle using a Github repository
- The late policy is as stated in the degree handbook;
- Coursework portfolio due by ...I forget (check Moodle)
- Cutoff date is two weeks after the due date

How to get a good grade? I

How to get a good grade? II

- Start your assignments early!
  - This is the first and most important way to improve your grade
  - Programming takes a lot of time
  - Its not easy to predict how long a program will take
- Do as many exercises as you can (oh, and don't look at the answers, if they are provided, until you've had a good go at the problem)

How to get a good grade? III

- Test your programs thoroughly
  - One or two simple tests are not enough
  - We often provide simple but incomplete tests, just to get you started
  - We will do thorough testing, even if you dont!
- Read the assignments carefully
  - Do what is assigned, not *something like* what is assigned

How to get a good grade? IV

- Learn to use your tools (e.g., eclipse, IntelliJ, JUnit, FindBugs, Maven, Lombok, etc.)
- Use comments and good style right from the beginning, not as a last-minute addition
- Review and understand the (video) lectures

Who to ask when it all goes horribly wrong!

If you have questions about the course:

- If you are doing pair programming, first ask your partner
- Check with a teaching assistant, if they are available
- Make use of the forums
- Ask or email Keith (not good on the phone)

Keith and email. . .

Feel free to send me email at:

`keith@dcs.bbk.ac.uk`

I get lots of spam, including several virus-carrying messages a week, so I run several filters.

To avoid my spam and virus filters so please put **SDP 2016** or **SP3 2016** somewhere in the subject line

The End