Introduction and Overview

Software Design and Programming
Software and Programming III

KLM

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

keith@dcs.bbk.ac.uk

Spring term 2017

Welcome to the module

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

What is this course about? I

What is this course about? II

Part I

- Java — fill in the blanks (what you don't know and need to for this module). Inner classes, Java 8 features, etc.
- Scala — the basics to proficient
- SOLID — Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversion
- Meta object protocols
- Reflection

What is this course about? III

Part II

- Functional approach to programming compared to Object-Oriented
- Higher-order functions, currying and closures
- Further functional programming

What is this course about? IV

Part III

- Introduction to Design Patterns — The GoF patterns ++
  The examples will be in Java and Scala

What is this course about? V

Part IV

- Concurrent and distributed programming — through Akka and actors

What is this course about? VI

Part V

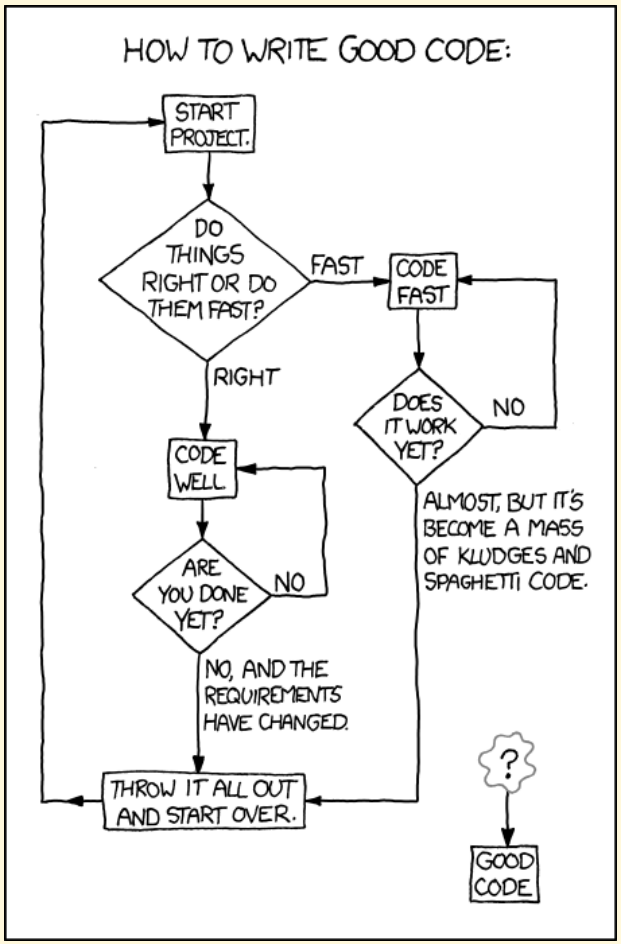
- Databases — JDBC, JPA, Slick, etc.
- GUI — JavaFX/ScalaFX and Swing
- Reactive — RxJava/RxScala

What is this course about? VII

- We will also cover appropriate *build* tools, e.g., Gradle and SBT
- We will presume that you will cover the IDE aspects of tooling

How to write good code? I

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



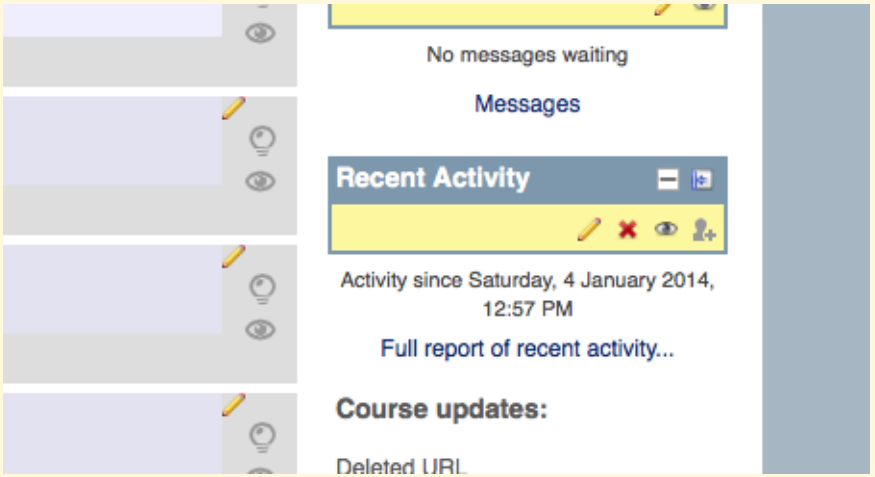
From the web comic, **xkcd**: `http://xkcd.com/844/`

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

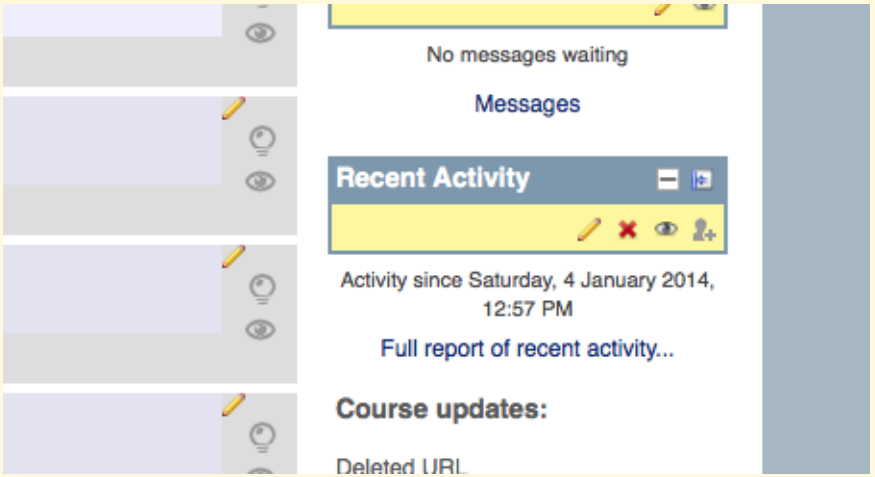
Check the website regularly. . .

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



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!

Teaching Philosophy II

- *Learning by Doing* (the PiJ philosophy)

Goals of the Class

- Provide students with knowledge and skills in both Object-Oriented and Functional programming models:

Goals of the Class

- Provide students with knowledge and skills in both Object-Oriented and Functional programming models:
  - concepts

Goals of the Class

- Provide students with knowledge and skills in both Object-Oriented and Functional programming models:
  - concepts
  - analysis, design and implementation techniques

Goals of the Class

- Provide students with knowledge and skills in both Object-Oriented and Functional programming models:
  - concepts
  - analysis, design and implementation techniques
  - design methods (software life cycles)

Goals of the Class

- Provide students with knowledge and skills in both Object-Oriented and Functional programming models:
  - 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

Discussion I

- How many people have used an object-oriented programming language before?

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

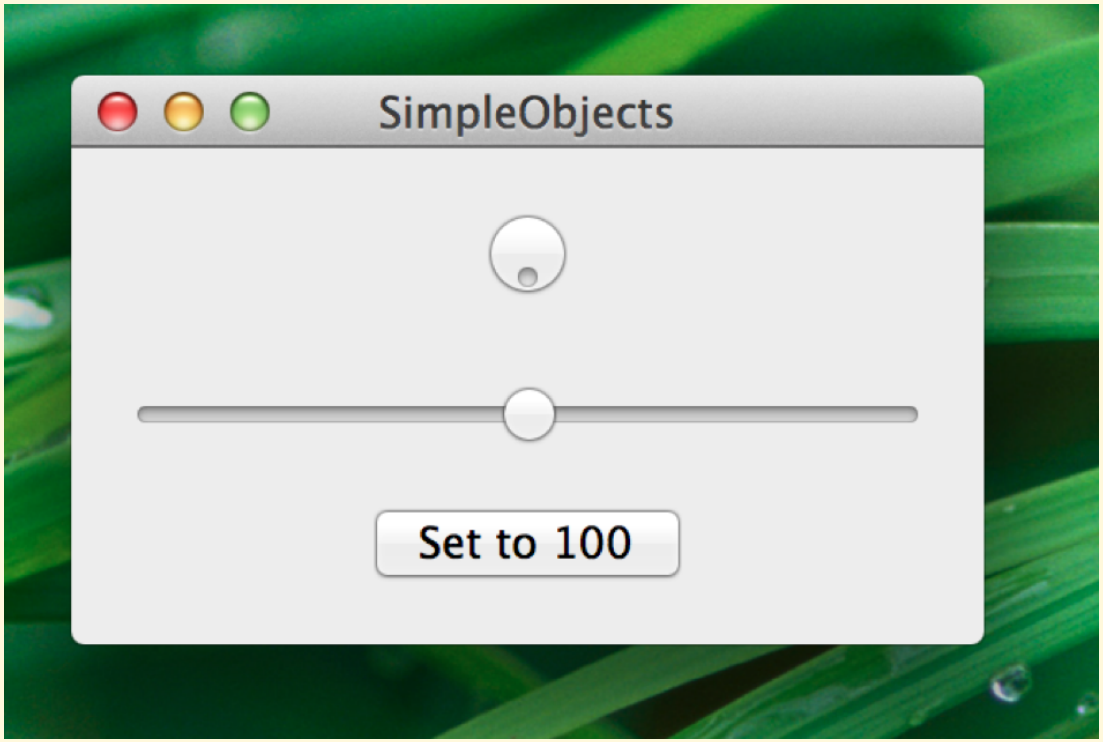
- When you create a program from scratch:
  - do you use OO techniques?
  - OO design heuristics?
  - design patterns?
- If not, what style of software design do you use?
- What styles of software design are you aware of?

Discussion IV

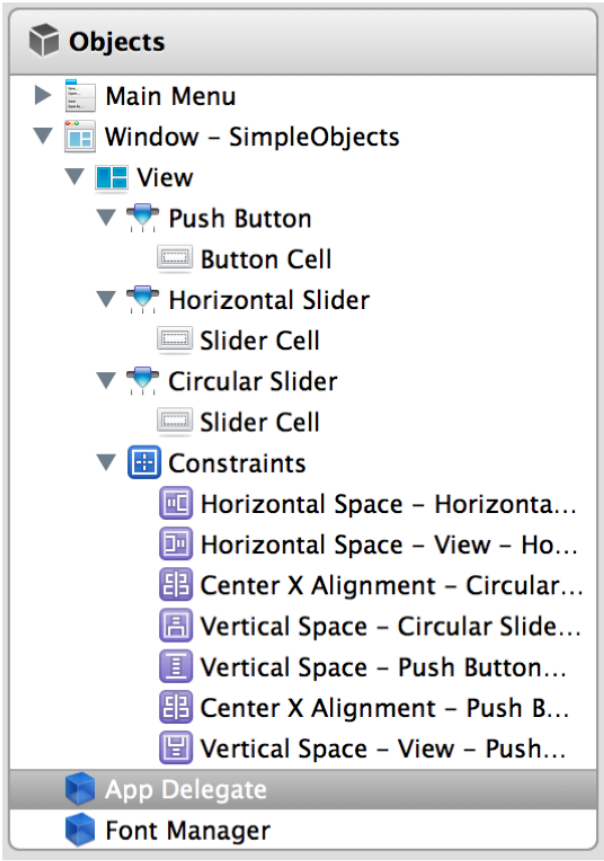
- What is design?
- What comes before design?
- What comes after design?
- Do these questions make sense in software development?
- What would make the process of software design object-oriented?

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

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

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

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

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

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

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

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?

Has the Module changed since last year?

Simple answer

Has the Module changed since last year?

Simple answer — **YES**

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

What are you getting yourself into?

- Programming, and software engineering, is intellectually challenging
  - It can be tremendous fun!

What are you getting yourself into?

- Programming, and software engineering, is intellectually challenging
  - It can be tremendous fun!

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

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

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

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

Changes in computer science

- Computer science is only about 60 years old
- Change is rapid and accelerating

Changes in computer science

- Computer science is only about 60 years old
- Change is rapid and accelerating
- Dominant language of the 1990s: C++

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

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 ...

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

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

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 ...

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?

Is programming all there is?

No

Is programming all there is?

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

Is programming all there is?

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

- tooling,

Is programming all there is?

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

- tooling,
- infrastructure, and

Is programming all there is?

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

- tooling,
- infrastructure, and
- networking

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

Small projects

You can build a kennel in a few hours

- You don't need a blueprint

Small projects

You can build a kennel in a few hours

- You don't need a blueprint
- The materials don't cost much

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

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)

Medium-sized projects

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

- You really do need blueprints (design patterns?)

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

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.

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!

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

Large projects

You cannot build a skyscraper by yourself

- It's just too much work for one person

Large projects

You cannot build a skyscraper by yourself

- It's just too much work for one person
- You don't have the money

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

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

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

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

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

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 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?

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

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

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?

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,

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?

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?

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

Pre-requisites

Java,

Pre-requisites

Java, Java,

Pre-requisites

Java, Java, Java,

Pre-requisites

Java, Java, Java, Java,

Pre-requisites

Java, Java, Java, Java, ...
OR

Pre-requisites

Java, Java, Java, Java, ...
OR

Pre-requisites

Java, Java, Java, Java, ...

OR

Scala,

Pre-requisites

Java, Java, Java, Java, ...

OR

Scala, Scala,

Pre-requisites

Java, Java, Java, Java, ...

OR

Scala, Scala, Scala,

Pre-requisites

Java, Java, Java, Java, ...

OR

Scala, Scala, Scala, Scala,

Pre-requisites

Java, Java, Java, Java, ...

OR

Scala, Scala, Scala, Scala, ...

via the primer (or any other means)

Java as a Programming Language

The Java programming language

Java as a Programming Language

The Java programming language

- has been a huge success but it is showing its age

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)

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

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

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.

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

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

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

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*

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

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

Java as a Platform (JVM)

- The JVM is proven to be a great run-time platform
  - Secure, highly performing, mature, etc.

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)

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

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

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

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

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

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

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

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

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)

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

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

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

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*!

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

Practical work

Everyone is expected to attend lab, but...

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

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

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:

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

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

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

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 2017** or **SP3 2017** somewhere in the subject line

The End