ECS510
Algorithms and Data Structures in an Object Oriented Framework

“ADSOOF”

Matthew Huntbach

matthew.huntbach@qmul.ac.uk

Big Title - What does it mean?

- Algorithms - ways of doing things
- Data Structures - ways of holding collections of things
- Object Oriented - a way of structuring programs, supported by languages like Java

Algorithms and Data Structures

- Traditionally, the second core module of a Computer Science degree – “CS2”
- Low level - close to actual representation on hardware
- Assumption is you will program your own algorithms and data structures

Object Oriented Programming

- Introduces an additional layer of complexity to computer programming languages
- The complexity helps us structure programs better
- Emphasis on re-use - we are no longer writing all our code from scratch
- Common algorithms and data structures given in code libraries

So why study ADS?

- Still need to know what's going on underneath
- Good exercise and development of programming skills
- Need to know basic principles of algorithms and data structures in order to build more complex ones
- Algorithms – “the spirit of computing”

OOF

- The “OOF” part of this module is as important as the ADS part
- The underlying idea is dividing a big program into small parts each of which has a coherent and logically separate identity
- Algorithms and data structures are very general
- An Object Oriented Framework means generalised code for algorithms and data structures can be re-used whenever we need it for whatever sort of objects we are working with

What’s new?

- Less encyclopedic approach
- Coverage of using code libraries for algorithms and data structures (Java’s is just an example)
- Consideration of generalising code for re–use, including “generics”
- Think in terms of building “components”: the bottom-up approach of this fits in with top-down approach of Software Engineering

MOST IMPORTANT!

- You can use a piece of code (class, static method) without knowing the code, so long as you know its specification and it works according to its specification
- You can write a piece of code (class, static method) without knowing how it is going to be used, so long as you know its specification and it works according to its specification

Abstraction

- Being able to think and reason in an abstract way is, perhaps, the key skill for a Computer Scientist
- It means being able to view a situation only in terms of its essential aspects
- But you did this in primary school when you learned 1+1=2

Primary school abstraction

- 1 apple + 1 apple = 2 apples
- 1 table + 1 table = 2 tables
- 1 fire engine + 1 fire engine = 2 fire engines
- 1 skjtyyrt + 1 skjtyyrt = 2 skjtyyrts

…

- 1 thing + 1 thing = 2 things

…

- 1 + 1 = 2

Example: becoming more abstract in sorting

- A method to sort an array of 100 integers from lowest to highest
- A method to sort an array of any number of integers from lowest to highest
- A method to sort an indexed collection of any number of integers from lowest to highest
- A method to sort an indexed collection of any type of objects which has its own order
- A method to sort an indexed collection of any type of objects in any order given

Modern programming

- Modern software systems are large and involve gluing together a variety of different aspects: databases, graphics, web links etc.
- Java provides code libraries (APIs) for doing this
- APIs (Application Programming Interfaces) are based on the principles of object orientation
- ADSOOF will develop your understanding of these principles if you work hard at it
- The same principles apply in other object-oriented languages, though the details may differ

Core Programming

- ADSOOF concentrates on the core aspects of programming
- It moves from what the core of the language provides you towards using it to perform particular tasks
- It is fairly abstract, examples tend not to be “real world”, whole point is to be general
- Other 2nd year modules move from core programming to linking other aspects of a full IT system

“Remedial” Programming

- Some aspects of ADSOOF overlap with what you covered in the 1st year
- If you are a confident programmer, don’t be tempted to “slack off”, it’s easy to miss the point where it takes you further
- If you are a less confident programmer, take the opportunity to revise and gain a deeper understanding of topics you were uncertain about first time round

Efficiency

- You may not care how a component works, but you do care that it does its job quickly
- A specification may be met by more than one algorithm or data structure
- Different algorithms and data structures for the same specification may have big differences in efficiency

Two basic approaches to algorithms (problem solving)

- Iterative - start with an initial state, and keep on changing it till you get a solution state.
- Recursive (divide and conquer) - break problem into parts, solve each part, put together to get a solution. If a part is a version of the same problem, we can use the same algorithm to solve it.

Two basic approaches to data structures

- Indexed - the structure is a collection of items, each of which can be accessed in one step from its position in the structure (array)
- Linked - the structure consists of an item and links to further structures (linked list, tree)

If I may be so brash, it has been my humble experience that there are two things traditionally taught in universities as a part of a computer science curriculum which many people just never really fully comprehend: pointers and recursion.

Joel Spolsky

http://www.joelonsoftware.com/articles/ThePerilsofJavaSchools.html

Java

- All examples illustrated using Java
- Some use of Java APIs
- The principles are more important than the details, you won't be expected to memorise and know large numbers of library classes and their methods
- This is a practical programming oriented module
- A more theoretical module would concentrate on proof of correctness and efficiency of algorithms

Programming Principles

- If this were a "Java course", we would start with the Java Collections Framework
- It is a course in programming principles which uses Java for convenience
- Implementing what is found in the Collections Framework will help you understand those principles
- It covers aspects of programming which any Computer Scientist ought to have some familiarity with

Learning v. Memorisation

- The idea that education is about memorising then “re-gurgitating” in exams is common
- It doesn’t work in this subject
- If it has worked for you so far, now may be the point where it stops working
- It is better to understand the principles and from this reconstruct the details rather than try to memorise the details without understanding the principles

Learning by Doing

- Many things are best learnt by doing them - computer programming is a good example
- What seems complicated in theory often becomes simpler once you have tried it in practice
- So reading, trying examples, asking questions, going back to reading, trying more examples … is how to learn this material
- Just reading, especially “revision”, is not a good way to learn this material

Code

- This module provides plenty of code examples
- Most code has a “front-end” to run/test it, be careful to distinguish this from the code which implements algorithms and data structures
- You aren’t expected to memorise code, you are expected to understand it
- Experiment with code to see how it works, modify it, test your understanding to see if what you think should happen is what actually happens

Learning another human language

- Tourist approach - memorise useful sentences and repeat them in appropriate circumstances
- Traditional approach - memorise vocabulary lists and grammar rules
- Theoretical approach - understand the principles behind the grammar rules and vocabulary
- Practical approach - try using the language in practice
- Expert approach - having used the language in practice, go back to the theory and see if the rules make sense from what you have experienced
- Fluent speaker - the rules are so natural you don’t have to think about them, sometimes you need to check them to make sure you haven’t become a sloppy speaker

Asking for help

- Don’t ask too soon - try to work it out for yourself first
- Don’t ask too late - if you read and experiment, but it’s still not working and you don’t know why, there may be something simple you have missed, and asking will get you through it
- Focus your question - try to work out exactly what it is you don’t understand, a question with a short answer is fine, a question where the answer would be to repeat a whole lecture is not

Where to ask for help

- Short focused questions on immediate issues in lectures
- More detailed questions with examples in on-line forum
- Don’t be afraid to ask in public - for every one who asks there are often dozens who want to know the answer to the same question, answering in public means all benefit
- Lab sessions - clarify what is wanted in exercises, ask questions about your own code, ask questions about other module material
- E-mail - good for detailed questions on complex code
- Personal consultation – “chalk and talk” sessions can help, feel free to ask

Debugging Code

- The problem is often not where or what you think it is
- Check the code you are looking at is the code you are running
- It is highly unlikely the problem is with Java
- Insert print statements to narrow down on problem
- Or use debugging facilities of IDE
- Going away and thinking about can be a good strategy
- Random modification of code is a poor strategy
- "Over the shoulder" debugging sometimes works, sometimes doesn't - don't expect an expert always to be able to spot instantly where the problem is

IDEs

- Interactive Development Environments provide support for developing code: writing it, modifying it, debugging it, storing it, understanding its structure
- Use of IDEs is essential for modern programming
- But an expert programmer needs to know how the code works underneath
- ADSOOF examples are small, with a "front end" to test them
- No IDE assumed or needed, BlueJ may help, NetBeans may be too "heavy"

Debugging People

- Much more difficult than debugging code
- Only you know what is inside your head, others may guess by asking you questions or seeing what you do, and also from previous experience
- Learning involves picking up new ideas, but it may also involve “debugging” where you have misunderstood something
- Teaching involves debugging students, but it requires experience and it isn’t easy
- You can debug yourself by testing your hypotheses against what works in practice, and by asking experts

Difficulty

- The second level programming course is traditionally “difficult”
- Why?

Difficulty

- The second level programming course is traditionally "difficult"
- Why?

$\Rightarrow$ It is oriented towards the abstract rather than applications

$\Rightarrow$ Involves some concepts known to be difficult (like pointers and recursion)

- But …

Difficulty

- The second level programming course is traditionally “difficult”
- Why?
  - *Lecturer grumbles* “these students haven’t learnt to program yet” … “they are lazy”
  - *Students grumble* “he’s going too fast, we can’t keep up”

Difficulty

- *Lecturer grumbles* “these students haven’t learnt to program yet” … “they are lazy”
- *Students grumble* “he’s going too fast, we can’t keep up”
- *Lecturer* wants students to learn lots of new and useful stuff
- *Students* want to learn lots (?) of new and useful stuff

Difficulty

- *Lecturer grumbles* “these students haven’t learnt to program yet” … “they are lazy”
- *Students grumble* “he’s going too fast, we can’t keep up”
- *Lecturer* wants students to learn lots of new and useful stuff
- *Students* want to learn lots (?) of new and useful stuff
- *Lecturer* doesn’t want to fail people
- *Students* have competing life pressures

Difficulty

- *Lecturer* maybe finds this stuff easy, needs to be told by students where it’s hard
- *Students grumble* to themselves …
- *Lecturer* if s/he’s good, will have learnt where students find things hard and will want to help them through it
- *Students* can fall into the trap of thinking “I’m the only one who finds this hard”

Solutions: Good Learning Habits

- Learning is not memorisation (again)
- Programming is a “learning by doing” subject
- Regular study, keeping up with the pace of the module, is <u>much</u> better than putting it off as difficult and hoping to catch up later
- Divide your time evenly between modules regardless of easiness/difficulty and different pressures and deadlines
- Try other sources of information

Solutions: Feedback

- Tests are not designed to catch you out, they are designed to give feedback to the lecturer, and to you when they are marked
- Marks for tests which contribute to final mark mean you take them seriously
- Regular lab attendance gives feedback
- Willingness to ask questions, in lectures, labs, on-line forum, email, tutorials gives feedback
- Make use of feedback, if you are getting something wrong, learn from being told that (this applies to the lecturer as well!)

Solutions: Honesty

- If you are struggling, don’t “hide” or pretend you have done the work at home
- Be honest with others about the amount of work you are doing and how easy/difficult you are finding it
- Don’t cheat in assessed work (or unassessed work)
- Working together is good, so long as it really does mean that
- Don’t tell yourself “I will do it tomorrow … next week … next month” when you know you won’t

Summary

- This module covers core aspects of Computer Science, you need to know this material
- It is presented in a way which emphasises and develops practical programming skills
- You will need to work hard at it during term-time, it is not something you can put aside to pick up later in “revision”
- You are offered assistance in various ways with picking up this module material, please make best use of it