Code Standards
Introduction to Testing
Large Scale Programming, 1DL410, autumn 2009
Cons T Åhs
Clear Code

- Code must be easy to understand. Why?
  - Coding is not a *write only* activity - we’ll spend quite a lot of time reading and modifying code, much more than writing new code.
  - In general, you will not be maintaining your own code; somebody else will.
  - In general, you will be maintaining code written by someone else.
  - You will forget the details.
- Code needs to be understood by
  - compilers - the *easy* part; just follow the rules of syntax and semantics and you’re done! The needs of the compiler are very concrete.
  - humans - the *difficult* part; syntax and semantics are too low level - we need to be able to read and understand the code both at a high level (to start with) and a lower level (when we need the details).
- Clear code must form the bridge between the concrete needs of the compiler and the abstract needs of humans.
Clear Code

› Abstract code is easier to understand and modify.
› A good and abstract design will translate to code that is clear.
› The converse is also true, i.e., if your design is bad and not well thought through, it will reflect in code that is opaque and difficult to modify.
› Think before you start to write code!
› Spend more time on thinking about the code than writing it.
Documentation

- Documented code is easier to understand since the documentation, if well written, helps to navigate safely across the bridge from the abstract to the concrete.

- What is well written documentation?
  - Understandable
  - Available on different levels (overview, file, class, method, code)
  - Accessible at the right place.
  - Up to date!

- JavaDoc is an excellent tool that will help you to make the documentation accessible at the right place. The Eclipse IDE can show the JavaDoc for classes and method while you’re wiring code.

- Documentation should be written before and during writing code, not afterwards - it is during the act of writing the code you have the great insights and understanding of what is going on. Then you start to forget..

- Documentation that is written afterwards tends to be too concrete and just describes what’s happening in the code, but not why. In that case, you might as well just read the code. The code never lies.
The written documentation has several different target audiences.

Consider this when writing documentation.

Utopia, Inc. has again produced a piece of software, with non technical persons as end users, put together of pieces of elegantly crafted abstract and reusable components. There is the odd bug, but new features are added all the time.

Audiences:

- end user - needs to know how to use the software, but needs no technical details at all.
- technical managers - technical overview of systems and components, possibly method signatures, but not likely.
- library users, i.e., developers reusing, but not maintaining, specific components - method signatures, technical documentation to the level of the abstraction, but nothing on the inner workings.
- developers, i.e., the person fixing the odd bug and adding new features - technical and gory details, algorithms, quirks, TODOs etc.
Documentation of Code

- For development, the most important documentation is that which is written in conjunction with the code itself.

- It is important to maintain the documentation as well as the code - this is much simpler if it is in the same place, i.e., the same file.

- The written documentation, i.e., comments of different types are called *explicit documentation*.

- There is also *implicit documentation*, i.e., methods of writing code that make it easier to understand.
  - naming of interfaces, classes, methods and variables
  - formatting of code to show logic and structure

- This leads us to Code Conventions.
Code Conventions

- Code that is written according to a set of rules are easier to understand.
- All members of a team, or a whole company, should follow the same set of rules when writing code.
- The result is that it will look like the code has been written by a single individual, which makes it much easier to follow.
- This becomes more apparent when code written by different team members do not follow a convention.
  - Each team member writes differently so when reading the code of another team member you have to start with penetrating a new style - this takes time.
  - When adding or modifying code written using coding conventions that are different from your own, you will have the problem of deciding on which convention to follow. Again, this takes time and focus from the real problem.
  - Reading code (in the same file) that is written according to different conventions is hard to follow.
- Code style is a matter of personal opinion, so there is no single True Way of writing code.
- Don’t spend time arguing about code style - just use one!
Code Conventions

- In this course, you are to follow the code standard recommended by SUN when writing Java programs.


- These form a rather hard set of rules that you are to follow - if you really feel that you need to break them, you have to motivate it well.
  - You are, after all, deliberately making your code less understandable.
Examples of Rules

› Note that these are just examples - all the rules should be followed, not just these.
  › Be sure to read the whole code standard document and follow it.
› Line length - avoid lines longer than 80 characters.
› Line wrapping - rules regarding where and how to break expressions/statements that don’t fit on one line.
› Variable declaration
  › one per line
  › document the variable
  › at the beginning of a block, not inside blocks
  › if possible, initialise the variable when declaring it
› if statements always use braces!
Examples of Rules

- Naming conventions
  - packages - all lowercase
  - classes - nouns, CamelCase
  - interfaces - as classes; nice to have initial I, e.g., IImageSource
  - methods - verbs, camelCase with initial letter lowercase
  - variables - descriptive, camelCase
  - constants - uppercase, words separated with underscore

- Additional considerations
  - Names should not reveal representation, access method or algorithm.
  - Be consistent!
Introduction to Testing

Exercise - do it now!

1. Why does one test programs?

2. Write test cases for the following program for characterising triangles:
   1. input: three numbers, which should be lengths of the sides
   2. output: equilateral, isosceles or scalene

3. Why does one test programs?