

Software Workshop Team Java 2005

Dr Hayo Thielecke

(Pronunciation hint: TEA-lacker.)

Learning Outcomes

- **Design** and **implement** a larger software project than undertaken previously.
- **Document** good software **engineering practice**.
- Work effectively in a **team**.

Java is a prerequisite.

Some specifics

- Teams are assigned **randomly**.
- Team project means that **teamwork** is a factor in the **assessment**, not that anyone is entitled to their team mates' marks!
Example: marks of 80% and 15% in the same team.
- You need to **work steadily** in the semester. Starting in week 9 is too late. Giving yourself only trivial tasks is not smart either.
But: no revision, no exam.
- This module gives you some **practice** for **final-year projects**.

Schedule of the module

- Each week: meeting with demonstrator to report progress.
- Week 6: inspection of a prototype.
- Week 11: final software demonstration (\approx 20 minutes).
- 24 March: final report submission. (Penalties for late submission.)

The size is similar to final-year projects:

5 people for 10 credits = 500 hours in Team Java

1 person for 40 credits = 400 hours in the final-year project

Assessment

- Evidenced: Demonstration, regular progress meetings, and a final report. (CVS logs may also be used.)
- Every team member makes a case for their share of the marks in the report.
- There is no percentage breakdown of marks:
I use my academic judgement.
- Feedback during semester; final mark is only summative assessment (no discussion).

Running of this module

- There will be very few lectures.
- Read the Team Java Web pages *regularly*.
- You meet your demonstrators to discuss progress each week. It is your responsibility to prepare for the meeting: one-page weekly progress report.
- Send mail to `teamjava@cs.bham.ac.uk`. That goes to me and the demonstrators, *not* to students. A FAQ will be maintained. Check before you mail.

What you need to do

- You get some high-level user requirements.
I (HT) am the market for your product, so to speak.
- The requirements vary in difficulty and hence price.
I will pay you between 0 and 100 rather than £££.
- You need to select a realistic subset of the requirements, refine the requirements, (including fallback positions) perhaps develop precise specifications.
- Then: (incrementally) design, implement and document.

My requirements: program understanding tool

- The tool must process the abstract syntax tree generated by ANTLR.
- It extracts high-level structure from the code and
- interactively presents it to the user.
- Graphical display can be nice, but you may keep it simple (graphical layout is often harder than one thinks).
- The information can be like UML notations or different.

Some examples of functionality

1. Display the class inheritance tree, similarly for aggregation.
2. Slightly more difficult: with interfaces, no longer a tree.
3. Display the names of all classes and their public methods.
4. Clicking on a method displays the signature; you can click on the classes in that
5. Warn about style, e.g. access of public fields, cyclometric complexity.
6. Compute static software metrics.
7. Construct a call graph: there is an edge from C_1 to C_2 labelled m if a method of C_1 contains a call of method m from C_2 . (Note: formal specification.)
8. You are welcome to design your own—if you have ideas, ask us.

Tree Walking and code

- You will use a parser generated by the ANTLR tool.
- Models of Computation may be helpful (but not necessary) for grammars and parse trees. You don't have to understand parsing.
- Your program **must** extract information from the syntax tree. No external packages allowed for that (→ plagiarism!)
- Code is itself a complex recursive data structure. Recursion is fundamental in CS (e.g., see things like XML).

Team organisation

- Each team needs Team manager/secretary (required for each team to call meetings)
- Every team member is a programmer (This is a CS module after all.)
- Possible: Chief programmer or architect
- Possible ANTLR expert, GUI expert etc; or split work between modules

Software Engineering in this module

- Software Design and documentation is part of the Learning Outcomes
- You will need to use some software that is useful for building large systems:
ANTLR and CVS.
- We will use iterative development
(not Waterfall; nor Extreme Programming):
design and built a prototype, then refine it

Software Design

What is important is a separation between:

- requirements and specification (*what, not how*),
- design of the system (*don't miss the wood for the trees*),
- implementation.

This has to come out clearly in the final report.

Required and background reading

For high-level design, the book:

Lano, Fiadeiro and Andrade:

Software Design using Java 2

(Palgrave Macmillan, 2002)

is **required** reading.

Sommerville is the standard Software Engineering text.

Brooks, *The Mythical Man Month* can be insightful.

Some advice

- Start working on the module now, not in week 9.
- Let us know if there are problems.
- Do not just hack code, but bear the Learning Outcomes in mind.
- Start writing the report early.