Module Induction

Advanced Java Programming 2012-13

OO Design Patterns and Principles

Lecture Outline

- Module induction
  - Health warning
  - What is this module all about?
  - When are the lessons / lab sessions?
  - How will I learn?
  - How is this module assessed?
  - How do I get feedback on my work?
  - What are my responsibilities?
  - How do I get help?
  - What books should I read?
  - What software do I need?
  - Academic prizes
- Introduction to Design Patterns

Module Induction

- **Health warning**
- Welcome to Advanced Java Programming (AJP for short)
  - This is a 20 credit module lasting the whole academic year
- Important – if you scraped through JPR you are going to find this module very difficult ! (scraped is less than 50%)
- The module assumes that you understand the OO paradigm and can program in Java
  - If your Java/OO is weak then you will struggle
- If you are worried about your programming skills, please come and see me after the lecture

Module Induction

- **What is this module all about?**
- Last year you took ‘Programming 101’
- This module is about going to the next level
  - It's less about programming and syntax, more about software engineering
- A very good module for those of you who are considering SE as a career
- A very good module for those of you who enjoy technical, practical subjects
- A very bad module for anyone who dislikes programming

Module Induction

- **What is this module all about?**
- The first half of the module (October-January) will concentrate on OO design patterns and principles
  - This part of the module will be taught by me
  - I will also supervise all the lab sessions
- The second half of the module (January-May) will concentrate on concurrency and MAS
  - This part of the module will be taught by Dr. Simon Lynch
  - Simon will supervise the lab sessions
- We may swap in and out in case of illness etc.

Module Induction

- **When are the lectures / lab sessions?**
  - Check your timetable
  - Please note that labs do not start until the second week of teaching
  - Lab groups will be finalised before the start of the week
  - You cannot change lab groups without some form filling

Module Induction

- **How will I learn?**
  - Like last year, not all of your learning hours for this module will appear on your timetable
  - This module requires 'self-directed study time'
  - For every hour spent in class, you are expected to work for 2 hours outside of class
  - This means you should budget for around 4 hours of AJP SDST per week
- Self directed study time could include
  - Practicing your programming skills (most important)
  - Reading books about Java / patterns / MAS
  - Want to get ahead, get involved in a live project!

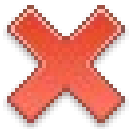
Module Induction

- **How will I be assessed?**
- In this module, you will be assessed by a single in-course assessment (ICA) with two components
  - Component one will be worth 60% of the module and will assess your knowledge of design patterns
    - Individual work, submit January 2013
  - Component two will be worth 40% of the module and will assess your knowledge of agents and concurrency
    - Group work, submit April 2013
    - Peer assessment scheme
- The first year of studies was about qualification
  - From here on, everything counts.....

Module Induction

- **How do I get feedback on my work?**
- You will receive formal feedback every time you submit a piece of ICA work
- You will also receive informal feedback every time you complete a lab assignment
  - There are 8 lab assignments in this part of the module
  - Each lab assignment has a due date (3 working weeks)
  - Passing lab assignments will be graded
  - I will keep a public record of your progress in a feedback matrix on EAT
  - Three failing assignments triggers a review

Module Induction



Fail - You did not submit a passing assignment on time. Try harder next time. If you miss three assignment deadlines you will trigger an automatic progress review.

Bronze - You have completed the basic task and your application produces the correct output. However, your Javadoc and/or your code style needs improvement.

Silver - You have completed the basic task and your application produces the correct output. Your Javadoc is complete and your code meets the house style.

Gold - You have completed the basic task and your application produces the correct output. Your Javadoc is complete and your code meets the house style. You have also completed the advanced task.

That's fine, but how will I know if my code is producing the *correct output* and how will you assess my *code style*?

Module Induction

- **Output**
- Every assignment will consist of the following
  - A basic problem specification, including a precise description of the required command line output
  - An advanced problem, to challenge you
- You will also receive a code skeleton you can use when developing your application
  - Part of this skeleton will be a set of *unit tests* that allow you to check your output at the press of a button
  - If your application produces the correct result, the tests will pass, otherwise they will fail

Module Induction

- **Code style**
- Last year you created code according to some fairly loose rules e.g camel case
- In the real world, programmers are usually forced to follow a *house style*
- This house style will usually describe
  - How code should be laid out, in *exact detail*
  - How code should be documented
- Sometimes, programmers are prevented from checking code back into repositories if they break the rules

Module Induction

- Checking code against a house style is usually automated, rather than done manually
- *Checkstyle* is a development tool to help programmers write Java code that adheres to a coding standard: http://checkstyle.sourceforge.net/
- *Checkstyle* is highly configurable and can be made to support almost any coding standard
  - An example configuration file is supplied supporting the Sun Code Conventions
- You can download it and use it from the command line
- Or, you can you integrate it in your build process or your development environment (RECOMMENDED)

Module Induction

- SQE offer a free CheckStyle plug-in for Netbeans.
  - Plug-in installation is simple, described here
  - The plug-in integrates with NB and flags any style violations on screen (in the same way as syntax violations)
- Lab assignments containing *CheckStyle* violations are limited to bronze medals
- ICA work that contains *CheckStyle* violations will be marked down
- There is no defence of 'That's just my style....'

Module Induction

- What rules will we be using?
  - The default set of rules are a bit strict
  - I have create some custom rules (you can download from EAT, in the LAB section)
- You can find out more about the rules by examining the XML file in conjunction with this website
  - For example, here is documentation for the rule referenced by this line of XML

```
<module name="MissingSwitchDefault"/>
```

Module Induction

- Other rules enforced by scm_checks.xml
  - All classes (except abstract classes) must define a constructor
  - Classes must follow a specific declaration order – static variables, instance variables, constructor, methods.
  - No use of star operator in imports, to avoid polluting the name space
  - All left curly braces on the end of line
  - All methods, classes, types and variables must have javadoc documentation
  - …..and about 20 more

Module Induction

- **What are my responsibilities?**
- You need to attend lectures and practical sessions
  - Always swipe your card outside of the lecture theatre
- You need to keep up to date with your lab assignments
  - By looking at the feedback matrix and the attendance register, I can get a good idea of your commitment
  - Non-attenders and work shy will get emails, progress reviews, then referrals to the course leader ....
- You need to pre-read for the lectures
  - I give non-optional reading assignments every week

Module Induction

- **How do I get help?**
- An academic problem (lecture/worksheet/ICA)?
  - Talk to Simon or myself in the practical session
  - An email is acceptable, but don't expect instant results
- Problem with the module (ICA extensions, falling behind, illness etc.)?
  - Talk to me first (I am the module leader)
  - Then => Personal Tutor, Course Leader, Program Director

Module Induction

- **What books should I read?**
- For general Java syntax
  - Deitel, P.J. & Deitel, H.M. (2011) *Java: How to Program*.
- For OO design patterns
  - Holzner, S. (2006) *Design Patterns for Dummies*
  - Freeman, E. & Freeman, E. & Sierra, K. & Bates, B. (2004) *Head First Design Patterns*, Prentice Hall
  - Gamma, E et al. (1995) *Design patterns: elements of reusable object-oriented software*
- All available via the e-Library!

Module Induction

- **What software do I need?**
- Java SE Development Kit 7u7 or better
- Netbeans IDE 7.2 for Linux
  - All of your lab sessions will be in the Linux labs
  - Your ICA work and lab assignments must compile and run in the Linux labs
  - There are Windows and Mac versions of Netbeans, but I do not support them
  - I recommend that you all install Linux on an old machine, dual boot or work only in labs (back up first!)
- SQE CheckStyle plugin for Netbeans

Module Induction

- **Academic prizes**
- Each year there are prizes for '*Academic Excellence in Software Engineering*'
  - The prizes are awarded for performance in the first half of the AJP module (ICA, feedback log and attendance)
  - The prize is awarded to 2 students
  - The prize is usually a technical book of your choice
  - You can add this award to your CV
  - I will *always* write glowing references for winners
  - One of last years winners is currently on placement with IBM, the other is headed for a 1st class degree

Introduction to Design Patterns

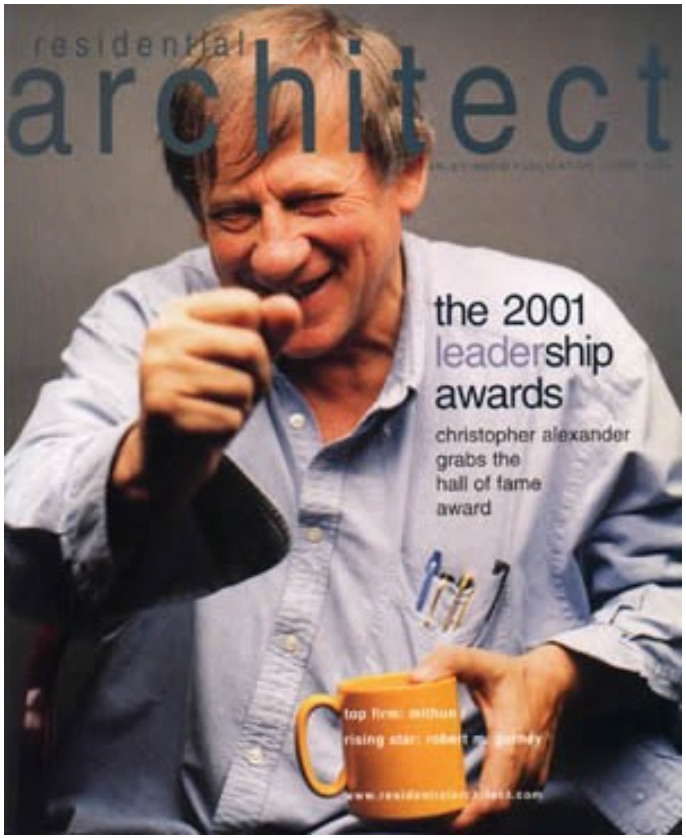
- **What is a design pattern?**
- We are used to using class libraries
  - We have been using the Java API since last year
- This is a form of *code re-use*
  - You are re-using an implementation of a *thing*
  - There is little point re-writing the *JLabel* class if we are perfectly happy with the version in the API
- Design patterns are a form of *experience re-use*
  - Somebody has already faced, and solved your problems
  - Let's exploit that experience!

Introduction to Design Patterns

- In software engineering, a design pattern is a reusable solution to a commonly occurring problem e.g. how to keep many objects informed of changes in the state of another object
  - It is not a finished design that can be transformed directly into code
  - It is a template for solving problems that can lead to many different implementations
- Think along these lines - a design pattern is a tried and tested way of approaching a particular problem
  - Think of each pattern as a tool for your toolkit

Introduction to Design Patterns

- Design patterns emerged from the field of architecture



Each pattern describes a problem that occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over without ever doing it the same way twice – **Christopher Alexander**

Introduction to Design Patterns

- In 1994, Alexander's work was exported to the field of computer science
  - Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides (the Gang of Four, or GoF) published *Design Patterns: Elements of Reusable Object Oriented Software*
  - The same year a dedicated conference (Pattern Languages of Programming) was launched
  - In 1995 the Portland Pattern Repository was set up for documentation of design patterns
- Patterns have been very influential on the SE industry
  - Learning about patterns offers you the opportunity to think about SE 'in the large'

Introduction to Design Patterns

- The original Design Patterns book specified three main types of design pattern:
  - **Structural** patterns are design patterns that identifying simple ways to realize relationships between objects e.g. Adapter
  - **Behavioural** patterns are design patterns that identify communication patterns between objects e.g. Observer
  - **Creational** patterns are design patterns that deal with object creation e.g. Singleton
- Simon Lynch will introduce a 4th category, which deal with concurrency patterns

Summary

- Design patterns allow us to draw on the experience of other software engineers
  - Design patterns are not implementations
  - Design pattern are general solutions to commonly occurring problems
- There are three different types of pattern – creational, behavioral and structural
- To prepare for next weeks lecture, please read the following:
  - Oracle tutorial on interfaces
  - Just Java, chapter 11, as far as *Interface Comparable*
  - Learn about the Singleton pattern in HFDP chapter 5
  - You will be questioned.....
- Install Linux, Netbeans and the SQE CheckStyle plug-in