Assignment 2

= Summary of the course so far ☺

Recap of what we've learned

- Version control (svn)
- Project setup (maven)
- Testing (junit, spring-test)
- Persistence (hibernate)
- Wiring application together (spring)
- (Web layer – Struts2)

…And what we haven't really covered

- IDE environment (eclipse)

= problems in assignment2

Assignment2

- Still trouble commiting only the right things
- Maven setup not correct
- Formatting, unnecessary code and comments, duplicate declarations
- Transactions
- Understanding how to deal with many-to-many
- Not all of this is your fault ☺

Commiting only the right things

- Typically target folder and IDE files
- Solution
  - Look at what you commit!
  - All version control systems have mechanisms to ignore files
  - svn:ignore property =

    .project

    .classpath

    .settings

    target
  - svn propedit svn:ignore <folder>

Maven setup not correct

- Important to realize I* only care about your maven setup, I wont even see your IDE setup!
- mvn in parent project **must** work and run all modules. That's sort of the point..
- I don't care if you do whatever to get it working in your IDE of choice
  - It doesn't even have to run there

* I, as in everyone else looking at your code.

Code mess

- A lot about experience
  - Assignment to unused variables
  - Using iterator instead of foreach
  - Etc…
- But… your IDE will help you
  - At least format using Ctl-Shift-F…
- If your IDE is too helpful..
  - Comment templates..
  - Configure it, don’t leave me to deal with your spam.

Transactions

- Really important to understand what transactions are!
- It might seem natural to put transactions on the DAOs…
  - But what about logic involving several DAOs?
  - Or even several transactional systems
    - a database + a message system sending notifications to a remote system?

Transactions in assignment

- Use @Transactional
- @Transactional belongs on the service layer
- A transaction is a **logical** unit of work
  - "Whatever is done within this method, either everything should succeed or nothing"

@Transactional and DAOs

- You need to explicitly use the DAO when
  - You want to persist a newly created object (save)
  - You want to remove an object from your persisted model (delete)
  - Ask for objects
- Demarcating your transaction (@Transactional)
  - within this context, all entity modifications should either be persisted or rollbacked.
  - Persisting handled by framework, explicit updates are meaningless within transcational boundary

many-to-many entity relationships

- Can seem difficult - especially delete
- “Safe” version
  - Before deleting a course, delete all associations to it
- Better
  - The association will be deleted by hibernate on the side where inverse=true is **not** set (“mappedBy” in JPA)
  - But hear reports of 2. level cache bug…

Limitations in assignment

Generic DAOs

- Standard dao functionality should be reused
- Subclass and implement extra behaviour only for specific functionality
- http://community.jboss.org/wiki/GenericDataAccessObjects

Testing

- Relevant for this layer…
  - Test data (DBUnit)
  - Mocking (and stubbing)
    - Mockito, easymock…
- But also all kinds of other testing..
  - load, stress, acceptance, web..

Spring-test – False positives

- Really good with rollback and autowiring, but…
- spring-tests don't flush session for individual operations
  - Transactinoal is around the whole test
  - Don't actually test sql towards db
  - Outdated entities aren't necessarily evicted from session cache
  - Need to flush (and evict) explicitly in tests

Realistic spring-test testing

One of these tests will fail if Course <-> Student association is not handled correctly in assignment, but only if flush() is used

```
@Autowired
private SessionFactory sessionFactory;
```

```
private void flush() {
        sessionFactory.getCurrentSession().flush();
}

private void evict(Object o) {
        sessionFactory.getCurrentSession().evict(o);
}
```

```
@Test
  public void testDelCourseWithStudent()
  {

     int courseId = studentSystem.addCourse( "INF5750", "Open Source.." );
     int studentId = studentSystem.addStudent( "John" );

     studentSystem.addAttendantToCourse( courseId, studentId );

     studentSystem.delCourse(courseId);

     flush();

     Course course = studentSystem.getCourse( courseId );
     Student student = studentSystem.getStudent( studentId );
     evict(student);
     student = studentSystem.getStudent( studentId );

     assertNull(course);
     assertTrue(student.getCourses().isEmpty() );
  }
```

```
@Test
  public void testDelStudentWithCourse()
  {
     int courseId = studentSystem.addCourse( "INF5750", "Open Source.." );
     int studentId = studentSystem.addStudent( "John" );
     studentSystem.addAttendantToCourse( courseId, studentId );

     studentSystem.delStudent(studentId);
     flush();

     Course course = studentSystem.getCourse( courseId );
     evict(course);
     course = studentSystem.getCourse( courseId );
     Student student = studentSystem.getStudent( studentId );

     assertNull(student);
     assertTrue(course.getAttendants().isEmpty() );
  }
```