

20.1 Tips for Lab Ex. 6

- Build your own web browser by connecting together pre-existing classes.
- Go back over URLs and HTML from Lecture 7.
- JEditorPane, HyperlinkListener (c.f. L.7.)

Lab 6 (Cont.)

- JEditorPane needs an HTMLToolkit
- HTMLToolkit needs a stream
InputStreamReader and an
HTMLDocument
- HTMLDocument has an iterator
HTMLDocument.Iterator

Lab 6 (Cont.)

- Jtable needs a `String[][]` data structure
- Place components in a `JScrollPane` for arbitrary required space.
- `JTextField` has a method `postActionEvent()`
- Otherwise can call `actionPerformed()` for your `Jframe` with newly created `ActionEvent`

20.2 User Defined Events

- Can create your own event class
- This has its own interface
- Class implementation must implement listener methods
- Post an event by creating an event object and sending it to a listener method.

20.3 Introspection and Reflection

- Java beans contain introspection which is based on reflection
- When we execute a Java program class files are loaded as needed
- At this point we can ask a class for its name, type, methods, method parameters etc.

20.4 Sound

- Easy to play sound using an applet
- Connection to sound device has security aspects
- Obviously also real-time aspects
- Check out class `MidiSystem`
- See also
http://java.sun.com/j2se/1.4.2/docs/guide/sound/programmer_guide/contents.html

20.5 Databases

- JDBC is the Java API for databases
- Connection to an SQL database
- Overview at:

<http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/getstart/GettingStartedTOC.html>

- Establish a connection with a data source
- Send queries and update statements to the data source
- Process the results

```
Context ctx = new InitialContext();  
DataSource  
    ds=(DataSource)ctx.lookup("jdbc/myDatabase");  
Connection con =  
    ds.getConnection("myLogin","myPassword");  
  
Statement stmt = con.createStatement();  
ResultSet rs = stmt.executeQuery("SELECT a, b, c,  
    FROM Table1");  
  
while (rs.next()) {  
    int x = rs.getInt("a");  
    String s = rs.getString("b");  
    float f = rs.getFloat("c");  
}
```

20.6 Numerical Methods

- A numerical library exists for Java
- <http://math.nist.gov/javanumerics/>
- General library incl. Linear algebra
- Benchmarks
- Connections to Matlab and Mathematica
- Visualisation

20.7 Java for Mobile Phones

- J2ME Java 2 micro edition
- <http://developers.sun.com/techttopics/mobility/midp/articles/wtoolkit>
- MIDP 1.0 standard works on many mobile phones. CLDC 1.0 configuration.
- Screen resolution varies across phone models

Java for Mobile Phones (Cont.)

- Jar files created, class files must be signed.
- Methods: `startApp()`, `pauseApp()`, `destroyApp()`, `commandAction()`
- Highest score lists can be saved as files on phone
- Download text via WAP from homepages

Lecture 12

22. Course Summary

22.1 Main Themes

- OOP and OOD
 - Modularisation
 - Communication between program components
- Design patterns
- Object modeling with UML
- Software development lifecycles
- Advanced Java, APIs, GUIs
- Also testing, execution models, law, security, data structures

22.2 Complex Systems

- Have a hierarchical structure
- Choice of primitives is fairly arbitrary
- Coupling within components is stronger than between components
- Often just a small number of subsystems with different arrangements
- A correct complex system has often started from a correct simple system

22.3 Abstraction Methods

- Type systems
 - Create new data types or methods
 - Use inheritance, interfaces, polymorphism, reflection
- Services
 - Method calls
 - Event handling + registration
 - Message handling (overloading, override, polymorphism, RMI)

Abstraction Methods

- Process control
 - Multitasking
 - multithreading
- Relations
 - Associations
 - “depends on” (i.e. parameters in methods)
 - “is an” (i.e. inheritance, interfaces)
 - aggregation “has an” (i.e. instance variables)

Abstraction Methods

- Behaviour (how classes provide services)
 - Static via method call
 - Dynamic (dependent on internal state)
- Control
 - Exception handling

22.4 Modularisation

- Classes
- Inner classes
- Interfaces
- Package
- Process
- Threads
- Dynamic loading, reflection
- Bean
- Servlet

22.5 Communication

- Methods (many kinds, see design patterns)
- Events, registration
- Messages (overloading, overriding, polymorphism)
- RMI
- CORBA, .NET, SOAP

22.6 Design Patterns

- Solutions to common design problems
- Study the question and a simple example
- See exercises and labs for concrete examples
- Lab 1 Observer
- Lab 2, 3 Observer, MVC
- Lab 4 Composite, Iterator, MVC
- Lab 5 Façade, MVC
- Lab 6 Iterator MVC

Design Patterns

- L2. Observer, Model-view-controller, singleton
- L3. Mock object, Relation (Mediator)
- L5. Mediator, Factory, Command
- L6. Iterator
- L10. Proxy, Abstract factory, Decorator
- See also L9 and L5 for good OOD principles

22.7 UML

- Systematic modeling of requirements
- Why did we talk about use-cases?
- Why did we draw class diagrams?
- Why did we draw sequence diagrams?

22.8 Use-cases

- At project start we need to identify specific classes (domain specific) and general classes.
- Starting from the required functionality one can assign responsibility to classes
- By recording/simulating typical scenarios (use-cases) we can identify actors, relationships, responsibilities and sequencing

22.9 Class Diagrams

- Shows how classes are interdependent
- Shows division of responsibility
- Shows what services a class provides
- Shows actors and roles, e.g client-server
- Shows where data is stored
- Should show effects of class changes
- Static, no states or behaviour

22.10 Sequence Diagrams

- Shows how functionality is divided up
- Sequential order of method calls
- Shows relative timing of events/results
- Shows multiple threads
- Can see if semaphores are needed

22.11 Software Development Lifecycles

- Project control & decision making are difficult
- Waterfall model, phased, sequential
- Spiral model (iterated)
- Agile/XP (changes/testing)
- RUP (phases, specification)
- Rapid prototyping (e.g. top-down from GUI)

22.12 Java

- Small language, big API
- Object-oriented inheritance + interfaces
- Network programming, Applets, Servlet, HTML and XML parsers, RMI

Java (Cont.)

- Robust, secure
 - **Strongly typed, defined word lengths, variables must be initialised**
 - **No pointers**
 - **Garbage collector**
 - **Runtime control of array index etc**
 - **Exception handling**
 - **Sandbox, security manager**
 - **Encryption, signatures**

Java (Cont.)

- Architecture neutral, portable
- Multithreaded
- Dynamic, dynamic loading
- Internationalisable, Unicode, locale: classes for time, date, currency

22.13 Henriks Questions

- Go through Henriks questions in the notes for L12, good for revision
- Exam will not consider details of JBuilder, Beans, Servlets, CORBA,