Mitschrift Programmieren:

11. Serialization

**11.1 Overview**

* Soring of instance values in files(in a recursive manner)
* Restoring instance values from files

Comparison to relational database:

+references (between instances) are also stored (recursive)

-From working with data you have to restore the whole database

**11.2 Store**

* Classes must implement Serializable
* Otherwise java.io.NotSerializableException
* Empty Interface
* Idea: controlling whether class data should be serialized reconstructed or not (eg security reason)
* Static attributes cannot be serialized

**11.3 Reconstruction**

Beware of correct sequence of reading the data

Example\_

Oos.writeObject(a1);

Oos.writeObject(a2);

🡪

Adventure a1 = (Adventure) ois.readObject();

Adventure a2 = (Adventure) ois.readObject();

**11.4 Serialization Version Identification**

=

…….

Receiver

SVUID=#

Sender

SVUID=#

Each serializable class should have a serialVerisonUID.

Which is used to verify that sender an receiver of a serialized object deal with compatible classes.

**11.5 Controlling of Serialization**

* IF whole class must not be stored THEN remove implements Serializable
* IF single attributes must not be stored THEN use transient

11.6 Serialization to XML (JAXB Overview)

11.6.1

App 1

(Java)

Serial xml

App2

(Java)

Serial bin

App 3

(eg: C#)

JavaArchitecture for XML Binding (JAXB)

* is a Java standard that defines how Java objects are converted from and to XML. It uses a standard set of mappings.

JAXB uses annotations to indicate the central elements.

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
| @XmlRootElement | Defines the root element of a XML tree mandatory |
| @XMLElement (name = “new Name”) | Define the XML element which will be used. Only needed if newname is different than the name of the Java attribute |
|  |  |
|  |  |
|  |  |

Bean – Defenition

Each attribute needs appropriate getter & setter

Default Constructor

# 12 JTable

## 12.1 Overview

GUI – Component

(=observer)

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

TableModel

(observable)

* enbed JTable into ScrollPane
* implement TableModel as sublclass of AbstractTableModel
* working with DefaultTableModel is semiprof becaust
  + collection type is fixed
  + inflexible concerning layout-info for GUI
  + writing

# 13 Pattern: Singleton

# 13.1 Overview

Sometimes it is important to have only one (or not more than ‘n’) instance(s) of a class

For example in a system there should be only one file print database manager

Usually singlestons are used for centralized management of internal or external resources.

App-Controller

DB-Controller

App

App

Singleton

State uniqueInstanc

SingleTon Data

Static newInstance ();

singletonOperation();

get/setSeingletonDaata();

# 1 Software Test an Introduce

## Static Test

* Non-execution based methods
* Software will not be executed (for testing)
* Test is on source code
* 🡪white box test
* E.g. “Walk Through”
* = meeting with developer of SW, other developers
* Up to 70% of bugs can be found

## 1.2 Dynamic Test

### 1.2.1 General Remarks

* + Precondition SW is executable
  + Testing with predetermined test cases and
  + Comparison of the outcome with (expected) results
  + If there is deviation 🡪 SSW (method) does not work correct
  + 🡪 black box test

Problem:

* + It is very likely that there are an endless number of possible test cases
  + E.g. Int sum(int a, int b);
  + 🡪 cover all possibilities with a suitable number of test cases

### 1.2.2 Equivalence Classes

* Divide all possible input data
* Into equivalent classes
* If an equivalent class consists of value ‘x’ and ‘y’
* We assume that
* If processing with ‘x’ is correct, processing with ‘y’ would be correct too
* If processing with x fails y will fail too.

Example 1

* Specification of an online banking system
* Valid amount between 0.01 and 500€
* If 123,44 == ok 🡪 123,45 would be ok too
* Ec amount -1: between 0.02 and 500 🡪 valid
* Ec amount -2: x<0.01 🡪 invalid
* Ec amount -3: x>500 🡪 invalid

Boundary Value Analysis

* It is frequently observed that processes fail because of
* Values near the boundary of equivalence classes
* If(amount >=0.01…)
* So we should test also
* 0.01-0.00-0.01
* 499.99-500.00-499.99