## **7.2 Interfaces**

### **7.2.1 Overview**

**Idea:**

* method x gets some objects an
* the behaviour of these objects must be guaranteed
* this is: objects have to implement some methods (signature)
* so, an Interface sets guide lines for (sub)classes that wants to guarantee that behaviour

# **8 Collections**

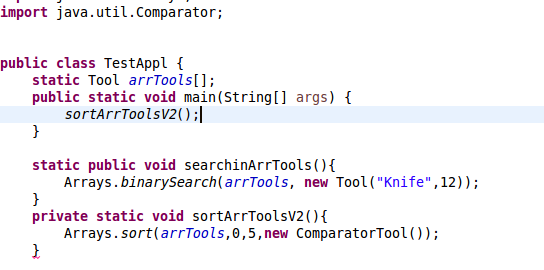
## **8.1 Overview**

Why use java-Collection (instead of own implementation)?

* not to reinvent the wheel
* to increase performance, correctness and reliability
* to decrease time of development
* easy communication with other classes/system

## **8.2 java.utils.Arrays**

Array.sort (arrTools,0,5);  
Arrays.binarySearch (arrTools, new Tool(“knife”,12)));



* array must be sorted (see interface comparable)
* returns
  + if found->index in array
  + if not found -> expected position less 0

## **8.3 Collections- Overview**

SortedMap

Map

Set

SortedSet

Collections

List

2 basic types

* Sequential storage (with direct access)
* Objects are stored associatively (access with key)
* Collection= can contain many objects
  + Set is a collection; without duplicates and ranking
    - SortedSet is a Set: with ranking (see Comparable)
  + List is a Collection without ranking; duplicates allowed;   
    access via index (position of element)
* Map= member have unique keys
  + SortedMap is a Map, members are sorted by key

hints:

* A collection has only references of its object
* For sorting you have 2 alternatives:
  1. Interface (Comparable)  
     which has to be implemented by these objects  
     that have to be sorted
  2. Interface Comparator  
     which has to be implemented by the comparison-classes

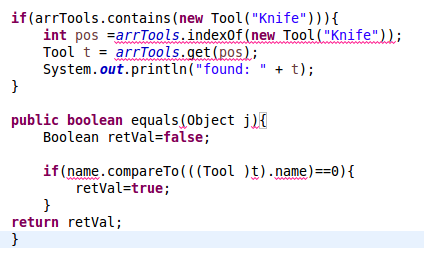
Implementations:

|  |  |  |  |
| --- | --- | --- | --- |
| Interface | Implementationclasses | | |
| Set | HashSet |  | TreeSet  (sorted set) |
| List |  | ArrayList LinkedList Vector (depricated) |  |
| Map | HashMap, -table |  | TreeMap  (sorted set) |

## **8.4 ArrayList**

* Static ArrayList <Tools> arrTools= new ArrayList<Tool> ();  
  arrTools.add(Tool);
* method it.next() returns Tool
* for (Tool i: arrTools){

system.out.println(“seas” + i);  
}

* for(Iterator <Tool> it=arrTools.Iterator();it.hasNext();){  
   System.out.println(t);  
  }
* arrTools.contains(t2)….. true  
  arrTools.contains( new Tool with the same value)…… **false new reference**   
  arrTools.indexOf(t2); **--------**🡪 ( same with new operator, is doing the same as contains, it starts counting by 0)  
  contains is using equals

## **8.5 ArrayList**

* no efficient sorting (if new element -> new sorting necessary)
* no efficient seeking (loop)

## **8.6 TreeSet** 10/01/2017

* unique objects
* sorted
* be aware of seeking
* alternate sorting

## **8.7 HashSet**

in comparison in TreeSet: better performance

## Hash Code 11/01/2017

* every Object-Class provides a hashCode()  
  which transforms the stored data into a  
  single (hash) value (32bit signed int)
* this hash is used by other apps when  
  storing or manipulating the instance  
  eg. hash tables, hash map, hash set

**🡺no guarantee that  
 different instances produce distinct hashcode** if different instances have equal hashcodes then  
 only performance problem (if apps be aware of that fact) 🡸

**🡺Attention:** different runs can produce different hashcodes  
 (eg. change code slightly and start new run)

🡺upgrade brings

* unique hashcode (no guarantee, because hashcode of  
  string could be buggy)
* therefore contains() works(mostly) correct
* but still hashcode could change from run to run

🡺 according to Javadoc  
the class makes no guarantees as to the  
iteration order of the set

Properties of HashMap

* fast access
* elements in the set are not sorted in any way  
  (the ordering isn’t fixed at all)
* using many keys with the same hashCode()  
  is a sure way to slow performance

## **8.9 TreeMap**

* objects identified by key
* sorted by key
* key has to implement Comparable

private static void searchTools(){

s.o.pln(arrTools.containsKey(“Scissors”));

s.o.pln(arrTools.get(“Scissors”));

s.o.pln(arrTools.get(“Hammer”));

}

//in printArrTool()

for(Tool t : arrTools.values())

s.o.p(t);

# **9.Deployment (Introduction)**

### all activities that make a software system available for use

## **9.1 Java-environment**

schueler@ubuntu:-$ set | grep java

## **9.2 Simple Compile & Run**

rm -r bin 2>/dev/null

mkdir bin

javac ./scr/\*.java -d bin

cd bin

java -cp “.“ TestApplHashMap /\* -cp … class path \*/

cd ..

chmod 0777 \*.sh CHANGE the rights

## **9.3 Java-Package**

**Idea:**

Define packages

* to get an overview over classes
* avoid conflicts of class names
* easier administration of class-groups

**Properties of JAR-Files (Java ARchive)**

* platform – independent format (ZIP)
* => data compression possible

**Syntax – Overview**

jar [ options ] [ manifest ] destination input – files  
 manifest => holding meta information   
 destination => name of jar-file  
 input-files => contents

Options

|  |  |
| --- | --- |
| c | creates a new archive |
| x | extracts all files |
| f *file* | The second argument specifies a jar file to process |
| e classname | Defines entrypoint of application |
| m textfile | Adds data from textfile to manifest of jar file |

rm -r bin 2>/dev/null

mkdir bin

javac ./scr/\*.java -d bin

cd bin

java cfe ../CollApp.jar TestApplHashMap \*.class

cd ..

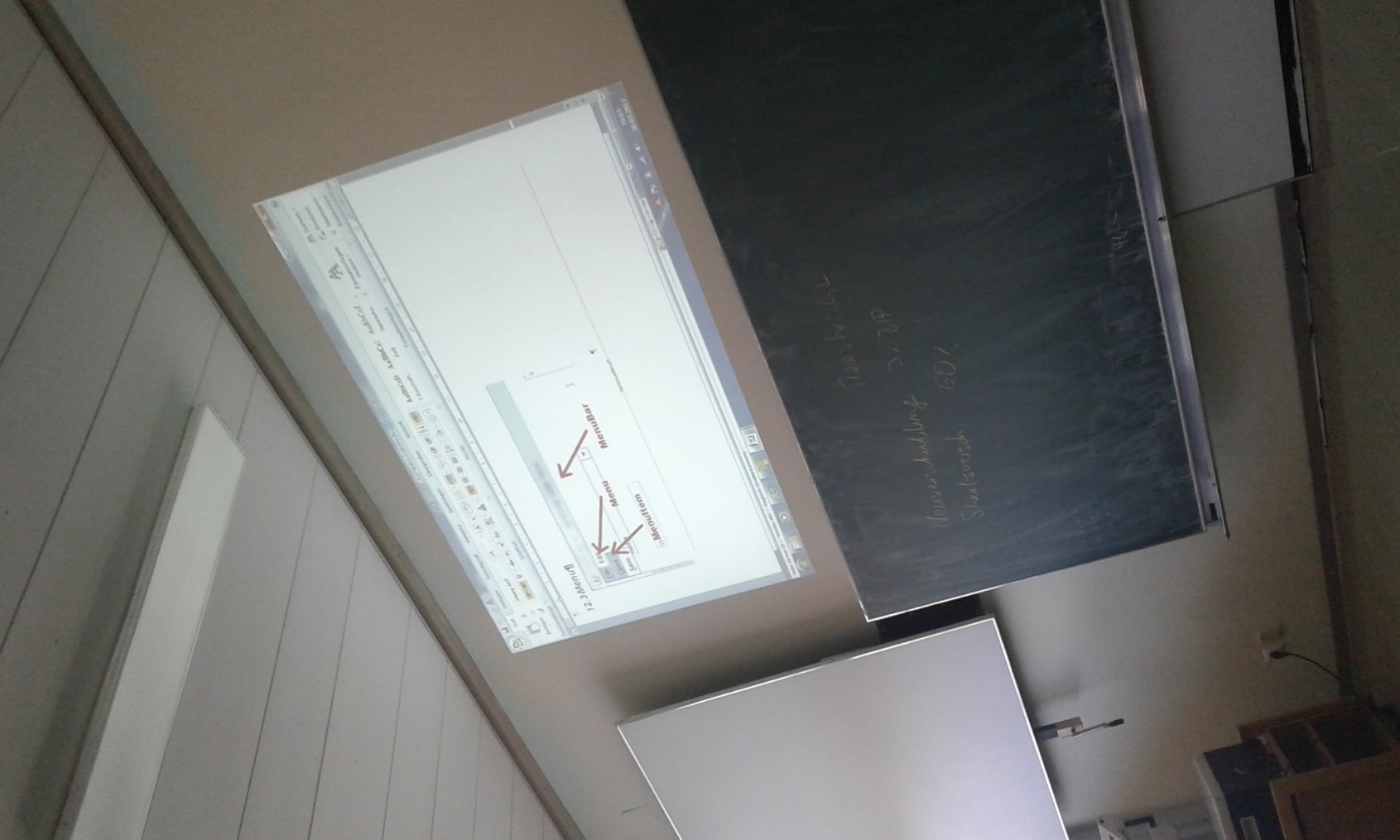
java -cp “.“ -jar CollApp.jar

# **10. GUI – Swing(Introduction)**

Event Selection

listCar.addListSelectionListener(ListSeltionListener listener);

# **10.3 Menu**



# **11.Deployment (Introduction)**

11.1 Java Environment

schueler@ubuntu:~ set | grep java

11.2

# **12.Deployment(continued)**

GUI\_Car

pkgMisc

pkgGUI

pkgData

steps:

1. make directory “bin”
2. compile all packages (at once) into “bin”  
   javac ./src/pkgData/\*.java ./src/pkgGUI/\*.java…..
3. make jar File of all generated class files
4. delete “bin”

# **13.Pattern: Observer**

## 13.1 Overview

Why use pattern?

* not to reinvent the wheel
* to increase performance, correctness and reliability
* to decrease time of development
* easy communication with other classes/systems

**Observer Pattern:**

an object has information,  
(many) other objects are interested in these information  
Information could change and so other objects should be informed immediately   
*Observable (or Subject)* …. object which holds information  
*Observer, Listener …* interested object  
Java delivers a class *Observable* and an interface *Observer*

## 13.2 Structure

**class diagram:**



🡸 knows observable

<<uses>>

knows observers 🡺

return state;

observerState=  
observable.getState();

for all o in observers  
o.update();

update (Observable, Object)

update(…)

observerState

**ConcreteObserver**

<<interface>> **Observer**

state

**ConcreteObservable**

getState()  
setState()

addObserver(Observer)  
setChanged()  
notifyObserver()  
…

observs Collection

**Observable**

directed association to an interface

addObserver() 🡺 observer can registrate itself

notifyObservers() 🡺 if data has changed then notify all observers

🡺 call update() of each observer

update() 🡺 is called when data has changed

🡺 parameters: Observable … source of event

Object … optionally changed data

**sequence diagram:**



**aConcObservable**

**aConcObs1**

**aConcObs2**

setState ()

notifyObservers ()

update ()

update ()

getState ()

getState ()

bar 🡺 method is running  
arrow 🡺 an observer calls getState() of observable -> aConcObservable.setState()