Organisatorisches

- Zwischentest am 12.12. ab 10h im Hilbertraum
- VO+UE am 12.12. entfallen
- Zusätzliche VO am 10.12. von 9:15-10:45 (Fragestunde)
- Alte Prüfungsangaben unter

  infotech.unileoben.ac.at/lehre/lva.htm

Nachbetrachtung Wissensüberprüfung

- Wird Objekt mit min./max. Eigenschaft gesucht, initialisiert man dies mit *null*.
  Ein Konstruktoraufruf ist weder nötig noch sinnvoll!
- Eine Abfrage auf Spezialfälle (wie z.B. leere Liste) ist bei guter Programmierung nicht nötig.

Nachbetrachtung Wissensüberprüfung

- Strings (und andere Objekte) vergleicht man mit *equals* auf Gleichheit. Nur in speziellen Fällen ist ein Vergleich mit == sinnvoll.
- Bei Vorinitialisierung mit *Integer.MAX_VALUE* oder *Double.MAX_VALUE* muss bei Min-/Maxsuche genau genommen mit <= verglichen werden.

Projekt Scheduling

- Haben Liste von Aufträgen unterschiedlicher Bearbeitungslänge, die möglichst gleichmäßig auf zwei Anlagen verteilt werden sollen.

Projekt Scheduling

- Wie schwierig ist das?
- Wie misst man Schwierigkeit?
- Wie gut lässt sich optimale Lösung approximieren?

Projekt Scheduling

- **Wie misst man Schwierigkeit?**

  → Wie wächst Laufzeit mit Problemgröße?
- **Wie schwierig ist das?**

  → I.a. sehr schwierig (NP-schwer)
- **Wie gut lässt sich optimale Lösung approximieren?**

  → recht gut:

  - Unser einfacher Algorithmus ist höchstens um Faktor 2 schlechter als optimale Lösung

  - Wird vorher sortiert höchstens um Faktor 4/3 schlechter als optimale Lösung

Java *interfaces*

- *Interfaces* sind eine besondere Form von abstrakten Klassen:
  - Alle Methoden sind *public* und *abstract*.
  - Es gibt keine Konstruktoren.
  - Es gibt keine Attribute außer Konstantendefinitionen, die *public*, *static* und *final* sind.

- D.h., dass eine nicht-abstrakte Unterklasse eines Interfaces **alle** Methoden des Interfaces implementieren muss.

- Insofern ist ein Interface eine Spezifikation, die angibt, über welche Methoden eine Objekt dieses Typs verfügt.

Interface *Comparable* (in java.lang)

class T implements Comparable<T>

legt fest, dass Objekte der Klasse *T* mit anderen Objekten der Klasse *T* vergleichbar sind.

Dazu muss die Methode

public int compareTo(T obj)

überschrieben werden.

Interface *Comparable* (in java.lang)

- Die Methode

  public int compareTo(T obj)

  muss so implementiert werden, dass sie
  - eine negative Zahl zurückgibt, wenn *this* „kleiner" als *obj* ist,
  - eine positive Zahl zurückgibt, wenn *this* „größer" als *obj* ist,
  - 0 zurückgibt, wenn *this* gleich *obj* ist.

Interface *Comparable* (in java.lang)

- Die Methode

  public int compareTo(T obj)

  muss so implementiert werden, dass gilt:

- Wenn *x.compareTo(y)==0*, dann auch *x.equals(y)==true*.

- *x.compareTo(y) == -y.compareTo(x)*

Interface *Comparable* (in java.lang)

Entsprechend:

- Wird Comparable implementiert, so muss auch *equals* und in der Folge *hashCode* überschrieben werden!

Klasse *TreeSet* (in java.util)

- Implementierung des Interfaces *Set*, das die Vergleichbarkeit der Elemente mittels *compareTo()* ausnutzt.
  - Die Elemente können in der durch *compareTo()* definierten Reihenfolge zugegriffen werden.
  - D.h. *for-each*-Schleife durchläuft Elemente in *TreeSet* in aufsteigender Reihenfolge.

Klassen *Arrays* und *Collections* (in java.util)

Die Klasse *Arrays* stellt zahlreiche Methoden für Arrays zur Verfügung:

- *static void sort(Object[] a)*

  (funktioniert nur wenn Objekte im Array Comparable implementiert haben!)
- *static String toString(Object[] a)*
- *static boolean equals(Object[] a1, Object[] a2)*
- *static int hashCode(Object[] a)*

Für Collections (inkl. List, Set) gibt es ähnliche Klasse *Collections*.