Informatik II (D-ITET)

Tutorial 7

TA: Anwar Hithnawi, E-mail: hithnawi@inf.ethz.ch
Distributed Systems Group, ETH Zürich
Outlook

- Exercise 6: Solution discussion

- Exercise 7: Overview (Generics, Binary Trees, Reversi)
Tips on U7.A1 – Generics

- U6 Generics
  - All classes inherit from Object *(abstract base class)*
  - Cast when extended from container (here List)

    ```java
    MyType Elem = (MyType) Kollektion.getNext();
    such casts can lead to runtime ClassCastException
    Better this way:
    Object obj = Kollektion.getNext();
    if( obj instanceof MyType )
    doSomething( (MyType)obj );
    ```
Tips on U7.A1 – Generics

- U7 Generics
  - Collection of Java Generics *(generic class)*
  
  ```java
  class MyPair<T> {
      public T first, second;
  }
  
  An object *pair of type* `MyPair<Float>` *contains two Float references*: `pair.first` *und* `pair.second`
  
  An object *pair of type* `MyPair<Integer>` *contains two Integer references*: `pair.first` *und* `pair.second`
  
- Advantage of generics:
  - Type check at the compiler time which increases type safety
  - Compiler takes care of type casting
Tips on U7.A1 – Generics

- ArrayList Container

Double nesting:
- ArrayList contains groups
  ```java
  ArrayList<ArrayList<Student>> groups;
  ```
- Groups contain students
  ```java
  ArrayList<Student> group;
  ```

- Filter: „can obtainTestat”
Tips on U7.A1 – Generics

a. **FilterFactory** and (empty) **IFilter** implementation
   
   Input: ArrayList of groups, that are actually ArrayLists of students.
   
   Output: ArrayList of students obtaining the Testat.

b. Implementation of **filterRaw**
   
   - filterRaw(ArrayList)
   
   No Generics: ArrayList as raw type (compiler warnings)
   
   Filter out all students who do not have enough points for the Testat...

c. Implementation of **filterGeneric**
   
   - filterGeneric(ArrayList<ArrayList<Student>>)
   
   ArrayList<T> indicates what is stored inside it
   
   Type checking when adding elements to the list
   
   ArrayList<T> directly provides objects of the correct type (no casting required)
Tips on U7.A2 – Binary Tree

- Each node contains pointers to:
  - Left successor
  - Right successor
  - (Parent)

- Recursive traversal:
  - Pre-order: P-L-R
  - In-order: L-P-R
  - Post-order: L-R-P
Tips on U7.A2 – Why only Binary Trees?

General trees can also be represented by binary trees:

„The binary tree can be thought of as the original tree tilted sideways, with the black left edges representing *first child* and the blue right edges representing *next sibling*. ... This is called *left-child-right-sibling binary tree (LCRS tree)*”

http://en.wikipedia.org/wiki/Binary_tree
Tips on U7.A2 – Binary Search Trees

- **Structure:**
  - The nodes contain data elements, or pointers to data elements(*record*)
  - Each node also has a **key attribute**(*key*)
  - The set of key attributes is **totally ordered** (*a≤b*)
  - Search is done by key comparison

- For every node with key attribute *s*, we have:
  - All keys in the **left** subtree are **smaller** than *s*
  - All keys in the **right** subtree are **greater** than *s*

- The subtrees are also binary search trees

- See elementary methods in the slides of lecture 7!

What happens if there are multiple objects with the same key?
Tips on U7.A2 – Binary Search Tree

Subtask a (by hand)
Delete, replace smallest element of the right subtree

Subtask b
Implementation of a binary search tree
IBinarySearchTreeUtils<T>

UtilsFactory.create() should generate a Utils for the type String →
new MyTreeUtils<String>();

UnlinkSmallestResult<T> contains the result of unlinkSmallest():
the smallest element and the rest of tree (i.e., one pair)
Tips on U7.A2 – Binary Search Tree

Methods to implement:

- height, isLeaf, hasOneChild
- preOrder, inOrder, postOrder
- insert
- find
- unlinkSmallest & remove
Tips on U7.A3 – Reversi

- This task starts a series, that aims to implement Reversi Player

- Rules and more information:
  - http://www.vs.inf.ethz.ch/edu/FS2013/I2/reversi
  - Login for reversi-papers:
    - username: i2bib
    - password: reversi
Tips on U7.A3 – Reversi

- Reversi tournament at the end of the semester
- Great awards!

In case of problems with the framework:
1. Documentation
2. Me
3. Simon Mayer (simon.mayer@inf.ethz.ch)
Tips on U7.A3 – Reversi

- First, the basic principles of the game are to be implemented

- Later, strategies are developed to improve the game of the computer player
  - Optimal search (MinMax, Alpha-Beta, …)
  - Game theory
Tips on U7.A3 – Reversi

- Resources are found on the Reversi website
- Note the installation instructions (Eclipse >3.2)
- HumanPlayer in u7a3
- Trick:
  - You first create an Abstract class (PlayerBase,…etc)
  - Implement functions for your different players in your general useful helper functions
Tips on U7.A3a – Play!

- Setup Framework
- Play a game against your team mate (or yourself)
- Take snapshot
package reversi;
public interface ReversiPlayer
{
    void initialize(int myColor, long timeLimit);
    Coordinates nextMove(GameBoard gb);
}

package randomTeam;
public abstract class PlayerBase implements ReversiPlayer
{
    private int m_color = 0;
    private long m_timeout = 0;
    protected final int getColor() { return m_color; }
    protected final long getTimeout() { return m_timeout; }
    ...
    protected abstract void foo();
}

package randomTeam;
public class RandomPlayer extends PlayerBase {
    protected void foo() { … }
    …
}
Tips on U7.A3b – RandomPlayer

- Implement a computer player, that randomly selects a valid move
- Possible strategy (naïve)
  - Pick a random move
  - Then check whether it is valid or not
    - If valid → return
    - If not valid → ?
- Possible strategy
  - In an array, mark all possible moves
  - Randomly select a move from this array
    - Extremely more efficient
    - Standard approach afterwards → evaluate moves
Have Fun!