Informatik II
Übung 07

TA: Marian George, E-mail: marian.george@inf.ethz.ch
Distributed Systems Group, ETH Zürich
Solution U6.A1 – Classes and Interfaces

Remark:
empty triangle with arrows!

Can be instantiated:
Non-abstract classes (D, E, F)
Solution U6.A1 – Classes and Interfaces

Type casts

- Static (*implicit cast*):
  
  only subclasses to parent classes

- Dynamic (*explicit cast*):

  ```java
  T t = (T)obj;
  
  valid, if the actual object pointed to by reference obj is of type T (including all children of T)
  ```
Solution U6.A1 – Classes and Interfaces

Interfaces vs. Abstract Class: why Interfaces?

- Functionality is an important point in the program. *what* is done *where* and *who* has access?

- Interfaces represent exactly this concept: It is guaranteed, *what* is done exactly and the interface defines it (*who* and *where*). The implementation (*how*) is completely irrelevant.
Solution U6.A2 – IStack expanded

Eclipse DEMO
/**
 * Inserts a value into a sorted list so that
 * the resulting list is still sorted.
 * The sort order is ascending.
 */
private GenericList insertSorted(GenericList list, Object value)
{
    if (list == null)
        return new GenericList(value, null);

    Comparable lhs = (Comparable) value;
    Comparable rhs = (Comparable) list.value;
    if (lhs.smallerThan(rhs))
        return new GenericList(value, list);

    list.next = insertSorted(list.next, value);
    return list;
}
EXERCISE 7

Hints
**Tips on Ex7.Q1– Generics**

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

  \[
  \text{MyType Elem} = (\text{MyType}) \text{Kollektion.getNext()};
  \]
  such casts can lead to runtime ClassCastException

**Better this way:**

\[
\text{Object obj} = \text{Kollektion.getNext()};
\]

\[
\text{if ( obj instanceof MyType )}
\]

\[
\text{doSomething( (MyType)obj );}
\]
Tips on Ex7.Q1– 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` *and* `pair.second`

- An object *pair of type* `MyPair<Integer>` *contains two Integer references*: `pair.first` *and* `pair.second`

- Advantage of generics:
  - Type testing through the compiler and its acquisition through the container → No dynamic casting is necessary
Tips on Ex7.Q1– Generics

- ArrayList Container

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

- Filter: „can obtain Testat”
Tips on Ex7.Q1– 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
   ▪ No Generics: ArrayList as raw type (compiler warnings)
   ▪ Filter out all students who do not have enough points for the Testat...
   ▪ When taking them out first from ArrayList, then cast to Student

c. Implementation of filterGeneric
   ▪ 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)
Hints to Ex7.Q2 – Tic-Tac-Toe

- Reflections on game trees…

- Think about how the attribute of a node is calculated based on the attributes of the successor, when you are for example in your opponent’s place
Tips on Ex7.Q3 – 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 Ex7.Q3 – Binary Tree

Pre-order: F, B, A, D, C, E, G, I, H

In-order: A, B, C, D, E, F, G, H, I

Post-order: A, C, E, D, B, H, I, G, F
Tips on Ex7.Q3 – 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 Ex7.Q3 – 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 \((a \leq b)\)
  - 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 Ex7.Q3 – 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 Ex7.Q3 – Binary Search Tree

Subtask b

**UnlinkSmallestResult<T> contains the result of unlinkSmallest():**
the smallest element and the rest of tree (i.e., one pair)

Case#1: no children

![Tree diagram for Case#1: no children]

Case#2: one child

![Tree diagram for Case#2: one child]

http://www.algolist.net/Data_structures/Binary_search_tree/Removal
Tips on Ex7.Q3 – Binary Search Tree

Subtask b

UnlinkSmallestResult<T> contains the result of unlinkSmallest(): the smallest element and the rest of tree (i.e., one pair)

Case#3: two children

http://www.algolist.net/Data_structures/Binary_search_tree/Removal
Tips on Ex7.Q3 – Binary Search Tree

Methods to implement:

- height, isLeaf, hasOneChild
- preOrder, inOrder, postOrder
- insert
- find
- unlinkSmallest & remove
Tips on Ex7.Q4 – Reversi

- This task starts a series, that aims to implement Reversispieler

- Rules and more information:
Tips on Ex7.Q4 – Reversi

Starting position

Where dark may play

After dark play

Where light may play

After light play
Tips on Ex7.Q4 – Reversi

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

In case of problems with the framework:
1. Documentation
2. Me
3. Leyna Sadamori (leyna.sadamori@inf.ethz.ch)
Tips on Ex7.Q4 – 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 Ex7.Q4 – Reversi

- Resources are found on the Reversi website
- Note the installation instructions (Eclipse >3.2)
- HumanPlayer in u7a4
- Trick:
  - You first create an Abstract class (PlayerBase,…etc)
  - Implement functions for your different players in your general useful helper functions
Tips on Ex7.Q4a – Play!

- Setup Framework
- Play a game against your team mate (or yourself)
- Take snapshot
**Tips on U7.A3b – ReversiPlayer**

```java
package reversi;
public interface ReversiPlayer {
    void initialize(int myColor, long timeLimit);
    Coordinates nextMove(GameBoard gb);
}
```

```java
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();
}
```

```java
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 $\rightarrow$ return
    - If not valid $\rightarrow$ ?
- Possible strategy
  - In an array, mark all possible (valid) moves
  - Randomly select a move from this array
    - Extremely more efficient
    - Standard approach afterwards $\rightarrow$ evaluate moves
...have fun!