Informatik II (D-ITET)

Tutorial 7

TA: Hông-Ân Cao, E-mail: hong-an.cao@inf.ethz.ch
Distributed Systems Group, ETH Zürich
Outlook

- **Exercise 6 solution discussion**
- **Exercise 7 hints**
Solution Ex6.Q1 – Classes and Interfaces

Can be instantiated:
Non-abstract classes (D, E, F)
Solution Ex6.Q1 – Classes and Interfaces

- Type casts
  - Implicit cast: only subclasses to parent classes
    \[ \text{superclass} = \text{subclass} \]
  - Explicit cast: \( T \ t = (T)\text{obj} \);
    Valid, if the actual object pointed to by the reference \( \text{obj} \) is of type \( T \) (including all parents of \( T \))

- Link resolution:
  - Static: Apparent type of the variable
  - Dynamic: Effective type of variable → Java

```
public class Animal {
    protected boolean dead;
    public void die() {
        dead = true;
    }
}

public class Dog extends Animal {
}

public class Phoenix extends Animal {
    public void die() {
        dead = false;
    }
}

public class Killer {
    public void kill(Animal a1, Animal a2) {
        a1.die();
        a2.die();
    }
}

Killer evil = new Killer();
Dog poorDog = new Dog();
Phoenix poorPhoenix = new Phoenix();
evil.kill(poorDog, poorPhoenix);
```

Source: Introduction to OOP for I&C, J. Sam, EPFL
Solution Ex6.Q1 – Classes and Interfaces

- Static casts

```java
void c1() {
    D d = new D();
    A a = d;
    B b = d;
    C c = d;
    // E e = d;
    // F f = d;
}
```

```java
void c2() {
    F f = new F();
    A a = f;
    // B b = f;
    C c = f;
    // D d = f;
    // E e = f;
}
```

```java
void c3() {
    E e = new E();
    A a = e;
    B b = e;
    // C c = e;
    // D d = e;
    // F f = e;
}
```

- Dynamic casts

```java
public static void d1() {
    A a = new D();
    B b = (B) a;
    C c = (C) a;
    D d = (D) a;
    // E e = (E) a;
    // F f = (F) a;
}
```

```java
public static void d2() {
    A a = new E();
    B b = (B) a;
    // C c = (C) a;
    // D d = (D) a;
    E e = (E) a;
    // F f = (F) a;
}
```

```java
public static void d3() {
    B b = new D();
    A a = (A) b;
    C c = (C) b; // i.e. cross-cast!
    D d = (D) b;
    E e = (E) b;
}
```
Solution Ex6.Q1 – 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.

Reminder of abstraction in your program:
- Use a class when the relationship "is-a" can be applied to your object
- Each attribute of a class is justifiable because your object "has-a" propriety
- An interface comes in handy because your object "behaves-as-a"
Solution Ex6.Q2 – IStack expanded

```java
public class StackFactory {
    public static IStack create() {
        return new ListStack();
    }
}

public interface IStack {
    public boolean empty();
    //…
}

public class ListStack implements IStack {
    public boolean empty() {
        return list == null;
    }
    //…
}

public class Tests {
    /**
     * Test for empty method
     */
    @Test public void empty() {
        IStack stack = StackFactory.create();
        Assert.assertTrue(stack.empty());
        stack.push(42);
        Assert.assertFalse(stack.empty());
    }
    //…
}
```
Solution Ex6.Q3 – Generic Lists

/**
 * 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;
}

generic objects compared through interfaces

public GenericList sort(GenericList list) {
    if (list == null) return null;
    return insertSorted(sort(list.next), list.value);
}
 Solution Ex6.Q3 – GeometricObject

```java
/**
 * abstract class for geometric objects
 */
public abstract class GeometricObject implements Comparable {
    public abstract int area();

    public boolean smallerThan(Comparable rhs) {
        GeometricObject other = (GeometricObject) rhs;
        return this.area() < other.area();
    }
}

public class Rectangle extends GeometricObject {
    private int a;
    private int b;

    public String toString() {
        return String.format("Rectangle(%d,%d)",
                            a, b);
    }

    public int area() {
        return a * b;
    }

    public Rectangle(int base, int height) {
        this.a = base;
        this.b = height;
    }
}
```
public boolean empty() {
    return size() == 0;
}

public int peek() throws EmptyStackException {
    if (empty()) {
        throw new EmptyStackException();
    }
    return chunks.buffer[used-1];
}

public int pop() throws EmptyStackException {
    int value = peek();
    used--;

    if (used == 0 && chunks.size() > 1) {
        chunks = chunks.removeChunk();
        used = ChunkList.chunkSize;
    }

    return value;
}

public void push(int number) {
    if (used >= ChunkList.chunkSize) {
        chunks = chunks.addChunk();
        used = 0;
    }

    chunks.buffer[used] = number;
    used++;
}

public int size() {
    return (chunks.size()-1) * ChunkList.chunkSize + used;
}
Outlook

- Exercise 6 solution discussion
- Exercise 7 hints
Hints Ex7.Q1 – Generics

- Ex6 Generics
  - All classes inherit from `Object` *(abstract base class)*
  - cast when extended from container (here List)
    ```java
    MyType Elem = (MyType) Collektion.getNext();
    such casts can lead to runtime ClassCastException
    ```
  Better this way:
  ```java
  Object obj = Collektion.getNext();
  if( obj instanceof MyType )
    doSomething( (MyType)obj );
  ```
Hints Ex7.Q1 – Generics

- Ex7 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
Hints 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"
Hints Ex7.Q1 – Generics

- **FilterFactory** and (empty) IFilter implementation
  - Input: `ArrayList` of groups, that are actually `ArrayList` of `Student`.
  - Output: `ArrayList` of `Student` obtaining the Testat.

- 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`

- 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 Ex7.Q2 – Tic Tac Toe

- Easy, theoretical exercise: Draw a game tree given the following game state:

- Mark all situations (starting from the bottom) with \{-1, 0, 1\} depending on the possible outcome of the game.

- What is the optimal move?

Hints Ex7.Q3 – Binary Tree

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

- Recursive traversal: \((W = \text{Würzel} = \text{Root})\)
  - Pre-order: W-L-R
  - In-order: L-W-R
  - Post-order: L-R-W
Hints 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)"

Source: http://en.wikipedia.org/wiki/Binary_tree
Hints 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 ≤ 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!
Hints 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)
Hints Ex7.Q3 – Binary Search Tree

Methods to implement:

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

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

- Rules and more information:
  - Platform: https://reversi.ethz.ch/
  - FAQ: https://reversi.ethz.ch/faq.php

- Credentials for the documents:
  - Username: i2bib
  - Password: reversi
Hints 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)
Hints 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
Hints Ex7.Q4 – Reversi

- Resources are found on the Reversi website
  - Import reversi.jar (Buildpath > Add External Archive)
  - Link the reverse-doc.zip Javadoc to the jar (Right click on reversi.jar > Properties > Javadoc Location > Javadoc in archive. Specify the path to the zip file and select the Javadoc root folder within the archive)

- 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
Hints Ex7.Q4a – Play!

- Setup Framework
- Play a game against your team mate (or yourself)
- Take a snapshot
Hints Ex7.Q4b – ReversiPlayer

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

```java
package randomTeam;
import reversi.*;
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 getTimout() { return m_timeout; }
    //…
    protected abstract void foo();
}
```

```java
package randomTeam;
import reversi.*;
public class RandomPlayer extends PlayerBase {
    protected void foo() { //… }
    //…
}
```
Hints Ex7.Q4b – RandomPlayer

- Implement a computer player, that \textit{randomly} selects a \textit{valid} move
- Possible strategy (naive)
  - 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 moves
  - Randomly select a move from this array
    - Extremely more efficient
    - Standard approach afterwards$\rightarrow$ evaluate moves
Have fun!

Source: [https://xkcd.com/974/](https://xkcd.com/974/)