Informatik II
Tutorial 7
Mihai Bâce
mihai.bace@inf.ethz.ch
Overview

- Debriefing Exercise 6
- Briefing Exercise 7
U6.A1 – Classes and interfaces

Can be instantiated:
Non-abstract classes (D, E, F)
Type casts

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

- **Dynamic (*explicit cast*)**: valid, if the actual object pointed to by reference `obj` is of type `T` (including all children of `T`)

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

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 U6.A2 – IStack expanded

```java
public interface IStack {
    public void push(int number);
}

public class ListStack implements IStack {
    public void push(int number) {
        list = new List(number, list);
        size += 1;
    }
}

public class StackFactory {
    public static IStack create() {
        return new ListStack();
        //return new u6a5.ChunkedStack();
    }
}

@Test public void push() {
    IStack stack = StackFactory.create();
    ...
}
U6.A3 Implementing the Comparable interface

```java
package u6a3;

/**
 * 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();
    }
}
```
U6.A3 Extending the GeometricObject class

```java
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;
    }
}
```

```java
public class Triangle extends GeometricObject {
    private int base;
    private int height;

    public String toString()
    {
        return String.format("Triangle(%d,%d)", base, height);
    }

    public int area()
    {
        return base * height / 2;
    }

    public Triangle(int base, int height)
    {
        this.base = base;
        this.height = height;
    }
}
```
U6.A3 Sorting a GenericList

```java
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;
}
```

```java
public GenericList sort(GenericList list)
{
    if (list == null) return null;
    return insertSorted(sort(list.next), list.value);
}
```
Overview

- Debriefing Exercise 6
- Briefing Exercise 7
Java Generics

- Generics allow parameterization of types
  - Input to formal parameters are values (e.g. f(int a))
  - Input to type parameters are types e.g. ArrayList<T>

- Reuse the same code for different input types
- Same algorithm
  - E.g. sorting Integers, Floats, Students etc.

- Stronger type checks at compile time
  - Compile-time errors are easier to fix than run-time errors
  - No need to typecast

- Code is easier to maintain and read
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`
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 way:

```java
Object obj = Kollektion.getNext();
if( obj instanceof MyType )
doSOMething((MyType)obj);
```
U07.A01

- ArrayList and Generics
  - Each group consists of multiple students:
    ```java
    ArrayList<Student> group
    ```
  - There are multiple groups of students:
    ```java
    ArrayList<ArrayList<Student>> groups;
    ```

a) Implement factory method
b) Implement filterRaw (without generics: ArrayList)
c) Implement filterGeneric (using Generics: ArrayList<Student>)
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)
U7.A2 Tic-Tac-Toe

- Draw 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?
Reminder: Binary Trees

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

- Recursive traversal:
  - Preorder: W-L-R
  - Inorder: L-W-R
  - Postorder: L-R-W
Tree traversal:

Inorder: A, B, C, D, E, F, G, H, I
Postorder: A, C, E, D, B, H, I, G, F
Binary Search Tree

- **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
U7.A3 Binary Search Trees

a) Delete elements 15, 12, 20

a) Implement `IBinarySearchTreeUtils<T>` and `UtilsFactory.create`

height
isLeaf
hasOneChild
preOrder
inOrder
postOrder
insert
find
remove
U7.A4 Reversi

- Game


- Ongoing series until the end of the semester

- Tournament at the end!

- Cool prizes!
Cool prizes?

3x Raspberry Pi 3 Model B  
+ Gehäuse + Speicherkarte

3x Reversi-Brettspiel aus Holz

2x Amphiro b1

1x Fitbit Aria

3x Google Chromecast

4x Tile Dongles

1x Fitbit Flex
U7.A4 Reversi

a) Reversi framework
   - Setup the framework
   - Play a game against your team mate (or yourself)
   - Take snapshot

b) Implement a Random Player
   - 2 strategies
     1. Find a random move. If valid accept, otherwise?
     2. Find all possible moves. Choose one at random.
How to do it?

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

```java
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
public class RandomPlayer extends PlayerBase {
    protected void foo() {
        ...
    }
}
```
U7.A4 Reversi Questions

- Check the documentation

- E-mail me first!

- Reversi Coordinator
  - Alexander Viand: alexander.viand@inf.ethz.ch
Have Fun!
Tree traversal...

```javascript
preOrder(node) {
    print(node)
    if left != null then preOrder(left)
    if right != null then preOrder(right)
}
```

- Pre-Order (root, left, right) 8, 3, 1, 6, 4, 7, 10, 14, 13
- In-Order (left, root, right)
- Post-Order (left, right, root)
Tree traversal...

```javascript
inOrder(node) {
  if left != null then inOrder(left)
  print(node)
  if right != null then inOrder(right)
}
```

- **Pre-Order (root, left, right)**
  8, 3, 1, 6, 4, 7, 10, 14, 13

- **In-Order (left, root, right)**
  1, 3, 4, 6, 7, 8, 10, 13, 14

- **Post-Order (left, right, root)**
Tree traversal...

- **Pre-Order (root, left, right)**
  
  \[
  \text{postOrder(node)} \{
  \text{if left} \neq \text{null then postOrder(left)}
  \text{if right} \neq \text{null then postOrder(right)}
  \text{print(node)}
  \}
  
  8, 3, 1, 6, 4, 7, 10, 14, 13

- **In-Order (left, root, right)**

  1, 3, 4, 6, 7, 8, 10, 13, 14

- **Post-Order (left, right, root)**

  1, 4, 7, 6, 3, 13, 14, 10, 8