Informatik II (D-ITET)
Tutorial 8

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

- Exercise 7: Solution discussion
- Exercise 8: Overview (Generics, Binary Search, Reversi)
Solution Ex7.Q1 – IFilter

Helper Method

```java
private boolean filter(Student student) {
    return student.getPoints() >=
    (IFilter.criteria / 100 * IFilter.maxNumberOfPoints);
}
```
Solution Ex7.Q1 – IFilter

Possible Solution - filterRaw

```java
public ArrayList filterRaw( ArrayList groups ) {
    ArrayList result = new ArrayList();
    for( int i = 0; i < groups.size(); ++i){
        ArrayList group = (ArrayList) groups.get(i);
        for( int j = 0; j < group.size(); ++j ){
            Student student = (Student) group.get(j);
            if( filter( student ) ) {
                result.add( student );
            }
        }
    }
    return result;
}
```
Solution Ex7.Q1 – IFilter

Possible Solution - filterGeneric

```java
public ArrayList<Student> filterGeneric(ArrayList<ArrayList<Student>> groups){
    ArrayList<Student> result = new ArrayList<Student>();
    for( int i = 0; i < groups.size(); i++ ){
        ArrayList<Student> group = groups.elementAt(i);
        for( int j = 0; j < group.size(); j++ ){
            Student student = group.elementAt(j);
            if( filter( student ) ) {
                result.add( student );
            }
        }
    }
    return result;
}
```
For-each loop

The basic for loop was extended in Java 5 to make iteration over arrays and other collections more convenient

- **Arrays and Collections**
  - for-each is commonly used to iterate over an array or a Collection `Iterable<E>`. It can also iterate over anything that implements the `Iterable<E>` interface.
  - Many of the Collections classes (e.g., `ArrayList`) implement `Iterable<E>`, which makes the *for-each* loop very useful. You can also implement `Iterable<E>` for your own data structures.

Source: [http://download.oracle.com/javase/tutorial/collections/interfaces/index.html](http://download.oracle.com/javase/tutorial/collections/interfaces/index.html)
[http://leepoint.net/notes-java/flow/loops/foreach.html](http://leepoint.net/notes-java/flow/loops/foreach.html)
for-each loop

Example:

for-each loop

```java
for( type var : arr ) {
    //body of loop
}
```

Equivalent for loop

```java
for( int i = 0; i < arr.length; i++ ){
    type var = arr[i];
    //body of loop
}
```

```java
for( type var : coll ) {
    //body of loop
}
```

```java
while(coll.iterator().hasNext()){
    type var = iter.next();
    //body of loop
}
```
for-each loop

Interface Collection<T>

We say: “for each currentThing (of type T) in myCollectionOfThings (do) funnyMethod”

```java
Collection<T> myCollectionOfThings = ... ; // fill with objects

for( T currentThing : myCollectionOfThings ){
    funnyMethod( currentThing );
}
```
for-each loop

Although the enhanced *for* loop can make code much clearer, it can't be used in some common situations.

- **Only access.** Elements can not be assigned to, e.g., not to increment each element in a collection.
- **Only single structure.** It's not possible to traverse two structures at once, e.g., to compare two arrays.
- **Only single element.** Use only for single element access, e.g., not to compare successive elements.
- **Only forward.** It's possible to iterate only forward by single steps.
- **At least Java 5.** Don't use it if you need compatibility with versions before Java 5.
Solution Ex7.Q1 – IFilter

Possible solution with for-each loops

```java
public ArrayList<Student> filterGeneric(ArrayList<ArrayList<Student>> groups){
    ArrayList<Student> result = new ArrayList<Student>();

    for(ArrayList<Student> group : groups ) {
        for( Student student : group ) {
            if( filter( student ) ) {
                result.add( student );
            }
        }
    }

    return result;
}
```
Solution Ex7.Q2 – Tic-Tac-Toe
Solution Ex7.Q3 – BinarySearchTreeUtils\(\langle T\rangle\)

Remove 15

Remove 12
Solution Ex7.Q3 – BinarySearchTreeUtils<T>

Remove 20

Final outcome
Solution Ex7.Q3 –
BinarySearchTreeUtils<T>

```java
public int height(BinarySearchTree<T> tree) {
    if (tree == null) {
        return 0;
    }
    return 1 +
            Math.max(height(tree.left),
                       height(tree.right));
}

public boolean isLeaf(BinarySearchTree<T> tree){
    return (tree.left == null &&
            tree.right == null);
}

public boolean hasOneChild(BinarySearchTree<T> tree){
    return (tree.left != null) ^
           (tree.right != null);
}

private ArrayList<T> postOrder(BinarySearchTree<T> tree, ArrayList<T> arrayList){
    if (tree == null) {
        return arrayList;
    }
    arrayList = postOrder(tree.left,
                            arrayList);
    arrayList = postOrder(tree.right,
                            arrayList);
    arrayList.add(tree.thing);
    return arrayList;
}

public ArrayList<T> postOrder(BinarySearchTree<T> tree) {
    return postOrder(tree,
                     new ArrayList<T>());
}
```
Solution Ex7.Q3 – BinarySearchTreeUtils<T>

```java
class BinarySearchTreeUtils<T> {
    public BinarySearchTree<T> insert(BinarySearchTree<T> tree, int key, T thing) {
        if (tree == null) {
            return new BinarySearchTree<T>(key, thing);
        } else if (tree.key == key) {
            tree.thing = thing;
        } else if (key < tree.key) {
            tree.left = insert(tree.left, key, thing);
        } else {
            tree.right = insert(tree.right, key, thing);
        }
        return tree;
    }

    public T find(BinarySearchTree<T> tree, int key) {
        if (tree == null) { return null; }
        if (key == tree.key) { return tree.thing; }
        if (key < tree.key) { return find(tree.left, key); }
        return find(tree.right, key);
    }

    public ArrayList<T> preOrder(BinarySearchTree<T> tree, ArrayList<T> arrayList) {
        if (tree == null) { return arrayList; }
        arrayList.add(tree.thing);
        arrayList = preOrder(tree.left, arrayList);
        arrayList = preOrder(tree.right, arrayList);
        return arrayList;
    }

    public ArrayList<T> preOrder(BinarySearchTree<T> tree) {
        return preOrder(tree, new ArrayList<T>());
    }

    public ArrayList<T> inOrder(BinarySearchTree<T> tree, ArrayList<T> ArrayList) {
        if (tree == null) { return ArrayList; }
        ArrayList.add(tree.thing);
        ArrayList = inOrder(tree.left, ArrayList);
        ArrayList = inOrder(tree.right, ArrayList);
        return ArrayList;
    }

    public ArrayList<T> inOrder(BinarySearchTree<T> tree) {
        return inOrder(tree, new ArrayList<T>());
    }
}
```
Solutions Ex7.Q3 – BinarySearchTreeUtils<T>

- If the node to be deleted only has one child → straightforward
- If the node to be deleted has two children:
  - Locate the smallest key (min) in the right subtree.
    - If the root of the subtree has no left child, stop there
    - Else, keep traversing the left side until the node has no left child

public UnlinkSmallestResult<T> unlinkSmallest(BinarySearchTree<T> tree){
    if (tree.left == null) {
        return new UnlinkSmallestResult<T>(tree, tree.right);
    }
    UnlinkSmallestResult<T> result = unlinkSmallest(tree.left);
    tree.left = result.tree;
    return new UnlinkSmallestResult<T>(result.smallest, tree);
}

public BinarySearchTree<T> remove(BinarySearchTree<T> tree, int key){
    if (tree == null) { return null; }
    if (tree.key == key) {
        if (isLeaf(tree)) { return null; }
        if (hasOneChild(tree)) {
            if (tree.left != null) {
                return tree.left;
            }
            return tree.right;
        }
        UnlinkSmallestResult<T> result =
            unlinkSmallest(tree.right);
        result.smallest.right = result.tree;
        result.smallest.left = tree.left;
        return result.smallest;
    }
    if (key < tree.key) {
        tree.left = remove(tree.left, key);
    } else {
        tree.right = remove(tree.right, key);
    }
    return tree;
}

class UnlinkSmallestResult<T> {
    public BinarySearchTree<T> smallest;
    public BinarySearchTree<T> tree;

    public UnlinkSmallestResult(BinarySearchTree<T> smallest, BinarySearchTree<T> tree) {
        this.smallest = smallest;
        this.tree = tree;
    }
}
Solution Ex7.Q4b – RandomPlayer

```java
public class RandomPlayer implements ReversiPlayer {
    private int color = 0;
    private Random rand = new Random();

    public void initialize(int color, long timeout) {
        this.color = color;
    }

    public Coordinates nextMove(GameBoard gb) {
        ArrayList<Coordinates> validMoves = new ArrayList<Coordinates>(
            gb.getSize() * gb.getSize());
        for (int row = 1; row <= gb.getSize(); row++) {
            for (int col = 1; col <= gb.getSize(); col++) {
                Coordinates coord = new Coordinates(row, col);
                if (gb.checkMove(color, coord)) {
                    validMoves.add(coord);
                }
            }
        }
        if (validMoves.size() > 0) {
            int randIndex = rand.nextInt(validMoves.size());
            return validMoves.get(randIndex);
        } else {
            return null;
        }
    }
}
```
Outlook

- Exercise 7: Solution discussion
- Exercise 8: Overview (Generics, Binary Search, Game Tree, Reversi)
Hints to Sheet 8

- Generics (Part 2)
- Binary Search
- 0-1 Knapsack Problem
- Reversi (Part 2)
More Generics: extends

Example from last tutorial session

- class MyPair<T> { public T first, second; }
- An Object pair from the type MyPair<Float> contains two Float-References: pair.first and pair.second

Sort by MyPairs after pair.first?
- Already seen: After Comparable casting
  - Requires many Checks (or Exception-handling)
- Better
  - class MyPair<T extends Comparable<T>> { ... }
  - For T only types can be used, the Comparable<T> implements (e.g Float, Integer, String)
  - pair.first is now determined int compareTo(T other)
  - Without Casts
  - Compiler treats all exceptions (Compilation errors)
More Generics: Maps

- Often indexed data is needed
  - Particular for AHV-Number (unique id)
  - Document Identifiers (e.g. file name)
  - …

- Such data structures are called Maps
  - Identifier (key) is "mapped" on content (value)
  - In the java standard library
    - interface Map<Key extends Comparable<Key>, Value>
    - Implementation: TreeMap, HashMap,…
Hints to Ex8.Q1 – Binary Search

- Binary Search Algorithm (illustrated)

- Decision Tree

```
query: 16
sub-array-length: 1
return: null
```

```
query: 23
sub-array-length: 2
return: "23"
```
Hints to Ex8.Q1 – Binary Search

- Draw the decision tree and have some thoughts
  - Superposition, factors

- Implementation:
  - `find(List<Unit<Key, Value>> haystack, Key needle)`
  - `setFactor(int factor)`
    - Generalize the search → unbalanced search trees
  - `getNumberOfCalls()`
    - Benchmarking with various factors
    - Average # of recursive calls to various factors
Ex8.Q2– 0-1 Knapsack problem and Backtracking
Hints Ex8.Q2

The general 0-1 Knapsack problem

- k items $x_1, \ldots, x_k$ and each has known value and weight
- Choice of items, such that total weight is not exceeded
- Optimization problem: Maximize the value of the chosen items
  a) Theory
  b) Brute-force approach
  c) Backtracking approach
  d) Comparison of Brute-force and Backtracking approaches

Can a value of at least $V$ be achieved without exceeding the weight $W$?

What is the (valid) configuration that maximizes the value?

Decision problem (NP-complete)

Optimization problem (NP-hard)
How many different possibilities does our thief have?
- **S** = Set of items at our disposal
- The thief can only take a subset home
- The thief can also choose the empty subset \( \phi \) (lazy thief) or the whole set \( S \) (strong thief with big bag)!

- #items:= #elements in the power set of \( S \)
- Example:
  - \( S = \{x_1, x_2\}, |S| = K = 2 \)
  - 4 Subsets: \( \phi, \{x_1\}, \{x_2\}, \{x_1, x_2\} \)
Ex8.Q2 – Backtracking

- What does "Backtracking" mean?
  - Principle: "trial and error"

- Example: Looking for a maze exit
  - Decide upon a direction
  - Continue in this direction
  - If eventually unsuccessful
    - Return and choose another direction
  - If eventually successful
    - Done…

In case all directions were tried → keep going back.
Ex8.Q2 – A simple strategy for the thief

- Implement simple algorithm:
  - \( W = \max(\sum_{i=0}^{K-1} b_i w_i), \sum_{i=0}^{K-1} b_i g_i \leq G \)

- With \( b_i = \begin{cases} 
0, & \text{if item } x_i \text{ is not packed} \\
1, & \text{if item } x_i \text{ is packed} 
\end{cases} \)

- As a reminder:
  - A set \( S \) with \( |S| = K \) contains \( 2^K \) subsets
Ex8.Q2 – A simple strategy for the thief

- Apply the implementation in pseudo-code

1. Initialization
2. Take the next configuration (how exactly?)
3. Compute the overall weight
   - if (overall weight < W)
     - Compute overall value
       - if (new overall value > overall value of current optimal solution)
         - Current configuration becomes optimal solution
4. If more configurations remain ,
   - Go to point 2
   else
   - Computation done
Ex8.Q2.a – A simple strategy for the thief

- Does the thief's strategy always yield the optimal results?
  - Yes/No
  - Why?...

- Is there always exactly one optimal solution?
  - Yes/No
  - Why?...
Ex8.Q2.b,c – Bit value

- Configuration as a sequence of bits: class Selection
- The bit value describes the value of single bits, which depends on their position in the overall binary form of a number.

MSB - Most Significant Bit/Byte
- The most significant bit (MSB, also called the high-order bit) is the bit position in a binary number having the greatest value.

LSB - Least Significant Bit/Byte
- The least significant bit (LSB) is the bit position in a binary integer having the smallest value.
Ex8.Q2.b,c – Hints for the implementation

- class Selection is well documented
- Beware: If you increase the configuration (if you put a new item in the bag, Q2.c), the new status has to be updated
- Example of selections for $S$
  - $M = \{x_1, x_2, x_3, x_4\}, |S| = K = 4$
  - Subsets: $\emptyset, \{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}, \{x_1, x_2\}, \{x_1, x_3\}, \ldots$

(b_1, b_2, b_3, b_4) = (0,0,0,0)
(b_1, b_2, b_3, b_4) = (0,0,0)
(b_1, b_2, b_3, b_4) = (1,0,0,0)
(b_1, b_2, b_3, b_4) = (1,0,1,0)
Ex8.Q2.b,c – Hints for the implementation

//Bruteforce approach

public Selection findBest(ArrayList<Integer> values, ArrayList<Integer> weights, int maxWeight)
{
    int last = java.Math.pow(2, values.size());
    ...

    //Number of subsets
    for( int i = 0; i < last; i++ ){
        new Selection(values.size(), i);
        //Selection bit field with value I
        ...
    }
    ...
}
Backtracking approach:

- **FindResult** Class (Selection and Value together)
- Recursive method:

  ```java
  FindResult fr = find(currSelection, currWeight, values, weights, maxWeight);
  ```

- End condition: `selection.size() == values.size();` //all things considered
- In the method: two directions to continue
  
  //Leave item
  Selection without = new Selection(...); //Increase by one, set bit to 0
  //then continue down the tree

  //Check whether weight is OK, then take item
  ...
  Selection with = new Selection(...); //Increase by one, set bit to 1
  //then continue down the tree
Hints to Ex8.Q3 – Reversi (Part 2)

HumanPlayer

nextMove()
Wait for input from The command line

RandomPlayer

nextMove()
Random selection (but valid move!)
Next move

GreedyPlayer

nextMove()
Select the next move using a simple, None-recursive Evaluation function

Download
Exercise 7
Exercise 8
Hints to Ex8.Q3 – Reversi (Part 2)

a. Implementing `ICheckMove` without Framework-Function. Ideas?

b. Implement a player that selects the best move among all possible moves

   `Best move: Move, after its execution one owns max. more stones than the opponent: «Depth = 1»`

   Given: Game Tree is not needed!

   `Determining the best move: Copy Board (clone), run the move, count...`
Hints Ex8.Q3a – checkMove()

```java
boolean checkMove(GameBoard gb, int player , Coordinates c) {

    //Check all directions

    //Unless at least one direction is valid
    //...

    //GameBoard.checkMove is not allowed to be used !

}
```
Hints Ex8.Q3b – greedyPlayer()

- Simple computer opponents
  - Move selection: Best first
  - Search depth: 1 (my move)
  - Evaluation Function: The number of stones gained after the move

- Tips
  - A GameBoard can be copied with gb.clone()
  - You are allowed to use GameBoard.checkMove() here
  - With a more advanced function for selecting your next move, you’re facing a relatively better opponent (without Recursion)
Have fun!

https://xkcd.com/287/