Informatik II
Tutorial 5
Mihai Bâce
mihai.bace@inf.ethz.ch
Overview

- Debriefing Exercise 4
- Briefing Exercise 5
U4.A1 Stack

- Noteworthy
  - Two attributes: buffer length and size
  - capacity : buffer.length (Array indices from 0 to length-1)
  - empty : size == 0
  - size : index of first free space at the top

- void push(int value) { ... buffer[size++] = value; }
- grow
  - Conditions of Grow in push: size() == capacity()
  - Java-library functions (search and copy)
  - int[] Arrays.copyOf(int[] original, int newLength)
- JavaDoc
  - How it is documented!
U4.A2 Ackermann function

- Recursive definition

\[
A(0, m) = m + 1
\]
\[
A(n + 1, 0) = A(n, 1)
\]
\[
A(n + 1, m + 1) = A(n, A(n + 1, m))
\]
U4.A2 Pseudocode sample

while(stack.size() > 1) {
    ....
    if n = 0
        result = m + 1
    else if m = 0
        push(n - 1), push(1)
    else
        push(n - 1), push(n), push(m - 1)

the uppermost element from the stack is the result
U4.A3

- SourceCode-Bytecode, assignment clear?
- Order of parameters / return, clear?

| return $A(n-1, A(n, m-1))$ | 21: aload 0  
22: iload 1  
23: icall 1  
24: isub  
25: aload 0  
26: iload 1  
27: iload 2  
28: icall 1  
29: isub  
30: invokevirtual  
31: invokevirtual  
33: invokevirtual  
36: ireturn |
Data types

- **Primitive Types**
  - E.g. byte, int, float, char

- **Reference-Type**
  - E.g. Arrays, Strings, Classes
Call by

- Call by value
  - The method receives a copy of the variables
  - No connection between the data in the caller and the data in the function

- Call by reference
  - Instead of copying the data, you assign a reference to it
  - Method calls of a referenced object work on the same object which is visible from outside.
Call by value vs. call by reference

- In C++ both are possible
  - Call by value

- Call by reference

- Java is always call by value
  - This means, that when passing reference types, the address value is copied a local variable!
  - In case of transferring from a primitive types, the value would be copied in local copy.
JAVA: Call by reference vs. call by value

- Modification is possible, interchanging not

Overview

- Debriefing Exercise 4
- Briefing Exercise 5
U5 Lists

- Features:
  - Dynamic size - no initialization like in arrays
    - How do we determine the size?
    - How do we iterate over it?
    - When are we at the end of the list?

```
value    76
next

value    15
next

value    22
next

value    3
next

value    32
next

null
```

myList
**U5 Lists**

- `toString(List list)`

```java
public static String toString(List list) {
    if (list == null) {
        return "null";
    }
    return list.value + "," + toString(list.next);
}
```

```
myList
value: 76
next: 

value: 15
next: 

value: 22
next: 

value: 3
next: 

value: 32
next: 

null
```

`u5a1.Lists.toString(myList)`

`76,15,22,3,32,null`
U5A1 Lists – Implementation (1)

- add
  - Add a value to the front of the list

- size
  - Calculate the length of the list

- sum
  - Sum the values in the list

- last
  - End of list (last node before the zero, otherwise the zero)
U5A1 Lists – Implementation (2)

- sublist
  - "Sublist" from a given index

- valueAt
  - Return the value of a given index in the list

- index
  - Index of the first node with a given value

- Tip: Consider Helper functions (code reusability!)
  - E.g. nodeAt
    - Similar usability in sublist and valueAt
    - You use when manipulating the list as well…
  - Must also be recursively implemented!
U5.A2 More Lists

- **append**
  - Attach a value at the end of list

- **concat**
  - Attach a list to the back of another list

- **insertAt**
  - Insert an element to list after certain index

- **remove**
  - Delete a value in the list at certain position
U5A3 Sorting lists

- `insertSorted`
  - Insert a value in a sorted list

- `sort`
  - Sort a given list
Implement a stack using a list

- push – first element of the list is at the top of the stack
- pop – don’t forget to update the references
- peek
- empty
- size
Have Fun!