Informatik II (D-ITET)

Tutorial 3

TA: Anwar Hithnawi, E-mail: hithnawi@inf.ethz.ch
Distributed Systems Group, ETH Zürich
Outlook

- Exercise 2 solution discussion
- Exercise 3 overview (Strings, Syntax Diagram, Syntax checker)
Solution Ex2.Q1

- Bracket and indented form OK

\[ S(R(H(K)), P(A(N,O), Q,T), V(J,F(G))) \]

- Can you reconstruct the tree given by the comma representation shown above without any ambiguity?

Yes, if the position of the nodes is irrelevant (left or right successor?)

- Height, leaves, longest path/s!
Solution Ex2.Q2: recursiveSort()

```
[ 5 1 9 2 ]
[ 5 1 9 2 ]
[ 5 1 9 2 ]
[ 5 1 9 2 ]
[ 5 1 9 2 ]
[ 5 1 9 2 ]
[ 5 1 9 2 ]
[ 5 1 9 2 ]
[ 9 1 5 2 ]
[ 9 1 5 2 ]
[ 9 5 1 2 ]
[ 9 5 1 2 ]
[ 9 5 2 1 ]
[ 9 5 2 1 ]
```

```python
recursiveSort(4)
No need for further swap!
```
Solution Ex2.Q2

SWAP Function

```java
/**
 * swaps two fields of {@link RandomArray#numbers}
 * @param i a valid index into {@link RandomArray#numbers}
 * @param j a valid index into {@link RandomArray#numbers}
 */
private void swap(int i, int j)
{
    int tmp = numbers[j];
    numbers[j] = numbers[i];
    numbers[i] = tmp;
}
```
Solution Ex.Q2

- SWAP without temporary variable, any idea?
  (hint: using XOR)

\[
\begin{align*}
  x & := x \oplus y \\
  y & := x \oplus y \\
  x & := x \oplus y \\
  1010 \oplus 0011 & = 1001 \rightarrow x \\
  1001 \oplus 0011 & = 1010 \rightarrow y \\
  1001 \oplus 1010 & = 0011 \rightarrow x \\
  0011 & \quad 1010
\end{align*}
\]
Solution Ex2.Q2e

SWAP inside a loop: good idea?

```java
void recursiveSort( int until ) {
    // 0 elements are considered to be sorted
    if( until == 0 )
        return;

    // sort first until-1 elements in the array
    recursiveSort( until - 1 );

    // bring the greatest element from the rest to position until-1
    for( int i = until; i < a.length; i++ )
    {
        if( a[i] > a[until-1] ){
            swap(until-1, i);
        }
    }
}
```
Better: search first, then exchange

```java
void recursiveSort( int until ) {
    // 0 elements are considered to be sorted
    if( until == 0 )
        return;

    // sort first until-1 elements in the array
    recursiveSort( until - 1 );

    // find index of greatest element after until-1
    int maxIndex = until - 1;
    for( int i = until; i < a.length; i++ ) {
        if( a[i] > a[maxIndex] ) {
            maxIndex = i;
        }
    }

    // swap elements at maxIndex and until-1
    swap( until-1, maxIndex );
}
```

For 15 values, on average ~57 swaps in the first case and ~11 swaps in the second case.
Solution Ex2.Q2

- Coding Style – Formatting

Eclipse: *Ctrl+Shift+F* and the code is nicely formatted (indented)

```java
while ((e+i)<=14) {
    if (a[e]> a[e+i]) {
        e++;
        i=1;
    }
    else
        i++;
}
```

In languages like java/C++ indentation is not a requirement but it better conveys the structure of the programs to human readers.
Solution Ex2.Q2

Coding Style: Avoid hard-coding and magic numbers!

```java
x < 10

x < a.length

if (myString.compareTo("hello world") == 0);

private static final String REF = "hello world";

... if (myString.compareTo(REF) == 0);

private static final double PI = 3.14159265;

double radius, area;

area = PI * (radius * raduis);

double radius, area;

area = 3.14159265 * (radius * raduis);
```
Solution Ex2.Q2

- Coding Style – Loops

**for** is used when iterating

```java
for(int i; i < MAX_I; ++i)
{
    nextIterationStep();
}
```

**While** is used for specific cases

```java
int timeout = 0;
while( !userInteraction() )
{
    Thread.yield();
    timeout++;
}
```
Solution Ex2.Q2

- Coding Style – Efficiency

Object instansiation is expensive!

```java
void initialize() {
    for (int i=0; i<a.length; i++) {
        Random r = new Random();
        a[i] = r.nextInt(1000);
    }
}
```

```java
void initialize() {
    Random r = new Random();
    for (int i=0; i<a.length; i++) {
        a[i] = r.nextInt(1000);
    }
}
```
Solution Ex2.Q3a

- Root at index 0
- Successor of \( i \) is at \( 2i + 1 \) and \( 2i + 2 \)

\[
2^{\text{height}-1} = 2^{\text{depth}} \leq \text{array.length} < 2^{\text{depth}+1} = 2^{\text{height}}
\]

```c
int leftChild( node ){
    return 2 * node + 1;
}

int rightChild( node ){
    return 2 * node + 2;
}

int father( node ){
    return (node - 1) / 2;
}
```

\( \text{father}(0) = -1 / 2 = 0 \)
Solution Ex2.Q3b&c

- **checkTree**(char [] array)
  
  Check if the given array represents a valid binary tree

- **Test:**
  - Each node has a father
  - The root is its own father
  - Solution is trivial

- What happens with empty nodes?
  - No need for a father

- **toString**()
Solution Ex2.Q3

- `checkTree(char [] array)`

```java
private static void checkTree(char[] array) {
    if (array.length == 0) throw new IllegalArgumentException("At least one, probably empty node is required.");
    for (int i=0; i<array.length; i++) {
        if (array[i] != ' ') {
            int f = father(i);
            if (array[f] == ' ') {
                throw new IllegalArgumentException("node number " + i + " has no father");
            }
        }
    }
}
```
Solution Ex2.Q3

- `toString()`

```java
private String toString(int node, String indentation) {
    String s = indentation;
    s = s + tree[node] + '\n';

    int c1 = leftChild(node);
    if (isNode(c1)) {
        s = s + toString(c1, indentation + " ");
    }

    int c2 = rightChild(node);
    if (isNode(c2)) {
        s = s + toString(c2, indentation + " ");
    }

    return s;
}
```
Outlook

- Exercise 2 solution discussion

- Exercise 3 overview (Strings, Syntax Diagram, Syntax checker)
Exercise 3

1. Objects and references (e.g. Strings)
   - Sting and StingBuffer
   - Program analysis: Objects and references at runtime?

2. Syntax diagrams
   - Given the diagrams, which expressions can be produced?

3. Syntax checker for trees
   - Complete the syntax diagrams from the class
   - Implement the syntax checker
Exercise 3 – Q 1

- String – Immutable
  - Compiler optimizations. Efficient for reads.
  - Cannot be modified after creation or initialization. Only “modification” by copying.

- StringBuffer – Mutable
  - Easily modifiable (without copy)
  - Some operations are more expensive (e.g. search)
Exercise 3 – Q 1

- Difference between `String` and `StringBuffer`

```java
String myString = "hello";
myString = myString + " world";
```

**JAVA String concatenation**

```java
StringBuffer myStringBuffer = "hello";
myStringBuffer.append(" world");
```

**StringBuffer Method**

Memory

"hello"  "world"

"hello world"

"world"
Exercise 3 – Q 1

```java
String myString = "hello";
myString = myString + " world";
myString = myString + " how";
myString = myString + " are";
myString = myString + " you";
myString = myString + " today";
```

Exercise 3 – Q 1

Insider’s tip: Use Eclipse’s debugger

Warning: The debugger doesn’t prompt temporary Strings, because they are created at one stage and directly discarded

```java
String myString = "hello";
myString = myString+" world";
...

myString = "hello world"
...
```

When the line turns yellow, a modification has taken place.
Exercise 3

- Analyze the program and state which objects and references exist at the marked spot at runtime?

Example:

```java
String str = "foo";
StringBuffer buf = new StringBuffer("foobuffer");
// Markierung 1
```

- Objects @ Markierung 1
  1. (1, String, "foo")
  2. (2, String, "foobuffer")
  3. (3, StringBuffer, "foobuffer")
- References @ Markierung 1
  1. (str, 1)
  2. (buf, 3)
Exercise 3

```java
void addString(String s1, String s2){
    s1 = s1 + s2;
    // MARKIERUNG 3
}

public static void main(String[] args) {

    StringObject ref = new StringObject();
    String mystring = "Give ";
    StringBuffer mybuffer = new StringBuffer("Give ");
    // MARKIERUNG 1

    mystring = mystring + "me six ";
    mybuffer.append("me six ");
    // MARKIERUNG 2

    ref.addString(mystring, "lines");
    mybuffer.append("lines");
    // MARKIERUNG 4
}
```
Solution – Markierung 1

```java
public static void main(String[] args) {
    StringObject ref = new StringObject();
    String mystring = "Give ";
    StringBuffer mybuffer = new StringBuffer("Give ");
}
```

Each string literal is a reference to an instance of class String. (…) Literal strings within the same class in the same package represent references to the same String object. [Java Language Reference Abschnitt 3.10.5]
Solution – Markierungen 2

mystring = mystring + "me six ";
mybuffer.append("me six ");

Common Mistake!
Solution – Markierung 3

```java
void addString(String s1, String s2){
    s1 = s1 + s2;
}
```

```java
ref.addString(mystring, "lines");
```
Solution – Markierung 4

```java
mybuffer.append("lines");
```

S1 und S2 lokale Variablen nicht mehr existieren!
Exercise Q 1 - String vs. StringBuffer

Caesar chiffre

a) Implement (small) method that performs caesar chiffre on StringBuffer

b) Go through the main program
   I. Explain the main program
   II. Run the program and note the time it takes using String and using your implementation of StringBuffer → It should be much faster using a StringBuffer: why?

c) Why using Strings at all? Can you imagine a scenario where using String are more efficient?
Exercise 3 – Syntax checker for trees Q 3

Methodology:

- Methods for «Tree», «Successor» and «Nodes»
- Recursion!
- Offset Parameter

```java
public static int f(String str, int offset){...}
```

Give the new offset to the above mentioned method. Think about how large the offset should be in the end and what happens if it doesn’t get that length. Hint: Solution of 3a) should be integrated

- Bounds-checking:
  `StringIndexOutOfBoundsException` – you are trying to access character `n` in the string, but the array is shorter than `n`. 
Exercise 3 – Syntax checker for trees Q 3

parseEmptyOrSubTree()

parseSubTree()

parseChildren()

parseNode()
Exercise 2 – Simple Syntax diagram (DEMO!)

Let's have a look at the code of this diagram!

Node

Single → A → B → A

SIMPLESEQUENCE

Baum → Node → ,

Eclipse DEMO
Have Fun!