Overview

- Debriefing Exercise 3
- Briefing Exercise 4
U3.A1 Program verification

a) What is the loop invariant for this code

One solution:

```java
static int f(int i, int j) {
    int u = i;
    int z = 0;
    while (u > 0) {
        z = z + j;
        u = u - 1;
    }
    return z;
}
```

Term will always be 0
b) Proove the correctnes of the code with the loop invariant.

```
static int f(int i, int j) {
    int u = i;
    int z = 0;
    while (u > 0) {
        z = z + j;
        u = u - 1;
    }
    return z;
}
```

Proof:

- **Initial State:**
  - $z = 0$ → $u \times j = i \times j$

- **Loop Invariant:**
  - $z = n \times j$, $u = i - n$, $inv = 0$

- **Loop Body:**
  - $z = (n+1) \times j$, $u = i - (n+1)$, $inv = 0$

- **Termination:**
  - $u = 0$ → $z = i \times j$
U3.A1 Program verification

c) what if line 5 and 6 are changed to z=z; and u=u?

The loop invariant is still valid

\[ z + u \times j - i \times j \]

```java
1 static int f(int i, int j) {
2     int u = i;
3     int z = 0;
4     while (u > 0) {
5         z = z
6         u = u
7     }
8     return z;
9 }
```
c) what if line 5 and 6 are changed to \( z = z \); and \( u = u \)?

Using Hoare logic:

If condition and invariant
Are true before
The loop body

\[
\{ C \wedge I \} \quad \text{body} \quad \{ I \} \\
\{ I \} \quad \text{while} \ (C) \quad \text{body} \quad \{ \neg C \wedge I \}
\]

then the negated condition
\( (u=0) \) holds after the loop body

But does it mean that the implementation is correct?
c) what if line 5 and 6 are changed to z=z; and u=u?

No, the proofs so far only show **partial correctness**
For **total correctness**, we also need to prove termination

Does the program terminate?
NO

Only **partially correct**

```
static int f(int i, int j) {
    int u = i;
    int z = 0;
    while (u > 0) {
        z = z
        u = u
    }
    return z;
}
```

1. Objects and references (e.g. Strings)
   - String vs. StringBuffer
   - Caesar cypher
   - Encrypt and decrypt, understand how the program works
U3.A2 Decrypt

- Inverse of encrypt
- Take each character and subtract 3 from its ASCII code

How do you access each character? \texttt{s.chartAt(index)}

```java
/**
 * Decrypts input text based on the CaesarChiffre (i.e., removing 3 from the
 * ASCII code of each character). The decryption employs StringBuffers
 * (instead of Strings).
 *
 * @param s ciphertext to be decrypted
 */
public static String decrypt(String s) {
    StringBuffer ret = new StringBuffer();
    for (int i = 0; i != s.length(); ++i) {
        ret.append((char) (s.charAt(i) - 3));
    }
    return ret.toString();
}
```
<table>
<thead>
<tr>
<th>ASCII Character Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>\n</td>
</tr>
<tr>
<td>\n</td>
</tr>
<tr>
<td>\n</td>
</tr>
<tr>
<td>\n</td>
</tr>
<tr>
<td>\n</td>
</tr>
<tr>
<td>0 0 000 NUL (null)</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>64</td>
</tr>
<tr>
<td>96</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>33</td>
</tr>
<tr>
<td>65</td>
</tr>
<tr>
<td>97</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>34</td>
</tr>
<tr>
<td>66</td>
</tr>
<tr>
<td>98</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>35</td>
</tr>
<tr>
<td>67</td>
</tr>
<tr>
<td>99</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>36</td>
</tr>
<tr>
<td>68</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>37</td>
</tr>
<tr>
<td>69</td>
</tr>
<tr>
<td>101</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>38</td>
</tr>
<tr>
<td>70</td>
</tr>
<tr>
<td>102</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>39</td>
</tr>
<tr>
<td>71</td>
</tr>
<tr>
<td>103</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>72</td>
</tr>
<tr>
<td>104</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>41</td>
</tr>
<tr>
<td>73</td>
</tr>
<tr>
<td>105</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>42</td>
</tr>
<tr>
<td>74</td>
</tr>
<tr>
<td>106</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>43</td>
</tr>
<tr>
<td>75</td>
</tr>
<tr>
<td>107</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>44</td>
</tr>
<tr>
<td>76</td>
</tr>
<tr>
<td>108</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>45</td>
</tr>
<tr>
<td>77</td>
</tr>
<tr>
<td>109</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>46</td>
</tr>
<tr>
<td>78</td>
</tr>
<tr>
<td>110</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>47</td>
</tr>
<tr>
<td>79</td>
</tr>
<tr>
<td>111</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>48</td>
</tr>
<tr>
<td>80</td>
</tr>
<tr>
<td>112</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>49</td>
</tr>
<tr>
<td>81</td>
</tr>
<tr>
<td>113</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>82</td>
</tr>
<tr>
<td>114</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>51</td>
</tr>
<tr>
<td>83</td>
</tr>
<tr>
<td>115</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>52</td>
</tr>
<tr>
<td>84</td>
</tr>
<tr>
<td>116</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>53</td>
</tr>
<tr>
<td>85</td>
</tr>
<tr>
<td>117</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>54</td>
</tr>
<tr>
<td>86</td>
</tr>
<tr>
<td>118</td>
</tr>
<tr>
<td>23</td>
</tr>
<tr>
<td>55</td>
</tr>
<tr>
<td>87</td>
</tr>
<tr>
<td>119</td>
</tr>
<tr>
<td>24</td>
</tr>
<tr>
<td>56</td>
</tr>
<tr>
<td>88</td>
</tr>
<tr>
<td>120</td>
</tr>
<tr>
<td>25</td>
</tr>
<tr>
<td>57</td>
</tr>
<tr>
<td>89</td>
</tr>
<tr>
<td>121</td>
</tr>
<tr>
<td>26</td>
</tr>
<tr>
<td>58</td>
</tr>
<tr>
<td>90</td>
</tr>
<tr>
<td>122</td>
</tr>
<tr>
<td>27</td>
</tr>
<tr>
<td>59</td>
</tr>
<tr>
<td>91</td>
</tr>
<tr>
<td>123</td>
</tr>
<tr>
<td>28</td>
</tr>
<tr>
<td>60</td>
</tr>
<tr>
<td>92</td>
</tr>
<tr>
<td>124</td>
</tr>
<tr>
<td>29</td>
</tr>
<tr>
<td>61</td>
</tr>
<tr>
<td>93</td>
</tr>
<tr>
<td>125</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>62</td>
</tr>
<tr>
<td>94</td>
</tr>
<tr>
<td>126</td>
</tr>
<tr>
<td>31</td>
</tr>
<tr>
<td>63</td>
</tr>
<tr>
<td>95</td>
</tr>
<tr>
<td>127</td>
</tr>
</tbody>
</table>
U3.A2 Main

- What is different?
  - Encrypt is much slower than decrypt. Why?
  - StringBuffer is more efficient for appending
  - Strings are immutable
    - Any modification leads to a new copy of the object.
U3.A2 Strings

- Why use Strings in the first place?
  - Strings are constants and allow for optimizations
  - Strings are immutable, which could be a requirement in some cases
  - The biggest benefit for StringBuffer is when we append/modify the string at runtime
2a) Clause

<table>
<thead>
<tr>
<th>Expression</th>
<th>Possible</th>
<th>Impossible</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(\sim X_1)$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$\sim (X_1 \lor \sim X_2)$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$(X_2) \lor (\sim X_1 \lor X_2)$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
</tbody>
</table>
U3.A3 Syntax diagrams

2b) Expr

<table>
<thead>
<tr>
<th>Possible</th>
<th>Impossible</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(X_1 \text{ OR } X_2) \text{ AND } (\sim X_1)$</td>
<td>✗</td>
</tr>
<tr>
<td>$(X_1) \text{ AND } (\sim X_1 \text{ OR } \sim X_2) \text{ AND } (X_2)$</td>
<td>✗</td>
</tr>
</tbody>
</table>
How do we change it to allow empty trees and successors?
U3.A4 Syntax checker

```java
private static int parseTree(String kd, int offset) throws ParseException {
    if (offset >= kd.length()) {
        throw new ParseException("Unexpected end of string", offset);
    }
    if (kd.charAt(offset) == '-') {
        return offset + 1;
    }
    else {
        offset = parseNode(kd, offset);
        if (((offset < kd.length()) && (kd.charAt(offset) == '(')) {
            offset += 1;
            offset = parseSubtree(kd, offset);
            if (((offset < kd.length()) && (kd.charAt(offset) == ')'))) {
                offset += 1;
            }
            else {
                throw new ParseException("expected '\)'", offset);
            }
        }
        return offset;
    }
}
```
private static int parseSubtree(String kd, int offset) throws ParseException {
    if (offset >= kd.length()) {
        throw new ParseException("unexpected end of string after '(', offset);
    }
    offset = parseTree(kd, offset);
    while ((offset < kd.length()) && (kd.charAt(offset) == ',')) {
        offset += 1;
        offset = parseTree(kd, offset);
    }
    return offset;
}
U3.A4 Syntax checker

private static int parseNode(String kd, int offset) throws ParseException {
    if (offset >= kd.length()) {
        throw new ParseException("Expected a node", offset);
    }

    if (Character.isUpperCase(kd.charAt(offset))) {
        return offset + 1;
    } else {
        throw new ParseException(String.format("'%c' is not a valid node name", kd.charAt(offset)), offset);
    }
}
public static void parse(String kd) throws ParseException {
    int offset = parseTree(kd, 0);
    if (offset != kd.length()) {
        throw new ParseException("Garbage at the end of the tree", offset);
    }
}
Overview

- Debriefing Exercise 3
- Briefing Exercise 4
Stack

- Abstract data type
- Collection of elements
- LIFO principle
  - Last in, first out
- Two main operations: Push and Pop
U4.A1

- Constructor
  - Initializes internal Array
  - Capacity is an argument to the constructor

- `toString()` with StringBuffer
  - Expected Output: "[e0, e1, e2, ...]"
  - Concatenation
    - String: `str += "bar";`
    - StringBuffer: `buf.append("bar");`

- `grow()`
  - Capacity doubled, copy old values
U4.A1

- push(), pop(), peek(), empty()
  - Standard stack functions
  - Arguments are of type `int`
  - If necessary, call `grow()`

- size()
  - Number of elements currently on the stack

- capacity()
  - Total number of elements which fit on the current stack until the next `grow`
Ackermann function

- Recursive Definition

\[ A(0, m) = m + 1 \]
\[ A(n + 1, 0) = A(n, 1) \]
\[ A(n + 1, m + 1) = A(n, A(n + 1, m)) \]

- Grows extremely fast
  - \( A(3,3) = 61 \)
  - \( A(4, 2) \) has already 19729 decimal places!!
U4.A2

- A(1,1) given as example in the homework

- Calculate A(2,1) by hand
  - A(2,1) = A(1+1, 0+1) = A(1, A(2,0)) …

- Write down all the steps!
\[ A(4, 3) = A(3, A(3, 2)) \]
\[ = A(3, A(3, A(4, 1))) \]
\[ = A(3, A(3, A(3, A(4, 0)))) \]
\[ = A(3, A(3, A(3, A(4, 0)))) \]
\[ = A(3, A(3, A(3, A(3, 1)))) \]
\[ = A(3, A(3, A(3, A(3, A(3, 0)))))) \]
\[ = A(3, A(3, A(3, A(2, A(2, 1)))))) \]
\[ = A(3, A(3, A(3, A(2, A(1, A(0, 0)))))) \]
\[ = A(3, A(3, A(3, A(2, A(1, A(0, 1)))))) \]
\[ = A(3, A(3, A(3, A(2, A(1, A(0, 2)))))) \]
\[ = A(3, A(3, A(3, A(2, A(1, 3)))))) \]
\[ = A(3, A(3, A(3, A(2, A(0, A(1, 1)))))) \]
\[ = A(3, A(3, A(3, A(2, A(0, A(0, A(1, 1)))))) \]
\[ = A(3, A(3, A(3, A(2, A(0, A(0, A(0, A(1, 1)))))) \]
\[ = A(3, A(3, A(3, A(2, A(0, A(0, A(0, A(0, 1)))))) \]
\[ = A(3, A(3, A(3, A(2, A(0, A(0, A(0, A(0, 2)))))) \]
\[ = A(3, A(3, A(3, A(2, A(0, A(0, A(0, 3)))))) \]
\[ = A(3, A(3, A(3, A(2, A(0, 3)))) \]
\[ = A(3, A(3, A(3, A(2, 5)))) \]
\[ = \ldots \]
\[ = A(3, A(3, A(3, 13))) \]
\[ = \ldots \]
\[ = A(3, A(3, 65533)) \]
\[ = \ldots \]
\[ = A(3, 2^{65536} - 3) \]
\[ = \ldots \]
\[ = 2^{65536} - 3. \]
U4.A2

- Specify the algorithm using the usual two stack operations:
  - push(x)
  - x = pop()

- Pseudocode:
  - No language-specific syntax
  - Pseudocode is self-explanatory
  - Based on comments

- The function has the property that one can not say in advance how deep the recursion is
  - Use while instead of for-loop!
U4.A2 Iterative approach

- Ackermann’s formula always requires (exactly) two values:
  - The currently required values should be at the top of the stack…
  - What does it mean when there is one item left in the stack?

```java
Stack stack = new Stack();
stack.push(4);
stack.push(7);

while(stack.size() != 1)
{
    ...
}
```
U4.A2 Implementation

stack.push(m)
stack.push(n)

if n == 0 → result = m + 1
else if m == 0 → push(n-1), push(1)
else push(n-1), push(n), push(m-1)
U2.A2

A(1,1) -> A(1,0) -> A(0, 1) <- 2
A(0, 2) <- 3

Start

A(1,1) Push

Iteration

size >= 2?

m = 0
n = 0 Pop

n == 0?
m + 1 Push

m == 0?
n - 1 Push

else

n - 1 Push

m - 1 Push

End

A(1,1) = 3 Pop

By Leyna Sadamori
U4.A2 Hints

- **Stack**
  - The stack from U4.A1
  - The interface should NOT be modified

- **“Snapshots”**
  - With `toString()` method of the stack

- **I cannot do U4.A1**
  - Use `java.util.Stack<Integer>`
    - you just need `push()`, `pop()`, `size` and `toString()`
  - If necessary: send me an Email
U4.A3 Bytecode

- Before you disassemble the code, it must be compiled

- For Linux and Mac users:
  - Use the `>>` operator in the terminal to send the output to a file
  - E.g.: `javap -c RecursiveAckermann >> output.txt`
U4.A3 Bytecode

- For Windows:

```
D:\Projects\DisassemblerDemo>
javac JavapTip.java //compiler
java JavapTip //run
javap -c -private JavaTip //disassembler
```

Common mistake: "javap is not recognized as an internal or external command, operable program or batch file"

Reason: java binaries are not defined in System variable PATH

Solution: RClick on Computer → Properties → Advanced System Settings → Environment Variables → PATH → add (where you installed the Java JDK) save and restart Windows

;C:\Program Files\Java\jdkX.Y.Z\bin
U4.A3 Bytecode example

```java
public int greaterThen(int intOne, int intTwo) {
    if (intOne > intTwo) {
        return 0;
    } else {
        return 1;
    }
}
```

```
0: iload_1
1: iload_2
2: if_icmple    7
5: iconst_0
6: ireturn
7: iconst_1
8: ireturn
```
U4.A3 Bytecode

- Instructions:
  - iload_n : load int from local variable
  -aload_n : load reference from local variable
  - if_icmp<cond> : Branch if int comparison succeeds
    - E.g. if_icmple: le = less or equal
  - if<cond>: Branch if comparison to zero succeeds
    - Ifeq: equal to 0
    - Ifne: not equal to 0
  - Invokevirtual: invoke instance method

- Documentation:
Have Fun!