

zu Aufgabe 1:

1. Fehler: falscher Aufruf, weil z.B. Parameter falsch: im Aufruf `https://.../<id>/...` wird falsche id übergeben -> Error oder Daten werden bei falschem Automaten gespeichert
2. Fehler: Fehler im XML: Maschine vergibt anderen Namen in XML als API erwartet -> unvollständige Informationen
3. Fehler: falsche Serverversion aufgerufen, die vielleicht gar nicht mehr existiert

Aufgabenblatt 3

Aufgabe 1

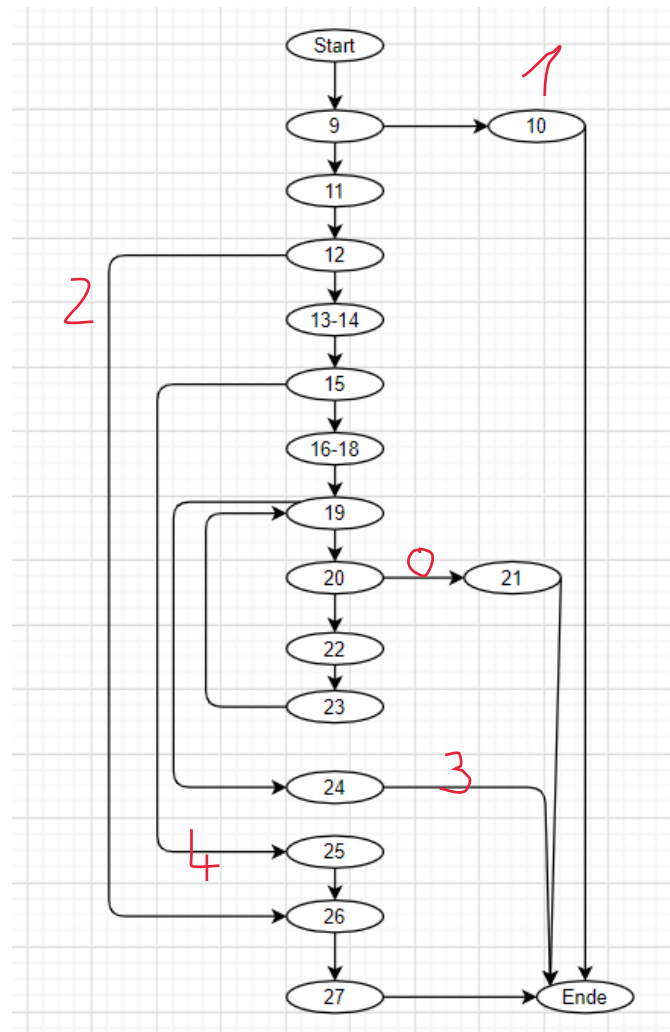
Eine Vertriebsfirma betreibt Verkaufsmaschinen, die an Bahnhöfen stehen und Süßigkeiten verkaufen. Die Automaten melden regelmäßig Verkaufs- und Zustandsinformationen im XML Format über ein REST Interface an einen Server in einem Rechenzentrum. Welche Integrationsfehler können hierbei auftreten? Nennen Sie drei mögliche Fehlertypen und versuchen Sie, für jeden Fehlertyp ein möglichst realistisches und konkretes Szenario zu entwerfen und zu beschreiben.

Aufgabe 2

Gegeben ist die Klasse „*MyString*“ mit der Methode „*equals(...)*“:

```
1  import java.util.Arrays;
2  public class MyString {
3      private final char value[];
4      public MyString(String aString) {
5          this.value = aString.toCharArray();
6      }
7
8      public boolean equals(Object anObject) {
9          if (this == anObject) {
10             return true;
11         }
12         if (anObject instanceof MyString) {
13             MyString anotherString = (MyString) anObject;
14             int n = value.length;
15             if (n == anotherString.value.length) {
16                 char v1[] = value;
17                 char v2[] = anotherString.value;
18                 int i = 0;
19                 while (n-- != 0) {
20                     if (v1[i] != v2[i])
21                         return false;
22                     i++;
23                 }
24                 return true;
25             }
26         }
27         return false;
28     }
29 }
```

a.) Bitte zeichnen Sie den Programmgraph für die Methode „*equals()*“.



b.) Zeigen Sie an dem Graphen für die Methode „equals ()“ welche Testfälle nötig sind für 100% Zweigabdeckung.

	MyString	anObject	Ergebnis
0	„a“	„b“	false
1. Fall, selben Objekte	"a"	"a"	true
2	"a"	1	false
3	"aa"	"aa"	true
4	"a"	"aa"	false

c.) Schreiben Sie ein JUnit Test für die Funktion mit 100 % Zweigabdeckung. Nutzen Sie dazu das „vier Phasen Test Muster“, markieren Sie die einzelnen Phasen mit entsprechenden Java Kommentaren.

siehe Github: <https://github.com/jo851rit/SWQS/blob/master/Aufgabenblatt4/Aufgabe2c/src/MyStringTest.java>

Aufgabe 3

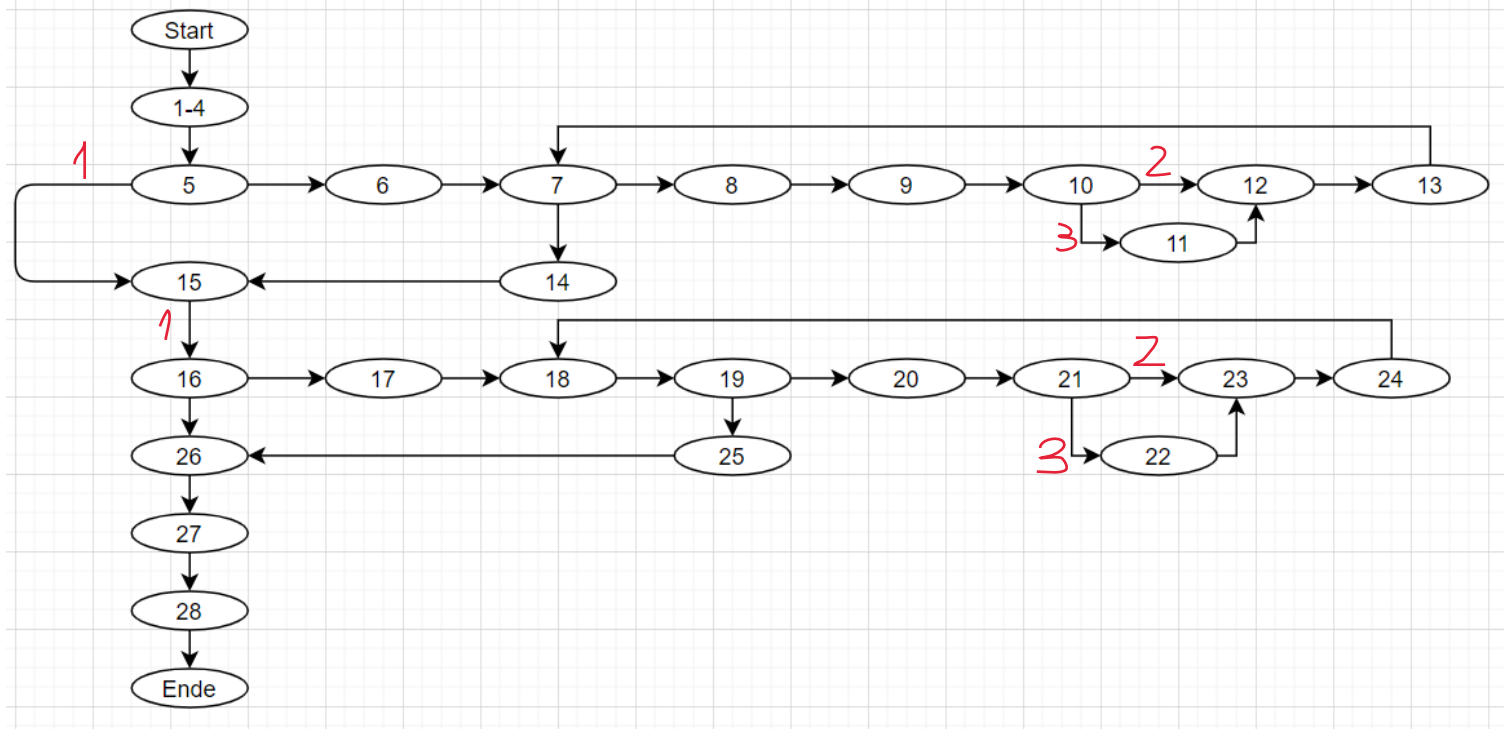
Gegeben ist die Klasse StackMapEntry mit der Methode toString() :

```
public final class StackMapEntry
{
    private int frame_type;
    private int byte_code_offset;
    private String[] types_of_locals;
    private String[] types_of_stack_items;
    public StackMapEntry(int frame_type,
        int byte_code_offset,
        String[] types_of_locals,
        String[] types_of_stack_items) {
        this.frame_type = frame_type;
        this.byte_code_offset = byte_code_offset;
        this.types_of_locals = types_of_locals;
        this.types_of_stack_items = types_of_stack_items;
    }

    public final String toString() {
1      final StringBuilder buf = new StringBuilder(64);
2      buf.append("(");
3      buf.append("ft=").append(frame_type);
4      buf.append(", off=").append(byte_code_offset);
5      if (types_of_locals.length > 0) {
6          buf.append(", locals={");
7          for (int i = 0; i < types_of_locals.length; i++)
8              {
9                  buf.append(types_of_locals[i]);
10                 if (i < types_of_locals.length - 1) {
11                     buf.append(", ");
12                 }
13             }
14         buf.append("}");
15     }
16     if (types_of_stack_items.length > 0) {
17         buf.append(", items={");
18         for (int i = 0; i < types_of_stack_items.length;
19             i++) {
20             buf.append(types_of_stack_items[i]);
21             if (i < types_of_stack_items.length - 1) {
22                 buf.append(", ");
23             }
24         }
25         buf.append("}");
26     }
27     buf.append(")");
28     return buf.toString();
    }
}
```

a) Bitte zeichnen Sie den Programmgraph für die Methode „toString()“ (7 Punkte)

Anmerkung: Im Programmgraph darf Java Pseudocode für die Anweisungen verwendet werden. Es müssen nicht die vollständigen Anweisungen als Java Code in den Graph übernommen werden, die Zeilennummern sind ausreichend. Wichtig sind die Anzahl der Anweisungen im Graph und die Wege zwischen den Anweisungen. Es sollte nachvollziehbar sein welche Anweisung im Graph welche Zeilen der Klasse modelliert.



b. Zeigen Sie an dem Graphen für die Methode „toString()“ welche Testfälle nötig sind für 100% Zweigabdeckung.

	frame_type	code_offset	types_of_locals	types_of_stack_items	Ergebnis
	0	1	["a"]	["c", "d"]	„(ft=0, ...“
1	0	1	[]	[]	“(ft=0, off=1)”
2	0	1	["a"]	["c"]	“(ft=0, off=1, locals={a}, items={c})”
3	0	1	["a", "b"]	["c", "d"]	“(ft=0, off=1, locals={a,b}, items={c,d})”