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(...)*“:

```
import java.util.Arrays;
public class MyString {
	private final char value[];
	public MyString(String aString) {
		this.value = aString.toCharArray();
	}

	public boolean equals(Object anObject) {
		if (this == anObject) {
			return true;
		}
		if (anObject instanceof MyString) {
			MyString anotherString = (MyString) anObject;
			int n = value.length;
		if (n == anotherString.value.length) {
			char v1[] = value;
			char v2[] = anotherString.value;
			int i = 0;
			while (n-- != 0) {
				if (v1[i] != v2[i])
					return false;
				i++;
			}
			return true;
		}
	}
	return false;
	}
}
```

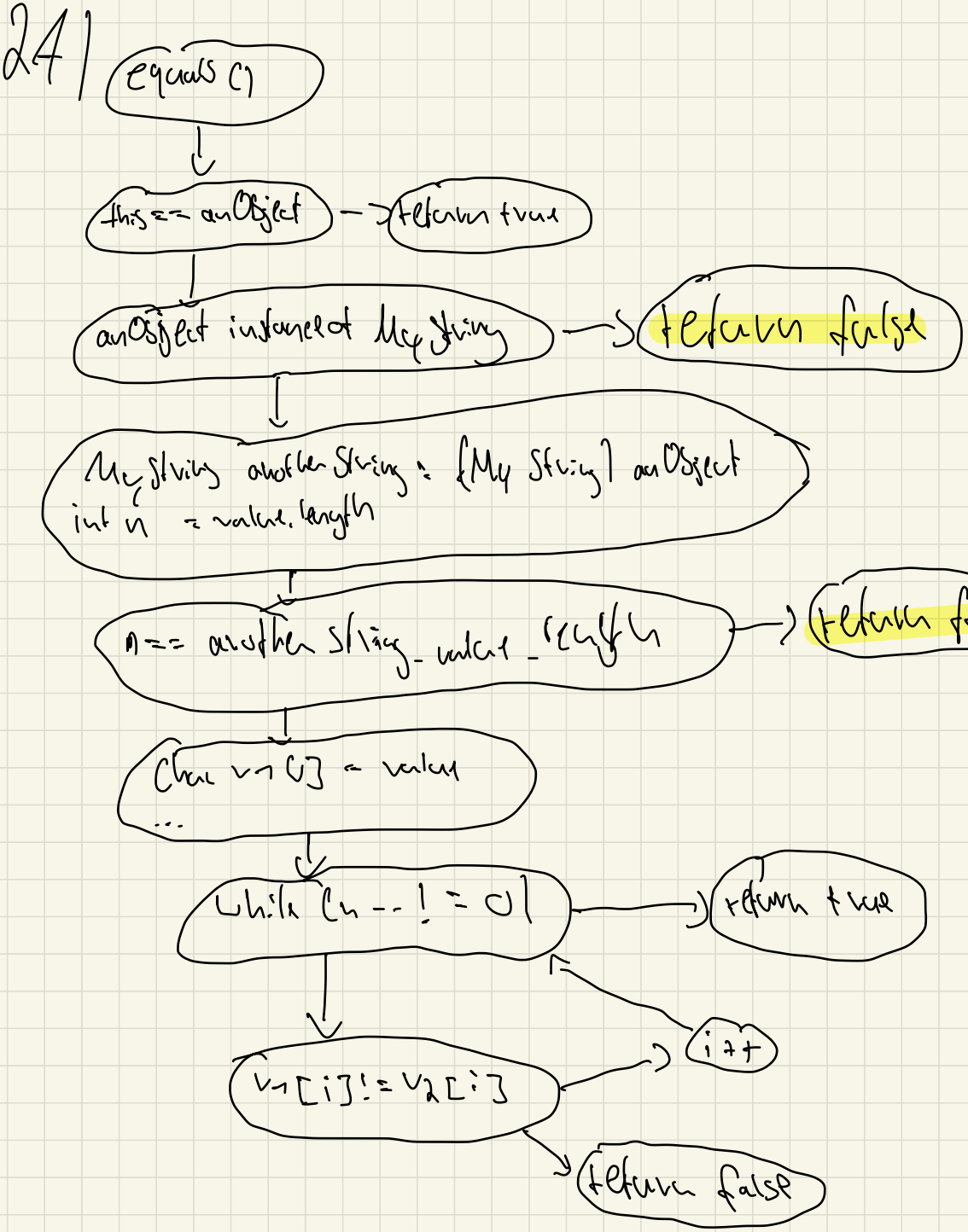
a.) Bitte zeichnen Sie den Programmgraph für die Methode „*equals ()*“.

Aufgabe 1

1) Der REST-Interface wird mit fehlerhafter URL aufgerufen, wodurch kein oder der falsche Service aufgerufen wird.

2) Das XML kann das falsche Format haben, wenn die Werte in der falschen Reihenfolge eingegeben werden. Das Format wird durch das XSD vorgegeben.

3) Falsche Ein- und Ausgabewerte können ebenfalls zu Fehlern führen -> XML muss immer passend zur XSD sein. Das Bedeutet, wenn etwas in der XSD geändert wird, dann muss auch das Schema dementsprechend angepasst werden.


24)
equals ()
this == anObject
return true
anObject instanceof MyString
return false
MyString anotherString = (MyString) anObject
int n = value.length
n == anotherString.value.length
return false
char v1[] = value
...
while (n-- != 0)
return true
i++
v1[i] != v2[i]
return false

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

MyString	anObject	Ergebnis
„a“	„b“	false
„a“	„a“	true
„“	„“	true
„ab“	„bb“	false
„abbb“	„abba“	false
„aba“	„aba“	true

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.

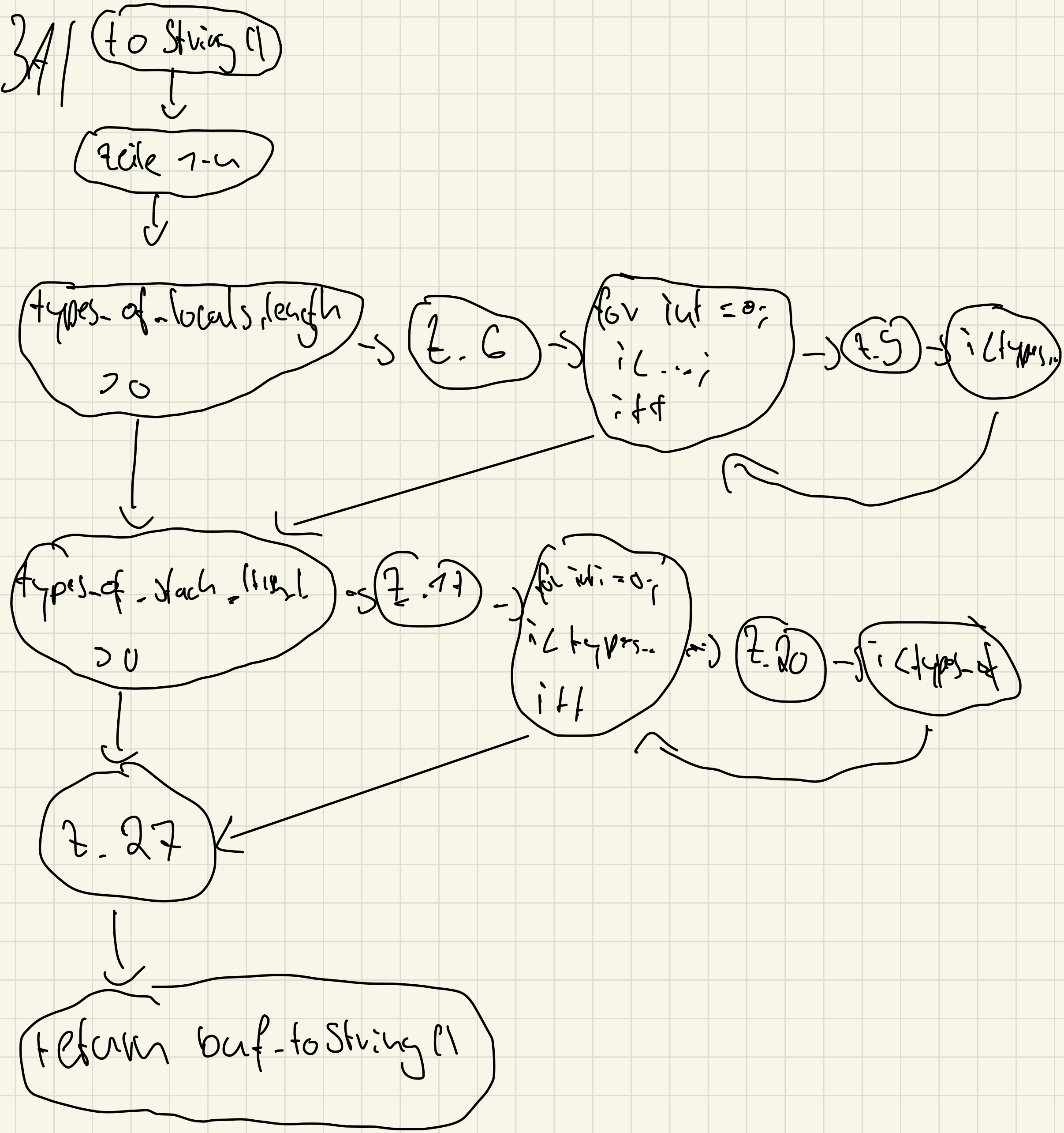
Aufgabe 3

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;
    }

0   public final String toString() {
1       final StringBuilder buf = new StringBuilder(64);
2       buf.append("(");
        buf.append("ft=").append(frame_type);
        buf.append(", off=").append(byte_code_offset);
        if (types_of_locals.length > 0) {
            buf.append(", locals={");
            for (int i = 0; i < types_of_locals.length; i++)
            {
                buf.append(types_of_locals[i]);
                if (i < types_of_locals.length - 1) {
                    buf.append(", ");
                }
            }
            buf.append("}");
        }
        if (types_of_stack_items.length > 0) {
            buf.append(", items={");
            for (int i = 0; i < types_of_stack_items.length;
            i++) {
                buf.append(types_of_stack_items[i]);
                if (i < types_of_stack_items.length - 1) {
                    buf.append(", ");
                }
            }
            buf.append("}");
        }
        buf.append(")");
        return buf.toString();
    }
}
```

3A)


toString()
Zeile 1-4
types_of_locals.length > 0
Z. 6
for int i=0;
i<...;
i++
Z. 5
i<types..
> 0
Z. 17
for int i=0;
i<types..
i++
Z. 20
i<types_of
Z. 27
return buf.toString()

Frame_Type	Code_offset	Types_of_locals	Types_of_stack_items	Ergebnis
0	1	[„a“]	[„c“,„d“]	(ft=0, off=1, locals={a}, items={c, d})
1	2	[„a“,„b“]	[„c“]	(ft=1, off=2, locals={a, b}, items={c})
2	3	[„“]	[„“]	(ft=2, off=3)
3	4	[„a“]	[„“]	(ft=3, off=4, locals={a})
4	5	[„“]	[„c“]	
5	6	[„“]	[„c“,„d“]	
6	7	[„a“,„b“]	[„“]	
7	8	[„c“,„d“]	[„a“,„b“]	
8	9	[„a“,„c“,„e“]	[„b“,„d“,„f“]	

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, …