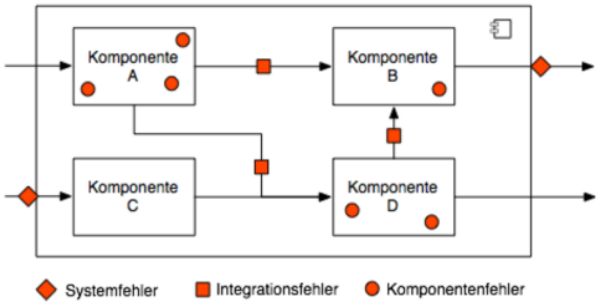
Aufgabenblatt 4

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.



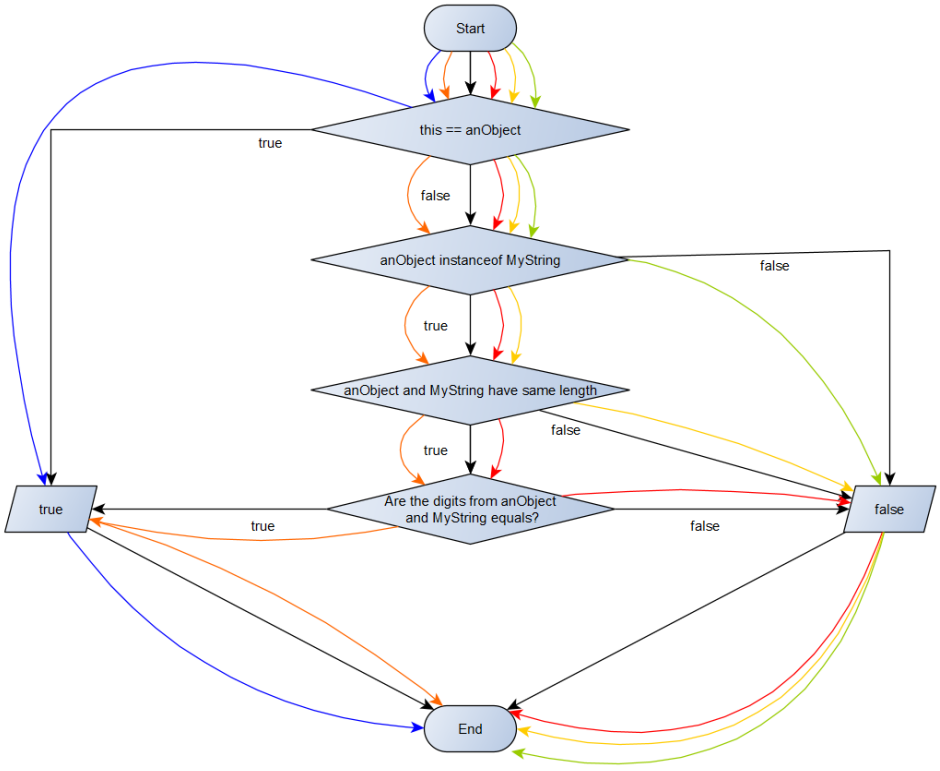
Möglicher Fehlertyp	**Szenario**
Systemfehler	♦ Dem System werden falsche Ein- oder Ausgabewerte übergeben
Integrationsfehler	■ Schnittstellen wurde falsch konfiguriert: ☐ Falsche Interpretation der Ausgabewerte: Ausgabe der Verkaufs- und Zustandsinformationen an den Server des Rechenzentrums erfolgt über ein anderes Format als XML ☐ Ausgabe erfolgt über eine andere Schnittstelle als vorgesehen (also nicht über REST Interface) ☐ Falsche Interpretation der Eingabewerte (z.B. Vertauschen von Verkaufs- und Zustandsinformation) ☐ Unerlaubtes Ändern von Eingabeparameter (z.B. Überschreiben von Default-Wert) ■ Schnittstellenaufruf funktionieren nicht wie vorgesehen: ☐ Timeouts (z.B. bei Stromausfall) werden nicht behandelt ☐ Unregelmäßige statt regelmäßige Antwortzeit (z.B. letzter Zustand wird nicht abgespeichert, sondern immer Daten immer neu aufrufen und berechnet; Je mehr Dateneinträge, desto länger dauern die Berechnungen) ☐ Falsche Reihenfolge der Serviceaufrufe (z.B. Berechnung ausführen vor Einlesen der Daten) ☐ Serviceaufruf am falschen Baustein ☐ Falsche Serviceversion aufgerufen ☐ Deadlocks (z.B. Komponenten brauchen sich gegenseitig um Informationen zu vervollständigen)
Komponentenfehler	• Falsche Interpretation der Übergabe-Werte innerhalb der Methode einer Komponente • Methode weist gar nicht oder nur teilweise die geforderte Funktionalität gemäß Komponentenspezifikation auf (z.B. liest falsche Daten aus dem XML-Format aus)

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 ()*“.



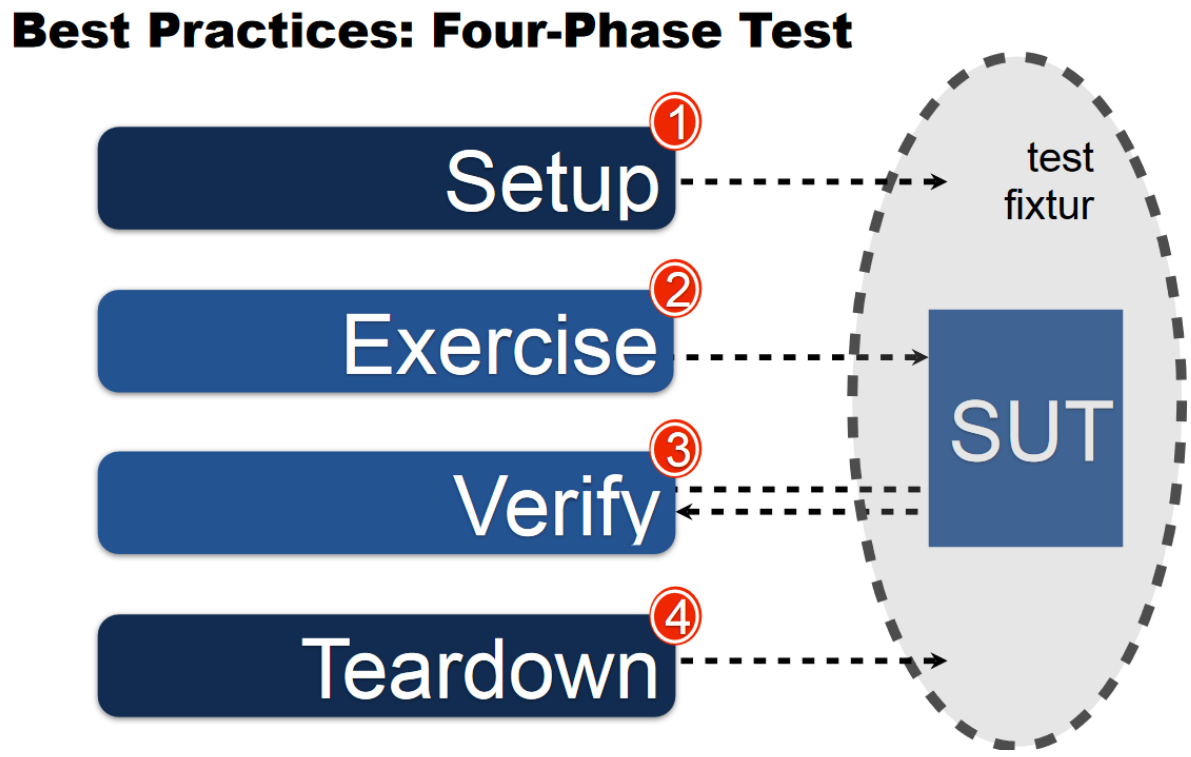
HINWEIS: Bunt eingezeichnete Pfade zeigen auf, mit welchem Testfall welcher Pfad abgedeckt wird. (Je Testfall ist eine Farbe eingezeichnet.)

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

MyString	anObject	Ergebnis	Methode in JUnit-Klasse
"a"	"b"	false	testDifferentDigits()
"a"	"a"	true	testSameDigits()
"a"	"ba"	false	testDifferentLength()
"a"	String str ="a"	false	testNoInstanceOfMyString()
myString = "a"	myString	true	testSameObject()

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.

Link zur Testklasse: https://github.com/ju851han/Software-Qualitaetssicherung/blob/master/ue04/src/MyStringTest.java

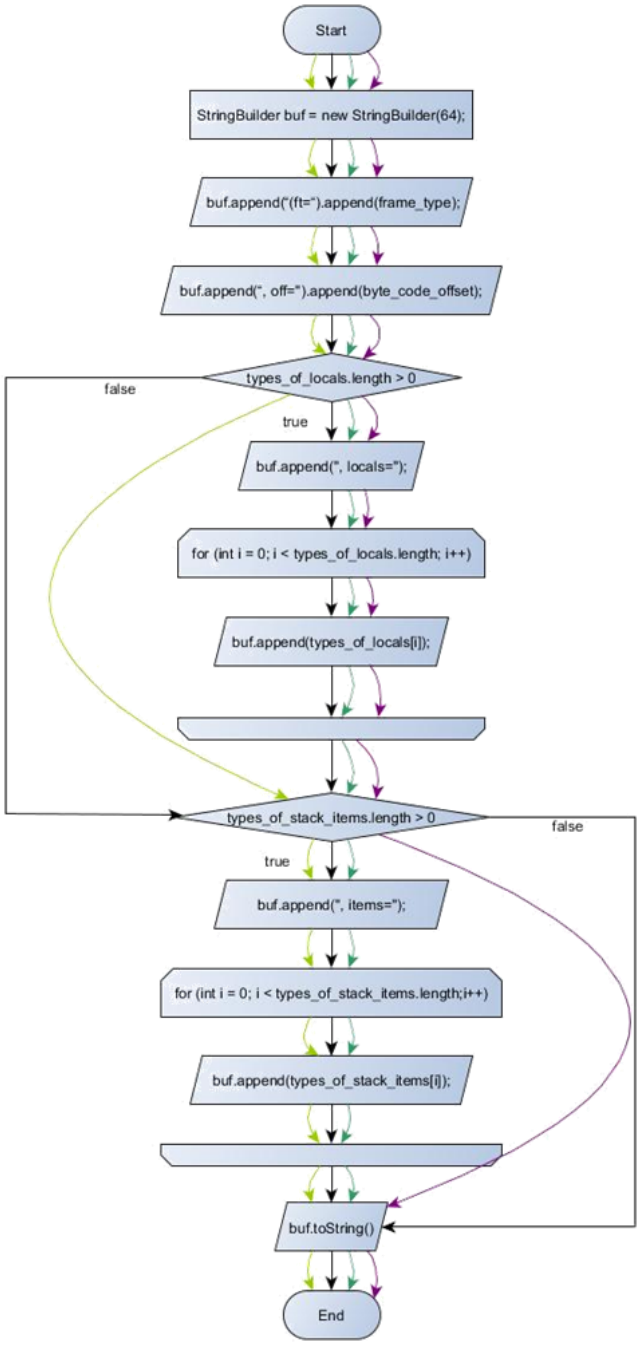


Aufgabe 3

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() {
        final StringBuilder buf = new StringBuilder(64);
        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();
        }
}
```



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.

Siehe Seite 4 rechts neben Code.

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	Methode in JUnit-Klasse
0	1	["a"]	["c","d"]	"(ft=0, off=1, locals={a}, items={c, d})"	testLocalsAndStackItemsLengthBiggerThanZero()
0	1	[]	["c", "d"]	"(ft=0, off=1, items={c, d})"	testLocalsLengthLessOrEqualsThanZero()
0	1	["a"]	[]	"(ft=0, off=1, locals={a})"	testStackItemsLengthLessOrEqualsThanZero()

Link zur Testklasse: https://github.com/ju851han/Software-Qualitaetssicherung/blob/master/ue04/src/StackMapEntryTest.java