Prof. Dr. Stephan Kleuker
Hochschule Osnabrück
Fakultät Ing-Wiss. und Informatik
- Software-Entwicklung -

Software-Qualität

Sommersemester 2025
8. Aufgabenblatt

Aufgabe 0.8 (1 Punkt)
Geben Sie das Lösungswort des Quiz aus der Lernnotiz an.

Aufgabe 17 (4 Punkte)
Gegeben sei folgende Klasse `Gegner`, deren Methode `knoten` getestet werden soll. Es liegen keine weiteren Implementierungen anderer Klassen vor, so dass Sie gezwungen sind, Mocks zu schreiben. Da dies bei Enumerations nicht möglich ist, muss hier eine konkrete Implementierung angegeben werden. Da leider keine Spezifikation der erwarteten Ergebnisse vorliegt, müssen Sie sich auf die Überdeckung konzentrieren.

```
public class Gegner {

  @SuppressWarnings("unused")
  private String typ;
  private boolean hatFK;
  private int feuerweite;

  public Gegner(String typ, boolean hatFK, int feuerweite) {
    this.typ = typ;
    this.hatFK = hatFK;
    this.feuerweite = feuerweite;
  }

  public int knoten(Alarm alert, Eigenschiff e, int abstand) {
    int ergebnis = 42;
    if (alert == Alarm.ROT & abstand < this.feuerweite) {
      ergebnis = e.getMaxKnoten();
    }
    if (alert == Alarm.ORANGE & e.wichtigerAuftrag()) {
      ergebnis = e.sicherheit(e.hatAbwehr(this.hatFK), abstand);
    }
    return ergebnis;
  }
}
```

a) Geben Sie möglichst einfache, aber dennoch sinnvoll nutzbare Mock-Klassen an (ohne Nutzung eines Mock-Frameworks), so dass Sie eine 100%-Branch-Überdeckung für Ihre zu erstellenden Tests erhalten. Nutzen Sie das Projekt qsAufgabeGegnerMock.
b) Schreiben Sie Ihre Lösung aus a) um (oder neu), so dass die Mock-Objekte durch Mockito erstellt werden. Nutzen Sie das Projekt qsAufgabeGegnerMockito.

Aufgabe 18 (4 Punkte)
Ein Überwachungsprogramm für ältere Maschinen fragt zyklisch zwei Werte von nachträglich installierten Sensoren ab. Die Maschinen sind über folgendes Interface angeschlossen.

```
public interface MInterface {
    Integer getWert1() throws NetworkException, HardwareException;
    String getZustand1() throws NetworkException;
}
```

Das Überwachungsprogramm endet, wenn ein schwerer Fehler festgestellt wird, was u. a. an der Variable `alarm` zu erkennen ist. Ein schwerer Fehler tritt auf, wenn in drei aufeinanderfolgenden Zyklen jeweils mindestens einer der abgefragten Werte als Antwort eine Exception oder aufeinander folgende Messwerte mit einem Abstand größer 5 liefert. Ein

schwerer Fehler tritt weiterhin auf, wenn die Frage nach dem Status1 einmal mit „defekt“ beantwortet wird.

<table>
<tr><td rowspan="3">schwerer Fehler</td><td rowspan="2">drei beliebige einfache Fehler direkt hintereinander</td><td>einfacher Fehler: im Zyklus treten eine oder mehrere Exceptions auf</td></tr>
<tr><td>einfacher Fehler: aufeinander folgende Messwerte haben Abstand größer 5 (bei Exceptions wird dabei immer der letzte Messwert vor der Exception betrachtet)</td></tr>
<tr><td colspan="2">oder Status1 ist „defekt“</td></tr>
</table>

Zum nachfolgenden zu testenden Überwachungsprogramm `Ueberwachung` aus dem von der Veranstaltungsseite herunterladbaren Projekt qsAufgabeMockitoMaschinenueberwachung fehlen leider Maschinen, die das Interface MInterface realisieren.

a) Prüfen Sie die Funktionalität der Klasse Überwachung, genauer der Methode `sensorenLesen()`, mit Tests, nutzen Sie dazu passende, mit Mockito zu erstellende, Mocks für Maschinen. Sie sollten zumindest eine 100%-Branch-Überdeckung dieser Methode mit erreichen.
b) In a) sollten Sie mindestens zwei (miteinander „verwandte“) Fehler finden, die zu einem zu frühen und zu einem fehlenden Alarm führen. Korrigieren Sie das Programm in einer neuen Klasse und nutzen Sie Ihre Tests aus a) zur Überprüfung.

zu 0.8)

Prof. Dr. Stephan Kleuker
Hochschule Osnabrück
Fakultät Ing-Wiss. und Informatik
- Software-Entwicklung -

Software-Qualität

Sommersemester 2025
8. Aufgabenblatt

Quiz 1: Mock-Nutzung

Gegeben sei folgende Methode.

```
public class Rechnernutzung {
 public int nutzen(Rechner r) {
   if (r.mach(42) > 0) {
     if(r.mach(42) < 0) {
       return 1;
     } else {
       return 2;
     }
   }
   return 3;
 }
}
```

Mit welchen der folgenden Mock-Objekte liefert der folgende Aufruf das Ergebnis 1?
`new Rechnernutzung().nutzen(r)`

☐
```
Rechner r = Mockito.mock(Rechner.class);
```

☐
```
Rechner r = Mockito.mock(Rechner.class);
Mockito.when(r.mach(42)).thenReturn(1);
```

☑
```
Rechner r = Mockito.mock(Rechner.class);
Mockito.when(r.mach(42))
       .thenReturn(Integer.MAX_VALUE).thenReturn(Integer.MIN_VALUE);
```

☑
```
Rechner r = Mockito.mock(Rechner.class);
Mockito.when(r.mach(ArgumentMatchers.anyInt())).thenReturn(1,-1);
```

☑
```
Rechner r = Mockito.mock(Rechner.class);
Mockito.when(r.mach(42)).thenAnswer( new Answer<>() {
  private int c = 0;

  @Override
  public Object answer(InvocationOnMock inv) throws Throwable {
    if (c++ == 0) {
      return 42;
    } else {
      return -42;
    }
}});
```

Antwort überprüfen

Quiz 2: Mockito

Welche der folgenden Aussagen über Mockito sind korrekt?

- [x] Mit Mockito ist es möglich, dass gemockte Methoden abhängig von übergebenen Parameterwerten unterschiedliche Ergebnisse erzeugen.
- [x] Mit Mockito ist es möglich, dass gemockte Methoden immer das gleiche Ergebnis zurückliefern.
- [ ] Werden bei Mockito die Rückgabewerte von gemockten Mothoden nicht spezifiziert, wird bei der Nutzung ein zufälliger Wert zurückgegeben.
- [x] Mit Mockito ist es möglich zu prüfen, wie oft eine gemockte Methode in einem Test aufgerufen wurde.
- [x] Mit Mockito ist es möglich Objekte zu erstellen, für die nur ein Interface gegeben ist.

Antwort überprüfen

Quiz 3: Spy

Welche der folgenden Aussagen über das Spy-Konzept sind korrekt?

- [ ] Mit Spy-Objekten kann sehr gut der innere Aufbau existierender Klassen erforscht werden.
- [x] Mit Spy-Objekten können einfach Exceptions geworfen werden, die sonst schwerer zu testen sind.
- [ ] Es ist generell sehr sinnvoll alle zum Testen genutzten Objekte ausser die der zu testenden Klasse selbst durch Spy-Objekte zu ersetzen.
- [x] Mit Spy-Objekten können Extremsituationen einfach erzeugt werden, die sonst sehr aufwendig zu erstellen sind.
- [ ] Es ist generell beim Testen sinnvoll Mock-Objekte zu nutzen und diese mit Spy-Funktionalität zu ergänzen.

Antwort überprüfen

zu 17)
Es kann sinnvoll sein, in die Enumeration einen weiteren, nicht im Programm genannten Wert, zu ergänzen.

```
package tracking;

public enum Alarm {
       CLEAR,GRUEN,GELB,ORANGE,ROT
}
```

```
package tracking;

public class Eigenschiff {

    private boolean wichtigerAuftrag;
```

```
    public Eigenschiff(boolean wichtigerAuftrag) {
        this.wichtigerAuftrag = wichtigerAuftrag;
    }

    public int getMaxKnoten() {
        return 0;
    }

    public boolean wichtigerAuftrag() {
        return wichtigerAuftrag;
    }

    public int hatAbwehr(boolean hatFK) {
        return 0;
    }

    public int sicherheit(int hatAbwehr, int abstand) {
        return 0;
    }

}
```

```
package tracking;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class GegnerTest {

    private Gegner g;
    private Eigenschiff e;

    @BeforeEach
    public void setUp() {
        g = new Gegner("K130", true, 100);
        e = new Eigenschiff(true);
    }

    @Test
    public void testAlert1() { // auch Abstand1
        g.knoten(Alarm.ROT, e, 20);
    }

    @Test
    public void testAlert2() { // auch WichtigerAuftrag1
        g.knoten(Alarm.ORANGE, e, 20);
    }

    @Test
    public void testAbstand2() {
        g.knoten(Alarm.ROT, e, 200);
    }

    @Test
    public void testWichtigerAuftrag2() {
        g.knoten(Alarm.ORANGE, new Eigenschiff(false), 20);
    }
}
```

```
16    public int knoten(Alarm alert, Eigenschiff e, int abstand) {
17        int ergebnis = 42;
18        if (alert == Alarm.ROT & abstand < this.feuerweite) {
19            ergebnis = e.getMaxKnoten();
20        }
21        if (alert == Alarm.ORANGE & e.wichtigerAuftrag()) {
22            ergebnis = e.sicherheit(e.hatAbwehr(this.hatFK), abstand);
23        }
24        return ergebnis;
25    }
26 }
27
```

Problems | Javadoc | Declaration | Console | Coverage | Call Hierarchy

GegnerTest (27.02.2024 15:07:33)

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
qsAufgabeGegnerMockLoesung	100,0 %	177	0	177
src	100,0 %	177	0	177
tracking	100,0 %	177	0	177
Alarm.java	100,0 %	54	0	54
Eigenschiff.java	100,0 %	15	0	15
Gegner.java	100,0 %	52	0	52
Gegner	100,0 %	52	0	52
Gegner(String, boolean, int)	100,0 %	12	0	12
knoten(Alarm, Eigenschiff, int)	100,0 %	40	0	40
GegnerTest.java	100,0 %	56	0	56

b)

```
package tracking;

public enum Alarm {
       CLEAR,GRUEN,GELB,ORANGE,ROT
}
```

```
package tracking;

public interface Eigenschiff {
      public int getMaxKnoten();

      public boolean wichtigerAuftrag();

      public int hatAbwehr(boolean hatFK);

      public int sicherheit(int hatAbwehr, int abstand);

}
```

```
package tracking;


import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.Mockito;
```

```
public class GegnerTest {

  private Gegner g;
  private Eigenschiff e;

  @Before
  public void setUp() throws Exception {
    g = new Gegner("K130", true, 100);
    e = Mockito.mock(Eigenschiff.class);
    Mockito.when(e.wichtigerAuftrag()).thenReturn(true);
  }

  @Test
  public void testAlert1() { // auch Abstand1
    g.knoten(Alarm.ROT, e, 20);
  }

  @Test
  public void testAlert2() { // auch WichtigerAuftrag1
    g.knoten(Alarm.ORANGE, e, 20);
  }

  @Test
  public void testAbstand2() {
    g.knoten(Alarm.ROT, e, 200);
  }

  @Test
  public void testWichtigerAuftrag2() {
    e = Mockito.mock(Eigenschiff.class);
    Mockito.when(e.wichtigerAuftrag()).thenReturn(false);
    g.knoten(Alarm.ORANGE, e, 20);
  }

  @Test
  public void testC0nr2() {
    e = Mockito.mock(Eigenschiff.class);
    Mockito.when(e.wichtigerAuftrag()).thenReturn(false);
    g.knoten(Alarm.ORANGE, e, 200);
  }
}
```

```
10    public Gegner(String typ, boolean hatFK, int feuerweite) {
11        this.typ = typ;
12        this.hatFK = hatFK;
13        this.feuerweite = feuerweite;
14    }
15
16    public int knoten(Alarm alert, Eigenschiff e, int abstand) {
17        int ergebnis = 42;
18        if (alert == Alarm.ROT & abstand < this.feuerweite) {
19            ergebnis = e.getMaxKnoten();
20        }
21        if (alert == Alarm.ORANGE & e.wichtigerAuftrag()) {
22            ergebnis = e.sicherheit(e.hatAbwehr(this.hatFK), abstand);
23        }
24        return ergebnis;
25    }
26 }
27
```

Problems | Javadoc | Declaration | Console | Coverage | Call Hierarchy

GegnerTest (1) (27.02.2024 16:12:05)

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
qsAufgabeGegnerMockitoLoesung	100,0 %	205	0	205
src	100,0 %	205	0	205
tracking	100,0 %	205	0	205
Alarm.java	100,0 %	54	0	54
Gegner.java	100,0 %	52	0	52
Gegner	100,0 %	52	0	52
Gegner(String, boolean, int)	100,0 %	12	0	12
knoten(Alarm, Eigenschiff, int)	100,0 %	40	0	40
GegnerTest.java	100,0 %	99	0	99

zu 18)
Fehler-Zaehler wird bei Exceptions bei beiden Pruefungen in einem Durchlauf um 2 erhoeht, so dass alarm zu schnell gesetzt wird

kann auch dazu führen, dass Fehlerzähler von 2 auf 4 springt und so der genau geforderte Wert von 3 nicht erreicht wird (Erinnerung: generell keine exakte Prüfung == x, wenn nicht notwendig)

a)

```
package maschine;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.Mockito;

public class UeberwachungTest {

    private UeberwachungInterface u;
```

```
    @Before
    public void setUp(){
        this.u = new Ueberwachung();  // new UeberwachungOK();
    }


    @Test
    public void testOhneFehler() throws NetworkException, HardwareException
{
        MInterface m = Mockito.mock(MInterface.class);
        Mockito.when(m.getWert1()).thenReturn(1);
        Mockito.when(m.getZustand1()).thenReturn("ok");
        this.u.setMaschine(m);
        for (int i = 1; i < 100; i++) {
            this.u.sensorenLesen();
        }
        Assert.assertFalse("kein Fehler erwartet", this.u.isAlarm());
    }

    @Test
    public void testOKmitzweiFehlernZwischenzeitlich()
            throws NetworkException, HardwareException {
        MInterface m = Mockito.mock(MInterface.class);
        Mockito.when(m.getWert1()).thenReturn(1)
                .thenThrow(new NetworkException())
                .thenThrow(new NetworkException())
                .thenReturn(1);
        Mockito.when(m.getZustand1()).thenReturn("ok");
        this.u.setMaschine(m);
        for (int i = 1; i < 100; i++) {
            this.u.sensorenLesen();
        }
        Assert.assertFalse("kein Fehler erwartet", this.u.isAlarm());
    }

    @Test
    public void testOKmitzweiAbwechselndenFehlernZwischenzeitlich()
            throws NetworkException, HardwareException {
        MInterface m = Mockito.mock(MInterface.class);
        Mockito.when(m.getWert1())
                .thenReturn(1)
                .thenThrow(new HardwareException())
                .thenReturn(1)
                ;
        Mockito.when(m.getZustand1())
                .thenReturn("ok")
                .thenReturn("ok")
                .thenThrow(new NetworkException())
                .thenReturn("ok")
                ;
        this.u.setMaschine(m);
        for (int i = 1; i < 100; i++) {
            this.u.sensorenLesen();
        }
        Assert.assertFalse("kein Fehler erwartet", this.u.isAlarm());
    }

    @Test
    public void testOKmitzweiMehrfachAbwechselndenFehlernZwischenzeitlich()
```

```
            throws NetworkException, HardwareException {
        MInterface m = Mockito.mock(MInterface.class);
        Mockito.when(m.getWert1())
                .thenReturn(1)
                .thenThrow(new HardwareException())
                .thenReturn(6) // Sprung ok
                .thenReturn(1)
                .thenReturn(7) // Sprung nicht ok
                .thenReturn(2)
                ;
        Mockito.when(m.getZustand1())
                .thenReturn("ok")
                .thenReturn("ok")
                .thenThrow(new NetworkException())
                .thenReturn("ok")
                .thenReturn("ok")
                .thenThrow(new NetworkException())
                .thenReturn("ok")
                ;
        this.u.setMaschine(m);
        for (int i = 1; i < 100; i++) {
            this.u.sensorenLesen();
        }
        Assert.assertFalse("kein Fehler erwartet", this.u.isAlarm());
    }

    // dieser Test findet ersten Fehler, da Fehler-Zaehler
    // bei Exceptions bei beiden Pruefungen in einem Durchlauf
    // um 2 erhoeht wird
    @Test
    public void testKeinDefektbeiZweiGemeinsamenFehlern()
            throws NetworkException, HardwareException {
        MInterface m = Mockito.mock(MInterface.class);
        Mockito.when(m.getWert1())
                .thenReturn(1)
                .thenThrow(new HardwareException())
                .thenThrow(new NetworkException()) // gleichzeitig
                .thenReturn(6)
                .thenReturn(1)
                .thenReturn(6)
                .thenReturn(1)
                ;
        Mockito.when(m.getZustand1())
                .thenReturn("ok")
                .thenReturn("ok")
                .thenThrow(new NetworkException())// gleichzeitig
                .thenReturn("ok")
                ;
        this.u.setMaschine(m);
        for (int i = 1; i <= 7; i++) {
            this.u.sensorenLesen();
        }
        Assert.assertFalse("kein Fehler erwartet", this.u.isAlarm());
    }


    @Test
    public void testZwischenzeitlicherDefekt()
            throws NetworkException, HardwareException {
```

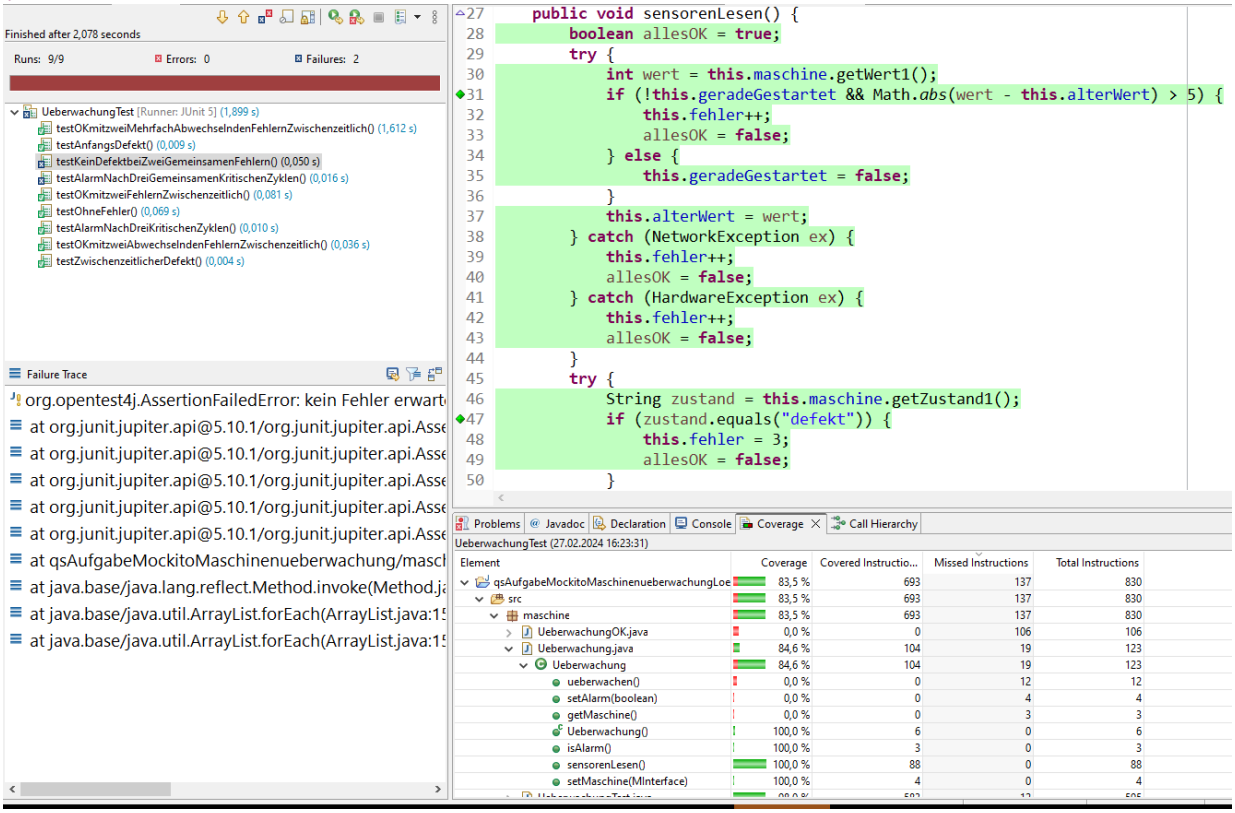
```
        MInterface m = Mockito.mock(MInterface.class);
        Mockito.when(m.getWert1())
                .thenReturn(1)
                .thenThrow(new HardwareException())
                .thenReturn(6)
                .thenReturn(1)
                .thenReturn(7)
                .thenReturn(2)
                ;
        Mockito.when(m.getZustand1())
                .thenReturn("ok")
                .thenReturn("ok")
                .thenThrow(new NetworkException())
                .thenReturn("ok")
                .thenReturn("ok")
                .thenThrow(new NetworkException())
                .thenReturn("defekt")
                ;
        this.u.setMaschine(m);
        for (int i = 1; i <= 7; i++) {
            this.u.sensorenLesen();
        }
        Assert.assertTrue("Fehler erwartet", this.u.isAlarm());
    }

    @Test
    public void testAnfangsDefekt()
            throws NetworkException, HardwareException {
        MInterface m = Mockito.mock(MInterface.class);
        Mockito.when(m.getWert1())
                .thenReturn(1)
                ;
        Mockito.when(m.getZustand1())
                .thenReturn("defekt")
                ;
        this.u.setMaschine(m);
        for (int i = 1; i <= 1; i++) {
            this.u.sensorenLesen();
        }
        Assert.assertTrue("Fehler erwartet", this.u.isAlarm());
    }

    @Test
    public void testAlarmNachDreiKritischenZyklen()
            throws NetworkException, HardwareException {
        MInterface m = Mockito.mock(MInterface.class);
        Mockito.when(m.getWert1())
                .thenReturn(1)
                .thenThrow(new HardwareException())
                .thenReturn(7) // Fehler
                .thenReturn(3)
                ;
        Mockito.when(m.getZustand1())
                .thenReturn("ok")
                .thenReturn("ok")
                .thenReturn("ok")
                .thenThrow(new NetworkException())
                ;
        this.u.setMaschine(m);
```

```
        for (int i = 1; i <= 4; i++) {
            this.u.sensorenLesen();
        }
        Assert.assertTrue("Fehler erwartet", this.u.isAlarm());
    }


    // dieser Test findet den Fehler, da Fehler-Wert immer um 2
    // erhoeht wird und so nie genau den Grenzwert 3 erreicht
    @Test
    public void testAlarmNachDreiGemeinsamenKritischenZyklen()
            throws NetworkException, HardwareException {
        MInterface m = Mockito.mock(MInterface.class);
        Mockito.when(m.getWert1())
                .thenReturn(1)
                .thenThrow(new HardwareException())
                .thenReturn(7) // Fehler
                .thenReturn(1) // Fehler
                ;
        Mockito.when(m.getZustand1())
                .thenReturn("ok")
                .thenThrow(new NetworkException())
                .thenThrow(new NetworkException())
                .thenThrow(new NetworkException())
                ;
        this.u.setMaschine(m);
        for (int i = 1; i <= 4; i++) {
            this.u.sensorenLesen();
        }
        Assert.assertTrue("Fehler erwartet", this.u.isAlarm());
    }
}
```



b)

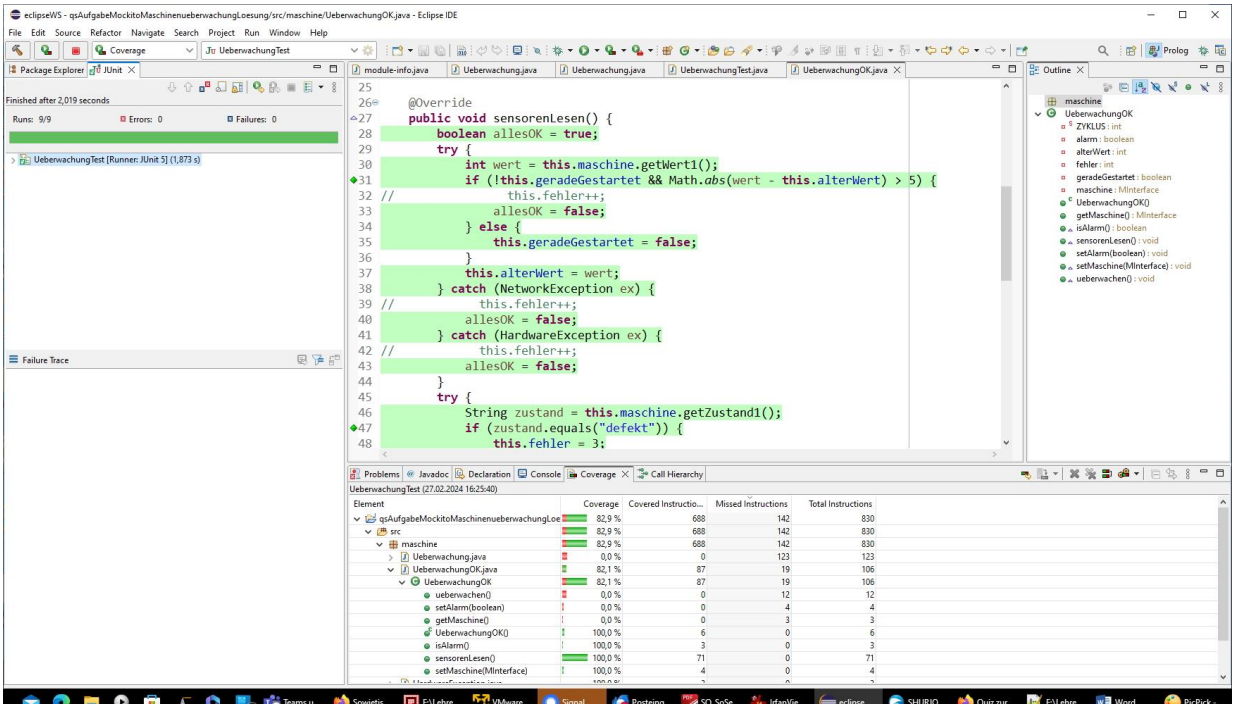
```
package maschine;

public class UeberwachungOK implements UeberwachungInterface {

    private MInterface maschine;
    private static int ZYKLUS = 1000; // Waertezeit in ms
    private boolean alarm;
    private int fehler;
    private int alterWert;
    private boolean geradeGestartet = true;

    public UeberwachungOK() {
    }

    @Override
    public void ueberwachen() {
        while (!alarm) {
            try {
                Thread.sleep(ZYKLUS);
            } catch (InterruptedException ex) {
            }
            this.sensorenLesen();
        }
    }

    @Override
    public void sensorenLesen() {
        boolean allesOK = true;
        try {
            int wert = this.maschine.getWert1();
            if (!this.geradeGestartet
                    && Math.abs(wert - this.alterWert) > 5) {
//                this.fehler++;
                allesOK = false;
            } else {
                this.geradeGestartet = false;
            }
            this.alterWert = wert;
        } catch (NetworkException ex) {
//            this.fehler++;
            allesOK = false;
        } catch (HardwareException ex) {
//            this.fehler++;
            allesOK = false;
        }
        try {
            String zustand = this.maschine.getZustand1();
            if (zustand.equals("defekt")) {
                this.fehler = 3;
                allesOK = false;
            }
        } catch (NetworkException ex) {
//            this.fehler++;
            allesOK = false;
        }
//        if (this.fehler == 3) {
```

```
//              this.alarm = true;
//          }
            if (allesOK) {
                this.fehler = 0;
            } else {  // neu
                this.fehler ++;
            }
            if (this.fehler >= 3) {
                this.alarm = true;
            }
        }

        public MInterface getMaschine() {
            return maschine;
        }

        @Override
        public void setMaschine(MInterface maschine) {
            this.maschine = maschine;
        }

        @Override
        public boolean isAlarm() {
            return alarm;
        }

        public void setAlarm(boolean alarm) {
            this.alarm = alarm;
        }
}
```