Umsetzung der Softwareprojekte

Für die Umsetzung der Softwareprojekte muss man sich zunächst für ein Vorgehensmodell entscheiden. Die agile Herangehenweise hat hier in der Regel in allen Punkten die Nase vorn. Damit man ein Projekt so angehen kann, dass man die genannten typischen Fehler bei der Entwicklung von Code nicht macht, ist es notwendig, dass man sich noch einiger Hilfsmittel bedient. Essentielle Hilfsmittel werden hier vorgestellt.

Dokumentieren in Java

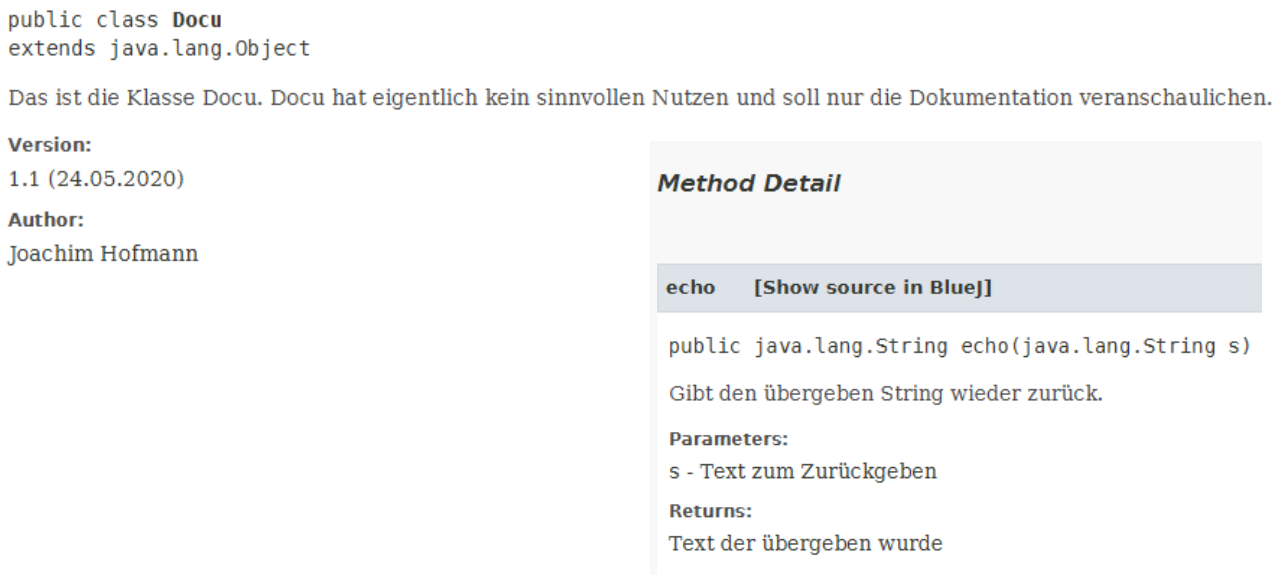
Im Folgenden wird gezeigt, wie das Dokumentieren in Java an den üblichen Codestellen funktioniert: Eine Dokumentation wird mit /** eingeleitet und endet mit */
In jeder Zeile wird wieder in * am Anfang der Zeile benötigt.

Kommentare: // für eine Kommentarzeile **oder**
der Kommentar beginnt mit /* und endet mit */

Für die Dokumentation gibt man neben einer Kurzbeschreibung noch einige ***Tags*** an. Eine Dokumentation könnte folgendermaßen aussehen:

```
/**
 * Das ist die Klasse Docu. Docu hat eigentlich kein sinnvollen Nutzen
 * und soll nur die Dokumentation veranschaulichen.
 *
 * @author      Joachim Hofmann
 * @version     1.1 (24.05.2020)
 *
 * @changelog   1.1 Änderungen bei Version 1.1 im Vergleich zu 1.0
 *                  z.B. Bug bei Methode xy() behoben
 *              1.0 Grundlegende Erklärungen für die Klasse in der Version 1.0
 */
public class Docu{
    /**
     * Konstruktor Docu
     */
    public Docu(){      }
    /**
     * Gibt den übergeben String wieder zurück.
     *
     * @param   s   Text zum Zurückgeben
     *
     * @return      Text der übergeben wurde
     */
    public String echo(String s){
        return s;
    }
}
```

Aus der im Code geschriebenen Dokumentation wird eine Dokumentationsseite im Java-Stil erstellt. Diese kann bei BlueJ ***eingesehen werden***, wenn man rechts oben im Fenster ***Dokumentation*** anstatt Quelltext auswählt.



public class **Docu**
extends java.lang.Object

Das ist die Klasse Docu. Docu hat eigentlich kein sinnvollen Nutzen und soll nur die Dokumentation veranschaulichen.

Version:
1.1 (24.05.2020)

Author:
Joachim Hofmann

Method Detail

echo **[Show source in BlueJ]**

public java.lang.String echo(java.lang.String s)

Gibt den übergeben String wieder zurück.

Parameters:
s - Text zum Zurückgeben

Returns:
Text der übergeben wurde

Das Semaphorprinzip – Synchronisierung

Problemstellung:

Im Kino gibt es mehrere Kassen. Angenommen zwei Personen möchten nahezu gleichzeitig an verschiedenen Kassen die exakt gleichen Sitzplätze ein und derselben Vorstellung reservieren und kaufen. Wer bekommt die Karten?

→ ***Das System muss sicherstellen, dass Karten nicht doppelt reserviert werden können!***

Wenn ***nebenläufige Vorgänge*** auf gemeinsame Ressourcen (hier: Sitzplätze) zugreifen und sich dabei gegenseitig stören können, müssen die Zugriffe ***synchronisiert*** werden.

Der ***wechselseitige Ausschluss*** kann mithilfe von ***Semaphoren*** realisiert werden.

Achtung: im Vergleich zu anderen Programmiersprachen (z.B. C) gibt es in Java keine Semaphore. Aber es gibt das Schlüsselwort **synchronized** (Stoff der 12. Jahrgangsstufe Synchronisation von Prozessen)

Ein ***kritischer Abschnitt*** ist der Teil eines Vorgangs, in dem auf gemeinsam genutzte Betriebsmittel/Ressourcen zugegriffen wird (hier: Reservieren des Sitzplätze). Das nebenläufige Ausführen dieses Abschnitts ***kann*** zu Fehlern führen und er muss deshalb synchronisiert werden.

Eine ***Verklemmung (deadlock)*** ist entstanden, wenn keiner von mehreren nebenläufigen Vorgängen fortgesetzt werden kann. (z.B. weil alle Vorgänge gegenseitig aufeinander warten → ***Philosophenproblem***)

Versionsverwaltungsprogramme

Da bei Softwareprojekten in einem Team gearbeitet wird, ist es notwendig, dass man manchmal auch gleichzeitig an Dateien/Codeabschnitten arbeiten kann. Damit dies nicht zu Fehler führt (siehe letztes Abschnitt: Das Semaphorprinzip – Synchronisierung), muss man eine Möglichkeit der „Synchronisierung“ finden. Die einfachste Variante ist, dass der Hilfe von ***Versionsverwaltungsprogrammen*** bedient.

Diese Programme sind Systeme zur Erfassung von Änderungen an Dokumenten oder Dateien. Zudem ermöglichen sie das gemeinsame Arbeiten an Dateien: Falls zwei Personen am selben Codeabschnitt Änderungen vorgenommen haben, muss der Zweite beim Einreichen seiner Änderungen (commit) eine ***Zusammenführung (merge)*** bei Konflikten durchführen. Er muss dann die beiden Versionen der beiden Personen zu einer Version zusammenführen.

Eines der bekanntesten Versionsverwaltungsprogrammen ist ***Git***.

Projektmanagement (insbesondere Taskboard)

Bei einem agilen Vorgehensmodell gibt es immer ein ***Taskboard*** (dt. Aufgabenbrett), in welchem die ***User Stories / Aufgaben*** meist in den ***Kategorien (ToDo, in Progress, Done)*** festgehalten werden. Hier ist es auch verschiedenen Gründen (räumliche Distanz zwischen den Teammitgliedern, Homeoffice, besserer Überblick, Analyseoptionen usw.) hilfreich, wenn das Taskboard sowie ***viele andere Aspekte des Projektmanagement*** digitalisiert sind.

Bekannte Projektmanagementprogramme sind z.B. Trello, Jira und viele mehr.

Zudem ist es möglich und auch sehr hilfreich, dass diese Projektmangementsysteme mit Git ***kombiniert/verknüpft*** werden, um z.B. so fertige Aufgaben mit den passenden Commits zu versehen und viele weitere Vorteile zu erhalten.

Testen mit JUnit-Tests

Ein wesentlicher Bestandteil der Softwareentwicklung ist das Testen. Hierbei soll zugesichert werden, dass die Software auch entsprechend gesteckten Anforderungen erfüllt. Neben vieler andere Möglichkeiten ist das Schreiben von Testfällen ein üblicher Bestandteil. Mithilfe von ***JUnit*** lassen sich solche Testfälle einfach implementieren.

In BlueJ lässt sich ein JUnit-Test wie eine Klasse erzeugen. Dieser besteht in der Regel aus drei Teilen:

- ***@Before*** kennzeichnet die Methode für das Initialisieren der Tests z.B. Erstellen von Objekten

- ***@Test*** kennzeichnet eine Testmethode. Hiervon kann es sehr viele in einer Testklasse geben.
- ***@After*** kennzeichnet die Methode zum Aufräumen. Dies wird in Java aufgrund der ***Garbage Collectors*** oft nicht benötigt.

```
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
public class TestTaschenrechner{
    Taschenrechner t;
    @Before
    public void setUp(){
        t = new Taschenrechner();
    }
    @Test
    public void testPlus(){
        assertEquals(13,t.plus(5, 8));
        assertEquals(-7,t.plus(-2, -5));
    }
    @Test
    public void testWurzel(){
        //Parameter(expected, actual, [difference])
        assertEquals(1.4142135623730951,t.wurzel(2),0.0000001);
        assertTrue(Double.isNaN(t.wurzel(-5)));
    }
    @After
    public void tearDown(){
    }
}
```

Es gibt für nahezu jede Anwendung eine passende ***assertXXX(...)-Methode***. Selbstverständlich können zum Beispiel auch Objekte miteinander verglichen werden oder deren Attribute (Inhalte). Damit ist es auch kein Problem ***Testfälle für Listen, Bäume, Graphen oder Entwurfsmuster*** zu bauen.

Aufgabe:

Schreibe für eine oder mehrerer Methoden aus einer Klasse der genannten Themengebiete aus diesem Schuljahr eine JUnit-Testklasse.