Softwarequalität, Testen und Versionsverwaltung

Softwarequalität geht über die reine Funktionalität hinaus.

Softwarequalität

- **Wartbarkeit** (Software wird in der Regel noch längere Zeit gewartet und erweitert)
- **Guter Code muss lesbar sein** (= klare Struktur und Namensgebung von Klassen, Methoden und Variablen), leichte (erneute) Einarbeitung in das Projekt
- **Modularer Aufbau** (= eine Klasse eine Aufgabe) **statt Spaghetticode** (= zu komplex verwobener Code, sodass nachträgliche Änderungen oder Erweiterungen kaum möglich sind)
- **Nachträgliche Ordnung im Code durch Refactoring** (= Anpassungen z. B. von Namen ohne die Funktionalität zu ändern → Bei einer Namensänderung einer Variablen erkennt die Entwicklungsumgebung alle Nutzungen der Variablen und ändern dort ebenfalls den Namen.)

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.

```java
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 oder Entwurfsmuster*** zu bauen.

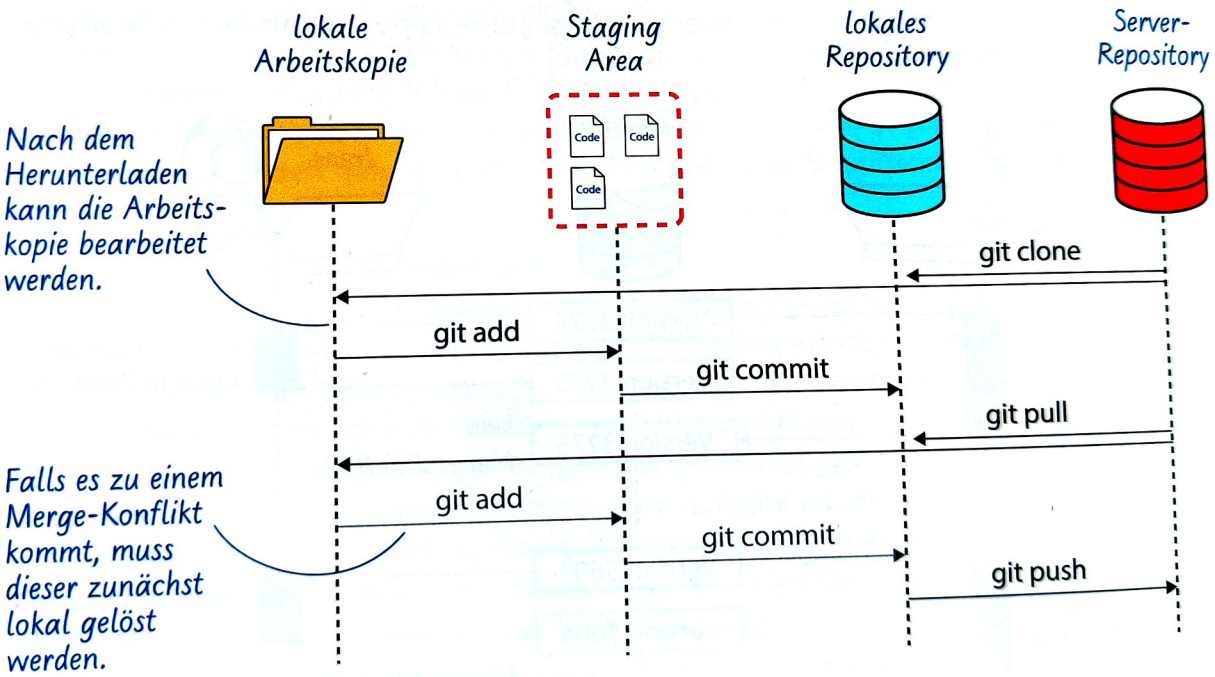
Aufgabe:

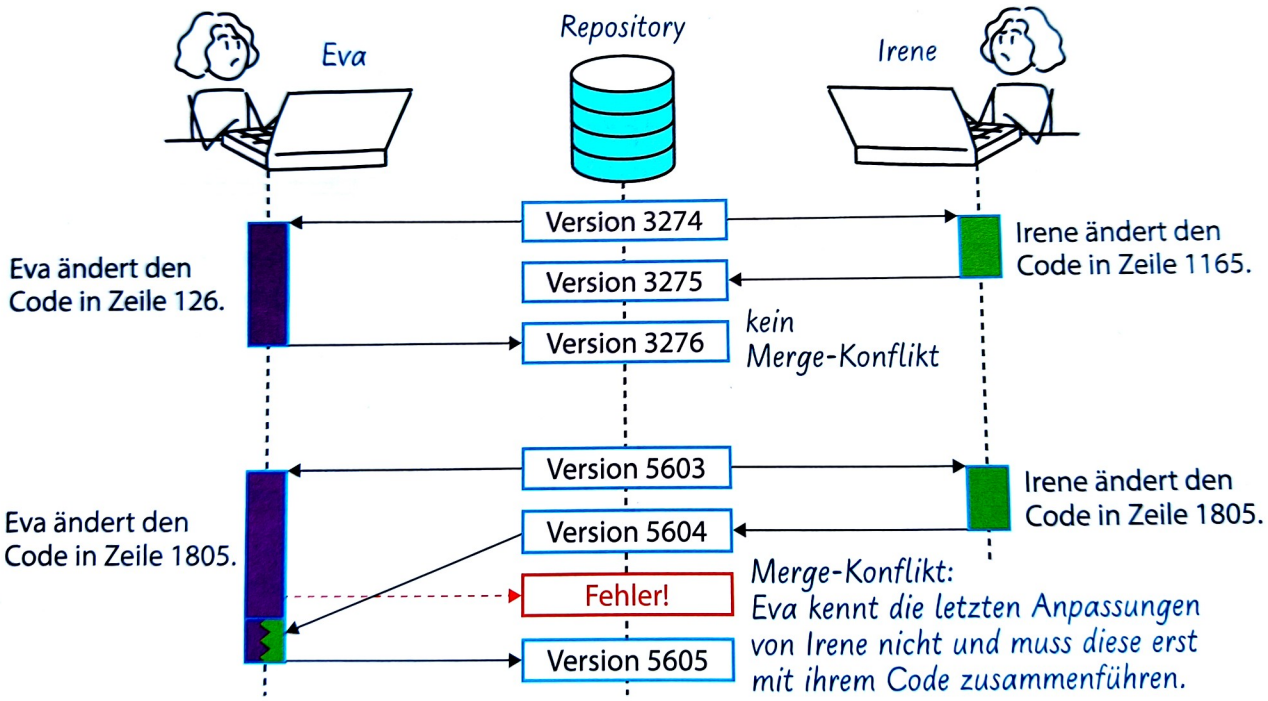
Schreibe für eine oder mehrerer Methoden aus einer Klasse der genannten Themengebiete aus diesem Schuljahr eine JUnit-Testklasse.

Versionsverwaltungsprogramme

Diese ***Versionsverwaltungsprogramme*** 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***.


lokale Arbeitskopie
Staging Area
lokales Repository
Server-Repository
Code
Code
Code
Nach dem Herunterladen kann die Arbeits-kopie bearbeitet werden.
git clone
git add
git commit
git pull
Falls es zu einem Merge-Konflikt kommt, muss dieser zunächst lokal gelöst werden.
git add
git commit
git push


Eva
Repository
Irene
Version 3274
Version 3275
Version 3276
Eva ändert den Code in Zeile 126.
Irene ändert den Code in Zeile 1165.
kein Merge-Konflikt
Version 5603
Version 5604
Fehler!
Version 5605
Eva ändert den Code in Zeile 1805.
Irene ändert den Code in Zeile 1805.
Merge-Konflikt:
Eva kennt die letzten Anpassungen von Irene nicht und muss diese erst mit ihrem Code zusammenführen.