| **NACHNAME:** | **SEMESTER:** ☐ **M5** ☐ **M6** ☐ M3 ☐ M4 ☐ M7 |
|---|---|
| **VORNAME:** | **VERTIEFUNG:** ☐ **FV** ☐ **IM** |

**VERWENDETE KLASSEN:**

- Als **Anlage** erhalten Sie einen Ausdruck des vorab bekannt gemachten Quelltextes von verschiedenen Klassen, insbesondere von mehreren Versionen einer Klasse `Wort`.

**UNBEDINGT BEACHTEN:**

- **Bevor** Sie mit der Bearbeitung beginnen, müssen die Angaben zur Person auf dieser Seite **vollständig** ausgefüllt sein.
- Es sind **keinerlei Hilfsmittel** zugelassen.

**GENERELLE VORGABEN:**

- Es sind keinerlei Kommentare verlangt, weder `javadoc`-Kommentare noch andere.
- Methoden müssen vollständig, auch mit Annotationen, angegeben werden. Die umgebende Klasse und etwaige `import`–Anweisungen werden nicht verlangt.
- Die einzelnen Zeichen – auch Groß- oder Klein-Schreibung – müssen klar erkennbar sein.
- Programmier-Richtlinien (insbesondere Checkstyle) sind zu beachten. Bei Testklassen dürfen aber *magic numbers* verwendet werden.

## *Aufgabe 1: (35 Punkte)*

Die Aufgabe bezieht sich (unter anderem) auf die **drei Versionen** der Klasse `Wort` und die Methode `tuWasMit` der Klasse `BleedsGschwaetz`.

Alle von `Wort` abhängigen Klassen beziehen sich auf dieselbe Version. Zur gleichen Zeit wird demnach immer nur eine Version der Klasse `Wort` verwendet.

**a)** An geeigneter Stelle werde die Klasse `HauptWort` definiert:

```
package aufgabe2;
import anhang.version?.Wort; // ? durch jeweilige Version ersetzen
import static anhang.Strings.*;

public class HauptWort extends Wort{
    public HauptWort(String s) {
        super(s);
        if (!istHauptWort(s)) throw new IllegalArgumentException();
    }
}
```

An einer anderen Stelle werden zwei `Wort`e initialisiert:

```
Wort w = new Wort("Integrabilitätskriterium");
Wort m = new HauptWort("Integrabilitätskriterium");
```

Geben Sie an, welchen Wert dann die in der linken Spalte angegebenen Ausdrücke jeweils haben. Abkürzungen sind nicht erlaubt. Tragen Sie **Exc.** ein, wenn der Aufruf zu einer `Exception` führt.

| | version1 | version2 | version3 |
|---|---|---|---|
| `tuWasMit(w, w, m)` | | | |
| `tuWasMit(m, m, w)` | | | |
| `tuWasMit(w, w)` | | | |
| `tuWasMit(w, m)` | | | |
| `tuWasMit(m, w, m)` | | | |

**b)** Nun geht es um die Klasse `Name`:

```
package aufgabe2;
import anhang.*;

public class Name extends HauptWort {
    private Anrede mAnrede;
    public Name(String s, Anrede a) {
        super(s);
        mAnrede = a;
    }
    @Override
    public boolean equals(Object vergl) {
        if (!super.equals(vergl)) return false;
        Name n = (Name) vergl;
        return n.mAnrede == mAnrede;
    }
}
```

Irgendwo werden drei `Wort`e initialisiert:

```
Wort w = new Wort("Lich");
Wort m1 = new Name("Lich", FRAU);
Wort m2 = new Name("Lich", HERR);
```

Tragen Sie wieder die Ergebnisse der angegebenen `tuWasMit`-Aufrufe in die Tabelle ein. Das Auftreten von `Exception`s soll wie zuvor mit **Exc.** markiert werden.

| | version1 | version2 | version3 |
|---|---|---|---|
| `tuWasMit(w, w, m1)` | | | |
| `tuWasMit(m1, m1, w)` | | | |
| `tuWasMit(m1, m1, m2)` | | | |
| `tuWasMit(m1, m2)` | | | |

**c)** Schließlich soll die Klasse `BenutzerName` betrachtet werden:

```
package aufgabe2;
import anhang.*;
import anhang.version?.*; // ? durch jeweilige Version ersetzen

public class BenutzerName extends Name {
    private boolean mGast;
    private BenutzerName(String s, Anrede a, boolean gast) {
        super(s, a);
        mGast = gast;
    }
    public static BenutzerName gast(String s, Anrede a) {
        return new BenutzerName(s, a, true);
    }
    public static BenutzerName benutzer(String s, Anrede a) {
        return new BenutzerName(s, a, false);
    }
    public boolean equals(Wort vergl) {
        if (!super.equals(vergl)) return false;
        BenutzerName b = (BenutzerName) vergl;
        return b.mGast == mGast;
    }
}
```

In der Tabelle soll nun sein:

```
Wort m1 = BenutzerName.gast("Lich", FRAU);
Wort m2 = BenutzerName.gast("Lich", HERR);
Wort m3 = BenutzerName.benutzer("Lich", HERR);
```

| | version1 | version2 | version3 |
|---|---|---|---|
| `tuWasMit(m1, m1, m2)` | | | |
| `tuWasMit(m2, m2, m3)` | | | |

## *Aufgabe 2: (30 Punkte)*

Die Aufgabe bezieht sich auf die Klasse `MusterStrings` der Anlage.

**a)** Ergänzen Sie (in den eckigen Klammern):

Ein Aufruf der Methode `entsprichtMuster` über ein Objekt der Klasse, etwa `ms.entsprichtMuster("?ier", "vier")` wird nicht empfohlen. Stattdessen soll [ ] geschrieben werden, um klar zu machen, dass die Methode [ ] ist, also nicht von konkreten Objekten abhängt. Mit Hilfe eines [ ] Imports können alle derartigen Methoden kürzer angesprochen werden, hier konkret [ ]. Die zugehörige Anweisung lautet [ ].

Die schlechte Art des Aufrufs (über ein Objekt) kann außerhalb der Klasse durch einen [ ] verhindert werden.

**b)** Eine der Methoden in `MusterStrings` erzeugt einen String, bei welchem alle Fragezeichen in einem Muster durch ein vorgegebenes Zeichen ersetzt wurden. Nun wird eine zusätzliche Methode `worteNachMuster` benötigt, welche dies nacheinander für alle Zeichen eines vorgegebenen Strings macht und die insgesamt dabei entstehenden Strings als Set zurück gibt. Insbesondere soll `worteNachMuster("da??", "sn")` ein Set mit den beiden Strings `"dass"` und `"dann"` liefern.

Schreiben Sie eine Test-Methode, welche zunächst genau dieses Beispiel prüft und dann das Ergebnis von `worteNachMuster("?ier", "BETT")`.

**c)** Implementieren Sie diese Methode `worteNachMuster`.

## *Aufgabe 3: (25 Punkte)*

Die Aufgabe bezieht sich auf die Klasse `MusterStrings` der Anlage **mit der** in Aufgabe 2 vorgenommenen **Erweiterung um** die Methode `worteNachMuster`.

**a)** Die folgende Test-Methode definiert Anforderungen an eine neue Methode der Klasse `MusterStrings`:

```
@Test
public void testeNeueMethode() {
    String[] ist1 = worteNachMuster("?* = ?stern", "O", "ge");
    String[] soll1 = {"O* = Ostern", "ge* = gestern"};
    assertTrue(Arrays.equals(soll1, ist1));
    String[] ist2 = worteNachMuster("?braham", "A", "B", "Ze");
    String[] soll2 = {"Abraham", "Bbraham", "Zebraham"};
    assertTrue(Arrays.equals(soll2, ist2));
    String[] ist3 = worteNachMuster("Was geht?");
    assertEquals(0, ist3.length);
}
```

Implementieren Sie die neue Methode so, dass dieser Test erfolgreich ist:

**b)** Als Muster werde im Folgenden verwendet:

```
String muster = "wenn hinter ? ? ?, ? ? ? nach";
```

Warum führt die folgende Zeile zu einem Compile-Fehler?

```
String[] erg = worteNachMuster(muster, "fliegen");
```

Warum wird dann das Folgende problemlos compiliert? Wie viele Zeilen werden ausgegeben? Begründen Sie das Ergebnis.

```
for (String s : worteNachMuster(muster, "fliegen"))
    System.out.println(s);
```