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.
- Die direkt in diesen Aufgabenblättern gezeigten Code-Auszüge sind sinnvoll ergänzt zu denken, insbesondere durch geeignete `import`-Anweisungen.

UNBEDINGT BEACHTEN:

- **Bevor** Sie mit der Bearbeitung beginnen, müssen die Angaben zur Person auf dieser Seite **vollständig** ausgefüllt sein.
- Es sind **keinerlei Hilfsmittel** zugelassen, auch kein zusätzliches Konzeptpapier.
- Die zusammengehefteten Blätter dürfen **nicht getrennt** werden.

GENERELLE VORGABEN:

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

Aufgabe 1: (7 Punkte)

Durch Überladen soll in der Klasse `Stellung` eine Methode ohne Argumente entstehen. Die Methode soll die Länge der längsten Kette an gleichfarbigen (roten oder gelben) Steinen in einer Linie (waagrecht, senkrecht oder diagonal) liefern. Implementieren Sie die neue Methode.

Aufgabe 2: (12 Punkte)

Die Klasse `StellungsEntwicklung` soll um eine Methode namens `rueckgaengig` erweitert werden, welche den zuletzt ausgeführten Halbzug zurück nimmt.

a) Ein zugehöriger Test bezieht sich auf eine `StellungsEntwicklung se`, die durch zwei Halbzüge entstanden ist. Überprüfen Sie die Zurücknahme von einem beziehungsweise von beiden Halbzügen. Testen Sie dabei jeweils lediglich die zugehörige `Stellung`. Überprüfen Sie abschließend das Verhalten bei dem Versuch, einen dritten Halbzug zurück zu nehmen.

```
@Test
public void testeRueckgaengig() {
    StellungsEntwicklung se = new StellungsEntwicklung(GELB, B, E);
```

```
}
```

b) Implementieren Sie nun die Methode `rueckgaengig`.

Aufgabe 3: (15 Punkte)

a) Der folgende Code werde mit dem Debugger von Eclipse durchlaufen. An mehreren Stellen werden die Werte der Variablen `s1` und `s2` überprüft. Tragen Sie in den Kästen den jeweils aktuellen Wert ein, **genau** wie er vom Debugger angezeigt wird.

```
Stellung s1 = new Stellung(ROT);
Stellung s2 = new Stellung(s1, C);
```

s1	
s2	

```
Farbe[][] m1 = s1.matrix();
Farbe[][] m2 = s2.matrix();
m1[0][0] = ROT;
m2[0][0] = GELB;
m1[1][0] = GELB;
m2[1][0] = ROT;
```

s1	
s2	

```
m1[0][0] = LEER;
m2[1][6] = ROT;
```

s1	
s2	

b) Wie in der vorigen Teilaufgabe soll auch für den nachstehenden Code die **genaue** Anzeige der Werte von `s1` und `s2` angegeben werden.

```
Stellung s1 = new Stellung(GELB, D, F, D);
Stellung s2 = new Stellung(s1, F);
```

s1	
s2	

```
Farbe[][] m1 = s1.matrix();
Arrays.sort(m1[0]);
Arrays.sort(m1[1]);
```

s1	
s2	

```
Farbe[][] m2 = s2.matrix();
m1 = m2;
m2[0] = m2[1];
```

s1	
s2	

```
s1 = new Stellung(s2, A);
Arrays.sort(m2[0]);
```

s1	
s2	

Aufgabe 4: (13 Punkte)

Die folgenden Zeilen zeigen die Verwendung der Klasse `DarStellung` zur übersichtlichen Console-Anzeige von Stellungen.

```
StellungsEntwicklung se = new StellungsEntwicklung(ROT, G, A, G, A);
DarStellung.darstellen(se); // 1
se.einwerfenBei(E, B, B, E); // 2
se.einwerfenBei(B, A, D); // 3
se.stellung().matrix()[0][6] = LEER; // 4
DarStellung.darstellen(se); // 5
se.einwerfenBei(G, A, G); // 6
```

In der folgenden Tabelle sollen die Anzeigen eingetragen werden. Die erste Zelle der Tabelle ist beispielhaft komplett ausgefüllt. Sie zeigt die Console-Anzeige durch die mit `// 1` markierte Code-Zeile. In den anderen Zellen sind die dort fehlenden Angaben zu ergänzen. Beachten Sie dazu unbedingt die auf der nächsten Seite angegebenen Erläuterungen.

- - - - - - - **// 1** - - - - - - - - - - - - - - Beobachter: #1 - - - - - - - G - - - - - R Halbzuege: 4 G - - - - - R am Zuge: ROT	- - - - - - - **//** __ - - - - - - - _____ Beobachter: #__ _____ _____ Halbzuege: ____ _____ am Zuge: _____
- - - - - - - **//** __ - - - - - - - _____ Beobachter: #__ _____ _____ Halbzuege: ____ _____ am Zuge: _____	- - - - - - - **//** __ - - - - - - - _____ Beobachter: #__ _____ _____ Halbzuege: ____ _____ am Zuge: _____
- - - - - - - **//** __ - - - - - - - _____ Beobachter: #__ _____ _____ Halbzuege: ____ _____ am Zuge: _____	- - - - - - - **//** __ - - - - - - - _____ Beobachter: #__ _____ _____ Halbzuege: ____ _____ am Zuge: _____

Die nachstehenden Hinweise und Vorgaben sind unbedingt einzuhalten. Sie wurden aber allesamt bereits in der Vorlesung bekannt gegeben und anhand von Beispielen erklärt.

- Insgesamt erfolgen im angegebenen Code genau 6 Ausgaben einer Stellung. Nicht in jeder Zeile muss aber genau eine Ausgabe erfolgen. Es könnten mehrere sein oder aber auch gar keine.
- Oben rechts soll deshalb zusätzlich zur Console-Ausgabe die Zeile angegeben werden, in welcher diese Ausgabe erfolgt. Im Beispiel ist dies `// 1`.
- Sofern in einer Zeile mehrere Ausgaben erfolgen, ist die Reihenfolge der Angaben belanglos.
- Wo Ergänzungen nötig sind können Sie durch Vergleich mit der Beispiel-Zelle erkennen. Zusätzlich sind die betreffenden Stellen durch einen Unterstrich gekennzeichnet.
- Es sind alle Zeichen klar und deutlich einzutragen, also auch das `-` als Kennzeichen einer unbelegten Stelle. Dies gilt sogar, wenn eine ganze Reihe unbelegt ist.
- Für nötige Korrekturen stehen als Reserve weitere Zellen zur Verfügung:

`- - - - - - - -      // __`
`- - - - - - - -`
`_____________ Beobachter: #__`
`_____________`
`_____________ Halbzuege: ____`
`_____________ am Zuge: ______`
	`- - - - - - - -      // __`
	`- - - - - - - -`
	`_____________ Beobachter: #__`
	`_____________`
	`_____________ Halbzuege: ____`
	`_____________ am Zuge: ______`

`- - - - - - - -      // __`
`- - - - - - - -`
`_____________ Beobachter: #__`
`_____________`
`_____________ Halbzuege: ____`
`_____________ am Zuge: ______`
	`- - - - - - - -      // __`
	`- - - - - - - -`
	`_____________ Beobachter: #__`
	`_____________`
	`_____________ Halbzuege: ____`
	`_____________ am Zuge: ______`

`- - - - - - - -      // __`
`- - - - - - - -`
`_____________ Beobachter: #__`
`_____________`
`_____________ Halbzuege: ____`
`_____________ am Zuge: ______`
	`- - - - - - - -      // __`
	`- - - - - - - -`
	`_____________ Beobachter: #__`
	`_____________`
	`_____________ Halbzuege: ____`
	`_____________ am Zuge: ______`

Aufgabe 5: (22 Punkte)

Es soll ein `Beobachter` für eine `StellungsEntwicklung` entstehen, der merkt, wenn die Partie zu Ende ist. In diesem Falle soll der `Beobachter` das Spielergebnis feststellen. Sieger ist, wer zuerst vier Steine seiner Farbe in einer Linie (waagrecht, senkrecht oder diagonal) erreicht. Die weiteren Vorgaben sind dem folgenden Test zu entnehmen.

```
@Test
public void testeSpielKontrolle() {
    StellungsEntwicklung se = new StellungsEntwicklung(ROT);
    SpielKontrolle sk = new SpielKontrolle();
    se.anmelden(sk);
    while (!sk.istBeendet()) {
        Spalte s = se.stellung().zufaelligeSpalte();
        se.einwerfenBei(s);
    }
    assertNotSame(LEER, sk.sieger());
    assertEquals(sk.istUnentschieden(), sk.sieger() == null);
    if (sk.istUnentschieden()) return;
    assertEquals("" + sk.sieger() + " hat gewonnen", "" + sk);
}
```

Implementieren Sie die Klasse `SpielKontrolle`. Für die Überprüfung, ob durch den letzten Zug vier Steine der gleichen Farbe in einer Linie erreicht wurden, ist die in **Aufgabe 1** neu erstellte Methode hilfreich.

Aufgabe 6: (21 Punkte)

Vervollständigen Sie die nachstehenden Aussagen über Klassen und Methoden der Anlage.

a) An zwei Stellen der Anlage wird die Klasse `InternalError` verwendet. Korrekt ist dies in der Methode namens _____. An der zweiten Stelle sollte stattdessen eine _____ geworfen werden.

b) Die Methode _____ der Klasse `Stellung` verletzt die Regel „_____“.
Dadurch wird die Klasse `Stellung` _____.
Ein Objekt `stellung` der Klasse könnte sogar korrupt werden, etwa durch die Anweisung _____.

c) In der Anlage wird die Annotation _____ benutzt. Zusätzlich sollte damit mindestens noch die Methode _____ der Klasse _____ und die Methode _____ der Klasse _____ gekennzeichnet werden. In Frage kommt dafür auch noch die Methode _____ der Klasse _____.

d) Ein wichtiges Ziel in der Software-Entwicklung ist, die _____ so locker wie möglich zu gestalten. Eine gute Methode hierzu ist das in der Anlage vorkommende _____. Das zugehörige Prinzip wird auch in Swing genutzt. Man spricht dort aber nicht von einem _____, sondern von einem _____.

e) Beim _____ von Methoden wird vom Compiler anhand der _____ entschieden, welche von mehreren Funktionen gleichen Namens zu verwenden ist. In der Anlage kommt diese Art der _____ _____ bei Methoden namens _____ und _____ vor. Hingegen findet beim _____ _____ von Methoden die Entscheidung für eine der Funktionen _____ _____ statt.

f) Beim Aufruf `new Stellung(ROT, _____)` wird eine `ArrayIndexOutOfBoundsException` geworfen. Es fehlt nämlich die Überprüfung der _____. Das gilt auch für den zweiten _____ der Klasse. Dort kann das leicht behoben werden, indem als erste Anweisung ergänzt wird:

```
if (_____)
    _____;
```