FACH:	**Modellierung**	**NAME:**	
DATUM:	Dienstag, 27. Mai 2008		
ZEIT:	11:30 – 13:00	**SEMESTER:**	
PRÜFER:	Erben		
		STUDIENGANG:	☐ **D**iplom Mathematik ☐ **B**achelor Mathematik

ANLAGEN: **HILFSMITTEL:**	• Quelltext einiger Klassen (**bereits vorab ausgegeben**). • Handschriftliche Ergänzungen auf der vorab ausgegebenen Anlage
UNBEDINGT BEACHTEN:	• Tragen Sie **jetzt gleich** auf dieser Seite Ihren **Name** und ihr **Semester** ein und kreuzen Sie Ihren **Studiengang** an. Bearbeitungen ohne diese vorherige Kennzeichnung sind ungültig. • Führen Sie die Bearbeitung direkt auf diesen Aufgabenblättern in den dafür vorgesehenen Rahmen aus. Der dort vorhandene Platz sollte ausreichen. • Aus Gründen der Übersichtlichkeit sind Erläuterungen und Hinweise teilweise erst nach den auszufüllenden Feldern platziert.
HINWEISE ZU QUELLTEXT:	• Bei dem angegebenen Quellcode handelt es sich um Auszüge, welche sinnvoll ergänzt zu denken sind. Insbesondere trifft dies auf das Importieren benötigter Klassen in anderen Packages zu. • `Intervall` und `VereinigungVonIntervallen` sind zwei Klassen, welche den Anforderungen der (ersten) Studienarbeit genügen. Insbesondere sind beide Klassen kanonisch. • An vielen Stellen ist der Quelltext absichtlich schlecht. Insbesondere werden häufig nichtssagende Namen verwendet.
EMPFEHLUNGEN ZUR BEARBEITUNG:	• Lesen Sie die Aufgaben und Aufgabenteile bitte in der vorgegebenen Reihenfolge durch. An manchen Stellen ist dies für das Verständnis der Aufgabenstellung wichtig. • Einzelne Bearbeitungen können aber zurückgestellt werden. Die vollständige Lösung einer Aufgabe ist für die folgenden Aufgaben nicht erforderlich.

Aufgabe 1: (12 Punkte)

In dieser Aufgabe geht es um allgemeine Zusammenhänge. Es können generell mehrere oder auch gar keine Antwort zutreffen.

a) Welche der folgenden Aussagen sind richtig?

	ja	nein
Jede Klasse braucht einen öffentlichen Konstruktor		✓
Eine Klasse hat immer einen Default-Konstruktor. Wird er nicht selbst definiert, so wird er automatisch generiert.		✓
Methoden sind automatisch `public`, wenn sie nicht explizit als `private` oder `protected` deklariert werden.		✓
Daten sind automatisch `private`, wenn sie nicht explizit als `public` oder `protected` deklariert werden.		✓
Daten, die `final` sind, können in keiner Weise verändert werden.		✓

b) Welche der folgenden Schlussfolgerungen sind richtig?

	ja	nein
Die Klasse ist `final` ⇒ alle Daten der Klasse sind `final`		✓
Alle Daten sind `final` ⇒ die zugehörige Klasse ist `final`		✓
Die Klasse ist `final` ⇒ alle Methoden der Klasse sind `final`	✓	
Alle Methoden sind `final` ⇒ die zugehörige Klasse ist `final`		✓
Alle Methoden und alle Daten sind `final` ⇒ die Klasse ist `final`		✓

c) Welche der folgenden Schlussfolgerungen sind richtig?

	ja	nein
Einige Daten sind `final` ⇒ diese Daten sind `private`		✓
Einige Daten sind `private` ⇒ diese Daten sind `final`		✓
Einige Methoden sind `final` ⇒ diese Methoden sind `private`		✓
Einige Methoden sind `private` ⇒ diese Methoden sind `final`	✓	
Die Klasse ist `final` ⇒ Objekte der Klasse sind unveränderbar		✓

d) Welche der folgenden Empfehlungen sollten möglichst befolgt werden?

	ja	nein
Alle Daten einer Klasse sollten `private` sein.	✓	
Ist eine Klasse für Vererbung vorgesehen, dann sollten die Daten als `protected` deklariert werden.		✓
Wenn nur statische Methoden vorhanden sind, sollte der Default-Konstruktor `private` sein.	✓	
Methoden mit Rückgabewert sollten das Objekt nicht verändern.	✓	
Berechnungen sollten mit `float`-Werten vorgenommen werden.		✓

Aufgabe 2: (12 Punkte)

In dieser Aufgabe geht es um Design-Entscheidungen hinsichtlich Aufbau und Zusammenspiel der Klassen `Intervall` und `VereinigungVonIntervallen`.

a) Die Implementierung der Klasse `VereinigungVonIntervallen` geschieht über eine `ArrayList<Intervall>`. Vererbung wäre dabei eine schlechte Idee. Überhaupt gilt in den allermeisten Fällen die allgemeine Design-Faustregel

Komposition ist besser als Vererbung

Falsch eingesetzte Vererbung erkennt man fast immer mit dem Liskovschen Substitutionsprinzip. Wie lautet dieses Prinzip?

Ein Objekt einer abgeleiteten Klasse kann stets und überall anstelle eines Objektes der Basisklasse verwendet werden.

b) Begründen Sie, dass `VereinigungVonIntervallen` nicht von `ArrayList<Intervall>` abgeleitet werden sollte.

Bei VereinigungVonIntervallen sind fast alle von ArrayList geerbten Methoden (z.B. add) falsch. Sie müssten damit überschrieben werden. Andere (z.B. remove) sind sogar unerwünscht.

c) Begründen Sie, dass `VereinigungVonIntervallen` auch nicht von `Intervall` abgeleitet werden sollte.

Eine VereinigungVonIntervallen ist kein Intervall. (Beispielsweise koennte die Methode schnittMit(Intervall) gar nicht korrekt implementiert werden.)

Aufgabe 3: (18 Punkte)

Die Verwendung der Klassen `Intervall` und `VereinigungVonIntervallen` soll verglichen werden.

a) Betrachten Sie die Methode einer Klasse `A2`

```
public static Intervall tuWasMit(Intervall x,
                                 Intervall y) {
    Intervall z = new Intervall(3., 7.);
    y = z;
    return x.vereinigungMit(y);
}
```

Das Intervall `a` habe den (abstrakten) Wert `[1,5]`, das Intervall `b` den Wert `[0,4]` und `c` den Wert `[0,1]`. Welchen Wert haben diese Mengen nach dem Aufruf

```
c = A2.tuWasMit(a, b);
```

a	[1,5]	b	[0,4]	c	[1,7]

b) Ebenfalls in der Klasse `A2` befinde sich die Methode

```
public static VereinigungVonIntervallen
                  tuWasMit(VereinigungVonIntervallen x,
                           VereinigungVonIntervallen y) {
    Intervall z = new Intervall(3., 7.);
    y = new VereinigungVonIntervallen(z);
    x.vereinigenMit(y);
    return y;
}
```

Die Vereinigung von Intervallen `a` habe den (abstrakten) Wert `[1,5]`, `b` den Wert `[0,4]` und `c` den Wert `[0,1]`. Welchen Wert haben diese Mengen nach dem Aufruf

```
c = A2.tuWasMit(a, b);
```

a	[1,7]	b	[0,4]	c	[3,7]

Aufgabe 4: (24 Punkte)

In dieser Aufgabe geht es um die Gleichheit von Objekten. Eine `JUnit`-Testklasse enthält die Methode

```
public void testeGleichheit() {
    Klasse x = new Klasse();
    if (!x.equals(x)) fail("#1");
    Klasse y = new Klasse();
    if (!x.equals(y)) fail("#2");
    if (!y.equals(x)) fail("#3");
    if (x.hashCode() != y.hashCode()) fail("#4");
    Object z = y;
    if (x.hashCode() != z.hashCode()) fail("#5");
    if (!x.equals(z)) fail("#6");
    if (!z.equals(x)) fail("#7");
}
```

Dabei werde `Klasse` (an allen vier Stellen) durch jeweils eine Klasse der **Anlage** ersetzt. Die Testergebnisse sind in nachstehender Tabelle festgehalten. Vervollständigen Sie diese Tabelle.

`Klasse`	Ergebnis	Grund für `fail`
Leerer	**#2**	`equals` nicht überschrieben
Polyticker	**OK**	**---**
Mackler	**#1**	**equals falsch (!= statt ==)**
HochschulLeerer	**#6**	`equals` überladen statt überschrieben
Drecksler	**#1**	**erbt falsches equals von Mackler**
Staticker	**#2**	Default verändert sich
Maler	**#5**	**hashCode verändert sich**

Hinweis: In die zweite Spalte (Ergebnis) gehört

- OK, wenn die jeweilige Klasse den Test besteht. Die dritte Spalte bleibt dann leer.
- Ansonsten das Argument des ersten `fail`, welches die Klasse auslöst. Der restliche Testcode braucht dann nicht mehr überprüft zu werden.

In die dritte Spalte (Grund für `fail`) gehört eine knappe Begründung für das Auslösen des `fail`. Ausschweifende Erklärungen werden nicht akzeptiert.

Aufgabe 5: (24 Punkte)

In dieser Aufgabe geht es um die Verwendung von Konstruktoren einer Klasse in einem **anderen** `package`. Kreuzen Sie jeweils an, ob der angegebene Konstruktur-Aufruf zu einem Compile-Fehler (inkl. Warnung) oder zu einer `RuntimeException` führt oder aber korrekt ist.

a) Aufrufe ohne Argumente:

Konstruktor-Aufruf	Comp.-F.	Exception	korrekt
`new Mueller()`	✓		
`new Turner()`		✓	
`new Dreher()`		✓	
`new AbstrakterMaler()`	✓		
`new Maler()`			✓
`new Drecksler()`			✓

b) Aufrufe mit einem Argument:

Konstruktor-Aufruf	Comp.-F.	Exception	korrekt
`new Mueller(0)`		✓	
`new Mueller(0.)`			✓
`new Mueller(1/2)`		✓	
`new Macheniker(0)`			✓
`new Macheniker(0.)`	✓		
`new Mackler(0)`	✓		
`new Polyticker(0)`			✓
`new Polyticker(-1)`		✓	