Prof. Dr. Stefan Brass
M.Sc. Mario Wenzel

Martin-Luther-Universität Halle-Wittenberg
Institut für Informatik
Wintersemester 2018/19

Objektorientierte Programmierung: Hausaufgabenblatt 11

Abgabe: 21.01.2019, 11:00

Das Ziel dieses Übungsblattes ist es, dass Sie ihre Kenntnisse zu Klassenhierarchien, sowie dem Unterschied zwischen dynamischen und statischen Zugriffen festigen.
Diese Übungsserie finden Sie im Stud.IP auf der Seite der Veranstaltung unter Lernobjekte → Kurs in ILIAS → Hausaufgabenblatt 11.

Hausaufgabe 1: (6 Punkte)

Bitte beachten Sie: die Einsendung der Aufgabe erfolgt über die ILIAS-Plattform.

Gegeben sind folgende 4 Klassenhierarchien

`//1:`
`class A {}`
`class B extends A {}`
`class C extends B {}`
`class D extends A {}`
	`//2:`
	`class A {}`
	`class B extends A {}`
	`class C extends A {}`
	`class D extends A {}`

`//3:`
`class A {}`
`class B extends A {}`
`class C extends B {}`
`class D extends C {}`
	`//4:`
	`class A {}`
	`class B extends A {}`
	`class C {}`
	`class D extends C {}`

In A-F sind jeweils 2-3 Deklarationen gegeben. Eine auskommentierte Deklarationen würde zu einem Compilerfehler (statischer Typfehler) führen. Die anderen Deklarationen sind erlaubt. Ordnen Sie zu, welche der Deklarationsgruppen in welcher der Klassenhierarchien möglich ist (und entsprechend zum entsprechenden Fehler führen würde). Die mehrfache Zuordnung ist erlaubt.

A:

```
B x = new C();
A y = new B();
D z = new D();
```

B:

```
A x = new B();
A y = new D();
A z = new C();
```

C:

```
A f() { return new B(); }
C x = new D();
```

D:

```
// B x = new C();
A f(B p) { return p; }
```

E:

```
A x = new D();
// C y = new D();
```

F:

```
A f(B p) { return p; }
// B g() { return new D(); }
```


Hausaufgabe 2: (3 Punkte)

Bitte beachten Sie: die Einsendung der Aufgabe erfolgt über die ILIAS-Plattform.

Gegeben sei die Klasse `I` wie folgt:

```
class I {
  public static int s = 0;
  public int d;
  I()      { d = 3*s; }
  I(int x) { s++; d = x; }
  void swp(I o) { int tmp = o.d; o.d = d; d = tmp; }
}
```

Eine andere Methode führt die folgenden Anweisungen (und keine weiteren) in dieser Reihenfolge aus. Geben Sie den Wert von `I.s`, sowie die Werte des Attributs `d` der Instanzen `i1` bis `i5` nach Ausführung der Anweisungen an.

```
I i1 = new I();
I i2 = new I(5);
I i3 = new I(I.s);
I i4 = new I();
I i5 = new I(i2.d/2);
i5.swp(i2);
i3.swp(i3);
i1.swp(i2);
// -> Hier Auswerten
```

Ausdruck	Wert
I.s	
i1.d	
i2.d	
i3.d	
i4.d	
i5.d	

Hausaufgabe 3: (4 Punkte)

Gegeben Sei die folgende Klassenhierarchie der Klassen `A` und `B` mit 4 markierten Stellen (S1 bis S4), an denen Anweisungen eingefügt werden können. Außerdem sind Ihnen 8 mögliche Anweisungen gegeben. Geben Sie an (ankreuzen), welche der Anweisungen an welcher der 4 Stellen nach den bekannten Zugriffsregeln erlaubt ist. Die Semantik der Aufrufe brauchen Sie dabei nicht zu beachten (einige führen zu endlos-Rekursionen).

```
class A {
  public  static int x;
  private static int y;
  public  int a;
  private int b;
  static void f() { /* S1 */ }
  void g() { /* S2 */ }
}
class B extends A {
  static void h() { /* S3 */ }
  void i() { /* S4 */ }
}
```

Anweisung	S1	S2	S3	S4
x = 4;				
y = 4;				
a = 4;				
b = 4;				
f();				
g();				
h();				
i();				

Hausaufgabe 4: (6 Punkte)

Bitte beachten Sie: die Einsendung der Aufgabe erfolgt nur im YAPEX.
Open exercises via code → Freigabecode: `5d0fap0j8d1j-3595` (Unveränderliche Rechtecke)

Die Änderung von Attributen eines Objekts ist eine häufige Fehlerquelle, da sich durch weitergegebene Objektreferenzen die Änderungen eines Attributs auf das Verhalten anderer Programmbestandteile auswirken kann. Die Änderung des beobachtbaren Zustands oder Verhaltens eines Objekts oder des Systems nennt man "Nebenwirkung" (Englisch "side effect" manchmal als "Seiteneffekt" rückübersetzt). Natürlich ist die Ausgabe eines Programmergebnisses eine notwendige und gewünschte Nebenwirkung, sonst könnte man nie das Ergebnis einer Berechnung auf der Konsole lesen. In einer Nebenwirkungsfreien oder -armen Programmiermethode beschränken sich die Nebenwirkungen auf Ein- und Ausgabe innerhalb weniger Programmbestandteile. Anstatt ein Objekt zu verändern, werden immer neue Objekte mit den gewünschten Eigenschaften konstruiert und zurückgegeben. Dieser Programmierstil ist auch in Java möglich, wenn auch in komplexeren Fällen umständlich.

Schreiben Sie eine Klasse Rechteck mit einem Konstruktor, welcher zwei double-Parameter (breite und höhe) erwartet. Implementieren Sie außerdem die folgenden Methoden:

Name	Argumente	Rückgabewert	Kommentar
width	keine	double	Breite des Rechtecks
height	keine	double	Höhe des Rechtecks
scale	double	Rechteck	Neues um diesen Faktor skaliertes Rechteck
rotate	keine	Rechteck	Neues um 90 Grad gedrehtes Rechteck
area	keine	double	Fläche des Rechtecks
equals	Rechteck	boolean	die Seitenlängen stimmen überein

Achten Sie darauf, dass Ihre Klasse nur als final markierte Attribute enthält. Wenn Sie außerdem eine `toString`-Methode erstellen, erhalten Sie schöneres Feedback.
Ihre Klasse wird automatisch von YAPEX getestet. Sie brauchen keine Main-Funktion zu schreiben! Dabei prüfen wir unter Anderem folgende Identitäten für das Rechteck `a`, die sich logisch ergeben:

```
a.equals(a)
a.scale(1).equals(a)
a.scale(0).equals(new Rechteck(0,0))
a.scale(x).scale(1.0/x).equals(a)
a.scale(n).area() == a.area()*n*n
a.rotate().rotate().equals(a)
a.area() == a.rotate().area()
```

Instanzmethoden, die wieder eine Instanz des selben Typs (oder Interfaces) zurückgeben, können, wie Sie im Beispiel sehen, beliebig aneinander gehangen werden. Diese Programmierweise nennt man "Method Chaining".