

Blatt 3: Vererbung

Diskussionsteil

a) ☐ ★

- Erstellen Sie eine Klasse Student, die von der Person-Klasse (`Person.java`) erbt.
- Fügen Sie dieser Klasse ein Feld studentID (int) mit Getter und Setter hinzu.
- Die neue Klasse soll einen Konstruktor mit allen Feldern erhalten:

```
public Student(String firstName, String lastName, int studentID){...
```

- **Überschreiben** Sie die Methode greet mit einem (leicht erkennbar) anderen Text. Führen Sie dann folgenden Code aus und erklären Sie, warum welche Ausgabe auf der Konsole erscheint.

```
1 Person person = new Person("Paul", "Person");
2 Student student = new Student("Sam", "Student");
3 person.greet(student);
4 student.greet(person);
```

b) ☐ ★

Hinweis



In java erben alle Klassen von der vorgegebenen Object-Klasse. Darin sind bereits einige Methoden vorgegeben und implementiert. In den folgenden Aufgaben sollen Sie diese Methoden überschreiben und analysieren. Eclipse kann automatisch anzeigen, welche Methoden von Superklassen überschrieben werden können. Drücken Sie dazu im Body der zutreffenden Klasse `Strg` + `Leer`.


Finden Sie einen Weg, mit dem die Klasse Person (zu finden in `Person.java`) leserlicher auf der Konsole ausgegeben wird. Ändern Sie die greet-Methode anschließend so, dass diese Mechanik ausgenutzt wird.

Ist:

```
1 Person person = new Person("Donald", "Duck");
2 System.out.println(person); // --> Person@1b6d3586
```

Soll:

```
1 Person person = new Person("Donald", "Duck");
2 System.out.println(person); // --> Donald Duck
```

- c)  Der ==-Operator ist in manchen Fällen nicht geeignet, um Gleichheit von Objekten zu kontrollieren. Geben Sie für jedes der folgenden Beispiele aus, warum der Vergleich den jeweiligen Wert ausgibt.

Operation	Ausgabe	Grund
"String" == "Strin" + "g"		
"String" == "Strin" + new String("g")		
0.3 == 0.1 + 0.1 + 0.1		

Überlegen Sie sich nun, wie Sie Objekte stattdessen auf Gleichheit überprüfen können (befragen Sie dazu auch gerne Ihre Mitstudenten oder das Internet). Gewappnet mit neuem Wissen, versuchen Sie nun folgendes Verhalten in die Student-Klasse einzubauen:

Zwei Student-Objekte sollen als gleich erachtet werden, wenn ihre studentID gleich ist. Ihre Namen sollen dabei keine Rolle spielen.

Hinweis



Eclipse kann die notwendigen Methoden vollständig automatisch generieren.

Testen Sie Ihre Implementierung, indem Sie zwei Studenten mit unterschiedlichem Namen, aber selber ID in ein Set speichern und dieses danach ausgeben. Erklären Sie, welche Methoden Sie überschrieben haben und warum.

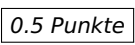
```
1 Set<Student> students = new HashSet<>();
2 students.add(new Student("Donald", "Duck", 42));
3 students.add(new Student("Mickey", "Mouse", 42));
4 System.out.println(students); // --> sollte nur "Donald Duck" ausgeben
```

Übungsteil (selbstständig zu lösen)

Aufgabe 1 (Statische Elemente)

[2 Punkte]

Im folgenden Beispiel sollen Sie eine Klasse Circle implementieren, die einen Kreis repräsentiert. Setzen Sie dabei Folgendes um:

- a)  Die Klasse Circle soll entweder durch Angabe des Radius oder des Durchmessers angelegt werden können, darf aber nicht ohne Angabe erzeugt werden. Eine erste intuitive, aber falsche Umsetzung könnte wie folgt aussehen:

```
1 public class Circle {
2     public Circle(double radius) {
3         ...
4     }
5
6     public Circle(double diameter) {
7         ...
8     }
9 }
```

Warum funktioniert dies nicht? Finden Sie eine Variante, um die Erzeugung sowohl per Radius als auch per Durchmesser zu ermöglichen.

- b) 0.5 Punkte Erzeugen Sie 2 Methoden, um die Fläche und den Umfang des Kreises zu berechnen. Verwenden Sie dazu für Pi eine eigens definierte Konstante, die nur 2 Nachkommastellen umfasst.
- c) 0.5 Punkte Ändern Sie die Implementierung so ab, dass die Methoden zum Berechnen der Fläche und des Umfangs auch für andere Klassen verfügbar sind, ohne dabei ein Circle-Objekt anlegen zu müssen.
- d) 0.5 Punkte Erstellen Sie eine Methode `printInfo()`, die den Radius, die Fläche und den Umfang des aktuellen Kreis-Objekts ausgibt. Erzeugen Sie schlussendlich noch eine passende `main`-Methode, um Ihre Implementierung zu testen. Testen Sie insbesondere auch die Verwendung der obigen Methoden 'von außen'.

Aufgabe 2 (Interfaces, Vererbung und Abstrakte Klassen) [5 Punkte]

In dieser Aufgabe geht es darum, verschiedene Checker für Credentials (siehe `Credential.java`^{OLAT}) zu implementieren, die beliebig kombinierbar sein sollen. Beispielsweise testet ein Checker, ob eine gewisse Mindestlänge für Username/Passwort vorhanden ist, und ein anderer ob keine nicht erlaubten Usernamen verwendet werden. Um die Checker einheitlich zu erstellen, wird das Interface `ICredentialChecker` (`ICredentialChecker.java`^{OLAT}) verwendet, das nur die Methode `check(Credential c)` definiert. Diese Methode erwartet ein `Credential` als Eingabe und gibt `true` zurück, wenn das Credential ok ist, d.h. den Check besteht, andernfalls `false`.

- a) Erstellen Sie folgende Checker, die das Interface `ICredentialChecker` implementieren:
 - 1 Punkt `MinimumLengthChecker`: überprüft, ob der Username und das Passwort je eine definierte Mindestlänge haben. Übergeben Sie im Konstruktor je eine Mindestlänge für Username und Passwort (d.h. die Längen können variieren).
 - 1 Punkt `CharacterClassChecker`: überprüft, ob der Username ausschließlich aus den übergebenen Zeichen besteht, und ob das Passwort mindestens n Zeichen eines Spezial-Zeichen-satzes enthält (es müssen dabei nicht unterschiedliche Zeichen sein, d.h. wenn n -Mal dasselbe Zeichen aus dem Zeichensatz vorkommt genügt dies). Verwenden Sie zum Übergeben der Zeichen im Konstruktor Character-Arrays (`char []`), siehe auch `CheckerTest.java`^{OLAT} zur Verwendung). In Summe müssen Sie also 3 Variablen übergeben: die erlaubten Zeichen für Usernamen, den Spezialzeichensatz für Passwörter und das Mindestvorkommen n dieses Zeichensatzes im Passwort.
 - 1 Punkt `ReservedUsernamesChecker`: überprüft, ob der Username keines der übergebenen reservierten Wörter ist. Verwenden Sie zum Übergeben eine `List`, die die Einträge "admin" und "superadmin" enthält.
- b) 1 Punkt Implementieren Sie zusätzlich die Klassen `And` und `Or` (beide implementieren das Interface `ICredentialChecker`, um die obigen Checker zu verknüpfen):
 - `And`: überprüft, ob zwei Bedingungen gleichzeitig erfüllt sind. Dazu werden im Konstruktor 2 Objekte vom Typ `ICredentialChecker` übergeben, ein linker und ein rechter Operand. Wird `check` aufgerufen, delegiert ein `And` Objekt diese Aufgabe einfach an seine beiden Operanden weiter und verknüpft das Ergebnis mit einem logischen Und.
 - `Or`: gleich wie `And`, aber das Ergebnis wird mit einem logischen Oder verknüpft.

- c) 1 Punkt Testen Sie Ihre Implementierung, indem Sie die bereitgestellte Klasse `CheckerTest.java`^{OLAT} entsprechend vervollständigen. Beachten Sie hier zusätzlich, dass das Programm nicht abstürzen sollte, wenn `null` für Username und/oder Passwort übergeben wird.

Aufgabe 3 (Statischer und Dynamischer Typ)

[3 Punkte]

Für diese Aufgabe werden die 3 Klassen A, B und C (`A.java`^{OLAT} , `B.java`^{OLAT} , `C.java`^{OLAT}) verwendet, die in folgender Vererbungslinie stehen: C erbt von B, und B erbt von A. Sehen Sie sich die Implementierungen dieser Klassen sowie die Test-Klasse an (`Test.java`^{OLAT}) und beantworten Sie folgende Fragen:

- a) 2 Punkte Welche Ausgabe erzeugt die Klasse Test und warum? Begründen Sie, warum was ausgegeben wird.

Hinweis



Nutzen Sie die Möglichkeit, in Eclipse mehrere Dateien gleichzeitig anzeigen zu lassen (mittels Drag&Drop), um einen Überblick über die vier relevanten Klassen zu haben.

- b) 1 Punkt Erklären Sie anhand dieses Beispiels den Unterschied zwischen dynamischem und statischem Binden.

Wichtig: Laden Sie bitte Ihre Lösung (.txt, .java oder .pdf) in OLAT hoch und geben Sie mittels der Ankreuzliste auch unbedingt an, welche Aufgaben Sie gelöst haben. Die Deadline dafür läuft am Vortag des Proseminars um 23:59 (Mitternacht) ab.