Fragen, Antworten, Kommentare zur aktuellen Vorlesung

Frage: Wenn ich krank werde oder in Quarantäne muss, fliege ich dann aus dem Praktikum?

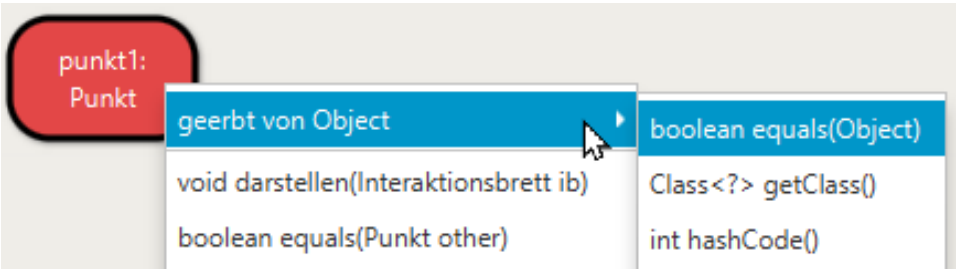
Nein, wenn Sie sich beim Praktikumsleiter oder/und mir, kurz mit einer informellen E-Mail entschuldigen. Planen Sie für sich selbst und dann mit uns, wie Sie die Nacharbeiten ausführen wollen. Denken Sie daran, dass per Mail an mich auch Extra-Online-Termine planbar sind. Sollten Sie häufiger oder längerfristiger ausfallen, melden Sie sich frühzeitig bei mir, um zu schauen, ob Sie trotzdem das Praktikum bestehen können, bei dem eine regelmäßige Anwesenheit ein Bestehenskriterium ist.

Frage: Ich habe die equals()-Methode für die Klasse Punkt geschrieben und jetzt werden mir bei einem Punkt-Objekt zwei equals()-Methoden angeboten.

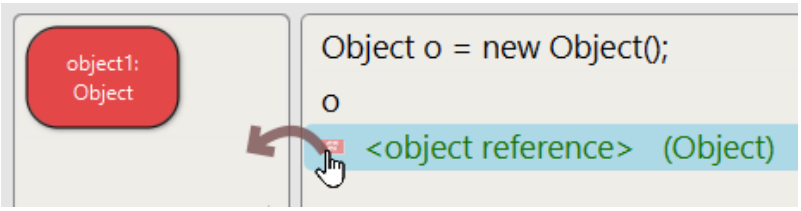
Das stimmt, wenn eine neue Klasse angelegt und ein Objekt erstellt wird, steht eine equals(Object)-Methode sofort zur Verfügung. Dies wird in der Veranstaltung bewusst zunächst ignoriert und später erklärt. Ihre jetzt geschriebene equals()-Methode wird dann später Teil der vollständigen Lösung. Um dies zu verdeutlichen erfolgt jetzt ein kleiner Vorgriff, der von Personen am absoluten Anfang der Programmierausbildung nicht verstanden werden muss. Die korrekte Zwischenlösung für das Praktikum sieht wie folgt aus.

```
// Punkt
public boolean equals(Punkt other) {
    if (other == null) {
        return false;
    }
    return this.x == other.getX() && this.y == other.getY();
}
```

Nach der Objekterstellung werden zwei equals()-Methoden zur Nutzung angeboten.



Die Methode equals(Object) und einige andere sind bereits da, da jede Klasse automatisch von der Klasse Object „erbt“, was grob bedeutet, dass die Funktionalität dieser Klasse in allen Klassen zur Verfügung steht. Die Klasse Object ist damit die Basis aller Klassen. Es handelt sich dabei aber um eine normale, in Java bereits vorhandene Klasse. Die Erstellung von Objekten ist mit new Object() problemlos möglich.



Die bereits angebotene Methode equals(Object) kann natürlich aufgerufen werden, liefert aber nicht das gewünschte Ergebnis. Inhaltlich wird einfach eine Prüfung mit „==“ auf Identität durchgeführt.

Später wird die Methode equals() für beliebige Objekte zugelassen, was an dem Parameter vom Typ Object erkennbar ist. In der üblichen Programmierung muss dann zunächst geprüft werden, ob der Parameter einen sinnvollen Typen hat und dann folgt die uns bekannte inhaltliche Prüfung. Bald wird die Lösung dann wie folgt aussehen, rote Teile sind ergänzt bzw. geändert. Falls Sie die oberen beiden if zusammenfassen wollten, wäre das sinnvoll.

```
@Override
public boolean equals(Object o) {
    if (o == null) {
      return false;
    }
    if (this.getClass() != o.getClass()) {
      return false;
    }
    Punkt other = (Punkt) o;
    return this.x == other.getX() && this.y == other.getY();
}
```

Mit Aufgabe 26 haben Sie die erste komplexere geschlossene Aufgabe vor sich, die zwar in Teilschritte zerlegt ist, trotzdem einigen Planungsaufwand beinhaltet. Bei unerfahrenen Studierenden gehe ich grob von einer Bearbeitungszeit von 3 Stunden aus. Hier noch ergänzend einige Planungsschritte.

1. Lesen Sie sich die Aufgabe einmal vollständig durch. Verstehen Sie das Ziel der Aufgabe. Wenn Sie sich unsicher sind, fragen Sie (mich, Praktikumsleiter, Mitstudierende). Als Spezialfall steht bereits eine ausführbare exe-Datei zur Verfügung, die das gewünschte Programmierziel verdeutlicht. Führen Sie mehrere Experimente mit dem Programm durch.
2. Überlegen Sie welche Klassen Sie brauchen (steht eigentlich im Aufgabentext) und wozu die einzelnen Klassen da sein sollen.
3. Beginnen Sie mit der Implementierung einer Klasse. die möglichst keine der anderen Klassen nutzt, die Sie noch programmieren müssen.
4. Entwickeln Sie gerade komplexere Methoden inkrementell. D. h. Sie programmieren erstmal einen kleinen Schritt, z. B. eine null-Behandlung, probieren es dann aus und erst, wenn es wie erwartet läuft, ergänzen Sie weitere Funktionalität. Dieses Vorgehen, mit ganz etwas größeren Schritten, wird immer Ihre Arbeitsweise sein.
5. Treten Fehler auf, versuchen Sie herauszufinden, wo der Fehler erkannt wird (z. B. Ort der NullPointerException) und überlegen Sie dann, an welcher Stelle das Problem erzeugt wird (typischerweise eine andere Stelle im Programm). Gehen Sie zunächst von einem Flüchtigkeitsfehler aus und prüfen Sie die Programmstücke darauf. Erinnern Sie sich, dass es den Debugger gibt, der ein elementares Werkzeug der SW-Entwicklung ist und von Ihnen beherrscht werden muss. Natürlich kann man bei einfachen Programmen auch Textausgaben auf dem Bildschirm erzeugen, mit denen verschiedene Werte ausgegeben werden. Es kann auch sinnvoll sein, sich kleine Hilfsprogramme zu schreiben, um z. B. nicht für jeden Test erst von Hand mehrere Objekte in BlueJ erstellen zu müssen.
6. Vor jeder Änderung, die Sie dann testen werden, überlegen Sie warum Sie jetzt meinen, warum dies Ihr Problem löst. Falls Sie jetzt in einen Try-and-Error-Ansatz verfallen und „mal sehen was passiert, wenn ich aus dem && ein || mache“, wissen Sie, dass Sie auf dem falschen Weg sind und nochmals grundsätzlich über die Aufgabenstellung nachdenken sollten, dann Ihre bisherige Lösung wegschmeißen und neu anfangen.