Dr. Annette Bieniusa
Mathias Weber, M. Sc.
Peter Zeller, M. Sc.

**TU Kaiserslautern**
**Fachbereich Informatik**
**AG Softwaretechnik**

# Übungsblatt 10: Software-Entwicklung 1 (WS 2017/18)

Ausgabe: 08.01.18
Abgabe: 15.01.18

## Aufgabe 1 Equals, Hashcode, Comparator (12 Punkte)

a) Schreiben Sie eine Klasse `Ort` mit einem Konstruktor `Ort(int postleitzahl, String name)` und entsprechenden Attributen und `get`-Methoden (`getPostleitzahl` und `getName`).

b) Schreiben Sie eine Klasse `Adresse` mit einem Konstruktor `Adresse(Ort ort, String strasse, String hausNummer)` und entsprechenden Attributen und `get`-Methoden (`getOrt`, `getStrasse` und `getHausNummer`).

c) Wir wollen nun eine Map erstellen, in der für verschiedene Adressen die Anzahl der Bewohner gespeichert ist.

```
Map<Adresse, Integer> anzahlBewohner = new TreeMap<>();
Ort kaiserslautern = new Ort(67663, "Kaiserslautern");
Adresse addr = new Adresse(kaiserslautern, "Gottlieb-Daimler-Str.", "34");
anzahlBewohner.put(addr, 23);
```

Nun erhalten wir beim Aufruf von `put` leider den Fehler `java.lang.ClassCastException: Adresse cannot be cast to java.lang.Comparable`. Dies liegt daran, dass Adressen nicht miteinander vergleichbar sind (sie implementieren nicht das Interface `Comparable`).

Wir können diesen Fehler beheben, indem wir beim Erstellen der `TreeMap` einen `Comparator<Adresse>` angeben:

```
Map<Adresse, Integer> anzahlBewohner = new TreeMap<>(new AdressVergleicher());
```

Schreiben Sie geeignete Klasse `AdressVergleicher`, welche das Interface `Comparator<Adresse>` implementiert.

d) Statt einer `TreeMap` können wir auch eine `HashMap` verwenden. Dabei tritt jedoch ein anderes Problem auf:

```
Map<Adresse, Integer> anzahlBewohner = new HashMap<>();
Ort kaiserslautern = new Ort(67663, "Kaiserslautern");
Adresse addr = new Adresse(kaiserslautern, "Gottlieb-Daimler-Str.", "34");
anzahlBewohner.put(addr, 23);
// Cast auf Integer ist hier nötig, weil Overloading
// wegen Autoboxing nicht eindeutig ist
assertEquals((Integer) 23, anzahlBewohner.get(addr));

Ort kaiserslautern2 = new Ort(67663, "Kaiserslautern");
Adresse addr2 = new Adresse(kaiserslautern2, "Gottlieb-Daimler-Str.", "34");
assertEquals((Integer) 23, anzahlBewohner.get(addr2));
```

Wenn wir ein neues Adresse-Objekt mit den gleichen Werten erstellen, können wir damit nicht den passenden Eintrag in der `Map` finden.

Schreiben Sie eine geeignete `equals`- und `hashCode`-Methoden für die Klassen `Adresse` und `Ort` um dieses Problem zu beheben.

## Aufgabe 2 Grafikeditor (12 Punkte)

In dieser Aufgabe sollen Sie den Grafikeditor weiterentwickeln, den Sie als `Grafikeditor.zip` herunterladen können.

a) Übersetzen Sie die vorgegebene Klasse `Main` und führen Sie das Programm dann aus. Sie können dann mit dem Grafikeditor Linien zeichnen. Falls Sie Probleme mit dem Ausführen haben, kontaktieren Sie bitte Ihren Tutor.

   Machen Sie sich mit dem Code in den Dateien `Main`, `ZeichenBlatt`, `Punkt`, `Figur`, `Linie`, `ZeichenTool`, `ZeichenToolLinie` und `ZeichenToolVerschieben` vertraut.

   Die Klasse `Gui` stellt die grafische Benutzeroberfläche bereit und muss in dieser Aufgabe nicht verstanden oder angepasst werden.

b) Erweitern Sie den Editor um eine Funktion, um Kreise zu zeichnen. Dazu sind die folgenden Schritte notwendig:

   - Erstellen Sie eine neue Unterklasse von `Figur` namens `Kreis`.
   - Erstellen Sie eine neue Unterklasse von `ZeichenTool` für das Zeichnen von Kreisen.
   - Passen Sie die `main`-Methode in der Klasse `Main` an, indem Sie dort ihr Tool zum Kreise zeichnen in die Liste der Tools eintragen.

c) Erweitern Sie den Editor noch um Rechtecke. Gehen Sie dabei analog zur vorherigen Teilaufgabe vor.

d) Die Methode `verschiebenUm` muss für jede Figur implementiert werden. Wie könnte man den Code umbauen, so dass es nur eine Implementierung der Methode gibt? Es ist ausreichend, wenn Sie Ihre Idee nur beschreiben und nicht implementieren.

## Aufgabe 3 Auswertung von Femto Programmen (7 Punkte)

Auf Übungsblatt 02 haben wir bereits kennen gelernt, wie wir Femto-Ausdrücke parsen können. Jedoch hat dieser Parser nur erkannt, ob es sich bei der Eingabe um ein gültiges Femto-Programm handelt. In dieser Aufgabe wollen wir die Auswertung von Femto-Programmen betrachten.

Laden Sie sich das Template (`femto_parser.zip`) für den Femto-Parser von der Vorlesungsseite herunter. Darin befindet sich eine erweiterte Implementierung des Femto-Parsers, der einen abstrakten Syntaxbaum erzeugt. Ausdrücke werden durch eine Subklasse von `Expression` repräsentiert, Anweisungen durch eine Subklasse von `Statement`. Die Auswertung von Ausdrücken ergibt einen Integer-Wert, die Auswertung von Anweisungen jedoch führt nur zu Seiteneffekten (wie zum Beispiel die Ausgabe eines Wertes auf die Konsole bei der print-Anweisung). Da in Femto Variablen in Ausdrücken verwendet werden können, muss zur Auswertung ein sogenannter Auswertungskontext verwendet werden. In diesem Kontext wird während der Auswertung gespeichert, was der aktuelle Wert der Variablen ist. In Java modellieren wir diesen Kontext als `Map<String, Integer>`.

Implementieren Sie die Methoden `int eval(Map<String, Integer> context)` für Ausdrücke und die Methode `void eval(Map<String, Integer> context)` für Anweisungen. Implementieren Sie anschließend die Methode `void execute()` in der Klasse `Program`, welche die Wertzuweisungen auswertet und anschließend die print-Anweisung ausführt und somit das Ergebnis der Auswertung des Femto-Programms auf die Konsole ausgibt.

## Aufgabe 4 HashMap mit offener Kollisionsauflösung (8 Punkte)

In dieser Aufgabe sollen Sie eine HashMap mit einer offenen Kollisionsauflösung implementieren. Dazu soll das "Linear Probing" Verfahren verwendet werden.

Schreiben Sie dazu eine Klasse `LPMap`, welche das `Map` Interface aus der Vorlesung implementiert und wie die HashMap aus der Vorlesung einen Konstruktor anbietet, der die Größe (`size`) der internen Tabelle angibt. Wir gehen in dieser Aufgabe davon aus, dass die angegebene Größe ausreichend ist und nie `size` oder mehr Einträge in der Map gespeichert werden.

Die interne Tabelle ist ein Array von `HashEntry`-Objekten. Allerdings haben die `HashEntry`-Objekte in dieser Implementierung keine Referenz auf einen nächsten Eintrag, sondern speichern nur den Schlüssel und Wert.

Gehen Sie dann zur Implementierung der einzelnen Methoden wie folgt vor:

a) Die `put`-Methode verwendet die `hash`-Funktion, um den passenden Index in der Tabelle zu finden. Falls dieser Index schon von einem anderen Schlüssel belegt ist, wird der nächst höhere Index betrachtet (wenn das Ende der Tabelle erreicht ist, wird am Anfang weiter gesucht).

   Da wir davon ausgehen, dass die Tabelle nie komplett gefüllt wird, finden wir immer irgendwann einen freien Index, an dem wir den Eintrag speichern können.

b) In der Methode `get` wird analog vorgegangen. Es wird zuerst der passende Index für den gesuchten Schlüssel mit der `hash`-Funktion berechnet. Falls an diesem Index kein Eintrag vorhanden ist, so wird **`null`** zurückgegeben. Falls ein Eintrag mit passendem Schlüssel vorhanden ist, wird der entsprechende Wert zurückgegeben. Wenn ein Eintrag mit anderem Schlüssel an der Stelle steht muss wie in der `put`-Methode der nächst höhere Index betrachtet werden.

c) (*Freiwillige Zusatzaufgabe*) Das Entfernen von Einträgen ist der komplizierteste Teil der Implementierung und daher nur eine freiwillige Zusatzaufgabe. Sie können Ihre Lösung zu dieser Zusatzaufgabe im Exclaim in der Zusatzübung "SE1WS17Z" abgeben. Beim Löschen ist zu beachten, dass keine ungewollten Lücken entstehen.

## Aufgabe 5 Zusatzaufgabe: Generische Map (0 Punkte)

*Sie können Ihre Lösung zu dieser Zusatzaufgabe im Exclaim in der Zusatzübung "SE1WS17Z" abgeben.*

Laden Sie sich die Map-Implementierung `HashMap.java` aus den Materialien zur Vorlesung herunter. Diese Implementierung verwaltet Schlüssel vom Typ **`int`** und Werte vom Typ `String`.

Passen Sie die Implementierung so an, dass die `HashMap` über die Typen von Schlüssel und Wert parametrisiert ist. Die `HashMap` soll dazu das folgende generische Interface implementieren:

```
interface Map<Key, Value> {
    // Liefert ein String unter einem Schlüssel key
    Value get(Key key);

    // Fügt einen String value unter einem Schlüssel key ein
    void put(Key key, Value value);

    // Entfernt den Eintrag zum Schlüssel key
    void remove(Key key);
}
```

*Hinweis*: Java erlaubt es nicht Arrays von generischen Typen zu erstellen. Sie können hier bei der Erstellung des Arrays die Typparameter weglassen und mit `@SuppressWarnings("unchecked")` die Warnung ignorieren:

```
@SuppressWarnings("unchecked")
HashEntry<Key, Value>[] table = new HashEntry[tabsize];
```