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

TU Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Lösungshinweise/-vorschläge zum Übungsblatt 10: Software-Entwicklung 1 (WS 2017/18)

Die Hinweise und Vorschläge in diesem Dokument sollen der Lösungsfindung dienen und erheben demnach weder Anspruch auf Vollständigkeit noch Korrektheit. Sollten Sie Fehler finden, würden wir uns freuen, wenn Sie uns diese mitteilen. (Kontaktdaten finden Sie auf unserer Webpräsenz.)

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`).

Siehe unten.

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`).

Siehe unten.

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.

```
import java.util.Comparator;

public class OrtVergleicher implements Comparator<Ort> {
    @Override
    public int compare(Ort o1, Ort o2) {
        if (o1.getPostleitzahl() < o2.getPostleitzahl()) {
            return -1;
        } else if (o1.getPostleitzahl() > o2.getPostleitzahl()) {
            return 1;
        } else {
```

```
            return o1.getName().compareTo(o2.getName());
        }
    }
}

import java.util.Comparator;

public class AdressVergleicher implements Comparator<Adresse> {
    @Override
    public int compare(Adresse a1, Adresse a2) {
        OrtVergleicher ortVergleicher = new OrtVergleicher();
        int ortVergleich = ortVergleicher.compare(a1.getOrt(), a2.getOrt());
        if (ortVergleich != 0) {
            return ortVergleich;
        } else {
            int strassenVergleich = a1.getStrasse().compareTo(a2.getStrasse());
            if (strassenVergleich != 0) {
                return strassenVergleich;
            } else {
                return a1.getHausNummer().compareTo(a2.getHausNummer());
            }
        }
    }
}
```

Seit Java 8 lassen sich Vergleiche auch deutlich kompakter implementieren, zum Beispiel:

```
import java.util.Comparator;

public class Vergleicher {

    public static Comparator<Ort> ortVergleicher() {
        return Comparator
                .comparing(Ort::getPostleitzahl)
                .thenComparing(Ort::getName);
    }

    public static Comparator<Adresse> addressVergleicher() {
        return Comparator
                .comparing(Adresse::getOrt, ortVergleicher())
                .thenComparing(Adresse::getStrasse)
                .thenComparing(Adresse::getHausNummer);
    }
}
```

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.

```java
import java.util.Objects;

public class Ort {
    private int postleitzahl;
    private String name;

    public Ort(int postleitzahl, String name) {
        this.postleitzahl = postleitzahl;
        this.name = name;
    }

    public int getPostleitzahl() {
        return postleitzahl;
    }

    public String getName() {
        return name;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj instanceof Ort) {
            Ort ort = (Ort) obj;
            return postleitzahl == ort.postleitzahl
                    && Objects.equals(name, ort.name);
        }
        return false;
    }

    @Override
    public int hashCode() {
        return Objects.hash(postleitzahl, name);
    }
}

import java.util.Objects;

public class Adresse {
    private Ort ort;
    private String strasse;
    private String hausNummer;

    public Adresse(Ort ort, String strasse, String hausNummer) {
        this.ort = ort;
        this.strasse = strasse;
        this.hausNummer = hausNummer;
    }

    public Ort getOrt() {
        return ort;
    }

    public String getStrasse() {
        return strasse;
    }

    public String getHausNummer() {
        return hausNummer;
    }

   @Override
   public boolean equals(Object obj) {
       if (obj instanceof Adresse) {
           Adresse a = (Adresse) obj;
```

```
            return Objects.equals(ort, a.ort)
                    && Objects.equals(strasse, a.strasse)
                    && Objects.equals(hausNummer, a.hausNummer);
        }
        return false;
    }

    @Override
    public int hashCode() {
        return Objects.hash(ort, strasse, hausNummer);
    }
}
```

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`.

```
public class Kreis extends Figur {
    private Vec2 mittelpunkt;
    private double radius;

    public Kreis(Vec2 mittelpunkt, int radius) {
        this.mittelpunkt = mittelpunkt;
        this.radius = radius;
    }

    @Override
    public void zeichnen(SEGraphics seg) {
        super.zeichnen(seg);
        seg.drawCircle(mittelpunkt.x, mittelpunkt.y, radius);
    }

    @Override
    public double abstandZu(Vec2 p) {
        return Math.max(0, mittelpunkt.distanceTo(p) - radius);
    }


    @Override
    public void verschiebenUm(Vec2 delta) {
        mittelpunkt = mittelpunkt.plus(delta);
    }

    public Vec2 getMittelpunkt() {
        return mittelpunkt;
```

```
    }

    public void setRadius(double radius) {
        this.radius = radius;
    }

}
```

- Erstellen Sie eine neue Unterklasse von `ZeichenTool` für das Zeichnen von Kreisen.

```
public class ZeichenToolKreis extends ZeichenTool {

    @Override
    public String getName() {
        return "Kreis";
    }

    private Kreis kreis;

    @Override
    public void start(Vec2 pos) {
        kreis = new Kreis(pos, 1);
        zeichenBlatt.addFigur(kreis);
    }

    @Override
    public void ziehen(Vec2 pos) {
        kreis.setRadius(kreis.getMittelpunkt().distanceTo(pos));
    }

}
```

- Passen Sie die `main`-Methode in der Klasse `Main` an, indem Sie dort ihr Tool zum Kreise zeichnen in die Liste der Tools eintragen.

```
   tools.add(new ZeichenToolKreis());
```

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

```
public class Rechteck extends Figur {
    private Vec2 start;
    private Vec2 groesse;

    public Rechteck(Vec2 start, Vec2 groesse) {
        this.start = start;
        this.groesse = groesse;
    }

    @Override
    public void zeichnen(SEGraphics seg) {
        super.zeichnen(seg);
        seg.drawRect(start.x, start.y, groesse.x, groesse.y);
    }


    @Override
    public double abstandZu(Vec2 p) {
        Vec2 untenLinks = new Vec2(start.x, start.y + groesse.y);
        Vec2 obenRechts = new Vec2(start.x+groesse.x, start.y);
        Vec2 untenRechts = start.plus(groesse);
        if (p.x < start.x) {
```

```
            // Links vom Rechteck
            return p.distanceToLine(start, untenLinks);
        } else if (p.x > start.x + groesse.x) {
            // Rechts vom Rechteck
            return p.distanceToLine(obenRechts, untenRechts);
        } else {
            // x-Koordinate im Rechteck
            if (p.y < start.y) {
                // Oberhalb des Rechtecks
                return p.distanceToLine(start, obenRechts);
            } else if (p.y > start.y + groesse.y) {
                // Unterhalb des Rechtecks
                return p.distanceToLine(untenLinks, untenRechts);
            } else {
                // Im Rechteck
                return 0;
            }
        }
    }


    @Override
    public void verschiebenUm(Vec2 delta) {
        start = start.plus(delta);
    }


    public void setStart(Vec2 start) {
        this.start = start;
    }


    public void setGroesse(Vec2 groesse) {
        this.groesse = groesse;
    }


    public Vec2 getGroesse() {
        return groesse;
    }

}

public class ZeichenToolRechteck extends ZeichenTool {

    private Rechteck reckteck;
    private Vec2 startPunkt;

    @Override
    public String getName() {
        return "Rechteck";
    }

    @Override
    public void start(Vec2 pos) {
        reckteck = new Rechteck(pos, new Vec2(1, 1));
        startPunkt = pos;
        zeichenBlatt.addFigur(reckteck);
    }

    @Override
    public void ziehen(Vec2 pos) {
        double startX = Math.min(startPunkt.x, pos.x);
        double startY = Math.min(startPunkt.y, pos.y);
        reckteck.setStart(new Vec2(startX, startY));
```

```
            double groesseX = Math.abs(startPunkt.x - pos.x);
            double groesseY = Math.abs(startPunkt.y - pos.y);
            reckteck.setGroesse(new Vec2(groesseX, groesseY));
        }

    }
```

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.

Man kann der abstrakten Klasse `Figur` einen Punkt geben, der die Position der Figur festlegt. Dann muss man die Methode `verschiebenUm` nur in der Klasse `Figur` implementieren und dort diese Position anpassen. Wichtig ist dann aber, dass man keine anderen absoluten Punkte mehr verwendet, sondern nur noch relative. Zum Beispiel müsste der End-Punkt der Linie relativ zur Start-Position gegeben werden.

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.

```
// In Klasse Constant:
    @Override
    public int eval(Map<String, Integer> context) {
        return value;
    }

// In Klasse Variable:
    @Override
    public int eval(Map<String, Integer> context) {
        return context.get(name);
    }

// In Klasse Addition:
    @Override
    public int eval(Map<String, Integer> context) {
        return left.eval(context) + right.eval(context);
    }

// In Klasse Multiplication:
    @Override
    public int eval(Map<String, Integer> context) {
```

```
        return left.eval(context) * right.eval(context);
    }

// In Klasse Assignment:
    @Override
    public void eval(Map<String, Integer> context) {
        int value = exp.eval(context);
        context.put(var, value);
    }

// In Klasse Print:
    @Override
    public void eval(Map<String, Integer> context) {
        System.out.println(exp.eval(context));
    }


// In Klasse Program:
    public void execute() {
        Map<String, Integer> context = new HashMap<>();
        for (Assignment assgn : assignments) {
            assgn.eval(context);
        }
        printExp.eval(context);
    }
```

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.

```
public class LPMap implements Map {
    private HashEntry[] table;

    public LPMap(int size) {
        table = new HashEntry[size];
    }

    private int hash( int key ) {
        return Math.abs(key) % table.length;
    }

    public String get(int key) {
        int i;
        for (i = hash(key); table[i] != null; i = (i+1)%table.length) {
            if (table[i].key == key) {
                return table[i].data;
            }
        }
        return null;
    }

    public void put(int key, String data) {
        int i;
        for (i = hash(key); table[i] != null && table[i].key != key; i = (i+1)%table
    .length);
        table[i] = new HashEntry(key, data);
    }

    public void remove(int key) {
        int i;
        for (i = hash(key); table[i] != null && table[i].key != key; i = (i+1)%table
    .length);
        if (table[i] == null) {
            return;
        }
        // delete element
        table[i] = null;
        // rehash all elements until we reach the end of the cluster (table[j]==null
    )
        for (int j = (i+1)%table.length; table[j] != null; j = (j+1)%table.length) {
            // delete and re-add the element
            int pKey = table[j].key;
            String pData = table[j].data;
            table[j] = null;
            put(pKey, pData);
        }
    }
}

class HashEntry {
    int key;
    String data;
    HashEntry(int key, String data) {
      this.key = key;
      this.data = data;
    }
  }
```

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];
```

```
class HashMap<Key, Value> implements Map<Key, Value> {

    private HashEntry<Key, Value>[] table;

    public HashMap(int tabsize) {
    /* tabsize sollte eine Primzahl sein */
        @SuppressWarnings("unchecked")
        HashEntry<Key, Value>[] table = new HashEntry[tabsize];
        this.table = table;
    }

    private int hash(Key key) {
        return Math.abs(key.hashCode()) % table.length;
    }

    public Value get(Key key) {
        // ermittle Index in der Hashtabelle
        int hix = hash(key);
        HashEntry<Key, Value> entry = table[hix];

        while (entry != null) {
            // iteriere ueber die Elemente, bis Schluessel gefunden
            if (entry.key.equals(key)) {
                return entry.data;
            }
            entry = entry.next;
        }
        return null;
    }

    public void put(Key key, Value data) {
        if (data != null) {
            int hix = hash(key);
            HashEntry<Key, Value> currentEntry = table[hix];
            HashEntry<Key, Value> newEntry = new HashEntry<>(key, data);
            newEntry.next = currentEntry;  // fuege neuen Eintrag am Anfang ein
            table[hix] = newEntry;
        }
    }

    public void remove(Key key) {
        int hix = hash(key);
```

```
        HashEntry<Key, Value> entry = table[hix];

        // entferne alle HashEntries mit dem gegebenen Schlüssel
        HashEntry<Key, Value> dummy = new HashEntry<>(null, null);
        dummy.next = entry;
        HashEntry<Key, Value> current = dummy;
        while (current.next != null) {
            if (current.next.key.equals(key)) { // entferne den Eintrag
                current.next = current.next.next;
            } else {                    // gehe zum nächsten Eintrag
                current = current.next;
            }
        }
        table[hix] = dummy.next;
    }
}
```