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

TU Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Lösungshinweise/-vorschläge zum Übungsblatt 11: 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 Defensive Programmierung (4 Punkte)

```
public static Object find(List<Entry> entries, String key) {
    if (entries == null) {
        throw new IllegalArgumentException("entries ist null");
    }
    if (key == null) {
        throw new IllegalArgumentException("key ist null");
    }
    for (Entry e : entries) {
        if (e == null) {
            throw new IllegalArgumentException("entries enthaelt einen null-Eintrag"
    );
        }
    }
    for (Entry e : entries) {
        if (key.equals(e.getKey())) {
            return e.getValue();
        }
    }
    throw new IllegalArgumentException("Kein Eintrag mit gesuchtem Schluessel
    vorhanden");
}
```

Aufgabe 2 Warteschlange (15 Punkte)

```
public class ElementNullException extends Exception {}
public class QueueEmptyException extends Exception {}
public class QueueFullException extends Exception {}


public class ArrayQueue<T> implements Queue<T> {
    private final T[] buffer;
    private int size = 0;
    private int headPos = 0;

    ArrayQueue(int capacity) {
        @SuppressWarnings("unchecked")
        T[] a = (T[]) new Object[capacity];
        buffer = a;
    }

    public void add(T e) throws QueueFullException, ElementNullException {
        if (e == null) {
            throw new ElementNullException();
        }
        if (size >= buffer.length) {
            throw new QueueFullException();
        }
        buffer[(headPos + size) % buffer.length] = e;
        size++;
    }

    public T remove() throws QueueEmptyException {
        if (size == 0) {
            throw new QueueEmptyException();
        }
        T e = buffer[headPos];
        buffer[headPos] = null; // allow garbage collection to clean element
        headPos = (headPos + 1) % buffer.length;
        size--;
        return e;
    }

    public T element() throws QueueEmptyException {
        if (size == 0) {
            throw new QueueEmptyException();
        }
        return buffer[headPos];
    }

    public boolean isEmpty() {
        return size == 0;
    }

}
```

Aufgabe 3 Grafikeditor und Gson (7 Punkte)

In dieser Aufgabe sollen Sie den Grafikeditor von Blatt 10, Aufgabe 2, um Funktionen zum Speichern und Laden von Grafiken erweitern. Sie können sich dazu die Vorlage `Grafikeditor.zip` herunterladen, welcher den fertigen Grafikeditor von Blatt 10 enthält. Außerdem wurde die Datei Gui.java um ein Menü zum Speichern und Laden erweitert. Wenn ein Benutzer eine Datei Speichern will, wird die Methode `save` in der Klasse `SaveLoad` aufgerufen. Zum Laden wird entsprechend die Prozedur `load` aufgerufen. Diese beiden Prozeduren sollen Sie in dieser Aufgabe mit Hilfe der Gson-Bibliothek implementieren. Die genaue Struktur

der Json-Dokumente bleibt Ihnen überlassen. Wenn beim Speichern oder Laden ein Fehler auftritt, soll eine `RuntimeException` geworfen werden.

Die `save`-Prozedur erhält ein `ZeichenBlatt` und einen Dateipfad und soll den Inhalt des Zeichenblatts in der entsprechenden Datei als Json speichern und dazu die Gson-Bibliothek verwenden. Wenn die Datei bereits existiert, soll sie überschrieben werden.

Die `load`-Prozedur erhält ein `ZeichenBlatt` und einen Dateipfad und soll den Inhalt des Zeichenblattes aus der gegebenen Datei laden.

Beachten Sie die Hinweise zur Verwendung von Gson im Skript (Kapitel 18).

```
import com.google.gson.*;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.lang.reflect.Type;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.Arrays;
import java.util.List;

public class SaveLoad {
    public static void save(Path path, ZeichenBlatt zeichenBlatt) {
        Gson gson = getGson();
        try (BufferedWriter w = Files.newBufferedWriter(path)) {
            gson.toJson(zeichenBlatt, w);
        } catch (IOException e) {
            throw new RuntimeException("Konnte Datei " + path + " nicht speichern.")
    ;
        }
    }

    public static void load(Path path, ZeichenBlatt zeichenBlatt) {
        Gson gson = getGson();
        try (BufferedReader r = Files.newBufferedReader(path)) {
            ZeichenBlatt blatt = gson.fromJson(r, ZeichenBlatt.class);
            zeichenBlatt.setFiguren(blatt.getFiguren());
        } catch (IOException e) {
            throw new RuntimeException("Konnte Datei " + path + " nicht speichern.")
    ;
        }
    }

    private static Gson getGson() {
        GsonBuilder builder = new GsonBuilder();
        builder.registerTypeAdapter(Figur.class, new FigurSerialization());
        return builder.create();
    }


    public static class FigurSerialization implements JsonSerializer<Figur>,
    JsonDeserializer<Figur> {

        List<Class<?>> subClasses = Arrays.asList(Rechteck.class, Linie.class, Kreis
    .class);

        @Override
        public Figur deserialize(JsonElement json, Type typeOfT,
    JsonDeserializationContext context) throws JsonParseException {
            JsonObject obj = json.getAsJsonObject();
            String type = obj.get("type").getAsString();
            for (Class<?> subClass : subClasses) {
```

```
                if (type.equals(subClass.getSimpleName())) {
                    return context.deserialize(json, subClass);
                }
            }
            throw new JsonParseException("Unsupported type: " + type);
        }

        @Override
        public JsonElement serialize(Figur fig, Type typeOfSrc,
    JsonSerializationContext context) {
            String type = fig.getClass().getSimpleName();
            JsonObject obj = (JsonObject) context.serialize(fig);
            obj.add("type", new JsonPrimitive(type));
            return obj;
        }
    }
}
```

Aufgabe 4 Ströme (8 Punkte)

Gegeben ist das folgende Interface, welches einen Strom von String-Werten repräsentiert:

```
interface StringStream {
  // Liefert das naechte Element im Stream oder null wenn das Ende erreicht ist
  String read() throws IOException;
}
```

Implementieren Sie eine Klasse `WordStream`, welche dieses Interface implementiert. Die Klasse `WordStream` soll im Konstruktor einen `Reader` (aus dem Paket `java.io`) nehmen. Dieser `Reader` liefert einen Strom von Zeichen aus denen wir Worte generieren wollen. Die generierten Strings sollen also die Worte im `Reader` sein, wobei Worte jeweils durch ein oder mehrere Zeichen voneinander getrennt sind, die keine Buchstaben sind.

Beispiel:

```
Reader r = new StringReader("How are you, Donald?");
WordStream ws = new WordStream(r);
assertEquals("How", ws.read());
assertEquals("are", ws.read());
assertEquals("you", ws.read());
assertEquals("Donald", ws.read());
assertEquals(null, ws.read());
```

```
import java.io.IOException;
import java.io.Reader;

public class WordStream implements StringStream {
    private Reader reader;
    private StringBuilder buffer = new StringBuilder();

    public WordStream(Reader reader) {
        this.reader = reader;
    }

    @Override
    public String read() throws IOException {
        buffer.setLength(0);
        while (true) {
            int read = reader.read();
            if (read < 0) {
                // Ende der Eingabe
                if (buffer.length() == 0) {
```

```
                    return null;
                } else {
                    return buffer.toString();
                }
            }
            char c = (char) read;
            if (Character.isLetter(c)) {
                buffer.append(c);
            } else if (buffer.length() > 0 ) {
                return buffer.toString();
            }
        }
    }
}
```