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

TU Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Übungsblatt 11: Software-Entwicklung 1 (WS 2017/18)

Ausgabe: 15.01.18
Abgabe: 22.01.18

Aufgabe 1 Defensive Programmierung (4 Punkte)

Bei der defensiven Programmierung werden ungültige Eingaben explizit abgefangen und behandelt. Programmierfehler können somit schneller erkannt und behoben werden. Vorbedingungen in Methoden-Spezifikationen sind vom Aufrufer der Methode sicherzustellen. In der defensiven Programmierung können sie aber auch von der Methode selbst überprüft werden, so dass bei Fehlern im Aufruf eine gut Fehlermeldung an den Aufrufer gegeben werden kann. Da diese Art der Fehler Programmierfehler darstellen, verwendet man hier in der Regel keine checked Exceptions sondern Exceptions eines Subtyps von `RuntimeException`.

Passen Sie die Methode `find` im folgenden Code so an, dass sie eine `IllegalArgumentException` mit passender Nachricht wirft, wenn eine der Vorbedingungen nicht erfüllt ist. Die Nachricht kann im Konstruktor der `IllegalArgumentException` angegeben werden.

```
interface Entry {
  String getKey();
  Object getValue();
}

class DefProg {
  /*
   * requires entries != null
   *       && key != null
   *       && für alle e in entries: e != null
   *       && entries enthält einen Eintrag e mit key.equals(e.getKey())
   * returns e.getValue(), wobei e der erste Eintrag in entries
   *                       mit key.equals(e.getKey()) ist
   */
  public static Object find(List<Entry> entries, String key) {
    for (Entry e : entries) {
      if (key.equals(e.getKey())) {
        return e.getValue();
      }
    }
    return null;
  }
}
```

Aufgabe 2 Warteschlange (15 Punkte)

Eine Warteschlange (auf englisch "Queue" oder "FIFO (First In, First Out) Queue") ist ein abstrakter Datentyp, der wie eine Liste Daten speichern kann. Eine Warteschlange bietet zwei Operationen zum Verändern an: Man kann ein neues Element hinten in die Warteschlange einfügen, und man kann das erste Element vorne von der Warteschlange wegnehmen.

Implementieren Sie eine Warteschlange, in der Objekte eines gegebenen Typs gespeichert werden können. Dazu sollen Sie eine Klasse `ArrayQueue<T>` schreiben, welche das folgende Interface `Queue` implementiert.

```
public interface Queue<T> {
    /** Fuegt ein Element hinten in die Warteschlange an.
    * Wirft eine QueueFullException, wenn die Warteschlange voll ist und
    * eine ElementNullException, wenn der Parameter e null ist. */
    void add(T e) throws QueueFullException, ElementNullException;

    /** Gibt das erste Element in der Warteschlange zurueck und entfernt es aus der
    * Warteschlange.
    * Wirft eine QueueEmptyException, wenn die Warteschlange leer ist. */
    T remove() throws QueueEmptyException;

    /** Gibt das erste Element in der Warteschlange zurueck.
    * Wirft eine QueueEmptyException, wenn die Warteschlange leer ist. */
    T element() throws QueueEmptyException;

    /** Testet, ob die Warteschlange leer ist. */
    boolean isEmpty();
}
```

Es gibt mehrere Möglichkeiten, die Daten einer Warteschlange intern zu verwalten. Für diese Aufgabe sollen Sie ein Array mit fester Größe verwenden, wobei die Größe beim Erstellen der Queue festgelegt werden kann. Informieren Sie sich dazu über die Funktionsweise sogenannter "Ring-Buffer"[1]. Definieren Sie die Ausnahme-Klassen, die im Interface genannt sind, als direkte Unterklassen von `Exception`. Testen Sie Ihre Implementierung mit den Testfällen in der Datei `QueueTest.java`.

Hinweis: Es ist nicht möglich in Java ein Array eines generischen Typs `T` zu erstellen. Sie können statt dessen ein `Object`-Array erstellen und dieses zu einem `T`-Array casten. Da dieser Cast von Java zur Laufzeit nicht überprüft werden kann, gibt es beim Übersetzen eine Warnung. Diese können Sie wie in folgendem Beispiel gezeigt unterdrücken:

```
@SuppressWarnings("unchecked")
T[] a = (T[]) new Object[size];
```

[1] http://en.wikipedia.org/wiki/Circular_buffer

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).

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());
```

Aufgabe 5 Hallo C (keine Abgabe, Vorbereitung auf Blatt 12)

In der nächsten Woche werden wir die Programmiersprache C behandeln. Als Vorbereitung sollen Sie in dieser Aufgabe ein einfaches C-Programm übersetzen und ausführen.

Falls Sie einen eigenen Computer verwenden, installieren Sie sich bitte einen C-Compiler (Anleitungen dazu finden Sie auf der SE1-Webseite beim Material zu dieser Übung. Falls Sie Probleme bei der Installation haben, wenden Sie sich bitte an Ihren Tutor.). Auf den Linux-Rechnern des SCI ist der C-Compiler bereits installiert. Die folgenden Befehle beziehen sich auf den C-Compiler clang. Wenn Sie einen anderen Compiler verwenden, müssen Sie entsprechend andere Befehle zum Übersetzen verwenden.

Erstellen Sie eine Datei `hello.c` mit dem folgenden Inhalt:

```
#include <stdio.h>

int main() {
    printf("Hallo C!\n");
    return 0;
}
```

Übersetzen Sie dann die Datei mit dem folgenden Befehl:

```
$ clang hello.c -o hello
```

Führen Sie dann das Programm `hello` wie folgt aus:

```
$ ./hello
```