Praktische Informatik II
FSS 2012 Programmiernachklausur

Prof. Dr. Heiner Stuckenschmidt

24.08.2012

Name, Vorname: _____

IP-Endung: _____

Unterschrift: _____

- Blättern Sie erst um und loggen Sie sich auf Ihrem Rechner erst ein, wenn Sie dazu aufgefordert werden! User und Passwort werden Ihnen dann mitgeteilt.
- Bitte unterschreiben Sie jetzt gleich auf diesem Blatt den Programmiertest (leeres Feld oben)!
- Nachdem Sie sich auf Ihrem Rechner eingeloggt haben, finden Sie dort Eclipse mit dem benötigten Projekt und allen erlaubten digitalen Nachschlagewerken.
- Die im Projekt enthaltenen Methodensignaturen, Klassen- und Packagenamen dürfen nicht verändert werden, da ansonsten die JUnit-Tests nicht erfolgreich ausgeführt werden können. Nutzen Sie die bereitgestellten Testfälle, um zu überprüfen, ob Sie die geforderte Funktionalität korrekt implementiert haben!
- Die Korrektheit Ihrer Abgaben wird mit Hilfe weiterer Testfälle (nicht mitgeliefert) überprüft! Eine Implementierung die „hart-codiert“ die vorhandenen Testfälle löst, aber keine allgmeingültige Implementierung darstellt, wird als ungültige Lösung gewertet.
- Die Klausur gilt als bestanden, wenn drei von vier Aufgaben vollständig gelöst werden konnten.
- Legen Sie sich einen Stift und Ihre ECUM bereit. Die Klausur beginnt in Kürze nach einigen Erläuerungen durch die Klausuraufsicht.
- Wenn Sie die Klausur vorzeitig beenden wollen, so ist dies der Klausuraufsicht mitzuteilen. Melden Sie sich und bleiben Sie bitte eingeloggt. Die Aufsicht wird dann direkt am Rechner eine zusätzliche Sicherungskopie anfertigen.
- Auch wenn die Klausur für Sie nach Ende der vorgesehene Zeit endet, gilt: Loggen Sie sich nicht aus, da wir eine zusätzliche Sicherungskopie anfertigen werden. Sie können dem Anfertigen der Sicherungskopie beiwohnen, oder direkt nach Ende der vorgesehenen Zeit den Raum verlassen.

Viel Erfolg bei der Bearbeitung der Aufgaben!

Aufgabe 1: Gerade oder Ungerade (de.unima.ki.pi2.ptest.aufgabe1)

In dieser Aufgabe geht es um die Klasse `LogicEvenOrOddImpl`, in welcher die leeren Methodenrümpfe der Methoden `isEven` und `isOdd` zu implementieren sind. Beide Methoden erhalten eine boolsche Feldvariable als Eingabe und geben `true` oder `false` zurück, abhängig davon ob die Anzahl der auf `true` gesetzten Felder gerade oder ungerade ist (*Übersetzungshilfe: even = gerade, odd = ungerade).* Zum besseren Verständnis ein Beispiel:

```
boolean[] x = new boolean[]{true, false, true, true, false};
LogicEvenOrOdd logic = new LogicEvenOrOddImpl();
boolean even = logic.isEven(x);
```

Führt man dieses Codefragment aus, so soll die Variable `even` den Wert `false` haben. Hat die Eingabefeldvariable die Länge 0, dann ist folglich keines ihrer Felder auf `true` gesetzt und die Anzal der `true` Werte ist 0. Die Zahl 0 ist eine gerade Zahl.

Ihre Aufgabe besteht darin, die Methoden `isEven` und `isOdd` zu vervollständigen!

Aufgabe 2: Seltene Buchstaben (de.unima.ki.pi2.ptest.aufgabe2)

Diese Aufgabe verwendet Code aus der Musterlöung zur ersten PI-2 Programmierklausur. Die vorliegende Aufgabe ist jedoch in sich abgeschlossen und kann ohne Kenntnisse der ersten Klausur gelöst werden. Wir betrachten zunächst die vier Methoden der Klasse `CharCounterXImpl`.

`parse` Parst einen Text und merkt sich in einer HashMap wie oft welcher Buchstabe vorkommt.

`getNumOfChars` Gibt zurück wie oft ein bestimmter Buchstabe in dem geparsten Text vorkommt.

`toString` Erzeugt einen String, der beschreibt wie oft jeder Buchstabe in dem geparsten Text vorkommt.

`getRareChars` Gibt die beiden Buchstaben zurück, die am seltensten im Text vorkommen.

Die ersten drei Methoden sind bereits vollständig implementiert. Die letzte Methode muss von Ihnen implementiert werden. Diese Methode soll die zwei am seltensten im Text vorkommenenden Buchstaben zurückgeben.[1] Dabei handelt es sich um diejenigen Buchstaben, denen in der HashMap die kleinsten Zahlen zugeordnet sind. Der Rückgabewert der Methode ist ein zwei-elementiges Char-Array. Das erste Element soll der Buchstabe sein, der am seltensten vorkomment. Das zweite Element soll der Buchstabe sein, der am zweit-seltensten vorkommt. Hierzu ein Beispiel:

```
CharCounterX cx = new CharCounterXImpl();
cx.parse("aaabbccdeeeeee");
char[] r = cx.getRareChars();
```

Nach Aufführung dieses Codefragments soll `r[0]` den Wert `'d'` haben und `r[1]` soll den Wert `'b'` oder den Wert `'c'` haben. In diesem Fall ist der zweitseltenste Buchstabe nicht eindeutig bestimmt. Es kann auch vorkommen, dass es mehrere seltenste Buchstaben gibt. Das erste und zweite Element des Rückgabewertes soll dann einen beliebigen seltensten Buchstaben enthalten (allerdings muss es sich um zwei verschiedene Buchstaben handeln). Enthält der zu parsende Text weniger als zwei verschiedenen Buchstaben, so soll `null` zurückgegeben werden.

Ihre Aufgabe besteht darin, die Methode `getRareChars` zu vervollständigen, so dass die oben beschriebene Funktionalität implementiert wird!

[1] Diese Funktionalität wird unter anderem für die Huffman-Codierung benötigt.

Aufgabe 3: Prozessse und Resourcen (de.unima.ki.pi2.ptest.aufgabe3)

In der Klasse `ProcessCheckerImpl` werden aktuell verfügbare Resourcen mittels eines int-Arrays `currentlyAvailableResources` verwaltet. Nehmen wir einmal an, dass zur Zeit drei Einheiten der Resource R_0, zwei Einheiten der Resource R_1 und keine Einheit der Resource R_2 verfügbar sind, dann hat `currentlyAvailableResources` den Wert $\{3,2,0\}$.

Ihre Aufgabe besteht darin, die Methode `getExecutableProcess` zu implementieren. Diese Methode erhält als Eingabe eine ArrayList, deren Elemente int-Arrays sind. Jedes Element dieser Liste stellt einen Prozess dar bzw. die Einheiten an Resourcen, die vorraussichtlich benötigt werden, damit dieser Prozess ausgeführt werden kann. Hierbei gilt, wie auch in Bezug auf den int-Array `currentlyAvailableResources`, dass sich das n-te Element auf die Resource R_n bezieht. Ein Prozess kann ausgeführt werden, wenn aktuell genügend Resourcen zur Verfügung stehen, um ihn auszuführen. Die Methode `getExecutableIndex` gibt den Index des ersten Prozesses zurück, der ausgeführt werden kann. Gibt es keinen ausführbaren Prozess, so soll -1 zurückgeben werden. Im folgenden ein Beispiel.

```
int[] curAvailRes = {3,3,2};
ProcessCheckerImpl pc = new ProcessCheckerImpl(curAvailRes);
ArrayList<int[]> processList = new ArrayList<int[]>();
processList.add(new int[]{1,2,3});
processList.add(new int[]{1,2,0}); // ausfuehrbar
processList.add(new int[]{3,4,2});
processList.add(new int[]{1,1,1}); // ausfuehrbar
int x = pc.getExecutableIndex(processList);
```

Nach Auführung dieses Codes muss `x` den Wert 1 haben, da der erste ausführbare Prozess in der Liste an Index 1 steht.

Ihre Aufgabe besteht darin, die Methode `getExecutableIndex` zu vervollständigen, so dass die oben beschriebene Funktionalität implementiert wird!

Aufgabe 4: Binärzahlen zwischen -1 und 1 (de.unima.ki.pi2.ptest.aufgabe4)

In der Klasse `FixPointZOImpl` geht es um die Binärdarstellung von Zahlen zwischen -1 und 1 (exklusive). Die Repräsentation einer solchen Zahl soll auf einer boolschen Feldvariable beruhen. Diese wird im folgenden `value` genannt. Das erste Element der Feldvariable (`value[0]`) bestimmt das Vorzeichen der Zahl, wobei der Wert `true` für eine negative Zahl (Vorzeichenbit ist gesetzt) und der Wert `false` für eine positive Zahl steht. Der Wert der Zahl selbst ist als Summe $\frac{1}{2}w_1 + \frac{1}{4}w_2 + \frac{1}{8}w_3 + \ldots$ definiert, wobei w_i den Wert 1 hat, wenn `value[i]` den Wert `true` hat, ansonsten hat w_i den Wert 0. So ergibt {`true, false, true, true`} beispielweise die Zahl $(-1)*(\frac{1}{2}*0+\frac{1}{4}*1+\frac{1}{8}*1) = -0{,}375$.

Die Methode `setValue` ist bereits vorgegeben und bestimmt den Wert der Zahl als boolsche Feldvariable. Der Rumpf der Methode `getDoubleValue` ist leer und soll von Ihnen implementiert werden. Diese Methode soll den Wert der Zahl als `double` zurückgeben. Beachten Sie, dass die boolsche Feldvariable mindestens 1 Element haben muss, die Länge ansonsten jedoch nicht eingeschränkt ist. Hieraus folgt, dass es mehrere Darstellungen der Zahl 0 gibt. Im folgenden ein Anwendungsbeispiel

```
FixPointZO fp = new FixPointZOImpl();
boolean[] value = {true, false, true, true};
fp.setValue(value);
double d = fp.getDoubleValue();
System.out.println("Doublewert der Zahl: " + d);
```

Ihre Aufgabe besteht darin, die Methode `getDoubleValue` zu vervollständigen, so dass die oben beschriebene Funktionalität implementiert wird!