Praktikum Grundlagen der Programmierung

Prof. Dr. Harald Räcke, Prof. Dr. Felix Brandt, J. Kranz, A. Reuss WS 2017/18

Übungsblatt 13 Abgabefrist: keineAbgabe

Σ Punkte: 0

Aufgabe 13.1 (P) Synchronisiertes Lesen und Schreiben

Betrachten Sie folgendes Programm:

```
3  public class TimesTwo implements Runnable {
4    private final SyncNumber n;
5
6    public TimesTwo(SyncNumber c) {
7      n = c;
8    }
9
10   @Override
11   public void run() {
12     int i = n.read();
13     n.write(i * 2);
14     System.out.println("write " + i * 2);
15   }
16
17   public static void main(String[] args) {
18     SyncNumber n = new SyncNumber();
19     TimesTwo d1 = new TimesTwo(n);
20     TimesTwo d2 = new TimesTwo(n);
21     new Thread(d1).start(); // Thread 1
22     new Thread(d2).start(); // Thread 2
23   }
24
25 }
```

Die Klasse `SyncNumber` ist dabei gegeben durch:

```
3  public class SyncNumber {
4    private int c = 1;
5
6    public synchronized int read() {
7      return c;
8    }
9
10   public synchronized void write(int c) {
11     this.c = c;
12   }
13 }
```

Geben Sie zwei (verschiedene) mögliche Konsolenausgaben des Programms an und füllen Sie für jeden dieser Programmdurchläufe die Tabelle unten aus. Jede Zeile steht für einen Zeitschritt und darf daher auch nur genau einen Eintrag enthalten. Als Einträge sind *ausschließlich* folgende erlaubt:

- `d1.run:Enter` / `d1.run:Return` analog für `d2`
- `n.read:Enter` / `n.read:Return`
- `n.write:Enter` / `n.write:Return`
- `n.lock:Enter` / `n.lock:Return`
- `n.unlock()`
- `println(txt)`

Dabei bedeuten die Einträge folgendes:

- `x.foo:Enter` gibt an, dass die Methode `foo` auf dem Objekt `x` aufgerufen wurde. Das Pendant `x.foo:Return` gibt an, dass der Aufruf zum Aufrufer zurückkehrt.
- `x.lock:Enter` versucht das Lock auf dem Objekt `x` zu akquirieren. Das Pendant `x.lock:Return` gibt an, dass das Lock akquiriert wurde.
- `x.unlock()` gibt das Lock auf dem Objekt `x` frei (dies ist eine Kurzschreibweise für die zwei konsekutiven Kommandos `x.unlock:Enter; x.unlock:Return`).
- `println(txt)` gibt an, dass der String `txt` via einem `System.out.println()`-Aufruf auf der Konsole ausgegeben wird. D.h. `txt` enthält keine Variablen oder noch auszuwertende Ausdrücke.

Wird eine `synchronized`-Methode mit dem Namen `foo` von dem Objekt `x` aufgerufen, so wird erst `x.foo:Enter` ausgeführt und dann das jeweilige Lock `x.lock:Enter` akquiriert. Den Aufruf `System.out.println(String txt)` können Sie durch `println()` darstellen.

Programmdurchlauf Variante 1:

Thread 1	Thread 2
d1.run:Enter	
	d2.run:Enter
n.read:Enter	
n.lock:Enter	
	n.read:Enter
	n.lock:Enter
n.lock:Return	
n.unlock()	
n.read:Return	
n.write:Enter	
	n.lock:Return
	n.unlock()
	n.read:Return
n.lock:Enter	
	n.write:Enter
	n.lock:Enter
n.lock:Return	
n.unlock()	
n.write:Return	
println(write 2)	
d1.run:Return	
	n.lock:Return
	n.unlock()
	n.write:Return
	println(write 2)
	d2.run:Return

Ausgabe auf der Konsole:

```
write 2
write 2
```

Programmdurchlauf Variante 2:

Thread 1	Thread 2
d1.run:Enter	
	d2.run:Enter
n.read:Enter	
n.lock:Enter	
	n.read:Enter
n.lock:Return	
n.unlock()	
n.read:Return	
n.write:Enter	
n.lock:Enter	
n.lock:Return	
n.unlock()	
n.write:Return	
	n.lock:Enter
	n.lock:Return
	n.unlock()
	n.read:Return
	n.write:Enter
	n.lock:Enter
	n.lock:Return
	n.unlock()
	n.write:Return
	println(write 4)
println(write 2)	
	d2.run:Return
d1.run:Return	

Ausgabe auf der Konsole:

```
write 4
write 2
```

Lösungsvorschlag 13.1

Siehe Anhang

Aufgabe 13.2 (P) ConcModificationException

In der Vorlesung wurden Iteratoren vorgestellt. Die Java Standardbibliothek stellt für die Implementierung von Iteratoren zwei Interfaces zur Verfügung: Das Interface `Iterator<E>`[1] für den Iterator und das Interface `Iterable<T>`[2] für Datenstrukturen, die einen Iterator zur Verfügung stellen. Für diese Datenstrukturen kann die Kurzschreibweise für `for`-Schleifen verwendet werden. Beispiel: Alle Collections der Java Standardbibliothek implementieren das Interface `Iterable<T>`, stellen also einen Iterator zur Verfügung. Dieser wird bei der folgenden Kurzschreibweise verwendet:

[1] https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html
[2] https://docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html

```java
LinkedList<Integer> list = new LinkedList<>();
// add some elements to list
for (int tmp : list)
   System.out.println(tmp + " ");
```

Alternativ kann man auch direkt mit dem Iterator über die Liste iterieren. Dafür stehen die Methoden `hasNext()` und `next()` zur Verfügung.

```java
LinkedList<Integer> list = new LinkedList<>();
// add some elements to list
for (Iterator<Integer> it = list.iterator(); it.hasNext(); )
   System.out.println(it.next() + " ");
```

Möchte man nicht nur über eine Liste iterieren, sondern diese auch gleichzeitig modifizieren, darf man das nur über Methoden, die vom Iterator selbst zur Verfügung gestellt werden. Der `listIterator` der Klasse `List` implementiert das Interface `ListIterator<E>` und stellt deshalb einige Funktionen zur Modifizierung einer Liste (`add(E e)`, `remove()`) zur Verfügung[3]. Verändert man jedoch eine Liste anderweitig, während man mit einem Iterator über diese iteriert, wird eine `ConcurrentModificationException` geworfen[4]. Probieren Sie das mit dem folgenden Code aus:

```java
LinkedList<Integer> list = new LinkedList<>();
for (int i = 0; i < 20; i++)
   list.add(i);
for (int tmp : list) {
   System.out.print(tmp + " ");
   list.remove(tmp); //ConcurrentModificationException
}
```

Prinzipiell sollte man sich deshalb merken, dass man eine Liste nicht anderweitig modifiziert während man mit einem Iterator darüber iteriert! Für Modifikationen kann man auf die vom Iterator zur Verfügung gestellten Methoden zurückgreifen. Insbesondere sollte man sich auch nicht auf eine `ConcurrentModificationException` verlassen, da es bestimmte Konstellationen gibt, in denen keine `ConcurrentModificationException` geworfen wird und der Iterator dann nicht vorhersehbares Verhalten zeigt. Probieren Sie dazu den folgenden Code aus:

```java
LinkedList<Integer> list = new LinkedList<>();
for (int i = 0; i < 20; i++)
 list.add(i);
System.out.println(list);
int i = 0;
for (int tmp : list) {
```

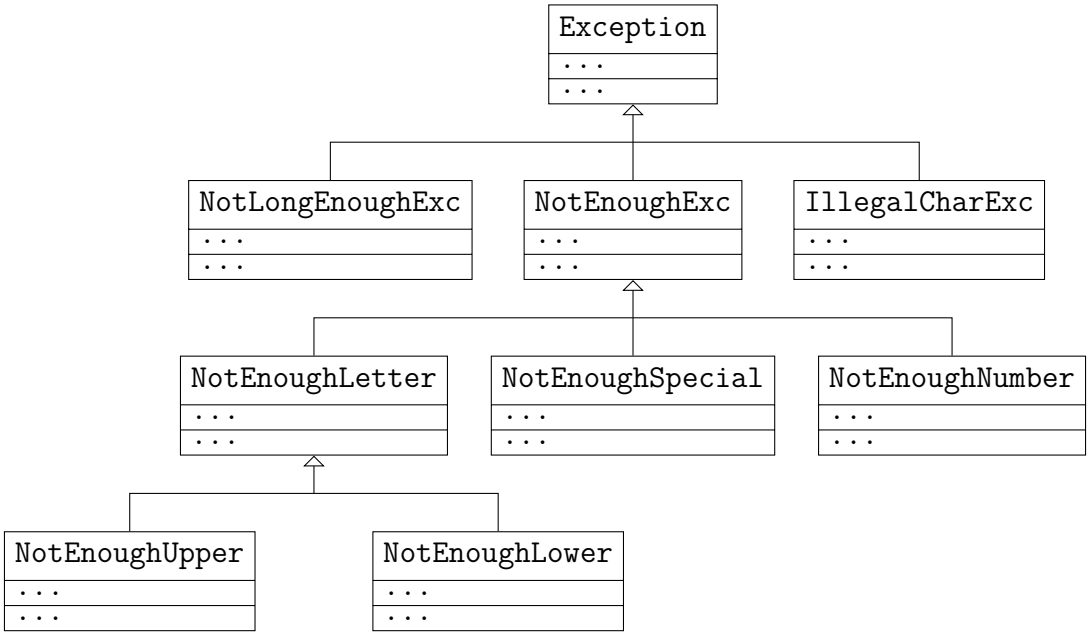
[3] https://docs.oracle.com/javase/8/docs/api/java/util/ListIterator.html

[4] https://docs.oracle.com/javase/8/docs/api/java/util/ConcurrentModificationException.html

```
7   System.out.print(tmp + " ");
8   i++;
9   if(i > 18)
10    list.remove(0);
11  }
12  System.out.println("\n" + list);
```

Aufgabe 13.3 (P) Exceptions

In dieser Aufgabe wollen wir einige Exception-Klassen (Klassen, die von der Klasse `Exception` erben), die bei der Überprüfung der Gültigkeit eines Passworts verwendet werden können, implementieren. Dafür soll die folgende Klassenhierarchie umgesetzt werden, wobei `Exception`, die Klasse `Exception` der Java-Standardbibliothek ist.



Für ein Passwort können die folgenden Mindestanforderungen gestellt werden, die bei Missachtung zu einer entsprechenden Exception führen:

- Das Passwort muss eine Mindestlänge haben, andernfalls wird eine `NotLongEnoughExc`-Exception geworfen.
- Das Passwort muss eine Mindestanzahl an Großbuchstaben enthalten, andernfalls wird eine `NotEnoughUpper`-Exception geworfen.
- Das Passwort muss eine Mindestanzahl an Kleinbuchstaben enthalten, andernfalls wird eine `NotEnoughLower`-Exception geworfen.
- Das Passwort muss eine Mindestanzahl an Sonderzeichen enthalten, andernfalls wird eine `NotEnoughSpecial`-Exception geworfen.
- Das Passwort muss eine Mindestanzahl an Ziffern enthalten, andernfalls wird eine `NotEnoughNumber`-Exception geworfen.
- Das Passwort darf bestimmte Sonderzeichen *nicht* enthalten, andernfalls wird eine `IllegalCharExc`-Exception geworfen.

Für die konkrete Implementierung gelten folgende Anforderungen:

1. Die Klasse `NotLongEnoughExc` hat zwei private `int`-Variablen `should` und `is`. Diese repräsentieren die minimale Länge (`should`), die ein Passwort haben muss und die echt kleinere Länge (`is`), die das Passwort, das die Exception auslöst, hat. Die beiden Variablen werden im Konstruktor `public NotLongEnoughExc(int should, int is)` entsprechend gesetzt. Die `toString()`-Methode liefert eine Fehlermeldung in Form eines Strings, die unter Verwendung der beiden Membervariablen auf die Missachtung der Mindestlänge des Passworts hinweist.

2. Die Klasse `NotEnoughExc` hat zwei `int`-Variablen `should, is`. Diese repräsentieren die Mindestanzahl Zeichen einer bestimmten Kategorie, die ein Passwort enthalten muss und die echt kleinere Anzahl an Zeichen, die das Passwort, das die Exception auslöst, hat. Die beiden Variablen werden entpsrechend im Konstruktor `public NotEnoughExc(int should, int is)` gesetzt.

3. Die Klasse `NotEnoughLetter` hat einen Konstruktor `public NotEnoughLetter(int should, int is)`, der die beiden Variablen der Oberklasse `NotEnoughExc` sinngemäß initialisiert.

4. Die Klasse `NotEnoughUpper` hat einen Konstruktor `public NotEnoughUpper(int should, int is)`, der die beiden Variablen der Oberklasse `NotEnoughExc` sinngemäß initialisiert. Die Methode `toString()` liefert eine Fehlermeldung in Form eines Strings, die unter Verwendung der beiden Membervariablen auf die Missachtung der Mindestanzahl an Großbuchstaben im Passwort hinweist.

5. Die Klasse `NotEnoughLower` hat einen Konstruktor `public NotEnoughLower(int should, int is)`, der die beiden Variablen der Oberklasse `NotEnoughExc` sinngemäß initialisiert. Die Methode `toString()` liefert eine Fehlermeldung in Form eines Strings, die unter Verwendung der beiden Membervariablen auf die Missachtung der Mindestanzahl an Kleinbuchstaben im Passwort hinweist.

6. Die Klasse `NotEnoughSpecial` hat einen Konstruktor `public NotEnoughSpecial(int should, int is)`, der die beiden Variablen der Oberklasse `NotEnoughExc` sinngemäß initialisiert. Die Methode `toString()` liefert eine Fehlermeldung in Form eines Strings, die unter Verwendung der beiden Membervariablen auf die Missachtung der Mindestanzahl an Sonderzeichen im Passwort hinweist.

7. Die Klasse `NotEnoughNumber` hat einen Konstruktor `public NotEnoughNumber(int should, int is)`, der die beiden Variablen der Oberklasse `NotEnoughExc` sinngemäß initialisiert. Die Methode `toString()` liefert eine Fehlermeldung in Form eines Strings, die unter Verwendung der beiden Membervariablen auf die Missachtung der Mindestanzahl an Ziffern im Passwort hinweist.

8. Die Klasse `IllegalCharExc` hat eine private Variable `char used`, die ein Zeichen repräsentiert, das in einem Passwort nicht verwendet werden darf. Die Klasse hat einen Konstruktor `public IllegalCharExc(char used)`, wobei das Zeichen `used` ein Zeichen ist, das im Passwort nicht enthalten sein darf. Im Konstruktor wird die Membervariable entsprechend initialisiert. Die Methode `toString()` liefert eine Fehlermeldung in Form eines Strings, die unter Verwendung der Membervariablen auf die Verwendung des nicht erlaubten Zeichens im Passwort hinweist. In der `toString()`-Methode soll auch eindeutig auf nicht darstellbare Zeichen hingewiesen

werden: Stellen Sie sicher, dass es bei Verwendung der nicht darstellbaren Steuerzeichen `\n`, `\t`, `\r`, `\b`, `\f` eine verbale Beschreibung dieser Steuerzeichen gibt. Ein Backslash kann in einem `String` in Java durch `\\` dargestellt werden, z.B. liefert `System.out.println("line break (\\n)");` auf der Konsole die Ausgabe `line break (\n)`.

Schreiben Sie nun eine Klasse `Password` mit einem Konstruktor

```
1 public Password(int nrUpperShould, int nrLowerShould, int
  ↪ nrSpecialShould, int nrNumbersShould, int lengthShould, char[]
  ↪ illegalChars)
```

und einer Methode `public void checkFormat(String pwd)` sowie einer `main`-Methode. Dabei soll gelten:

a) Die Klasse `Password` hat die privaten `int`-Variablen `nrUpperShould`, `nrLowerShould`, `nrSpecialShould`, `nrNumbersShould`, `lengthShould`, die die Mindestanzahl an Großbuchstaben, Kleinbuchstaben, Sonderzeichen (alle Zeichen, die erlaubt sind und keine Groß- und Kleinbuchstaben (A-Z bzw. a-z) oder Ziffern $(0-9)$ sind) und Ziffern im Passwort sowie die Mindestlänge des Passworts repräsentieren. Außerdem gibt es die private `char[]`-Variable `illegalChars`, die alle Zeichen enthält, die nicht im Passwort enthalten sein dürfen. Alle Klassenvariablen sollen entsprechend im Konstruktor initialisiert werden.

b) Die öffentliche Methode `void checkFormat(String pwd)` soll den übergebenen String `pwd` auf die oben vorgestellten Kriterien eines Passworts, die durch den Konstruktor exakt festgelegt wurden, überprüfen. Wird ein Kriterium verletzt, soll eine entsprechende Exception geworfen werden. D.h. die Methode hat den Zusatz `throws IllegalCharExc, NotEnoughExc, NotLongEnoughExc`

c) In der `main`-Methode soll ein Objekt der Klasse `Password` erzeugt werden. Die konkreten Kriterien für Passwörter, also die Parameter bei der Erzeugung des Objekts dürfen frei gewählt werden. Überprüfen Sie einen `String` mithilfe der `checkFormat`-Methode des zuvor erzeugten `Password`-Objekts auf die dadurch festgelegten Kriterien eines Passworts. Wird eines der Kriterien verletzt, soll der Rückgabewert der `toString()`-Methode der entsprechenden Exception auf der Konsole ausgegeben werden.

Vermeiden Sie Codeduplikate so gut wie möglich durch Verwendung von `super(...)`. Sie können in allen zu implementierenden Klassen davon ausgehen, dass die Konstruktoren nur mit sinnvollen/gültigen Parametern aufgerufen werden. Alle vorgegebenen Membervariablen müssen als `final` deklariert werden.

Lösungsvorschlag 13.3

Siehe Anhang