Institut für Informatik
Prof. Dr.-Ing. Elke Pulvermüller
Dipl.-Systemwiss. Mathias Menninghaus

Universität Osnabrück, 12.06.2018
Testat bis 20.06.2018, 14:00 Uhr

Übungen zu Informatik B

Sommersemester 2018

Blatt 8

Aufgabe 8.1: Ströme (32 Punkte)

Machen Sie sich mit den Klassen im Paket `java.io` der Java-API vertraut. Nutzen Sie das in der Vorlesung vorgestellte *Decorator*-Pattern, um eine eigene `Reader`-Klasse zu implementieren mit der man

- über die Methode `readLine()` alle Zeichen von einem `Reader` bis zum nächsten Zeilenumbruch einlesen und als einen `String` zurückgeben kann. Der Zeilenumbruch soll nicht mit zurückgegeben werden. Beim Erreichen des Dateiendes soll `null` zurückgegeben werden.
- über die Methode `getLineNumber()` die Nummer der zuletzt gelesenen Zeile ermitteln kann.
- über die Methode `getAmountOfMatches()` ermitteln kann, wie oft ein dem Konstruktor übergebener regulärer Ausdruck in der zuletzt gelesenen Zeile gefunden wurde.

Sie können zur Verarbeitung der regulären Ausdrücke die Klassen `java.util.regex.Pattern` und `java.util.regex.Matcher` verwenden.

Implementieren Sie anschließend ein Kommandozeilenprogramm, das mit einem regulären Ausdruck aufgerufen wird und den Inhalt einer Datei über den *Pipe-Operator* `<` zugewiesen bekommt. Das Programm soll jede Zeile der Datei, die den regulären Ausdruck mindestens einmal enthält, zusammen mit der Zeilennummer auf der Standardkonsole ausgeben. Zusätzlich soll für jede ausgegebene Zeile die Anzahl der Vorkommen des regulären Ausdrucks ausgeben werden.

Ein Aufruf des Programms kann beispielsweise wie folgt aussehen:

```
java io/SearchLines "pu.*c" < Beispiel.java
```

Der Inhalt von `Beispiel.java` wird damit in den Standard-Eingabestream `System.in` geschrieben.

Aufgabe 8.2: Standard-Serialisierung (18 Punkte)

Betrachten Sie das Programm zum Berechnen der Fibonacci-Zahlen `Fibonacci.java`, das bereits berechnete Fibonacci-Zahlen in einer `HashMap` vorhält.

Verändern Sie dieses Programm dahingehend, dass die eingesetzte `java.util.HashMap` beim Start aus einer Datei gelesen und nach der Beendigung des Programms wieder in diese Datei zurückgeschrieben wird. Sollte das Programm zum ersten Mal überhaupt aufgerufen werden, soll eine neue `java.util.HashMap` erzeugt werden.

Aufgabe 8.3: Spezielle Serialisierung (32 Punkte)

Erweitern Sie das `OpenHashSet` aus der Musterlösung um das Interface `java.io.Serializable`.

Überschreiben Sie die Standard-Serialisierung soweit, dass das serialisierte `OpenHashSet` möglichst wenig Platz verbraucht. Dafür soll vor allem vermieden werden, dass `null`-Werte oder leere Listen serialisiert werden.

Schreiben Sie anschließend eine Testklasse, in deren `main`-Methode Sie ein Beispiel-`OpenHashSet` erzeugen, serialisieren und in eine Datei schreiben, um es anschließend wieder aus der Datei auszulesen und zu deserialisieren. Lassen Sie die Testklasse automatisiert vergleichen, ob das `OpenHashSet`, das sie serialisieren, das gleiche ist, wie jenes, welches Sie deserialisieren.

Aufgabe 8.4: Suchen (18 Punkte)

Implementieren Sie ein Kommandozeilenprogramm mit dem nach Dateien anhand eines als Kommandozeilenparameter übergebenen regulären Ausdrucks gesucht werden kann. Alle Dateien, dessen Namen dem Ausdruck genügen, sollen auf der Standardkonsole mit Ihrem vollständigen Pfad aufgelistet werden. Das Kommando sei konkret wie folgt definiert:

```
java Search [-r] [-p Pattern] {Verzeichnisname}
```

Mit der Option `-r` wird angegeben, ob Unterverzeichnisse rekursiv durchsucht werden sollen. Das Argument hinter der Option `-p` ist der reguläre Ausdruck, mit dem gesucht werden soll. Wird die Option nicht genutzt, soll nach allen möglichen Namen gesucht werden. Ist kein Verzeichnis angegeben, soll im aktuellen Verzeichnis gesucht werden.