P. Thiemann und K. Anton Sommersemester 2010

Programmieren in Java

`http://proglang.informatik.uni-freiburg.de/teaching/java/2010/`

Betreutes Java-Programmieren 6

2010-06-07

Anfang Loggen Sie sich in der normalen Linux-Umgebung ein.

Starten und Speichern Starten Sie Eclipse, indem Sie in einem Terminalfenster `eclipse` eingeben. Wenn Eclipse Sie fragt, wo der Workspace hin soll, antworten Sie bitte `/home/`*`ihrloginname`*`/workspace`, was normalerweise auch schon voreingestellt sein sollte.

Drumherum Unter der URL `http://nonopapa:8080/teaching/java/2010/` steht Ihnen das Vorlesungsmaterial zur Verfügung. Die Sun-Java-Doku gibt es unter `http://nonopapa:8080/javadoc/`

Bei Fragen zur Rechnerbenutzung, zu den Aufgaben oder zu diesen Hinweisen wenden Sie sich bitte sofort an einen der Tutoren.

Ende Erzeugen Sie ein ZIP- oder tgz-Archiv Ihres Eclipse-Projects: Rechtsklick aufs Project, "Export...", im Ast "General" die Zeile "Archive File" auswählen, prüfen, dass links oben genau Ihr Project angekreuzt ist, im Feld "To archive file" einen Dateinamen eingeben, der Ihren Namen enthaelt: zB `/home/mustermm/Desktop/abgabe3-mustermm.zip` für Max Mustermann, "Finish" klicken. Dabei sollte ein ZIP-Archiv enstehen, das Sie am Ende dem Tutor mailen. Nur ein Attachment pro Mail!

Dieses Blatt geben Sie bitte den Tutoren zurück.

Teams. Nächste Woche geht das Projekt los. Es handelt davon, ein Spiel zu programmieren. Teamgröße: 2-3 (Montagsgruppe: 3, weil weniger Tutoren). Wir werden außerdem Versionsverwaltung einsetzen. Zwei Hausaufgaben bis nächsten Montag:

- Gründen Sie (*nach* den heutigen Programmierübungen) Teams und melden Sie bis nächste Woche Montag für jedes Team Namen, Loginnamen und Emailadressen an `anton@informatik.uni-freiburg.de`.
- Setzen Sie zu Ihrem TF-Account ein WWW-Passwort (Link: siehe Vorlesungshomepage).

Jetzt aber Java. Heute geht alles ins Package `queue`.

Aufgabe 1 (Einfache verkettete Listen, 4 Punkte)
In den wenigen Javakurswochen haben Sie funktionale einfach-verkettete Listen schon oft gesehen. In dieser Übung brauchen wir nochmal eine, und zwar eine generische, also eine, die sich für beliebige Elementtypen `E` instantiieren lässt.

Spätestens jetzt sollten Sie sich die Eclipse-Zaubertasten für Vervollständigung (Ctrl+Space), Erzeugen von Gettern und Settern (Alt+Shift+S R), Erzeugen eines Konstruktors für Felder (Alt+Shift+S O) und allgemeine Quick-Fixes (Ctrl+1) angewöhnen, sonst dauert es zu lange. Nutzen Sie ebenfalls die in den Eclipse-Dialogen angebotene Generierung von Kommentaren. Schreiben Sie

- ein Interface `IList<E>` mit Methoden `isEmpty()`, `getFirst()`, `getRest()` mit sinnvollen Typen.
- eine Klasse `EmptyList<E>`, die `IList<E>` als leere Liste implementiert. Die `get...`-Methoden dürfen `null` zurückliefern oder eine Exception werfen (war noch nicht dran).
- eine Klasse `ListNode<E>`, die `IList<E>` als nichtleere Liste, bestehend aus einem ersten Element vom Typ `T` und einem Rest vom Typ `IList<E>`, implementiert.

Aufgabe 2 (FIFOs, 2+7 Punkte)
Eines der naheliegendsten Anwendungsgebiete von Java-Generics sind Container-Datenstrukturen. Wir basteln heute eine first-in-first-out-Warteschlange (Queue). Eine Queue für Elemente vom Typ `T` habe die Operationen `boolean isEmpty()`, `E get()` und `void put(E x)`. Die `put`-Methode fügt einen Wert in die Queue ein. Die `get`-Methode gibt den am längsten in der Queue wartenden Wert zurück und entfernt ihn (wenn die Queue leer ist, gibt sie `null` zurück oder wirft eine Exception). `isEmpty()` gibt false zurück, gdw mindestens ein Element in der Queue wartet.

(a) Schreiben Sie ein passendes generisches Interface `IQueue<E>`.

(b) Eine Queue kann mit einem Paar von Stapeln implementiert werden: geputtete Elemente werden oben auf den linken Stapel gelegt; bei `get` gibt man das oberste Element des rechten Stapels zurück. Wenn man beim `get` den rechten Stapel leer vorfindet, dreht man zunächst den linken Stapel um und legt ihn auf den Platz für den rechten Stapel. Stapel kann man mit `IList<E>` implementieren. Bitte zwei Tests und eine generische Klasse `DoubleStackQueue<E>`, die `IQueue<E>` implementiert.

Aufgabe 3 (Benchmark, 1+3+2+1 Punkte)
Wir vergleichen die Geschwindigkeit von Queue-Implementationen anhand verschiedener Aufruffolgen. Jede Queue-Implementation soll mit jeder Aufruffolge getestet werden. Die Klasse, die Benchmarks durchführt, soll aber beliebige Klassen (nicht nur Queues) mit beliebigen Aufruffolgen testen können – Generics! Wir bauen daher eine Klasse `Benchmark<T>`, die Instanzen des Typs `T` testet. Die Aufruffolgen werden als Interface `ICallSequence<T>` realisiert. Die Prüflinge, also in unserem Fall, die Queue-Implementationen, werden in Form einer Fabrik `IFactory<T>` hineingereicht, die frische Instanzen von `T` erzeugt.

(a) Schreiben Sie ein Interface `IFactory<T>` mit einer Methode `T make()` und bauen Sie eine Klasse, die `IFactory<IQueue<String>>` implementiert[1]. Jeder Aufruf von `make` in dieser Klasse soll eine frische Instanz von `DoubleStackQueue<String>` liefern.

(b) Schreiben Sie ein Interface `ICallSequence<T>` mit einer Methode `void run(T)` und einer Methode `String getName()`. Bauen Sie eine Klasse, die `ICallSequence<IQueue<String>>` implementiert. In der `run`-Methode soll irgendein Benutzungsszenario von Queues simuliert werden, z.B. 10000mal `put` gefolgt von genau so vielen `get`.

(c) Schreiben Sie eine Klasse `Benchmark<T>` mit einer Methode `void runSingle(IFactory<T>, ICallSequence<T>)`, in der die Fabrik eine Instanz erzeugt und dann die Aufruffolge diese Instanz quält. Messen Sie, wie lange der Aufruf von `ICallSequence.run` dauert, mittels `System.nanoTime()`. Schreiben Sie dann Namen der Aufruffolge und verbrauchte Zeit mit `System.out.print` auf die Konsole.

(d) Schreiben Sie eine Klasse, in deren `main`-Methode Sie mit der Benchmarkklasse ermitteln, wie lange Ihre Queue-Implementation für Ihre Aufruffolge braucht.

(e) (2 Bonuspunkte) Die Java-Standardbibliothek hat schon ein Interface `Queue<T>`, implementiert etwa von `java.util.LinkedList`. Schreiben Sie eine alternative `IQueue`-Implementation, die intern `java.util.LinkedList` verwendet, und benchmarken Sie sie mit `Benchmark<T>` gegen die Implementation aus der vorigen Aufgabe.

[1] Man kann generische Interfaces auch eingeschränkt auf bestimmte Typparameter implementieren, also etwa `class Bla implements IFoo<Blub>`.