Karlsruher Institut für Technologie

Programmieren – Sommersemester 2017

Architecture-driven Requirements Engineering (ARE)
`https://sdqweb.ipd.kit.edu/wiki/Programmieren`
Jun.-Prof. Dr.-Ing. Anne Koziolek

Übungsblatt 3

Ausgabe: 22.05.2017 – 13:00
Abgabe: 06.06.2017 – 06:00

Allgemeine Hinweise

- Achten Sie darauf nicht zu lange Zeilen, Methoden und Dateien zu erstellen[1]
- Programmcode muss in englischer Sprache verfasst sein
- Kommentieren Sie Ihren Code angemessen: So viel wie nötig, so wenig wie möglich
- Wählen Sie geeignete Sichtbarkeiten für Ihre Klassen, Methoden und Attribute
- Verwenden Sie keine Klassen der Java-Bibliotheken ausgenommen Klassen der Pakete `java.lang`, `java.io` und `java.util`, es sei denn die Aufgabenstellung erlaubt ausdrücklich weitere Pakete[1]
- Achten Sie auf fehlerfrei kompilierenden Programmcode[1]
- Halten Sie alle Whitespace-Regeln ein[1]
- Halten Sie die Regeln zu Variablen-, Methoden und Paketbenennung ein und wählen Sie aussagekräftige Namen

Abgabemodalitäten

Die Praktomat-Abgabe wird am **Montag, den 29. Mai um 13:00 Uhr**, freigeschaltet.

- Geben Sie die Java-Klassen zu Aufgabe A als `*.java`-Dateien ab.

Checkstyle

Denken Sie daran, den Checkstyle-Regelsatz von unserer Homepage zu beziehen. Planen Sie, wie immer, ausreichend Zeit für die Abgabe ein, sollte der Praktomat Ihre Abgabe wegen einer Regelverletzung ablehnen.

Öffentliche Tests

Bitte beachten Sie, dass das erfolgreiche Bestehen der öffentlichen Tests für eine erfolgreiche Abgabe dieses Blattes nötig ist. Planen Sie entsprechend Zeit für Ihren ersten Abgabeversuch ein.

[1] Der Praktomat wird die Abgabe zurückweisen, falls diese Regel verletzt ist.

Objektorientierte Modellierung

Achten Sie darauf, dass Ihre Abgaben sowohl in Bezug auf objektorientierte Modellierung, als auch Funktionalität bewertet werden.

Terminal-Klasse

Laden Sie für **diese Aufgabe** die `Terminal`-Klasse[a] von unserer Homepage herunter und platzieren Sie diese unbedingt im Paket `edu.kit.informatik`. Die Methode `Terminal.readLine()` liest eine Benutzereingabe von der Konsole und ersetzt `System.in`. Die Methode `Terminal.printLine()` schreibt eine Ausgabe auf die Konsole und ersetzt `System.out`. Verwenden Sie für jegliche Konsoleneingabe oder Konsolenausgabe die `Terminal`-Klasse. Verwenden Sie in keinem Fall `System.in` oder `System.out`.
Fehlermeldungen werden ausschließlich über die Terminal-Klasse ausgegeben und müssen aus technischen Gründen unbedingt mit `Error,` beginnen.
Laden Sie die Terminal-Klasse niemals zusammen mit Ihrer Abgabe hoch.

[a] https://sdqweb.ipd.kit.edu/lehre/SS17-Programmieren/Terminal.java

A Warteschlangen-Simulation (20 Punkte)

Warteschlangen sind nicht nur Teil des alltäglichen Lebens–bei der Essensausgabe in der Mensa oder beim Einsteigen in den Bus–, sondern spielen auch in der Informatik eine große Rolle. Während in der Mensa Personen darauf warten bedient zu werden, sind es in Computersystemen Prozesse, die darauf warten vom Prozessor oder der Festplatte bedient zu werden. Eine zentrale Frage solcher Warteschlangensysteme lautet: wie schnell müssen die Wartenden bedient werden, um eine maximale Wartezeit nicht zu überschreiten? Konkreter ausgedrückt, wie viele Kassen muss ein Supermarkt öffnen, so dass kein Kunde länger als 5 Minuten warten muss?

In dieser Aufgabe modellieren Sie ein solches Wartschlangensystem, bestehend aus *Aufträgen* (Personen, Prozesse, ...), die in *Wartebereichen* (Kassenbereich, Wartezimmer, ...) darauf warten durch *Bedieneinheiten* (Kassierer, CPU, Festplatte, ...) bedient zu werden. Darauf basierend schreiben Sie eine einfache *Simulation*, die für ein gegebenes Szenario beantwortet, wie lange die einzelnen Aufträge warten mussten, um vollständig bedient zu werden.

A.1 Grundlagen

Aus dem alltäglichen Leben kennen Sie bereits verschiedene Arten von **Wartebereichen**:

- Im Supermarkt reihen Sie sich am Ende der Schlange ein und es werden zunächst diejenigen bedient, die vor Ihnen in der Schlange standen. Dieses Prinzip heißt *first in, first out (FIFO)*. Einen Wartebereich, der nach diesem Prinzip organisiert ist, nennt man **Queue (Warteschlange)**.
- In der Mensa-Küche werden die Teller zunächst gestapelt, um anschließend–in umgekehrter Reihenfolge, von oben nach unten–gespült zu werden[2]. Dieses Prinzip heißt *last in, first out (LIFO)*. Einen Wartebereich, der nach diesem Prinzip organisiert ist, nennt man **Stack (Stapel, oder auch: Kellerspeicher)**.
- Beim Arzt werden bestimmte Patienten eher behandelt als andere. Treffen wir für den Moment die Annahme, dass der Arzt diejenigen Patienten bevorzugt, die diesen nur kurz beanspruchen. Dieses Prinzip heißt *shortest job first (SJF)*. Einen Wartebereich, der nach diesem Prinzip organisiert ist, nennt man **Priority Queue (Vorrangswarteschlange)**.
- In den Prozessoren werden auch in der Regel ein abgewandeltes Round Robin (RR)-Verfahren eingesetzt. Bei RR-Verfahren werden die Prozesse (oder auch die Aufgaben genannt) wie mit einem FIFO-Prinzip bearbeitet. Der Unterschied ist, dass jede Aufgabe jeweils für eine kurze Zeitscheibe bearbeitet wird. Dabei

[2] Dabei handelt es sich selbstverständlich um eine stark vereinfachende Darstellung.

wird die erste Aufgabe in der Warteschlange für eine Zeitscheibe bearbeitet. Wie vielen Zeiteinheiten eine Zeitscheibe entspricht, ist frei wählbar. Anschließend wird sie am Ende der Warteschlange eingereiht, falls ihre Bearbeitung noch nicht abgeschlossen ist. Dieses wird dann wiederum bei allen Aufgaben fortgesetzt, bist ihre Bearbeitung abgeschlossen ist. Dieses Verfahren wird als **Round Robin (RR)-Wartebereich** bezeichnet.

Sie sehen also: Die Art des Wartebereichs bestimmt die Reihenfolge, in der wartende Aufträge bearbeitet werden bzw. den Wartebereich verlassen.

A.2 Aufträge

Ein Wartebereich enthält **Aufträge verschiedener Art**. Dabei hat jeder Auftrag eine bestimmte *Komplexität*, die den Aufwand zur Bearbeitung des Auftrags repräsentiert (beispielsweise die Anzahl der Artikel im Einkaufswagen). Um diese Komplexität in einen konkreten Zeitbedarf (z.B. die Zeit, um den gesamten Einkaufswagen abzukassieren) umzuwandeln, bietet jeder Auftrag eine Methode mit der Signatur `public int process()` an. Die Art und Weise, wie diese Methode Komplexität in Zeitbedarf umgerechnet, hängt vom Typ des Jobs ab. Bei Aufträgen vom Typ `SimpleJob` entspricht der Zeitbedarf d_{simple} einfach der Komplexität k des Auftrags: $d_{simple} = k$. Bei Aufträgen vom Typ `ComplexJob` berechnet sich der Zeitbedarf $d_{complex}$ als quadrierte Komplexität des Auftrags: $d_{complex} = k^2$. Unabhängig vom Typ besitzt jeder Auftrag einen *Namen*. Jeder Auftrag hat zudem eine *Ankunftszeit*. Das ist die Zeit, zu dem der Auftrag im Wartebereich ankommt. Zeit ist im Kontext dieser Aufgabe dimensionslos und wird in ganzen Zahlen (`int` genügt) gemessen.

A.3 Wartebereiche

Es gibt **Wartebereiche verschiedener Art**, wie zuvor in Abschn. A.1 eingeführt. Unabhängig vom Typ besitzt ein Wartebereich mindestens die folgenden Methoden:

- Die **`add`-Methode** fügt einen übergebenen Auftrag dem Wartebereich hinzu.
- Die **`remove`-Methode** entnimmt dem Wartebereich den nächsten Auftrag, indem sie den Auftrag aus dem Wartebereich entfernt und als Ergebnis des Methodenaufrufs zurückliefert. Welcher Auftrag als nächstes an der Reihe ist, hängt vom Typ des Wartebereichs ab (vgl. Abschn. A.1). Sollten bei einem SJF-Wartebereich mehrere Aufträge den gleichen Zeitbedarf haben, wird aus diesen Aufträgen der Auftrag mit der kleinsten Ankunftszeit ausgewählt, um als nächstes bedient zu werden.

A.4 Simulierte Bedieneinheit

Eine simulierte Bedieneinheit besitzt genau einen Wartebereich, dessen Typ beim Programmstart bestimmt wird (vgl. Abschn. A.6.2).

Nach Programmstart liest die simulierte Bedieneinheit eine Szenariobeschreibung (vgl. Abschn. A.6.3) ein. Das beschriebene Szenario wird anschließend simuliert. Was das bedeutet, ist im Folgenden beschrieben.

Die simulierte Bedieneinheit besitzt das Attribut `time` vom Typ `int`. Dieses Attribut wird schrittweise inkrementiert. Sobald `time` der Ankunftszeit eines Auftrags entspricht, wird der Auftrag dem Wartebereich hinzugefügt. Der erste in den Wartebereich eingeführte Auftrag gelangt direkt vom Wartebereich in die Bedieneinheit. Nachfolgende Aufträge warten im Wartebereich bis die Bedieneinheit frei wird. Sobald ein Auftrag in die Bedieneinheit gelangt, wird zunächst der Zeitbedarf berechnet (vgl. Abschn. A.2). Dieser berechnete Zeitbedarf verringert sich mit jeder Inkrementierung von `time` um 1 bis der Auftrag schließlich „abgearbeitet“ ist. Die Simulation endet sobald alle in der Eingabedatei spezifizierten Aufträge auf diese Weise abgearbeitet wurden.

Beachten Sie auch folgende Hinweise:

- zu jedem Zeitschritt werden zuerst eventuell ankommende Aufträge in den Wartebereich eingereiht, und

erst anschließend entschieden welcher Auftrag bearbeitet wird, sollte die Bedieneinheit zu dem Zeitpunkt frei sein.

- Sie dürfen annehmen, dass zwei Aufträge niemals zum gleichen Zeitpunkt im Wartebereich ankommen.

Beispielabläufe finden Sie in Abschn. A.8.

A.5 Aufgabenstellung

Implementieren Sie:

- *Aufträge* vom Typ *SimpleJob* und *ComplexJob* wie in Abschn. A.2 beschrieben.
- *Wartebereiche* vom Typ *Queue*, *Stack*, *Priority Queue*, sowie *Round-Robin-Prinzip* wie in Abschn. A.1 bzw. Abschn. A.3 beschrieben. Beachten Sie dabei die Implementierungshinweise.
- die *Simulierte Bedieneinheit* wie ein Abschn. A.4 beschrieben. Ihr Programm soll über genau eine simulierte Bedieneinheit verfügen.
- die Funktionalität, beim Programmstart den Typ des Wartebereichs als Kommandozeilenargument zu übergeben, wie in Abschn. A.6.2 beschrieben.
- die Funktionalität, Szenariobeschreibungen aus einer Datei einzulesen, wie in Abschn. A.6.1 beschrieben.
- die Funktionalität, den Verlauf der Simulation durch Terminalausgaben zu beobachten, wie in Abschn. A.7 beschrieben.

Benutzen Sie bei der Modellierung Ihres Programms an geeigneten Stellen die aus der Vorlesung bekannten Vererbungskonzepte. Dokumentieren Sie die öffentliche Schnittstelle Ihrer Klassen mittels Javadoc[3].

A.5.1 Implementierungshinweise

Zur Implementierung der verschiedenen Wartebereiche dürfen Sie in dieser Aufgabe Klassen aus dem Paket `java.util`[4] nutzen. Nachfolgende Vorschläge für die ersten drei Wartebereiche dienen zur Orientierung und sind nicht verpflichtend. Achten Sie aber darauf, für jeden Wartebereich eine geeignete Datenstruktur als unterliegende Datenstruktur für Ihre eigene Wartebereichsimplementierung zu nutzen.

- Für die Implementierung eines FIFO-Wartebereichs wird die Nutzung des Interface `java.util.Queue` in Verbindung mit einer `java.util.LinkedList` empfohlen.
- Für die Implementierung eines LIFO-Wartebereichs wird die Nutzung von `java.util.Stack` empfohlen.
- Für die Implementierung eines SJF-Wartebereichs kann eine `java.util.LinkedList` genutzt werden. Um den kürzesten Auftrag innerhalb einer Liste zu bestimmen, müssen Sie jedoch über die ganze Liste iterieren. Besser, aber schwieriger in der Handhabung, ist daher die Nutzung des Interface `java.util.Queue` in Verbindung mit einer `java.util.PriorityQueue`.

A.6 Eingabe des Programms

Ihr Programm akzeptiert zwei Arten von Kommandozeilenparametern.

[3] `http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html`
[4] die Javadoc-Dokumentation finden Sie unter `http://docs.oracle.com/javase/7/docs/api/`

A.6.1 Bestimmung der Szenariodatei

Als erstes Kommandozeilenargument nimmt Ihr Programm einen Pfad auf eine Textdatei entgegen. Diese Textdatei beinhaltet die Szenariobeschreibung, wie in Abschn. A.6.3 beschrieben. Dieses Kommandozeilenargument muss immer übergeben werden.

A.6.2 Bestimmung des Wartebereichtyps

Als zweites Kommandozeilenargument nimmt Ihr Programm den Namen des Wartebereichtyps entgegen, der für die Simulation genutzt werden soll. Dies bedeutet:

- Wird das Programm mit `waitingarea=fifo` gestartet, wird der Queue-Wartebereich ausgewählt.
- Wird das Programm mit `waitingarea=lifo` gestartet, wird der Stack-Wartebereich ausgewählt.
- Wird das Programm mit `waitingarea=sjf` gestartet, wird der Priority-Queue-Wartebereich ausgewählt.
- Wird das Programm mit `waitingarea=rr` gestartet, wird das Round-Robin-Verfahren ausgewählt. Zusätzlich erwartet der Wartebereich RoundRobin eine Zeitscheibe mit `timeSlice=x`, während $x \in \mathbb{N}^+$.

Wird das Programm ohne zweites (bzw. drittes) Kommandozeilenargument gestartet, wird der Queue-Wartebereich ausgewählt. Wird dem Programm ein ungültiger Wartebereichtyp übergeben, wird eine Fehlermeldung ausgegeben. Wird dem Programm ein ungültiges Kommandozeilenargument übergeben, reagiert dieses ebenso mit einer Fehlermeldung.

A.6.3 Aufbau der Szenariodatei

Die Szenariodatei enthält eine oder mehrere Zeilen. Jede Zeile beschreibt einen Auftrag. Ein Auftrag wird (in dieser Reihenfolge) beschrieben durch dessen Name, Typ, Ankunftszeitpunkt und Komplexität. Der Auftragstyp wird repräsentiert durch die Zeichenkette `simple` oder `complex`. Aufeinanderfolgende Elemente einer Zeile sind durch ein Komma separiert. Weder vor noch nach einem Komma tauchen Leerzeichen auf. Dabei darf angenommen werden, dass Auftragsnamen weder Kommata noch Doppelpunkte[5] enthalten.

Jede Zeile hat somit folgende Form: `Auftragsname,Auftragstyp,Ankunftszeitpunkt,Auftragskomplexität`

Folgendes Beispiel präsentiert die Programmeingabe wie in Tab. 1 dargestellt (wobei die Auftragsnamen verschieden sind):

```
Task1,simple,0,5
Task2,simple,3,4
Task3,simple,5,3
Task4,simple,6,5
```

Lesen Sie eine entsprechend gestaltete Datei geeignet ein.

A.7 Ausgabe des Programms

Ihr Programm gibt zu jedem Zeitpunkt der Simulation den Zustand der simulierten Bedieneinheit und dessen Wartebereich auf die Konsole aus (mittels `Terminal`-Klasse). Das Ausgabeformat am Beispiel des Szenarios aus Abschn. A.6.3 sehen Sie weiter unten. Vor dem ersten Doppelpunkt steht der aktuelle Zeitpunkt. Direkt dahinter steht der Name des Auftrags, der zu diesem Zeitpunkt bearbeitet wird, gefolgt von der für diesen Auftrag verbleibenden Bearbeitungszeit in Klammern. Daraufhin folgt eine Komma und die Zeichenkette „`Waiting:`“. Dahinter wird der Zustand des Wartebereichs ausgegeben als komma-separierte Liste der wartenden Aufträge

[5] Doppelpunkte vermeiden wir in Auftragsnamen aufgrund des geforderten Ausgabeformats

(ohne den Auftrag, der sich in der Bedieneinheit befindet). Auch hier schließt sich an jeden Auftrag dessen verbleibende Bearbeitungszeit in Klammern an. Die Reihenfolge diese Liste spiegelt die Reihenfolge der Aufträge innerhalb des Wartebereichs wieder, wobei der am weitesten links stehende Auftrag derjenige Auftrag ist, der als nächstes prozessiert werden wird. Ist der Wartebereich leer, soll als Zustand `empty` ausgegeben werden.

```
0:Task1(5),Waiting:empty
1:Task1(4),Waiting:empty
2:Task1(3),Waiting:empty
3:Task1(2),Waiting:Task2(4)
4:Task1(1),Waiting:Task2(4)
5:Task2(4),Waiting:Task3(3)
6:Task2(3),Waiting:Task3(3),Task4(5)
7:Task2(2),Waiting:Task3(3),Task4(5)
8:Task2(1),Waiting:Task3(3),Task4(5)
9:Task3(3),Waiting:Task4(5)
10:Task3(2),Waiting:Task4(5)
11:Task3(1),Waiting:Task4(5)
12:Task4(5),Waiting:empty
13:Task4(4),Waiting:empty
14:Task4(3),Waiting:empty
15:Task4(2),Waiting:empty
16:Task4(1),Waiting:empty
```

A.8 Beispielabläufe

Im Folgenden betrachten wir einen Beispielablauf je Wartebereichstyp für das in Tabelle 1 beschriebene Szenario. Mit Ausnahme der Auftragsnamen entspricht dieses Szenario dem in Abschn. A.6.3 eingeführten Beispielszenario. Anhand dieses Beispiels wird nachfolgend die Reihenfolge der Abarbeitung der jeweiligen Form der Wartelinien illustriert. Dabei nehmen wir an, dass alle Aufträge vom Typ `simple` sind.

Auftragsname	Ankunftszeitpunkt	Komplexität
A	0	5
B	3	4
C	5	3
D	6	5

Tabelle 1: Beispielszenario, beschrieben durch ankommende Aufträge

A.8.1 Queue

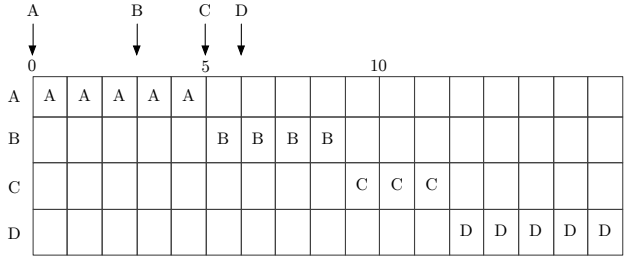

Abbildung 1: Die Bearbeitungsreihenfolge bei einer Queue-Wartelinie

A.8.2 Stack

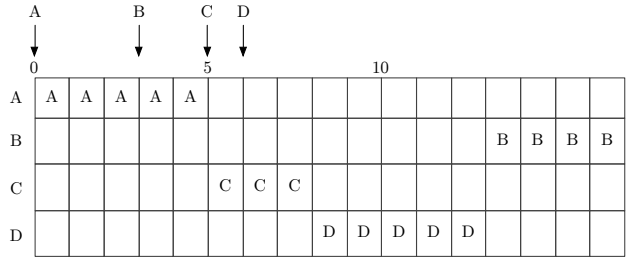

Abbildung 2: Die Bearbeitungsreihenfolge bei einer Stack-Wartebereich

A.8.3 Shortest-Job-First

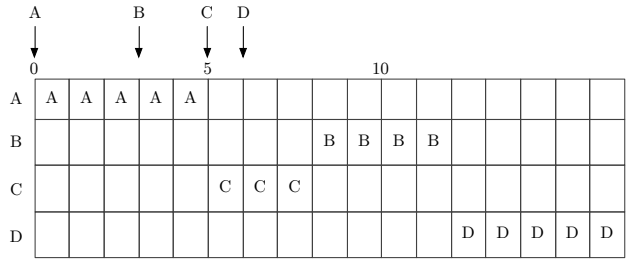

Abbildung 3: Die Bearbeitungsreihenfolge bei einer SJF-Wartebereich

A.8.4 Round-Robin

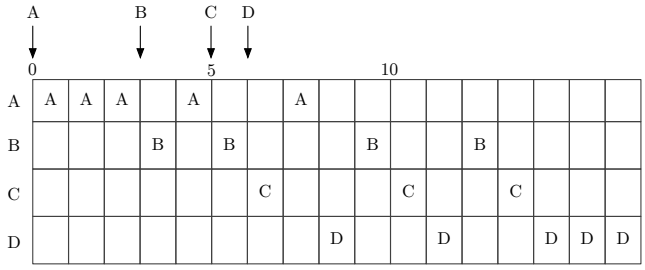

Abbildung 4: Die Bearbeitungsreihenfolge bei einem RR-Verfahren