AU01
ROBOTERFABRIK

Dokumentation

Von Rene Hollander, Samuel Schober und Simon Wortha
4AHITT
stand 01.10.2014

Inhalt

Angabe

Es soll eine Spielzeugroboter-Fabrik simuliert werden. Die einzelnen Bestandteile des Spielzeugroboters (kurz Threadee) werden in einem Lager gesammelt. Dieses Lager wird als Verzeichnis und die einzelnen Elementtypen werden als Files im Betriebssystem abgebildet. Der Lagermitarbeiter verwaltet regelmäßig den Ein- und Ausgang des Lagers um Anfragen von Montagemitarbeiter und Kunden zu beantworten. Die Anlieferung der Teile erfolgt durch Ändern von Files im Verzeichnis, eine Lagerung fertiger Roboter ebenso.

Ein Spielzeugroboter besteht aus zwei Augen, einem Rumpf, einem Kettenantrieb und zwei Armen. Die Lieferanten schreiben ihre Teile ins Lager-File mit zufällig (PRNG?) erstellten Zahlenfeldern. Die Art der gelieferten Teile soll nach einer bestimmten Zeit gewechselt werden.

Die Montagemitarbeiter müssen nun für einen "Threadee" alle entsprechenden Teile anfordern und diese zusammenbauen. Der Vorgang des Zusammenbauens wird durch das Sortieren der einzelnen Ganzzahlenfelder simuliert. Der fertige "Threadee" wird nun mit der Mitarbeiter-ID des Monteurs versehen.

Es ist zu bedenken, dass ein Roboter immer alle Teile benötigt um hergestellt werden zu können. Sollte ein Monteur nicht alle Teile bekommen, muss er die angeforderten Teile wieder zurückgeben um andere Monteure nicht zu blockieren. Fertige "Threadee"s werden zur Auslieferung in das Lager zurück gestellt.

Alle Aktivitäten der Mitarbeiter muss in einem Logfile protokolliert werden. Verwenden Sie dazu Log4J [1].

Die IDs der Mitarbeiter werden in der Fabrik durch das Sekretariat verwaltet. Es dürfen nur eindeutige IDs vergeben werden. Das Sekretariat vergibt auch die eindeutigen Kennungen für die erstellten "Threadee"s.

Beachten Sie beim Einlesen die Möglichkeit der Fehler von Files. Diese Fehler müssen im Log protokolliert werden und entsprechend mit Exceptions abgefangen werden.

Tipps und Tricks

Verwenden Sie (optional) für die einzelnen Arbeiter das ExecutorService mit ThreadPools. Achten Sie, dass die Monteure nicht "verhungern". Angeforderte Ressourcen müssen auch sauber wieder freigegeben werden.

Beispiel für Teile-Files

-- "auge.csv"
Auge,11,24,3,4,25,6,8,8,9,10,11,12,13,14,15,16,17,18,195,5
Auge,91,62,3,4,54,6,7,8,9,10,11,12,13,14,15,16,17,18,119,32
Auge,91,62,3,4,54,6,7,8,9,10,11,12,13,14,15,16,17,18,119,520

-- "rumpf.csv"

Rumpf,91,62,3,4,54,6,7,8,9,10,11,12,13,14,15,16,17,18,119,21

Beispiel für Threadee-File

-- "auslieferung.csv"
Threadee-ID123,Mitarbeiter-ID231,Auge,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,Auge,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,Rumpf,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,Kettenantrieb,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,Arm,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,Arm,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
Threadee-ID124,Mitarbeiter-ID231,Auge,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,Auge,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,Rumpf,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,Kettenantrieb,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,Arm,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,Arm,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20

Ausführung

Zu bedenken sind die im Beispiel angeführten Argumente. Diese können mit eigenem Code oder mit einer CLI-Library implementiert werden (z.B. [2]).
Alle Argumente sind verpflichtend und die Anzahl muss positiv sein. Die obere Grenze soll sinnvoll festgelegt werden. Vergessen Sie auch nicht auf die Ausgabe der Synopsis bei einer fehlerhaften Eingabe! Sollten Sie zusätzliche Argumente benötigen sind diese erst nach einer Rücksprache implementierter.

java tgm.sew.hit.roboterfabrik.Simulation --lager /verzeichnis/zum/lager --logs /verzeichnis/zum/loggen --lieferanten 12 --monteure 25 --laufzeit 10000

Resources

[1] http://logging.apache.org/log4j/2.0/manual/configuration.html
[2] http://commons.apache.org/sandbox/commons-cli2/manual/index.html

Zeit

Zeitabschätzung

Wir schätzen mit einem Arbeitsaufwand von circa 10 Stunden pro Person. Also insgesamt 30 Stunden.

Zeitaufstellung

Person	Arbeit	Zeit
Rene Hollander, Samuel Schober, Simon Wortha	Erstellung Requirements Analyse	90 Minuten (30 Minuten pro Person)
Rene Hollander, Samuel Schober, Simon Wortha	Erstellung UML	270 Minuten (90 Minuten pro Person)
Rene Hollander	Projekt aufsetzen: Maven einrichten Log4J konfigurieren	120 Minuten
Rene Hollander	Importieren der aus dem UML generieren Java Source Files und hinzufügen der Klassenspezifischen Logger	10 Minuten
Rene Hollander	Implementierung der CLI Argumente mithilfe von Commons CLI	60 Minuten
Simon Wortha	Implementierung von Office	5 Minuten
Simon Wortha	Implementierung von Employee	15 Minuten
Samuel Schober	Implementierung Supply, Supplier, Part	60 Minuten
Samuel Schober	Implementierung Watchdog	60 Minuten
Rene Hollander	Implementation Warehouser, PartType, Simulation	180 Minuten
Simon Wortha	Aktualisierung Office, Javadoc geaded	60 Minuten
Simon Wortha	Implementierung von Threadee und Javadoc	50 Minuten
Rene Hollander	Ein paar Performance improvements im Warehouser eingebaut	45 Minuten
Rene Hollander	Testfälle für Office, IntegerWrapper und Warehouser hinzugefügt	45 Minuten
Simon Wortha	Testfälle für Part, PartType	60 Minuten
Simon Wortha	Testfälle für Simulation und Supply	60 Minuten
Simon Wortha	Testfälle für Threadee	15 Minuten
Simon Wortha	Javadoc und Dokubearbeitet	30 Minuten
Rene Hollander	Fehlendes Javadoc zu Tests hinzugefügt	30 Minuten
Simon Wortha	Javadoc für Simulation und SupplyTest	30 Minuten
Rene Hollander	Weitere Tests und Test Javadoc Kommentare hinzugefügt	45 Minuten
Samuel Schober	Testfälle für Employee, Supplier	45 Minuten
Samuel Schober	Testfälle für Threadee	30 Minuten
Samuel Schober	Testfälle für Watchdog	20 Minuten
Samuel Schober	Aktualisierung aller Javadocs	30 Minuten
Samuel Schober	Fehlende Javadoc der Testfälle hinzugefügt	40 Minuten

Requirementsanalyse

Arbeiter die Roboter zusammenbauen
- Zusammenbau wird durch sortieren der Zahlen simuliert
- Holt sich für jeden Roboter Teile vom Lagermitarbeiter
- Arbeiter bekommt Mitarbeiter ID vom Sekretariat für eindeutige zuordnung
- Wenn Roboter fertig, Arbeiter holt sich ID vom Sekretariat für Threadee
- Threadee mit ID wird dem Lagermitarbeiter übergeben

Lagerarbeiter der Teile ins Lager bringt und holt und fertige Threadees lagert
- Lagerarbeiter holt auf Anfrage von Arbeiter Teile aus dem Lager
- Lieferungen werden richtig sortiert (Nach Typ: Auge, Rumpf, ...) im Lager aufbewahrt
- Fertige Threadees werden im Lager aufbewahrt

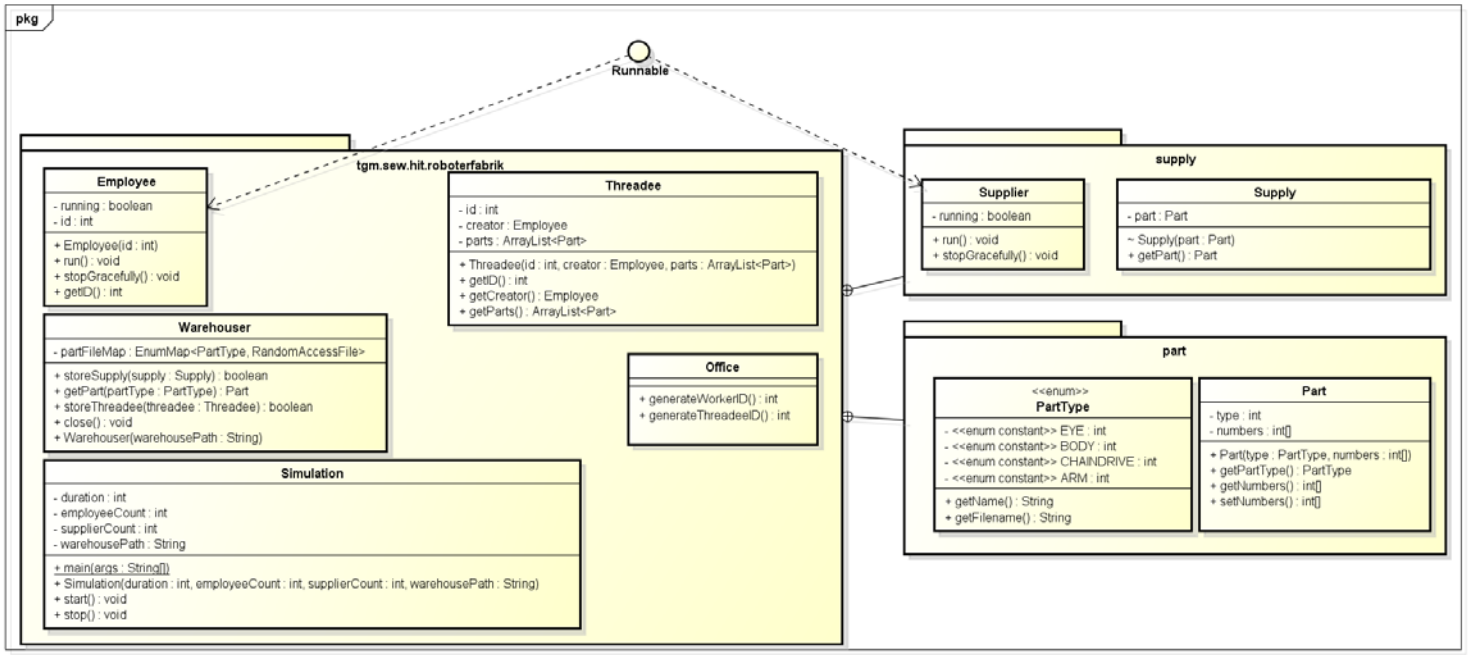
Zulieferer der Lagermitarbeiter verschiedene Teile bringt
- Lieferant liefert z.b: 10 Augen mit zufälligen Zahlen, die später vom Arbeiter sortiert werden
- Lieferant wechselt die Teile nach einer bestimmten Zeit (20 Stück?)

Sekretariat verteilt IDs an Arbeiter und Threadees
- Jeder Arbeiter bekommt eindeutige ID
- Jeder Threadee bekommt eindeutige ID

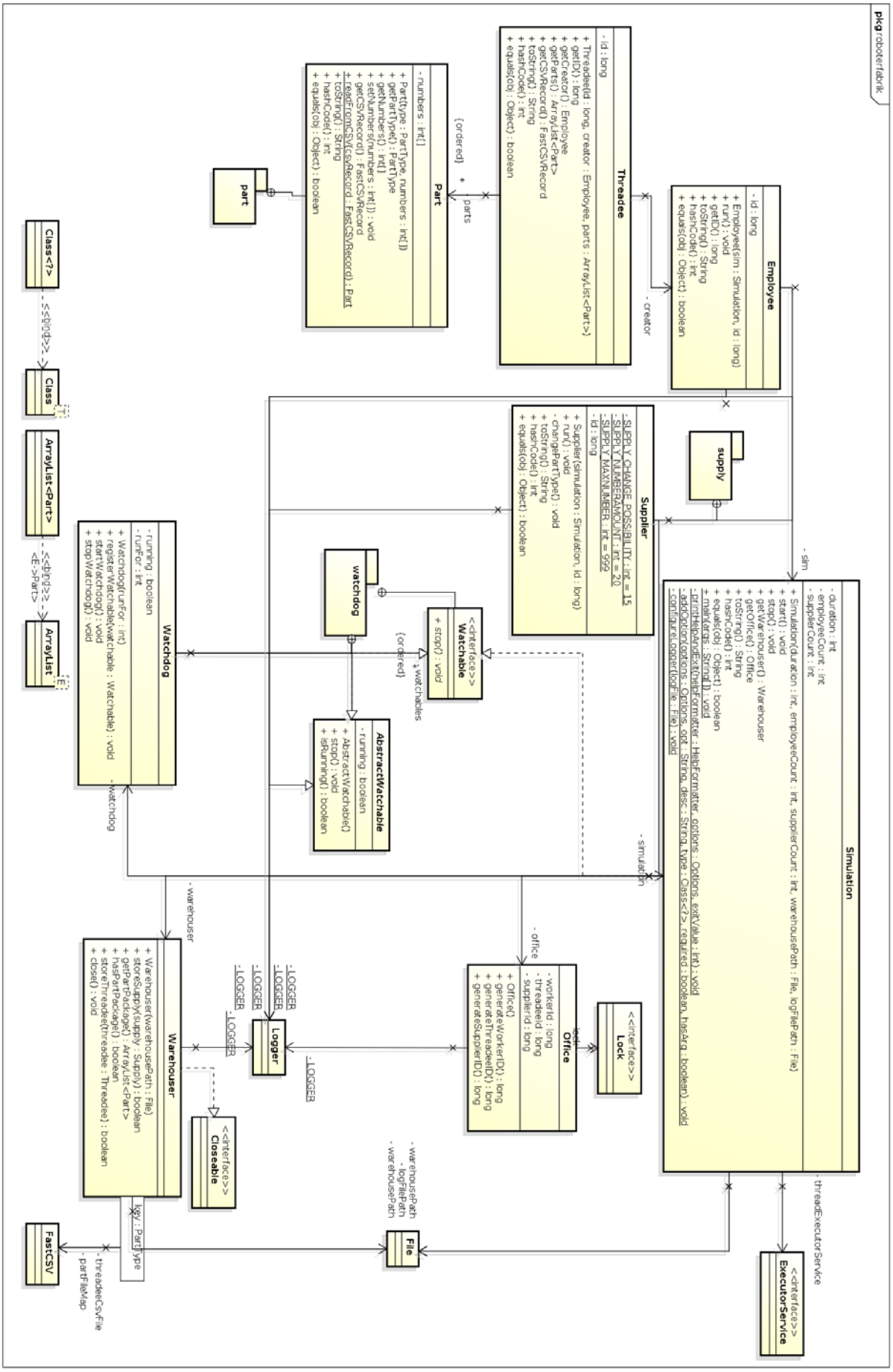
Jeder Arbeiter hat eigenen Thread
Jeder Lieferant hat eigenen Thread
Es gibt nur 1 Lagermitarbeiter
Es gibt nur 1 Sekretariat

UML-Diagramm

Unser erstes Diagramm nach dem wir auch gearbeitet haben:



Nach einigen Änderungen haben wir ein neues generiert um alle Klassen und Funktionen dabei zu haben: (siehe nächste Seite)


pkg roboterfabrik
Simulation
- duration : int
- employeeCount : int
- supplierCount : int
+ Simulation(duration : int, employeeCount : int, supplierCount : int, warehousePath : File, logFilePath : File)
+ start() : void
+ stop() : void
+ getWarehouser() : Warehouser
+ getOffice() : Office
+ toString() : String
+ hashCode() : int
+ equals(obj : Object) : boolean
+ main(args : String[]) : void
- printHelpAndExit(helpFormatter : HelpFormatter, options : Options, exitValue : int) : void
- addOption(options : Options, opt : String, desc : String, type : Class<?>, required : boolean, hasArg : boolean) : void
- configureLogger(logFile : File) : void
- sim
- threadExecutorService
- simulation
Employee
- id : long
+ Employee(sim : Simulation, id : long)
+ run() : void
+ getID() : long
+ toString() : String
+ hashCode() : int
+ equals(obj : Object) : boolean
- creator
Threadee
- id : long
+ Threadee(id : long, creator : Employee, parts : ArrayList<Part>)
+ getID() : long
+ getCreator() : Employee
+ getParts() : ArrayList<Part>
+ getCSVRecord() : FastCSVRecord
+ toString() : String
+ hashCode() : int
+ equals(obj : Object) : boolean
{ordered}
parts
Part
- numbers : int[]
+ Part(type : PartType, numbers : int[])
+ getPartType() : PartType
+ getNumbers() : int[]
+ setNumbers(numbers : int[]) : void
+ getCSVRecord() : FastCSVRecord
+ readFromCSV(csvRecord : FastCSVRecord) : Part
+ toString() : String
+ hashCode() : int
+ equals(obj : Object) : boolean
part
supply
Supplier
- SUPPLY_CHANGE_POSSIBILITY : int = 15
- SUPPLY_NUMBERAMOUNT : int = 20
- SUPPLY_MAXNUMBER : int = 999
- id : long
+ Supplier(simulation : Simulation, id : long)
+ run() : void
- changePartType() : void
+ toString() : String
+ hashCode() : int
+ equals(obj : Object) : boolean
watchdog
<<interface>>
Watchable
+ stop() : void
{ordered}
watchables
Watchdog
- running : boolean
- runFor : int
+ Watchdog(runFor : int)
+ registerWatchable(watchable : Watchable) : void
+ startWatchdog() : void
+ stopWatchdog() : void
- watchdog
AbstractWatchable
- running : boolean
+ AbstractWatchable()
+ stop() : void
+ isRunning() : boolean
Class<?>
- <<bind>>
Class
T
ArrayList<Part>
- <<bind>>
<E->Part>
ArrayList
E
- warehouser
Warehouser
+ Warehouser(warehousePath : File)
+ storeSupply(supply : Supply) : boolean
+ getPartPackage() : ArrayList<Part>
+ hasPartPackage() : boolean
+ storeThreadee(threadee : Threadee) : boolean
+ close() : void
- office
Office
- workerId : long
- threadeeId : long
- supplierId : long
+ Office()
+ generateWorkerID() : long
+ generateThreadeeID() : long
+ generateSupplierID() : long
<<interface>>
Lock
- LOGGER
Logger
<<interface>>
Closeable
- warehousePath
- logFilePath
- warehousePath
File
key : PartType
FastCSV
- threadeeCsvFile
- partFileMap
<<interface>>
ExecutorService

Abgabe

Das Repository ist hier zu erreichen: https://github.com/ReneHollander/au01_roboterfabrik

Things I have done

Hollander:

Umgesetzt habe ich Simulation, Warehouser und PartType. Wenn irgendwo etwas nicht gepasst hat, habe ich es passend gemacht. Bei Fragen habe ich meinen Teamkameraden geholfen und ihnen Sachen die sie bis dato nicht wussten erklärt (Enums).

Schober:

Von mir wurden die Klassen Supply, Supplier und Part implementiert. Da ich leider gesundheitlich nicht in der Lage dazu war an dem Unterricht mit dem Thema Watchdog teilzunehmen, half mir mein Gruppenleiter Rene Hollander bei der Implementierung der Watchdog Klasse. Ich habe mich auch noch darum gekümmert dass die Javadoc richtig ausgebessert wurde und Unklarheiten in der Javadoc beseitigt wurden.

Wortha:

Ich habe die Klassen Office, Threadee und Employee implementiert. Ich hatte relativ wenig Probleme, wenn welche aufgetreten sind konnte mir mien Teamleiter helfen. Natürlich wurden hier und da auch sachen „nach gegoogle“.
Weiters habe ich mich um Teile der Dokumenation und der Diagramme gekümmert.

Lessons Learned

Hollander:

Gelernt habe ich, dass weniger syncronized mehr sind. Wichtig ist vorallem was man syncronisiert. Anfangs war die Roboterfabrik sehr langsam (maximal 10 Threadees in 10 Sekunden). Durch ein paar überlegungen mit dem Lagermitarbeiter und dem entfernen einiger syncronized bzw einer besseren Aufteilung der syncronisierung konnte ich schnell über 3500 Threadees in nur 10 Sekunden bauen. Dadurch dass ich das Projekt aufgesetzt habe, habe ich ein paar Sachen bei Log4J wie dynamisches Setzen des Log Pfades oder dynamisch zwischen Debug und Info Level zu schalten herausgefunden. Auch bin ich nun mit der Verwendung von Commons CLI vertraut.

Schober:

Ich habe gelernt wie man effizient mit Threads arbeitet sodass es zu keinen Komplikationen zwischen den Threads gibt (wenn sie z.B. gleichzeitig auf eine Datei zugreifen wollen). Weiters habe ich über die Funktion/Ausführung des Watchdog gelernt.

Wortha:

Ich habe gelernt mit Threads zu arbeiten (mit synchronized, lock usw.) und die Anwendung von Log4J.

Quellen

Keine (Leider weiß ich auch nicht was dazu zählt, also zum Beispiel Verwendete 3rd Party Libarys und der gleichen)