ROBOTERFABRIK-PROTOKOLL

A01-SEW

Alexander Kölbl, Thomas Taschner & Michael Weinberger 4AHITT am 1.10.2014

Inhaltsverzeichnis

Inhaltsangabe 2
Aufgabenstellung 2
Designüberlegung 3
Aufwandeinschätzung 3
Endzeitaufteilung 3
Arbeitsdurchführung & Klassenübersicht 3
Arbeitsdurchführung 3
Resultate 3
Niederlagen 3
Klassenübersicht 4
Testbericht 6
Quellenangabe 8

Inhaltsangabe

Das vollendete Programm simuliert eine Roboterfabrik und deren Arbeitsablauf.

Die Roboterfabrik setzt sich zusammen aus:

- einem Lager (in denen sich die Teile des Roboters befinden)
- einem Lagermitarbeiter (verwaltet die Anfragen an das Lager)
- einem Montagemitarbeiter (baut den Roboter zusammen)
- einem Sekretariat (erstellt IDs für die Mitarbeiter & Roboter)

Ziel des Programm ist es das der Ablauf der Roboterfabrik fehlerfrei & nach vorgegeben Ablauf (siehe Aufgabenstellung) funktioniert.

Aufgabenstellung

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.

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.

Designüberlegung

Das UML-Diagramm wurde in Zusammenarbeit aller an dem Projekt beteiligten Personen durchdacht und erzeugt.

Anschließend wird das UML-Diagramm mit dieser Klassenstruktur in das Programm übertragen.

Aufwandeinschätzung

An diesem Projekt beteiligen sich 3 Personen. Der geschätzte Zeitaufwand beträgt 20 Stunden.

Geschätzter Aufwand der Projektphasen:

Projektphase	geschätzter Zeitaufwand
Planungsphase (Inhalterfassung, UML, etc.)	5 Stunden
Implementierungsphase	10 Stunden
Test- und Korrekturphase	5 Stunden

Endzeitaufteilung

Folgende Arbeitszeiten wurden von den Teammitgliedern vollbracht:

Teammitglied	Gesamte Arbeitszeit am Projekt
Alexander Kölbl	13 Stunden
Thomas Taschner	18 Stunden
Michael Weinberger	21 Stunden

Zusammen ergibt das eine Arbeitszeit von **52 Stunden** an diesem Projekt.

Arbeitsdurchführung & Klassenübersicht

Arbeitsdurchführung

Resultate

Niederlagen

Wie aus der Aufwandeinschätzung und Endzeitaufteilung gut ersichtlich ist, hat sich unser Team deutlich bei der Zeiteinschätzung verkalkuliert. Wir sind von einer viel kürzeren Arbeitszeit ausgegangen. Daher wurde es in der letzten Woche der Projektlaufzeit ziemlich zeitaufwändig. Aus dieser Erfahrung werden wir lernen, sodass dieser Fall bei nächsten Projekten nicht mehr passiert.

Klassenübersicht

Assembler

Diese Klasse simuliert den Aufbau eines Roboters.

Konstruktor: Assembler (Secretary s, Storage st)

In dieser Klasse befinden sich folgende Methoden (keine Parameter angegeben):

- **run()**: startet den Thread.
- **build()**: simuliert das zusammenbauen des Roboters
- **toInt()**: speichert String-Werte eines String Arrays in einen Int Array als Int-Werte.
- **sort()**: sortiert die Werte eines Int Arrays in aufsteigender Reihenfolge

Ausführliche Information zu den Klassen (und Parameter) in der JavaDoc.

Furnisher

Diese Klasse kontrolliert den Inhalt der csv-Files, wenn aufgerufen.

Konstruktor: enthält keinen Konstruktor

In dieser Klasse befinden sich folgende Methoden (keine Parameter angegeben):

- **run()**: kontrolliert die csv-Files (wird von einem Thread ausgeführt)
- **getNumber()**: gibt eine Zufallszahl von 0 bis 999 zurück
- **OneRow()**: wie die Methode write(), nur ohne Zeilenumbruch am Ende (kein \n)

Ausführliche Information zu den Klassen (und Parameter) in der JavaDoc.

CLI

Diese Klasse simuliert ein Console Line Interface (CLI).

Konstruktor: CLI()

In dieser Klasse befinden sich folgende Methoden (keine Parameter angegeben):

- **run()**: startet die CLI
- **help()**: Hilfe für den User
- **welcome()**: Begrüßungsnachricht
- **check()**: kontrolliert den User-Input
- **getArguments()**: getter Methode (für Arguments)
- **setArguments()**: setter Methode (für Arguments)
- **getLagerVerzeichnis()**: getter Methode (für lagerVerzeichnis)
- **setLagerVerzeichnis()**: setter Methode (für lagerVerzeichnis)
- **getLogVerzeichnis()**:getter Methode (für logVerzeichnis)
- **setLogVerzeichnis()**:setter Methode (für logVerzeichnis)
- **getLieferantenAnzahl()**:getter Methode (für lieferantenAnzahl)
- **setLieferantenAnzahl()**:setter Methode (für lieferantenAnzahl)

- **getMonteurAnzahl()**: getter Methode (für monteurAnzahl)
- **setMonteurAnzahl()**: setter Methode (für monteurAnzahl)
- **getLaufzeit()**:getter Methode (für laufzeit)
- **setLaufzeit()**:setter Methode (für laufzeit)

Ausführliche Information zu den Klassen (und Parameter) in der JavaDoc.

IO

Diese Klasse vereinfacht das lesen bzw. schreiben der csv-files.

Konstruktor: IO(String path)

In dieser Klasse befinden sich folgende Methoden (keine Parameter angegeben):

- **write()**: schreibt einen String-Wert in eine Datei -> Roboter in Storage speichern
- **writeRow()**: schreibt einen String-Wert in eine Datei (in einer Zeile) -> Roboter in Storage speichern
- **overWrite()**: überschreibt vorhandene Werte -> Roboter in Storage speichern
- **read()**: gibt ein File als String zurück -> Roboter aus Storage holen
- **getWorkingDir()**: gibt das Verzeichnis zurück, indem der Code gestartet wurde.
- **CheckFiles()**: leere csv-Files werden im working directory erstellt.
- **checkDir()**: kontrolliert, ob das Verzeichnis vorhanden ist.

Ausführliche Information zu den Klassen (und Parameter) in der JavaDoc.

Secretary

Diese Klasse vergibt die IDs für Threadees und Mitarbeiter

Konstruktor: Secretary(int amount)

In dieser Klasse befinden sich folgende Methoden (keine Parameter angegeben):

- **getID()**: gibt eine zufällige, einzigartige Nummer zurück

Ausführliche Information zu den Klassen (und Parameter) in der JavaDoc.

Storage

Diese Klasse stellt dem Monteur die Teile des Roboters zu Verfügung.

Konstruktor: Storage()

In dieser Klasse befinden sich folgende Methoden (keine Parameter angegeben):

- **deliver()**: liefert eine Reihe von Teile, gespeichert in einem Array

Ausführliche Information zu den Klassen (und Parameter) in der JavaDoc.

UniqueNumbers

Diese Klasse erzeugt zufällige, einzigartige Zahlen.

Konstruktor: UniqueNumbers()

In dieser Klasse befinden sich folgende Methoden (keine Parameter angegeben):

- **add()**: fügt eine neue zufällige Zahl dem HashSet hinzu.
- **add()**: fügt eine vorgegeben Anzahl an zufälligen Zahlen dem HashSet hinzu.
- **getRandomNumbers()**: gibt ein Array mit den Zufallszahlen zurück.

Ausführliche Information zu den Klassen (und Parameter) in der JavaDoc.

Watchdog

Implementiert einen Watchdog für das Projekt.

Testbericht

Folgende Test-Methoden wurden geschrieben:

In der Klasse TestCLI befinden sich folgende Test-Methoden:

- **test()**
- **welcome()**
- **help()**
- **runOk()**
- **runStringIndexOutOfBoundException()**
- **runNumberFormatException()**
- **getLagerVerzeichnis()**
- **getLogVorzeichnis()**
- **getLieferantenAnzahl()**
- **getMonteurAnzahl()**
- **getLaufzeit()**

In der Klasse TestFurnisher befinden sich folgende Test-Methoden:

- **testfillUp()**
- **testOneRow()**

In der Klasse TestIO befinden sich folgende Test-Methoden:

- **setup()**
- **testgetWorkingDirectory()**
- **testRead()**
- **testReadFileNotFoundException()**
- **testWrite()**

- **testWriteIOException()**
- **testWriteNullPointerException()**
- **testCheckfiles()**
- **testCheckfilesIOException()**
- **testCheckdir()**
- **testCheckdirother()**

In der Klasse TestAssembler befinden sich folgende Test-Methoden:

- **testSort()**
- **testtoInt(()**
- **testBuild()**

In der Klasse TestSecretary befinden sich folgende Test-Methoden:

- **setup()**
- **testGetID()**
- **testGetID100times()**

In der Klasse TestStorage befinden sich folgende Test-Methoden:

- **setup()**
- **testDeliverArm()**
- **testDeliverEye()**
- **testDeliverChaindrive()**
- **testDeliverTorso()**
- **testDeliverFalse()**

In der Klasse TestUniqueNumbers befinden sich folgende Test-Methoden:

- **setup()**
- **addOneNumber()**
- **addMoreNumbers()**
- **addTooFewNumbers()**
- **addTooFewNumbers2()**
- **addLotsOfNumber()**

In der Klasse TestWatchDog befinden sich folgende Test-Methoden:

- **setup()**
- **add()**
- **addNull()**
- **killAll()**
- **run()**
- **zeroRuntime()**
- **negativeRuntime()**

Quellenangabe

http://logging.apache.org/log4j/2.0/manual/configuration.html

➔ Informationen zu Log4j

http://commons.apache.org/sandbox/commons-cli2/manual/index.html

➔ Informationen zu CLI

http://stackoverflow.com/

➔ Forum

http://docs.oracle.com/javase/8/docs/api/

➔ Java API

https://community.oracle.com/community/java

➔ Forum