University of Zurich[UZH]

s.e.a.l. software evolution & architecture lab

Informatik I – Eprog HS12

Übung 7

1 Aufgabe: Logger

1.1 Lernziele

1. Verwendung von statischen Methoden, von privaten Konstruktoren und von Arrays üben.

1.2 Aufgabenstellung

Bei grossen Programmen kann die Menge an Konsolenausgaben sehr gross und damit sehr unübersichtlich werden. In der Regel wird bei der Entwicklung sowie bei der Fehlersuche eine grössere Menge an Log-Nachrichten benötigt, als im normalen Betrieb der Software. Um die Menge an Log-Nachrichten bedarfsgerecht regulieren zu können, verwendet man in Softwareprojekten in der Regel eine spezielle Logger Klasse, an Stelle von direkten *System.out.println()* Aufrufen. Einen Logger kann man auf verschiedene Level einstellen. Ebenfalls wird jeder an den Logger übergebenen Nachricht ein Level zugeordnet. Abhängig davon, auf welchen Level der Logger gestellt ist, wird einen Nachricht ausgegeben oder auch nicht.

Das Ziel dieser Aufgabe ist es, eine eigene Logger Klasse zu implementieren.

Der Logger soll folgende Log-Level unterstützen:

ERROR, *INFO*, *DEBUG* und *TRACE*

Dabei ist *TRACE* der detailierteste Level - daher, es werden alle Nachrichten angezeigt - und *ERROR* der restriktivste Level - daher, es werden nur Nachrichten vom Typ *ERROR* angezeigt. Ist ein Logger also zum Beispiel auf den Level *INFO* eingestellt, werden Nachrichten vom Typ *ERROR* und *INFO* ausgegeben, aber keine vom Level *DEBUG* und *TRACE*.

Erzeugen Sie eine Klasse *Logger* mit einer Methode *log*, an welche eine Nachricht sowie der Log-Level der Nachricht übergeben werden kann. Es soll dabei nicht direkt eine Instanz der Klasse mittels eines öffentlichen Konstruktors erstellt werden können! Es soll eine statische Methode *createLogger* geben, an welche ein Logger-Name sowie ein Log-Level übergeben werden kann. Diese Methode soll die einzige Möglichkeit für eine andere Klasse sein, eine Instanz der Klasse *Logger*

zu erhalten.

Ein Logger soll mehrfach verwendet werden können! Beim Aufruf von *createLogger* soll also geprüft werden, ob es schon einen Logger mit dem gegebenen Namen und dem gegebenen Log-Level gibt. Falls dies der Fall ist, wird eine Referenz auf die vorhandene Instanz zurückgegeben. Falls dies nicht der Fall ist, wird ein neuer Logger erzeugt.

Testen Sie Ihren Logger mit einem passenden TestDriver.

Weitere Hinweise:

1. Erzeugen Sie für die verschiedenen Log-Level je eine statische Variable in der Klasse Logger. Eine andere Klasse soll diese verwenden können, wenn Sie die Methoden *log* oder *createLogger* verwendet. Beispiel siehe Code unten.
2. Sie werden in der Klasse Logger ein Array zum speichern der vorhandenen Logger brauchen. Dieses Array soll immer genauso viele Felder wie gerade vorhandene Logger haben. Sie müssen also bei Bedarf ein neues, grösseres Array erzeugen.
3. Ein Aufruf der Methode *log* soll eine Ausgabe des Formats <*Logger-Name*> *"["<Log-Level-Der-Nachricht>"]" ">"* <*Nachricht*> erzeugen. Beispiel: *OtherClass [INFO] > A log message*
4. Sichern Sie ab, dass nur Log-Level übergeben werden können, die auch existieren. Falls an die *log* Methode ein invalider Log-Level übergeben wird, soll keine Ausgabe erfolgen. Falls an die *createLogger* Methode ein ungültiger Log-Level übergeben wird, soll eine Fehlermeldung auf der Konsole erscheinen und *null* wird zurückgegeben.
5. Beispiel-Code:

```
1 public class Logger{
2    [..]
3    public static Logger createLogger(String name, int level){}
4    public void log(String message,int level){}
5    [..]
6 }
```

```
1 public class OtherClass{
2    private Logger logger = Logger.createLogger("OtherClass",Logger.DEBUG);
3
4    public void someMethod(){
5       logger.log("A log message",Logger.INFO);
6    }
7 }
```

2 Aufgabe: Formater

2.1 Lernziele

1. Wiederholung und Vertiefung der Nutzung von Strings.

2.2 Aufgabenstellung

Schreiben Sie eine Klasse *Formater*. Diese hat eine statische Methode *format*, welche einen Template-String und ein String-Array als Parameter erhält. Im Template-String können Platzhalter der Art *{0}* enthalten sein. Die Platzhalter werden durch Elemente des String-Arrays ersetzt. Die Methode liefert als Resultat den Template-String mit ersetzten Platzhaltern.

Beispiel: *Formater.format("Der erste Buchstabe im {0}{1}{2} ist {0}!",new String[]{"A","B","C"});*
Dieser Aufruf würde den folgenden String liefern: *Der erste Buchstabe im ABC ist A!*.

Der Platzhalter *{0}* wird also durch den ersten Wert im Array ersetzt, der Platzhalter *{1}* durch den zweiten Wert und so weiter.

Schreiben Sie einen Test-Driver. Erweitern Sie die Klasse *Logger* um eine Methode *logWithFormater*. Diese soll das Gleiche tun wie die Methode *log*, aber Ihren *Formater* dabei verwenden.

Hinweise:

1. Sie können davon ausgehen, dass "{" und "}" nur als Teil der Platzhalter im Template-String vorkommen.
2. Sie werden die Methoden *substring* und *indexOf* der Klasse String benötigen. Informieren Sie sich in der Java-Doc.