

**Programmierung**  
**Prof. Dr.-Ing. Nobert Fuhr**

Gudrun Fischer  
Sascha Kriewel  
programmierung@is.informatik.uni-duisburg.de

**Übungsblatt Nr. 12**

**Aufgabe 23:    Threads**

In einer Autowerkstatt werden Autos abgegeben und von den zehn dort beschäftigten Automechanikern repariert. Die Autos werden dabei in eine Warteschlange gestellt. Sobald ein Automechaniker mit dem Reparieren eines Autos fertig ist, holt er das nächste Auto aus der Warteschlange.

Die Klasse Auto ist aus früherer Übung bekannt und wird um die neue Methode `public void reparieren()` erweitert. Die Klasse Warteschlange soll wie vorgegeben implementiert sein:

Listing 1: Auto.java

```
1 public void reparieren () {
2     System.out.println ("Das_Auto_" + kennzeichen +
3         "_wird_repariert.");
4 }
```

Listing 2: Warteschlange.java

```
1 package de.unidue.is.prog.verkehr;
2 import java.util.ArrayList;
3
4 /**
5  * Diese Klasse repraesentiert eine Warteschlange
6  * fuer Autos in einer Werkstatt. Hinten fahren
7  * Autos rein, vorne werden Autos zur Reparatur
8  * herausgefahren.
9  *
10 * @author kriewel
11 *
12 */
13 public class Warteschlange {
14
15     // Warteschlange einer Werkstatt
16     private ArrayList schlange = new ArrayList ();
17
18     /**
19     * Fuegt ein neues Auto zur Warteschlange hinzu.
```

```

20     *
21     * @author krielwel
22     * @param auto das hinzuzufuegende Auto
23     */
24     public void autoRein(Auto auto) {
25         // fuegt Auto am Ende der Schlange hinzu
26         schlange.add(schlange.size(), auto);
27     }
28
29     /**
30     * Entnimmt das erste Auto aus der Warteschlange
31     * und liefert es zurueck. Ist die Warteschlange
32     * leer, wird null zurueckgeliefert
33     *
34     * @author krielwel
35     * @return das erste Auto in der Schlange oder null
36     */
37     public Auto autoRaus() {
38         if (schlange.isEmpty())
39             return null;
40         else // entferne das erste Auto aus der Schlange
41             return (Auto) schlange.remove(0);
42     }
43 }

```

Beide Klassen können als zusätzliches Übungsmaterial von der Webseite heruntergeladen werden. Auf diesen aufbauend sollen nun zwei Klassen `Werkstatt` und `Automechaniker` implementiert werden, die folgende Eigenschaften besitzen:

- Die `Werkstatt` soll über eine `Warteschlange` von Autos verfügen.
- Die `Werkstatt` soll eine Methode `void autoAbgeben(Auto auto)` erhalten, mit der ein neues Auto in die Warteschlange gestellt werden kann.
- Die `Werkstatt` soll im Konstruktor zehn `Automechaniker`-Threads erzeugen.
- Zum Simulieren der Reparatur soll ein `Automechaniker`-Objekt am `Auto`-Objekt die Methode `reparieren()` aufrufen. Die Reparatur soll eine zufällige Zeit von bis zu 1 Sekunde in Anspruch nehmen.
- Wenn zur Zeit kein Auto in der Warteschlange steht, soll der `Automechaniker` eine kurze Zeit (z.B. eine Sekunde) warten. Sonst soll er das nächste Auto aus der Werkstatt holen und reparieren.
- Die `Werkstatt` (in der neue Autos abgegeben werden) und alle zehn `Mechaniker` müssen auf dasselbe `Warteschlange`-Objekt zugreifen. Deshalb muss der Zugriff auf die Warteschlange korrekt synchronisiert werden.
- Zur besseren Unterscheidung kannst Du den Mechanikern noch Namen geben und bei der Reparatur ausgeben, welcher Mechaniker gerade das Auto repariert.

Schreibe dann eine Testklasse oder ergänze Werkstatt um eine `main()`-Methode zum Testen. Erstelle eine neue Werkstatt und füge etwa 50 Autos in die Warteschlange hinzu. Beobachte, wie die fleissigen Mechaniker sich um die Autos kümmern.

#### **Aufgabe 24:    Technischer Umgang mit Code**

Nimm Dir Deine Abgabe zum zweiten Testat oder auch andere frühere Aufgaben her und versuche, sie an die in der Vorlesung genannten Codingkonventionen anzupassen. Nutze dazu die Möglichkeiten von Eclipse beim Umbenennen von Bezeichnern oder dem Refaktorisieren von Code.

- Der erste Buchstabe eines Klassennamen wird stets groß geschrieben.
- Der erste Buchstabe eines Methodenbezeichners wird stets klein geschrieben.
- Der erste Buchstabe eines Variablenbezeichners wird stets klein geschrieben.
- Bezeichner von Konstanten werden komplett groß geschrieben.
- Setzt sich ein Bezeichner aus mehrere Worten zusammen, so wird der sogenannte CamelCase (also interne Großschreibung) benutzt.

Ausserdem sollten öffentliche Klassen in separaten Dateien gespeichert werden (nur eine `public` Klasse pro Quelldatei).