

U08 Entwurfsmuster (II)

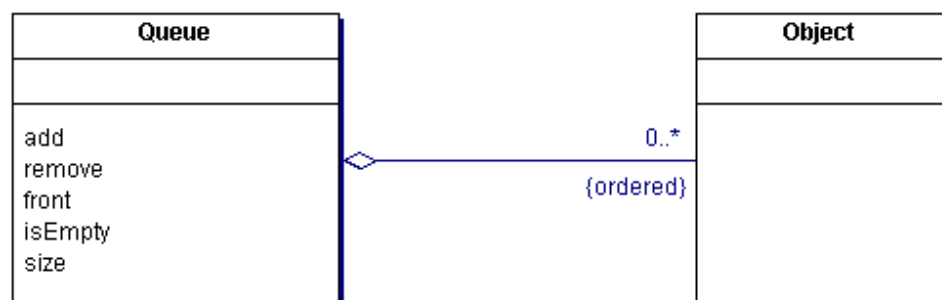
Inhalt der Übung

- Diskussion und Implementierung von Entwurfsmustern

Übungsaufgaben

Aufgabe 1 (Queue)

Gegeben ist das folgende Analysemodell einer Warteschlange (Queue):



Eine Warteschlange (Queue) besteht aus einer geordneten Reihenfolge von Objekten (Object). Objekte werden nach dem FIFO-Prinzip („first-in, first out“) in eine Warteschlange als letztes Element aufgenommen (`add()`) und als erstes Element entfernt (`remove()`).

`front()` stellt das erste Element der Warteschlange bereit.

`isEmpty()` testet, ob die Warteschlange leer ist.

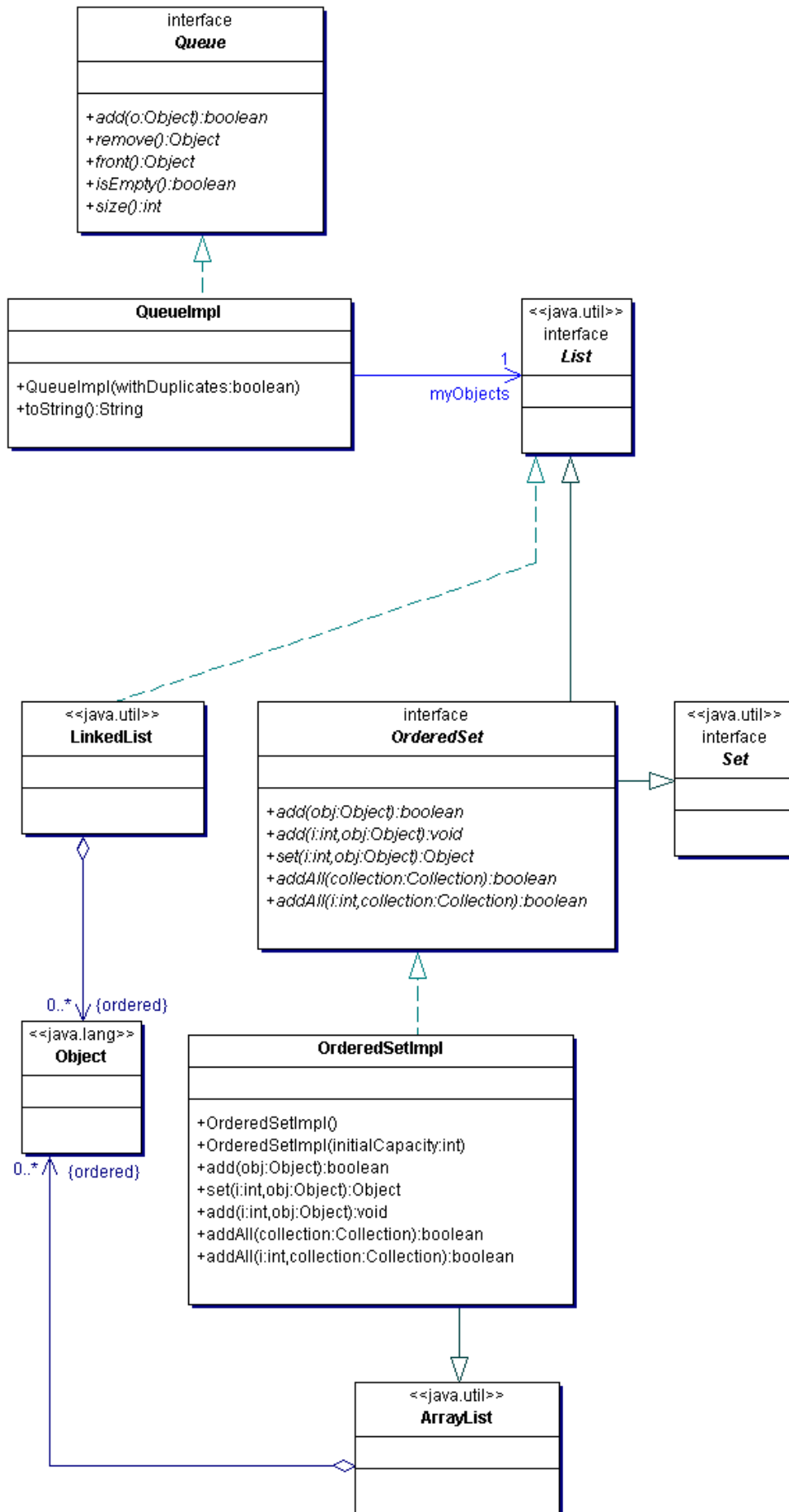
`size()` ermittelt die Anzahl der Objekte in der Warteschlange.

Es können zwei Arten von Warteschlangen erzeugt werden (siehe Klassendiagramm unten):

- Warteschlangen **ohne** duplizierte Objekte (d.h., ein Objekt kann sich nur genau einmal in der Warteschlange „anstellen“ → `withDuplicates = false`)
- Warteschlangen **mit** duplizierten Objekten (d.h., ein Objekt kann sich mehrfach in der Warteschlange „anstellen“ → `withDuplicates = true`)

Dementsprechend wird zur Implementation entweder

- eine Liste ohne duplizierte Objekte („geordnete Menge“ → `OrderedSet`, nicht im Java-Collection-Framework enthalten, erbt von `java.util.List` und `java.util.Set`) oder
- eine „herkömmliche“ Liste (hier `java.util.LinkedList`) benutzt.



Um die Warteschlange entsprechend dem gegebenen Modell zu implementieren, brauchen Sie (wie zuvor erklärt) eine geordnete Menge. Da im Java-Collection-Framework eine solche Datenstruktur nicht verfügbar ist, müssen Sie diese zunächst selbst implementieren (Klasse `OrderedSetImpl`). Gegeben ist dazu das Interface `OrderedSet.java`. Die Erläuterung der Methoden ist im Java-Code von `OrderedSet.java` enthalten.

Lösen Sie folgende Teilaufgaben:

- Welche Entwurfsmuster sind im Modell berücksichtigt? Diskutieren Sie diese Entwurfsmuster und zeichnen Sie sie in UML-Notation in das Klassendiagramm ein!
- Implementieren und testen Sie! (**Praktomat: Queue**)
Hinweise:
 - Um die Warteschlange entsprechend dem gegebenen Modell zu implementieren, brauchen Sie (wie zuvor erklärt) eine geordnete Menge. Da im Java-Collection-Framework eine solche Datenstruktur nicht verfügbar ist, müssen Sie diese zunächst selbst implementieren (Klasse `OrderedSetImpl`). Gegeben ist dazu das Interface `OrderedSet.java`. Die Erläuterung der Methoden ist im Java-Code von `OrderedSet` enthalten.
 - Jetzt können Sie mit Hilfe der Datenstruktur „geordnete Menge“ eine Warteschlange implementieren (`QueueImpl`)! Gegeben ist dazu das Interface `Queue.java`.

Aufgabe 2 (Vehicle Queue)

Gegeben ist ein Entwurfsmodell für die Simulation einer Verkehrs(warte)schlange (VehicleQueue) an einer (Rot-Grün-)Ampel.

Fahrzeuge (Vehicle) in dieser Schlange können Busse (Bus), Autos (Car) und Fahrräder (Bicycle) sein. Die Methode createVehicle() der Klasse VehicleGenerator erzeugt per Zufallsgenerator (randomGenerator) entweder einen Bus, ein Auto oder ein Fahrrad.

In der Verkehrsschlange stellen sich Fahrzeuge (enter()) mit einer zu definierenden Häufigkeit pro Sekunde (vehiclesPerSecondEnter) an.

Falls die Ampel auf grün steht (greenLight == true), verlassen Fahrzeuge die Schlange (leave()) mit einer ebenfalls zu definierenden Häufigkeit (vehiclesPerSecondLeave). Die Variable trafficLightRate gibt die Zeitdauer einer Rot- als auch Grünphase der Ampel in Sekunden an. Initialisiert ist die Ampel mit rot (greenLight == false).

Die Methoden getLength() und getSize() berechnen die Länge der Verkehrsschlange in Metern bzw. in Anzahl von Fahrzeugen.

Die Zeit wird durch ein Time-Objekt simuliert. Time kennt die aktuelle Zeit in Sekunden (currentTime) sowie eine Zeitbegrenzung in Sekunden (endTime). Die Methode run() zählt (immer mit 0 beginnend) die Zeit sekundenweise bis zur Zeitbegrenzung hoch. Der folgende Codeausschnitt aus der Klasse Simulation demonstriert beispielhaft die Anwendung der Klassen Time und VehicleQueue.

```
public class Simulation {
    static Time myTime . . .

    public static void main (String args[]) {
        VehicleQueue queue =
            new VehicleQueue(0.1, 30, 0.5, new
VehicleGenerator());
        . . .
        myTime.initEndTime(70);
        myTime.run();
    }
}
```

Lösen Sie folgende Teilaufgaben:

- Überdenken Sie den gegebenen Entwurf der Klasse Time, indem Sie die Klasse zu einem Singleton verändern. Ergänzen Sie dementsprechend die Klasse im UML-Klassendiagramm!
- Implementieren Sie die Klassen VehicleQueue und Time! Nach jeder Sekunde sollen entsprechend der oben beschriebenen Variablen Fahrzeuge die Schlange betreten als auch verlassen. Nutzen Sie zur Simulation (Simulation.java) des Sekundentaktes das Observer-Entwurfsmuster! Gegeben sind die Klassen Vehicle.java, VehicleGenerator.java, Bus.java, Bicycle.java und Car.java.

- Welches weitere Entwurfsmuster (neben Singleton und Observer) ist im Entwurf enthalten?
- Zeichnen Sie die Entwurfsmuster in UML-Notation in das nachfolgende Klassendiagramm ein!
- Implementieren Sie das Modell und testen Sie! (**Praktomat: Vehicle Queue**)!

