U08 Entwurfsmuster (II)

Inhalt der Übung

- Diskussion von Entwurfsmustern
- Implementierung von Entwurfsmustern in INLOOP

Übungsaufgaben

Aufgabe 1 (AudioClip Manager)

Gegeben ist die Klasse `AudioClipManager.java`. Diese Klasse verhindert das gleichzeitige Abspielen zweier Audio-Clips. Welches Entwurfsproblem musste bei der Implementierung gelöst werden? Welches Entwurfsmuster wurde verwendet? Erläutern Sie anhand eines Klassendiagramms die Lösung des Problems und zeichnen Sie das Entwurfsmuster in das Klassendiagramm ein!

Aufgabe 2 (Bibliothek)

Gegeben ist der Quellcode `Bibliothek.java`.

- Welche Entwurfsmuster erkennen Sie?
- Zeichnen Sie ein dazugehöriges Klassendiagramm einschließlich der Darstellung der Entwurfsmuster in UML-Notation!

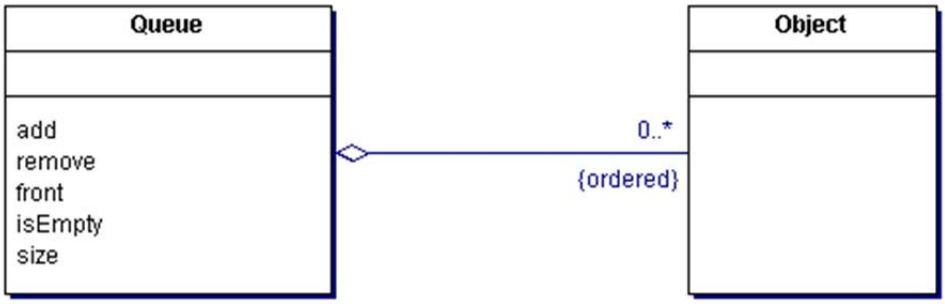
Aufgabe 3 (Abweichung)

Gegeben ist der Quellcode `Abweichung.java`.

- Welche Entwurfsmuster erkennen Sie?
- Zeichnen Sie ein dazugehöriges Klassendiagramm einschließlich der Darstellung der Entwurfsmuster in UML-Notation!

Aufgabe 4 (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()`)
- 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 Entwurfsklassendiagramm auf Seite 4):

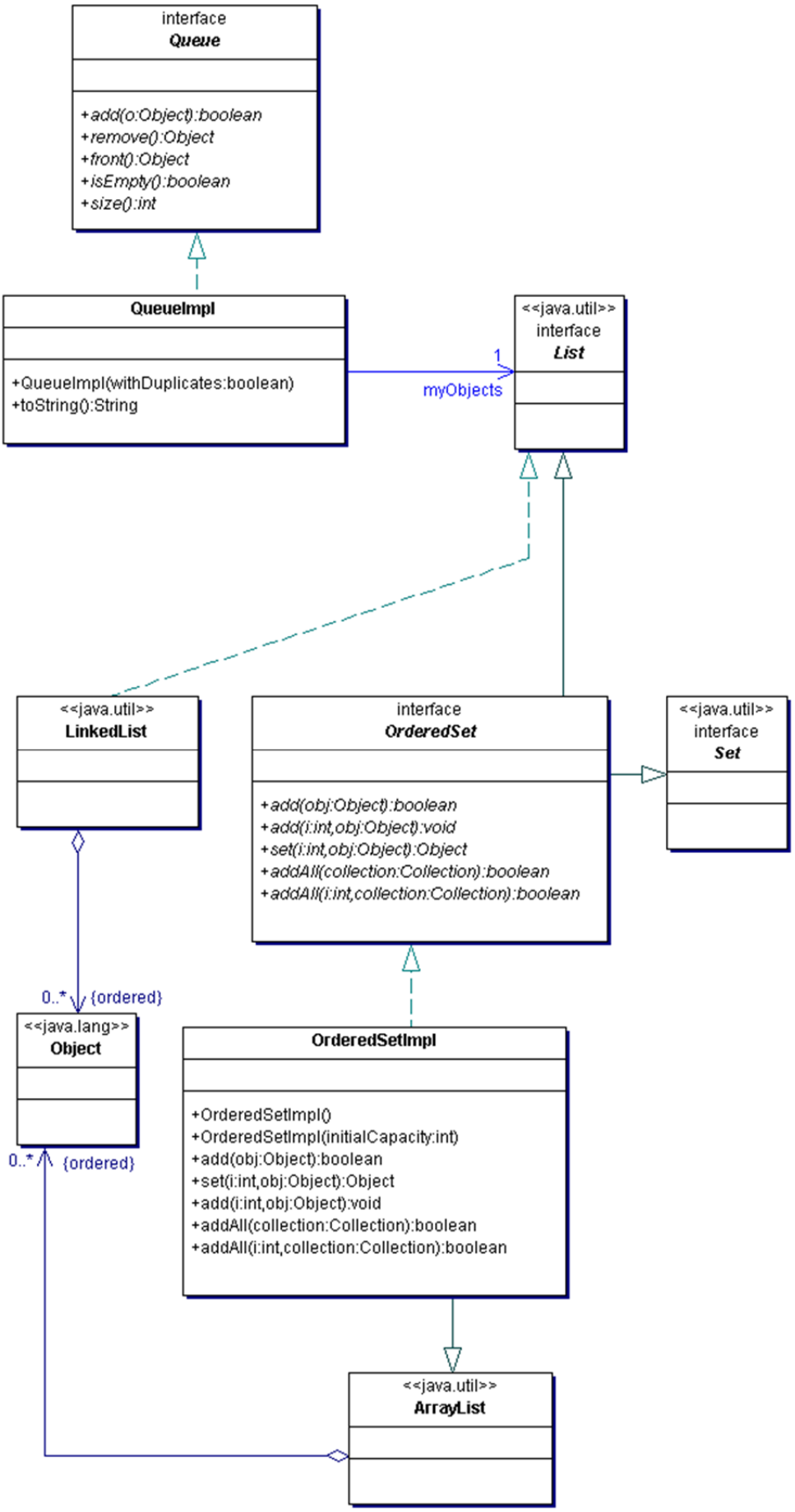
1. Warteschlangen **ohne** duplizierte Objekte (d.h., ein Objekt kann sich nur genau einmal in der Warteschlange „anstellen“ → `withDuplicates = false`)
2. Warteschlangen **mit** duplizierten Objekten (d.h., ein Objekt kann sich mehrfach in der Warteschlange „anstellen“ → `withDuplicates = true`)

Dementsprechend wird zur Implementierung entweder

- eine Liste ohne duplizierte Objekte („geordnete Menge“ → **`OrderedSet`**, nicht im Java2-Collection-Framework enthalten, erbt von `java.util.List` und `java.util.Set`) oder
- eine „herkömmliche“ Liste (hier **`java.util.LinkedList`**) benutzt.
- Beachten Sie, dass in Java 8 die Interfaces `Set` und `List` default-Implementierungen für die spliterator() Methode haben. Bei der gleichzeitigen Verwendung beider Interfaces muss die Precedence des Iterators klar festgelegt werden.

Teilaufgaben:

- Welche Entwurfsmuster sind im Modell berücksichtigt? Diskutieren Sie diese Entwurfsmuster und zeichnen Sie sie in UML-Notation in das Entwurfsklassendiagramm ein!
- Erweitern Sie das Entwurfsklassendiagramm um Generics!


interface
Queue
+add(o:Object):boolean
+remove():Object
+front():Object
+isEmpty():boolean
+size():int
QueueImpl
+QueueImpl(withDuplicates:boolean)
+toString():String
1
myObjects
<<java.util>>
interface
List
<<java.util>>
LinkedList
interface
OrderedSet
+add(obj:Object):boolean
+add(i:int,obj:Object):void
+set(i:int,obj:Object):Object
+addAll(collection:Collection):boolean
+addAll(i:int,collection:Collection):boolean
<<java.util>>
interface
Set
0..*
{ordered}
<<java.lang>>
Object
0..*
{ordered}
OrderedSetImpl
+OrderedSetImpl()
+OrderedSetImpl(initialCapacity:int)
+add(obj:Object):boolean
+set(i:int,obj:Object):Object
+add(i:int,obj:Object):void
+addAll(collection:Collection):boolean
+addAll(i:int,collection:Collection):boolean
<<java.util>>
ArrayList