
SoSe 2014
Prof. Dr. Margarita Esponda
ALP2
10. Übungsblatt (Abgabe am 30. Juni)

Ziel: Auseinandersetzung mit Dynamischen Datenstrukturen, Datenabstraktion, Innere Klassen und Iterator-Pattern in Java.

1. Aufgabe (10 Punkte)

Vervollständigen Sie die **ListQueue** Klasse aus der Vorlesung, die eine Warteschlange modelliert, indem Sie folgende Schnittstellen implementieren.

```
public interface Queue<E> {  
    /* fügt ein Element am Ende der Warteschlange ein */  
    public void enqueue( E element );  
  
    /* das erste Element der Warteschlange wird entfernt und  
       als Rückgabewert der Operation zurückgegeben */  
    public E dequeue() throws EmptyQueueException;  
  
    /* die Referenz des ersten Elements der Warteschlange  
       wird zurückgegeben ohne diese zu entfernen */  
    public E head() throws EmptyQueueException;  
  
    /* überprüft, ob die Warteschlange leer ist */  
    public boolean empty();  
  
    /* nur die Elemente der Warteschlange werden in die richtige  
       Reihenfolge als Zeichenkette zurückgegeben */  
    public String toString();  
}  
  
public interface Iterable<E> {  
    Iterator<E> iterator();  
}
```

- a) Die Warteschlange soll mit Hilfe eine **einfach verkettete Liste** implementiert werden (siehe Vorlesungsfolien).
- b) Eine **EmptyQueueException**-Klasse soll als erstes programmiert werden.
- c) Um die **Iterator**-Objekte zu erzeugen, soll eine **QueueIterator**-Klasse als innere Klasse implementiert werden, die folgende Schnittstelle implementiert:

```

public interface Iterator <T> {
    /* kotrolliert ob ein nächstes Element existiert */
    public boolean hasNext();
    /* das nächste Element der Aufzählung wird zurückgegeben oder ein
       NoMoreElementsException wird geworfen, wenn keine weitere Elemente vorhanden
       sind */
    public T next();
}

```

d) Schreiben Sie eine getrennte **TestListQueue** Klasse, die Ihre Warteschlange und entsprechende Methoden ausführlich testet.

2. Aufgabe (6 Punkte)

Vervollständigen Sie die Warteschlange Implementierung aus der Vorlesung, indem Sie folgende Schnittstellen implementieren.

```

public interface Queue<E> {
    /* fügt ein Element am Ende der Warteschlange ein */
    public void enqueue( E element );

    /* das erste Element der Warteschlange wird entfernt und
       als Rückgabewert der Operation zurückgegeben */
    public E dequeue() throws EmptyQueueException;

    /* die Referenz des ersten Elements der Warteschlange
       wird zurückgegeben ohne diese zu entfernen */
    public E first() throws EmptyQueueException;

    /* überprüft, ob die Warteschlange leer ist */
    public boolean empty();

    /* die Elemente der Warteschlange werden in die richtige
       Reihenfolge als Zeichenkette zurückgegeben */
    public String toString();
}

public interface Iterable<E> {
    Iterator<E> iterator();
}

```

- a) Die Warteschlange sollte ein selbst implementiertes dynamisches Array (siehe Vorlesungsfolien) verwenden.
- b) Eine Wrap-Around-Strategie soll für die Einfüge- und Lösch-Operationen benutzt werden (siehe Vorlesungsfolien).
- c) Die **EmptyQueueException**-Klasse ist nicht vorgegeben und soll als erstes programmiert werden.

- d) Um die **Iterator**-Objekte zu erzeugen, soll eine **QueueIterator**-Klasse als innere Klasse implementiert werden, die folgende Schnittstelle implementiert:

```
public interface Iterator <K> {  
    /* looks if a next object after a current position in the queue exists */  
    public boolean hasNext();  
    /* returns the next object in the list and advances the current position */  
    public K next();  
}
```

- e) Schreiben Sie eine getrennte **TestArrayQueue** Klasse, die Ihre Warteschlange und entsprechende Methoden ausführlich testet.

3. Aufgabe (18 Punkte)

Definieren Sie eine **PriorityQueue** Klasse in Java, die mit einer $O(\log n)$ Komplexität die Nachricht mit der höchsten Priorität aus der Warteschlange entfernen kann.

- a) Ihre generische Klassendefinition soll wie folgt starten:

```
public class PriorityQueue <P extends Comparable<P>, Data> {...}
```

- b) Für die Modellierung der Warteschlange verwenden Sie wie in der **4. Aufgabe des 4. Übungsblatts** eine binäre Heap-Struktur, die aber mit einem selbst implementierten dynamischen Array simuliert werden soll.

- c) Definieren Sie eine geeignete innere Klasse für die Elemente der Warteschlange, das aus einen Datenanteil **Data** und eine Priorität **P** bestehen.

- d) Definieren Sie für Ihre **PriorityQueue** Klasse folgende Methoden:

```
public void enqueue(P priority, Data data);  
/* speichert die Daten (data) mit der vorgegebenen Priorität P in die  
   Warteschlange */  
  
public Data dequeue() throws EmptyQueueException;  
/* die Daten des Objekts mit der höchsten Priorität werden als Ergebnis  
   zurückgegeben, und das Objekt wird aus der Warteschlange entfernt */  
  
public Data highest() throws EmptyQueueException;  
/* die Daten des Objekts mit der höchsten Priorität werden zurückgegeben  
   ohne das Objekt aus der Warteschlange zu entfernen */  
  
public boolean empty();
```

- e) Verwenden Sie **assert**-Anweisungen, um an verschiedenen kritischen Stellen die Korrektheit Ihrer Methoden und deren Anwendung rechtzeitig zu überprüfen.
- f) Schreiben Sie eine **SimulateMessageTraffic** Klasse, die unter Verwendung einer **PriorityQueue**-Objekt Nachrichtenverkehr simuliert. Die Nachrichten sollen in zufällige Reihenfolge produziert und verbraucht werden.

Wichtige Hinweise:

- 1) Verwenden Sie **selbsterklärende Namen** von Variablen und Methoden.
- 2) Für die Namen aller Bezeichner müssen Sie die **Java-Konventionen** verwenden.
- 3) Verwenden Sie **vorgegebene Klassen und Methodennamen**.
- 4) **Methoden sollten klein gehalten werden**, sodass auf den ersten Blick ersichtlich ist, was diese Methode leistet.
- 5) Methoden sollten möglichst **wenige Argumente** haben.
- 6) Methoden sollten entweder den Zustand der Eingabeargumente ändern oder einen Rückgabewert liefern.
- 7) Verwenden Sie **geeignete Hilfsvariablen** und definieren Sie **sinnvolle Hilfsmethoden** in Ihren Klassendefinitionen.
- 8) **Zahlen** sollten **durch Konstanten** ersetzt werden.
- 9) **Löschen Sie alle Programmzeilen und Variablen, die nicht verwendet werden.**