



Name: _____

Abiturprüfung 2008

Informatik, Grundkurs

Aufgabenstellung:

Bei Warteschlangen kommt es häufig vor, dass einige Elemente aus berechtigten Gründen nicht am Ende der Schlange eingefügt werden sollen, sondern an einem weiter vorne gelegenen Platz eingereiht werden. Dies sind zum Beispiel bestimmte Druckaufträge in einer Druckerwarteschlange oder Patienten mit verschiedenen Dringlichkeiten in einer Arztpraxis.

Zur Realisierung einer solchen Schlange werden die einzufügenden Objekte mit einer ganzzahligen Priorität versehen. Je höher die Priorität, desto weiter nach vorne gelangen die Objekte innerhalb der Schlange. Besitzen mehrere Objekte dieselbe Priorität, so soll das neue Objekt hinter die bereits enthaltenen Objekte gleicher Priorität eingereiht werden. Aus diesem Grund werden derartige Schlangen auch als Prioritätenwarteschlange (Priority Queue) bezeichnet.

- a) In einer Arztpraxis mit anfangs leerem Wartezimmer spielt sich folgendes Szenario ab:
- Herr Arendt erhält die Priorität 4 und betritt das Wartezimmer.
 - Frau Wolff erhält die Priorität 2 und betritt das Wartezimmer.
 - Frau Fritz erhält die Priorität 5 und betritt das Wartezimmer.
 - Der erste Patient wird aufgerufen und verlässt das Wartezimmer.
 - Herr Kluge erhält die Priorität 4 und betritt das Wartezimmer.

Stellen Sie die Belegung der Warteschlange nach jedem Einfügen eines Patienten geeignet dar. (6 Punkte)

- b) Basierend auf den bekannten Listenklassen `List` oder `Queue` (siehe Anhang) kann eine Klasse `PriorityQueue` jeweils durch eine Ist-Beziehung (Vererbung) oder durch eine Hat-Beziehung realisiert werden. Beschreiben Sie die Vor- und Nachteile dieser vier Möglichkeiten. (12 Punkte)



Name: _____

- c) Objekte, die in eine Prioritätenwarteschlange eingefügt werden sollen, sollen vom Typ `PriorityObject` sein, d. h., sie sollen stets einer Klasse angehören, die Unterklasse der Klasse `PriorityObject` ist. Für den Fall des Wartezimmers könnte dies etwa eine Klasse `Patient` sein.

Entwerfen Sie ein UML-Klassendiagramm für die Klassen `PriorityQueue`, `PriorityObject` und `Patient`. Geben Sie dabei alle Methoden und Beziehungen an.

Erläutern Sie kurz die Wirkung aller in `PriorityQueue` und `PriorityObject` enthaltenen Methoden mit Ausnahme der Konstruktoren. *(18 Punkte)*

- d) In der folgenden Aufgabe ‚haben‘ Objekte der Klasse `PriorityQueue` eine Liste `hList`, in der sie die Objekte der Klasse `PriorityObject` verwalten. In der Klasse `PriorityQueue` soll nun eine Methode `add(PriorityObject pObj)` zur Aufnahme eines Objektes der Klasse `PriorityObject` in die Prioritätenwarteschlange zur Verfügung stehen. Die Methode soll das einzufügende Objekt gemäß seiner Priorität an der entsprechenden Position in die Warteschlange einfügen.

Erläutern Sie die Vorgehensweise einer solchen Methode und implementieren Sie diese. *(14 Punkte)*

Zugelassene Hilfsmittel:

- Wörterbuch zur deutschen Rechtschreibung
- Taschenrechner



Name: _____

Anlagen

Die Klasse Queue

Objekte der Klasse Queue (Schlange) verwalten beliebige Objekte nach dem First-In-First-Out-Prinzip, d. h., dass das zuerst abgelegte Element als erstes wieder entnommen wird.

Die Klasse Queue stellt Methoden in folgender Syntax zur Verfügung:

```
public Queue()  
public boolean isEmpty()  
public void enqueue (Object pObject)  
public void dequeue ()  
public Object front()
```

Dokumentation der Methoden der Klasse Queue

Konstruktor Queue()

nachher Eine leere Schlange ist erzeugt.

Anfrage isEmpty()

nachher Die Anfrage liefert den Wert **true**, wenn die Schlange keine Elemente enthält, sonst liefert sie den Wert **false**.

Auftrag enqueue(Object pObject)

vorher Die Schlange ist erzeugt.

nachher pObject ist als letztes Element in der Schlange abgelegt.

Auftrag dequeue()

vorher Die Schlange ist nicht leer.

nachher Das vorderste Element ist aus der Schlange entfernt.

Anfrage front(): Object

vorher Die Schlange ist nicht leer.

nachher Die Anfrage liefert das vorderste Element der Schlange. Die Schlange ist unverändert.



Name: _____

Die Klasse List

Objekte der Klasse `List` verwalten beliebige Objekte nach einem Listenprinzip. Ein interner Positionszeiger wird durch die Listenstruktur bewegt, seine Position markiert ein aktuelles Objekt. Die Lage des Positionszeigers kann abgefragt, verändert und die Objektinhalte an den Positionen können gelesen oder verändert werden.

Die Klasse `List` stellt Methoden in folgender Syntax zur Verfügung:

```
public List()  
public boolean isEmpty()  
public boolean isInFrontOf()  
public boolean isBehind()  
public void next()  
public void previous()  
public void toFirst()  
public void toLast()  
public Object getItem()  
public void replace (Object pObject)  
public void insertInFrontOf (Object pObject)  
public void insertBehind (Object pObject)  
public void remove()  
public void addList (List pList)
```



Name: _____

Dokumentation der Methoden der Klasse List

Konstruktor **List()**

nachher Eine leere Liste ist angelegt. Der interne Positionszeiger steht vor der leeren Liste.

Anfrage **isEmpty(): boolean**

nachher Die Anfrage liefert den Wert **true**, wenn die Liste keine Elemente enthält, sonst liefert sie den Wert **false**.

Anfrage **isInFrontOf(): boolean**

nachher Die Anfrage liefert den Wert **true**, wenn der Positionszeiger vor dem ersten Listenelement oder vor der leeren Liste steht, sonst liefert sie den Wert **false**.

Anfrage **isBehind(): boolean**

nachher Die Anfrage liefert den Wert **true**, wenn der Positionszeiger hinter dem letzten Listenelement oder hinter der leeren Liste steht, sonst liefert sie den Wert **false**.

Auftrag **next()**

nachher Der Positionszeiger ist um eine Position in Richtung Listenende weitergerückt, d. h., wenn er vor der Liste stand, wird das Element am Listenanfang zum aktuellen Element, ansonsten das jeweils nachfolgende Listenelement. Stand der Positionszeiger auf dem letzten Listenelement, befindet er sich jetzt hinter der Liste. Befand er sich hinter der Liste, hat er sich nicht verändert.

Auftrag **previous()**

nachher Der Positionszeiger ist um eine Position in Richtung Listenanfang weitergerückt, d. h., wenn er hinter der Liste stand, wird das Element am Listenende zum aktuellen Element, ansonsten das jeweils vorhergehende Listenelement. Stand der Positionszeiger auf dem ersten Listenelement, befindet er sich jetzt vor der Liste. Befand er sich vor der Liste, hat er sich nicht verändert.

Auftrag **toFirst()**

nachher Der Positionszeiger steht auf dem ersten Listenelement. Falls die Liste leer ist, befindet er sich jetzt hinter der Liste.

Auftrag **toLast()**

nachher Der Positionszeiger steht auf dem letzten Listenelement. Falls die Liste leer ist, befindet er sich jetzt vor der Liste.

Anfrage **getItem(): Object**

nachher Die Anfrage liefert den Wert des aktuellen Listenelements bzw. **null**, wenn die Liste keine Elemente enthält bzw. der Positionszeiger vor oder hinter der Liste steht.

Auftrag **replace (Object pObject)**

vorher Die Liste ist nicht leer. Der Positionszeiger steht nicht vor oder hinter der Liste.
nachher Der Wert des Listenelements an der aktuellen Position ist durch **pObject** ersetzt.



Name: _____

Auftrag

insertInFrontOf (Object pObject)

vorher

Der Positionszeiger steht nicht vor der Liste.

nachher

Ein neues Listenelement mit dem entsprechenden Objekt ist angelegt und vor der aktuellen Position in die Liste eingefügt worden. Der Positionszeiger steht hinter dem eingefügten Element.

Auftrag

insertBehind (Object pObject)

vorher

Der Positionszeiger steht nicht hinter der Liste.

nachher

Ein neues Listenelement mit dem entsprechenden Objekt ist angelegt und hinter der aktuellen Position in die Liste eingefügt worden. Der Positionszeiger steht vor dem eingefügten Element.

Auftrag

remove()

vorher

Der Positionszeiger steht nicht vor oder hinter der Liste.

nachher

Das aktuelle Listenelement ist gelöscht. Der Positionszeiger steht auf dem Element hinter dem gelöschten Element bzw. hinter der Liste, wenn das gelöschte Element das letzte Listenelement war.

Auftrag

addList (List pList)

nachher

Die Liste pList ist der Liste angefügt. Die übergebene Listenstruktur von pList existiert nicht mehr.