Wirtschaftsinformatik
Hochschule Ravensburg-Weingarten

Testat

Objektorientierte Programmierung Praktikum

Gruppe 1 (09:00)

Studiengang

Wirtschaftsinformatik und E-Business (Bachelor) WI PLUS

Prof. Dr. Thomas Bayer

Sommersemester 2020

(Bitte in Druckbuchstaben ausfüllen!)

Matrikel-Nr:	Datum: 13.07.2020
Sitzplatz:	Semester: **SS 2020**
Hilfsmittel: **keine**	Bearbeitungszeit: **90 Minuten**

(Bitte nicht ausfüllen!)

	Unterschrift Prüfer:	Note:

Aufgabe	CL	Add	Poll	Co/re	getP	setP	Size	forAll		Gesamt
max. Punkte	**15**	**20**	**20**	**15**	**10**	**15**	**5**	**10**		**110**
Punkte										

Ausgangssituation: Prioritätswarteschlange

Es soll eine Prioritätswarteschlange implementiert werden. Die Berechnung der Priorität wird mittels des Interfaces **`IPDetermination`** realisiert und soll mit dem Konstruktor übergeben werden können. Um zu verhindern, dass Tickets nie an die Reihe kommen, wird mit Alterung (Ageing) gearbeitet.
Gesamtpriorität = die von der Implementierung IPDetermination errechnete Zahl. In die Berechnung fließen die Priorität des Tickets und das Alter des Tickets mit ein.

Interface

```
public interface IPriorityQueue<T extends Ticket> {

// Nimmt t in die Queue auf, sofern t != null und t noch nicht enthalten und
// die Priorität >= 0 ist. Ist t bereits enthalten, wird es nicht erneut
// aufgenommen, aber die Priorität des bestehenden Tickets durch die Priorität
// von t ersetzt. Rückgabe: False, wenn t == null, true sonst
boolean add(T t);

// Entnimmt das Ticket mit der größten Gesamtpriorität (gemäß Implementierung
// von IPDetermination. Falls zwei Tickets die gleiche Gesamtpriorität
// aufweisen, wird das Ticket gewählt, dessen BeginOfTicket-Datum kleiner ist
// (das ältere Ticket). Bei den restlichen Tickets wird das Alter mit der
// Methode incrementAge() erhöht (Siehe Ticket im Paket testat_g11)
T poll();

boolean contains(T t); // True: Ticket t ist enthalten, false sonst

// True: Ticket t wurde entfernt, false sonst. Der Status des entfernten
// Tickets wird auf TicketStatus.FINISHED gesetzt
boolean remove(T t);

// Liefert die Gesamtpriorität (IPDetermination) von t bzw. -1 falls t nicht
// enthalten ist
int getPriority(T t);

// Setzt die Priorität von t auf newPriority. Rückgabe wie getPriority()
int setPriority(T t, int newPriority);

int size(); // Anzahl der Elemente in der Queue

// Wendet die Methode action.accept(…) auf alle Tickets an, die in der Queue
// enthalten sind
void forAll(Consumer<T> action);
}
```

Aufgaben

- Implementieren Sie das Interface **`IPriorityQueue`**
- Verwenden Sie ein Array, um die Queue zu implementieren! Das Array soll mindestens 10 Tickets aufnehmen können
- Verwendung von Klassen aus java.util (ArrayList, LinkedList, …) ist nicht gestattet!

Vorbereitung

1. Starten Sie Eclipse mit einem passenden Workspace: Java Version mindestens 10 (besser 11)
2. Importieren Sie das Projekt OOPP-2020SS-Testat-G11 (Import existing project)!
3. Benennen Sie das Paket solution in solution##### um
4. Erstellen Sie die Klasse **`SimplePriorityQueueTest#####`**, die von **`ASimplePriorityQueueTest`** erbt
5. Bringen Sie die Unit-Tests zum Laufen

Vorgehen

- Implementieren Sie das Interface **`IPriorityQueue`** mit der Klasse **`SimplePriorityQueue#####`** (##### = Ihre Matrikelnummer) – Nur die entsprechenden Methodenrümpfe generieren!
- Passen Sie Ihre Testklasse so an, dass eine Instanz Ihrer Interfaceimplementierung übergeben wird
    a. Methode **`getInstance()`** – Geben Sie eine Instanz der Klasse **`SimplePriorityQueue#####`** zurück
    b. Tests, Errors, Failures:
- Programmieren Sie die Methoden aus
- Exportieren Sie Ihr Projekt als Archiv in das Verzeichnis C:\InsightFiles
- Vor Abgabe melden Sie sich!

Hinweise

- Syntax der Klassendefinition:

```
public class SimplePriorityQueue#####<T extends Ticket> implements IPriorityQueue<T>
```

- Instanziieren des Arrays:

```
<arrayname> =  (T[]) java.lang.reflect.Array.newInstance(Ticket.class, 10);
```

- Verwenden Sie die Implementierung des Interface IPDetermination, die in dem Attribut **`prioDetermination`** der abstrakten Testklasse gespeichert ist.
- `Date d = new Date();` liefert das aktuelle Datum und die Systemzeit
- Um zwei Date-Objekte d1, d2 zu vergleichen, können Sie folgende Methoden verwenden:
    - `d1.before(d2)`: True, falls d1 vor d2 liegt
    - `d1.after(d2)`: True, falls d1 nach d2 liegt

Hinweise zur Benotung (100 Punkte)

- **Es reichen 50 Punkte zum Bestehen!**
- **(Versuchte) Manipulation der gegebenen Klassen/Interfaces: max. 25 Punkte**
- **Kopieren der gegebenen Klassen/Interfaces in ein eigenes Paket: 0 Punkte**
- **Programm nicht lauffähig (Syntaxfehler): max. 25 Punkte**
- **Artefakte aus der anderen Gruppe tauchen auf: 0 Punkte!**
- Sowohl Unit-Tests als auch der Quelltext werden für die Benotung herangezogen
- Unit-Tests dienen nur zur Orientierung!

Anhang

Klasse Ticket

```
public class Ticket {
	private TicketStatus status;
	private Date beginOfTicket;
	public TicketStatus getStatus() {
	public int getPriority() {
	public void setPriority(int priority) {
	public void incrementAge() {
	public void setStatus(TicketStatus status) {
		this.status = status;
	}
	public Date getBeginOfTicket() {
	public void setEndOfTicket() {
	public Object getPayload() {
}
```

Collections

Interface/Klasse	Eine implementierende Klasse	Wichtige Methoden
Map	HashMap	put(k,v), get(k), size(), remove(k)
Set	HashSet	add(e), contains(e), remove(e)
List	ArrayList	add(e), add(i,e), get(i), contains(e), remove(e), remove(i)
Date	-	boolean before(Date d) boolean after(Date d)