Name: _____

Abiturprüfung 2015

Informatik, Leistungskurs

Aufgabenstellung:

In einer Telefonzentrale der Polizei sollen eingehende Notrufe aufgenommen und mithilfe einer Einsatzverwaltungssoftware verwaltet werden. Für jeden eingehenden Notruf wird ein Einsatz geplant, für den zunächst die folgenden Informationen in die Software aufgenommen werden:

(1) die Uhrzeit des Eingangs des Notrufs,
(2) eine Einsatzbeschreibung, aus der hervorgeht, worum es in dem Einsatz geht,
(3) der Einsatzort und
(4) eine fortgeführte Nummer, die mit jedem Einsatz um eins erhöht wird. Der erste Einsatz erhält die Nummer `1`.

Zudem wird jedem Einsatz von den Mitarbeitern in der Telefonzentrale

(5) eine Priorität in Form einer Zahl zwischen `1` und `3`

zugewiesen. Der Wert `3` heißt dabei „Einsatz mit höchster Priorität“, der Wert `1` heißt „Einsatz mit niedrigster Priorität“. Die Einsätze werden in eine Folge von wartenden Einsätzen absteigend nach Priorität einsortiert und der Reihe nach an frei werdende Einsatzwagen vergeben. Da es bei parallelen Aufnahmeplätzen vorkommen kann, dass fast gleichzeitig eingehende Notrufe nicht in der zeitlichen Reihenfolge ihres Anrufs der Folge der wartenden Einsätze von den Mitarbeitern hinzugefügt werden, ist es erforderlich, dass die Einsätze in die Folge nicht nur nach Priorität, sondern innerhalb gleicher Priorität zudem zeitlich einsortiert werden. Unter den Einsätzen gleicher Priorität stehen dabei diejenigen, deren zugehörige Notrufe später eingegangen sind, weiter hinten.

Name: _____

a) In der folgenden Tabelle ist eine Auflistung von eingegangenen Notrufen und Übernahmen von Einsätzen dargestellt. Dabei wird der Folge der wartenden Einsätze ein Einsatz gemäß der Sortierung nach Priorität und Zeit hinzugefügt, wenn ein Anruf eingeht. Das vorderste Element wird entfernt, wenn ein Einsatzwagen sich des Einsatzes annimmt.

Uhrzeit	Einsatz-nummer	Einsatzbeschreibung	Einsatzort	Prio-rität
18:37	1	Ruhestörung	Lange Straße 12	2
18:40	2	Leichter Verkehrsunfall (nur Blechschaden)	Steinstraße 2	1
18:41	Ein Polizeiwagen übernimmt den nächsten Einsatz			
18:42	3	Verkehrsunfall mit Personenschaden	Bergstraße 55	3

Tabelle 1: Eingegangene Notrufe und Übernahme von Einsätzen

Dokumentieren Sie die zeitliche Entwicklung der Folge der wartenden Einsätze, indem Sie für jede in Tabelle 1 dargestellte Veränderung die gesamte Folge der wartenden Einsätze angeben. Geben Sie dabei von jedem Einsatz nur die Einsatznummer an.

(8 Punkte)

In Abbildung 1 ist ein Implementationsdiagramm der Klassen `EinsatzVerwaltung`, `Einsatz`, `Zeitstempel` und `EinsatzGeber` der Einsatzverwaltungssoftware dargestellt, deren Dokumentationen im Anhang aufgeführt sind. Die Klasse `EinsatzGeber` modelliert die oben beschriebene Folge der wartenden Einsätze. Ein Objekt der Klasse `EinsatzGeber` dient dazu, den jeweils unter Berücksichtigung der Priorität und des Zeitpunkts der Notruf-zeit nächsten zu bearbeitenden Einsatz zu liefern.

Die Implementierung nutzt für die Verwaltung der Einsätze

- ein Objekt der Klasse `EinsatzGeber` für die Einsätze, die noch warten, da noch kein Einsatzwagen frei ist, und die im Folgenden **wartende Einsätze** genannt werden,
- ein Objekt der Klasse `List`, das alle gerade von einem der Einsatzwagen bearbeiteten Einsätze verwaltet, die im Folgenden **aktive Einsätze** genannt werden,
- ein Objekt der Klasse `List`, das die bereits abgeschlossenen Einsätze zur Archivierung verwaltet, die im Folgenden **archivierte Einsätze** genannt werden.

Die Klasse `EinsatzVerwaltung` ist für die Verwaltung dieser Datenstrukturen und somit aller Einsätze zuständig.

Name: _____

Geht ein Notruf bei der Polizei-Notrufzentrale ein, so wird mit der Methode `neuerNotruf` der Klasse `EinsatzVerwaltung` ein neues `Einsatz`-Objekt mit den zugehörigen Informationen inklusive einem aktuellen Zeitstempel erzeugt und dem Einsatzgeber mit seiner Methode `fuegeEin` hinzugefügt. Der Einsatzgeber ordnet `Einsatz`-Objekte, die ihm hinzugefügt werden, nach Priorität und Notrufzeit sortiert in die von ihm verwaltete Liste ein.

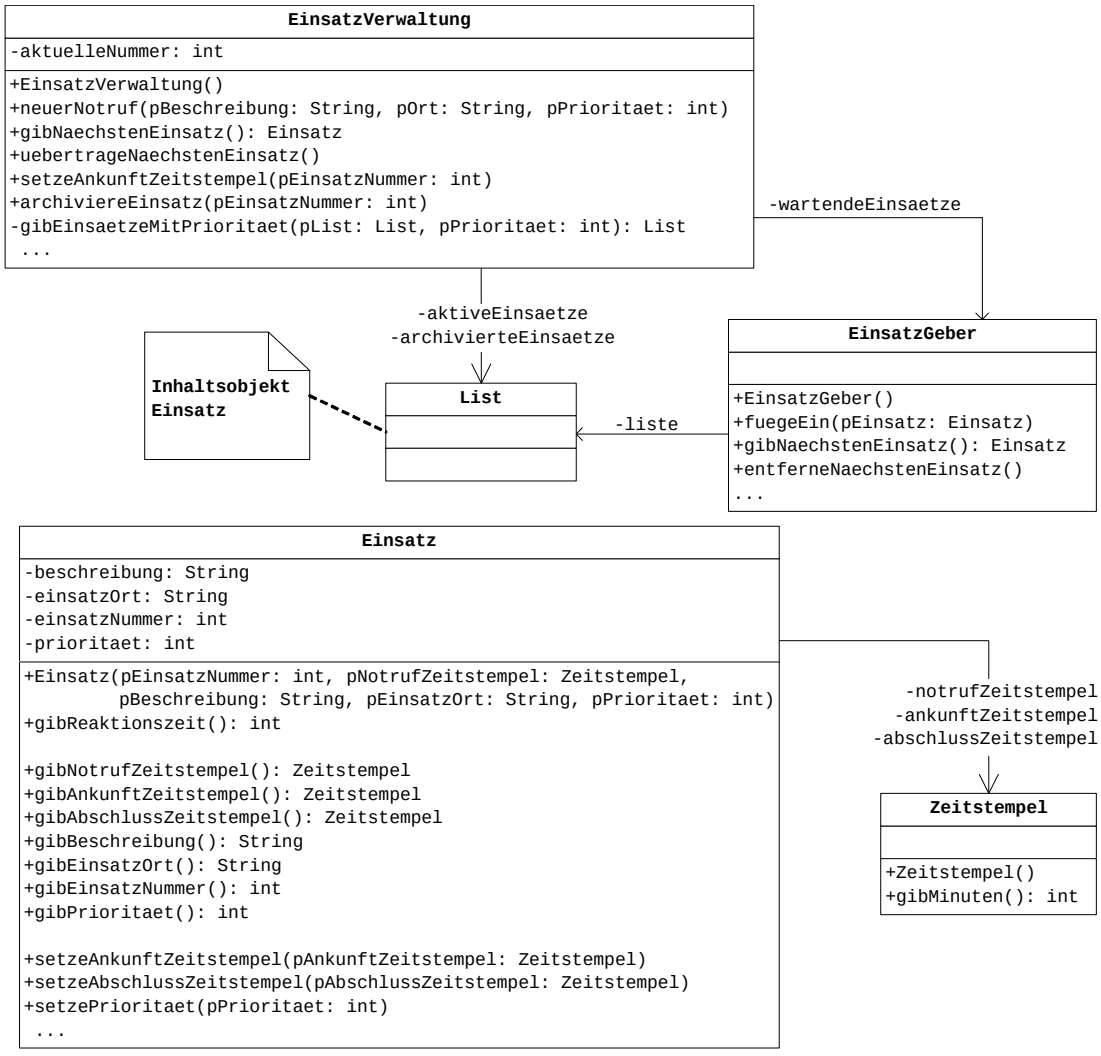

Abbildung 1: Ausschnitt aus einem Implementationsdiagramm der Einsatzverwaltungssoftware

Wird ein Einsatzwagen frei, so wird in der Telefonzentrale mit der Methode `gibNaechstenEinsatz` der Klasse `EinsatzVerwaltung` der nächste wartende Einsatz aus dem Einsatzgeber ausgelesen. Über Funk wird dem Einsatzteam des Wagens dieser Einsatz zugewiesen und die zugehörigen Informationen einschließlich der vom Programm automatisch vergebenen laufenden Einsatznummer mitgeteilt. Anschließend wird mit der Methode `uebertrageNaechstenEinsatz` das zugehörige `Einsatz`-Objekt aus dem Einsatzgeber entfernt und an die Liste der aktiven Einsätze angehängt.

Name: _____

b) Sobald ein Einsatzwagen meldet, am Einsatzort eingetroffen zu sein, wird mit der Methode `setzeAnkunftZeitstempel` der Klasse `EinsatzVerwaltung` die Ankunftszeit in Form eines Objekts der Klasse `Zeitstempel` in das in der Liste der aktiven Einsätze gespeicherte zugehörige `Einsatz`-Objekt eingetragen.

Implementieren Sie die Methode mit dem Methodenkopf

```
public void setzeAnkunftZeitstempel(int pEinsatzNummer)
```

gemäß der im Anhang gegebenen Beschreibung.

(10 Punkte)

c) Im Folgenden ist der Quelltext einer Methode `macheEtwas` der Klasse `EinsatzGeber` angegeben.

```
 1 public void macheEtwas(Zeitstempel pAktuelleUhrzeit,
                                                    int pMinuten) {
 2   Einsatz aktEinsatz;
 3   List einsaetze = new List();
 4   liste.toFirst();
 5   while (liste.hasAccess()) {
 6     aktEinsatz = (Einsatz) liste.getObject();
 7     if ((pAktuelleUhrzeit.gibMinuten()
       - aktEinsatz.gibNotrufZeitstempel().gibMinuten() > pMinuten)
                 && (aktEinsatz.gibPrioritaet() < 3)) {
 8       aktEinsatz.setzePrioritaet(aktEinsatz.gibPrioritaet() + 1);
 9       einsaetze.append(aktEinsatz);
10       liste.remove();
11     } else {
12       liste.next();
13     }
14   }
15   einsaetze.toFirst();
16   while (einsaetze.hasAccess()) {
17     aktEinsatz = (Einsatz) einsaetze.getObject();
18     einsaetze.next();
19     fuegeEin(aktEinsatz);
20   }
21 }
```

Die Methode werde um 14:14 Uhr mit einem Zeitstempel der aktuellen Uhrzeit und dem Parameter `pMinuten = 60` aufgerufen. Die intern verwaltete Liste `liste` des Einsatzgebers habe zur Zeit des Aufrufs den in Abbildung 2 dargestellten Inhalt. Es werden von jedem Einsatz nur die Einsatznummer, die Priorität und der Notruf-Zeitstempel in Form einer Uhrzeit dargestellt, wobei sämtliche Einsätze in dieser Liste am gleichen Tag eingegangen sind, so dass ein Tageswechsel hierbei nicht berücksichtigt werden muss.

Name: _____

Listen-anfang	Nr. 3 Prio. 3 13:10 Uhr	Nr. 4 Prio. 2 13:12 Uhr	Nr. 5 Prio. 2 14:00 Uhr	Nr. 2 Prio. 1 13:00 Uhr	Listen-ende

Abbildung 2: Die interne Liste, in der die wartenden Einsätze verwaltet werden

Analysieren Sie die Methode `macheEtwas` *und ermitteln Sie die Inhalte der intern verwalteten Liste* `liste` des `EinsatzGeber`*-Objekts nach Ausführung der Methode mit den oben angegebenen Parametern.*

Erläutern Sie, was die Ausführung der Methode im Sachzusammenhang bewirkt.

(12 Punkte)

Gibt ein Einsatzwagen durch, einen Einsatz abgeschlossen zu haben, wird das zugehörige `Einsatz`-Objekt aus der Liste der aktiven Einsätze entfernt und an die Liste der archivierten Einsätze hinten angehängt. Zudem wird dem zugehörigen `Einsatz`-Objekt ein aktueller Zeitstempel als Abschlusszeitstempel zugewiesen.

d) Zur Auswertung der Qualität des Notrufsystems sollen die archivierten abgeschlossenen Einsätze nach der Reaktionszeit, also der vergangenen Zeit zwischen dem Eingang des Notrufs und dem Eintreffen eines Einsatzwagens am Einsatzort, sortiert ausgegeben werden können.

Die Klasse `EinsatzVerwaltung` soll hierfür über eine Methode mit dem Methodenkopf

```
public List gibArchivEinsaetzeNachReaktionsZeitSortiert(
                                                    int pPrioritaet)
```

verfügen.

In der Klasse `EinsatzVerwaltung` gibt es bereits eine private Hilfsmethode

```
private List nachReaktionsZeitSortiertEinfuegen(List pListe,
                                                Einsatz pEinsatz),
```

die bei Übergabe einer bereits nach Reaktionszeit absteigend sortierten Liste `pListe` eine neue Liste erstellt, die genau mit der Liste `pListe` übereinstimmt, außer dass zusätzlich das übergebene Einsatz-Objekt `pEinsatz` entsprechend der Sortierung in diese neue Liste eingefügt wurde. Bei Einsätzen gleicher Reaktionszeit wird der neu hinzuzufügende Einsatz `pEinsatz` hinter den Einsätzen gleicher Reaktionszeit in die neu erstellte Liste eingefügt.

Name: _____

In der Methode `gibArchivEinsaetzeNachReaktionsZeitSortiert` wird eine neue Liste erstellt, in die mithilfe der Methode `nachReaktionsZeitSortiertEinfuegen` sämtliche `Einsatz`-Objekte der Liste `archivierteEinsaetze` mit der übergebenen Priorität `pPrioritaet` nach Reaktionszeit absteigend sortiert eingefügt werden. Einsätze mit gleicher Reaktionszeit sollen in der Reihenfolge, in der sie archiviert wurden, in der Liste enthalten sein. Die neue Liste wird als Ergebnis zurückgegeben.

Implementieren Sie die Methode `gibArchivEinsaetzeNachReaktionsZeitSortiert` *entsprechend diesem Verfahren unter Verwendung der Methode* `nachReaktionsZeitSortiertEinfuegen`.

(10 Punkte)

e) Die Klasse `EinsatzGeber` ist für die Einsatzverwaltungssoftware von zentraler Bedeutung, da sie für die Sortierung der Einsätze nach Priorität und innerhalb gleicher Priorität nach Notrufzeit verantwortlich ist.

Beim Entwurf der Klasse `EinsatzGeber` wurde die Alternative untersucht, anstelle einer internen Liste für alle `Einsatz`-Objekte drei `Queue`-Objekte für die drei verschiedenen Prioritäten so zu verwenden, dass je ein `Queue`-Objekt sämtliche `Einsatz`-Objekte einer Priorität aufnimmt.

Beurteilen Sie die Entscheidung für die Verwendung eines internen `List`*-Objekts anstelle der Verwendung dreier* `Queue`*-Objekte anhand der durch die jeweilige Wahl der Datenstrukturen erforderlichen Vorgehensweisen beim Einfügen. Eine Analyse und ein Vergleich des Aufwands sind nicht gefordert.*

(10 Punkte)

Zugelassene Hilfsmittel:

- Wörterbuch zur deutschen Rechtschreibung
- Taschenrechner

Name: _____

Anhang

Die Klasse `Einsatz`

Ein Objekt der Klasse `Einsatz` repräsentiert einen Einsatz, der aus einem eingegangen Notruf hervorgeht. Es speichert die für die Verwaltung und für die Einsatzwagen erforderlichen Daten des Einsatzes.

Ausschnitt aus der Dokumentation der Klasse `Einsatz`

Konstruktor `Einsatz(int pEinsatzNummer, Zeitstempel pNotrufZeitstempel, String pBeschreibung, String pEinsatzOrt, int pPrioritaet)`

Ein Einsatz wird erzeugt. Die Werte der Parameter werden gespeichert.

Anfrage `int gibReaktionsZeit()`

Wenn noch kein Ankunftzeitstempel gesetzt ist, wird -1 zurückgegeben. Sonst wird die Zeit zwischen dem Notrufzeitstempel und dem Ankunftzeitstempel in Minuten zurückgegeben.

Anfrage `Zeitstempel gibNotrufZeitstempel()`

Gibt den Zeitstempel zurück, der den Zeitpunkt des Notrufs repräsentiert.

Anfrage `Zeitstempel gibAnkunftZeitstempel()`

Gibt den Zeitstempel zurück, der den Zeitpunkt repräsentiert, zu dem der betreffende Einsatzwagen der Notrufzentrale gemeldet hat, am Einsatzort angekommen zu sein. Wurde noch kein Ankunftzeitstempel gesetzt, so wird `null` zurückgegebenen.

Anfrage `Zeitstempel gibAbschlussZeitstempel()`

Gibt den Zeitstempel zurück, der den Zeitpunkt repräsentiert, zu dem der betreffende Einsatzwagen der Notrufzentrale gemeldet hat, den Einsatz abgeschlossen zu haben. Wurde noch kein Abschlusszeitstempel gesetzt, so wird `null` zurückgegebenen.

Anfrage `String gibBeschreibung()`

Gibt die Einsatzbeschreibung zurück.

Anfrage `String gibEinsatzOrt()`

Gibt den Ort des Einsatzes zurück.

Name: _____

Anfrage `int gibEinsatzNummer()`
Gibt die Einsatznummer zurück.

Anfrage `int gibPrioritaet()`
Gibt die Priorität zurück.

Auftrag `void setzeAnkunftZeitstempel(Zeitstempel pAnkunftZeitstempel)`
Setzt den Zeitstempel für den Zeitpunkt, zu dem der betreffende Einsatzwagen der Notrufzentrale meldet, am Einsatzort angekommen zu sein.

Auftrag `void setzeAbschlussZeitstempel(Zeitstempel pAbschlussZeitstempel)`
Setzt den Zeitstempel für den Zeitpunkt, zu dem der betreffende Einsatzwagen der Notrufzentrale meldet, den Einsatz abgeschlossen zu haben.

Auftrag `void setzePrioritaet(int pPrioritaet)`
Setzt die Priorität.

Die Klasse `Zeitstempel`

Ein Objekt der Klasse `Zeitstempel` repräsentiert einen Zeitpunkt. Die genauere innere technische Realisierung wird in dieser Klausur nicht thematisiert.

Ausschnitt aus der Dokumentation der Klasse `Zeitstempel`

Konstruktor `Zeitstempel()`
Ein neuer Zeitstempel wird erzeugt, der die zum Zeitpunkt der Erstellung aktuelle Uhrzeit repräsentiert.

Anfrage `int gibMinuten()`
Gibt die Anzahl der Minuten, die seit dem 01.01.2000 um 00:00 Uhr bis zum Zeitpunkt dieses Zeitstempels vergangen sind, zurück.

Name: _____

Die Klasse EinsatzVerwaltung

Ein Objekt der Klasse `EinsatzVerwaltung` verwaltet die wartenden, die aktiven und die archivierten Einsätze.

Ausschnitt aus der Dokumentation der Klasse `EinsatzVerwaltung`

Konstruktor `EinsatzVerwaltung()`

Ein Objekt der Klasse `EinsatzVerwaltung` wird erzeugt. Dabei wird ein Objekt der Klasse `EinsatzGeber` für die Verwaltung der wartenden Einsätze, eine Liste für die Verwaltung der aktiven Einsätze, eine Liste für die Verwaltung der archivierten Einsätze erzeugt.

Auftrag **`void neuerNotruf(String pBeschreibung, String pOrt, int pPrioritaet)`**

Es wird ein neues Objekt der Klasse `Einsatz` mit den entsprechenden Daten inklusive einer laufenden Nummer und einem aktuellen Zeitstempel erzeugt. Das Objekt wird dem Einsatzgeber hinzugefügt.
Die laufende Nummer wird dabei mit jedem Aufruf dieser Methode um eins erhöht. Beim ersten Aufruf dieser Methode erhält das erzeugte `Einsatz`-Objekt die Nummer `1`.

Anfrage **`Einsatz gibNaechstenEinsatz()`**

Es wird der nächste Einsatz aus dem verwalteten Objekt der Klasse `EinsatzGeber` ausgelesen und zurückgegeben. Ist kein nächster Einsatz vorhanden, so wird `null` zurückgegeben.

Auftrag **`void uebertrageNaechstenEinsatz()`**

Es wird der nächste Einsatz des verwalteten Objekts der Klasse `EinsatzGeber` ausgelesen, dort entfernt und an die Liste der aktiven Einsätze angehängt. Wenn es keinen nächsten Einsatz gibt, so geschieht nichts.

Auftrag **`void setzeAnkunftZeitstempel(int pEinsatzNummer)`**

Befindet sich der Einsatz mit der übergebenen Einsatznummer in der Liste der aktiven Einsätze, so wird ein neuer Zeitstempel erzeugt und diesem als Ankunftzeitstempel zugewiesen. Befindet sich kein Einsatz mit der übergebenen Einsatznummer in der Liste der aktiven Einsätze, so geschieht nichts.

Auftrag **`void archiviereEinsatz(int pEinsatzNummer)`**

Ist in der Liste der aktiven Einsätze kein `Einsatz`-Objekt mit der übergebenen Einsatznummer enthalten, so geschieht nichts. Ansonsten wird das Objekt mit der übergebenen Einsatznummer aus der Liste der aktiven Einsätze entfernt und an die Liste `archivierteEinsaetze` hinten angehängt.

Name: _____

Anfrage `List gibArchivEinsaetzeNachReaktionsZeitSortiert (int pPrioritaet)`

Es wird eine neue Liste erstellt, die sämtliche archivierten `Einsatz`-Objekte der als Parameter übergebenen Priorität enthält und die nach der Reaktionszeit absteigend sortiert ist. Einsätze mit gleicher Reaktionszeit sind in der Reihenfolge, in der sie archiviert wurden, in der Liste enthalten. Befindet sich kein `Einsatz`–Objekt, das die übergebene Priorität hat, in der Liste der archivierten Einsätze, so wird eine leere Liste zurückgegeben.

Die Klasse EinsatzGeber

Ein Objekt der Klasse `EinsatzGeber` verwaltet Einsätze nach Priorität absteigend sortiert. Die Objekte werden hierzu in einer Liste vorgehalten

Ausschnitt aus der Dokumentation der Klasse `EinsatzGeber`

Konstruktor `EinsatzGeber()`

Ein Objekt der Klasse `EinsatzGeber` wird erzeugt.

Auftrag `void fuegeEin(Einsatz pEinsatz)`

Das übergebene `Einsatz`-Objekt wird nach Priorität absteigend sortiert in den Einsatzgeber eingefügt. `Einsatz`-Objekte gleicher Priorität werden nach dem Notruf-Zeitstempel so sortiert eingefügt, dass später eingegangene Notrufe weiter hinten im Einsatzgeber stehen. Wird `null` übergeben, so wird nichts gemacht.

Anfrage `Einsatz gibNaechstenEinsatz()`

Ist der Einsatzgeber leer, so wird `null` zurückgegeben, sonst wird das am weitesten vorne stehende `Einsatz`-Objekt, also der als nächstes zu bearbeitende Einsatz, zurückgegeben. Der Einsatzgeber bleibt unverändert.

Auftrag `void entferneNaechstenEinsatz()`

Ist der Einsatzgeber leer, so geschieht nichts, sonst wird das am weitesten vorne stehende `Einsatz`-Objekt aus dem Einsatzgeber entfernt.

Name: _____

Die Klasse `List`

Objekte der Klasse **`List`** verwalten beliebig viele, linear angeordnete Objekte. Auf höchstens ein Listenobjekt, aktuelles Objekt genannt, kann jeweils zugegriffen werden. Wenn eine Liste leer ist, vollständig durchlaufen wurde oder das aktuelle Objekt am Ende der Liste gelöscht wurde, gibt es kein aktuelles Objekt. Das erste oder das letzte Objekt einer Liste können durch einen Auftrag zum aktuellen Objekt gemacht werden. Außerdem kann das dem aktuellen Objekt folgende Listenobjekt zum neuen aktuellen Objekt werden.

Das aktuelle Objekt kann gelesen, verändert oder gelöscht werden. Außerdem kann vor dem aktuellen Objekt ein Listenobjekt eingefügt oder ein Listenobjekt an das Ende der Liste angefügt werden.

Dokumentation der Klasse `List`

Konstruktor **`List()`**
Eine leere Liste wird erzeugt.

Anfrage **`boolean isEmpty()`**
Die Anfrage liefert den Wert `true`, wenn die Liste keine Objekte enthält, sonst liefert sie den Wert `false`.

Anfrage **`boolean hasAccess()`**
Die Anfrage liefert den Wert `true`, wenn es ein aktuelles Objekt gibt, sonst liefert sie den Wert `false`.

Auftrag **`void next()`**
Falls die Liste nicht leer ist, es ein aktuelles Objekt gibt und dieses nicht das letzte Objekt der Liste ist, wird das dem aktuellen Objekt in der Liste folgende Objekt zum aktuellen Objekt, andernfalls gibt es nach Ausführung des Auftrags kein aktuelles Objekt, d. h., `hasAccess()` liefert den Wert `false`.

Auftrag **`void toFirst()`**
Falls die Liste nicht leer ist, wird das erste Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts.

Auftrag **`void toLast()`**
Falls die Liste nicht leer ist, wird das letzte Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts.

Name: _____

Anfrage **`Object getObject()`**

Falls es ein aktuelles Objekt gibt (`hasAccess()` == `true`), wird das aktuelle Objekt zurückgegeben, andernfalls (`hasAccess()` == `false`) gibt die Anfrage den Wert `null` zurück.

Auftrag **`void setObject(Object pObject)`**

Falls es ein aktuelles Objekt gibt (`hasAccess()` == `true`) und `pObject` ungleich `null` ist, wird das aktuelle Objekt durch `pObject` ersetzt. Sonst bleibt die Liste unverändert.

Auftrag **`void append(Object pObject)`**

Ein neues Objekt `pObject` wird am Ende der Liste eingefügt. Das aktuelle Objekt bleibt unverändert. Wenn die Liste leer ist, wird das Objekt `pObject` in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt (`hasAccess()` == `false`). Falls `pObject` gleich `null` ist, bleibt die Liste unverändert.

Auftrag **`void insert(Object pObject)`**

Falls es ein aktuelles Objekt gibt (`hasAccess()` == `true`), wird ein neues Objekt vor dem aktuellen Objekt in die Liste eingefügt. Das aktuelle Objekt bleibt unverändert. Falls die Liste leer ist und es somit kein aktuelles Objekt gibt (`hasAccess()` == `false`), wird `pObject` in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt. Falls es kein aktuelles Objekt gibt (`hasAccess()` == `false`) und die Liste nicht leer ist oder `pObject` gleich `null` ist, bleibt die Liste unverändert.

Auftrag **`void concat(List pList)`**

Die Liste `pList` wird an die Liste angehängt. Anschließend wird `pList` eine leere Liste. Das aktuelle Objekt bleibt unverändert. Falls `pList` `null` oder eine leere Liste ist, bleibt die Liste unverändert.

Auftrag **`void remove()`**

Falls es ein aktuelles Objekt gibt (`hasAccess()` == `true`), wird das aktuelle Objekt gelöscht und das Objekt hinter dem gelöschten Objekt wird zum aktuellen Objekt. Wird das Objekt, das am Ende der Liste steht, gelöscht, gibt es kein aktuelles Objekt mehr (`hasAccess()` == `false`). Wenn die Liste leer ist oder es kein aktuelles Objekt gibt (`hasAccess()` == `false`), bleibt die Liste unverändert.

Name: _____

Die Klasse Queue

Objekte der Klasse **`Queue`** (Warteschlange) verwalten beliebige Objekte nach dem First-In-First-Out-Prinzip, d. h., das zuerst abgelegte Objekt wird als erstes wieder entnommen.

Dokumentation der Klasse Queue

Konstruktor **`Queue()`**

Eine leere Schlange wird erzeugt.

Anfrage **`boolean isEmpty()`**

Die Anfrage liefert den Wert `true`, wenn die Schlange keine Objekte enthält, sonst liefert sie den Wert `false`.

Auftrag **`void enqueue(Object pObject)`**

Das Objekt `pObject` wird an die Schlange angehängt. Falls `pObject` gleich `null` ist, bleibt die Schlange unverändert.

Auftrag **`void dequeue()`**

Das erste Objekt wird aus der Schlange entfernt. Falls die Schlange leer ist, wird sie nicht verändert.

Anfrage **`Object front()`**

Die Anfrage liefert das erste Objekt der Schlange. Die Schlange bleibt unverändert. Falls die Schlange leer ist, wird `null` zurückgegeben.

Unterlagen für die Lehrkraft

Abiturprüfung 2015

Informatik, Leistungskurs

1. Aufgabenart

Aufgabenart	Modellierung einer Problemstellung, Entwurf und Implementation von Algorithmen
Syntaxvariante	Java

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

- entfällt

4. Bezüge zu den Vorgaben 2015

1. *Inhaltliche Schwerpunkte*
   Objektorientiertes Modellieren und Implementieren von kontextbezogenen Anwendungen
   - Konzepte des objektorientierten Modellierens
   - Algorithmen und Datenstrukturen
     - Lineare Strukturen mit den Akzenten
       Lineare Liste
       Schlange

2. *Medien/Materialien*
   - entfällt

5. Zugelassene Hilfsmittel

- Wörterbuch zur deutschen Rechtschreibung
- Taschenrechner

[1] Die Aufgabenstellung deckt inhaltlich alle drei Anforderungsbereiche ab.

6. Modelllösungen

Die jeweilige Modelllösung stellt eine mögliche Lösung bzw. Lösungsskizze dar. Der gewählte Lösungsansatz und -weg der Schülerinnen und Schüler muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden mit entsprechender Punktzahl bewertet (Bewertungsbogen: Zeile „Sachlich richtige Lösungsalternative zur Modelllösung“).

Teilaufgabe a)

Schritt 1:

Anfang der Folge	*1*	*Ende der Folge*

Schritt 2:

Anfang der Folge	*1*	2	*Ende der Folge*

Schritt 3:

Anfang der Folge	*2*	*Ende der Folge*

Schritt 4:

Anfang der Folge	*3*	2	*Ende der Folge*

Teilaufgabe b)

Implementation:

```
public void setzeAnkunftZeitstempel(int pEinsatzNummer) {
  if (!aktiveEinsaetze.isEmpty()) {
    // aktuellen Einsatz mit pEinsatzNummer suchen
    aktiveEinsaetze.toFirst();
    Einsatz einsatz = (Einsatz) aktiveEinsaetze.getObject();
    while (aktiveEinsaetze.hasAccess() && pEinsatzNummer !=
                                          einsatz.gibEinsatzNummer()) {
      aktiveEinsaetze.next();
      einsatz = (Einsatz) aktiveEinsaetze.getObject();
    }
    // gefunden => Zeitstempel setzen
    if (aktiveEinsaetze.hasAccess()) {
      einsatz.setzeAnkunftZeitstempel(new Zeitstempel());
    }
  }
}
```

Teilaufgabe c)

Die Liste sieht nach der Ausführung wie folgt aus:

Listen-anfang	Nr. 3 Prio. 3 13:10 Uhr	Nr. 4 Prio. 3 13:12 Uhr	Nr .2 Prio. 2 13:00 Uhr	Nr. 5 Prio. 2 14:00 Uhr	Listen-ende

In der ersten `while`-Schleife werden sämtliche Einsätze, die bereits vor mehr als `pMinuten` Minuten in Auftrag gegeben wurden und deren Priorität unter `3` liegt, aus der Liste der wartenden Einsätze entfernt und ihre Priorität um `1` erhöht. Die entfernten Einsätze werden zudem in eine neue Liste eingefügt.

In der zweiten `while`-Schleife werden alle Einsätze der neuen Liste erneut in die Liste der wartenden Einsätze sortiert eingefügt. Das Sortiert-Einfügen erfolgt mit der Methode `fuegeEin` der gleichen Klasse.

Die Methode erhöht also die Priorität aller Einsätze, die schon vor mehr als die als Parameter `pMinuten` übergebenen Minuten in Auftrag gegeben wurden, um eins, wenn die Priorität nicht schon den maximalen Wert von `3` hat.

Teilaufgabe d)

```
public List gibArchivEinsaetzeNachReaktionsZeitSortiert(int pPrioritaet) {
  List ergebnisListe = new List();
  archivierteEinsaetze.toFirst();
  Einsatz einsatz;
  // Durchlauf durch die Liste und sortiertes Einfügen aller
  // Einsatz-Objekte der gesuchten Priorität
  while (archivierteEinsaetze.hasAccess()) {
    einsatz = (Einsatz) archivierteEinsaetze.getObject();
    archivierteEinsaetze.next();
    if (einsatz.gibPrioritaet() == pPrioritaet) {
      ergebnisListe = nachReaktionsZeitSortiertEinfuegen(
                                                  ergebnisListe, einsatz);
    }
  }
  return ergebnisListe;
}
```

Teilaufgabe e)

Die `Einsatz`-Objekte sollen nach zwei Kriterien sortiert vorliegen. Erstes Kriterium ist die Priorität und zweites Kriterium ist die Notrufzeit.

Bei der Verwendung einer Liste kann diese beim Einfügen von Anfang bis Ende so weit durchlaufen werden, bis die Stelle gefunden wurde, an der ein einzufügendes `Einsatz`-Objekt gefunden wurde. Hierbei können beide Kriterien berücksichtigt werden.

Bei der Verwendung von drei `Queue`-Objekten kann die Sortierung nach dem ersten Sortier-Kriterium in einem Schritt erfolgen, indem anhand der Priorität des einzufügenden `Einsatz`-Objekts das zugehörige `Queue`-Objekt ausgewählt wird, in das das jeweilige `Einsatz`-Objekt eingeordnet werden soll.

Als richtig betrachtete Angaben zum Aspekt des Sortiert-Einfügens nach der Notrufzeit innerhalb einer Priorität:

Alternative 1
Das Sortiert-Einfügen in das entsprechend der Priorität ausgewählte `Queue`-Objekt nach dem zweiten Kriterium, also der Notrufzeit, gestaltet sich aufgrund des eingeschränkten Zugriffs bei Schlangen nach dem FIFO-Prinzip schwieriger. Daher ist die Klasse `Queue` für die Verwaltung der Einsätze je einer Priorität nicht geeignet.

Alternative 2
Das Sortiert-Einfügen in das entsprechend der Priorität ausgewählte `Queue`-Objekt ist zum Beispiel wie folgt möglich: Zunächst merkt man sich das vorderste Element der Schlange, anschließend kann durch wiederholte Entnahme des vordersten Elements und hinten Anhängen dieses Elements jedes Element der Schlange zum hintersten Element in der Schlange gemacht werden und daher hinter jedem Element der Schlange angehängt werden. Um den nächsten Einsatz aus dem Einsatzgeber entnehmen zu können, ist nach dem Anhängen wieder eine wiederholte Entnahme des vordersten Elements und hinten Anhängen dieses Elements so lange erforderlich, bis das ursprünglich vorderste Element wieder zum vordersten Element der Schlange geworden ist oder bis der neu eingefügte Notruf das vorderste Element ist, falls dies der Anordnung nach der Notrufzeit entspricht. Daher wäre auch die Klasse `Queue` für die Verwaltung der Einsätze je einer Priorität geeignet.

Anmerkung zur Korrektur:
Eine Analyse und ein Vergleich des Aufwands sind nicht erforderlich.

7. Teilleistungen – Kriterien / Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings:_____ Kursbezeichnung:_____

Schule: _____

Teilaufgabe a)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**[2]	**ZK**	**DK**
1	gibt für jeden der vier Schritte eine Liste mit den Einsatznummern in richtiger Reihenfolge an.	8			
Sachlich richtige Lösungsalternative zur Modelllösung: (8) …… ……					
	Summe Teilaufgabe a)	**8**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	implementiert den Durchlauf durch die Liste.	6			
2	implementiert das Setzen des Zeitstempels.	2			
3	implementiert den Sonderfall der leeren Liste.	2			
Sachlich richtige Lösungsalternative zur Modelllösung: (10) …… ……					
	Summe Teilaufgabe b)	**10**			

[2] EK = Erstkorrektur; ZK = Zweitkorrektur; DK = Drittkorrektur

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	analysiert die Methode und ermittelt die Liste nach Ausführung der Methode.	8			
2	erläutert, was die Methode im Sachzusammenhang bewirkt.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (12) …… ……					
	Summe Teilaufgabe c)	**12**			

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	implementiert die Methode.	10			
Sachlich richtige Lösungsalternative zur Modelllösung: (10) …… ……					
	Summe Teilaufgabe d)	**10**			

Teilaufgabe e)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	beurteilt einen möglichen Einfüge-Vorgang bei Verwendung eines `List`-Objekts.	4			
2	beurteilt einen möglichen Einfüge-Vorgang bei Verwendung dreier `Queue`-Objekte.	6			
Sachlich richtige Lösungsalternative zur Modelllösung: (10) …… ……					
	Summe Teilaufgabe e)	**10**			

	Summe insgesamt	**50**			

Festlegung der Gesamtnote (Bitte nur bei der letzten bearbeiteten Aufgabe ausfüllen.)

	Lösungsqualität			
	maximal erreichbare Punktzahl	EK	ZK	DK
Übertrag der Punktsumme aus der ersten bearbeiteten Aufgabe	**50**			
Übertrag der Punktsumme aus der zweiten bearbeiteten Aufgabe	**50**			
Übertrag der Punktsumme aus der dritten bearbeiteten Aufgabe	**50**			
Punktzahl der gesamten Prüfungsleistung	**150**			
aus der Punktsumme resultierende Note gemäß nachfolgender Tabelle				
Note ggf. unter Absenkung um bis zu zwei Notenpunkte gemäß § 13 Abs. 2 APO-GOSt				
Paraphe				

ggf. arithmetisches Mittel der Punktsummen aus EK und ZK: _____

ggf. arithmetisches Mittel der Notenurteile aus EK und ZK: _____

Die Klausur wird abschließend mit der Note: _____ (____ Punkte) bewertet.

Unterschrift, Datum:

Grundsätze für die Bewertung (Notenfindung)

Für die Zuordnung der Notenstufen zu den Punktzahlen ist folgende Tabelle zu verwenden:

Note	**Punkte**	**Erreichte Punktzahl**
sehr gut plus	15	150 – 143
sehr gut	14	142 – 135
sehr gut minus	13	134 – 128
gut plus	12	127 – 120
gut	11	119 – 113
gut minus	10	112 – 105
befriedigend plus	9	104 – 98
befriedigend	8	97 – 90
befriedigend minus	7	89 – 83
ausreichend plus	6	82 – 75
ausreichend	5	74 – 68
ausreichend minus	4	67 – 60
mangelhaft plus	3	59 – 50
mangelhaft	2	49 – 40
mangelhaft minus	1	39 – 30
ungenügend	0	29 – 0