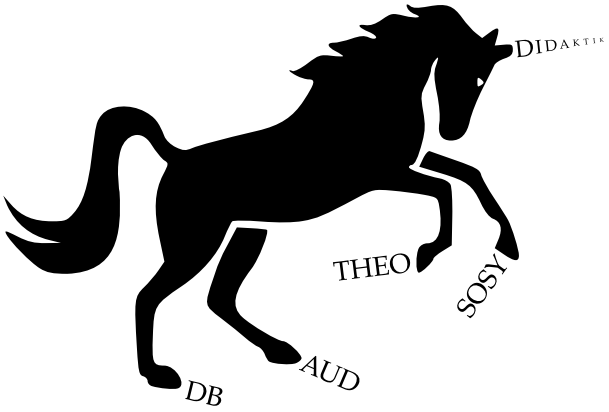
66115
Frühjahr
2007

Theoretische Informatik / Algorithmen (vertieft)

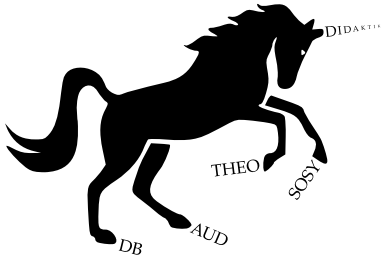
Aufgabenstellungen mit Lösungsvorschlägen



Die Bschlangaul-Sammlung

Hermine Bschlangaul and Friends

Aufgabenübersicht



Die Bschlangaul-Sammlung
Hermine Bschlangaul and Friends

Eine freie Aufgabensammlung mit Lösungen von Studierenden für Studierende zur Vorbereitung auf die 1. Staatsexamensprüfungen des Lehramts Informatik in Bayern.

Thema Nr. 1

Aufgabe 7 [Klassen „QueueElement“ und „Queue“]

Implementieren Sie die angegebenen Methoden einer Klasse `Queue` für Warteschlangen. Eine Warteschlange soll eine unbeschränkte Anzahl von Elementen aufnehmen können. Elemente sollen am Ende der Warteschlange angefügt und am Anfang aus ihr entfernt werden. Sie können davon ausgehen, dass ein Klasse `QueueElement` mit der folgenden Schnittstelle bereits implementiert ist .

```
class QueueElement {

  private QueueElement next;
  private Object contents;

  QueueElement(Object contents) {
    this.contents = contents;
  }

  Object getContents() {
    return contents;
  }

  QueueElement getNext() {
    return next;
  }

  void setNext(QueueElement next) {
    this.next = next;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2007/fruehjahr/queue/QueueElement.java`

Von der Klasse `Queue` ist folgendes gegeben:

```
class Queue {
  QueueElement first;
  QueueElement last;
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2007/fruehjahr/queue/Queue.java`

(a) Schreiben Sie eine Methode `void append (Object contents)`, die ein neues Objekt in der Warteschlange einfügt.

Lösungsvorschlag

```
public void append(Object contents) {
  QueueElement newElement = new QueueElement(contents);
  if (first == null) {
    first = newElement;
    last = newElement;
  } else {
    // neues Element hinten anhängen
    last.setNext(newElement);
    // angehängtes Element ist Letztes
```

```
    last = last.getNext();
  }
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66115/jahr_2007/fruehjahr/queue/Queue.java

(b) Schreiben Sie eine Methode `Object remove()`, die ein Element aus der Warteschlange entfernt und dessen Inhalt zurückliefert. Berücksichtigen Sie, dass die Warteschlange leer sein könnte.

Lösungsvorschlag

```
public Object remove() {
  Object tmp = null;
  if (first != null) {
    // Dein Inhalt des ersten Elements temporär speichern
    tmp = first.getContents();
    // Das erste Element aus der Schlange nehmen
    first = first.getNext();
  }
  // Den Inhalt des gelöschten Elements ausgeben bzw . null
  return tmp;
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66115/jahr_2007/fruehjahr/queue/Queue.java

(c) Schreiben Sie eine Methode `boolean isEmpty()`, die überprüft, ob die Warteschlange leer ist.

Lösungsvorschlag

```
public boolean isEmpty() {
  return (first == null);
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66115/jahr_2007/fruehjahr/queue/Queue.java

Klasse Queue

```
class Queue {
  QueueElement first;
  QueueElement last;

  public void append(Object contents) {
    QueueElement newElement = new QueueElement(contents);
    if (first == null) {
      first = newElement;
      last = newElement;
    } else {
      // neues Element hinten anhängen
      last.setNext(newElement);
      // angehängtes Element ist Letztes
      last = last.getNext();
    }
  }
```

```
  public Object remove() {
    Object tmp = null;
    if (first != null) {
      // Dein Inhalt des ersten Elements temporär speichern
      tmp = first.getContents();
      // Das erste Element aus der Schlange nehmen
      first = first.getNext();
    }
    // Den Inhalt des gelöschten Elements ausgeben bzw . null
    return tmp;
  }

  public boolean isEmpty() {
    return (first == null);
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2007/fruehjahr/queue/Queue.java`

Tests

```
import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class QueueTest {

  @Test
  public void methodAppend() {
    Queue queue = new Queue();
    assertEquals(true, queue.isEmpty());
    queue.append(1);
    assertEquals(false, queue.isEmpty());
  }

  @Test
  public void methodRemove() {
    Queue queue = new Queue();
    queue.append(1);
    queue.append(2);
    queue.append(3);

    assertEquals(1, queue.remove());
    assertEquals(2, queue.remove());
    assertEquals(3, queue.remove());
    assertEquals(null, queue.remove());
  }

  @Test
  public void methodIsEmpty() {
    Queue queue = new Queue();
    assertEquals(true, queue.isEmpty());
  }

}
```

Code-Beispiel auf Github ansehen: `src/test/java/org/bschlangaul/examen/examen_66115/jahr_2007/fruehjahr/queue/QueueTest.java`

Thema Nr. 2

Aufgabe 1 [Cent-Münzen]

(a) Beschreiben Sie in Pseudocode oder einer Programmiersprache Ihrer Wahl einen Greedy-Algorithmus, der einen Betrag von n Cents mit möglichst wenigen Cent-Münzen herausgibt. Bei $n = 29$ wäre die erwartete Antwort etwa 1×20ct, 1×5ct, 2×2ct.

(b) Beweisen Sie die Korrektheit Ihres Verfahrens, also dass tatsächlich die Anzahl der Münzen minimiert wird.

(c) Nehmen wir an, Bayern führe eine Sondermünze im Wert von 7ct ein. Dann liefert der naheliegende Greedy-Algorithmus nicht immer die minimale Zahl von Münzen. Geben Sie für dieses Phänomen ein konkretes Beispiel an und führen Sie aus, warum Ihr Beweis aus Aufgabenteil a) in dieser Situation nicht funktioniert.