Prof. Dr. Frank Victor

Technology
Arts Sciences
TH Köln

Praktikum 4

Aufgabe 1: Programmieren in Java

Schauen Sie sich das untenstehende Java Programm und beschreiben Sie **mündlich**, was hier passiert. **Wie sieht die Ausgabe aus? Warum?**

```
 1 public class VictorsArray {
 2     public static void main(String[]args) {
 3         String[] VictorsLieblingsObst = {"Birne", "Apfel", "Kiwi"};
 4
 5         System.out.println(VictorsLieblingsObst[1]);
 6
 7         int[] VictorsLieblingsZahlen = new int[7];
 8         VictorsLieblingsZahlen[0] = 7;
 9
10         for(int i = 1; i < VictorsLieblingsZahlen.length; i++){
11             VictorsLieblingsZahlen[i] = i + 3;
12         }
13
14         System.out.printf("%nVictors Lieblingszahlen: ");
15         for (int each: VictorsLieblingsZahlen)
16             System.out.print(each + " ");
17
18         int[] KohlsLieblingsZahlen = {2 , 5, 58, 4, 3, 444, 25};
19         VictorsLieblingsZahlen = KohlsLieblingsZahlen;
20
21         VictorsLieblingsZahlen[1] = 100;
22
23         System.out.printf("%nKohls Lieblingszahlen: ");
24         for(int each: KohlsLieblingsZahlen)
25             System.out.print(each + " ");
26     }
27 }
```

Aufgabe 2: Programmieren in Java

Schreiben Sie eine **rekursive Methode** für die *Ackermann-Funktion.*
Die Ackermann-Funktion ist eine Abbildung von zwei ganzen Zahlen `n` und `m` auf eine ganze Zahl `a(n,m)`:

```
a(0,m) = m + 1
a(n,0) = a(n - 1, 1)
a(n,m) = a(n - 1, a(n, m - 1))
```

Schreiben Sie in die gleiche Klasse auch eine `main`-Methode, die zwei Zahlen von der Tastatur einliest und dann das Ackermann-Ergebnis für diese Zahlen ausgibt. Bitte testen Sie, ab wann der Berechnungsaufwand zu groß wird, so dass wir nicht auf das Ergebnis warten können.

Aufgabe 3: Programmieren in Java

Zur Lösung dieser Aufgabe ist etwas Recherche-Arbeit von Ihnen zu leisten. Besorgen Sie sich Informationen über das Thema **Binäre Suche**. Schauen Sie dazu in die Literatur oder ins Internet. Beschreiben Sie bitte zunächst auf einer Seite, wie die Binäre Suche funktioniert und welchen Aufwand die Methode hat (O-Notation).

Entwickeln und testen Sie bitte eine Java-Methode **`binsearch`**, die eine binäre Suche auf einem **`int`**-Array realisiert. Die Methode gibt die Position eines gesuchten Wertes im **`int`**-Array zurück bzw. -1, wenn der Wert nicht vorkommt.

Aufgabe 4: Programmieren in Java

Schreiben Sie ein Java-Programm, das x^y berechnet, wenn die Zahlen **`x`** und **`y`** eingegeben werden. Verwenden Sie hierzu die Methode **`pow`** der Java Klasse **`Math`**.

Hinweise:

- Sie müssen sich in der Java API die Methode **`pow`** ansehen, um zu sehen, wie sie aufgerufen wird. Wie findet man die Klasse **`Math`** in der Java 13 API?
- Wenn Sie ohne **`import`** arbeiten wollen, dann funktioniert das mit **`Math.pow(x,y)`**.
- Wenn Sie **`import`** verwenden möchten, damit der Aufruf nur noch **`pow(x,y)`** heißt, versuchen Sie es mit **`import static java.lang.Math.*;`**

Die Ausgabe des Programms sollte in etwa so aussehen:

```
Programm zur Berechnung der Potenz
Bitte geben Sie x ein: ___________
Bitte geben Sie y ein: ___________
Die Potenz von x hoch y ist: ______ .
```

Aufgabe 5: Programmieren in Java

Kopieren Sie Ihr Programm aus Aufgabe 4 und fügen Sie eine kleine Änderung hinzu. Die Lokalisierung der **`advm1`**-Maschine (**`US`**) soll überschrieben werden. Das Programm soll für die deutsche Schreibweise für Dezimalzahlen funktionieren (Komma anstatt von Punkt).

Aufgabe 6: Programmieren in Java

Definieren Sie eine Klasse **`Queue`**. Eine **`Queue`** (deutsch: Schlange) ist eine Datenstruktur, bei der man das erste Element entnehmen kann (**`dequeue`**). Ein Datenelement kann nur an das Ende der Schlange angefügt werden (**`enqueue`**). Zusätzlich sind in der Klasse die Initialisierung (über den Konstruktor) und die Abfragen, ob die Schlange leer (**`is_empty()`**) bzw. voll (**`is_full()`**), definiert. Die Schlange soll **Integerzahlen** aufnehmen können und hat die folgenden Instanzvariablen:

```
private int nextFree;
private int[] arr;
```

Dabei gibt **`nextFree`** den Index der ersten freien Stelle in der Schlange an und das Array enthält die Elemente der Schlange. Freie Plätze werden enthalten eine 0.

Beispiel:

Schlange[5]: 10 20 30 0 0 **`nextFree = 3`**

(1) s.dequeue() lässt 10 die Schlange verlassen:
20 30 0 0 0 **`nextFree = 2`**

(2) s.enqueue(100) fügt 100 an das Ende der Schlange ein:
20 30 100 0 0 **`nextFree = 3`**

(3) s.dequeue()
s.dequeue()
s.dequeue()
s.is_empty() liefert **`true`**

0	0	0	0	0	**`nextFree = 0`**

Neben der Implementierung der Klasse **`Queue`** besteht Ihre Aufgabe darin, eine Anwendung zu schreiben, die eine Warteschlange für Integerzahlen simuliert. Formulieren Sie in der Anwendung 3 Szenarien so wie im Beispiel dargestellt, die zeigen, dass die Schlange ordnungsgemäß funktioniert und Fehlermeldungen wie „Schlange voll“ und „Schlange leer“ ausgibt.

Aufgabe 7: Programmieren in Java

Verwenden Sie die **`Queue`** und die Anwendung aus Aufgabe 6 in dieser letzten Aufgabe. Sie können die Schlange und die Anwendung kopieren und nach Ihren Bedürfnissen anpassen.
Die Aufgabe besteht darin, einen Taxistand zu implementieren und zu simulieren.
Es gibt insgesamt **8 Taxis** in der Stadt und **1 Taxistand**. Der Taxistand funktioniert als FIFO Struktur (Schlange, Queue, s.o.).

Der folgende Code kann für Sie hilfreich sein:

Die Klasse `Taxi` ist vorgegeben:

```
/**
 * Ein Taxi ist ein Objekt, das aus dem Fahrernamen, dem Kennzeichen und
 * einer eindeutigen Nummer besteht.
 */
public class Taxi {
    private String namefahrer;
    private String kennzeichen;
    private int nummer;

    /**
     * Der Konstruktor legt ein Taxi Objekt an und speichert darin den
     * Fahrernamen, das Kennzeichen und die Taxinummer.
     */
    public Taxi(String namefahrer, String kennzeichen, int nummer) {
        this.namefahrer = namefahrer;
        this.kennzeichen = kennzeichen;
        this.nummer = nummer;
    }

    /**
     * Diese Methoden verwenden wir zur Ausgabe der Meldungen in der Klasse
     * Schlange.
     */
    public String getnamefahrer() {
        return namefahrer;
    }

    public String getkennzeichen() {
        return kennzeichen;
    }

    public int getnummer() {
        return nummer;
    }
}
```

Die Klasse `Schlange` hat die folgenden Instanzvariablen und den folgenden Konstruktor:

```
public class Schlange {
    private int nextFree;
    private Taxi[] arr;

    public Schlange(int nextFree) {
      this.nextFree = nextFree;
      arr = new Taxi[5];
      …
    }
…
```

Die Klasse `App` hat folgenden Aufbau:

```
public class App {
    public static void main(String args[]) {
        Taxi a = new Taxi(…);
        Taxi b = new Taxi(…);
        …
        Schlange taxistand = new Schlange(0);       // Nächster freier Platz
                                                    // ist im Index 0

        System.out.println("Ausgangssituation");
        taxistand.clear();      // leert den Taxistand
        taxistand.ausgeben();   // zeigt den Taxistand (siehe Ausgabe)

        System.out.println("\1. Situation");
        taxistand.clear();      // leert den Taxistand
        taxistand.enqueue(a);
        taxistand.enqueue(b);
        taxistand.enqueue(c);
        taxistand.enqueue(d);
        taxistand.enqueue(e);
        taxistand.enqueue(f);
        taxistand.ausgeben();
        …
```

Die **Ausgabe** sieht dann in etwa so aus:

```
Ausgangssituation
Alle 5 Plätze sind leer.

Taxistand
frei frei frei frei frei

1. Situation
Alle 5 Plätze sind leer.
Das Taxi: 1, Frank Victor, BN – FV 300 fährt auf Platz 1
Das Taxi: 2, Angela Merkel, B - DE 001 fährt auf Platz 2
Das Taxi: 3, James Bond, BN - JB 007 fährt auf Platz 3
Das Taxi: 4, Manuel Neuer, M - MN 001 fährt auf Platz 4
Das Taxi: 5, Angelique Kerber, BN - AK 111 fährt auf Platz 5
Fehler: Das Taxi: 6, Boris Becker, M - BB 4911 kann nicht einfahren! Der Taxistand
ist picke packe voll!

Taxistand
1 2 3 4 5
```

Hinweise:

- Die Schlange des Taxistands und damit das Array enthält Objekte, und zwar Taxis. Wenn ein Platz nicht besetzt ist, verwenden Sie bitte die `null`-Referenz. Sie könnten also schreiben `arr[nextFree] = null;` wenn ein Platz frei wird.
- Die Methode `public void enqueue(Taxi x)` ist ganz leicht zu schreiben. Wenn der Stand nicht voll ist, wird `arr[nextFree] = x;` gesetzt. Sonst wird eine Meldung ausgegeben (siehe Ausgabe).
- Die Methode `public void dequeue()` ist etwas aufwändiger. Sie gibt aus, welches Taxi den Stand verlassen hat. Die anderen Taxis, die warten, müssen nachrücken.
- Verwenden Sie bitte `javadoc` Kommentare für die Klassen und Methoden!

Verständnisfragen im Praktikum

Im Praktikum werden sich die BetreuerInnen Ihren Code ansehen und die Lösungen mit Ihnen diskutieren. Zusätzlich werden Fragen zur Vorlesung gestellt. Bereiten Sie sich auf die untenstehenden Themen vor. Alle Fragen sind aus dem Kopf zu beantworten. **Unterlagen zur Beantwortung der Fragen sind im Praktikum nicht erlaubt.**

- Rekursion
- Klassen und Objekte
- private und public
- javadoc
- Konstruktoren
- this-Referenz zur Vermeidung von Namenskonflikten (insbesondere in Konstruktoren)
- Arrays und Strings in Java

Das war das letzte Aufgabenblatt in AP I. Viel Erfolg!

Wir wünschen Ihnen frohe Weihnachten und ein gutes neues Jahr!