Prof. Dr. Stephan Kleuker
Hochschule Osnabrück
Fakultät Ing.-Wissenschaften und Informatik
- Software-Entwicklung -

Programmierung 1

Wintersemester 2023/24
Aufgabenblatt 12

Nur zum Selbststudium (aufwändiger als ein normales Aufgabenblatt)!

38. Aufgabe (Exceptions analysieren)

```
public class Analyse {
  public int check(int x)
                throws MeineException {
    EinUndAusgabe io
        = new EinUndAusgabe();
    try {
      io.ausgeben("A");
      if (x < 20) {
        throw new MeineException(x);
      }
      io.ausgeben("B");
      if (x < 62) {
        io.ausgeben("C");
        return x;
      } else {
        io.ausgeben("D");
        throw new MeineException(x-10);
      }
    } catch (MeineException e) {
      try {
        io.ausgeben("E");
        if (e.getStufe() > 72) {
          throw new MeineException(10);
        }
        io.ausgeben("F");
        if (e.getStufe() < 62) {
          io.ausgeben("G");
        } else {
          io.ausgeben("H");
          throw new MeineException(x);
        }
      } finally {
        io.ausgeben("Y");
      }
    } finally {
      io.ausgeben("Z");
    }
    io.ausgeben("I");
    return -42;
  }
}
```

```
public class MeineException extends
Exception {

  private int stufe;

  public MeineException(int wert) {
    super();
    this.stufe = wert;
  }

  public int getStufe() {
    return this.stufe;
  }
}
```

```
public class Main {

  public static void main(
                        String[] args) {
    EinUndAusgabe io
        = new EinUndAusgabe();
    int[] test = {15, 35, 65, 75, 85};
    Analyse a = new Analyse();
    for(int t: test) {
      io.ausgeben(t +": ");
      try {
        io.ausgeben(a.check(t) + "\n");
      } catch (Exception e) {
        io.ausgeben("Schicht\n");
      }
    }
  }
}
```

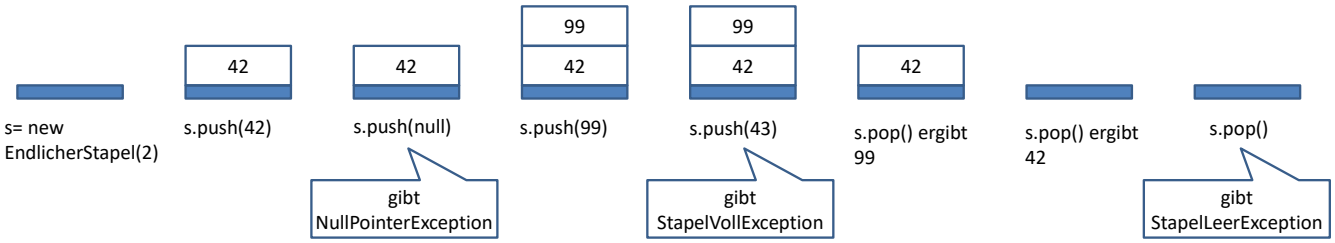
Geben Sie die Ausgaben des obigen Programms an.

Prof. Dr. Stephan Kleuker
Hochschule Osnabrück
Fakultät Ing.-Wissenschaften und Informatik
- Software-Entwicklung -

Programmierung 1

Wintersemester 2023/24
Aufgabenblatt 12

39. Aufgabe (Stapel-Variante, Umgang mit Exceptions, Testen)



Ein Stapel (Stack) ist eine Datenstruktur, die Objekte eines bestimmten Typs aufnehmen kann. Dabei werden Objekte mit einer Methode push(.) so auf den Stapel gelegt, dass das neue Element immer oben liegt. Mit der Methode pop(.) wird das oberste Element zurückgegeben und vom Stapel gelöscht. Weiterhin ist es sinnvoll, wenn es Methoden istLeer() bzw. istVoll() gibt, mit denen man prüfen kann, ob ein Stapel voll oder leer ist.

a) Realisieren Sie eine Klasse EndlicherStapel für Integer-Objekte, mit einem Konstruktor, der die mögliche Größe des Stapels als Parameter hat. *Nutzen Sie zur Realisierung des Stapels einen Array*.
b) Realisieren Sie eine Methode push(Integer), mit der ein Integer-Objekt auf den Stapel gelegt wird. Wird versucht, eine null-Referenz auf den Stapel zu legen, soll eine NullPointerException geworfen werden. Wird versucht ein Objekt auf einen vollen Stapel zu legen, soll eine von Ihren neu zu erstellende StapelVollException geworfen werden.
c) Realisieren Sie die Methode pop(), die das zuletzt auf den Stapel gelegte Integer-Objekt zurück gibt und vom Stapel löscht. Wird versucht, ein Objekt von einem leeren Stapel zu nehmen, soll eine von Ihren neu zu erstellende StapelLeerException geworfen werden.
d) Realisieren Sie die Methoden istVoll() und istLeer() zur Überprüfung des Stapelzustands.
e) Schreiben Sie zu allen realisierten Methoden JUnit-Tests, die alle möglichen Ergebnisse testen. Nutzen Sie dazu eine Test Fixture, also mehrere Objekte, mit einem leeren, einem vollen und einem „normal“ gefüllten Stack, auf denen Sie jeweils alle Ihre Methoden ausprobieren.
f) Kopieren Sie Ihr bisheriges Projekt in ein neues Projekt und ändern Sie Ihre Implementierung so ab, dass alle Tests aus e) erfüllt werden, sich aber trotzdem ein Fehler im Programm befindet. Ergänzen Sie dann einen Test, der den eingebauten Fehler findet.

Hinweis: Die Methoden push(.) und pop() sollen die Exceptions nur werfen, nicht selber bearbeiten. In den Tests ist mit try-catch-Blöcken zu prüfen, ob die richtigen Exceptions geworfen werden und die Methoden sonst die korrekten Ergebnisse liefern.

40. Aufgabe (Set, equals, hashCode)

Gegeben Seien folgende Klassen.

```
public class Studierend {
  private String name;
  private int matnr;

  public Studierend(String name, int matnr) {
    this.name = name;
    this.matnr = matnr;
  }
```

```
  @Override
  public String toString() {
    return "Studierend [name=" + name + ", matnr=" + matnr + "]";
  }
}
```

```
import java.util.HashSet;
import java.util.Set;

public class Main {

  public static void main(String[] args) {
    Set<Studierend> studis = new HashSet<Studierend>();
    Studierend s = new Studierend ("Olga", 100);
    studis.add(s);
    studis.add(new Studierend("Sandra", 101));
    studis.add(s);
    studis.add(new Studierend("Olga", 100));
    System.out.println(studis);
  }
}
```

Kopieren Sie die Klassen in vier verschiedene Projekte und ergänzen Sie die Klasse Studierend jeweils um eine equals- und eine hashCode-Methode, so dass folgende Ausgaben (eventuell mit anderer Reihenfolge der Objekte) ausgegeben werden.
Hinweis: Nur die letzte Ausgabe gehört zur vollständig korrekten Implementierung der Methoden, für die anderen Ausgaben sind die Methoden bewusst geschickt falsch zu programmieren.

a) `[Studierend [name=Olga, matnr=100], Studierend [name=Sandra, matnr=101], Studierend [name=Olga, matnr=100]]`
b) `[Studierend [name=Olga, matnr=100], Studierend [name=Sandra, matnr=101], Studierend [name=Olga, matnr=100], Studierend [name=Olga, matnr=100]]`
c) `[Studierend [name=Olga, matnr=100]]`
d) `[Studierend [name=Olga, matnr=100], Studierend [name=Sandra, matnr=101]]`

41. Aufgabe (Bearbeitung von Listen)

Zu entwickeln ist ein Programm zur Analyse der Wahrscheinlichkeiten von Poker-Blättern.

a) Schreiben Sie eine Klasse Karte mit den Objektvariablen Farbe mit den möglichen Werten „KARO“, „HERZ“, „KREUZ“, „PIK“ und Wert mit den möglichen Werten (in aufsteigender Reihenfolge) „SIEBEN“, „ACHT“, „NEUN“, „ZEHN“, „BUBE“, „DAME“, „KOENIG“, „AS“.
b) Ergänzen Sie eine toString()-Methode, so dass z. B. die Farbe als „Karo“ und der Wert als „7“ ausgegeben werden.

c) Schreiben Sie eine Klasse Kartenstapel, mit einer Klassenmethode gibAlleKarten(), die einen Kartenstapel mit den 32 Karten zurückgibt und folgende Objektmethoden anbietet. (Wieder: Methode entwickeln, dann ausprobieren/testen)

Methode	Funktionalität
`toString()`	schöne Ausgabe
`mischen()`	mischt den Stapel, dabei soll nicht die Methode shuffle() der Klasse ArrayList genutzt werden
`fuenfGeben()`	gibt ein neues Kartenstapel-Objekt mit den ersten fünf Karten zurück
`istPaar()`	gibt es zwei Karten im Kartenstapel mit gleichem Wert?
`istZweiPaar()`	gibt es zweimal zwei Karten mit gleichem Wert?
`istDrilling()`	gibt es drei Karten mit gleichem Wert?
`istStrasse()`	handelt es sich um fünf Karten mit direkt aufsteigenden Werten?
`istFlush()`	haben alle Karten die gleiche Farbe?
`istFullHouse()`	gibt es einen Drilling und ein zusätzliches Paar?
`istVierling()`	gibt es vier Karten mit gleichem Wert?
`istStraightFlush()`	ist es ein Flush in Straßenform?
`istRoyalFlush()`	ist es ein Straight-Flush, der ein As enthält?

Hinweis: Überlegen Sie, warum es geschickt sein könnte, die Karten vor der Analyse zu sortieren.

d) Schreiben Sie ein Programm, das 100000-mal neue gemischte Kartenstapel erzeugt, die ersten fünf Karten mit fuenfGeben() als Kartenstapel-Objekt abhebt, nach den Kriterien Paar bis Royal Flush untersucht und jeweils deren Auftreten zählt (ein Drilling soll dabei z. B. auch als Paar gezählt werden). Geben Sie weiterhin alle Straight Flushes aus. Abschließend sollen die Ergebnisse möglichst genau wie im folgenden Beispiel formatiert ausgegeben werden.

```
[Kreuz 8, Kreuz 9, Kreuz 10, Kreuz Bube, Kreuz Dame]
[Herz 8, Herz 9, Herz 10, Herz Bube, Herz Dame]
[Herz 8, Herz 9, Herz 10, Herz Bube, Herz Dame]
[Pik 10, Pik Bube, Pik Dame, Pik Koenig, Pik As]
 PAAR ZWEIPAAR DRILLING STRAIGHT FLUSH FULLHOUSE VIERLING STRAIGHTFLUSH ROYALFLUSH
71661    12783     6292     1949     95       692      119             4           1
```

e) Beim Pokern gibt es Varianten mit der Möglichkeit, Karten zu tauschen. Schreiben Sie eine Methode, die für gegebene fünf Karten berechnet, welche Karten gegen Karten aus dem Kartenstapel getauscht werden sollen. Lassen Sie wieder ein Simulationsprogramm mit 100000 Läufen laufen, das die Verbesserungen durch Ihr Tauschverfahren zählt. Beispiele: Verbesserung von PAAR auf ZWEIPAAR bringt einen Punkt, von PAAR auf DRILLING bringt zwei Punkte. Versuchen Sie ihr Tauschverfahren zu optimieren (Ziel ca. 37000 Punkte).