Programmierkurs Java

Dr. Dietrich Boles

Aufgaben zu UE35-Exceptions
(Stand 28.09.2012)

Aufgabe 1:

In dieser Aufgabe geht es um die Ausgabe von Fahrplänen eines Busnetzes. Dazu soll ein bereit gestelltes Paket genutzt werden. Gegeben sind folgende Klassen:

- drei Exceptionklassen
- eine Klasse *Haltestelle*, die Bushaltestellen repräsentiert
- eine Klasse *Busnetz*, die ein Busnetz repräsentiert
- eine Hilfsklasse *HaltestellenLinie*.

Im Einzelnen sind die Klassen folgendermaßen implementiert bzw. haben folgendes Protokoll:

```
// in Datei UnbekannteHaltestelleException.java
package bus;

public class UnbekannteHaltestelleException extends Exception {
	private String name;

	public UnbekannteHaltestelleException(String n) {
		this.name = n;
	}

	public String getName() {
		return name;
	}
}

// in Datei KeineVerbindungException.java
package bus;

public class KeineVerbindungException extends Exception {
}

// in Datei KeinNachbarException.java
package bus;

public class KeinNachbarException extends Exception {
}

// in Datei Haltestelle.java
package bus;

// die Klasse repraesentiert eine Haltestelle
public class Haltestelle {
	// Haltestellen koennen nicht explizit erzeugt werden
	private Haltestelle() {...}

	// liefert zu einem Namen das entsprechende Haltestellen-Objekt;
	// die Erzeugung erfolgt intern (feste Codierung!);
	// die Exception wird geworfen, wenn zu dem Namen keine Haltestelle
	// existiert
```

```
        public static Haltestelle getHaltestelle(String name)
                throws UnbekannteHaltestelleException {...}

        // liefert den Namen einer Haltestelle
        public String getName() {...}

        // liefert die Fahrzeit in Minuten zwischen der aufgerufenen
        // Haltestelle und der als Parameter uebergebenen Haltestelle;
        // die Exception wird geworfen, wenn die uebergebene Haltestelle
        // keine Nachbarhaltestelle der aufgerufenen Haltestelle ist
        public int getFahrzeit(Haltestelle nachbarStelle)
                throws KeinNachbarException {...}

        // liefert die Liniennummern, die die Haltestelle anfahren
        public int[] getLinien() {...}
}

// in Datei Haltestellenlinie.java
package bus;

// Hilfsklasse zum Speichern einer Haltestelle mit Liniennummer
public class HaltestellenLinie {
        private Haltestelle haltestelle;

        private int linie;

        public HaltestellenLinie(Haltestelle h, int l) {
                this.haltestelle = h;
                this.linie = l;
        }

        public Haltestelle getHaltestelle() {
                return haltestelle;
        }

        public int getLinie() {
                return linie;
        }
}

// in Datei Busnetz.java
package bus;

public class Busnetz {

        // initialisiert ein Busnetz
        public Busnetz() {...}

        // liefert Informationen zur Verbindung der Haltestellen "von" nach
        // "nach";
        // in den Elementen des gelieferten Arrays stehen in der
        // entsprechenden Reihenfolge die nacheinander angefahrenen
        // Haltestellen (inkl. der von-Haltestelle (erstes Element) und der
        // nach-Haltestelle (letztes
        // Element)) sowie die jeweils zu benutzende Linie;
        // die Exception wird geworfen, wenn keine Verbindung existiert
        public HaltestellenLinie[] getVerbindung(Haltestelle von,
                                                 Haltestelle nach)
                throws KeineVerbindungException {...}
}
```

Aufgabe:

Teilaufgabe (a):
Implementieren Sie die angegebenen Klassen!

Teilaufgabe (b):

Schreiben Sie ein Java-Programm `Fahrplaene`, das folgendes tut:
Beim Aufruf können dem Programm beliebig viele Namen als Parameter übergeben werden, die Haltestellennamen entsprechen sollen; Beispiel:
`"java Fahrplaene Uni OFFIS Melkbrink LKH-Wehnen".`

Für je zwei hintereinander stehende Namen sollen mit Hilfe der Methode `getVerbindung` der Klasse `Busnetz` Verbindungen ermittelt werden; im Beispiel also zunächst für die Strecke von Uni nach OFFIS und anschließend für die Verbindung von Melkbrink nach LKH-Wehnen. Nach der Ermittlung einer Verbindung sollen daraus durch Nutzung der Methoden der bereit gestellten Klassen Fahrpläne in folgender Form auf den Bildschirm ausgegeben werden:

```
Strecke <name 1> - <name 2>:
Start bei <name 1> (Linie <linie1>)
<haltestelle 1> <n 1> Minuten
<haltestelle 2> <n 2> Minuten
...
<name 2> <n n> Minuten
```

wobei die Minutenangabe der insgesamt bis dahin zurück gelegten Fahrzeit entsprechen soll.

Ist ein Umsteigen notwendig (Linienwechsel), soll zusätzlich eine Zeile in folgender Form ausgegeben werden:

```
Umsteigen in Linie <linie>
```

Existiert zu einem Namen keine Haltestelle, soll ausgegeben werden:

```
Haltestelle <name> ist unbekannt
```

und das entsprechende Parameterpaar soll nicht weiter betrachtet werden.

Existiert keine Verbindung, soll ausgeben werden:

```
Keine Verbindung zwischen <name 1> und <name 2>
```

Beispiel: Eine exemplarische Ausgabe beim Aufruf von
`java Fahrplaene Melkbrink LKH-Wehnen Uni OFFIS Bahnhof Lappan`:

```
Haltestelle Melkbrink ist unbekannt

Strecke Uni - OFFIS:
Start bei Uni (Linie 310)
Ofener Strasse 2 Minuten
Julius-Mosen-Platz 7 Minuten
Umsteigen in Linie 45
Peterstrasse 10 Minuten
OFFIS 16 Minuten

Keine Verbindung zwischen Bahnhof und Lappan
```

Aufgabe 2:

In dieser Aufgabe geht es um die Simulation der Buchausleihe in einer Bibliothek. Dazu soll ein bereitgestelltes Paket namens *bibliothek* genutzt werden. Das Paket enthält folgende Klassen:

- zwei Exception-Klassen
- eine Klasse *Nutzer*, die einen Bibliotheksnutzer repräsentiert
- eine Klasse *Buch*, die ein Buchexemplar der Bibliothek repräsentiert
- eine Klasse *BibliothekAusleihe*, die die Ausleihe einer Bibliothek repräsentiert
- eine Klasse *AusgeliehenesBuch*, die ausgeliehene Bücher repräsentiert.

Die Klassen haben dabei folgende Gestalt:

```
//-------------------------------------------------------------
// in Datei UnbekannterNutzerException.java
package bibliothek;

public class UnbekannterNutzerException extends Exception {}


//-------------------------------------------------------------
// in Datei KeineAusgeliehenenBuecherException.java
package bibliothek;

public class KeineAusgeliehenenBuecherException extends Exception {
  Nutzer nutzer; // Nutzer, der aktuell keine Buecher ausgeliehen
hat

  public KeineAusgeliehenenBuecherException(Nutzer nutzer) {
    this.nutzer = nutzer;
  }

  // liefert den Nutzer, der aktuell keine Buecher ausgeliehen hat
  public Nutzer getNutzer() { return this.nutzer; }
}


//-------------------------------------------------------------
// in Datei Nutzer.java
package bibliothek;

// Die Klasse Nutzer repraesentiert einen Bibliotheksnutzer
pulic class Nutzer {
  String name;          // Nutzer haben einen Namen
  int benutzungsnummer; // Nutzern wird intern eine eindeutige
                        // Benutzungsnummer zugeordnet

  // Nutzer koennen nicht explizit erzeugt werden
  private Nutzer() { ... }




  // liefert das dem uebergebenen Namen zugeordnete Nutzer-Objekt
  // wirft eine Exception, wenn zu dem Namen kein angemeldeter
```

```
  // Nutzer existiert
  public static Nutzer getNutzer(String name)
    throws UnbekannterNutzerException  { ... }

  // liefert den Namen des Nutzers
  public String getName() { return name; }

  // liefert die Benutzungsnummer des Nutzers
  public int getBenutzungsnummer() { return benutzungsnummer; }
}

//-------------------------------------------------------------
// in Datei Buch.java
package bibliothek;

// Die Klasse Buch repraesentiert ein Buchexemplar der Bibliothek
public class Buch {
  String autor;  // Name des Autors des Buches
  String titel;  // Titel des Buches

  // liefert den Autorennamen
  public String getAutor() { return autor; }

  // liefert den Titel des Buches
  public String getTitel() { return titel; }
}

//-------------------------------------------------------------
// in Datei AusgeliehenesBuch.java
package bibliothek;

// Die Klasse AusgeliehenesBuch repraesentiert ein ausgeliehenes
// Buch
public class AusgeliehenesBuch {
  Buch buch;     // das ausgeliehene Buch
  Nutzer nutzer; // der Nutzer, der das Buch ausgeliehen hat
  int kosten;    // angefallene Ausleihekosten (in vollen EUR)
                 // fuer das Buch

  // liefert das ausgeliehene Buch
  public Buch getBuch() { return buch; }

  // liefert den Nutzer, der das Buch ausgeliehen hat
```

```
  public Nutzer getNutzer() { return nutzer; }

  // liefert die aktuell angefallenen Ausleihkosten (in vollen EUR)
  // fuer das Buch
  public int getKosten() { return kosten; }
}



//-------------------------------------------------------------
// in Datei BibliothekAusleihe.java
package bibliothek;

// Die Klasse BibliothekAusleihe repraesentiert die Ausleihe einer
// Bibliothek
public class BibliothekAusleihe {
  public BibliothekAusleihe() { ... }

  public AusgeliehenesBuch[] getAusgelieheneBuecher(Nutzer nutzer)
    throws KeineAusgeliehenenBuecherException  { ... }
}
```

Aufgabe:

Schreiben Sie ein Java-Programm *Ausleihe*, das durch Nutzung der Klassen des Paketes *bibliothek* folgendes tut:

Beim Aufruf können dem Programm beliebig viele Namen (ohne Leerzeichen!) als Parameter übergeben werden, die Namen von Nutzern der Bibliothek entsprechen sollen; Beispiel: "`java Ausleihe Boles Meier Mustermann`".

Für jeden angegebenen Nutzer soll eine Übersicht seiner aktuell ausgeliehenen Bücher sowie die aktuell angefallenen Ausleihekosten auf den Bildschirm ausgegeben werden, und zwar in folgender Form:

```
<Nutzername> hat aktuell <Anzahl ausgeliehener Buecher> Buecher ausgeliehen:
<Autor Buch1>: <Titel Buch1>
<Autor Buch2>: <Titel Buch2>
...
<Autor Buchn>: <Titel Buchn>
Angefallene Ausleihekosten: 23 EUR.
```

Wird ein nicht bekannter Nutzer als Parameter übergeben, soll ausgegeben werden:

```
Ein Nutzer mit dem Namen <angegebener Name> ist unbekannt.
```

Hat ein Nutzer aktuell keine Bücher ausgeliehen, soll ausgegeben werden:

```
Nutzer <angegebener Name> (Benutzungsnummer <Benutzungsnummer des Nutzers>) hat aktuell keine Buecher ausgeliehen.
```

Eine exemplarische Ausgabe für folgenden Aufruf des Programms:
`java Ausleihe Mueller Meier Schulze`:

```
Ein Nutzer mit dem Namen Mueller ist unbekannt.

Meier hat aktuell 2 Buecher ausgeliehen:
Dietrich Boles: Programmieren spielend gelernt
Joachim Goll: Java als erste Programmiersprache
Angefallene Ausleihekosten: 4 EUR.

Nutzer Schulze (Benutzungsnummer 4711) hat aktuell keine Buecher ausgeliehen.
```

Aufgabe 3:

Schauen Sie sich die folgenden Klassen an:

```
class IsEmptyException extends Exception {
}

class IsFullException extends Exception {
}

class Stack {

	private Object[] speicher;

	private int aktIndex;

	public Stack(int maxSize) {
		this.speicher = new Object[maxSize];
		this.aktIndex = -1;
	}

	public final void push(Object wert) throws IsFullException {
		if (this.aktIndex == this.speicher.length - 1) {
			throw new IsFullException();
		}
		this.speicher[++this.aktIndex] = wert;
	}

	public final Object pop() throws IsEmptyException {
		if (this.aktIndex == -1) {
```

```
                throw new IsEmptyException();
            }
            return this.speicher[this.aktIndex--];
        }

}
```

Die Klasse Stack realisiert einen Stack, auf den mittels der Methode push Object-Werte abgelegt werden können und mittels der Methode pop das jeweils oberste Element herunter geholt werden kann. Anstelle zweier Test-Methoden isFull und isEmpty werden diese beiden Fehlerfälle mit Hilfe zweier Fehlerklassen IsFullException und IsEmptyException behandelt.

Auf der Basis der Klasse Stack möchte nun ein anderer Programmierer eine Klasse SwapStack realisieren, die zusätzlich eine Methode swap zur Verfügung stellt, mit deren Hilfe die beiden obersten Elemente des Stacks (falls es überhaupt zwei gibt) vertauscht werden können.

Aufgabe:

1. Leiten Sie eine Klasse SwapStack von der Klasse Stack ab (die beiden Methoden push und pop werden geerbt).

2. Implementieren Sie (falls notwendig) einen Konstruktor für die Klasse SwapStack.

3. Implementieren Sie genau eine (!) zusätzliche Methode public final void swap() throws NotEnoughElementsException, die die beiden obersten Elemente des Stacks vertauscht. Falls beim Aufruf der Methode keine zwei Elemente auf dem Stack liegen, soll eine geeignet zu definierende NotEnoughElementsException geworfen werden. In diesem Fehlerfall darf sich der Zustand des Stacks nicht ändern, d.h. falls dieser Fehler auftritt bzw. entdeckt wird, müssen vorher durchgeführte Aktionen, die den Zustand des Stacks verändert haben, wieder rückgängig gemacht werden.

Hinweise:

- Die Klasse Stack darf nicht verändert werden!

- Achten Sie darauf, dass die beiden Attribute der Klasse Stack als private und die beiden Methoden der Klasse Stack als final deklariert sind.

- Außer der Methode swap darf keine weitere public-Methode für die Klasse SwapStack implementiert werden.

Aufgabe 4:

Gegeben sei folgendes Java-Programm:

```
class Exc1 extends Exception {
     void println() {
          System.out.println("Exc1");
     }
}

class Exc2 extends Exception {
     void println() {
```

```
            System.out.println("Exc2");
        }
}

class Exc3 extends Exc1 {
        void println() {
            System.out.println("Exc3");
        }
}

public class Aufgabe4 {
        public static int f(int x) throws Exc1, Exc2, Exc3 {
            if (x == 1)
                throw new Exc1();
            if (x == 2)
                throw new Exc2();
            if (x == 3)
                throw new Exc3();
            return -x;
        }

        public static void main(String[] args) {
            try {
                int i = IO.readInt();
                int y = f(i);
                System.out.println(y);
            } catch (Exc1 exc1) {
                exc1.println();
            } catch (Exc2 exc2) {
                exc2.println();
                return;
            } finally {
                System.out.println("finally");
            }
            System.out.println("Ende");
        }
}
```

Welche Ausgaben erscheinen in welcher Reihenfolge auf dem Bildschirm, wenn ein Nutzer bei der Ausführung des Programms

a) eine 1 eingibt?

b) eine 2 eingibt?

c) eine 3 eingibt?

d) eine 4 eingibt?

Aufgabe 5:

Gegeben sei folgendes Java-Programm:

```
public class Aufgabe5 {

	public static final int NO_ERROR = 0;

	public static final int NULL_ERROR = 1;

	public static final int NEG_ERROR = 2;

	public static int error = NO_ERROR;

	public static int div(int zahl, int durch) {
		if (durch == 0) {
			error = NULL_ERROR;
			return 0;
		}
		return zahl / durch;
	}

	public static int fak(int n) {
		if (n < 0) {
			error = NEG_ERROR;
			return 0;
		}
		if (n == 0)
			return 1;
		else
			return n * fak(n - 1);
	}

	public static void main(String[] args) {
		int wert = IO.readInt();
		error = NO_ERROR;
		int erg = div(15, wert);
		if (error == NULL_ERROR) {
			System.out.println("Fehler: Division durch 0");
		} else {
			error = NO_ERROR;
			erg = fak(erg);
			if (error == NEG_ERROR) {
				System.out.println("Fehler: neg.
Parameterwert");
			} else {
				System.out.println("fak(" + (15 / wert) + ") =
" + erg);
			}
		}
	}
}
```

In diesem Programm wird eine Fehlerbehandlung durchgeführt, wie sie in C-Programmen üblich ist:

- Für jeden Fehlertyp wird eine Konstante definiert.
- Wenn in einer Funktion ein Fehler auftritt, wird einer globalen Fehlervariablen der Wert der entsprechenden Fehlerkonstanten zugewiesen.

- Programme, die eine Funktion aufrufen, müssen nach jedem Funktionsaufruf den Wert der Fehlervariablen überprüfen.

Wandeln Sie das Programm um in ein äquivalentes Java-Programm, das eine Fehlerbehandlung mit Hilfe von Exceptions durchführt. Passen Sie dazu die Funktion fak, div und main entsprechend an.

Aufgabe 6:

Schreiben Sie eine Methode `int readGeradeZahl()`, die mit Hilfe der Methode `readInt()` der Klasse `IO` Zahlen von der Tastatur einliest. Die Methode soll positive gerade Zahlen einlesen. Gibt der Benutzer eine negative oder ungerade Zahl ein, soll eine Exception geworfen werden. Definieren Sie dazu geeignete Exception-Klassen. Schreiben Sie ein Testprogramm, das die Methode aufruft und eine geeignete Fehlerbehandlung durchführt.

Aufgabe 7:

Spielen Sie ein wenig mit folgendem Java-Programm herum. Wann wird was ausgegeben und warum?

```
class DivNullException extends Exception {
}

public class Aufgabe7 {

	public static int div(int zahl, int durch) throws DivNullException {
		if (durch == 0)
			throw new DivNullException();
		return zahl / durch;
	}

	public static void main(String[] args) {
		try {
			while (true) {
				int zahl1 = IO.readInt("Bitte Zahl eingeben: ");
				if (zahl1 == 1)
					break;
				if (zahl1 == 2)
					return;
				int zahl2 = IO.readInt("Bitte weitere Zahl
eingeben: ");
				IO.println(zahl1 + "/" + zahl2 + " = " + div(zahl1,
zahl2));
			}
		} catch (DivNullException exc) {
			IO.println("Division durch 0 ist nicht definiert!");
		} finally {
			IO.println("Programm ist beendet!");
		}
		IO.println("Ende der main-Methode!");
	}
}
```

Aufgabe 8:

Ein Stellenwertsystem ist ein 3-Tupel S = (b, Z, f) mit folgenden Eigenschaften:

- b >= 2 ist eine natürliche Zahl; die *Basis* des Stellenwertsystems.
- Z ist eine b-elementige Menge von Symbolen, den *Ziffern*
- f : Z -> {0, 1, ..., b-1} ist eine Abbildung, die jedem Ziffernsymbol umkehrbar eindeutig eine natürliche Zahl zwischen 0 und b-1 zuordnet.

Eine *Zahl* ist eine endliche Folge von Ziffern.

Der Wert w(z) einer Zahl $z = z_n\ z_{n-1} \ldots z_1\ z_0$ mit z_i ist Element aus Z, n >= 0 bestimmt sich durch w(z) = Summe(i=0 bis n): $f(z_i) * (b^i)$.

Das Hexadezimalsystem ist ein Stellenwertsystem S, das folgendermaßen definiert ist:

```
S = (b, Z, f) mit
  •  b = 16,
  •  Z = {´0´, ´1´, ´2´, ´3´, ´4´, ´5´, ´6´, ´7´, ´8´, ´9, ´A´,
         ´B´, ´C´, ´D´, ´E´, ´F´} und
  •  f : Z -> {0, 1, ..., 15} mit

      f(´0´) = 0; f(´1´) = 1; f(´2´) = 2; f(´3´) = 3; f(´4´) = 4;

      f(´5´) = 5; f(´6´) = 6; f(´7´) = 7; f(´8´) = 8; f(´9´) = 9;

      f(´A´) = 10; f(´B´) = 11; f(´C´) = 12; f(´D´) = 13;

      f(´E´) = 14; f(´F´) = 15;
```

Beispiele: `w(´2´) = 2, w(´10´) = 16, w(´1F´) = 31, w(´1AF´) = 431`

Definieren und implementieren Sie eine Klasse `Hexa`, die den Umgang mit positiven Hexadezimal-Zahlen realisiert. Leiten Sie die Klasse `Hexa` von der abstrakten Klasse `java.lang.Number` ab. Die Klasse soll folgende Methoden zur Verfügung stellen:

- geeignete Konstruktoren (long-Parameter, String-Parameter, Copy-Konstruktor)
- Konvertierungsmethoden, die ein Hexa-Objekt in int-, long-, float-, double- und String-Werte konvertieren
- Vergleichsmethoden, die den Operatoren >, >=, <, <=, ==, != entsprechen
- Methoden, die den arithmetischen Operatoren +, -, *, /, %, +=, -=, *=, /=, %= entsprechen.

Tipps und Anmerkungen:

- Realisieren Sie die Menge Z als char-Array
- Codieren Sie Hexadezimal-Zahlen als Strings
- Rechnen Sie klassenintern mit int-Werten
- Die Konvertierungen funktionieren ähnlich wie bei Binärzahlen
- Überlegen Sie sich, welche Methoden Instanzmethoden und welche Methoden Klassenmethoden sind
- Achten Sie auf mögliche Fehlerfälle, wie Division durch 0 oder die Übergabe negativer Zahlen; definieren Sie geeignete **Exceptions**!

Aufgabe 9:

Implementieren Sie einen Taschenrechner, der auf der Basis der *Umgekehrten Polnischen Notation* (UPN) das Rechnen mit Hexadezimal-Zahlen ermöglicht. Der

Taschenrechner soll die Operationen +, -, *, / und % unterstützen. Nutzen Sie dabei die Klasse `Hexa` aus Aufgabe 8 und die Klasse `java.util.Stack` aus der JDK-Klassenbibliothek.

Anmerkungen und Tipps:

- Beachten Sie fehlerhafte und ungültige Zahl-Eingaben
- Beachten Sie fehlerhafte Operator-Eingaben
- Beachten Sie ungültige Fälle, wie Angabe eines Operators, ohne dass Zahlen auf dem Stack liegen, oder Division durch 0 oder Operationen, durch die negative Zahlen entstehen könnten.

Hinweis:

Bei der UPN werden eingelesene Zahlen jeweils oben auf den Stapel gelegt. Wird ein Operator eingegeben, werden die beiden oberen Elemente vom Stapel entfernt, darauf die Operation angewendet, das Ergebnis ausgegeben und das Ergebnis auf den Stapel gelegt.

Beispiel:

```
Eingabe: 10       (entspricht dezimal 16)
Eingabe: 21       (33)
Eingabe: 1F       (31)
Eingabe: -        (33 - 31 = 2)
Ausgabe: 2        (2)
Eingabe: +        (16 + 2 = 18)
Ausgabe: 12       (18)
```

Aufgabe 10:

Schauen Sie sich die Aufgaben aus UE25 an. Bauen Sie an den Stellen, wo es notwendig ist, Exceptions ein.

Aufgabe 11:

Schauen Sie sich die Implementierung der Klasse IO an.

Aufgabe 12:

Eine **Bitmenge** ist eine Datenstruktur, in der einzelne Bits gesetzt (1), andere nicht gesetzt sind (0). Beispielsweise repräsentiert die Bitfolge 10011000, dass die Bits 1, 4 und 5 gesetzt und alle anderen Bits nicht gesetzt sind.

Implementieren Sie in Java eine Klasse `BitSet`, welche eine Bitmenge als Abstrakten Datentyp realisiert. Die Länge der Bitfolgen soll dabei auf 8 festgesetzt sein, d.h. jedes Objekt der Klasse `BitSet` repräsentiert eine Bitfolge mit 8 Bits! Auf Objekten vom Datentyp `BitSet` sollen dabei folgende Funktionen ausführbar sein:

a) Initialisieren einer leeren Bitmenge, d.h. kein Bit ist gesetzt (Default-Konstruktor)

b) Konstruktor mit String, der Bitmenge repräsentiert

c) Copy-Konstruktor

d) Clonieren einer Bitmenge

e) Konvertieren einer Bitmenge in ein String-Objekt (Bitfolge)

f) Überprüfen auf Wertegleichheit zweier Bitmengen (zwei Bitmengen sind wertegleich, wenn in beiden Mengen exakt dieselben Bits gesetzt sind)

g) Setzen eines einzelnen Bits der Bitmenge. Das zu setzende Bit wird dabei als `int`-Wert übergeben

h) Löschen eines einzelnen Bits der Bitmenge. Das zu löschende Bit wird dabei als `int`-Wert übergeben

i) Ausführung eines logischen `ANDs` (Konjunktion) mit einer anderen Bitmenge

j) Ausführung eines logischen `ORs` (Disjunktion) mit einer anderen Bitmenge

k) Ausführung eines logischen `NOTs` (Negation) auf einer Bitmenge

Wenn Fehler auftreten können, setzen Sie Exceptions ein.

Schreiben Sie weiterhin ein Programm zum Testen.

Aufgabe 13:

Reimplementieren Sie die Klasse IO derart, dass bei fehlerhaften Nutzereingaben (bspw. Buchstaben bei readInt) adäquate Exceptions geworfen werden.

Aufgabe 14:

Implementieren Sie in Java eine Klasse Roman, die den Umgang mit römischen Zahlen (mit Werten zwischen 1 und 3999) ermöglicht. Im (für diese Aufgabe definierten) römischen Zahlensystem steht das Symbol I für 1, V für 5, X für 10, L für 50, C für 100, D für 500 und M für 1000. Symbole ergeben hintereinander geschrieben die römische Zahl. Symbole mit größeren Werten stehen dabei normalerweise vor Symbolen mit niedrigeren Werten. Der Wert einer römischen Zahl wird in diesem Fall berechnet durch die Summe der Werte der einzelnen Symbole. Falls ein Symbol mit niedrigerem Wert vor einem Symbol mit höherem Wert erscheint (es darf übrigens jeweils höchstens **ein** niedrigeres Symbol **einem** höheren Symbol vorangestellt werden), errechnet sich der Wert dieses Teils der römischen Zahl als die Differenz des höheren und des niedrigeren Wertes. Die Symbole I, X, C und M dürfen bis zu dreimal hintereinander stehen; die Symbole V, L und D kommen immer nur einzeln vor. Die Umrechnung von Dezimalzahlen in römische Zahlen ist übrigens nicht eineindeutig. So lässt sich z.B. die Dezimalzahl 1990 römisch darstellen als MCMXC bzw. MXM.

Beispiele:

```
 3999 = MMMCMXCIX
   48 = XLVIII
  764 = DCCLXIV
 1234 = MCCXXXIV
  581 = DLXXXI
```

Implementieren Sie folgende Methoden:

1. Konvertieren eines String-Objektes, das eine römische Zahl repräsentiert, in einen int-Wert.
2. Konvertieren eines int-Wertes in einen String, der eine römische Zahl repräsentiert
3. Initialisieren einer römischen Zahl mit einem String, der eine römische Zahl repräsentiert (Konstruktor)
4. Initialisieren einer römischen Zahl mit einem int-Wert (Konstruktor)
5. Initialisieren einer römischen mit einer bereits existierenden römischen Zahl, d.h. anschließend sind die beiden römischen Zahlen wertegleich (Copy-Konstruktor)
6. Clonieren einer römischen Zahl, d.h. initialisieren einer neuen römischen Zahl mit einer bereits existierenden römischen Zahl, d.h. anschließend sind die beiden römischen Zahlen wertegleich
7. Umwandeln einer römischen Zahl in ein String-Objekt
8. Überprüfen auf Wertegleichheit zweier römischer Zahlen
9. Addition zweier römischer Zahlen

Wenn Fehler auftreten können, setzen Sie Exceptions ein. Schreiben Sie weiterhin ein kleines Testprogramm.

Aufgabe 15:

In einem Programm soll ein abstrakter Datentyp *Menge* verwendet werden. Dieser soll es ermöglichen, mit Mengen im mathematischen Sinne umzugehen. Eine Menge soll dabei eine beliebig große Anzahl von int-Werten aufnehmen können, jeden int-Wert aber maximal einmal.

Schreiben sie eine Klasse *Menge*, welche diesen ADT implementiert.

Auf dem Datentypen Menge sollen folgende Funktionen möglich sein:

a) Erzeugen einer neuen leeren Menge.

b) Clonieren einer Menge, d.h. Erzeugen einer neuen Menge aus einer alten

c) Überprüfen auf Gleichheit zweier Mengen

d) Hinzufügen eines int-Wertes zu einer Menge.

e) Entfernen eines int-Wertes aus einer Menge.

f) Überprüfung, ob ein bestimmter int-Wert in der Menge enthalten ist.

g) Schnittmengenbildung zweier Mengen.

h) Vereinigung zweier Mengen.

i) Differenzbildung zweier Mengen.

Wenn Fehler auftreten können, setzen Sie Exceptions ein. Schreiben Sie weiterhin ein kleines Testprogramm für die Klasse *Menge*.

Aufgabe 16:

In dieser Aufgabe geht es um die Definition einer Klasse `Automat`, die für die Simulation eines einfachen Fahrkartenautomaten genutzt werden könnte. Der Fahrkartenautomat hat dabei drei Funktionsbereiche. Über den ersten Funktionsbereich kann der Nutzer die Anzahl an Kilometern (ganzzahlig) eingeben, die er fahren will. Jeder Kilometer kostet 1 Euro. Über den zweiten Funktionsbereich kann er dann das Geld einwerfen und über den dritten Funktionsbereich ein Ticket drucken lassen, auf dem die Anzahl an gültigen Kilometern vermerkt ist, die mit dem Ticket zurückgelegt werden können.

Die Klasse `Automat` soll folgende Methoden besitzen:

- Einen Konstruktor zum Initialisieren eines neuen Automaten. Pro Automat muss die aktuell eingegebene Kilometeranzahl und der aktuelle Stand der Geldeingabe gespeichert werden.
- Eine Methode `kmAuswahl`, der als Parameter die (ganzzahlige) Anzahl an Kilometern übergeben wird. Die Methode wird aufgerufen, wenn der Nutzer am Fahrkartenautomaten die Kilometeranzahl eingegeben hat.
- Eine Methode `geldEinwerfen`, der als Parameter eine (ganzzahlige) Anzahl an Euros übergeben wird. Die Methode wird aufgerufen, wenn der Nutzer am Fahrkartenautomaten eine gewisse Menge Geld eingeworfen hat.
- Eine Methode `getTicket`, die ein Ticket zurückgibt. Ein Ticket ist dabei ein Objekt einer weiteren Klasse `Ticket`, das die Anzahl an gültigen Kilometern dieses Tickets repräsentiert. Die Methode `getTicket` wirft Exceptions, wenn der Kilometerstand des Automaten ungültig ist (<= 0) oder wenn für den gewählten Kilometerstand noch nicht genügend Geld eingeworfen worden ist.
- Eine Methode `clone` zum Klonieren eines Objektes (Überschreiben der Methode `clone` der Klasse `Object`).
- Eine Methode `equals`, die vergleicht, ob die Zustände zweier Automaten gleich sind (Überschreiben der Methode `equals` der Klasse `Object`).
- Eine Methode `toString`, die eine String-Repräsentation (aktueller Geldstand sowie aktuelle Kilometer-Auswahl) des Automaten zurückliefert (Überschreiben der Methode `toString` der Klasse `Object`).

Aufgabe: Definieren Sie die Klasse `Automat`, eine geeignete Klasse `Ticket` und die zwei Exception-Klassen. Packen Sie die Klassen in ein Paket, so dass sie prinzipiell anderen Programmierern zur Verfügung gestellt werden könnten. Achten Sie auf Zugriffsrechte und das Prinzip der Datenkapselung!

Aufgabe 17:

Gegeben sei folgendes Java-Programm:

```
public class OhneExceptions {

	public static final int NO_ERROR = 0;

	public static final int NEG_ERROR = 1;
```

```
	public static final int OVERFLOW_ERROR = 2;

	public static final int PARAM_ERROR = 3;

	public static int error = NO_ERROR;

	public static int subtrahieren(int zahl1, int zahl2) {
		if (zahl1 < 0 || zahl2 < 0) {
			error = PARAM_ERROR;
			return -1;
		}
		if (zahl1 < zahl2) {
			error = NEG_ERROR;
			return -1;
		}
		error = NO_ERROR;
		return zahl1 - zahl2;
	}

	public static int addieren(int zahl1, int zahl2) {
		if (zahl1 < 0 || zahl2 < 0) {
			error = PARAM_ERROR;
			return -1;
		}
		if (zahl1 + zahl2 < 0) {
			error = OVERFLOW_ERROR;
			return -1;
		}
		error = NO_ERROR;
		return zahl1 + zahl2;
	}

	public static void main(String[] args) {
		for (int i = 0; i < 3; i++) {
			int zahl1 = IO.readInt("Zahl 1: ");
			int zahl2 = IO.readInt("Zahl 2: ");
			int diff = subtrahieren(zahl1, zahl2);
			if (error == PARAM_ERROR) {
				System.out.println("Ungueltiger Parameter!");
			} else if (error == NEG_ERROR) {
				System.out.println("Ungueltige Subtraktion!");
			} else {
				int summe = addieren(zahl1, zahl2);
				if (error == PARAM_ERROR) {
					System.out.println("Ungueltiger Parameter!");
				} else if (error == OVERFLOW_ERROR) {
					System.out.println("Ungueltige Addition!");
				} else {
					System.out.println(zahl1 + "-" + zahl2 + "=" +
					diff);
					System.out.println(zahl1 + "+" + zahl2 + "=" +
					summe);
				}
			}
		}
	}
}
```

In diesem Programm wird eine Fehlerbehandlung durchgeführt, wie sie in C-Programmen üblich ist:

- Für jeden Fehlertyp wird eine Konstante definiert.
- Wenn in einer Funktion ein Fehler auftritt, wird einer globalen Fehlervariablen der Wert der entsprechenden Fehlerkonstanten zugewiesen.
- Programme, die eine Funktion aufrufen, müssen nach jedem Funktionsaufruf den Wert der Fehlervariablen überprüfen.

Wandeln Sie das Programm um in ein äquivalentes Java-Programm, das eine adäquate Fehlerbehandlung mit Hilfe von Exceptions durchführt. Passen Sie dazu die Funktionen subtrahieren, addieren und main entsprechend an. Äquivalent bedeutet, dass sich beide Programme bei gleichen Benutzereingaben exakt gleich verhalten!

Aufgabe 18:

Ein Programmierer, der gerade erst von der Programmiersprache C zur Programmiersprache Java gewechselt hat, hat folgende Klasse CRoman als ADT-Klasse für den Umgang mit römischen Zahlen (mit Werten zwischen 1 und 3999) implementiert. Im römischen Zahlensystem steht das Symbol I für 1, V für 5, X für 10, L für 50, C für 100, D für 500 und M für 1000. Symbole ergeben hintereinander geschrieben die römische Zahl. „XLVIII“ steht bspw. für 48 und „DCCLXIV“ für 764.

```
public class CRoman {

	public final static int MAX_VALUE = 3999;

	// Fehlerkonstanten
	public static final int NO_ERROR = 0;
	public static final int INVALID_STRING_ERROR = 1;
	public static final int INVALID_VALUE_ERROR = 2;

	// globale Fehlervariable
	protected static int error = NO_ERROR;

	// interne Datenstruktur zur Speicherung des Wertes
	private int value;

	// erzeugt eine roemische Zahl mit dem angegebenen Wert
	// Fehler, wenn v <= 0 oder v > MAX_VALUE
	public CRoman(int v) {
		if (v > 0 && v <= MAX_VALUE) {
			this.value = v;
			error = NO_ERROR;
		} else {
			error = INVALID_VALUE_ERROR;
		}
	}

	// erzeugt eine roemische Zahl mit dem durch den String
repraesentierten
	// Wert
	// Fehler, wenn der String keine gueltige roemische Zahl darstellt
	public CRoman(String str) {
		if (check(str)) {
			this.value = convertRomanToInt(str);
			error = NO_ERROR;
```

```
            } else {
                  error = INVALID_STRING_ERROR;
            }
      }

      // addiert die roemische Zahl zur aufgerufenen roemischen Zahl
      public void add(CRoman roman) {
            if (this.value + roman.value <= MAX_VALUE) {
                  this.value += roman.value;
                  error = NO_ERROR;
            } else {
                  error = INVALID_VALUE_ERROR;
            }
      }



   public String toString() {
            return convertIntToRoman(this.value);
      }

      // ueberprueft, ob der String eine gueltige roemische Zahl darstellt
      private static boolean check(String str) {
            // Implementierung hier unwichtig
      }

      // liefert den int-Wert, den der String (gueltige roemische Zahl!)
      // darstellt
      private static int convertRomanToInt(String roman) {
            // Implementierung hier unwichtig
      }

      // wandelt den uebergebenen int-Wert in eine Darstellung als römische
Zahl
      // um
      private static String convertIntToRoman(int value) {
            // Implementierung hier unwichtig
      }

      // Testprogramm
      public static void main(String[] args) {
            int zahl = IO.readInt("int-Zahl: ");
            CRoman roman1 = new CRoman(zahl);
            if (error == INVALID_VALUE_ERROR) {
                  System.out.println("ungueltiger Wert");
            } else {
                  String str = IO.readString("roemische Zahl: ");
                  CRoman roman2 = new CRoman(str);
                  if (error == INVALID_VALUE_ERROR) {
                        System.out.println("ungueltiger String");
                  } else {
                        roman1.add(roman2);
                        if (error == INVALID_VALUE_ERROR) {
                              System.out.println("zu grosser Wert");
                        } else {
                              System.out.println(roman1);
                        }
                  }
            }
      }
```

```
}
```

Der Programmierer kannte noch nicht die Möglichkeit der Fehlerbehandlung durch die Verwendung von Exceptions in Java. In seiner Klasse CRoman hat er daher eine Fehlerbehandlung durchgeführt, wie sie in C-Programmen üblich ist:

- Für jeden Fehlertyp wird eine Konstante definiert.
- Wenn in einer Funktion ein Fehler auftritt, wird einer globalen Fehlervariablen der Wert der entsprechenden Fehlerkonstanten zugewiesen.
- Programme, die eine Funktion aufrufen, müssen nach jedem Funktionsaufruf den Wert der Fehlervariablen überprüfen.

Ein anderer Programmierer bekommt nun den Byte-Code sowie die Dokumentation der Klasse CRoman zur Verfügung gestellt und möchte mit möglichst wenig Aufwand durch Nutzung der Klasse CRoman eine Klasse JavaRoman implementieren, die dieselbe Funktionalität wie die Klasse CRoman bietet, aber zur Fehlerbehandlung das Exception-Konzept von Java einsetzt. Helfen Sie ihm dabei!

Aufgabe:
Leiten Sie von der Klasse CRoman eine Klasse JavaRoman ab, die analoge public-Methoden und -Konstruktoren definiert, wie die Klasse CRoman. Die Fehlerbehandlung soll in der Klasse JavaRoman jedoch auf das Exception-Konzept von Java umgestellt werden. Implementieren Sie die Klasse JavaRoman durch Nutzung der geerbten Methoden und Attribute, d.h. implementieren Sie die eigentliche Funktionalität nicht neu! Die Klasse CRoman darf natürlich nicht verändert werden.

Überführen Sie auch das Testprogramm der Klasse CRoman in ein äquivalentes Testprogramm für die Klasse JavaRoman. Äquivalent bedeutet dabei, dass das Programm bei gleichen Benutzereingaben gleiche Ausgaben erzeugt.

Aufgabe 19:

Ein Programmierer A hat anderen Programmierern folgende Klasse `Util` mit nützlichen Funktionen zur Verfügung gestellt:

```
public class Util {

	// liefert die kleinste Zahl des uebergebenen Arrays
	public static int minimum(int[] werte) {
		int min = werte[0];
		for (int i = 1; i < werte.length; i++) {
			if (werte[i] < min)
				min = werte[i];
		}
		return min;
	}

	// konvertiert den uebergebenen String in einen int-Wert
	public static int toInt(String str) {
		int erg = 0, faktor = 1;
```

```
            char ch = str.charAt(0);
            switch (ch) {
              case '-': faktor = -1; break;
              case '+': faktor = 1; break;
              default:  erg = ch - '0';
            }
            for (int i = 1; i < str.length(); i++) {
                  ch = str.charAt(i);
                  int ziffer = ch - '0';
                  erg = erg * 10 + ziffer;
            }
            return faktor * erg;
      }

      // liefert die Potenz von zahl mit exp, also zahl "hoch" exp
      public static long hoch(long zahl, int exp) {
            if (exp == 0)
                  return 1L;
            return zahl * hoch(zahl, exp - 1);
      }

      // Testprogramm
      public static void main(String[] args) {
            String eingabe = IO.readString("Zahl: ");
            int zahl = toInt(eingabe);
            System.out.println(zahl + " hoch " + zahl + " = " +
                              hoch(zahl, zahl));
      }
}
```

Er vertraut dabei darauf, dass beim Aufruf der Funktionen semantisch gültige Parameterwerte übergeben werden.

Sie finden die Klasse `Util` prinzipiell gut, haben jedoch kein solches Vertrauen in Programmierer, die die Klasse nutzen wollen. Sie möchten sicherstellen, dass andere Programmierer über ungültige Parameterwerte informiert werden.

Aufgaben:

- Überlegen Sie bei den einzelnen Funktionen, welche ungültigen Parameterwerte übergeben werden können.
- Definieren Sie hierfür adäquate Exception-Klassen.
- Schreiben Sie die Klasse `Util` so um, dass in den einzelnen Funktionen die übergebenen Parameterwerte überprüft und gegebenenfalls entsprechende Exceptions geworfen werden. Versehen Sie die Signatur der Funktionen mit den entsprechenden Exceptions. Es muss sichergestellt sein, dass bei der Ausführung der Funktionen keine anderen als die in der Signatur deklarierten Exceptions geworfen werden.
- Passen Sie das Testprogramm entsprechend an. Exceptions sollen abgefangen und adäquate Fehlermeldungen ausgegeben werden.

Aufgabe 20:

Implementieren Sie eine Klasse Cardinal, die das Rechnen mit Natürlichen Zahlen (1, 2, 3, ...) ermöglicht!

Die Klasse Cardinal soll folgende Methoden besitzen:

a) Einen Default-Konstruktor

b) einen Konstruktor, der ein Cardinal-Objekt mit einem übergebenen int-Wert initialisiert

c) einen Copy-Konstruktor, der ein Cardinal-Objekt mit einem bereits existierenden Cardinal-Objekt initialisiert

d) eine Methode clone, die ein Cardinal-Objekt klont

e) eine Methode equals, die zwei Cardinal-Objekte auf Wertegleichheit überprüft

f) eine Methode toString, die eine String-Repräsentation des Cardinal-Objektes liefert

g) eine Klassenmethode sub, die zwei Cardinal-Objekte voneinander subtrahiert

h) eine Methode sub, die vom aufgerufenen Cardinal-Objekt ein übergebenes Cardinal-Objekt subtrahiert

Setzen Sie Exceptions zur Fehlerbehandlung ein!

Aufgabe 21:

Gegeben sei die folgende Klasse `Stack`:

```
public class FullException extends Exception {
}

public class EmptyException extends Exception {
}

public class Stack {

	protected int[] store; // zum Speichern von Daten
	protected int current; // aktueller Index

	public Stack(int size) {
		this.store = new int[size];
		this.current = -1;
	}

	public boolean isFull() {
		return this.current == this.store.length - 1;
	}

	public boolean isEmpty() {
		return this.current == -1;
```

```
    }

    public void push(int value) throws FullException {
        if (!this.isFull()) {
            this.store[++this.current] = value;
        } else {
            throw new FullException();
        }
    }

    public int pop() throws EmptyException {
        if (!this.isEmpty()) {
            return this.store[this.current--];
        } else {
            throw new EmptyException();
        }
    }

}
```

Sie benötigen jedoch einen Stack mit einer einzigen zusätzlichen Methode `undoPop` mit folgender Semantik: Beim Aufruf von `undoPop` soll der Wert, der beim letzten Aufruf der Methode `pop` vom Stack entfernt wurde, wieder oben auf den Stack gelegt werden.

Leiten Sie daher eine Klasse `UndoPopStack` von der Klasse `Stack` ab, die diese Anforderung erfüllt. Überschreiben Sie gegebenenfalls geerbte Methoden. Beachten Sie weiterhin mögliche Fehlerfälle und behandeln Sie diese adäquat.

Beispiel:

Das folgende Programm

```
    public static void main(String[] args) throws Exception {
        UndoPopStack stack = new UndoPopStack(10);
        stack.push(8);
        stack.pop();
        stack.push(4);
        stack.push(7);
        stack.pop();
        stack.push(1);
        stack.undoPop();
        stack.push(2);
        stack.undoPop();
        while (!stack.isEmpty()) {
            System.out.println(stack.pop());
        }
}
```

sollte folgende Ausgabe erzeugen:

```
7
2
7
1
4
```

Aufgabe 22:

In dieser Aufgabe geht es um die Simulation der Buchausleihe in einer Bibliothek. Dabei sind folgende Klassen vorgegeben:

- eine Klasse *Bibliothek*, die eine Bibliothek repräsentiert
- eine Klasse *Nutzer*, die einen Bibliotheksnutzer repräsentiert
- eine Klasse *Buch*, die ein Buchexemplar der Bibliothek repräsentiert
- eine Klasse *AusgeliehenesBuch*, die ausgeliehene Bücher repräsentiert
- eine Exception-Klasse

Die Klassen haben dabei folgende Gestalt:

```
// Fehlerklasse fuer einen unbekannten Nutzer
class UnbekannterNutzerException extends Exception {
}

// Repraesentation von Bibliotheksnutzern
class Nutzer {
      private String name; // Nutzer haben einen Namen
      private int benutzerNummer; // Nutzern ist eine eindeutige Nummer
zugeordnet

      // liefert den Namen des Nutzers
      public String getName() {
            return name;
      }

      // liefert die Benutzernummer des Nutzers
      public int getBenutzerNummer() {
            return benutzerNummer;
      }

      // ... weitere Methoden

}


// Repraesentation von Buechern der Bibliothek
class Buch {
      private String autor; // Name des Autors des Buches
      private String titel; // Titel des Buches
```

```
	// liefert den Autorennamen
	public String getAutor() {
		return autor;
	}

	// liefert den Titel des Buches
	public String getTitel() {
		return titel;
	}

	// ... weitere Methoden
}



// Hilfsklasse zur Repraesentation von ausgeliehenen Buechern
class AusgeliehenesBuch {
	private Buch buch; // das ausgeliehene Buch
	private Nutzer nutzer; // der Nutzer, der das Buch ausgeliehen hat
	private int kosten; // angefallene Ausleihekosten (in vollen EUR)

	// liefert das ausgeliehene Buch
	public Buch getBuch() {
		return buch;
	}

	// liefert den Nutzer, der das Buch ausgeliehen hat
	public Nutzer getNutzer() {
		return nutzer;
	}

	// liefert die aktuell angefallenen Ausleihkosten (in vollen EUR)
	// fuer das Buch
	public int getKosten() {
		return kosten;
	}

	// ... weitere Methoden
}

// Repraesentation der Bibliothek
class Bibliothek {
	public Bibliothek() {
		// durch den Konstruktor werden die verwalteten Daten der
Bibliothek
		// (Nutzer, Buecher, ...) bspw. aus einer Datenbank geladen.
Wie das
		// geschieht und wie sie intern abgespeichert werden, spielt
hier keine
		// Rolle!
	}

	// liefert das dem uebergebenen Namen zugeordnete Nutzer-Objekt
	// wirft eine Exception, wenn zu dem Namen kein angemeldeter Nutzer
	// existiert
	public Nutzer getNutzer(String name) throws
UnbekannterNutzerException {
		// Implementierung hier unwichtig
	}

	// liefert die vom Nutzer mit der angegebenen Benutzernummer aktuell
	// ausgeliehenen Buecher
```

```
      public AusgeliehenesBuch[] getAusgelieheneBuecher(int benutzerNummer)
{
            // Implementierung hier unwichtig
      }

   // ... weitere Methoden
}
```

Aufgabe:

Schreiben Sie auf der Grundlage dieser Klassen mit den vorgegebenen Methoden ein Java-Programm ***Ausleihe***, das folgendes tut:

Beim Aufruf können dem Programm beliebig viele Namen (ohne Leerzeichen!) als Parameter übergeben werden, die Namen von Nutzern der Bibliothek entsprechen sollen; Beispiel: "`java Ausleihe Boles Meier Mustermann`".

Für jeden angegebenen Nutzer soll eine Übersicht seiner aktuell ausgeliehenen Bücher sowie die für ihn aktuell angefallenen Ausleihkosten auf den Bildschirm ausgegeben werden, und zwar in folgender Form:

```
<Nutzername> hat aktuell <Anzahl ausgeliehener Buecher> Buecher ausgeliehen:
<Autor Buch1>: <Titel Buch1>
<Autor Buch2>: <Titel Buch2>
...
<Autor Buchn>: <Titel Buchn>
Angefallene Ausleihekosten: <euro> EUR.
```

Wird ein nicht bekannter Nutzer als Parameter übergeben, soll für diesen ausgegeben werden:

```
Ein Nutzer mit dem Namen <angegebener Name> ist unbekannt.
```

Exemplarisches Beispiel:

Programmaufruf: `java Ausleihe Mueller Meier Schulze`

Bildschirmausgabe:

```
Ein Nutzer mit dem Namen Mueller ist unbekannt.

Meier hat aktuell 2 Buecher ausgeliehen:
Dietrich Boles: Programmieren spielend gelernt
Joachim Goll: Java als erste Programmiersprache
Angefallene Ausleihekosten: 5 EUR.

Schulze hat aktuell 0 Buecher ausgeliehen:
Angefallene Ausleihekosten: 0 EUR.
```

Aufgabe 23:

Stellen Sie sich vor, Sie haben `class`-Dateien der folgenden Klassen `IsEmptyException` und `Stack` zur Verfügung:

```
public class IsEmptyException extends Exception {
}

public class Stack {

	private java.util.ArrayList<Integer> store;

	public Stack() {
		this.store = new java.util.ArrayList<Integer>();
	}

	public final boolean isEmpty() {
		return this.store.isEmpty();
	}

	public final void push(int value) {
		this.store.add(value);
	}

	public final int pop() throws IsEmptyException {
		if (this.isEmpty()) {
			throw new IsEmptyException();
		}
		return this.store.remove(this.store.size() - 1);
	}
}
```

Für eine bestimmte Anwendung benötigen Sie jedoch einen Stack, bei dem zusätzlich die von der Klasse `Object` geerbte Methode `equals` überschrieben ist. Sie soll genau dann `true` liefern, wenn beide verglichene Stacks die gleichen `int`-Werte in der gleichen Reihenfolge speichern.

Aufgabe:

Leiten Sie von der Klasse `Stack` eine Klasse `CStack` ab, die die geerbte `equals`-Methode entsprechend überschreibt.

Hinweise:

- Sie dürfen an der Klasse `Stack` nichts ändern. Achten Sie insbesondere darauf, dass die interne ArrayList als `private` deklariert ist.
- Nach Beendigung der Methode `equals` müssen die Zustände der beiden Stacks identisch sein mit ihren jeweiligen Zuständen vor Aufruf der Methode.

Tipp:

Nutzen Sie Methoden-intern Hilfsobjekte vom Typ `Stack`, in die Sie durch Aufruf der Methoden `push` und `pop` die Werte der beiden Stacks temporär auslagern und während dieses Vorgangs die gespeicherten Werte entsprechend vergleichen.

Beispiel:

Folgendes kleine Testprogramm sollte die Ausgabe `truefalse` produzieren:

```
public class CStackTest {

	public static void main(String[] args) throws Exception {
		CStack stack1 = new CStack();
		stack1.push(2);
		stack1.push(3);
		stack1.push(4);
		CStack stack2 = new CStack();
		stack2.push(2);
		stack2.push(3);
		stack2.push(4);
		System.out.print(stack1.equals(stack2));
		stack2.pop();
		System.out.print(stack1.equals(stack2));
	}
}
```

Hinweis:

Die Klassen `IsEmptyException` und `Stack` und `CStackTest` stehen im StudIP zur Verfügung!

Aufgabe 24:

In dieser Aufgabe geht es um die Simulation des Ausleihens von Videos in einer Videothek. Dazu werden folgende Klassen zur Verfügung gestellt:

- zwei Exception-Klassen
- eine Klasse *Nutzer*, die einen Videotheksnutzer repräsentiert
- eine Klasse *Video*, die ein Videoexemplar der Videothek repräsentiert
- eine Klasse *Videothek*, die die Ausleihe einer Videothek repräsentiert
- eine Klasse *AusgeliehenesVideo*, die ein ausgeliehenes Video repräsentiert.

Die Klassen haben dabei folgendes Protokoll:

```
class UnbekannterNutzerException extends Exception {
}

class KeineAusgeliehenenFilmeException extends Exception {
```

```
	public KeineAusgeliehenenFilmeException(Nutzer nutzer)

	// liefert den Nutzer, der aktuell keine Filme ausgeliehen hat
	public Nutzer getNutzer()
}

// Die Klasse Nutzer repraesentiert einen Videotheksnutzer; Nutzer haben
// jeweils genau einen Namen und eine Benutzungsnummer
class Nutzer {

	// liefert das dem uebergebenen Namen zugeordnete Nutzer-Objekt
	// wirft eine Exception, wenn zu dem Namen kein angemeldeter Nutzer
	// existiert
	public static Nutzer getNutzer(String name)
			throws UnbekannterNutzerException

	// liefert den Namen des Nutzers
	public String getName()

	// liefert die Benutzungsnummer des Nutzers
	public int getBenutzungsnummer()
}

// Die Klasse Video repraesentiert ein Videoexemplar der
// Videothek; Videos haben jeweils genau einen Regisseur und
// einen Titel
class Video {

	// liefert den Regisseurnamen
	public String getRegisseur()


	// liefert den Titel des Videos
	public String getTitel()
}




// Die Klasse AusgeliehenesVideo repraesentiert ein ausgeliehenes
// Video; Objekte der Klasse speichern das ausgeliehene Video,
// den Nutzer, der das Video ausgeliehen hat, und die angefallenen
// Ausleihkosten
class AusgeliehenesVideo {

	// liefert das ausgeliehene Video
	public Video getVideo()

	// liefert den Nutzer, der das Video ausgeliehen hat
	public Nutzer getNutzer()

	// liefert die aktuell angefallenen Ausleihkosten (in vollen EUR)
	// fuer das ausgeliehene Video
	public int getKosten()
}

// Die Klasse Videothek repraesentiert eine Videothek
class Videothek {
	public Videothek() {
		// liest alle benoetigten Daten bspw. aus einer Datenbank ein
	}
```

```
        // liefert alle aktuell ausgeliehenen Videos des angegebenen Nutzers
        public AusgeliehenesVideo[] getAusgelieheneVideos(Nutzer nutzer)
                        throws KeineAusgeliehenenFilmeException
}
```

Aufgabe:

Schreiben Sie ein Java-Programm *Ausleihe*, das durch Nutzung der Klassen folgendes tut:

Beim Aufruf können dem Programm beliebig viele Namen als Parameter übergeben werden, die Namen von Nutzern der Videothek entsprechen sollen; Beispiel: "`java Ausleihe Boles Meier Mustermann`".

Für jeden angegebenen Nutzer soll eine Übersicht seiner aktuell ausgeliehenen Videos sowie die aktuell angefallenen Ausleihkosten auf den Bildschirm ausgegeben werden, und zwar in folgender Form:

```
<Nutzername> hat aktuell <Anzahl ausgeliehener Videos> Videos ausgeliehen:
<Regisseur Video1>: <Titel Video1>
<Regisseur Video2>: <Titel Video2>
...
<Regisseur Videon>: <Titel Videon>
Angefallene Ausleihkosten: <Kosten> EUR.
```

Wird ein nicht bekannter Nutzer als Parameter übergeben, soll ausgegeben werden:

```
Ein Nutzer mit dem Namen <angegebener Name> ist unbekannt.
```

Hat ein Nutzer aktuell keine Videos ausgeliehen, soll ausgegeben werden:

```
Nutzer <angegebener Name> (Benutzungsnummer <Benutzungsnummer des Nutzers>) hat aktuell keine Videos ausgeliehen.
```

Beispiel:

Eine exemplarische Ausgabe für folgenden Aufruf des Programms

```
java Ausleihe Mueller Meier Schulze
```

könnte dann folgende Gestalt haben:

```
Ein Nutzer mit dem Namen Mueller ist unbekannt.

Meier hat aktuell 2 Videos ausgeliehen:
Eric Toledano: Ziemlich beste Freunde
George Lucas: Star Wars Episode 1
Angefallene Ausleihkosten: 8 EUR.
```

Nutzer Schulze (Benutzungsnummer 4711) hat aktuell keine Videos ausgeliehen.