**Programmierkurs Java**

Dr.-Ing. Dietrich Boles

# Aufgaben zu UE 21 – Exceptions

(Stand 14.06.2017)

## 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 {...}
}
```

**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
public 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 EStack {

	private Object[] speicher;

	private int aktIndex;

	public EStack(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 *EStack* 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 *EStack* 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**:

- Leiten Sie eine Klasse *SwapStack* von der Klasse *EStack* ab (die beiden Methoden *push* und *pop* werden geerbt).
- Implementieren Sie (falls notwendig) einen Konstruktor für die Klasse *SwapStack*.
- 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 *EStack* darf nicht verändert werden!
- Achten Sie darauf, dass die beiden Attribute der Klasse *EStack* als `private` und die beiden Methoden der Klasse *EStack* 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 UE21Aufgabe4 {
	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 UE21Aufgabe5 {

    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 UE21Aufgabe7 {

    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:

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:

- Initialisieren einer leeren Bitmenge, d.h. kein Bit ist gesetzt (Default-Konstruktor)
- Konstruktor mit String, der Bitmenge repräsentiert
- Copy-Konstruktor
- Clonieren einer Bitmenge
- Konvertieren einer Bitmenge in ein String-Objekt (Bitfolge)
- Überprüfen auf Wertegleichheit zweier Bitmengen (zwei Bitmengen sind wertegleich, wenn in beiden Mengen exakt dieselben Bits gesetzt sind)
- Setzen eines einzelnen Bits der Bitmenge. Das zu setzende Bit wird dabei als int-Wert übergeben
- Löschen eines einzelnen Bits der Bitmenge. Das zu löschende Bit wird dabei als int-Wert übergeben
- Ausführung eines logischen ANDs (Konjunktion) mit einer anderen Bitmenge
- Ausführung eines logischen ORs (Disjunktion) mit einer anderen Bitmenge
- Ausführung eines logischen NOTs (Negation) auf einer Bitmenge

Wenn Fehler auftreten können, setzen Sie Exceptions ein.

Schreiben Sie weiterhin ein Programm zum Testen.

## Aufgabe 11:

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

## Aufgabe 12:

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:

- Konvertieren eines String-Objektes, das eine römische Zahl repräsentiert, in einen int-Wert.
- Konvertieren eines int-Wertes in einen String, der eine römische Zahl repräsentiert
- Initialisieren einer römischen Zahl mit einem String, der eine römische Zahl repräsentiert (Konstruktor)
- Initialisieren einer römischen Zahl mit einem int-Wert (Konstruktor)
- Initialisieren einer römischen mit einer bereits existierenden römischen Zahl, d.h. anschließend sind die beiden römischen Zahlen wertegleich (Copy-Konstruktor)
- 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
- Umwandeln einer römischen Zahl in ein String-Objekt
- Überprüfen auf Wertegleichheit zweier römischer Zahlen
- Addition zweier römischer Zahlen

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

## Aufgabe 13:

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:

- Erzeugen einer neuen leeren Menge.
- Clonieren einer Menge, d.h. Erzeugen einer neuen Menge aus einer alten
- Überprüfen auf Gleichheit zweier Mengen
- Hinzufügen eines int-Wertes zu einer Menge.
- Entfernen eines int-Wertes aus einer Menge.
- Überprüfung, ob ein bestimmter int-Wert in der Menge enthalten ist.
- Schnittmengenbildung zweier Mengen.

- Vereinigung zweier Mengen.
- Differenzbildung zweier Mengen.

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

## Aufgabe 14:

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 15:

Gegeben sei folgendes Java-Programm:

```
public class UE21Aufgabe15 {

	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 16:

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.

```
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 = CRoman.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 <= CRoman.MAX_VALUE) {
            this.value = v;
            CRoman.error = CRoman.NO_ERROR;
        } else {
            CRoman.error = CRoman.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 (CRoman.check(str)) {
			this.value = CRoman.convertRomanToInt(str);
			CRoman.error = CRoman.NO_ERROR;
		} else {
			CRoman.error = CRoman.INVALID_STRING_ERROR;
		}
	}

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

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

	// interne Datenstruktur fuer Umrechnungszwecke
	static String symbol = "IVXLCDM";
	static int[] wert = { 1, 5, 10, 50, 100, 500, 1000 };

	// ueberprueft, ob der String eine gueltige roemische Zahl darstellt
	private static boolean check(String str) {
		return true; // todo (muss hier gueltig sein)
	}

	// liefert den int-Wert, den der String (gueltige roemische Zahl!)
	// darstellt
	private static int convertRomanToInt(String roman) {
		int value = 0;
		// String der Reihe nach durcharbeiten
		for (int i = 0; i < roman.length(); i++) {
			// Index bestimmen
			int index1 = CRoman.symbol.indexOf(roman.charAt(i));
			if (i + 1 < roman.length()) { // es folgen weitere
Symbole
				int index2 = CRoman.symbol.indexOf(roman.charAt(i +
1));
				if (index1 < index2) { // Differenz bilden und
addieren
					value = value + CRoman.wert[index2] -
CRoman.wert[index1];
					i++;
				} else { // addieren
					value = value + CRoman.wert[index1];
				}
			} else { // letztes Symbol
				value = value + CRoman.wert[index1];
			}
		}
		return value;
	}

	// interne Datenstruktur fuer Umrechnungszwecke
	static String symbols[][] = {
			// Symbole fuer 1-9
```

```
                { "", "I", "II", "III", "IV", "V", "VI", "VII", "VIII",
"IX" },
                // Symbole fuer 10-90
                { "", "X", "XX", "XXX", "XL", "L", "LX", "LXX", "LXXX",
"XC" },
                // Symbole fuer 100-900
                { "", "C", "CC", "CCC", "CD", "D", "DC", "DCC", "DCCC",
"CM" },
                // Symbol fuer 1000
                { "", "M", "MM", "MMM" } };

    // wandelt den uebergebenen int-Wert in eine Darstellung als römische
    // Zahl um
    private static String convertIntToRoman(int value) {
        String roman = new String("");
        int potenz = 0;
        while (value > 0) {
            // bestimme letzte Ziffer
            int ziffer = value % 10;
            // erweitere die Roemische Zahl entsprechend der Ziffer
            // und der aktuellen Stelligkeit
            roman = CRoman.symbols[potenz][ziffer] + roman;
            // erhoehe Stelligkeit
            potenz++;
            // kuerze value um eine Stelle
            value = value / 10;
        }
        return roman;
    }
}

public class UE21Aufgabe16 {

    // Testprogramm
    public static void main(String[] args) {
        int zahl = IO.readInt("int-Zahl: ");
        CRoman roman1 = new CRoman(zahl);
        if (CRoman.error == CRoman.INVALID_VALUE_ERROR) {
            System.out.println("ungueltiger Wert");
        } else {
            String str = IO.readString("roemische Zahl: ");
            CRoman roman2 = new CRoman(str);
            if (CRoman.error == CRoman.INVALID_STRING_ERROR) {
                System.out.println("ungueltiger String");
            } else {
                roman1.add(roman2);
                if (CRoman.error == CRoman.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 17:

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

```
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 * Util.hoch(zahl, exp - 1);
      }
}

public class UE21Aufgabe17 {

      // Testprogramm
      public static void main(String[] args) {
            String eingabe = IO.readString("Zahl: ");
            int zahl = Util.toInt(eingabe);
            System.out.println(zahl + " hoch " + zahl + " = "
                        + Util.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 18:

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

Die Klasse *Cardinal* soll folgende Methoden besitzen:

- Einen Default-Konstruktor

- einen Konstruktor, der ein Cardinal-Objekt mit einem übergebenen int-Wert initialisiert
- einen Copy-Konstruktor, der ein Cardinal-Objekt mit einem bereits existierenden Cardinal-Objekt initialisiert
- eine Methode clone, die ein Cardinal-Objekt klont
- eine Methode equals, die zwei Cardinal-Objekte auf Wertegleichheit überprüft
- eine Methode toString, die eine String-Repräsentation des Cardinal-Objektes liefert
- eine Klassenmethode sub, die zwei Cardinal-Objekte voneinander subtrahiert
- eine Methode sub, die vom aufgerufenen Cardinal-Objekt ein übergebenes Cardinal-Objekt subtrahiert

Überlegen Sie, wo Fehler auftreten können und setzen Sie Exceptions zur Fehlerbehandlung ein!

## Aufgabe 19:

Gegeben sei die folgende Klasse `Stack`:

```
class FullException extends Exception {
}

class EmptyException extends Exception {
}

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 20:

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

```
public class StackIsEmptyException extends Exception {
}

public class AStack {

    private java.util.ArrayList<Integer> store;

    public AStack() {
        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 StackIsEmptyException {
		if (this.isEmpty()) {
			throw new StackIsEmptyException();
		}
		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 `AStack` eine Klasse `CStack` ab, die die geerbte `equals`-Methode entsprechend überschreibt.

**Hinweise**:

- Sie dürfen an der Klasse `AStack` 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 `AStack`, 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 UE21Aufgabe20 {

	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));
	}
}
```

## Aufgabe 21:

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:

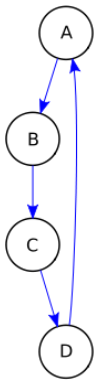
```
Ein Nutzer mit dem Namen Mueller ist unbekannt.

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.
```

## Aufgabe 22:

Ein *Graph* ist in der Graphentheorie eine abstrakte Struktur, die eine Menge von Objekten zusammen mit den zwischen diesen Objekten bestehenden Verbindungen repräsentiert. Die mathematischen Abstraktionen der Objekte werden dabei *Knoten* (auch Ecken) des Graphen genannt. Die paarweisen Verbindungen zwischen Knoten heißen *Kanten* (manchmal auch Bögen). Die Kanten können gerichtet oder ungerichtet sein. Häufig werden Graphen anschaulich gezeichnet, indem die Knoten durch Punkte und die Kanten durch Linien dargestellt werden (aus Wikipedia).

Anschauliche Beispiele für Graphen sind ein Stammbaum oder das U-Bahn-Netz einer Stadt. Bei einem Stammbaum stellt jeder Knoten ein Familienmitglied dar und jede Kante ist eine Verbindung zwischen einem Elternteil und einem Kind, sodass es sich insbesondere um einen Baum handelt. In einem U-Bahn-Netz stellt jeder Knoten eine U-Bahn-Station dar und jede Kante eine direkte Zugverbindung zwischen zwei Stationen.

In *gerichteten Graphen* werden Kanten statt durch Linien durch Pfeile gekennzeichnet, wobei der Pfeil vom ersten (Ausgangsknoten) zum zweiten Knoten (Endknoten) zeigt. Dies verdeutlicht, dass jede Kante des Graphen nur in eine Richtung durchlaufen werden kann.




**Aufgabe:**

Implementieren Sie eine ADT-Klasse `Graph`, die gerichtete Graphen repräsentiert. Knoten sollen durch Namen in Form von Strings repräsentiert werden. Die Klasse soll folgende Methoden bereitstellen:

- Einen Default-Konstruktor.

- Einen Konstruktur, dem über einen VarArgs-Parameter beliebig viele Knoten übergeben werden können.
- Eine Methode zum Hinzufügen eines weiteren Knotens zum Graphen.
- Eine Methode zum Hinzufügen einer gerichteten Kante zum Graphen. Der Ausgangsknoten der Kante wird dabei als erster, der Endknoten der Kante als zweiter Parameter übergeben.
- Eine Methode, die überprüft (und das Ergebnis der Überprüfung als Wert liefert), ob es eine direkte Verbindung (also eine Kante) zwischen zwei als Parameter übergebenen Knoten gibt. Der Ausgangsknoten der Kante wird dabei als erster, der Endknoten der Kante als zweiter Parameter übergeben.
- Eine Methode, die überprüft (und das Ergebnis der Überprüfung als Wert liefert), ob es eine (nicht-unbedingt direkte) Verbindung zwischen zwei als Parameter übergebenen Knoten gibt. Der Ausgangsknoten der Kante wird dabei als erster, der Endknoten der Kante als zweiter Parameter übergeben. Eine Verbindung existiert dabei genau dann, wenn der angegebene Endknoten entlang einer Menge an Kanten ausgehend vom Ausgangsknoten erreicht werden kann. In der obigen Abbildung gibt es also bspw. eine Verbindung zwischen Knoten „A“ und „D“, und zwar via „B“ und „C“.

Überlegen Sie sich eine adäquate Datenstruktur zur Speicherung von Knoten und Kanten. Überlegen Sie, wo Fehler auftreten können und setzen Sie zur Fehlerbehandlung adäquate Exceptions ein.

Testen Sie die Ihre Klasse `Graph` bspw. mit folgendem Programm. Es sollte die Ausgaben „true“ und „false“ produzieren:

```
public static void main(String[] args) throws Exception {
	Graph graph = new Graph("1", "2", "3", "4", "5");
	graph.addEdge("1", "2");
	graph.addEdge("2", "3");
	graph.addEdge("3", "2");
	graph.addEdge("3", "4");
	graph.addEdge("1", "5");
	System.out.println(graph.existsConnection("1", "4"));
	System.out.println(graph.existsConnection("2", "5"));
}
```

## Aufgabe 23:

Gegeben sei die folgende Klasse `Stack`:

```
class FullException extends Exception {
}

class EmptyException extends Exception {
}

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, bei dem sichergestellt ist, dass ein `int`-Wert nicht mehrfach im Stack vorkommt. Leiten Sie daher eine Klasse `UniqueStack` von der Klasse `Stack` ab, in der die Methode `push` derart überschrieben wird, dass die Anforderung erfüllt ist.

Definieren und implementieren Sie in der Klasse `UniqueStack` weiterhin einen Copy-Konstruktor und überschreiben Sie die von der Klasse `Object` geerbte Methode `equals`. Der Copy-Konstruktor soll ein `UniqueStack`-Objekt mit einem neuen Stack erzeugen, dessen Inhalt wertegleich dem Stack-Inhalt des übergebenen `UniqueStack`-Objektes ist. Die `equals`-Methode soll genau dann `true` liefern, wenn die einzelnen Elemente, die sich gerade im Stack des aufgerufenen Objektes befinden, wertegleich sind zu den Elementen im Stack des als Parameter übergebenen `UniqueStack`-Objektes.

## Aufgabe 24:

Im Paket `java.util` stellt das JDK mit der Klasse `Properties` eine Möglichkeit zur Verwaltung von Properties zur Verfügung. Ein Property ist dabei ein String-Paar (Name, Wert), bei dem einem bestimmten Namen ein Wert zugeordnet werden kann. Ihre Aufgabe besteht darin, einen Teil dieser Klasse nachzuimplementieren:

```
/**
 * verwaltet eine Liste mit Properties
 */
public class PropertyList {

      /**
```

```
     * Konstruktor mit einem Properties-Objekt als Parameter für das
gilt: Wird
     * beim Aufruf der Methode getProperty kein Property mit dem
angegebenen
     * Namen gefunden, wird im Properties-Objekt parent weitergesucht
(ggfls.
     * rekursive Fortsetzung dieses Prozesses bei dessen "parent")
     *
     * @param defaults
     *            "Parent"-PropertyListe
     */
    public PropertyList(PropertyList parent)

    /**
     * Default-Konstruktor: erzeugt eine leere Property-Liste ohne
     * "Parent"-PropertyListe
     */
    public PropertyList()

    /**
     * existiert bereits ein Property mit dem key, wird ihm der neue Wert
value
     *  zugeordnet; andernfalls wird ein neues Property (key/value)
erzeugt und
     * gespeichert
     *
     * @param key
     *            Name des Properties
     * @param value
     *            Wert des Properties
     */
    public void setProperty(String key, String value)

    /**
     * liefert den dem Property mit dem Namen key zugeordneten Wert;
existiert
     * kein entsprechendes Property, wird eine Exception geworfen
     *
     * @param key
     *            Name des Property
     * @return der dem Namen zugeordnete Wert
     * @throws PropertyNotDefinedException
     *             wird geworfen, falls kein Property mit dem Namen key
     *             existiert
     */
    public       String       getProperty(String       key)       throws
PropertyNotDefinedException
```

**Aufgabe**:

Implementieren Sie die obige Klasse `PropertyList` sowie die Klasse `PropertyNotDefinedException`. Wählen Sie geeignete Attribute, implementieren Sie die Methoden, definieren Sie ggfls. weitere Hilfsklassen. Für die Implementierung dürfen Sie weder die Klasse `java.util.Properties` noch eine Hashtabelle des JDK nutzen. Speichern Sie stattdessen die einzelnen Properties in einer ArrayList ab (`java.util.ArrayList`).

**Beispiel**:

Das folgende Programm, das die Klasse `PropertyList` nutzt, sollte die Ausgabe

```
eins
four
not defined
```

erzeugen:

```
public static void main(String[] args) {
            PropertyList standard = new PropertyList();
            standard.setProperty("1", "one");
            standard.setProperty("2", "two");
            standard.setProperty("3", "three");
            standard.setProperty("4", "four");
            PropertyList p = new PropertyList(standard);
            p.setProperty("1", "eins");
            p.setProperty("2", "zwei");
            p.setProperty("3", "drei");
            try {
                  System.out.println(p.getProperty("1"));
                  System.out.println(p.getProperty("4"));
                  System.out.println(p.getProperty("5"));
            } catch (PropertyNotDefinedException exc) {
                  System.out.println("not defined");
            }

      }

}
```

## Aufgabe 25:

Eine „Strich-Zahl“ ist ein Konstrukt, bei dem ein Zahlenwert durch eine entsprechende Anzahl an Strichen („|“) dargestellt wird. Beispiele:

**3 = |||**

**6 = ||||||**

**8 = ||||||||**

Implementieren Sie eine ADT-Klasse *StrichZahl*, die Strich-Zahlen zwischen 0 und 999 repräsentiert. Genauso wie in Java bei int-Werten führt ein Underflow bzw. Overflow dazu, dass wieder am anderen Ende der Skala begonnen wird (d.h. bspw. 999 + 1 = 0 bzw. 0 – 1 = 999).

Die Klasse *StrichZahl* soll folgende Methoden besitzen:

1. Einen Default-Konstruktor, der ein StrichZahl-Objekt mit dem Wert 0 initialisiert
2. einen Konstruktor, der ein StrichZahl-Objekt mit einem als Parameter übergebenen String mit „|“-Zeichen initialisiert
3. einen Konstruktor, der ein StrichZahl-Objekt mit einem als Parameter übergebenen int-Wert initialisiert, der den Wert der entsprechenden Strich-Zahl repräsentiert.
4. einen Copy-Konstruktor, der ein StrichZahl-Objekt mit einem als Parameter übergebenen bereits existierenden StrichZahl-Objekt initialisiert

5. eine Methode *clone*, die das aufgerufene StrichZahl-Objekt klont (überschreiben der entsprechenden Methode, die von der Klasse *Object* geerbt wird)
6. eine Methode *equals*, die überprüft, ob zwei StrichZahl-Objekte den gleichen Wert repräsentieren (überschreiben der entsprechenden Methode, die von der Klasse *Object* geerbt wird)
7. eine Methode *toString*, die die Strich-Zahl in Form eines Strings liefert (Überschreiben der Methode *toString*, die von der Klasse *Object* geerbt wird). Für eine bessere Übersicht sollen nach jeweils 5 Strichen immer ein Leerzeichen folgen (außer am Ende); bspw. „||||| |||“ für 8 oder „||||| ||||| |||||“ für 10.
8. eine Methode *add*, die zum aufgerufenen StrichZahl-Objekt ein anderes als Parameter übergebenes StrichZahl-Objekt addiert
9. eine Klassen-Methode *add*, die zwei als Parameter übergebene StrichZahl-Objekte addiert und deren Summe in Form eines StrichZahl-Objektes als Wert liefert.

Hinweise:

- Außer aus dem Paket java.lang dürfen Sie keine Klassen aus dem JDK verwenden!
- Überlegen Sie, bei welchen Methoden Fehler auftreten können, und behandeln Sie diese durch adäquate Exceptions.
- Tipp: Nutzen Sie als interne Datenstruktur zur Speicherung des Wertes der StrichZahl ein int-Attribut!

**Beispiel**:

Das folgende Programm

```
StrichZahl z1 = new StrichZahl(7);
StrichZahl z2 = new StrichZahl("||||||||");
System.out.println(z1.toString());
System.out.println(z2.toString());
z1.add(z2);
System.out.println(z1.toString());
System.out.println(StrichZahl.add(z1, z2).toString());
```

sollte auf der Konsole die Ausgabe

```
|||||
|||||
|||||
|||||
```

erzeugen.

## Aufgabe 26:

Gegeben sei folgendes Java-Programm:

```
class NullException extends Exception {}

class LeeresArrayException extends Exception {}

class KeineZahlException extends Exception {}

class NegativerExponentException extends Exception {}

public class Aufgabe3MitExceptions {

	// liefert die kleinste Zahl des uebergebenen Arrays
	static int minimum(int[] werte) throws NullException,
LeeresArrayException {
		if (werte == null) {
			throw new NullException();
		}
		if (werte.length == 0) {
			throw new LeeresArrayException();
		}
		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
	static int toInt(String str) throws NullException, KeineZahlException
{
		if (str == null) {
			throw new NullException();
		}
		if (str.length() == 0) {
			throw new KeineZahlException();
		}
		int erg = 0, faktor = 1;
		for (int i = 1; i < str.length(); i++) {
			char ch = str.charAt(i);
			int ziffer = ch - '0';
			if (ziffer < 0 || ziffer > 9) {
				throw new KeineZahlException();
			}
			erg = erg * 10 + ziffer;
		}
		return faktor * erg;
	}

	// liefert die Potenz von zahl mit exp, also zahl "hoch" exp
	static int hoch(int zahl, int exp) throws NegativerExponentException
{
		if (exp < 0)
			throw new NegativerExponentException();
		if (exp == 0) {
			return 1;
		} else {
```

```
                return zahl * hoch(zahl, exp - 1);
        }
    }


    // Testprogramm
    public static void main(String[] args) {
        try {
            int[] zahlen = new int[args.length];
            for (int i = 0; i < args.length; i++) {
                zahlen[i] = toInt(args[i]);
            }
            System.out.println(hoch(2, minimum(zahlen)));
        } catch (NullException exc) {
            System.out.println("Keine Argumente");
        } catch (KeineZahlException exc) {
            System.out.println("Ungueltige Argumente");
        } catch (LeeresArrayException e) {
            System.out.println("Leeres Array");
        } catch (NegativerExponentException e) {
            System.out.println("Negativer Exponent");
        }
    }

}
```

In diesem Programm wird eine Fehlerbehandlung mit Exceptions durchgeführt. Wandeln Sie das Programm um in ein äquivalentes Java-Programm, das eine Fehlerbehandlung ohne Einsatz bzw. Nutzung von Exception-Klassen und Exceptions durchführt, sondern stattdessen eine Fehlerbehandlung integriert, wie sie in C-Programmen üblich ist (wurde in der UE demonstriert):

- 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 und ggfls. eine Fehlerbehandlung einleiten.

Passen Sie die Funktionen `main`, `toInt`, `hoch` und `minimum` entsprechend an. Äquivalent bedeutet, dass beide Programme bei gleichem Programmaufruf exakt gleiche Ausgaben produzieren!

## Aufgabe 27:

### Teilaufgabe (a):

Definieren Sie eine ADT-Klasse *Telefon* zur Repräsentation eines Telefons. Mit einem Telefon sollen Nummern (ein String) gewählt werden. Ein Telefon speichert jeweils die zuletzt gewählte Nummer. Ein Telefon besitzt zudem eine Anzahl *n* an Speichertasten. Der Zugriff auf die Tasten erfolgt über Nummern von 0 bis *n*-1. Insgesamt soll Ihre Klasse die folgenden Methoden definieren:

(1) Einen Konstruktor, dem die Anzahl an Speichertasten des Telefons übergeben wird.
(2) Einen Copy-Konstruktor zum Werte-gleichen Klonen eines bereits existierenden Telefons.

(3) Eine Methode zum Wählen einer Nummer. Die zu wählende Nummer wird als Parameter übergeben (String). Die Methode gibt hier die Nummer einfach auf die Konsole aus.
(4) Eine Methode, die die zuletzt gewählte Nummer liefert.
(5) Eine Methode, um die zuletzt gewählte Nummer auf einer Speichertaste zu speichern. Die Nummer der Speichertaste wird als Parameter übergeben.
(6) Eine weitere Methode zum Wählen. Dieser wird die Nummer einer Speichertaste als Parameter übergeben. Die in der entsprechenden Speichertaste gespeicherte Nummer soll gewählt werden.

**Teilaufgabe (b):**

Überlegen Sie, bei welchen Methoden Fehler auftreten können, und behandeln Sie diese durch den Einsatz adäquater Exceptions.

## Aufgabe 28:

Stellen Sie sich Dateien vor, in denen zeilenweise Zahlen (int-Werte) stehen. Ein Programmierer hat die Aufgabe bekommen, eine Methode *summe* zu implementieren, die die einzelnen Zeilenwerte addiert und die errechnete Summe in eine andere Datei schreibt. Er implementiert dazu folgende Methode:

```
static void summe(String vonDateiName, String summeDateiName)
    throws Exception {
  // Datei oeffnen
  BufferedReader quelle =
    new BufferedReader(new FileReader(vonDateiName));

  // Zahlen zeilenweise auslesen und Summe berechnen
  int summe = 0;
  String zeile = quelle.readLine();
  while (zeile != null) {
    summe += new Integer(zeile);
    zeile = quelle.readLine();
  }

  // Datei schliessen
  quelle.close();

  // Datei oeffnen
  PrintWriter senke = new PrintWriter(new FileWriter(summeDateiName));

  // Summe in Datei schreiben
  senke.println(summe);
  senke.flush();

  // Datei schliessen
  senke.close();

  // Kontrollausgabe
  System.out.println("Datei wurde erzeugt und speichert Summe = " +
                     summe);
}
```

Leider kümmert er sich nicht um mögliche Exceptions, sondern leitet sie einfach weiter. Folgende Methoden können dabei Exceptions werfen:

- Konstruktor FileReader(String dateiName) throws java.io.IOException (wenn bspw. die Datei nicht existiert oder nicht lesbar ist)
- Konstruktor Integer(String wert) throws java.lang.NumberFormatException (wenn der übergebene String keine gültige Zahl repräsentiert)
- Konstruktor FileWriter(String dateiName) throws java.io.IOException (wenn bspw. die Datei nicht beschreibbar ist)

Passen Sie die Methode *summe* so, an, dass die Exceptions nicht weitergeleitet sondern folgendermaßen behandelt werden:

- Liefert der Konstruktor von FileReader eine Exception, wird das Lesen der Datei vonDateiName abgebrochen und eine 0 in die Datei summeDateiName geschrieben.
- Liefert der Konstruktor von Integer eine Exception, wird die entsprechende Zeile ignoriert.
- Liefert der Konstruktor von FileWriter eine Exception, wird diese zwar abgefangen, aber nicht weiter behandelt; die Methode *summe* wird jedoch ohne die Kontrollausgabe direkt verlassen.

Weiterhin ist programmiertechnisch zu beachten, dass einmal geöffnete Dateien innerhalb der Methode auf jeden Fall wieder geschlossen werden müssen (Aufruf der zwei Methoden *close*), was der andere Programmierer auch nicht beachtet hat.

## Aufgabe 29:

In dieser Aufgabe geht es um die objektorientierte Modellierung eines Geldspielautomaten. In einen solchen Automaten soll man Geld einwerfen, Spiele mit entsprechend festgelegten Einsätzen spielen und sich bestehende Guthaben auszahlen lassen können.

Definieren Sie eine Klasse **`GeldspielAutomat`**. Ein Geldspielautomat speichert ein Guthaben und einen jeweils festzulegenden Spieleinsatz für das folgende Spiel (jeweils int-Attribute für volle EUR-Beträge). Definieren und implementieren Sie dann folgende Methoden:

- Einen Default-Konstruktor: Anfangs sind Guthaben und Einsatz gleich 0.
- Eine Methode **`geldEinwerfen`**, die als Parameter einen int-Wert für den entsprechend einzuwerfenden EUR-Betrag besitzt. Das Guthaben des Automaten soll entsprechend dem Parameterwert erhöht werden.
- Eine Getter-Methode für das Guthaben des Automaten.
- Eine Methode **`auszahlen`**, die das aktuelle Guthaben des Automaten als Rückgabewert liefert und intern auf 0 setzt.
- Eine Methode **`einsatzFestlegen`**, die als Parameter einen int-Wert für den entsprechend festzulegenden Einsatz in EUR besitzt. Der im Automaten gespeicherte Einsatz soll entsprechend festgesetzt werden.
- Eine Methode **`spielen`**. In der Methode sollen zwei Zufallszahlen zwischen 0 und 5 generiert werden. Sind die Zahlen gleich, hat der Spieler gewonnen und das 3-fache des aktuell festgelegten Einsatzes wird dem Guthaben hinzugefügt. Sind die Zahlen ungleich, hat der Spieler verloren und der aktuell

festgelegte Einsatz wird vom Guthaben abgezogen. Der Einsatz wird in beiden Fällen wieder auf 0 gesetzt. Im Gewinnfall liefert die Methode *true*, ansonsten *false*.

Überlegen Sie, was fehlerhafte Parameterwerte bzw. Zustände sein können und behandeln Sie potentiell fehlerhafte Parameterwerte bzw. Situationen adäquat.

Ein Testprogramm könnte folgendermaßen aussehen:

```
GeldspielAutomat automat = new GeldspielAutomat();
// ...
automat.geldEinwerfen(IO.readInt("Einwerfen: "));
// ...
automat.einsatzFestlegen(IO.readInt("Einsatz: "));
if (automat.spielen()) {
      System.out.println("gewonnen");
} else {
      System.out.println("verloren");
}
IO.println("Guthaben = " + automat.getGuthaben());
// ...
int geld = automat.auszahlen();
// ...
```

## Aufgabe 30:

In dieser Aufgabe geht es um die Ausgabe von Fahrplänen eines einfachen Busnetzes. Gegeben sind folgende Klassen:

```
public class KeineVerbindungException extends Exception {}

public class UnbekannteHaltestelleException extends Exception {}

public class Haltestelle {

     private String name;

     // liefert zu einem Namen das entsprechende Haltestellen-Objekt;
     // die Exception wird geworfen, wenn zu dem Namen keine Haltestelle
     // existiert
     public static Haltestelle getHaltestelle(String name)
     throws UnbekannteHaltestelleException {
           // ... Implementierung hier unwichtig
     }

     // liefert den Namen einer Haltestelle
     public String getName() { return name;    }

     // liefert die Fahrzeit in Minuten zwischen der aufgerufenen
     // Haltestelle und der als Parameter uebergebenen Haltestelle;
     public int getFahrzeit(Haltestelle nachbarStelle) {
           // ... Implementierung hier unwichtig
   }

}

public class Busnetz {

     // initialisiert ein Busnetz
     public Busnetz() {
     // ... Implementierung hier unwichtig
```

```
    }

    // liefert Informationen zur Verbindung der Haltestellen "von" nach
    // "nach";
    // in den Elementen des gelieferten Arrays stehen in der
    // entsprechenden Reihenfolge die nacheinander angefahrenen
    // Haltestellen
    // die Exception wird geworfen, wenn keine Verbindung existiert
    public Haltestelle[] getVerbindung(Haltestelle von, Haltestelle nach)
     throws KeineVerbindungException {
          // ... Implementierung hier unwichtig
    }
}
```

Schreiben Sie mit Hilfe der Methoden der gegebenen Klassen 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 Hauptbahnhof Schule Krankenhaus Uni**"

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 *Hauptbahnhof* nach *Schule* und anschließend für die Strecke von *Krankenhaus* nach *Uni*. Nach der Ermittlung einer Verbindung sollen daraus Fahrpläne in folgender Form auf den Bildschirm ausgegeben werden:

**Strecke <name1> - <name2>:**
**Start bei <name1>**
**<haltestelle1> <n1> Minuten**
**<haltestelle2> <n2> Minuten**
**...**
**<name2> <nn> Minuten**

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

Existiert zu einem dem Java-Programm übergebenen Namen keine Haltestelle, soll ausgegeben werden:

**Haltestelle <name> ist unbekannt**

und das entsprechende Parameterpaar soll nicht weiter betrachtet werden.

Existiert zu einem dem Java-Programm übergebenen Parameterpaar keine Verbindung, soll ausgeben werden:

**Keine Verbindung zwischen <name1> und <name2>**

und das entsprechende Parameterpaar soll nicht weiter betrachtet werden.

Beispiel: Eine exemplarische Ausgabe beim Aufruf von

"**java Fahrplaene Hauptbahnhof Schule Krankenhaus Uni Badesee Aldi**":

```
Strecke Hauptbahnhof - Schule:
Start bei Haltestelle Hauptbahnhof
Freibad 7 Minuten
Turnhalle 16 Minuten
```

```
Friedhof 24 Minuten
Schule 30 Minuten

Haltestelle Krankenhaus ist unbekannt

Keine Verbindung zwischen Badesee und Aldi
```

## Aufgabe 31:

Welche Ausgabe wird beim Aufruf des folgenden Programms auf die Konsole ausgegeben:

(a) Bei einer Benutzereingabe von: **-1**

(b) Bei einer Benutzereingabe von: **0**

(c) Bei einer Benutzereingabe von: **1**

```
class Exc1 extends Exception {
  public String toString() { return "X"; }
}

class Exc2 extends Exception {
  public String toString() { return "Y"; }
}

public class Exceptions {
  public static int f(int x) throws Exc1 {
    try {
      x = g(x);
      System.out.print("F");
    } catch (Exc2 exc) {
      System.out.print("B");
      return x + 3;
    } finally {
      System.out.print("N");
    }
    return x + 5;
  }

  public static int g(int x) throws Exc1, Exc2 {
    if (x > 0) throw new Exc1();
    if (x == 0) throw new Exc2();
    System.out.print("G");
    return -x;
  }

  public static void main(String[] args) {
    try {
      int i = IO.readInt();
      int y = f(i);
      System.out.print(y);
    } catch (Exc1 exc) {
```

```
      System.out.print(exc.toString());
    }
    System.out.print("E");
  }
}
```

## Aufgabe 32:

Gegeben seien folgende Java-Klassen (die Implementierung der Methoden ist nicht von Interesse)

```
class Kunde {
  // liefert den (eindeutigen) Namen des Kunden
  public String getName()

  // liefert die bisherigen Bestellungen des Kunden
  public Bestellung[] getBestellungen()
}

class Artikel {
  // liefert eine (eindeutige) Nummer des Artikels
  public String getNummer()

  // liefert eine textuelle Bezeichnung des Artikels
  public String getBezeichnung()

  // liefert den Preis des Artikels
  public double getPreis()
}

class Bestellung {

  // liefert den Kunden, der die Bestellung gemacht hat
  public Kunde getKunde()

  // liefert die Artikel der Bestellung
  public Artikel[] getArtikel()
}

class KeinKundeException extends Exception {

  public KeinKundeException(String kundenName) {
    super(kundenName);
  }
}

class Shop {

  ArrayList<Kunde> kunden; // registrierte Kunden des Shops
  ArrayList<Artikel> artikel; // im Shop erhaeltliche Artikel

  public Shop() {
    kunden = new ArrayList<Kunde>();
    artikel = new ArrayList<Artikel>();
  }


  // .. weitere Methoden zum Anmelden neuer Kunden,
  // zum Einfügen von Artikeln und
  // zum Bestellen von Artikeln
```

```
}
```

Achtung: Sie dürfen die gegebenen Klassen nicht ändern oder um Attribute erweitern.

Aufgabe: Ergänzen Sie die Klasse **`Shop`** um eine Methode

**`public double getEinnahmen(String kundenName)`**

**`  throws KeinKundeException`**

Der Methode wird der Name eines Kunden übergeben. Existiert im Shop kein registrierter Kunde mit diesem Namen, soll eine **`KeinKundeException`** geworfen werden. Existiert im Shop ein registrierter Kunde mit diesem Namen, soll die Methode die gesamten Einnahmen aus allen Bestellungen dieses Kunden im Shop berechnen und liefern.

Hinweis: Sie können voraussetzen, dass die Methoden der vorgegebenen Klassen keine **`null`**-Werte liefern.

## Aufgabe 33:

Stellen Sie sich vor, zum Auslesen des Inhalts von Dokumenten (Text-Dateien) stehe Ihnen folgende API zur Verfügung:

```
class IOException extends Exception {}
class DocumentNotExistsException extends Exception {}
class AccessException extends Exception {}
class EndOfDocumentException extends Exception {}

class Document {
  /**
   * oeffnet ein Dokument mit dem uebergebenen Namen;
   *
   * @param name Name des zu oeffnenden Dokumentes
   * @throws DocumentNotExistsException
   *           wenn zum uebergebenen Namen kein Dokument
   *           existiert, wird das Dokument nicht geoeffnet,
   *           sondern eine DocumentNotExistsException geworfen
   * @throws AccessException
   *           wenn der Benutzer keine Zugriffsrechte auf
   *           das Dokument hat, wird das Dokument nicht
   *           geoeffnet, sondern eine AccessException geworfen
   */
  Document(String name)
    throws DocumentNotExistsException, AccessException

  /**
   * liest eine Zeile eines geoeffneten Dokumentes ein und liefert
   * sie als String-Objekt; Dokumente koennen von vorne bis hinten
   * zeilenweise ausgelesen werden; jeder Aufruf dieser Methode liefert
   * die jeweils naechste Zeile
   *
   * @return Inhalt der naechsten Zeile des Dokumentes als
   *         String-Objekt
   * @throws EndOfDocumentException
   *           wenn keine weitere Zeile mehr existiert, liefert die
   *           Methode kein String-Objekt, sondern wirft
```

```
     *          eine EndOfDocumentException
     * @throws IOException
     *           bei internen Problemen (bspw. wenn das Dokument nicht
     *           erfolgreich geoeffnet wurde) liefert die Methode kein
     *           String-Objekt, sondern wirft eine IOException
     */
    String readLine() throws EndOfDocumentException, IOException

    /**
     * schliesst ein geoeffnetes Dokument; ein Dokument, das erfolgreich
     * geoeffnet wurde, muss nach seiner Benutzung unbedingt wieder
     * geschlossen werden, da es ansonsten nie wieder zugreifbar ist
     */
    void close()
}
```

Implementieren Sie mit Hilfe dieser API eine Klasse **DocumentReader**, deren main-Funktion der Name eines Dokumentes als Parameter übergeben wird, dessen Inhalt zeilenweise auf die Konsole ausgegeben werden soll. Vor und nach dem Inhalt soll eine Zeile **Beginn <Dokumentname>** bzw. **Ende <Dokumentname>** ausgegeben werden (siehe Beispiel unten). Im Falle von Fehlern/Exceptions soll das Lesen des Dokumentes abgebrochen und adäquate Fehlermeldungen auf die Konsole ausgegeben werden.

Achtung: Die Verwendung mehrerer try-Blöcke ist zur Lösung dieser Aufgabe unnötig und unschön und führt zu Punktabzügen!

Beispiel:

Inhalt Dokument **Klausur.txt** (2 Zeilen):

**Aufgabe 1**

**Aufgabe 2**

Programmaufruf: **java DocumentReader Klausur.txt**

Programmausgabe im Erfolgsfall:

**Beginn Klausur.txt**

**Aufgabe 1**

**Aufgabe 2**

**Ende Klausur.txt**

Programmausgabe im Fehlerfall **DocumentNotExistsException**:

**Dokument Klausur.txt existiert nicht**

Programmausgabe im Fehlerfall **AccessException**:

**Sie haben keine Zugriffsrechte auf Dokument Klausur.txt**

Programmausgabe im Fehlerfall **IOException** (Beispiel):

**Beginn Klausur.txt**

**Aufgabe 1**

**Dokument Klausur.txt konnte wegen eines internen Fehlers nicht (vollstaendig) ausgelesen werden**

## Aufgabe 34:

Gegeben sei folgende Klasse **Stack**, die die aus der Vorlesung bekannte Datenstruktur eines Stacks implementiert:

```
final class FullException extends Exception { }
final class EmptyException extends Exception { }

class Stack {
    private int[] store; // zum Speichern von Daten
    private int current; // aktueller Index

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

    final void push(int value) throws FullException {
        if (this.current == this.store.length - 1)
            throw new FullException();
        this.store[++this.current] = value;
    }

    final int pop() throws EmptyException {
        if (this.current == -1)
            throw new EmptyException();
        return this.store[this.current--];
    }
}
```

Teilaufgabe (1):

Anstelle einer Abfrage-Methode **isFull** behandelt die Klasse **Stack** den entsprechenden Fall durch das Werfen einer **FullException** Leiten Sie von der Klasse **Stack** eine Klasse **ExtStack** ab, die neben ggfls. notwendigen Konstruktoren als einzige Methode eine zusätzliche Seiteneffekt-freie Methode **boolean isFull()** implementiert. Diese Methode soll genau dann **true** liefern, wenn der Stack voll ist, d.h. keine weiteren Werte mehr auf den Stack gepusht werden können. Beachten Sie unbedingt: Sie dürfen die gegeben Klassen nicht ändern oder erweitern. Die geerbten Methoden sind als **final** und die geerbten

Attribute als **`private`** deklariert. Tipp: Rufen Sie die geerbten Methoden adäquat auf und fangen Sie die Exceptions auf geeignete Art und Weise ab.

Folgendes Programm skizziert den Einsatz der Klasse **`ExtStack`**:

```
public static void main(String[] args) throws Exception {
        ExtStack stack = new ExtStack(5);
        while (!stack.isFull()) {
            stack.push(IO.readInt());
        }
        // ...
 }
```

Teilaufgabe (2):

Anstelle einer Abfrage-Methode **`isEmpty`** behandelt die Klasse **`Stack`** den entsprechenden Fall durch das Werfen einer **`EmptyException`** Leiten Sie von der Klasse **`Stack`** eine Klasse **`ExtStack`** ab, die neben ggfls. notwendigen Konstruktoren als einzige Methode eine zusätzliche Seiteneffekt-freie Methode **`boolean isEmpty()`** implementiert. Diese Methode soll genau dann **`true`** liefern, wenn der Stack leer ist, d.h. keine weiteren Werte mehr mittels **`pop`** vom Stack geholt werden können. Beachten Sie unbedingt: Sie dürfen die gegeben Klassen nicht ändern oder erweitern. Die geerbten Methoden sind als **`final`** und die geerbten Attribute als **`private`** deklariert. Tipp: Rufen Sie die geerbten Methoden adäquat auf und fangen Sie die Exceptions auf geeignete Art und Weise ab.

Folgendes Programm skizziert den Einsatz der Klasse **`ExtStack`**:

```
public static void main(String[] args) throws Exception {
        ExtStack stack = new ExtStack(5);
        // ...
        while (!stack.isEmpty()) {
            IO.println(stack.pop());
        }
}
```

## Aufgabe 35:

Definieren und implementieren Sie eine ADT-Klasse **`Statistik`**, die bestimmte statistische Funktionen auf einer Menge an **`int`**-Werten als Methoden zur Verfügung stellt:

- Einen Konstruktor, der eine anfangs leere Zahlenmenge initialisiert. Nutzen Sie für die interne Datenstruktur die Klasse **`java.util.ArrayList`**.
- Eine Methode zum Aufnehmen eines neuen **`int`**-Wertes in die Menge.
- Eine Methode, die die Summe aller Zahlen der Menge berechnet und als Funktionswert liefert.
- Eine Methode, die den Mittelwert aller Zahlen der Menge berechnet und als Funktionswert liefert.
- Eine Methode, die eine **`java.util.ArrayList`** liefert, die genau die positiven Zahlen der Menge enthält.

Überlegen Sie, wo Probleme auftreten können und behandeln Sie diese Probleme durch den adäquaten Einsatz von Exceptions.

Folgendes Programm demonstriert eine mögliche Nutzungsweise der Klasse **`Statistik`**. Halten Sie sich, was die Namen der Methoden Ihrer Klasse angeht, an die Namen der in dem Programm aufgerufenen Methoden.

```
    public static void main(String[] args) throws Exception {
        Statistik zahlen = new Statistik();
        int anzahl = IO.readInt("Anzahl: ");
        for (int i = 1; i <= anzahl; i++) {
            zahlen.neuerWert(new Random().nextInt());
        }
        System.out.println("S=" + zahlen.summe());
        System.out.println("MW=" + zahlen.mittelwert());
        for (int pos : zahlen.positiveWerte()) {
            System.out.println(pos);
        }
    }
```

## Aufgabe 36:

Stellen Sie sich vor, ein API-Programmierer stellt folgende Klasse mit mathematischen Funktionen zur Verfügung:

```
public class Mathe {

 public static final int KEIN_FEHLER = 0;
 public static final int KEINE_WERTE_FEHLER = 1;
 public static final int KEIN_POS_PENDANT_FEHLER = 2;
 public static final int AUSSERHALB_WERTEBEREICH_FEHLER = 3;

 public static int fehler = KEIN_FEHLER;

 public static double mittelwert(double[] werte) {
  if (werte == null || werte.length == 0) {
   fehler = KEINE_WERTE_FEHLER;
   return 0;
  }
  double erg = werte[0];
  for (int i = 1; i < werte.length; i++) {
   erg += werte[i];
  }
  fehler = KEIN_FEHLER;
  return erg / werte.length;
 }

 public static int betrag(double wert) {
  if (wert == Integer.MIN_VALUE) {
   fehler = KEIN_POS_PENDANT_FEHLER;
   return 0;
  }
  if (wert < Integer.MIN_VALUE || wert > Integer.MAX_VALUE) {
   fehler = AUSSERHALB_WERTEBEREICH_FEHLER;
   return 0;
  }
  fehler = KEIN_FEHLER;
  int intWert = (int) wert;
  return intWert < 0 ? -intWert : intWert;
```

```
  }
 }
```

Ein Anwendungsprogrammierer nutzt die Klasse zur Implementierung eines Auswertungsprogramms:

```
public class MatheNutzung {

 static void printBerechnungen(double[][] werte) {
  for (int r = 0; r < werte.length; r++) {
   IO.println("Messreihe " + (r + 1) + ": ");

   double mittel = Mathe.mittelwert(werte[r]);
   if (Mathe.fehler == Mathe.KEINE_WERTE_FEHLER) {
    IO.println("Fehler: keine Werte in Messreihe " + (r + 1));
   } else { // kein Fehler
    int ergebnis = Mathe.betrag(mittel);
    if (Mathe.fehler == Mathe.AUSSERHALB_WERTEBEREICH_FEHLER) {
     IO.println("Fehler: Ergebnis zu klein oder zu gross");
    } else if (Mathe.fehler == Mathe.KEIN_POS_PENDANT_FEHLER) {
     IO.println("Ergebnis ist 2.147.483.648");
    } else { // kein Fehler
     IO.println("Ergebnis = " + ergebnis);
    }
   }
  }
 }

 // nur ein beispielhaftes Testprogramm!
 public static void main(String[] args) {
  double[][] werte = { { 2, 8, 5 }, null,
                       { -2147483649.0, -2147483647.0 },
                       { -12147483649.0, -12147483647.0 },
                       { -4, -8 } };
  printBerechnungen(werte);
 }
}
```

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

- Für jeden Fehlertyp wird eine Konstante definiert.
- Wenn in einer Funktion (hier **`betrag`**, **`mittelwert`**) ein Fehler auftritt, wird einer globalen Fehlervariablen (hier **`fehler`**) der Wert der entsprechenden Fehlerkonstanten zugewiesen.
- Prozeduren bzw. Programme, die eine Funktion aufrufen (hier **`printBerechnungen`**), müssen nach jedem Funktionsaufruf den Wert der Fehlervariablen überprüfen.

**Aufgabe**:

Wandeln Sie zunächst die Klasse **`Mathe`** um in eine äquivalente Klasse, wobei eine Fehlerbehandlung mit Hilfe von Exceptions durchführt. Definieren Sie geeignete Exception-Klassen anstelle der Konstanten und passen Sie die Funktionen **`mittelwert`** und **`betrag`** entsprechend an.

Wandeln Sie anschließend die Prozedur **`printBerechnungen`** um in eine äquivalente Prozedur, die die geänderten Funktionen der geänderten Klasse **`Mathe`** nutzt. Äquivalent bedeutet, dass beide Prozeduren bei gleichen Parameterwerten exakt gleiche Ausgaben produzieren!