1. Aufgabe

In Kapitel 5 Aufgabe 11 wurde am Beispiel des d'Hondt-Verfahrens, ein Beispiel zur Ermittlung der Sitz-Verteilung in einem Landtag auf der Basis der gegebenen Stimmen-Verteilung. In dieser Aufgabe soll der entsprechende Algorithmus in eine separate Klasse gekapselt werden und im Stil einer mathematischen Bibliotheksfunktion angeboten werden.

Es soll also eine Klasse DeHondt angeboten werden, welche die statische Methode **berechne** anbietet mit der folgenden Signatur: **static int [ ] berechne (int [ ] stimmen, int sitze)**. Die Methode berechne gibt ein Feld zurück, welches für jede Partei die Anzahl der Sitze enthält. Des Weiteren soll eine Klasse Test implementiert werden, in der die entsprechenden Eingabewerte von der Tastatur eingelesen werden und die Daten auf dem Heap angelegt werden. Die Ausgabe soll ebenfalls in der Klasse Test umgesetzt werden.

2. Aufgabe

Die Methode random aus der Klasse Math liefert eine zufällige Zahl x zwischen 0 und 1. Dabei liegt x im Bereich [0,1). Die Methode ist so implementiert, dass dabei jede Zahl gleich verteilt angenommen wird.

Mit Hilfe dieser Methode können nun beliebige Zufalls-Generatoren realisiert werden. Im folgenden soll eine Klasse Wuerfel implementiert werden, die verschiedene Methoden zum „Würfeln" einer Zahl anbietet, also genau eine Zahl aus der Menge {1, 2, 3, 4, 5, 6} zurück liefert. Es sollen zwei solcher Methoden, nennen wir diese gut und falsch angeboten werden, die jeweils eine Zahl zwischen 1 und 6 zurück geben. Allerdings soll nur die Methode gut die Gleichverteilung korrekt erfüllen.

Schreiben Sie eine Klasse Test, welches die beiden Methoden gut und falsch auf die Eigenschaft, gleich verteilt zu sein, überprüft.

3. Aufgabe

(a) Besitzt die folgende Klasse einen Standard-Konstruktor?

```
class A {
      int a;
      A (int i) {
         a = i;
      } // Konstruktor
} // A
```

(b) Compiliert das folgende Java-Programm, welches aus den beiden Klassen Basis und Sub bestgeht?

```
class Basis {
        int zahl;
        Basis (int n) {
          zahl = n;
        }
} // Basis

und

class Sub extends Basis {
        int value:
 // Sub
```

4. Aufgabe

Eine in der Informatik häufig benötigte Datenstruktur ist der Stack (Kellerspeicher). Ein Stack ist ein Stapel von Objekten, auf den man Objekte drauflegen und von dem man Objekte entfernen kann. Das zuletzt auf dem Stack abgelegte Element wird als erstes wieder beim Herausnehmen entfernt. Daher wird der Stack auch eine LIFO-Datenstruktur (last in first out) genannt.

Mit Hilfe eines Java-Programms soll ein einfacher Stack (als Klasse) implementiert werden, der als Objekte nur Integer-Werte verwaltet. Es müssen also die beiden folgenden öffentlichen Methoden angeboten werden:

(a) void push (int element) und

(b) int pop ( )

Bei der Realisierung soll darauf geachtet werden, dass auf die internen Strukturen von Außen nicht zugegriffen werden kann. Im Wesentlichen soll ein Objekt der Klasse Stack nur über die beiden Methoden push und pop ansprechbar sein. Als interne Realisierung bietet sich ein Array an. Insgesamt sollen zwei Konstruktoren angeboten werden. Mit Hilfe des parameterlosen Konstruktors kann eine feste (im Programm festgelegte) Stackgröße erreicht werden. Ein weiterer Konstruktor erwartet einen Parameter, der zum Zeitpunkt der Objekt-Generierung die gewünschte Stackgröße angibt.

Zum Testen der Klasse Stack soll eine weitere Klasse TestStack geschrieben werden. Mit Hilfe der main-Methode in der Klasse TestStack sollen interaktiv die Methoden push und pop angestossen werden können. Das main-Programm erwartet wahlweise keinen oder einen Parameter, welche die Stackgröße angibt.

5. Aufgabe

Die Lösung der Aufgabe 4 soll nun um Exception Handling erweitert werden. Im Wesentlichen müssen also zwei Exceptions StackFull und StackEmpty definiert und berücksichtigt werden.

6. Aufgabe

Das Bank-Beispiel aus Kapitel 7.6 soll geeignet um Exception Handling erweitert werden. Im Wesentlichen muß dazu nur eine Exception-Klasse eingeführt werden.

7. Aufgabe

Erstellen Sie eine Klasse Beleg, deren Objekte automatisch eine bei der Zahl 10000 beginnende laufende Belegnummer erhalten.

8. Aufgabe

Sind die folgenden Anweisungen korrekt? Dabei soll davon ausgegangen werden, dass die Klasse K2 eine Subklasse von K1 sei:

```
K1 p1 = new K1 ( );
K2 p2 = new K2 ( );
p1 = p2;
p2 = (K2) p1;
```

9. Aufgabe

Fangen Sie die ganzzahlige Division durch 0 mit einer try-Anweisung ab. Die entsprechende Ausnahme trägt den Namen: ArithmeticException.

10. Aufgabe

Die ganzzahlige Variable *zufallszahl* soll in einer Schleife zufällige Werte zwischen 0 und 9 annehmen. Es soll eine Ausnahme ausgelöst und abgefangen werden, wenn der Wert 0 angenommen wird.

Zu diesem Zweck ist eine Ausnahmeklasse *IstNull* zu erstellen, die eine entsprechende Fehlermeldung speichert.

Bemerkung: Die Zufallszahl kann mittels *(int) (Math.random () * 10.)* erzeugt werden.

11. Aufgabe

Die Klasse Stack aus Aufgabe 4 soll zu einer Klasse AdaptiveStack erweitert werden, die den Kellerspeicher automatisch vergrößert, wenn er überläuft.

Zur Vergrößerung des Stacks muss mindestens ein um einen Speicherplatz vergrößertes Array angelegt werden und es müssen die Elemente des alten Arrays in das neue umkopiert werden. Des weiteren soll eine neue Methode size angeboten werden, welche die (aktuelle) Größe des Stacks zurückgibt. Insgesamt sollen wieder (genau wie in Aufgabe 4) zwei Konstruktoren angeboten werden.

Zur Realisierung der gewünschten Funktionalität müssen also neben den neuen Konstrukturen auch eine neue Methode size implementiert und die "alte" Methode push überschrieben werden. Bitte gehen Sie bei der Implementierung möglichst strukturiert vor und versuchen nur die neuen Anteile der Methode push zu implementieren und die "alten" Anteile der Methode push durch Aufruf der entsprechenden Methode der Basisklasse zu realisieren.

Genau wie in Aufgabe 4 sollen alle internen Datenstrukturen nicht von Außen zugreifbar sein. Zum Testen der Klasse AdaptiveStack soll eine weitere Klasse TestAdaptiveStack geschrieben werden. Mit Hilfe der main-Methode in der Klasse TestAdaptiveStack sollen nun neben den Methoden push und pop auch die Methode size interaktiv angestossen werden können Das main-Programm erwartet wahlweise keinen oder einen Parameter, welche die Stackgröße angibt.

12. Aufgabe

Implementieren Sie ein Wörterbuch als Klasse Dictionary. Das Wörterbuch soll Paare von Wörtern enthalten, wie z.B. ein deutsches Wort und seine spanische Übersetzung. Dabei soll also das erste Wort als Suchkriterium dienen und das zweite als Wert, der dazu gefunden werden soll.

Zumindest die beiden folgenden Operationen sollen öffentlich angeboten werden sowie geeignete Konstruktoren:

(a) void insert (String key, String value)

(b) String lookup (String key)

Mit Hilfe der Methode insert wird key und value in das Wörterbuch eingetragen. Mit Hilfe von lookup wird key im Wörterbuch gesucht und das entsprechende value oder null, falls key nicht gefunden wird, zurückgeliefert.

Bei der Realisierung soll darauf geachtet werden, dass auf die internen Strukturen von Außen nicht zugegriffen werden kann. Im Wesentlichen soll ein Objekt der Klasse Dictionary nur über die beiden Methoden insert und lookup ansprechbar sein. Mit Hilfe des parameterlosen Konstruktors kann eine feste (im Programm festgelegte) Größe des Wörterbuchs erreicht werden. Ein weiterer Konstruktor erwartet einen Parameter, der zum Zeitpunkt der Objekt-Generierung die gewünschte Größe angibt.

Zum Testen der Klasse Dictionary soll eine weitere Klasse TestDictionary geschrieben werden. Mit Hilfe der main-Methode in der Klasse TestDictionary sollen interaktiv Wörterpaare eingetragen und später einzelne Wörter erfragt werden können. Das main-Programm erwartet wahlweise keinen oder einen Parameter, welche die Größe des Wörterbuches angibt.

13. Aufgabe

Ein Vektor im zweidimensionalen Raum kann durch zwei reelle Zahlen dargestellt werden. Implementieren Sie eine Klasse Vector, welche Operationen für die Addition von Vektoren, die Multiplikation eines Vektors mit einem Skalar sowie für das Skalarprodukt zweier Vektoren anbietet.

Das Skalarprodukt (inneres Proukt) zweier Vektoren liefert einen Wert und ergibt sich aus der Summe der Produkte der Komponenten. Die Addition zweier Vektoren und die Multiplikation eines Vektors mit einem Skalar liefert einen neuen Vektor und ist kanonisch definiert.

Implementieren Sie zusätzlich eine Methode betrag, welche die Länge des Vektors der Klasse berechnet. Die Länge ergibt sich aus der euklidischen Norm und kann mit Hilfe des Skalarprodukts berechnet werden (Wurzel aus dem Skalarprodukt des Vektors mit sich selbst).

Stellen Sie auch einen geeigneten Konstruktor bereit und vergeben die Zugriffsrechte sparasam. Berücksichtigen Sie bitte auch die Bereitstellung einer Methode zur Ausgabe eines Vektors, um Detail-Zugriffe auf das Objekt zu vermeiden. Implementieren Sie eine weitere Klasse TestVector zum interaktiven Testen der Operationen der Klasse Vector.