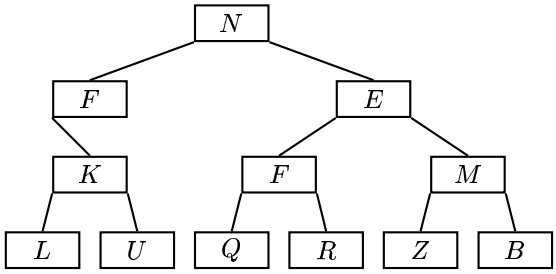
• **AUFGABE 1 (6 Punkte) Theorie.**

1. (2 Punkte) Erläutern Sie, unter welchen Bedingungen die Verwendung von *Insertion Sort* besser als die Verwendung von *Merge Sort* ist, und umgekehrt.

2. (2 Punkte) Erläutern Sie die Unterschiede und Gemeinsamkeiten der abstrakten Datentypen *Stack* und *Queue*.

3. (2 Punkte) Ermitteln Sie die Zeichenfolgen, die sich bei der Inorder-, Preorder- bzw. Postorder-Traversierung des folgenden Binärbaumes ergeben. Ist der Baum balanciert? Begründen Sie Ihre Antwort.



- **AUFGABE 2 (7 Punkte) Java.**

1. (1 Punkt) Wie können in Java Konstanten deklariert werden?

2. (2 Punkte) Was sind abstrakte Klassen und welchen Zweck erfüllen Sie? Welche Einschränkungen gelten für abstrakte Klassen?

3. (1 Punkt) Wozu dient das Schlüsselwort `this`?

4. (3 Punkte) Welche Fehler enthält folgendes Java-Code-Fragment? Geben Sie jeweils die Zeilennummer an und beschreiben Sie den Fehler. Folgefehler werden ignoriert.

```
 1 class A extends B {
 2    A () {
 3     }
 4
 5     A (int x) {
 6         this.x = x;
 7     }
 8 }
 9
10 class B extends C{
11     final int x = 0;
12
13     void f (int x) {
14         y = x;
15         x = 42;
16     }
17
18     int g () {
19         return x;
20     }
21 }
22
23 class C {
24     short y;
25
26     int d () {
27         return g ();
28     }
29 }
```

• AUFGABE 3 (5 Punkte) Numerik.

Schreiben Sie eine Java-Methode `double ln(double x)`, welche den natürlichen Logarithmus einer Zahl x berechnet.

Dabei soll der Logarithmus approximativ mittels der Näherungsformel

$$\ln x = -\sum_{k=1}^{\infty} \frac{(1-x)^k}{k}$$

berechnet werden, die für $0 < x < 2$ gilt.

Beenden Sie den Approximationsprozess, wenn eine Genauigkeit von 10 Stellen erreicht ist.

• AUFGABE 4 (9 Punkte) Vererbung.

Erstellen Sie ein objekt-orientiertes Modell für motorisierte Dienste: *Polizei*, *Notarzt* und *Pizza-Service*. Dieses Modell soll als Java-Code aufgeschrieben werden.

1. (2 Punkte) Definieren Sie eine Oberklasse `Fahrzeug`, die folgende Methoden bereitstellt:
   - `abstract int telefon()` liefert die Telefonnommer, unter der der jeweilige Dienst gerufen werden kann.
   - `abstract boolean blaulicht()` liefert `true`, falls das Fahrzeug mit Blaulicht fährt (d.h., ein Blaulicht besitzt und eingeschaltet hat).
   - `boolean bremstFür(Fahrzeug other)` (nicht abstrakt) liefert `true`, falls das Fahrzeug `this` dem Fahrzeug `other` auf der Straße die Vorfahrt überläßt.

     Allgemein gilt: es muß nur für Fahrzeuge gebremst werden, die mit Blaulicht fahren.
2. (1 Punkt) Definieren Sie eine Klasse `AmtsFahrzeug` als Unterklasse von `Fahrzeug` für Fahrzeuge der Polizei bzw. der Notärzte, die mit einem Blaulicht ausgestattet sind.
   - Der Zustand des Blaulichts (ein- oder ausgeschaltet) soll über ein von außen *nicht* zugängliches Attribut dargestellt werden.
   - Programmieren Sie auch die Methode `boolean blaulicht()`.
3. (4 Punkte) Definieren Sie zwei Klassen `Polizei` und `Notarzt` als Unterklassen von `AmtsFahrzeug`. Die Polizei ist immer unter der Telefonnummer 110 erreichbar, der Notarzt unter der Nummer 112.
   - Programmieren Sie in beiden Klassen die Methode `int telefon()`.
   - Überschreiben Sie in beiden Klassen die Methode `boolean bremstFür(Fahrzeug other)`. Dabei gelten folgende Vorfahrtsregeln:
     - Ein Fahrzeug der Polizei mit aktivem Blaulicht bremst nur für Notärzte, die ebenfalls mit Blaulicht fahren.
     - Ein Notarzt mit aktivem Blaulicht bremst für niemanden.
     - Polizei und Notarzt verhalten sich wie gewöhnliche Fahrzeuge, falls das Blaulicht nicht aktiv ist.
4. (2 Punkte) Definieren Sie eine Klasse `PizzaService` als Unterklasse von `Fahrzeug`. Pizza-Fahrzeuge dürfen *kein* Blaulicht besitzen.
   - Die Telefonnummer des jeweiligen Pizza-Services soll als von außen *nicht* zugängliches Attribut dargestellt werden.
   - Schreiben Sie einen geeigneten Konstruktor, um die Telefonnummer zu initialisieren.
   - Programmieren Sie auch alle geerbten Methoden, die noch abstrakt sind.

• AUFGABE 5 (10 Punkte) Supermarkt-Simulation.

Sie sollen ein Programm zur Simulation von Supermarktkassen schreiben. Dabei soll für alle fünf Kassen eines großen Supermarktes das Verhalten von Kunden an den Kassen simuliert werden. Da es vorkommen kann, dass sich die Kunden schneller an den Kassen anstellen, als sie bedient werden können, können sich Schlangen bilden.

Zur Bearbeitung steht Ihnen die Klasse `Queue` zur Verfügung, welche die folgenden Methoden besitzt:

- Der Konstruktor `Queue()` erzeugt eine leere Queue.
- Die Methode `void insert(Object o)` fügt ein Object an das Ende der Queue an.
- Die Methode `Object remove()` entfernt das erste Objekt aus der Queue und liefert es zurück. Diese Methode darf nicht aufgerufen werden, wenn die Queue leer ist.
- Die Methode `int length()` liefert die Anzahl der Elemente in der Queue.

Weiterhin ist die Klasse `Client` vordefiniert, die lediglich den Konstruktor `Client()` besitzt. Objekte dieser Klasse sollen die zu simulierenden Kunden repräsentieren.

1. (2 Punkte) Erstellen Sie eine Klasse `Simulation`, welche als Attribute fünf Queues verwaltet. Schreiben Sie einen geeigneten Konstruktor, der diese Queues initialisiert.
2. (2 Punkte) Schreiben Sie eine Methode `void enter(Client c)`, welche das Eintreffen eines Kunden `c` simuliert. Ankommende Kunden sollen sich immer an der kürzesten Schlange anstellen.
3. (2 Punkte) Schreiben Sie eine Methode `void exit()`, welche eine zufällige Schlange auswählt und aus dieser Schlange den ersten Kunden entfernt. Sollte die ausgewählte Schlange leer sein, passiert nichts.
4. (2 Punkte) Schreiben Sie eine Methode `void simulate()`, die eine Folge von 300 Simulationsschritten ausführt. Ein Simulationschritt besteht aus
   - dem Eintreffen eines neuen Kunden und
   - dem Abfertigen (Entfernen) eines Kunden.
5. (2 Punkte) Schreiben Sie eine Methode `void nervous(int i)`, die dafür sorgt, dass dem am Anfang der Schlange i stehenden Kunden einfällt, dass er etwas vergessen hat. Dieser Kunde soll seine Schlange verlassen und sich an der kürzesten Schlange wieder anstellen.

Hinweise:

- Zur Ermittlung einer zufälligen ganzen Zahl zwischen 0 und $n-1$ (inklusive) benutzen Sie den Ausdruck:

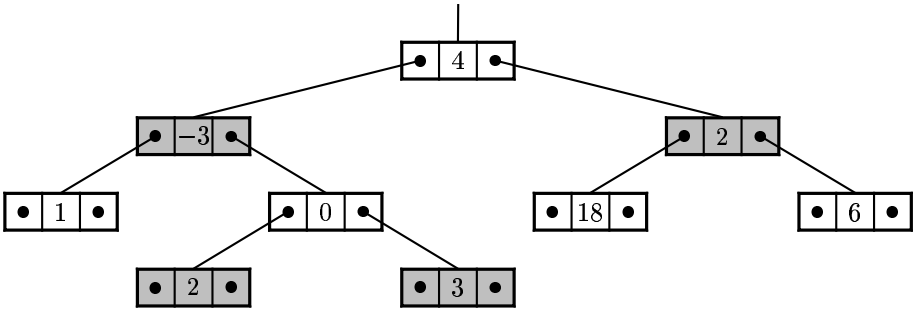
  ```
  (int) Math.floor (Math.random () * n)
  ```

- Wenn Sie für eine Teilaufgabe keine Lösung haben, können Sie die entsprechende Methode in anderen Teilaufgaben trotzdem aufrufen.

• AUFGABE 6 (7 Punkte) Zebra-Bäume.

Für diese Aufgabe definieren wir sogenannte Zebra-Bäume. Dies sind Bäume, deren Knoten entweder weiß oder schwarz gefärbt sind, und bei denen alle Knoten einer Ebene die gleiche Farbe haben. Bei Zebra-Bäumen haben alle inneren Knoten genau zwei Nachfolger und alle Blätter keine Nachfolger.

Beispiel:



Die Klasse `ZebraNode` stellt folgende Methoden zur Verfügung:

- `boolean isLeaf()` — `true`, wenn der aktuelle Knoten ein Blatt ist
- `boolean isBlack()` — `true`, wenn der aktuelle Knoten schwarz gefärbt ist
- `void setBlack(boolean black)` — ändert die Farbe des aktuellen Knotens auf `black`
- `int value()` — liefert den Wert des aktuellen Knotens
- `ZebraNode left()` — linker Nachfolger (darf nur auf inneren Knoten aufgerufen werden)
- `ZebraNode right()` — rechter Nachfolger (darf nur auf inneren Knoten aufgerufen werden)

1. (2 Punkte) Fügen Sie der Klasse `ZebraNode` eine Methode `void recolor()` hinzu, die die Farbe jedes Knoten im Baum umfärbt.
2. (3 Punkte) Fügen Sie der Klasse `ZebraNode` eine Methode `boolean check()` hinzu, die prüft, ob sie auf einem korrekten Zebra-Baum aufgerufen wurde. Ein Zebra-Baum ist dann korrekt, wenn
   - die direkten Nachfolger eines Knotens eine andere Farbe als der Knoten selbst haben *und*
   - die Unterbäume des Knotens korrekte Zebra-Bäume sind.
3. (2 Punkte) Fügen Sie der Klasse `ZebraNode` eine Methode `int sumWhiteOrBlack(boolean black)` hinzu, die die Werte an den schwarzen Knoten addiert, wenn der Parameter `black` true ist, und andernfalls die Werte an den weißen Knoten addiert.