Aufgaben zur Abgabe als Prüfungsleistung für Prog I

- Allgemeine Regeln
  - Es gelten dieselben Regeln wie für Aufgaben 1 und 2, nur ein anderer Abgabetermin

Tamagotchi

https://en.wikipedia.org/wiki/Tamagotchi

- Das Tamagotchi ist ein virtuelles Küken, um das man sich vom Zeitpunkt des Schlüpfens an wie um ein echtes Haustier kümmern muss. Es hat Bedürfnisse wie schlafen, essen, trinken, Zuneigung usw.
  - Es wurden bis 2010 rund 76 Mio. Tamagotchis verkauft
  - Wir erstellen ein sehr vereinfachtes Java-Programm hierzu

3. Aufgabe: Tamagotchi

- Erstellen Sie eine objektorientierte Lösung zur Simulation
  - Ein Tamagotchi hat einen Namen, einen Sättigungszustand (0.0 = verhungert bis 1.0 = satt), einen Glückszustand (0.0 bis 1.0 = sehr glücklich), ist lebend oder tot
  - Ein neues T ist 50% satt und 50% glücklich, es lebt natürlich
  - In der `main`-Methode wird ein neues T erzeugt und es werden drei Runden gespielt (sofern das T noch lebt, sonst wird abgebrochen)
    - Jede Runde zeigt den aktuellen Status an, ermöglicht eine Benutzereingabe (füttern oder unterhalten, jeweils 0,5 Erhöhung), vermindert den Sättigungszustand und Glückzustand durch eine Zufallszahl zwischen 0 und 0,5, erzeugt durch `Math.random()/2`
      Des weiteren wird in jeder Runde geprüft, ob das T noch lebt (keine Werte kleiner gleich 0 oder größer gleich 1)
    - Am Ende wird das Ergebnis angezeigt: Gewonnen (T lebt) oder verloren (mit Ursache: verhungert, geplatzt, verkümmert, verhätschelt)
  - Wenn ein T die drei Runden überlebt hat, kann es eine Familie gründen: Es wird in eine Liste eingefügt und kann bei erfolgreichem Spielverlauf maximal zwei Nachkommen in der Liste haben
    - Die Liste wird ausgegeben, sobald ein T in einer Familie stirbt, oder alle drei Angehörigen in der Familie ihre drei Runden überlebt haben; hat jedoch kein einziges T überlebt, wird eine leere Liste ausgegeben

Ausgaben für zwei beispielhafte Spiele und Ausgabe einer Familienliste (Zahlen mit zwei Nachkommastellen gerundet)

```
Runde 1 Animal1 satt: 0,50 gluecklich: 0,50 lebend: true
        Eingabe (f = fuettern, u = unterhalten): u

Runde 2 Animal1 satt: 0,45 gluecklich: 0,71 lebend: true
        Eingabe (f = fuettern, u = unterhalten): f

Runde 3 Animal1 satt: 0,64 gluecklich: 0,48 lebend: true
        Eingabe (f = fuettern, u = unterhalten): u

Gewonnen! Endzustand: Runde 3 Animal1 satt: 0,27 gluecklich: 0,01 lebend: true
```

```
Runde 1 Animal1 satt: 0,50 gluecklich: 0,50 lebend: true
        Eingabe (f = fuettern, u = unterhalten): f

Runde 2 Animal1 satt: 0,72 gluecklich: 0,45 lebend: true
        Eingabe (f = fuettern, u = unterhalten): u

Verloren ...verhungert Endzustand: Runde 2 Animal1 satt: -0,23 gluecklich: 0,80 lebend: false
```

```
Ausgabe der Familienliste:
Name1 satt: 0,50 gluecklich: 0,80 lebend: true
Name2 satt: 0,80 gluecklich: 0,50 lebend: true
Name3 satt: 0,60 gluecklich: 0,70 lebend: true
```

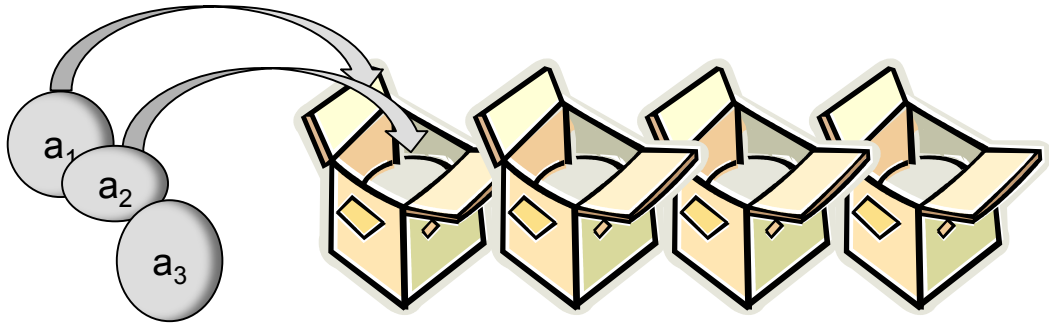
Tamagotchi

- Hinweis: Die Eingabe kann wie folgt implementiert werden, hierbei `import java.util.*;` hinzufügen

```
public void eingeben() {
      Scanner input = new Scanner(System.in);
      System.out.print("        Eingabe (f = fuettern, u = unterhalten): ");
      String eingabe = input.nextLine();
      if (eingabe.equals("f"))
                this.satt = this.satt + 0.5;
      if (eingabe.equals("u"))
                this.gluecklich = this.gluecklich + 0.5;
}
```

4. Aufgabe: Bin-Packing

- Aufgabenstellung des Behälterproblems (Bin-Packing)
  - Gegeben ist eine Anzahl k von Containern (engl.: bin, Behälter) der Kapazität b und eine Menge $a_1,\ldots, a_n$ von Objekten mit Gewicht (oder Größe) a_i
  - Kein Container darf überladen werden
  - Gesucht: Die ***kleinstmögliche*** Anzahl von Containern ohne Überladung
- Anwendungen
  - Wie kann man Lkws/Flugzeuge/Container ohne Platzverschwendung mit Paketen beladen?
  - Wie kann eine Zeitungsseite optimal mit verschieden großen Artikeln und Werbung ausgefüllt werden?



Vorgaben

- First-Fit-Algorithmus (kurz: FF, nur Näherungslösung)
  - Container werden der Reihe nach von links nach rechts aufgestellt
  - Die Objekte werden der Reihe nach gepackt: Jedes neue Objekt kommt in den am weitesten links stehenden Container, in den das Objekt hineinpasst
  - Falls keiner der teilweise befüllten Container mehr passt, wird ein neuer Container genommen
- First-Fit Decreasing-Algorithmus (kurz: FFD, Näherungslösung)
  - Analog First Fit, aber Objekte sind vor Beginn von groß nach klein sortiert
- Beispiel:
  - Maximale Kapazität eines Containers/Lkw: 7*t*
  - Objekte: 3*t*, 5*t*, 2*t*, 4*t*
  - First Fit:

  - First Fit Decreasing:

Vorgaben

- Erstellen Sie eine objektorientierte Lösung
  - Eine Liste von Ladungen (Klasse `LoadList.java`) besteht aus einem Feld von ganzen Zahlen, die die Objekte in Tonnen repräsentieren
  - Ein Container (Klasse `Container.java`) hat eine konstante, maximale Kapazität von 7 Tonnen; zur Vereinfachung können Sie festlegen, dass maximal 10 Ladungen in einem Container gespeichert werden. Auch die Liste von Containern dürfen Sie im Programm beschränken, es stehen nur 3 Container zur Verfügung, verwaltet in `ContainerList.java`. Implementierung mit Arrays.
  - Die `main`-Methode in `ContainerList.java` testet beide Methoden FF und FFD zur Beladung. Die Ausgabe soll wie nebenstehendes Beispiel aussehen

```
Ladeliste: [3, 5, 2, 4]
FIRST-FIT:
Container Nr.:  0
Anzahl Stuecke: 2
Inhalt: 3 2

Container Nr.:  1
Anzahl Stuecke: 1
Inhalt: 5

Container Nr.:  2
Anzahl Stuecke: 1
Inhalt: 4

Ladeliste: [3, 5, 2, 4]
FIRST-FIT-DECREASING:
Ladeliste sortiert: [5, 4, 3, 2]
Container Nr.:  0
Anzahl Stuecke: 2
Inhalt: 5 2

Container Nr.:  1
Anzahl Stuecke: 2
Inhalt: 4 3

Container Nr.:  2
Anzahl Stuecke: 0
Inhalt:
```