Informatik I: Einführung in die Programmierung

Prof. Dr. Bernhard Nebel — Universität Freiburg
Tim Schulte, Dr. Christian Becker-Asano, Dr. Stefan Wölfl — Institut für Informatik
Wintersemester 2014/2015

Übungsblatt 7

Abgabe: Freitag, 12. Dezember 2014, 18:00 Uhr

Wichtige Hinweise: Zur Bearbeitung der Übungsaufgaben legen Sie bitte ein neues Unterverzeichnis `sheet07` im Wurzelverzeichnis Ihrer Arbeitskopie des SVN-Repositories an. Ihre Lösungen werden dann entsprechend dem ersten Übungsblatt in Dateien in diesem Unterverzeichnis erwartet. Beachten Sie bitte bei allen Aufgaben die *Hinweise zur Bearbeitung der Übungsaufgaben* unter der folgenden URL:

`http://gki.informatik.uni-freiburg.de/teaching/ws1415/info1/hinweise_uebungen.txt`

Insbesondere müssen alle Python-Dateien und die darin definierten Funktionen entsprechend diesen Hinweisen dokumentiert und dabei PEP8 beachtet werden.

Unterbinden Sie alle nicht erforderlichen `print`-Anweisungen in Ihren Python-Dateien. Falls Sie zum Debuggen `print`-Anweisungen verwenden, benutzen Sie eine globale Variable oder Konstante, mit der Sie Ausgaben auf die Konsole ausschalten können.

Überprüfen Sie, dass Sie alle Lösungen ins Repository hochgeladen haben. Bewertet wird bei allen Aufgaben die letzte Version, die zur Deadline des Übungsblattes auf dem SVN-Server eingereicht ist.

Aufgabe 7.1 (Raute; Dateien: `raute.txt`, `hw_geoclasses.py`; Punkte: 1+4+4)

In dieser Aufgabe üben Sie die Implementierung einer eigenen Klasse `Rhombus`, mit der die Geometrie einer Raute[1] repräsentiert werden soll. Wir gehen dabei davon aus, dass diese Raute auf der Spitze steht, d.h. dass ihre Höhe und Breite parallel zu den Bildschirmkoordinaten sind. Um diese Aufgabe zu realisieren erweitern Sie die in der Vorlesung vorgestellten Geometrie-Klassen, die in der Datei "hw_geoclasses.py" definiert werden und die über folgenden URL bereitgestellt wird:
`http://gki.informatik.uni-freiburg.de/teaching/ws1415/info1/python/hw_geoclasses.py`

(a) Durch welche Parameter kann eine Raute sinnvoll und eindeutig beschrieben werden? Von welcher bereits existierenden Klasse sollte sich Ihre Klasse `Rhombus` sinnvoller Weise ableiten und warum?

Schreiben Sie Ihre Antworten zu dieser Teilaufgabe in die Datei "raute.txt" und vergessen Sie nicht, diese ins SVN zu committen.

(b) Ergänzen Sie die Datei `hw_geoclasses.py` mit Ihrer Implementierung der Klasse `Rhombus`. Diese Klasse soll neben einer geeigneten `__init__`-Methode auch die Methoden `area` und `change_size` enthalten. Dabei soll `area` analog zu den vorhandenen Klassen den Flächeninhalt zurückgeben und mittels `change_size`

[1] `http://de.wikipedia.org/wiki/Raute`

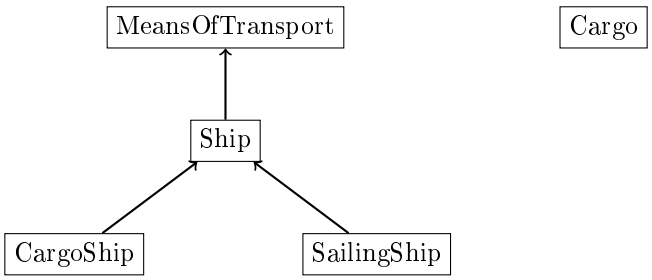
soll die Größe einer Raute prozentual verändert werden können. Erweitern Sie Ihre Klasse `Rhombus` nun um die beiden Methoden `stretch_height` und `stretch_width`, die die Höhe respektive Breite einer Raute prozentual verändern sollen, ohne dabei die jeweils andere Dimension zu verändern.

(c) Nutzen Sie das in der Vorlesung eingeführte Modul `tkinter`, um zwei verschiedene Instanzen der Klasse `Rhombus` nebeneinander als grafische Objekte zu visualisieren.

Vergessen Sie bitte nicht, Ihren Code sinnvoll und unter Berücksichtigung der obigen Bemerkungen zu dokumentieren und ins SVN zu committen.

Aufgabe 7.2 (Klassen; Datei: `logistics.py`; Punkte 1+4+2+2)

Betrachten Sie die folgende Klassenhierarchie.



(a) Implementieren Sie eine Klasse `Cargo`. Diese verfügt über die folgenden vier Instanzattribute:

1. `idnumber (int)`
2. `description (string)`
3. `weight (int)`
4. `location (string)`

(b) Implementieren Sie die Klasse `MeansOfTransport`. Diese repräsentiert Beförderungsmittel und verfügt über die Attribute:

- `location (string)`: aktueller Standort (ein Städtename)
- `cargocapacity (int)`: maximales Ladegewicht
- `cargo (list)`: Fracht, bestehend aus `Cargo`-Objekten

Ebenso verfügt sie über die folgenden Methoden:

- `__init__(self, position, cargocapacity)`
  Weist den Attributen initiale Werte zu.
- `move(self, destination)`
  Verändert die Position (`location`) des Objektes und seiner Fracht und gibt einen String der Form "Moves from <`position`> to <`destination`>" zurück.
- `cargo_weight(self)`
  Berechnet das Gesamtgewicht aller `Cargo`-Objekte und gibt dieses zurück.

- `add_cargo(self, new_cargo)`
  Fügt der `cargo`-Liste die Cargo-Instanz `new_cargo` hinzu sofern dadurch das maximale Ladegewicht nicht überschritten wird, sich das Transportmittel sowie die Cargo-Instanz am gleichen Ort befinden und `new_cargo` nicht bereits zur `cargo`-Liste gehört.
- `unload_cargo(self, cargo_id)`
  Entfernt ein Objekt aus der cargo-Liste anhand seiner Identifikationsnummer und gibt dieses zurück. Kann kein Objekt mit der entsprechenden Identifikationsnummer gefunden werden, so soll `None` zurückgegeben werden.
- `list_cargo(self)`
  Gibt zeilenweise die Beschreibung der Frachtgüter in `cargo` sowie deren jeweiliges Gewicht als Zeichenkette vom Typ `string` zurück.

(c) Erstellen Sie nun gemäß der Vererbungshierarchie die Klassen `Ship`, `CargoShip` und `SailingShip`. Jedes Schiff soll einen Namen (`name`) haben. Die Klasse `SailingShip` verfügt zudem über die Anzahl an Schiffsmasten (`mast_count`), während die Klasse `CargoShip` über die Füllmenge des Tanks (in Anzahl verbleibender Liter, `fuel_amount`) verfügt. Zu jeder Klasse gehört eine Methode `__str__(self)` welche die spezifischen Daten der Objekte als `string` zurückgibt.

(d) Schreiben Sie für die Funktionen `cargo_weight`, `add_cargo`, `remove_cargo` und `move` entsprechende Test-Funktionen mit jeweils 4 Assertions. Erstellen Sie hierzu Instanzen vom Typ `CargoShip` und `SailingShip` und beladen diese mit einigen `Cargo`-Instanzen.

Aufgabe 7.3 (Erfahrungen; Datei: `erfahrungen.txt`; Punkte: 2)

Legen Sie im Unterverzeichnis `sheet07` eine Textdatei `erfahrungen.txt` an. Notieren Sie in dieser Datei kurz Ihre Erfahrungen beim Bearbeiten der Übungsaufgaben (Probleme, benötigter Zeitaufwand nach Teilaufgabe, Bezug zur Vorlesung, Interessantes, etc.).