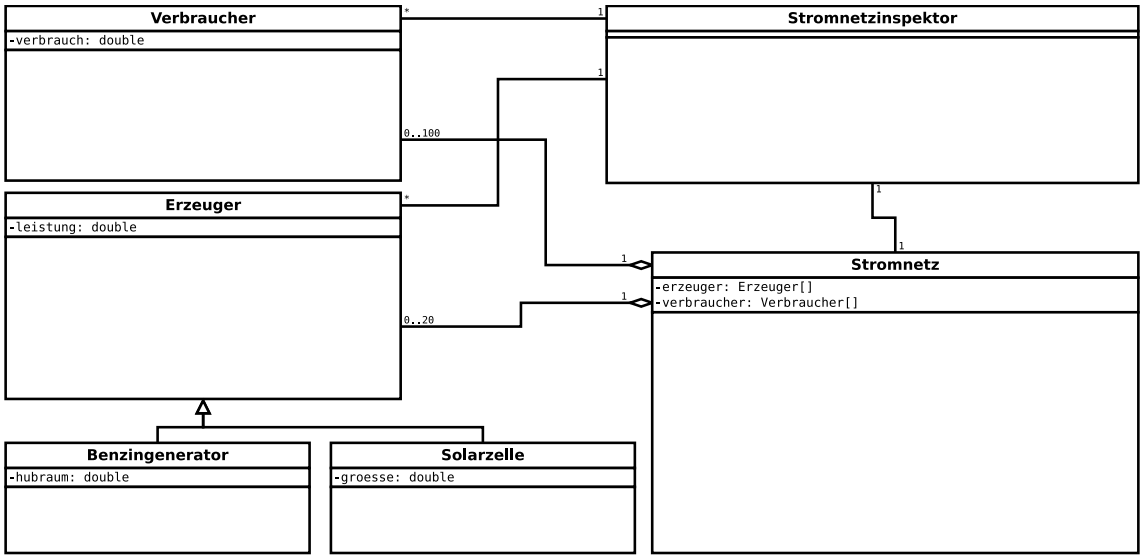
Universität Leipzig **WS 18/19**
Institut für Informatik

Modellierung und Programmierung 1 – Übungsblatt 2

Abgabetermin: 25.11.2018, 23:55 Uhr
Abgabeformat: 1 PDF-Dokument + 1 ZIP-Archiv mit Java Dateien
Max. Punkte: 35

---

1. *(10 Punkte)* UML Klassendiagramm mit Funktionalitäten: Stromnetz

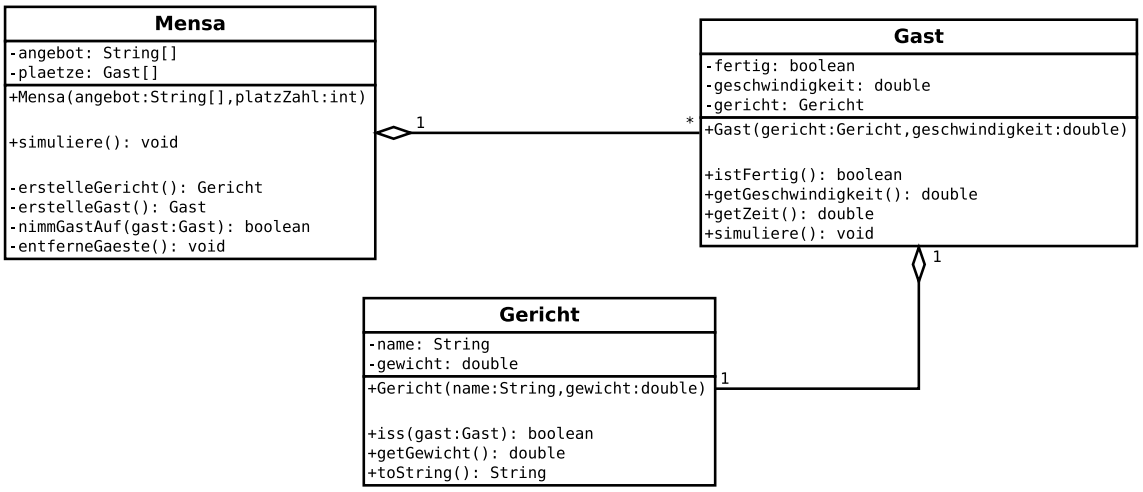


UML-Klassendiagramm

Zu modellieren ist ein Teil einer Software zur Stromnetzverwaltung mit dem oben gezeigten Klassengerüst. Im Mittelpunkt steht die Klasse `Stromnetz`, welche eine Menge von `Verbrauchern` und `Erzeugern` speichert. Zur Erstellung ist ein Stromnetz zunächst leer, hat aber Methoden um einzelne Verbraucher und Erzeuger hinzuzufügen und zu entfernen. Der hinzuzufügende oder zu entfernende Verbraucher oder Erzeuger soll jeweils als Parameter übergeben werden. Jeder Verbraucher speichert seinen Verbrauch und jeder Erzeuger seine Leistung. Beide Werte werden einmalig bei der Erstellung der Objekte gesetzt und können über entsprechende Methoden abgefragt werden. Verbraucher und Erzeuger bieten außerdem Funktionen, um sie individuell zu starten und anzuhalten. Diese sollen jeweils zurückgeben, ob der Start- oder Stopvorgang erfolgreich war. Es gibt zwei spezielle Erzeuger, den `Benzingenerator` und eine `Solarzelle`. Ihre Leistung wird über den Hubraum (cm^3) bzw. durch die Solarzellengröße (cm^2) bestimmt, welche im Konstruktor übergeben und dort zur Berechnung der Leistung genutzt werden. Ein Stromnetz muss regelmäßig inspiziert werden. Hierzu wurde der Beruf des `Stromnetzinspektor` ins Leben gerufen, welcher über eine Funktion `inspiziere` ein gegebenes Stromnetz prüfen kann. Zurückgegeben wird ein Bericht in Form eines Textes. Zur Prüfung des Netzes benötigt der Stromnetzinspektor außerdem Zugriff auf alle Verbraucher und Erzeuger, welchen er über zwei Getter-Methoden im Stromnetz selbst erhalten kann.

Ergänzen Sie das oben gezeigte Klassendiagramm um die genannten Funktionalitäten. Achten Sie dabei auf die korrekte Anzahl, Benennung und Typisierung von Parametern sowie eine vollständige Spezifikation aller Rückgabetypen. In jeder Klasse ist außerdem ein Konstruktor zu ergänzen, welcher gemäß der obigen Beschreibung in einigen Klassen auch Parameter erhält.

2. *(25 Punkte)* Zu erstellen ist eine Simulationssoftware für einen typischen Mensabetrieb. Gegeben ist das folgende Klassendiagramm:



UML-Klassendiagramm

Der grundlegende Aufbau ist wie folgt: Im Mittelpunkt steht die Klasse `Mensa`, welche ein gegebenes Essensangebot `angebot` sowie eine feste Anzahl von Sitzplätzen `plaetze` hat. Der Simulationsprozess wird in einzelnen Zeitschritten umgesetzt, welche jeweils eine Minute repräsentieren. Hierzu stellt die Mensa-Klasse eine Methode `simuliere` bereit, welche von der `main`-Funktion ein Mal für jeden Zeitschritt aufgerufen werden soll. Jeder `Gast` erhält bei der Erstellung ein Gericht, welches er zu sich nimmt, und eine Essgeschwindigkeit, welche in Gramm pro Minute angegeben ist. Für die Durchführung des Essens-Vorgangs, stellt die Klasse `Gericht` eine `iss`-Methode bereit, welche das vorhandene Gewicht entsprechend der Geschwindigkeit des gegebenen Gastes reduziert und zurückgibt, ob noch Nahrung vorhanden ist.

Erstellen Sie gemäß des Diagramms ein Gerüst aus 4 Java-Klassen.
Achten Sie dabei auf:

- korrekte Implementierung der Instanzvariablen und abgebildeten Beziehungstypen
- auftretende Multiplizitäten bei der Initialisierung von Arrays
- korrekte Implementierung der Sichtbarkeiten von Instanzvariablen und Funktionen
- Implementierung eines sinnvollen Konstruktors pro Klasse

Zudem sind folgende Funktionen zu implementieren:

a) **Gericht.java** *(4 Punkte)*

- `iss`: Führt den Essensvorgang durch, in dem das Gewicht des Gericht-Objektes um die Essgeschwindigkeit des gegebenen Gastes reduziert wird. Der Rückgabewert ist `true`, wenn nach dem Essen noch Nahrung vorhanden ist.
- `getGewicht`: Getter für das aktuell noch vorhandenen Gewichtes
- `toString`: Gibt textuelle Repräsentation in Form des Namens zurück

b) **Gast.java** *(6 Punkte)*

- `istFertig`: Gibt zurück, ob der Gast fertig ist
- `getGeschwindigkeit`: Getter für die Essgeschwindigkeit des Gastes
- `getZeit`: Berechnet die noch zu verbleibende Verweilzeit in Minuten anhand der Essgeschwindigkeit und des verbleibenden Gewichtes des Gerichtes.
- `simuliere`: Simuliert einen Zeitschritt (= eine Minute) des Gastes. Hier soll zunächst das Gericht gegessen werden und entsprechend das `fertig`-Attribut des Gastes aktualisiert werden. Ist der Gast fertig geworden, soll über die Kommandozeile die Nachricht "INFO: Gast hat Gericht aufgegessen" ausgegeben werden. Hierbei ist "Gericht" durch das entsprechende Gericht zu ersetzen.

c) **Mensa.java** *(13 Punkte)*

- `simuliere`: Simuliert einen Zeitschritt für die gesamte Mensa. Zunächst sollen unter Verwendung der `entferneGaeste`-Methode alle fertigen Gäste entfernt werden. Anschließend wird ein Simulationsschritt für alle noch vorhandenen Gäste durchgeführt. Abschließend sollen mittels `erstelleGast` und `nimmGastAuf` zufällig zwischen 5 und 20 neue Gäste in die Mensa kommen. Für jeden Gast, für den kein Platz mehr ist, soll in der Kommandozeile die Ausgabe "FEHLER: Kein Platz mehr vorhanden!" erfolgen. Für jeden erfolgreich platzierte Gast soll "INFO: Neuer Gast" ausgegeben werden.
- `erstelleGericht`: Erstellt ein zufälliges Gericht mithilfe des in der Mensa verfügbaren Angebots. Das Gewicht des Gerichtes soll zufällig im Bereich [100,200) Gramm liegen.
- `erstelleGast`: Erstellt einen zufälligen Gast mit einer Essgeschwindigkeit im Bereich [5,10) Gramm/Minute. Jeder Gast erhält außerdem ein zufälliges Gericht unter Verwendung der `erstelleGericht`-Methode.
- `nimmGastAuf`: Weist einen als Parameter gegebenen Gast einem freien Platz zu. Hierbei sollen Plätze, die im `plaetze`-Array weiter vorn liegen bevorzugt werden. Die Methode gibt zurück, ob die Platzzuweisung erfolgreich war.
- `entferneGaeste`: Entfernt alle Gäste aus dem `plaetze`-Array, die im letzten Simulationsschritt fertig geworden sind. Dies kann über die `istFertig`-Methode des Gastes geprüft werden.

(2 Punkte) Entwerfen Sie zudem eine Klasse `Main` in der Datei `Main.java`, welche die `main`-Methode enthält und die Simulation durchführt. Erstellen Sie dazu ein `Mensa`-Objekt mit 200 Sitzplätzen und folgendem Angebot:

```
String[] angebot = new String[] {
  "Veganes Gericht",
  "Nudelteller",
  "Schneller Teller",
  "Grillteller",
  "Salat",
  "Fischgericht",
  "Wok"
};
```

Anschließend soll durch wiederholtes Aufrufen der `simuliere`-Methode die Mensa für 540 Zeitschritte simuliert werden.