Aufgabenblatt #8

Wenn Sie es nicht bereits gemacht haben, lesen Sie bitte die relevanten Teile des Skriptums und erarbeiten Sie gemeinsam die bereits durchgegangen Fragen am Ende des Kapitels.

Siehe Skriptum bis inklusive Exceptions.

Universum

Schreiben Sie für jede Aufgabe auch ein Testprogramm.

Aufgabe 1: Sterne in Hauptreihe

Listing 1: Star

```
interface Star {
    void advanceTime() throws OutOfTime;
    // (3)
    // void collide(Star other)
    //      throws PhysicallyImpossible;
    // (4)
    // void accretion(Star other, int mass);
    int getSurfaceTemperature();
    int getSurfaceGravitationalAcceleration();
    int getMass();
    int getSize();
    int getIntrinsicBrightness();
}
```

Lesen Sie zuerst die gesamte Übungsaufgabe bis zum Theorieteil und betrachten Sie das Interface `Star`. Danach spezifizieren Sie das Programm in der Genauigkeit, die notwendig ist, damit Sie die Übung lösen können.

Analog zu Skriptum Seite 313.

Schreiben Sie nun die Klasse `MainSequenceStar`, welche den Lebenszyklus wie in Abbildung 1 durchlaufen kann. Dabei soll bereits bei der Konstruktion die Masse mitgegeben werden. Die Klasse soll das Interface `Star` sowie `String toString()` implementieren.

Schreiben Sie die Methode `advanceTime()` welche einen weiteren Schritt in der Zustandsmaschine macht. Ist bereits der letzte Schritt erreicht, dann wirft Sie die Exception `OutOfTime`.

Schreiben Sie Zusicherungen zur Klasse `MainSequenceStar` sowie allen in ihr vorhandenen Methoden.

Die Exceptions in dem Aufgabenblatt dienen zur Übung. Es wird nicht empfohlen eine API so zu gestalten(vgl. Aufgabe Akkretion).

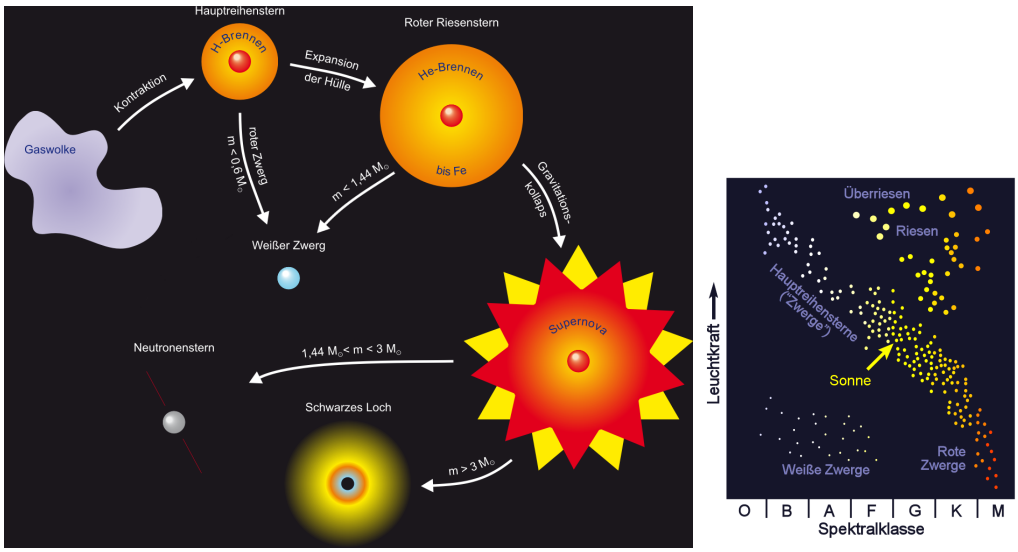

Abbildung 1: Sternentwicklung und Hertzsprung-Russell-Diagramm
Quelle: Wikipedia

Aufgabe 2: Galaxy

Schreiben Sie auch Zusicherungen zum Interface `Star`. Achten Sie darauf, dass zwischen `MainSequenceStar` und `Star` eine Untertypbeziehung erhalten bleibt.

Implementieren Sie die Klasse `Galaxy`, welche beliebig viele Sterne mit Aufrufen von `void add(Star)` hinzufügen kann. In der Galaxie soll es ebenfalls eine Methode `void advanceTime()` geben, welche alle Sterne älter werden lässt. Diese Methode soll allerdings keine Exception werfen, sondern zu alte Sterne entfernen.

Aufgabe 3: Kollisionen

Sowohl Sterne als auch Galaxien können kollidieren. Implementieren Sie die Methode `void collide(Star other)`. Dabei soll ein Stern dem anderen Masse wegnehmen. Sollte der Stern, bei dem Masse abgezogen werden soll, ein schwarzes Loch sein, dann ist die Exception `PhysicallyImpossible` zu werfen.

Dann erweitern Sie die Klasse `Galaxy` um die Methode `void collide(Galaxy galaxy)`. In diesem Fall kollidiert jeder Stern mit einer Schwerebeschleunigung von mehr als $300 m^2/s$ mit allen Sternen mit weniger als $100 m^2/s$ Schwerebeschleunigung.

Aufgabe 4: Akkretion

Akkretion ist der Vorgang, bei dem Materie aufgrund von Gravitation aufgesammelt wird. Anders als bei den Methoden `advanceTime` und `collide` wird hier keine Ausnahme geworfen, wenn der Client die Methode falsch benutzt. Schreiben Sie Zusicherungen für die Methode `accretion()`, die klar in einem Vertrag festlegen, wie diese zu benutzen ist.

Der Client soll sich darum kümmern, dass nur größere Sterne von kleinerer Sternen Masse abziehen. Der Server soll sich aber darum kümmern, dass kein Stern negative Masse bekommen kann.

Aufgabe 5: Weitere Sterne

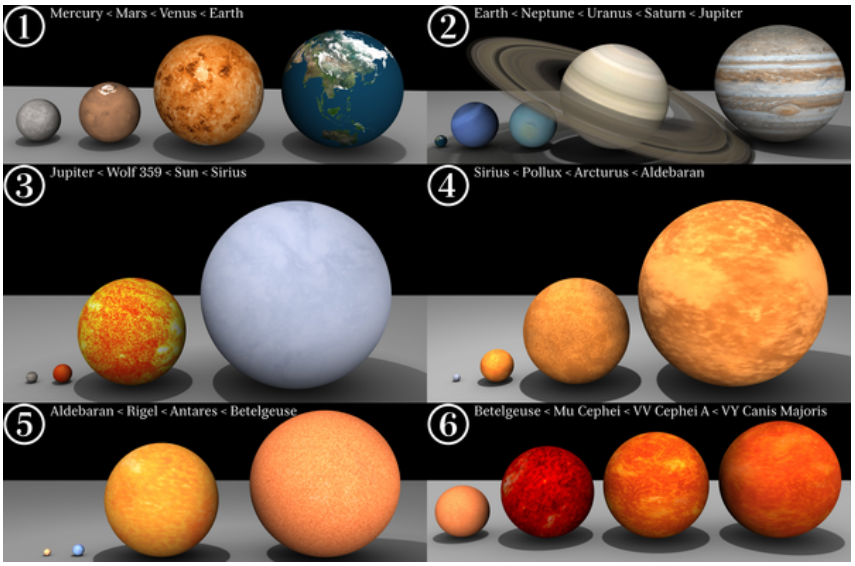

Abbildung 2: Größe anderer Sterne, Quelle: Wikipedia

Nicht jeder Stern ist in der Hauptreihe. Implementieren Sie die Klassen `Hypergiant` und `Supergiant` mit allen Methoden des Interfaces `Star`. Diese verlieren ihre Masse schnell und leben auch nur sehr kurz. Danach explodieren Sie als Supernovae.

Vergessen Sie nicht, auch für diese Methoden Zusicherungen zu schreiben. Schreiben Sie ein Testprogramm, in dem diese Sterne zu einer Galaxie hinzugefügt werden.

Theorie

Beantworten Sie die folgenden Fragen.

Aufgabe 6: Spezifikation

1. Was versteht man unter einer Spezifikation?
2. Wie genau soll eine Spezifikation sein? Von welchen Faktoren hängt diese Genauigkeit ab?
3. Auf welche Arten kann man ein Programm spezifizieren?
4. Wie geht man vor, wenn man Widersprüche oder Ungenauigkeiten in einer Spezifikation entdeckt?

Aufgabe 7: Design-by-Contract

1. Was versteht man unter Design-by-Contract?
2. Welche Bestandteile eines Softwarevertrags sind vom Client zu erfüllen, welche vom Server?
3. Welche Rolle spielen Namen in Programmen?
4. Inwiefern lassen sich Bedingungen statt als Zusicherungen auch in Form von Typen ausdrücken?

Aufgabe 8: Zusicherungen

1. Was sind Vor- und Nachbedingungen? Was sind Invarianten und History-Constraints?
2. Woran erkennt man, ob eine Bedingung ein Vorbedingung, Nachbedingung, Invariante oder History-Constraint darstellt?
3. Wie müssen sich Vor- und Nachbedingungen bzw. Invarianten und History-Constraints in Unter- und Obertypen zueinander verhalten?
4. Warum stellen Invarianten einen Grund dafür dar, dass man keine `public` Variablen verwenden sollte?

Aufgabe 9: Qualitätsverbesserung

1. Welche Maßnahmen können zu einer Qualitätsverbesserung führen?
2. Wodurch führt Testen zu einer Qualitätsverbesserung?
3. Was versteht man unter einem Code-Review?
4. Wie kann ein Code-Review ablaufen?
5. Welche Ziele, Gemeinsamkeiten und Unterschiede gibt es zwischen Unit-, Integrations-, System- und Abnahmetests.
6. Wozu verwenden wir `assert`-Anweisungen?

Aufgabe 10: Formale Methoden

1. Wodurch unterscheidet sich die partielle von der vollständigen Korrektheit eines Programms?
2. Wann terminiert eine Schleife oder Rekursion?
3. Was muss man zeigen um die Termination formal zu beweisen?
4. Was kann man mittels Model-Checking machen?
5. Kann man die Fehlerfreiheit eines Programms sicherstellen? Wenn ja, wie?