Exercise Sheet #8

If you haven't already, read the relevant parts of the course materials and answer the questions at the end of the chapter.

Book up until and including the chapter on Exceptions.

Universe

For each task, also write a testing program to test your implementation.

Aufgabe 1: Main sequence stars

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();
}
```

First, read the whole exercise sheet until the theory question section. Now, consider the `Star` interface. Specify your program as detailed as you need to solve this exercise.

As described on page 313 in the book.

Now write a class `MainSequenceStar` which can step through the individual phases of stellar evolution mentioned in figure 1. The star's mass should be set in the constructor. The class must implement the `Star` interface and the `String toString()` method.

The method `advanceTime()` should advance the star into the next stage of the state machine. If the last step has already been reached, it throws an `OutOfTime` exception.

Don't forget to write assertions for the class and all it's methods.

Note that this is just an exercise to practice the usage of exceptions. We do not recommend to design an API in this fashion(compare with exercise Accretion).

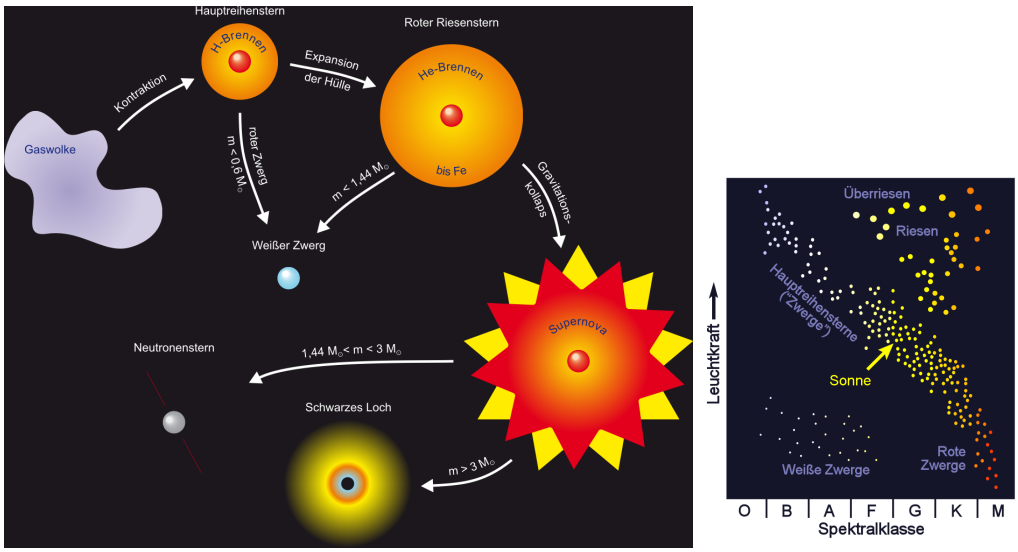

Abbildung 1: Stellar evolution and Hertzsprung-Russell diagram
Source: Wikipedia

Aufgabe 2: Galaxy

Also write assertions for the `Star` interface. Make sure that the assertions of `MainSequenceStar` and `Star` are compatible (`MainSequenceStar` must remain a subtype of `Star`).

Now, implement a class `Galaxy` that may contain any number of stars. New stars may be added to it with the `void add(Star)` method. A `Galaxy` should also provide an `advanceTime()` method, which simply advances all its stars by one phase. However, it doesn't throw an exception when stars become too old, but simply removes them.

Aufgabe 3: Collisions

Both stars and galaxies are able to collide. Add the method `void collide(Star other)` to the `Star` interface. During this collision, one of the stars absorbs the other's mass. If the absorbed star (the one that loses mass) is a black hole, the `PhysicallyImpossible` exception should be thrown.

Then, add a method `void collide(Galaxy other)` to the `Galaxy` class. During this collision, every star with a surface gravity larger than $300\frac{m^2}{s}$ collides with every star with a surface gravity less than $100\frac{m^2}{s}$

Aufgabe 4: Accretion

During accretion, a star collects additional matter via its gravitational attraction. This method (other than `advanceTime` and `collide`) should not throw an exception when it is used incorrectly. Therefore, write its assertions in a way that clearly specifies the contract for its usage.

The client must make sure that mass can only pass from smaller to larger stars via accretion. However, the server must make sure that no star can ever have negative mass.

Aufgabe 5: Other types of stars

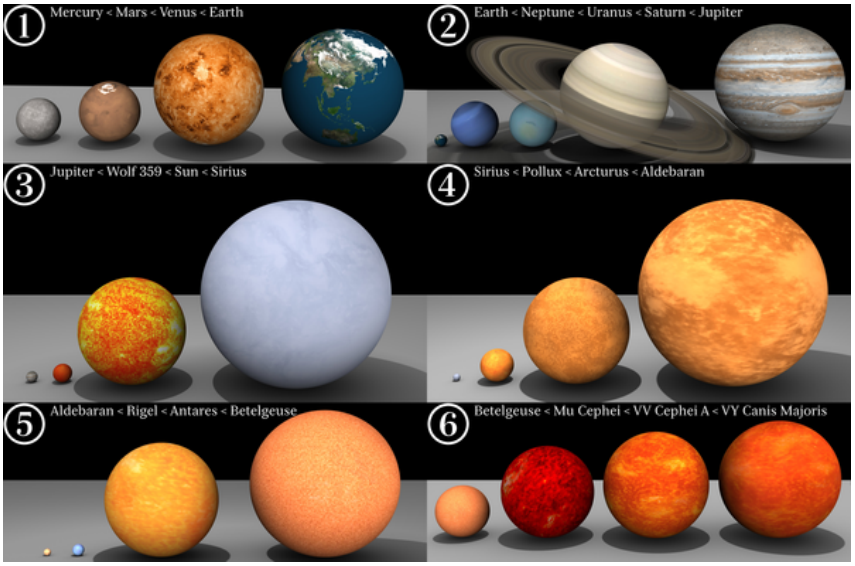

Abbildung 2: Comparison of star sizes, source: Wikipedia

Not every star is a main sequence star. Therefore, write the classes `Hypergiant` and `Supergiant`, both of which must implement the `Star` interface. Both these types of stars lose their mass very quickly and therefore have a short life. In the end, they explode in an event called Supernova.

Don't forget to write assertions for all these classes and their methods. Also write a test program, which adds such giants to a galaxy.

Theory questions

Answer the following questions.

Aufgabe 6: Specification

1. What is a specification?
2. How detailed should a specification be? What does the required level of detail depend on?
3. How can a program be specified?
4. What do we do if you find inconsistencies or inaccurate parts within our specification?

Aufgabe 7: Design by contract

1. What is design by contract?
2. Which parts of a software contract must be fulfilled by the client, and which must be fulfilled by the server?
3. What role do names play in software development?
4. How can contractual conditions be expressed as types instead of assertions?

Aufgabe 8: Conditions

1. What are pre- and postconditions? What are invariants and history constraints?
2. How can we determine the type (pre-/postcondition, invariant, history constraint) of a condition?
3. How must the individual assertion types behave between Sub- and Supertype?
4. How are invariants a reason for not using `public` variables?

Aufgabe 9: Quality assurance

1. How can we improve the quality of our software?
2. How does testing improve software quality?
3. What is a code review?
4. What are the goals of, and the similarities and differences between unit, integration, system and acceptance tests?
5. What do we use `assert` statements for?

Aufgabe 10: Formal methods

1. What is the difference between partial and total correctness?
2. When does a loop or a recursion terminate?
3. What must we show to prove termination?
4. What can we achieve with model checking?
5. Can we guarantee that a program contains no errors? If so, how?