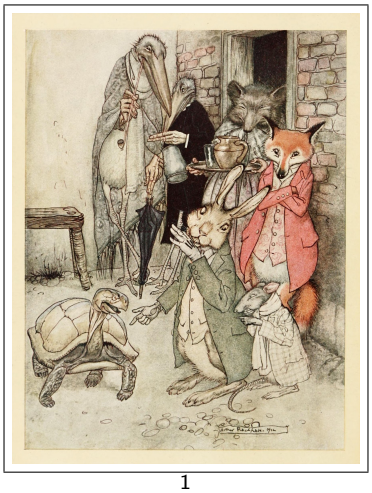
Objektorientierte Sprachen

Sommersemester 2022 - Vierte Prüfung



1

An- und Abgabe

Implementieren Sie ein Programm das die Schildkröte und den Hasen ein Rennen laufen lässt.
Sie haben dazu eine Grundstruktur vorgegeben, die bereits eine main-Funktion enthält. Erweitern Sie nun diese Grundstruktur, indem Sie eine Elternklasse und zwei Kindklassen implementieren und die main-Funktion anpassen.

Laden Sie Ihre Abgabe in einem ZIP-Archiv verpackt auf Moodle hoch, fehlerhaft oder unvollständig hochgeladene Abgaben können nicht nachgereicht werden.

Die Grundstruktur ist im Code durch Kommentare erklärt.

[1] https://de.wikipedia.org/wiki/Datei:Tortoise_and_hare_rackham.jpg

Aufgaben

Animal

- Mit folgenden **Attributen** die passende Zugriffsmodifikatoren haben:
  - Einem String für den Namen
  - Einem Integer-Wert für die Distanz zum Ziel
  - Einem booleschen Wert ob das Tier verwirrt ist
- Und folgenden **Funktionen**:
  - Einem **Konstruktor** der den Namen und die Distanz als Parameter übernimmt und setzt, die Verwirrung soll auf false gesetzt sein.
  - Einer rein virtuellen Funktion **run** die keinen Wert zurückgibt
  - Der Funktion **atGoal** die einen booleschen Wert retourniert, ob die Distanz zum Ziel kleiner als 1 ist.
  - Der Funktion **trick**, die einen Pointer auf ein Animal Objekt als Parameter erhält und keinen Wert zurückgibt.
    Die Funktion soll eine invalid_argument-Exception (mit der Meldung "Opponent NULL") werfen wenn ein Null-Pointer als Parameter übergeben wurde und eine logic_error-Exception (mit der Meldung "Opponent too far away"), wenn die Distanz zwischen dem Parameter und dem aufrufenden Objekt größer als 5 ist.
    Ansonsten soll die Funktion in $\frac{1}{6}$ der Fälle das Tier, das übergeben wurde verwirren und die Meldung "The ... got distracted" ausgeben, wobei ... der Name des Tiers sein soll, das übergeben wurde.

Rabbit

- Rabbit **erbt** von Animal und hat ein zusätzliches boolesches **Attribut** ob er schläft.
- Rabbit hat folgende **Funktionen**:
  - Einen **Konstruktor** der die Distanz und den Namen als Parameter erhält.
  - Eine Implementierung für die Funktion **run**.
    Der Hase verringert seine Distanz zum Ziel um 3, ist er dann nicht am Ziel kann es mit einer Wahrscheinlichkeit von $\frac{1}{3}$ sein, dass er einschläft. Geben Sie dann die Meldung "The Rabbit fell asleep ... from the goal" aus, ... ist die derzeitige Distanz zum Ziel.
    Schläft der Hase, wenn die Funktion aufgerufen wird hat er eine Chance von $\frac{1}{3}$ aufzuwachen, er läuft dann aber in diesem Funktionsaufruf nicht weiter. Wacht er nicht auf kann er natürlich auch nicht weiterlaufen.
    Ist der Hase verwirrt (alle Tiere können verwirrt sein), hat er eine Chance von $\frac{1}{3}$ klar zu werden, er läuft dann aber in diesem Funktionsaufruf auch nicht weiter. Bleibt er verwirrt kann er natürlich auch nicht weiterlaufen.

Turtle

- Turtle **erbt** von Animal
- Turtle hat folgende **Funktionen**:
  - Einen **Konstruktor** der die Distanz und den Namen als Parameter erhält.
  - Eine Implementierung für die Funktion **run**.
    Die Schildkröte verringert ihre Distanz zum Ziel um 1.
    Ist die Schildkröte verwirrt (alle Tiere können verwirrt sein), hat sie eine Chance von $\frac{2}{3}$ klar zu werden, sie läuft dann aber in diesem Funktionsaufruf nicht weiter. Bleibt sie verwirrt kann sie natürlich auch nicht weiterlaufen.

Race

Implementieren Sie weiters folgende Klassen **in getrennten Header- und CPP-Dateien**:

- Mit folgenden **Attributen** die passende Zugriffsmodifikatoren haben:
  - Einen Vector oder eine Liste, welcher die verschiedenen Tiere beinhaltet. Zur Laufzeit müssen die unterschiedlichen Methoden der Tiere aufgerufen werden können. (Polymorphie)
- Und folgenden **Funktionen**:
  - Eine **addAnimal** Methode ohne Rückgabewert. Die Methode fügt das übergebene Tier der internen Tierliste hinzu.
  - Der Methode **raceFinished** die einen booleschen Wert retourniert, ob eines der Tiere bereits das Ziel erreicht hat.
  - Der Methode **printResult**, die am Ende des Rennens die Namen und die Positionen (Distanz zum Ziel) der Tiere auf die Konsole ausgibt.
  - Der Methode **trickEachother**, jedes Tier übt einen Trick auf ein zufälliges anderes Tier aus der Tier-Liste aus.
  - Der Methode **competition**, die für den Ablauf des Rennens verantwortlich ist. In dieser Methode muss der Try Catch Block für den trickEachother Aufruf implementiert werden.

Benotungsaspekte

Aspekt	Bewertung
try-catch	10%
Header- und CPP-Dateien	5%
Vererbungen	5%
Attribute	5%
Konstruktoren	5%
Virtualität	5%
atGoal	5%
trick	15%
Rabbit-run	15%
Turtle-run	10%
trickEachother	5%
addAnimal	5%
printResult	5%
Speichermanagment	5%
Gesamt	100%