MCI DIE UNTERNEHMERISCHE HOCHSCHULE

Methoden der Softwareentwicklung II

SS 2020

Praktische Klausur

Freitag, 03. April 2020

Zoo

Da zur Zeit aufgrund von Covid-19 alle Zoos bis auf weiteres geschlossen sind, hat ihre Softwarefirma den Auftrag bekommen, einen virtuellen Zoo zu implementieren. Dieser virtuelle Zoo soll vor allem das Füttern der Tiere simulieren.

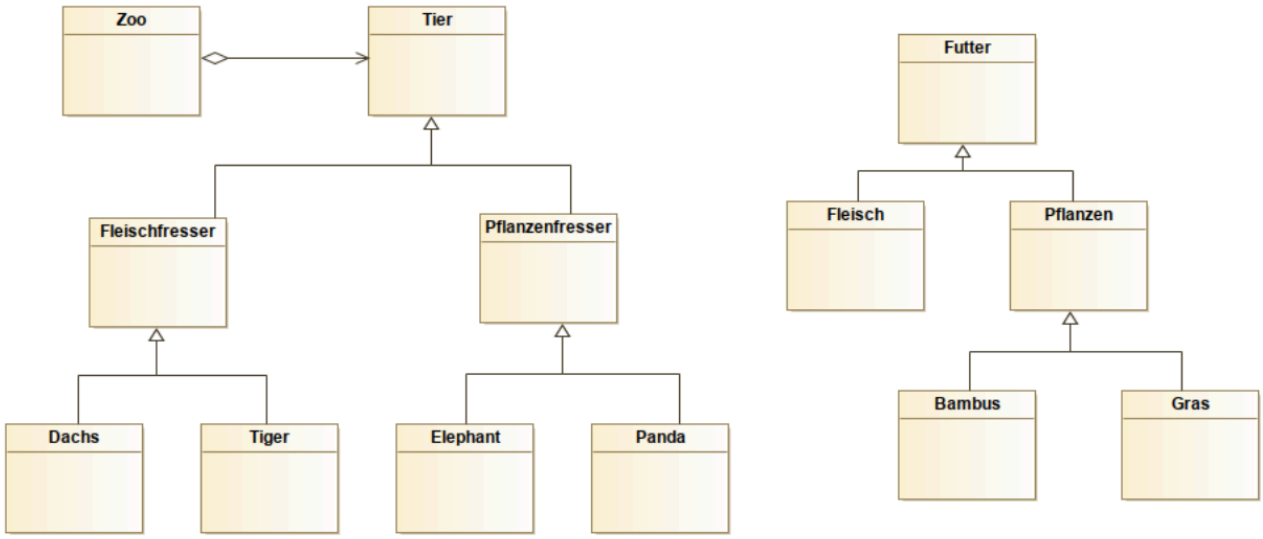
Ablauf Fütterung

Der Zoo beherbergt unterschiedliche Tiere, die unterschiedliche Arten von Futter benötigen. Da ihnen der Zoo vergessen hat mitzuteilen, was für Tier was für Futter benötigt, müssen sie die Tiere mit zufälligen Futter versorgen und die Reaktion der Tiere beobachten. Die Reaktion der Tiere wird über Exceptions mitgeteilt.

Während der Fütterungszeit wird jedes Tier mit zufällig ausgewählten Futter versort. Wenn das Futter das richtige war, wird eine `TierIstSattException` geworfen. Wenn das Futter das falsche war, wird eine `TierMagFutterNichtException` geworfen. In diesem Fall geben Sie dem Tier anderes Futter solange bis es satt ist.

Grundlegende Softwarearchitektur

Die grundlegende Software-Architektur schaut folgendermaßen aus:



Public Interface

Das im folgenden gegebene öffentliche Interface soll genauso wie angegebene implementiert werden. Sie können dieses auch um weitere Funktionen erweitern, wenn Sie dies für nötig erachten.

Klasse Zoo

Die Klasse `Zoo` repräsentiert ein globales Interface, über das die grundlegende Funktionalität des Zoos aufgerufen werden kann. Der Zoobesucher soll nur mit dieser Klasse in Berührung kommen, die ganze Komplexität des restlichen Systems wird von diese Klasse verborgen (Kapselung).

- Objektfunktionen:
  - `int addTier(Tier* sensor)` : Fügt ein neues Tier zum Zoo hinzu. Der Rückgabewert ist eine ID, die das jeweilige Tier eindeutig identifiziert.
  - `Tier* getTier(int id)` : Gibt das Tier mit der angegebenen ID zurück.
  - `void deleteTier(int id)` : Entfernt das Tier mit der angegebenen ID aus dem Zoo und gibt alle damit verbundenen Ressourcen wieder frei.
  - `void fuetterungszeit()` : Diese Funktion wird aufgerufen, wenn es Zeit ist die Tiere zu füttern. Ziel dieser Funktion ist es, alle Tiere satt zu bekommen. Für Vorgehensweise beim Füttern der Tiere siehe oben unter Punkt "Fütterung".
  - `Tier* getSchwerstenPflanzenfresser()` : Diese Funktion gibt den Pflanzenfresser mit dem größten Gewicht zurück.
  - `Tier* getSchwerstenFleischfresser()` : Diese Funktion gibt den Fleischfresser mit dem größten Gewicht zurück.

Klasse Futter

Die Klasse `Futter` bildet die Basisklasse für alle Arten von Futter. Die konkreten Futterarten sind oben unter dem Punkt "Grundlegende Softwarearchitektur" zu sehen.

- Objektfunktionen:
  - `std::string getType()` : Gibt die Art des Futters zurück ("Fleisch" oder "Pflanzen")

Die Subklasse `Pflanzen` wird nicht direkt beim Füttern verwendet, sondern nur deren Subklassen `Bambus` und `Gras` . Sie beinhaltet folgende ObjektFunktionen:

- Objektfunktionen:
  - `std::string getPflanzenart()` : Gibt die Art der Pflanze zurück ("Bambus" oder "Gras")

Klasse Tier

Die Klasse `Tier` bildet die Basisklasse für alle Tiere und definiert eine Schnittstelle, über die auf alle möglichen Arten von Tieren zugegriffen werden kann.

- Objektfunktionen:
  - `int fuettern(Futter& futter)` : Füttert das Tier mit dem angegebenen Futter. Das Futter gilt danach als verbraucht und darf nicht an ein anderes Tier verfüttert werden, sondern muss gelöscht werden.
  - `float getGewicht()` : Gibt das Gewicht des Tieres zurück (soll für jede Instanz eines Tieres ein anderes, zufällig ausgewähltes Gewicht zurückgeben, für dieselbe Instanz soll aber immer dasselbe Gewicht zurückgegeben werden).
  - `std::string getTierTyp()` : Gibt den Typ des Tieres zurück ("Pflanzenfresser" oder "Fleischfresser")
  - `std::string getTierArt()` : Gibt die Art des Tieres zurück ("Tiger", "Elephant", "Panda" oder "Dachs")

Essgewohnheiten der Tiere:

Die Tiere haben folgende Essgewohnheiten:

- Tiger: Ausschließlich Fleisch
- Elephant: Alle Pflanzenarten
- Dachs: Fleisch und Gras
- Panda: Ausschließlich Bambus

Exceptions

Die Basisklasse aller Exceptions soll `ZooException` heißen, die wiederum von `std::exception` erbt. Überschreiben Sie die Funktion `const char* what()` der Klasse `std::exception` , sodass eine aussagekräftige Fehlermeldung zurückgegeben wird (alternativ können Sie auch von `std::runtime_error` erben und die Fehlermeldung dessen Konstruktor übergeben).

Wenn Sie es für notwendig erachten, können Sie auch weitere Exception-Klassen neben den schon oben genannten hinzufügen.

Interne Implementierung

Wie Sie die gewünschte Funktionalität intern implementieren, bleibt Ihnen überlassen. Wählen Sie passende Sichtbarkeiten und vergessen Sie nicht, wenn möglich das `const` Keyword bei Eingabeparametern zu verwenden.

Überlegen Sie sich, wer die Object Ownership der Objekte des Zoos übernimmt. Wählen Sie selbstständig passende Datencontainer aus. Es darf zu keinem Zeitpunkt zu einem Ressourcenleck kommen.

Überlegen Sie sich außerdem, wo sie dynamische Bindung verwenden.

Separieren Sie bitte die `main()` Funktion in einer eigenen Datei, die Sie `main.cpp` nennen. Der Hintergrund ist, dass die automatischen Tests ihre eigene `main()` Funktion verwenden, daher muss diese einfach austauschbar sein. Der Inhalt ihrer `main()` -Funktion ist für die Prüfung nicht relevant.

Die restlichen Dateien können Sie benennen, wie Sie wollen.

Geben Sie mit der Abgabe auch ein Makefile ab, welches Ihr Programm kompiliert.

Unverbindliche Schritt-für-Schritt Anleitung

1. Beginnen Sie mit der Implementierung der Klasse `Futter` und deren Subklassen. Testen Sie diese ausführlich.
2. Implementieren Sie danach die Klasse `Tier` und deren Subklassen und testen diese ausführlich.
3. Implementieren Sie anschließend die Klasse `Zoo` und testen diese ausführlich.