C++ Übungsblatt 2: Konstruktur, Zugriffsspezifizierer

Aufgabe 2-1

Fortsetzung zu 1-2

Die Reiseagentur steht unter Druck! Soeben wurden von sämtlichen Reise-Daten die Preise auf 1€ gesetzt und dutzende Kunden buchen im Minutentakt.

Ein kurzer Scan des Systems zeigt, dass ein Mitarbeiter fälschlicherweise folgenden Programmcode geschrieben hat.

```
1  for(Trip& t : all_trips)
2  {
3      t.price = 1;
4  }
```

Die Zeilen wurden sofort entfernt. Um solche Missgeschicke zukünftig zu vermeiden, sollen die Daten, sobald sie aus der Datenbank in das Array eingespeist wurden, nicht mehr verändert werden können.
Sorgen Sie hierfür, dass die Daten einer Reise nur im Konstruktor gesetzt werden und sich fortan nicht mehr von außen ändern lassen, aber dafür immer noch abgefragt werden können.
Was passiert, wenn Sie nach Ihrem Sicherheitsupdate dennoch versuchen, Attribute eines Trips neu zuzuweisen?

Aufgabe 2-2

Im Kleinunternehmen Mathcrosoft wurde eine C++ Klasse entwickelt, die das Hantieren von Zahlen um längen vereinfacht.

```
1  class CalcBox
2  {
3  private:
4      double number;
5
6  public:
7      CalcBox() { reset(); }
8
9      void increment() { number += 1; }
10     void decrement() { number -= 1; }
11     void multiply(double other) { number *= other; }
12     void divide(double other) { if(other != 0) number /= other; }
13     void reset() { number = 0; }
14
15     double get_number() { return number; }
16 };
```

Die ersten Testentwickler sind sehr zufrieden mit diesem Produkt, allerdings wünschen sie sich eine undo-/redo-Funktion, mit der es möglich ist, Rechenoperationen rückgängig zu machen.

Hierzu soll im Konstruktor angegeben werden können, wie viele Schritte gespeichert werden sollen. Bei `new CalcBox(50)` sollen beispielsweise maximal 50 Schritte gespeichert werden, sodass nach dem 50. Undo sich nichts mehr rückgängig machen lässt.

Da diese Schritte vermutlich dynamisch angelegt werden müssen, achten Sie darauf, dass der von der Klasse allokierte Speicher am Ende wieder freigegeben wird.

Aufgabe 2-3

Erstellen Sie eine Klasse *Pet* (Haustier), das folgende Attribute besitzt und über den Konstruktor gesetzt werden können.

- Name für das Haustier
- Geschlecht
- Geburtsjahr

Zusätzlich soll mit der virtuellen Methode *speak()* der Laut des Haustiers in der Konsole ausgegeben werden. Für die Klasse *Pet* soll beim Methodenaufruf vorerst nichts passieren.

Erstellen Sie anschließend die Klassen *Dog* und *Cat*, die jeweils von *Pet* erben und die Methode *speak()* überschreiben.

Folgender Code sollte ausführbar sein (sofern sich die Klassen in den entsprechenden Header-dateien befinden).

```
 1  #include <iostream>
 2  #include <vector>
 3  #include "pet.h"
 4  #include "dog.h"
 5  #include "cat.h"
 6
 7  int main()
 8  {
 9      std::vector<Pet*> pets;
10
11      pets.push_back(new Cat("Meowto", FEMALE, 2011));
12      pets.push_back(new Dog("Doggo", MALE, 2014));
13
14      for(auto& p : pets)
15      {
16          std::cout << p->get_name() << " says: ";
17          p->speak();
18          std::cout << "\n";
19      }
20  }
```