Zugriff auf Elemente von Vektoren

Statt mit `eintraege[i]` kann man mit `eintraege.at(i)` auf das `i`-te Element eines Vektors namens `eintraege` zugreifen:

```
1  #include <iostream>
2  #include <vector>
3  int main()
4  {
5     /*Legt einen double-Vektor der Laenge 3 an, dessen Eintraege
6       alle mit 2.7 initialisiert sind: */
7     std::vector<double> eintraege(3, 2.7);
8
9     for(unsigned int i = 0; i < eintraege.size(); i++){
10       std::cout << eintraege.at(i) << std::endl;
11    }
12    return 0;
13 }
```

Vorteil von `eintraege.at(i)` gegenüber `eintraege[i]`:
Wenn `i` zu groß ist, erzeugt `eintraege.at(i)` einen kontrollierten Ausstieg, während bei `eintraege[i]` das Verhalten undefiniert ist.

Klassen

- Klassen erlauben die Zusammenfassung von Daten und Funktionen zu eigenen Objekten.
- Möglichkeit zur objektorientierten Programmierung in C++ ist der wesentliche Unterschied zu C.
- Mit Klassen lassen sich eigene Datentypen erzeugen, die wie elementare Datentypen (`int`, `double` etc) benutzt werden können.
- Ein Beispiel für eine Klasse ist `std::vector`.

Klassen

Syntax:

```
class KLASSENNAME {
public:
// Eintraege, die von aussen sichtbar sein sollen
// (insbesondere ein oder mehrere Konstrukoren)
private:
// Eintraege, die nicht von aussen sichtbar sein sollen
};
```

Klassen

Syntax für den Zugriff auf einen Klasseneintrag eines Objekts:

`OBJEKTNAME.EINTRAGNAME`

EINTRAGNAME kann dabei auch eine Funktion sein.

Bekanntes Beispiel:

```
std::vector<int> zahlen;
std::cout « zahlen.size();
/* Zugriff auf die Funktion size()*/
```

Konstruktoren

- Funktionen, die so heißen wie die Klasse und aufgerufen werden, sobald ein Objekt vom Klassentyp erzeugt wird.
- Konstruktoren dienen der Initialisierung der Daten und eventuell der Speicherbeschaffung.
- Es kann verschiedene Konstruktoren geben, die sich in ihren Schnittstellen unterscheiden.