Typisierung in Programmiersprachen

Seminar „Effiziente Programmierung“

Darwin Willers

Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

17.12.2020

Gliederung

- Typisierung
- Statisch & Dynamisch
  - Polymorphie
    - Upcast
    - Downcas
- Stark & Schwach
  - Typsicherheit
  - Speichersicherheit
- Optionale Typisierung
- Explizite & Implizite Deklarierung

Was ist Typisierung?

Komponenten

- Definition und Zuordnung von Typen für Objekte
- Regeln für den Umgang mit Typen

Nutzen

- Korrekte Nutzung von Komponenten
- Vermeidung von Laufzeitfehlern

Typfehler Beispiel

```
1 zahl = 42 // 4-Bytes reserviert
2 Console.write("Wie ist dein Name?")
3 // Wert durch die Konsole einlesen
4 zahl = Console.read() // "Albert" ~ 12-Bytes
```

Statische Typisierung

- Typen werden zur Entwicklungszeit festgelegt
- Typfehler werden zur Übersetzungszeit festgestellt

```
1 int zahl = 42
2 Console.write("Wie ist dein Name?")
3 // Wert durch die Konsole einlesen
4 zahl = Console.readString() // Fehler zur
      Uebersetzungszeit
```

Dynamische Typisierung

- Typen können sich dynamisch ändern.
- Es gibt keine Typfehler zur Übersetzungszeit

```
1 zahl = 42
2 Console.write("Wie ist dein Name?")
3 // Wert durch die Konsole einlesen
4 zahl = Console.read() // "Albert" => Kein Problem
```

Dynamische Typfehler

```
1 x = 4
2 y = 3
3 Console.write(x + y) // Ausgabe => "7"
4 x = "Hallo"
5 Console.write(x + y) // Typ-Fehler
```

Dynamische-Fehler behandeln

```
1 while(true) {
2     try {
3         x = Console.read("Welche Zahl soll quardriert
    werden?")
4         x2 = x * x // moeglicher Typ-Fehler
5         Console.write("{x} quadriert ist {x2}.")
6     } catch (TypeError e) {
7         Console.write("{x} ist leider keine Zahl,
    probier es nochmal.")
8     }
9 }
```

Kombinationen

- Basis Dynamisch + optional Statisch
- Basis Statisch + optional Dynamisch

Polymorphie

- Verwandtschaft von Klassen miteinander
- Abstraktion geteilter Eigenschaften in Super-Klassen

```
1 class Human {
2     public List<Pet> pets;
3 }
4
5 class Pet
6
7 class Cat extends Pet
8
9 class Dog extends Pet
```

Upcast

- Das umwandeln von einem spezielleren zu einem allgemeinerem Typ.

```
1 Cat c = new Cat()
2 Dog d = new Dog()
3 Human h = new Human()
4
5 h.pets.push(c)
6 h.pets.push(d)
```

Downcast

- Das umwandeln von einem allgemeineren zu einem speziellern Typ.

```
1 define PlayWithPet (Pet pet) {
2     if (pet is Cat) {
3         Cat c = pet as Cat
4         // ... was mit c machen
5     }
6 }
```

Starke & Schwache Typisierung

Starke und Schwache Typisierung ist nicht fest definiert, sondern eher eine Skala.

Faktoren sind:

- Typ-Sicherheit
- Speicher-Sicherheit

Typ-Sicherheit

- Typen werden nur gemäß der der Regeln genutzt
- Implizite Typumwandlung sind nur ohne Datenverlust erlaubt

```
1 float f = 1.6
2 int i = f
3 float sum = 1 + f
```

Typ-Sicherheit

- Typen werden nur gemäß der Regeln genutzt
- Implizite Typumwandlung sind nur ohne Datenverlust erlaubt
- eher Eigenschaft des Codes als der Sprache

```
1 float f = 1.6
2 int i = f       // Nicht Typ-sicher, da Verlust der
      Nachkommastellen
3 float sum = 1 + f // In diesem Fall Typ-sicher
```

Randwert Betrachtung

```
1 // int und float werden beide mit 32 bit dargestellt.
2 Console.write(int.MAX)       //  => 2147483647
3 float x = 2147483647         // => x = 2147483648
4 /*
5     Abweichung hier zufaellig nur um 1.
6     Naechst kleinere floatwert ohne Nachkommastelle
      ist: 2147483520
7     => Schrittweite 128
8 /*
```

Speicher-Sicherheit

- Freigabe und Reservierung von Speicher
- Überprüfung der Array-Grenzen
- Keine Zeiger-Arithmetik

Sicherheit

```
1 var x = "22"
2 var y =  5
3 var z = x + y
```

Sicherheit

```
var x = "22"
var y =  5
var z = x + y
/*
Moegliche Ergebnisse...
-----------------------------------------
    Sichere Sprachen:
        z = 27
        z = "225"
        Ein Typ-Fehler wird geworfen
-----------------------------------------
    Nicht sichere Sprachen:
        Ein Zeiger auf eine andere Stelle
        Skurrile Werte
-----------------------------------------
*/
```

Optionale Typisierung

- Typisierung und Programmiersprache sollten unabhängig sein
- Performance und Sicherheit austarieren

Explizite Deklarierung

- Typen werden immer ausdrücklich zugewiesen

```
1  // Increase x by one
2  define int Increment(int x) {
3      int result
4      result = x + 1
5      return result
6   }
7
8  // Increase x by one
9  define float Increment(float x) {
10     var result = x + 1
11     return result
12  }
```

Implizite Deklarierung

- Der Compiler weist die Typen anhand des Kontexts zu

```
1  // Integer
2  define Increment(x) {
3      result = x + 1
4      return result
5  }
6
7  // Float
8  define Increment(x) {
9      result = x + 1.0
10     return result
11 }
```

Implizite Deklarierung

```
1  // Integer Addition
2  define Add(x, y) {
3      int result = x + y
4      return result
5  }
6
7  // Float Addition
8  define Add(float x, y) {
9      result = x + y
10     return result
11 }
12
13 //String Addition
14 define str Add(x, y) {
15     result = x + y
16     return result
17 }
```

Implizite Deklarierung

```
1 define Add (x, y) {
2     result = x + y
3     return result
4 }
```

Zusammenfassung

- Typisierung - Typen und Regeln zum Umgang
- Statisch - Überprüfung zur Kompilierzeit
- Dynamisch - Überprüfung zur Laufzeit
- Typsicherheit - der richtige Umgang mit Typen
- Speichersicherheit - Kein direkter Speicher-zugriff
- Optionale Typisierung - Typisierung nach nutzen
- Explizite & Implizite Deklarierung

Literatur

[Hic14a] Michael Hicks. What is memory safety? http://www.pl-enthusiast.net/2014/07/21/memory-safety/, 2014. Accessed: 05.12.2020.

[Hic14b] Michael Hicks. What is type safety? http://www.pl-enthusiast.net/2014/08/05/type-safety/, 2014. Accessed: 05.12.2020.

[Pie03] B. C. Pierce. Types and programming languages: the next generation. In *18th Annual IEEE Symposium of Logic in Computer Science, 2003. Proceedings.*, page 32, 2003.

[SPWS13] L. Szekeres, M. Payer, T. Wei, and D. Song. Sok: Eternal war in memory. In *2013 IEEE Symposium on Security and Privacy*, pages 48–62, 2013.

[Wik] Type-Systems. https://en.wikipedia.org/wiki/Type_system. Accessed: 02.12.2020.