Programmierparadigmen

Programmierparadigmen

Imperative Programmierung

Bei der imperativen Programmierung werden Programme als aufeinander folgende Befehle formuliert. Die Befehle verändern während der Programmausführung in Variablen gespeicherte Werte und können ermitteln so Berechnungsergebnisse.

Imperative Programmierung ist ein Programmierparadigma und wird beispielsweise in der prozeduralen Programmierung und in der objektorientierten Programmierung umgesetzt.

Prozedurale Programmierung (imperativ)

Bei der prozeduralen Programmierung wird die Gesamtaufgabe, die eine Software lösen soll, in kleinere Teilaufgaben aufgelöst. Jede Teilaufgabe für sich ist einfacher zu beschreiben, programmieren und testen. Außerdem kann der entstehende Programmcode in anderen Programmen wieder verwendet werden, ein sehr wichtiger Aspekt in der Softwaretechnik.

Die bei der Aufgabenlösung entstehenden Programm-Module werden Prozeduren bzw. Funktionen genannt, wobei zu beachten ist das prozedurale Programmierung keineswegs mit funktionaler Programmierung gleichzusetzen ist - beide folgen unterschiedlichen Programmierparadigmen.

Prozedurale Programmierung (imperativ)

Prozeduren und Funktionen werden üblicherweise nach Aufgabengebieten gruppiert zu Bibliotheken zusammengefasst, die dann verteilt und in beliebig viele andere Programme eingebunden werden können.

Typische prozedurale Programmiersprachen sind beispielsweise Pascal, die Programmiersprache C und BASIC; alle bereits recht betagt. Modernere Programmiersprachen wie Java und C# betrachten den prozeduralen Ansatz als veraltet und setzen stattdessen auf die Weiterentwicklung zur objektorientierten Programmierung.

Objektorientierte Programmierung (imperativ)

Die objektorientierte Programmierung (OOP) ist eine Methode zur Modularisierung von Programmen, die sich stark von der klassischen prozeduralen Programmierung unterscheidet. Objektorientierte Software ist, wenn sie gut entworfen wurde, leichter zu warten und zu erweitern als prozedurale. Zudem vereinfacht sie durch die strenge Modularisierung Unit-Tests und Wiederverwendung von Softwareteilen. Sie folgt dem Programmierparadigma der imperativen Programmierung.

Objektorientierte Programmierung (imperativ)

- Einheiten, die Objekte genannt werden
- Jedes Objekt besitzt einen Zustand - Eigenschaften (Objektattribute)
- Nur die im Objekt selbst vorhandenen Funktionen (Methoden genannt), können dessen Daten manipulieren und so den Zustand verändern.
- Botschaften senden (Methoden aufrufen)
- Daten und Funktionen zu Objekten zusammengefasst

Objektorientierte Programmierung (imperativ)

- Klassen - Baupläne für Objekte
- Jedes Objekt ist eine Instanz seiner Klasse
- Datenkapselung
- Austauschbarkeit
- Vererbung
- Polymorphie

Deklarative Programmierung

Bei der deklarativen Programmierung, auch als Non-procedurale Language (NPL) bezeichnet, wird das ältere Paradigma der imperativen Programmierung umgekehrt. Das Programm beschreibt nicht mehr länger, was getan werden soll, sondern nur noch welches Ergebnis am Ende stehen soll. Es bleibt dann dem Programm überlassen, sich mit Hilfe entsprechender Algorithmen den korrekten Lösungsweg zu suchen. Programmierparadigmen die diesem Prinzip folgen sind die funktionale Programmierung und die logische Programmierung.

Funktionale Programmierung (deklarativ)

Funktionale Programmierung ist ein deklaratives Programmierparadigma, bei dem Programme als mathematische Funktionen formuliert werden. Während in prozeduralen Programmiersprachen nacheinander Befehle ausgeführt und auf diese Weise die Inhalte von Variablen verändert werden, hat in der funktionalen Programmierung ein Ausdruck während der Laufzeit immer den gleichen Wert. Diese Eigenschaft ist für einige akademische Anforderungen wie beispielsweise Beweisführungen sehr hilfreich.
Man unterscheidet rein funktionale Programmiersprachen (z.B. Haskell und Miranda) von Programmiersprachen, die das Programmierparadigma zwar aufnehmen aber mit imperativen Sprachelementen vermischen (z.B. Scheme und Tcl).

logische Programmierung (Prädikative Programmierung)

- mathematische Logik - Prädikatenlogik
- Menge von Axiomen (Fakten, Annahmen)
- Regeln
- Interpreter sucht aufgrund von Axiomen und Regeln (Logik) die Lösung
- Künstliche Intelligenz
- Prolog

logische Programmierung - Beispiel Prolog

Logischer Schluss:

Sokrates ist ein Mensch.
Alle Menschen sind sterblich.
⇒ Sokrates ist sterblich.

```
mensch(sokrates).
sterblich(X) :- mensch(X).

?- sterblich(sokrates).
   ⇒ yes

?- sterblich(X).
   ⇒ X = sokrates
```

logische Programmierung - Beispiel Prolog

```
mensch(sokrates).
mensch(aristoteles).
sterblich(X) :- mensch(X).

?- sterblich(X).
```

⇒ *X = sokrates; X = aristoteles; no*

Strukturierte Programmierung

Die strukturierte Programmierung ist ein Verfahren für die Programmentwicklung, das von einer Programmstruktur mit einheitlichen und selbständigen Programm-Bestandteilen ausgeht und die Bedingungen beschreibt, unter denen sie zusammenwirken.
Das Verfahren wurde von Nassi-Shneiderman entwickelt und gehört seit den 60er-Jahren zu den stärksten Anstößen für die produktive Programmentwicklung. Es machte das Programmieren bis zur Fehleraufdeckung hin einfacher und sicherer; außerdem bildete es die Grundlagen für die Modularisierung der Programme und für eine Programmstruktur, die das gezielte Eingehen auf die unterschiedlichen Anforderungen in der systemtechnischen Ebene und Benutzerebene ermöglicht.

Aspektorientierte Programmierung

Als Aspekte bezeichnet man in der Programmierung Komponenten-übergreifende Zusammenhänge. Die aspektorientierte Programmierung (AOP) führt Aspekte als eigene syntaktische Strukturen ein und erhöht damit die Modularität von objektorientierten (OO) Programmen bezogen auf diese Aspekte.

As·pẹkt

Substantiv [der]

1. gehoben
Gesichtspunkt.
"etwas unter einem anderen Aspekt sehen"

2. SPRACHWISSENSCHAFT
eine grammatische Kategorie des Verbs.
"der vollendete/unvollendete Aspekt"

Aspektorientierte Programmierung - zwei Bereiche

1. Die sogenannten Core-Level-Concerns (betreffen den logischen „Kern“ der Anwendung) oder funktionale Anforderungen. Dies sind Anforderungen an die Software, die man meist gut in einzelnen Funktionen kapseln kann. Ein Beispiel wäre die Berechnung eines Wertes.

Aspektorientierte Programmierung - zwei Bereiche

2. Die System-Level-Concerns (betreffen das gesamte System) oder technische Randbedingungen. Diese Anforderungen können nicht einfach gekapselt werden, da sie an vielen Stellen implementiert werden müssen. Ein Paradebeispiel dafür ist das Logging, die Protokollierung des Programmablaufs in Logdateien. Der Aufruf des Loggers ist für die eigentliche Funktionalität nicht notwendig, muss aber trotzdem in den Quelltext integriert werden. Ein weiteres Beispiel wäre die Transaktionierung von Zugriffen auf eine Ressource wie z. B. eine Datenbank.

Aspektorientierte Programmierung

Das Problem der miteinander verwobenen Anforderungen wird auch als Cross-Cutting Concerns bezeichnet, denn sie „schneiden“ quer durch alle logischen Schichten des Systems. AOP ist das Werkzeug, um die logisch unabhängigen Belange auch physisch voneinander zu trennen. Dabei wird angestrebt, Code zu erzeugen, der besser wartbar und wiederverwendbar ist.

Aspektorientierte Programmierung - Beispiel Tracing

```
public void eineMethode() {
    logger.trace("Betrete \"eineMethode\"");

    // Abarbeitung der Methode
    m = a + 2;

    logger.trace("Verlasse \"eineMethode\"");
}
```

Aspektorientierte Programmierung - Beispiel Tracing

Aspekt:

```
public aspect Tracing {
   pointcut traceCall():
        call(* AOPDemo.*(..));
   before(): traceCall() {
       System.out.println("Betrete \"" + thisJoinPoint + "\"");
   }
   after(): traceCall() {
       System.out.println("Verlasse \"" + thisJoinPoint + "\"");
   }
}
```

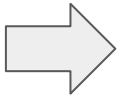
Aspektorientierte Programmierung - Beispiel Tracing

Methoden werden extrem vereinfacht!

```
public void eineMethode() {
   // Abarbeitung der Methode
   m = a + 2;
}
```

Generische Programmierung (Templates)

```
int abs(int i) {
    return i < 0 ? -i : i;
}
float abs(float r) {
    return r < 0 ? -r : r;
}
double abs(double d) {
    return d < 0 ? -d : d;
}
```



```
template<typename T>
T abs(T x) {
    return x < 0 ? -x : x;
}
```

Generative Programmierung - automatisch erzeugter Code

1. Der Programmierung eines bestimmten Programmgenerators
2. Der Parametrierung oder Ergänzung und Konfiguration des formalen Modells auf eine spezifische Modellausprägung
3. Dem Aufruf des Programmgenerators mit den spezifischen Inputparametern, welcher dann das spezifische Zielprogramm erstellt

Beispiele:

- UML
- Compiler-Compiler