Richtlinien für sauberen Code

Eine zentrale Aufgabe beim Programmieren ist es, korrekten, sauberen und einfach gestalteten Programmcode abzuliefern. Dafür gibt es in der Praxis zahlreiche Gründe:

- **Schnellere Fehlerkorrektur:** Auftretende Fehler sind einfacher zu finden und können schneller beseitigt werden.
- **Einfachere Anpassungen:** Änderungen in den Anforderungen der Software sind zügiger im Code umsetzbar.
- **Bessere Erweiterungsfähigkeit:** Zusätzliche Komponenten und Module sind mit geringerem Aufwand zu integrieren.
- **Höhere Qualität:** Anpassungen im Code führen an anderen Stellen zu deutlich weniger Folgefehler.
- **Verbesserte Teamarbeit:** Code ist für neue Teammitglieder schneller zu verstehen und anpassbar.
- **Geringerer Dokumentationsbedarf:** Verständlicher Code benötigt wenige Aufwand für die Dokumentation des Projektes.
- **Höhere Produktivität:** Strukturierter Code führt zu geringeren Entwicklungszeiten und niedrigeren Kosten.

Die folgenden Codierungsrichtlinien sollen Sie in die Lage versetzen, dass Sie erstens guten Code von schlechtem Code unterscheiden und zweitens schlechten Code in guten Code umwandeln können.

Regel 1: Geeignete Programmierumgebung
Benutzen Sie in Abhängigkeit der gegebenen Aufgabe eine geeignete Programmiersprache und eine benutzerfreundliche Programmierumgebung. Die Entwicklungsumgebung sollte unter Berücksichtigung der Kosten, Problemkomplexität, Geschwindigkeit und des Anwenderkreises ausgewählt werden. Verwenden Sie erprobte Komponenten und zuverlässige Softwarebibliotheken, um effizient sichtbare Ergebnisse zu erzielen.

Regel 2: Groß- und Kleinschreibung beachten
Schreiben Sie Variablen- und Methodennamen klein und Klassennamen mit einem Großbuchstaben. Für die bessere Lesbarkeit sollten Sie alle Wortanfänge im Namen groß schreiben. Konstanten bestehen in der Regel vollständig aus Großbuchstaben.

Regel 3: Aussagekräftige Variablennamen
Benennen Sie Variablen, Methoden und Klassen mit konsistenten, aussagekräftigen, aussprechbaren und unterscheidbaren Namen. Sinngebende Namen sollten angeben, was gemeint ist und in welcher Einheit gemessen wird (z.B. `zeitInSekunde`). Verwenden Sie kurze Verben oder Verben plus Substantive für Methodennamen (z.B. `schreibeDaten()`) und prägnante (beschreibende) Substantive für Klassennamen (z.B. `FirmenKonto`). Laufvariablen in Schleifen sollten aussagefähig sein, anstatt nur die Namen `i,j,k` zu benutzen. Variablen mit einem einzigen Buchstaben sind nur als lokale Variablen in kurzen Methoden zu verwenden. Irreführende Namen sind der Kleinbuchstabe `l` und der Großbuchstabe `O`, die wie eins und null aussehen. Vermeiden Sie redundante Leerwörter, wie `variable` in einem Variablennamen oder `Object` in einem Klassennamen. Empfehlenswert ist eine einheitliche Konvention für die unterschiedlichen Datenarten (z.B. Skalare, ein- und mehrdimensionale Arrays, dynamische Datenlisten). Verwirrend ist die Bezeichnung von Elementarzahlen mit `anz, n, m`, besser ist ein kurzes beschreibendes Wort mit einem sinnvollen Anhang, wie beispielsweise zahl (z.B. `knotenzahl`, `kantenzahl`).

Regel 4: Übersichtliche Klammersetzung
Von sehr großer Bedeutung ist die vernünftige Einrückung des Programmtextes. Schreiben Sie öffnende und schließende Klammern in einem Codeblock zur besseren Lesbarkeit untereinander. Durch die korrekte Einrückung des Codes sparen Sie viel Zeit bei der Suche nach vermeidbaren Fehlern.

Regel 5: Fehleranfällige Konstrukte vermeiden
Verwenden Sie in ihrem Programmcode keine genialen Programmiertricks, die nur sehr schwer nachzuvollziehen sind. Fehleranfällige Konstrukte, wie beispielsweise unbedingte Sprunganweisungen oder Zeiger, sind wenn möglichst zu umgehen. Gestalten Sie logische Aussagen ohne Negationen auf die einfachste Art und Weise. Stark verschachtelte Kontrollanweisungen sind zu vermeiden, da diese schwer zu testen und zu verstehen sind.

Regel 6: Leerstellen und Leerzeilen einfügen
Für die bessere Lesbarkeit sollten Sie jede Anweisung in eine neue Zeile schreiben. Jede Zeile im Code stellt einen Ausdruck und jede Gruppe von Zeilen einen vollständigen Gedanken dar. Wie Absätze in Artikeln sollten Sie diese durch eine Leerzeile trennen. Verwenden Sie Leerstellen, um Anweisungen übersichtlicher zu gestalten, beispielsweise mit einem Leerzeichen vor und nach dem Gleichheitszeichen. Trennen Sie Elemente die nicht zusammengehören mit einer Leerstelle, wie beispielsweise Argumente innerhalb eines Methodenaufrufs. Sinnvoll ist es, mehrere hintereinanderliegende Zuweisungen ausgerichtet untereinander zu schreiben.

Regel 7: Werte mit Variablen anlegen
Viele Programme enthalten Codezeilen, bei der feste Zahlengrößen miteinander verrechnet werden. Diese Werte können sich im Laufe der Zeit durch neue Anforderungen verändern. Legen Sie unbedingt für jeden Wert in einem Programm eine eigene Variable an, damit Sie später keine Zeit zum Suchen und Auswechseln der Werte verschwenden müssen. Mehrfach verwendete Werte sind außerdem schwer zu finden, sodass vergessene Änderungen den Programmcode fehlerhaft machen. Definieren Sie die notwendigen Variablen eng bei dem Ort der Verwendung.

Regel 8: Eine Aufgabe pro Methode
Vermeiden Sie sehr große und unstrukturierte Programme, da diese unübersichtlich und schwer wartbar sind. Sie erhalten keinen guten Code, wenn Sie nur Unmengen von Anweisungen aneinanderreihen. Die wichtigste Aufgabe beim Programmieren besteht darin, Aufgaben in kleine Teilaufgaben zu zerlegen. Schreiben Sie für jede dieser einzelnen Aufgabe eine Methode (ca. 20-100 Zeilen).

Regel 9: Duplizierende Codezeilen sind verboten
Vermeiden Sie unbedingt beim Programmieren das kopieren und duplizieren von Codezeilen. Unterteilen Sie wiederholende Teilaufgaben in passende Hilfsmethoden mit geeigneten Übergabeparametern. Das Hauptprogramm sollte so weit wie möglich nur die einzelnen Unterprogramme aufrufen.

Regel 10: Geringe Anzahl von Übergabeargumenten
Die Anzahl von Argumenten in Methoden sollte so gering wie möglich sein, um aufwendige Testfälle zu umgehen. Vermeiden Sie Methoden mit mehr als drei Argumenten, sodass sie ggf. Argumente zu Instanzvariablen befördern. Wenn eine Methode ein Eingabeargument transformiert, sollte das Ergebnis der Rückgabewert darstellen. Verwenden Sie möglichst keine verkomplizierenden Flag-Argumente mit `true` oder `false`, sondern teilen Sie die Methode in zwei separate auf.

Regel 11: Strukturierter Aufbau von Quelldateien
Der Code in einer Quelldatei sollte wie eine Erzählung von oben nach unten lesbar sein. Schreiben Sie zusammengehörige Fakten stets enge beieinander. Im oberen Teil der Datei sollten die Instanzvariablen und die wichtigsten Konzepte stehen. Die Detailtiefe nimmt nach unten hin zu, wobei am Ende die Hilfsmethoden stehen. Sinnvoll ist es hinter jeder Methode die Methode auf der nächsttieferen Abstraktionsebene zu schreiben. Die aufrufende Methode sollte möglichst über der aufgerufenen Methode stehen. Typischerweise sollte die Größe einer Datei nicht 500 Zeilen überschreiten.

Regel 12: Objektorientiert programmmieren
In der objektorientierten Programmierung werden Programme durch eine Menge von interagierenden Elementen erstellt. Fassen Sie zusammengehörige Daten und die darauf arbeitende Programmlogik in eine Klasse zusammen. Schränken Sie die Sichtbarkeit von Variablen ein, sodass keine Fehlanwendungen möglich sind. Erstellen Sie passende Schnittstellen mit Getter- und Settermethoden für die Rückgabe und Veränderungen einzelner Variablen. Benutzen Sie geeignete Programmierparadigmen und Entwurfsmuster zur Modellierung von flexiblen und wiederverwendbaren Klassen.

Regel 13: Jedes Objekt eine Klasse
Jede Klasse sollte nur eine Verantwortlichkeit und nur einen einzigen Grund zur Änderung besitzen. Teilen Sie eine Klasse auf, wenn diese mehrere Verantwortlichkeiten hat, oder gewisse Methoden nur bestimmte Variablen benutzen. Erstellen Sie eine Klasse in der Form, dass diese möglichst mit wenigen anderen Klassen zusammenarbeitet, um das gewünschte Verhalten zu erreichen. Jede Klasse sollte eine überschaubare Anzahl von Instanzvariablen besitzen.

Regel 14: Angemessene Kommentierung
Für die Verständlichkeit des Codes muss dieser ausreichend und einheitlich kommentiert sein. Ausdrucksfähiger Code mit wenigen Kommentaren ist besser, als komplizierter Code mit vielen Kommentaren. Kommentieren Sie keinen schlechten Code, sondern schreiben sie diesen um. Benutzen Sie für Kommentare eine korrekte Grammatik mit sorgfältig gewählten Wörtern. Eine gute Kommentierung hat die Aufgabe zu informieren, Absichten zu erklären, Bestandteile zu unterstreichen, vor Konsequenzen zu warnen oder ToDo-Vermerke zu erstellen. Vermeiden Sie redundante, irreführende oder vorgeschriebene Kommentare mit unnötigen Informationen. Auskommentierter Code führt zu unnötiger Verwirrung und ist zu entfernen.

Regel 15: Teamregeln festlegen
Ein Softwaresystem besteht aus einem Satz von ähnlich aufgebauten Quelldateien mit gleichen Formatierungsregeln. Wenn Sie in einem Team programmieren, legen Sie die zentralen Codierungsregeln für das Team fest: Klammersetzung, Größe der Einrückungen, Bezeichnung der Klassen, Methoden, usw. Jedes Mitglied des Teams sollte dann genau diesen Stil benutzen, sodass der gesamte Code konsistent ist.

Literaturhinweise

[1] R.C. Martin, *Clean Code - Refactoring, Patterns, Testen und Techniken für sauberen Code*, mitp, 2009.