# Heuristiken für sauberen Code (clean code)

## Einleitung

Eine Heuristik ist eine Vorgehensweise, welche auf Erfahrung basiert und die man für die Lösung von neuen Problemen anwenden kann. Dabei gibt es aber keine Garantie, dass die Heuristik wirklich zum optimalen Resultat führt.

Im Buch „Clean Code“ von Robert C. Martin, (Prentice Hall, 2009) wird eine Liste von Heuristiken präsentiert, welche angewendet werden können, um möglichst sauberen Code zu schreiben. Wikipedia (<https://de.wikipedia.org/wiki/Clean_Code>) definiert sauberen Code wie folgt:

„Als „sauber“ bezeichnen Software-Entwickler in erster Linie Quellcode, aber auch Dokumente, Konzepte, Regeln und Verfahren, die intuitiv **verständlich** sind. Als intuitiv verständlich gilt alles, was mit wenig Aufwand und in kurzer Zeit richtig verstanden werden kann. Vorteile von Clean Code sind stabilere und effizient wartbarere Programme, d. h. kürzere Entwicklungszeiten bei Funktionserweiterung und Fehlerbehebungen. Die Bedeutung wächst mit der Beobachtung, dass im Schnitt 80 % der Lebensdauer einer Software auf den Wartungszeitraum entfällt.“

Im Folgenden finden Sie eine Auswahl von Heuristiken für sauberen Code (in Klammer stehen die Verweise auf die Beschreibungen im erwähnten Buch). Verschaffen Sie sich einen Überblick und versuchen Sie diese Heuristiken in Ihren Programmen anzuwenden. Ihr Lohn ist bessere Qualität.

## Allgemein

|  |  |
| --- | --- |
| Don’t repeat yourself  DRY (G5) | Keinen duplizierten / redundanten Code schreiben (z.B. mit Copy / Paste). Redundanter Code soll in separate Methoden oder Klassen ausgelagert werden. |
| Keep it small and simple  KISS | Der Code soll so klein und einfach wie möglich gehalten werden. |
| You ain’t gona need it  YAGNI | Kein Code auf Vorrat! Man soll nur Code schreiben, der wirklich gebraucht wird, um die aktuellen Anforderungen zu erfüllen. Code, welcher im Hinblick auf mögliche Erweiterungen geschrieben wird, erweist sich später oft als ungeeignet, da entweder die Anforderungen geändert haben oder man das Problem mittlerweile besser versteht und anders löst. |
| Polymorphismus vor if/else oder switch  (G23) | Nicht jede switch-Anweisung ist nur schlecht, aber man soll sich immer überlegen, ob es nicht vorteilhafter ist, eine polymorphe Lösung (Klassenhierarchie mit überschriebener Methode) zu wählen. Dies ist immer dann der Fall, wenn abhängig vom Typ eines Objektes, eine bestimmte Aktion ausgeführt werden soll. |
| static Attribute und Operationen nur wenn wirklich angemessen  (G18) | Zur Laufzeit eines Programms stehen die Objekte im Zentrum, Jedes Objekt hat seine eigenen Daten. static Methoden können nicht auf diese Daten zugreifen. static Methoden können auch nicht überschrieben werden und verhindern so Polymorphismus. |

|  |  |
| --- | --- |
| Erklärende Variablen verwenden  (G19) | Der Code wird lesbarer, wenn man kompliziertere Berechnungen in einzelne Teilanweisungen aufteilt und Zwischenresultate in Hilfsvariablen mit einem sprechenden Namen abspeichert. |
| Zahlen durch Konstanten ersetzen  (G25) | Ein Ausdruck ist besser lesbar, wenn er statt reinen Zahlen Konstanten mit einem sprechenden Namen enthält. |
| Java Enums einsetzen  (J3) | Wenn es um mehr als Konstanten für Zahlen geht, sollen Enums verwendet werden. Enums können Instanzvariablen und Methoden haben und sind so viel mächtiger. |

## Namen

|  |  |
| --- | --- |
| Sprechende Namen (N1)(G20) | Namen sollen beschreibend sein:   * Variablen: Was speichert die Variable? Beispiel: anzahlBomben * Methoden: Was tut die Methode? Beispiel: bombeLegen() * Klassen: Was kapselt die Klasse? Beispiel: Bombe |
| Keine miss-verständlichen Namen (N4) | Namen sollen präzis sein. Wo anzahlBomben drauf steht soll Anzahl Bomben drin sein. |
| Keine Codierung in Namen (N6) | Namen sollen nicht codiert werden (z.B. mittels Pre- oder Postfixen für Typen, etc.) |
| Seiteneffekte sind im Namen erkennbar (N7) | Dies gilt vor allem für Methoden. Wenn eine Methode einen existierenden Server-Proxy zurückgibt, oder wenn es keinen solchen gibt einen neuen erzeugt, so könnte sie zum Beispiel returnOrCreateServerProxy() heissen. |

## Kommentare

|  |  |
| --- | --- |
| Überflüssige Kommentare  (C2) | Veraltete, irrelevante und falsche Kommentare sind überflüssig. Sie sollen aus dem Code entfernt oder wenigstens aktualisiert werden, sonst verwirren sie den Leser und verschlechtern die Verständlichkeit des Codes. |
| Redundante Kommentare  (C3) | Kommentare, welche nichts anderes sagen, als was man schon aus dem direkt herauslesen kann sind redundant und damit überflüssig. |
| Schlecht geschriebene Kommentare  (C4) | Wenn schon einen Kommentar schreiben, dann richtig. Das heisst der Kommentar soll das Verständnis des Codes fördern. R. Martin: „Nehmen Sie sich die Zeit, den besten Kommentar zu schreiben, den es gibt“. Das heisst: achten Sie auf korrekte Sprache, Prägnanz und Verständlichkeit. |

## Methoden

|  |  |
| --- | --- |
| Methoden sollen nur eine Sache machen  (G30) | Eine Methode soll möglichst nur etwas machen. Der Name beschreibt, was die Methode tut. Daraus folgt, dass Methoden kurz sind (vier bis fünf Zeilen). |
| Zu viele Parameter  (F1) | Eine Methode sollen so wenig Parameter wie möglich haben. Null ist am besten, gefolgt von eins bis maximal drei. Mehr als drei Parameter soll vermieden werden. |
| Flag-Parameter und Auswahlparameter  (F3)(G15) | Eine Methode, welche in Abhängigkeit vom Argument eines Parameters (boolean Parameter) entweder das eine oder das andere macht, macht mehr als eine Sache. Noch schlimmer sind Parameter, welche in einem switch-Statement ausgewertet werden. Solche Methoden sind schwer verständlich und sollen in mehrere Methoden aufgeteilt werden. |
| Methoden sollen nur eine Abstraktions-ebene tiefer gehen  (G34) | Wenn eine Methode mehrere ineinander verschachtelte Kontrollanweisungen (if, while, for, etc.) enthält ist dies ein Indiz dafür, dass die Methode zu stark in die Tiefe geht. Die verschachtelten Teile sollen in eigenen Methoden gekapselt werden. |
| Den Algorithmus verstehen  (G21) | Bevor man Code schreibt, muss man den Ablauf verstanden haben! Einen Ablauf durch Try-and-Error zu programmieren führt zu ineffizientem und sehr schwer verständlichem Code. |