Definition of Clean Code (DOCC)

Java Edition v1.0

Opgesteld door:
Wouter,
Patrick,
Alexander &
Bart

Intec Brussel 13/10/2017


Wat een prachtige code, zeg!

Voorwoord

Dit document bevat regels en richtlijnen waar de cursist zich aan dient te houden. Ze lleft onder de naam: “definition of clean code” van Intec Brussel, en wordt verder docc genoemd.

Clean code en coding houserules zijn van essentieel belang om een moderne organisatie vlot te doen lopen. Zo voorkomen ze bijvoorbeeld vele misverstanden en sparen ze veel tijd uit die anders verloren zou gaan. Bijvoorbeeld door debugging of studie van obscure stukken code.

Wanneer je je code schrijft, denk dan aan je mede programmeur. Denk aan de persoon die na jou de “coding dancefloor” op moet. (Of tegelijkertijd met jou de dans aangaat) Iedereen heeft natuurlijk zijn eigen gepatenteerde “moves”, manieren waarop je zelf een probleem aanpakt. Maar het is veel gemakkelijker wanneer iedereen op zijn minst dezelfde stijl volgt, of dit nu walz is of polka of hardstyle. Onze stijl wordt beschreven in dit document.

Belangrijk: Heb je vragen over 1 of meerdere (of alle) regels? Praat er over met je medecursisten of instructeurs. Niemand werd ooit meester in eenvak door stilzwijgend over problemen te mediteren.

In ieder geval, veel succes en plezier met het volgen van deze regels, ze zullen je sowieso helpen met het oog op de toekomst.

1. Fundamentals

Self documenting code:

Doorheen de opleiding voor java developers zullen jullie altijd geconfronteerd worden met beslissingen over *naamgeving of structuur*. Hoe noem ik mijn methoden, klassen, variabelen? Hoe los ik een bepaald probleem op een structureel correcte manier op. Een richtlijn die we volgen is dat de code zichzelf moet documenteren. Ze moet leesbaar zijn. Wanneer je een variabele nodig hebt die de spanwijdte van een eend in centimeter voorsteld, noem deze dan bijvoorbeeld *spanwijdteInCentimeter*. Wanneer je een gekend probleem oplost, gebruik dan een design pattern.
Self documenting code schrijven is een proces dat constante aandacht vereist. Code die je volgende maand schrijft zou je altijd duidelijker moeten proberen te maken dan de code die je vandaag produceerde.

1. Java basis

Pakketnamen:

Standaard maken we steeds gebruik van de pakketnaam *be.intecbrussel.**
(vb: be.intecbrussel.model)
Klassen, interfaces en enums worden NOOIT in het default package gestoken.

Klassenamen:

We starten onze klassen altijd met een hoofdletter. (vb: Car, Person)

Variabelen:

Variabelen starten altijd met een kleine letter. (vb: int distanceInKilometers)

Methoden:

Methodenamen starten altijd met een kleine letter. (vb: getName())

Code moet compileren:

Code die je afgeeft mag geen compilatiefouten bevatten. Een rood kruisje over je eclipseproject is dus uit den boze.

Indenteer correct:

Zorg er voor dat je code er letterlijk proper uitziet. Je methoden en instance- of klasse-variabelen lijn je uit op de zelfde breedte. Voor iedere codeblok die je opendoet zal je echter 1 tab verder van de linkerzijde moeten beginnen met het schrijven van je nieuwe code. Een voorbeeld van een correct geformatteerde klasse zie je in *bijlage A*.
Meestal kunnen we onze formattering door de IDE laten doen.

Accolades (voor codeblocks):

We openen onze accolades steeds vlak na onze klasse-, methode- en constructor-namen of controleverloop statements.
We sluiten de accolades aan het einde van onze codeblok waarbij we voor ieder niveau van nesting een extra tab toevoegen.
Zie voorbeeld in *bijlage B*.

2. Java advanced

Code in commentaar:

Code in kommentaar is niet toegelaten en zal niet verbeterd worden! Het is echter wel toegelaten om een boodschap naar de instructeur te schrijven in commentaar.
(vb: // Hier komt de implementatie van mijn methode, ik had geen tijd om ze te maken)

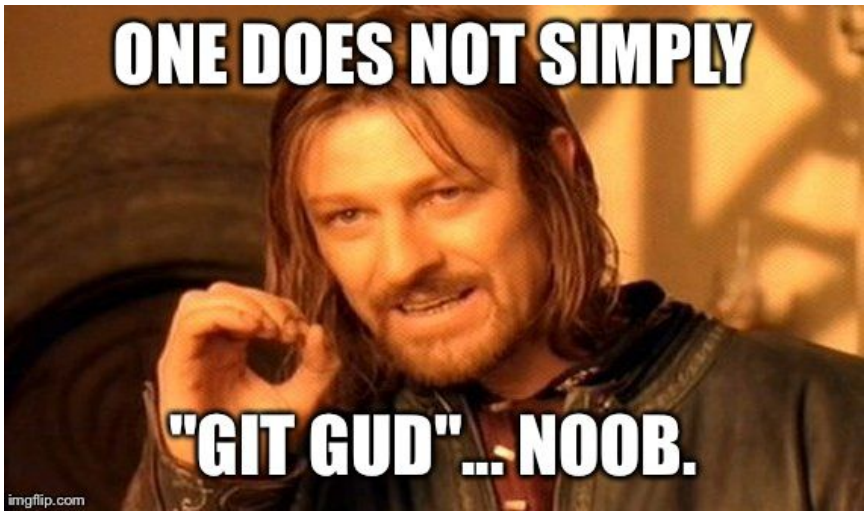
Internationalisering:

De namen van onze variabelen, methoden en klassen schrijven we in het Engels.

Codeblocks voor controleverloopstatements(if, else, for, while, switch) :

Al onze controleverloopstatements zullen een codeblock krijgen.
Geen if statement zonder {} dus!

2. Version control (git)



Commit messages:

- 50 tal characters. Wees bondig en to the point!
- Gebruik de commit body om te zeggen wat en waarom je iets doet. *Niet hoe!*
- Gebruik hoofdletters en leestekens waar nodig. Gebruik echter geen punctuatie in het onderwerp.
- Gebruik de imperatief (gebiedende wijs) in de onderwerplijn.

Commit:

One commit, one problem.
Commit echter veel, zelfs al heb je het volledige probleem niet opgelost.

Commit alleen compilerende code:

Commits don't break the project. Zeker niet op de main branch.

Branches

Gebruiken om te experimenteren en features toe te voegen.

Bijlage A

```
public class ExampleClass{

	int exampleInt = 10;

	public void exampleMethod(){
		System.out.println(“the example number is: “ + exampleInt);
	}

}
```

Bijlage B

```
public class ExampleClass{ // Curly bracket openen

	String exampleString = “Hello World“;

	public void exampleMethod(){ // Curly bracket openen
		System.out.println(“the example String is: “ + exampleInt);
	} //curly bracket sluiten (+ 1 tab)

} // curly bracket sluiten
```