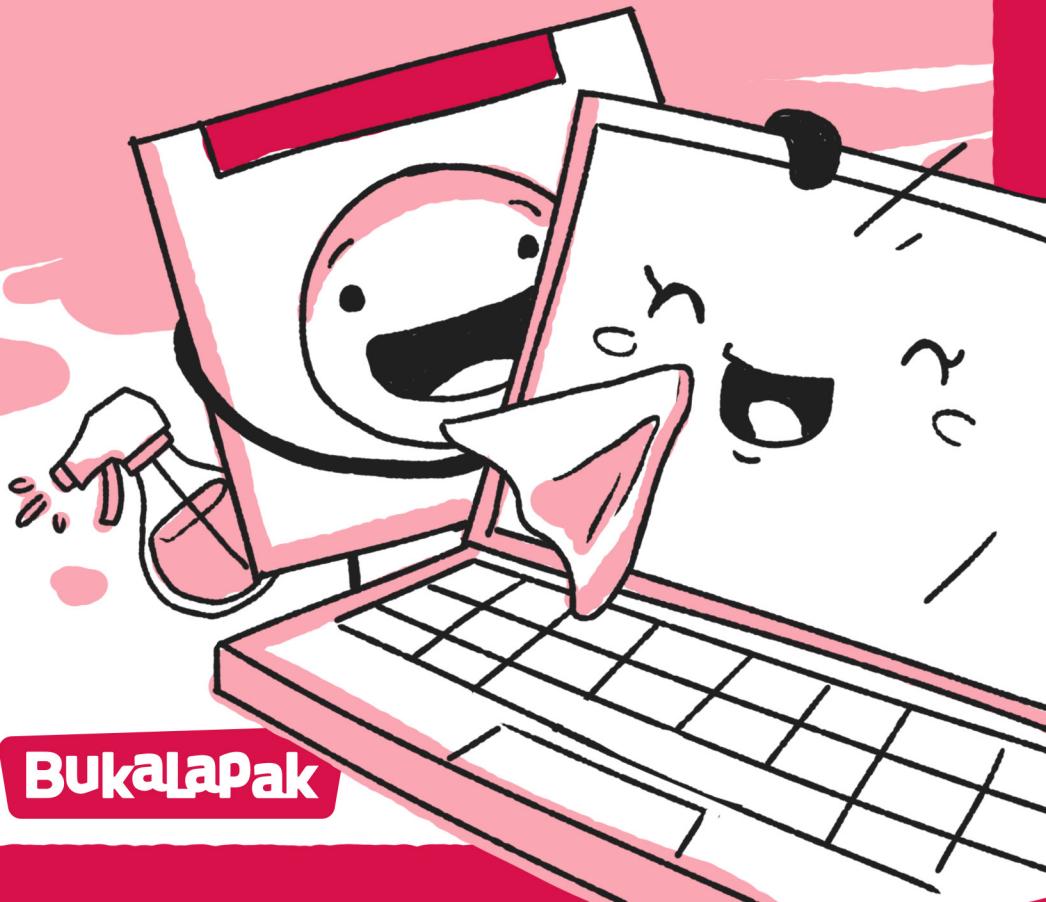


MINI-ENGINEERS #1

# CLEAN CODE



Bukalapak

Bukalapak



Masalah di atas pasti pernah kalian hadapi.

Yuk, coba pelajari prinsip-prinsip **CLEAN CODE**, supaya nantinya nggak terulang masalah sederhana yang bisa menghabiskan waktumu dengan percuma.

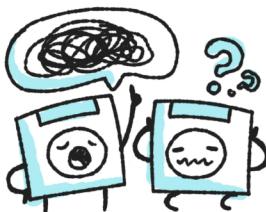
## Daftar Isi

Apa itu Clean Code?	3
Ciri-Ciri Penamaan Baik	5
KISS (Keep It So Simple)	7
DRY (Don't Repeat Yourself)	9
WET (Write Everything Twice)	10
AHA (Avoid Hasty Abstraction)	10
Formatting	11
Refactoring	13
Do it Now!	15

# Clean Code

Clean Code (Kode Bersih) adalah istilah untuk kode yang **mudah dipahami dan diubah**.

## Kenapa harus Clean Code?



1. Seorang coder seringkali perlu bekerja sama, jadi pastikan rekan kamu bisa dengan mudah memahami kode yang dikerjakan bersama.
2. Kode bisa aja terlihat bagus dan rapi, tapi sewaktu kamu harus mengubahnya, kode yang "kotor" bisa membuatmu kesulitan.
3. Penulisan kode untuk jangka panjang akan lebih cepat karena mudah dipahami dan diubah.

## Contohnya?

Ayo, coba kamu analisa kode di bawah. Ada yang janggal?

```
Class worklifebalance {  
    i = 5  
    days_since_creation = 27 #days since creation  
    genydmhs = DateTime.now  
    time[:start]=~ /(0[8-9]|1[0-9]|2[0-1]):00/  
    function search(id)  
}
```

Duh kurang jelas banget kan? Itu contoh sangat tidak baik. Coba bandingkan dengan kode di bawah. Ini baru contoh baik clean code. Bisa lihat bedanya?

```
Class work_life_balance {  
    work_days_per_week = 5  
    days_since_creation = 27  
    generatedTimestamp = DateTime.now  
    #validate for 08.00 until 21.00  
    time[:start]=~ /(0[8-9]|1[0-9]|2[0-1]):00/  
    function search_user(id)  
}
```

# 6 Ciri-Ciri Penamaan Baik

## 1. Mudah Dipahami

Arti atau tujuan cukup jelas tanpa melihat isinya lebih jauh atau pemakaianya.

```
function search(id) ✗  
function search_user(id) ✓
```

search? search apa?  
search\_user lebih jelas.

## 2. Mudah Dieja

Jangan sampai menyebut nama saja menjadi pekerjaan.

```
genydmhs = DateTime.now ✗  
generatedTimestamp = DateTime.now ✓
```

generated jadi g,  
year jadi y, dst.

## 3. Mudah Dicari

Dengan nama yang cukup panjang atau unik.

```
i = 5 ✗  
work_days_per_week = 5 ✓
```

"i" tuh apa? Tidak unik.

## 4. Kadang Tidak Perlu Komentar

Arti atau tujuan cukup jelas tanpa melihat isinya lebih jauh atau pemakaianya.

```
days_since_creation = 27 #days since creation ✗
```

komen tidak diperlukan  
karena nama sudah jelas

## 5. Kadang Perlu Komentar

Komentar diperlukan jika ada informasi tambahan pada kode yang perlu penjelasan.

```
#validate for 08.00 until 21.00  
time[:start] =~ /(0[8-9]|1[0-9]|2[0-1]):00/ ✓
```

## 6. Patuhi Konvensi, Standar, dan Peraturan

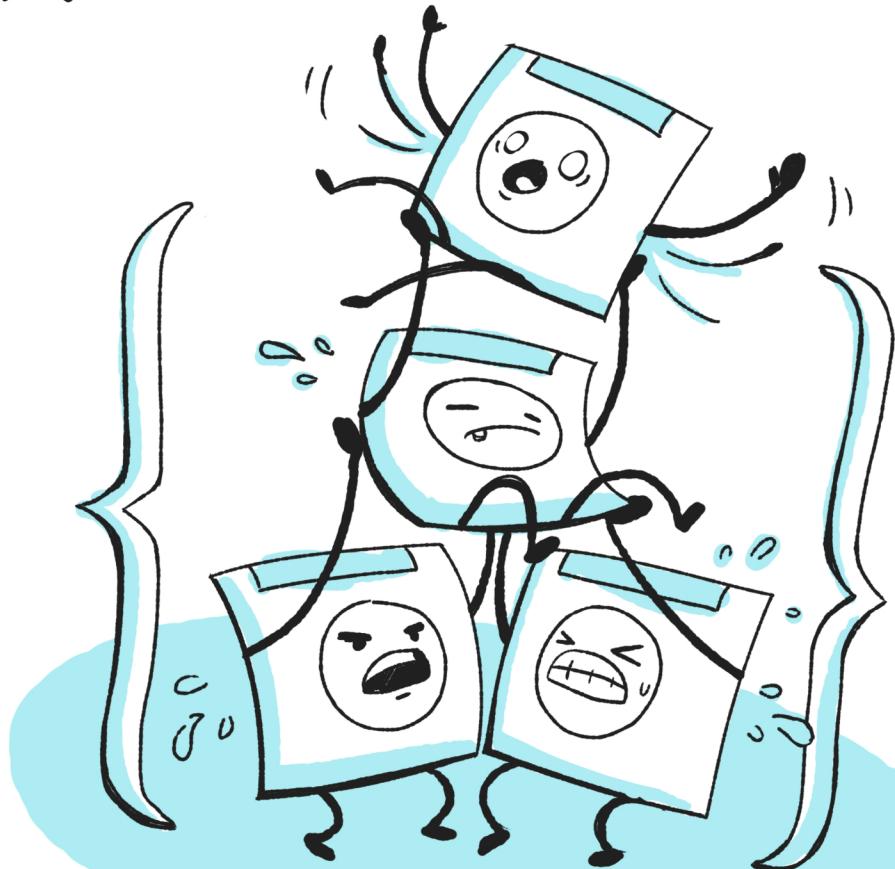
Beda tempat bekerja, bisa jadi beda ketentuan. Patuhi konvensi, standar, dan peraturan yang berlaku di tempat kamu bekerja ya, supaya bisa bekerja dengan nyaman.



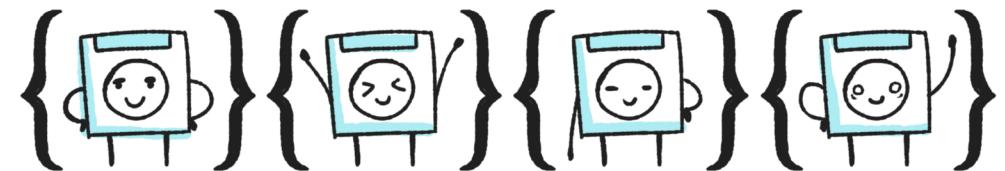
# KISS ( keep it so simple )

Seorang *coder* itu sering *multitasking*. Bagus sih, tapi sayangnya sering terjadi pada kodenya sendiri.

Misalnya nih, suatu fungsi atau *class* dibuat untuk melakukan A, sekaligus memodifikasi B, mengecek C, dst. Hal ini bisa menyulitkan dan jadinya tidak sesuai dengan nama yang kamu berikan.



## Gimana caranya KISS?



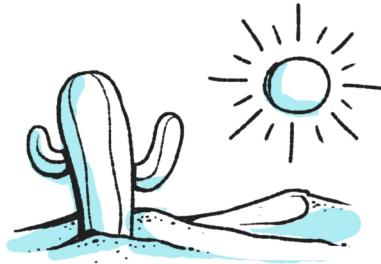
Mau tau cara paling mudah untuk Keep It So Simple fungsi dan *class*? Ini nih 3 cara yang bisa kamu ikuti:

1. Fungsi dan *class* harus kecil.
2. Fungsi dan *class* dibuat untuk melakukan hanya 1 tugas.
3. Jangan gunakan terlalu banyak argumen dalam fungsi.

Akan tetapi, perlu diingat walaupun fungsi dan *class* dibuat sekecil mungkin, bukan berarti kamu akan membuatnya sebanyak mungkin.

Sebagai *coder*, kita harus bisa mencapai kondisi yang seimbang dari kecil dan minimal jumlahnya.

## DRY ( Don't Repeat Yourself )



Kalau merasa *déjà vu* saat menulis kode, besar kemungkinan kamu memang menulis hal yang sama atau mirip di bagian lain.

Kode duplikat terjadi karena sering *copy paste*. yang akibatnya nambah pekerjaan kamu. Untuk menghindari pengulangan, buatlah fungsi atau *class* yang bisa menggantikannya.

```
class Mechanic {  
    function ServiceCar() {  
        System.out.println("servicing car now")  
        System.out.println("performing other tasks")  
    }  
  
    function serviceBike() {  
        System.out.println("servicing bike now")  
        System.out.println("performing other tasks")  
    }  
}
```

Jangan lupa: Kalau merasa *déjà vu* saat menulis kode, besar kemungkinan kamu memang menulis hal yang sama atau mirip di bagian lain.

## WET ( Write Everything Twice )



Selain DRY, ada juga konsep WET yang membolehkan duplikasi **maksimal dua kali**. Jika kamu melakukan duplikasi yang ketiga, saatnya kamu membuat abstraksinya.

## AHA ( Avoid Hasty Abstraction )



Konsep AHA lebih memilih **duplikasi daripada abstraksi** yang bisa saja salah. Konsep AHA ada karena kita tidak tahu bagaimana kode akan berubah ke depannya.

Jadi, duplikasi kode lebih baik dilakukan sebanyak apapun sampai kita yakin bahwa kita bisa membuat abstraksi kode yang tepat, walau akan ada perubahan-perubahan lagi.

---

Ketiga konsep penulisan ini (DRY, WET, dan AHA) adalah pilihan umum bagi coder. **Kamu bebas memilih yang mana saja kok**, yang menurutmu paling baik.

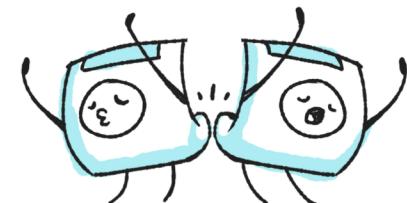
# 7 Saran Formatting Baik

Seperti menulis, biasanya ada format-format yang disarankan. Berikut adalah *formatting* yang baik saat menulis kode:

1. Lebar sebaris kode kurang lebih 80 - 120 karakter.



2. Satu *class* kira-kira 300 - 500 baris (*Class* kecil akan lebih mudah dipahami).

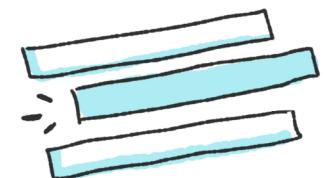


3. Usahakan baris kode yang berhubungan saling berdekatan, supaya bacanya nggak perlu loncat-loncat.

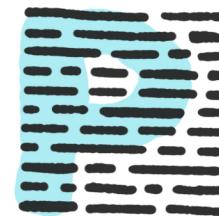
4. Dekatkan sebisa mungkin fungsi yang memanggil (*call*) fungsi lainnya.



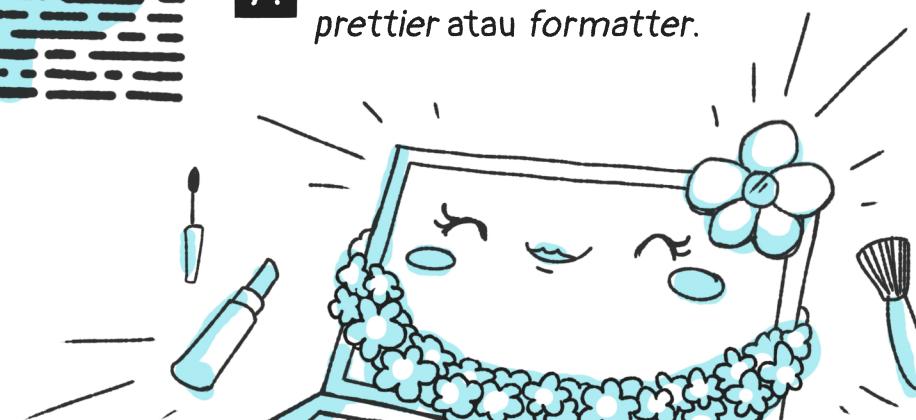
5. Deklarasikan variabel sedekat mungkin dengan penggunaannya.



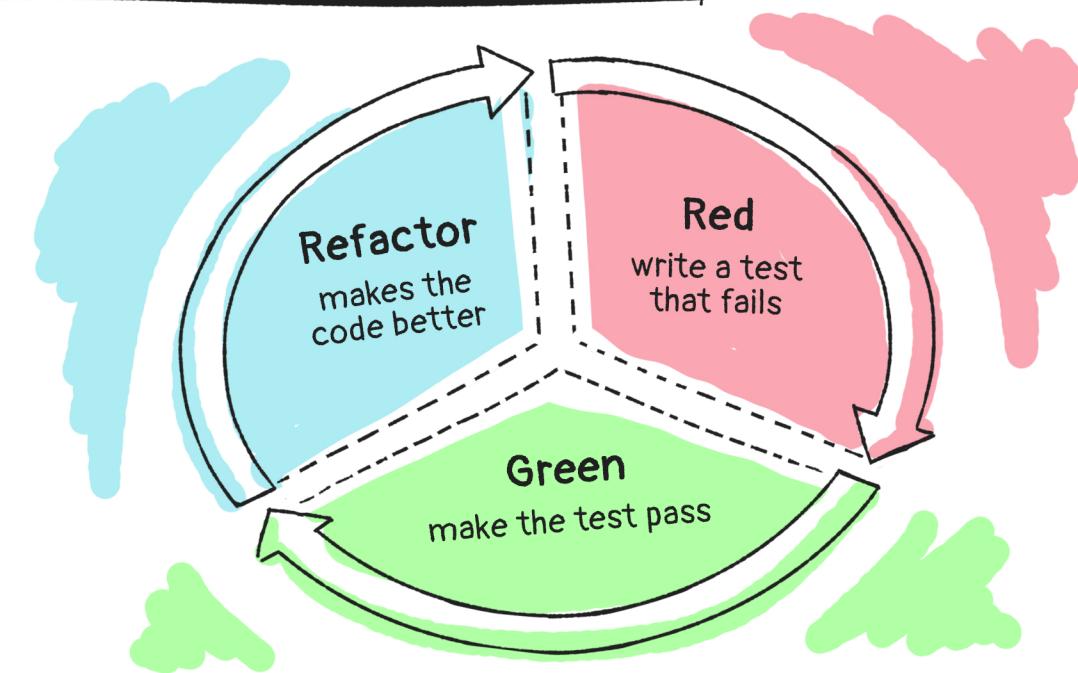
6. Perhatikan indentasi supaya tidak membuat salah paham.



7. Ada baiknya untuk memakai *prettier* atau *formatter*.



# Refactoring



Refactoring adalah proses restrukturisasi kode yang sudah dibuat, dengan cara mengubah struktur internal tanpa mengubah perilaku eksternal. Prinsip DRY dan KISS bisa dicapai dengan refactor kode yang sudah kamu buat.

Hasil dari seorang coder biasanya sih tidak langsung optimal dari awal, jadi, baca dan tes ulang dulu kode yang kamu buat, lalu lakukan refactor.

## Contoh-contoh teknik refactor

Jangan lupa cek kembali setelah refactor untuk memastikan kode sudah bersih dan lolos tes, ya!

1. Buat abstraksi,
  - a. Paksa akses field via getter/setter.
  - b. Pakai tipe yang lebih umum.
  - c. Ubah cek tipe jadi sub-class.
  - d. Ubah kondisi jadi polymorphism.
2. Memecah kode dengan fungsi/class.
3. Percantik penamaan atau lokasi kode.
4. Deteksi kode duplikat.

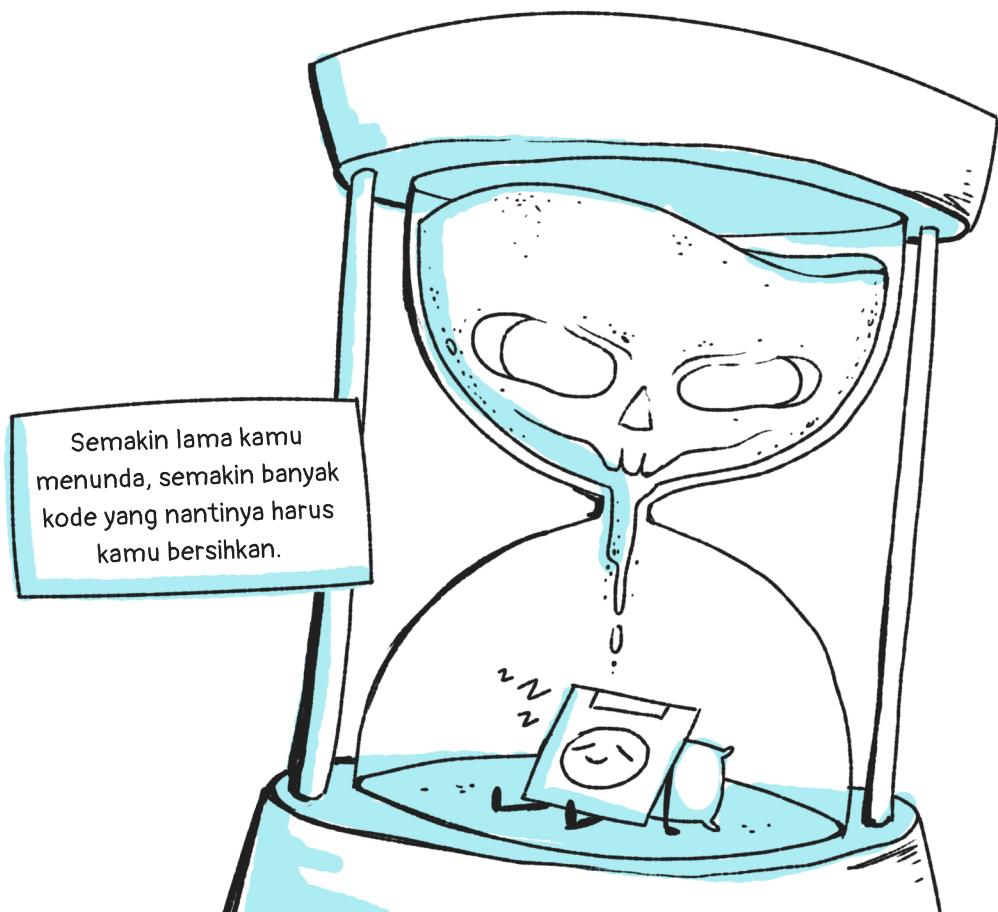




# Do it Now!

Sama halnya dengan utang di bank, semakin lama kamu menunda pembayaran, maka bunga dan denda yang harus kamu bayar akan semakin besar.

Jadi, yang paling penting adalah ikuti prinsip-prinsip yang sudah dijabarkan di atas sekarang, ya.



## Referensi

Martin, Robert C., editor. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2009.

Catalog of Refactorings. <https://refactoring.com/catalog/index.html> and accessed 26 March 2019.

"Code Refactoring." Wikipedia, 15 Mar. 2019.

[https://en.wikipedia.org/w/index.php?title=Code\\_refactoring&oldid=887853856](https://en.wikipedia.org/w/index.php?title=Code_refactoring&oldid=887853856)

Technical Debt. <https://refactoring.guru/refactoring/technical-debt> and accessed 26 March 2019.

AHA Programming. <https://kentcdodds.com//blog/aha-programming> and accessed 26 March 2019.

WTFs/m – OSnews. <https://www.osnews.com/story/19266/wt fsm/> and accessed 26 March 2019.

Code Refactoring (Software Gardening - Pruning) | DotNetCurry. <https://www.dotnetcurry.com/software-gardening/1105/code-refactoring> and accessed 26 March 2019.

# Penyusun

## Design Team

Nikita P. Wibisono

Wilman Winaya

Laura Susilo

Labib Ahmadin

Amanda Siswandani

## Tech Learning Team

Irvan Putra

Rinaldi Oesman

## Copywriting Team

Yuri Bulandari

## Engineering Team

Afiq Rasyid Muhammad

Falah Prasetyo

Galuh Buana Putra Kautsar

Muhammad Sabiq

Danurrohman

Shandy Darma

Jaffer Sadeq