1. **Esai (100 poin)**

Berikan **contoh snippet code dengan smell** yang termasuk dalam kategori berikut:

1. The bloater

2. The object orientation abuser

3. The change preventer

4. The dispensable

5. The couplers

Kemudian, berikan **snippet code solusinya**. **Jelaskan dengan detail** mengapa snippet code pertama mengandung smell yang dimaksud dan bagaimana snippet code solusi bisa menghilangkan smell tersebut **(LO 1, LO 2, LO 3, masing- masing smell berbobot 20 poin)**.

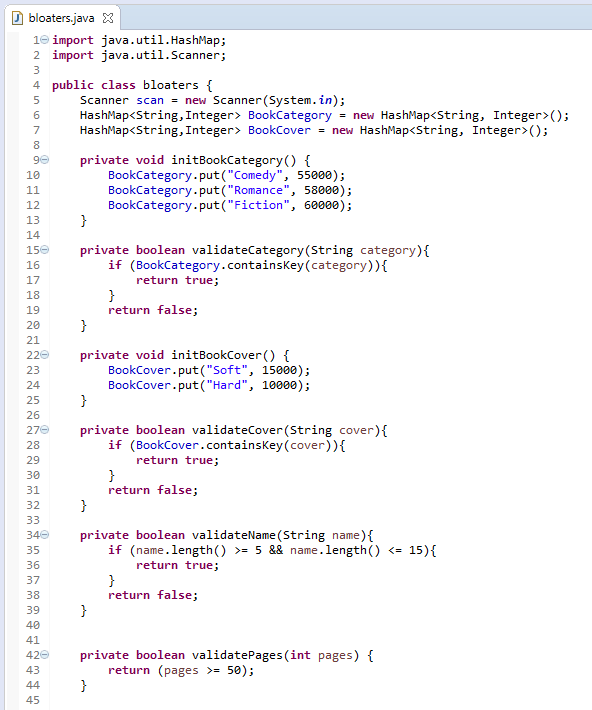
**Jawaban:**

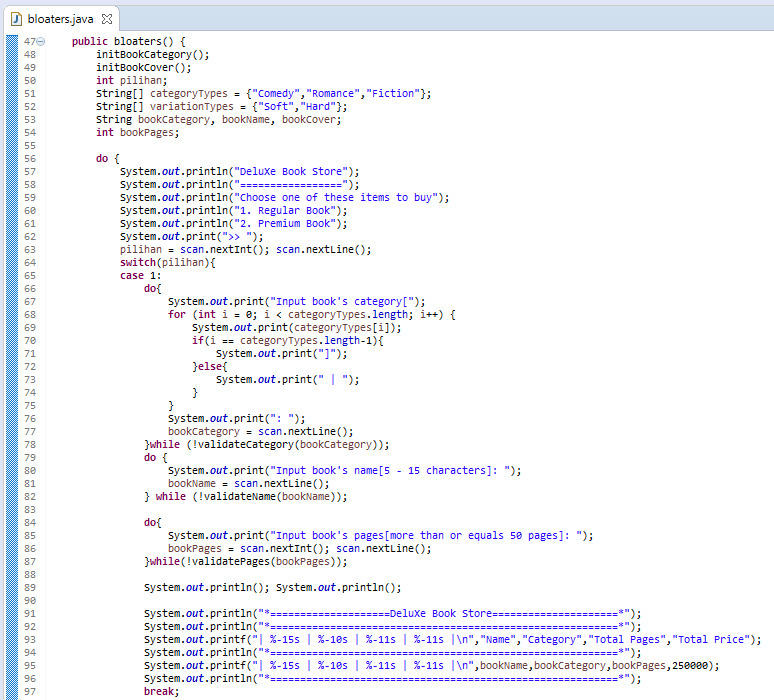
1. **The Bloaters**

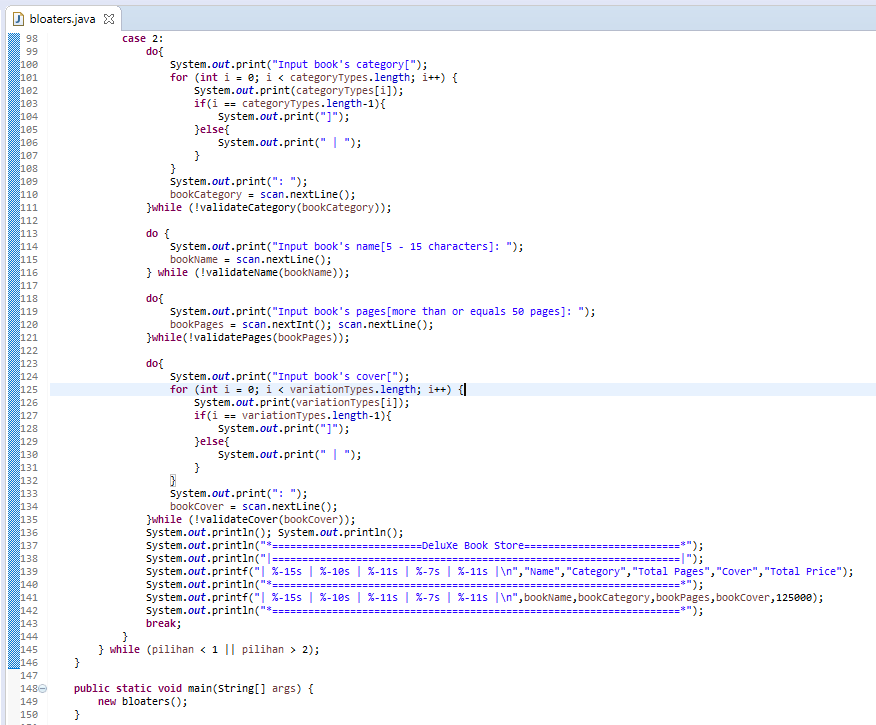
* Semua smell di dalam Bloaters berkaitan dengan bagian code terlalu berat atau membebani dalam suatu class atau method, sehingga akan kesulitan memahami isi kode.

**Jenis-Jenis Code Smell yang termasuk The Bloaters:**

1. **Long Method**







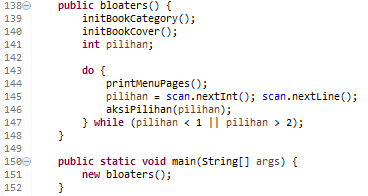
Snippet Code diatas termasuk Long Method Bloaters Code Smell **karena satu method memiliki banyak tanggung jawab**, sehingga kesulitan memahami method. Perlu membaca dengan detail baris per baris berulang-ulang untuk memahami tujuan code ini.

**Yang dimaksud memiliki banyak tanggung jawab** seperti ketika akan memasukkan data buku baik itu regular atau premium, **semba dilakukan di satu method yang sama , tentu itu akan membebani method tersebut**.

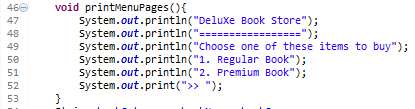
Data yang diinput nama ,variasi cover buku, jumlah halaman, dan kategori dari buku yang akan diinput. Dan memprosesnya hingga mencetak apa yang diinput secara langsung**. Itu semua dilakukan di satu method yang sama sehingga membebani method tersebut**.

**SOLUSI Large Method Code Smell diatas:**

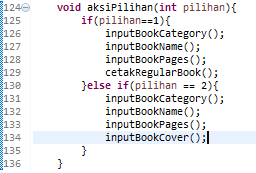
1. **Extract Method Menu**



Membuat Method printMenuPages(), dimana memilih Buku Regular atau Premium.

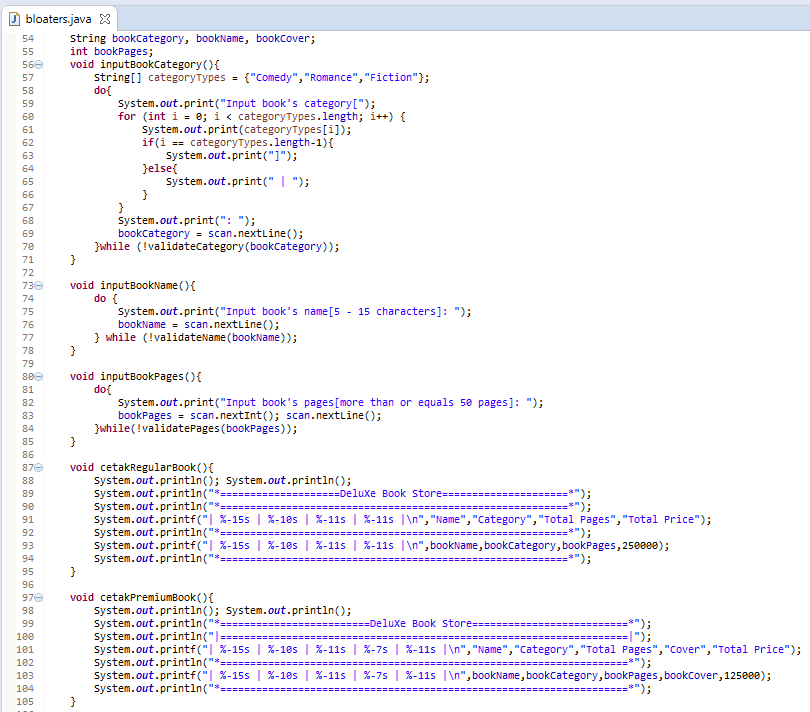


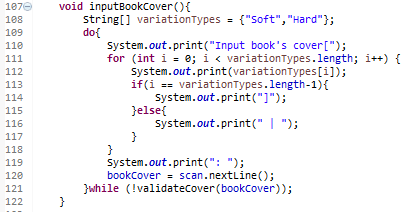
Dimana Method aksiPilihan() akan melakukan aksi sesuai input yang diterima . setiap aksi akan dibuat di method masing-masing sehingga tidak akan menimbulkan large method lagi di tempat lain.



1. **Extract Method untuk Regular dan Premium Book**

Berikut Method-Method yang dibuat terpisah masing-masing sehingga tidak membebani hanya di satu method saja dan lebih mudah untuk dibaca dan dipahami kegunaan methodnya.



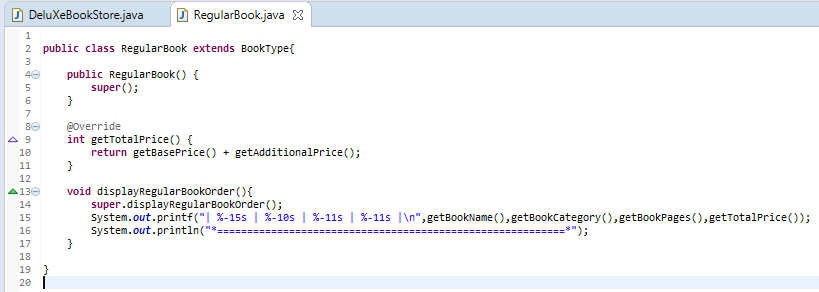


1. **Large Class**

* Smell ini terjadi ketika ada sebuah class yang memiliki terlalu banyak method dan/atau line of code.
* Pada contoh bloaters.java diatas (Long Method), class ini bukan hanya berurusan dengan menu dan input console untuk CRUD saja. Class ini juga pegang kendali untuk pembuatan hashMap. Class ini juga terbebani dengan banyaknya variasi cara input di method.

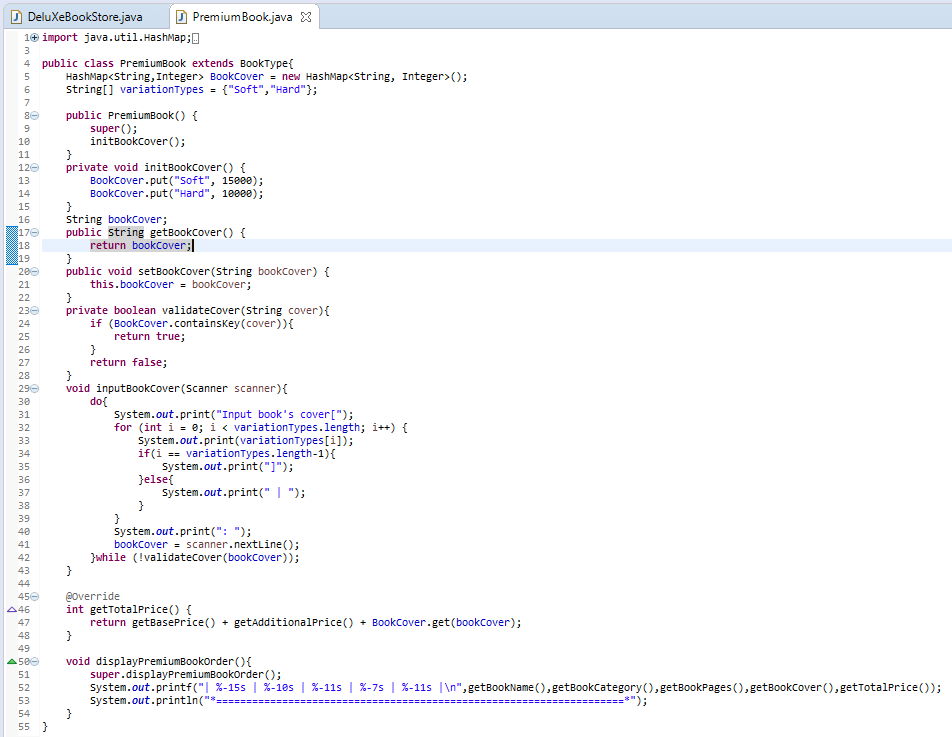
**SOLUSI Large Class Code Smell:**

1. **Extract Class dan Move Method untuk RegularBook**



**Dilakukan Move Method pada:**

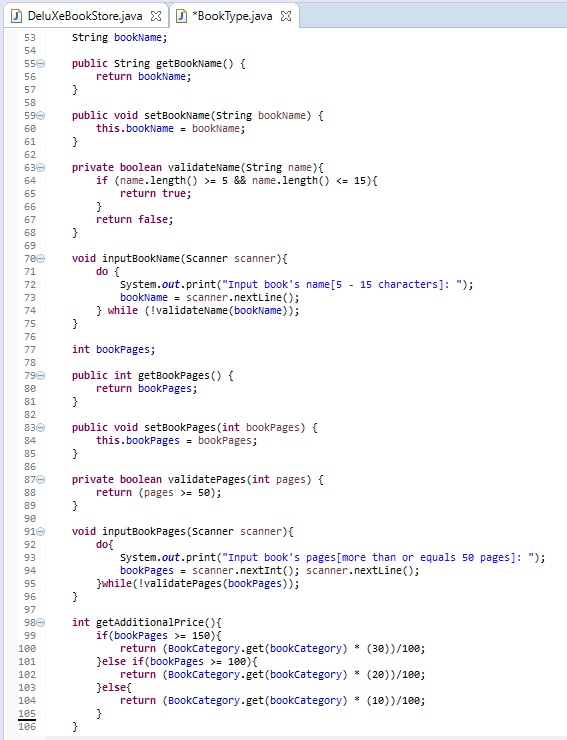
1. fungsi totalPrice **menjadi getTotalPrice di class RegularBook**.
2. fungsi cetakRegularBook **menjadi displayRegularBookOrder di class RegularBook**
3. fungsi Method yang sama akan extends dengan class BookType.
4. **Extract Class dan Move Method untuk PremiumBook**

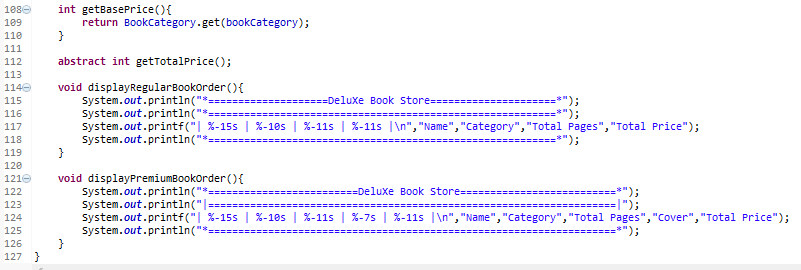


**Dilakukan Move Method pada:**

1. fungsi totalPrice **menjadi getTotalPrice di class PremiumBook**.
2. fungsi cetakRegularBook **menjadi displayRegularBookOrder di class PremiumBook.**
3. fungsi Method yang sama akan extends dengan class BookType.
4. **Move Method dan Extract Class untuk Method yang digunakan**

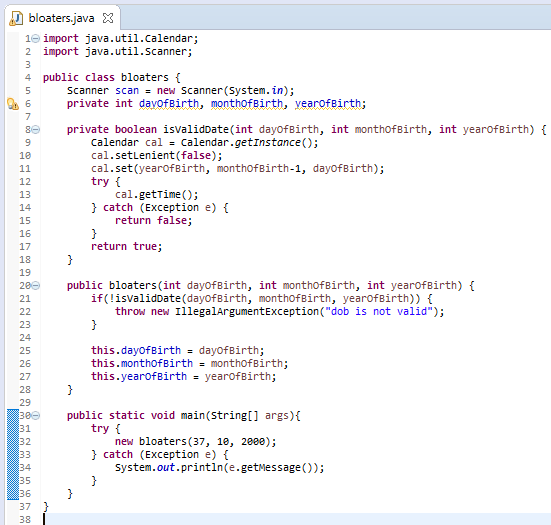






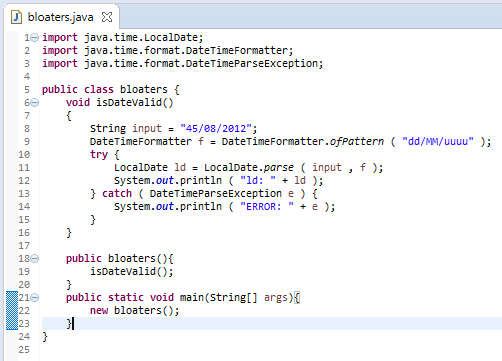
1. **Primitive Obsession**

* Smell ini terjadi bila programmer menyimpan data dengan sebuah tipe data, padahal behavior dari data itu lebih dari tipe data yang ia gunakan, berarti sebenarnya harus diganti atau dibuatkan class baru.



**Karena dayOfBirth, monthOfBirth, dan yearOfBirth menggunakan tipe data primitive.**

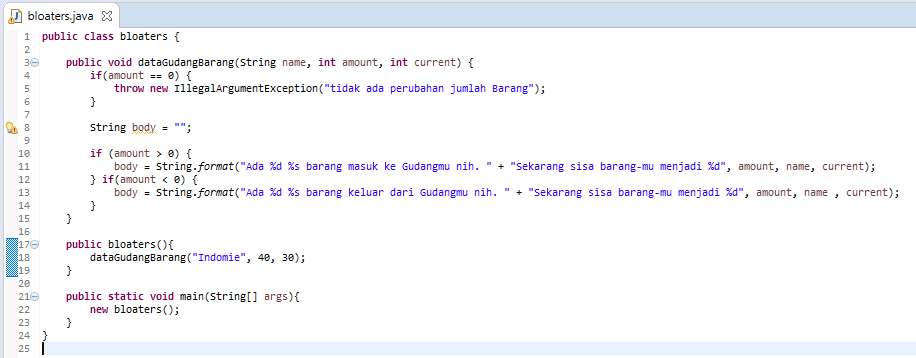
**SOLUSI Primitive Obsession Code Smell:**



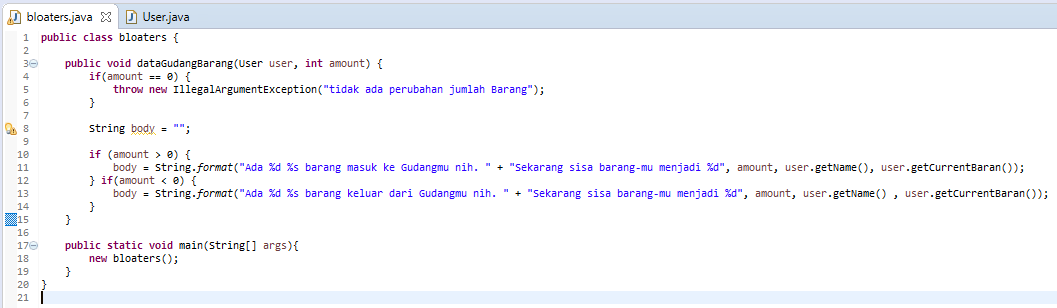
**Untuk integer dayOfBirth, monthOfBirth, dan yearOfBirth, kita ganti menggunakan class java.time.LocalDate bawaan dari Java**. Sehingga tidak perlu deklarasi banyak variable untuk tanggal lahir, karena sudah disediakan function nya.

1. **Long Parameter List**

* Smell ini terjadi ketika ada method yang memiliki jumlah parameter yang terlalu banyak membuat code menjadi sulit dibaca.
* menyebabkan cognitive overload pada programmer karena ada beban memori harus mengingat-ingat letak parameter.



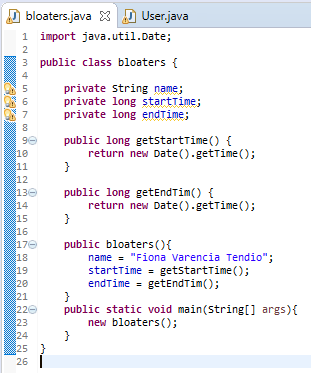
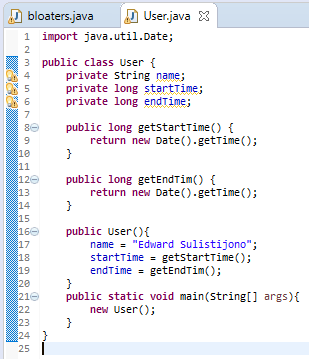
**SOLUSI Long Parameter List Code Smell:**



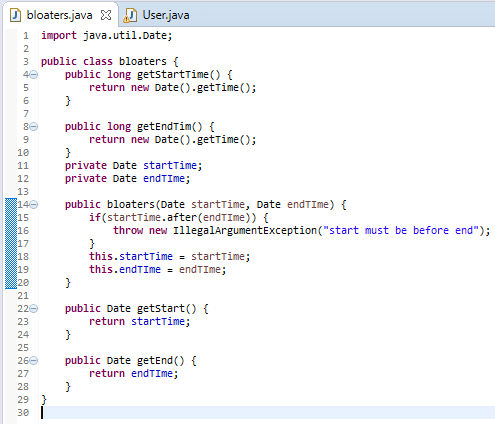
Dilakukan Preserve Whole Object pada method dataGudangBarang sehingga menjadi dataGudangBarang (User user, int amount). Karena name dan current balance bisa didapatkan langsung dari object user.

1. **Data Clumps**

* Smell ini terjadi ketika ada beberapa field atau parameter method yang selalu dipakai atau dioper bersama-sama.
* Bila terjadi hal seperti ini, berarti variabel-variabel ini bisa dipertimbangkan untuk digabungkan langsung dalam satu class.

**SOLUSI Data Clumps Code Smell:**



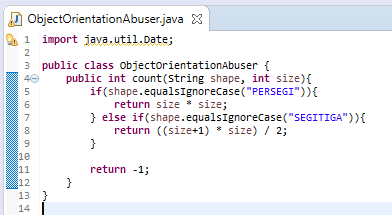
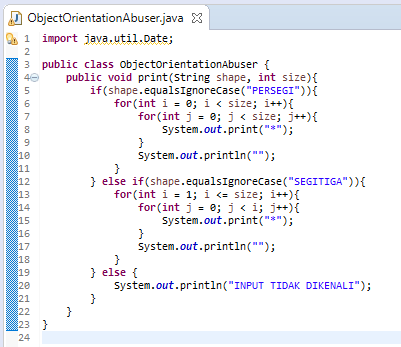
Karena selalu berbarengan, start dan end dipertimbangkan untuk digabung dalam class baru. Agar tidak hanya menjadi smell data class, kita dapat melengkapi class dengan behavior tertentu. Contohnya kita tambahkan validasi start tidak boleh melebihi end.

1. **The Object Orientation Abuser**

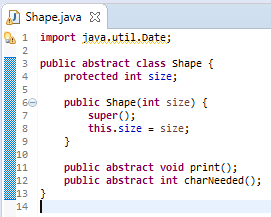
* Semua smell di dalam grup ini berkaitan dengan penerapan prinsip OOP yang kurang tepat, baik karena kurangnya pengetahuan mengenai OOP maupun pemikiran programmer yang terlalu procedural pada OOP.

1. **Switch Statements**

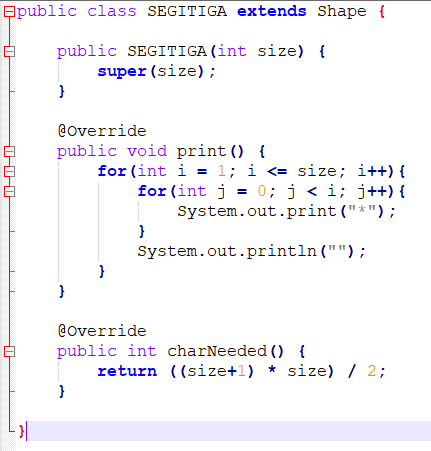
* Pemakaian switch atau if-else untuk menentukan operasi pada variasi tipe-tipe tertentu. Tidak semua switch atau if-else itu berbahaya.

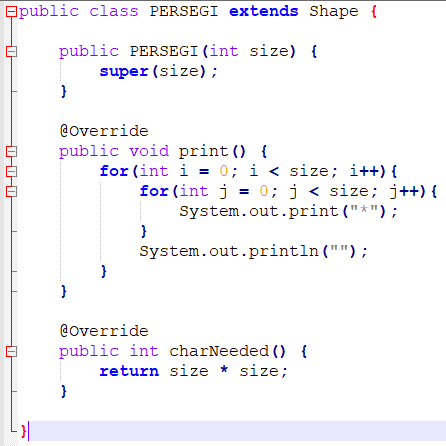


**SOLUSI Switch Statement Code Smell:**

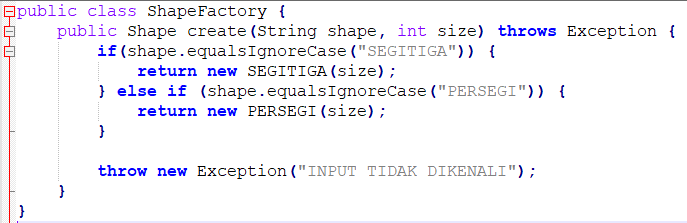


Buat class sebagai abstract class yang memiliki fungsi charNeeded dan print.





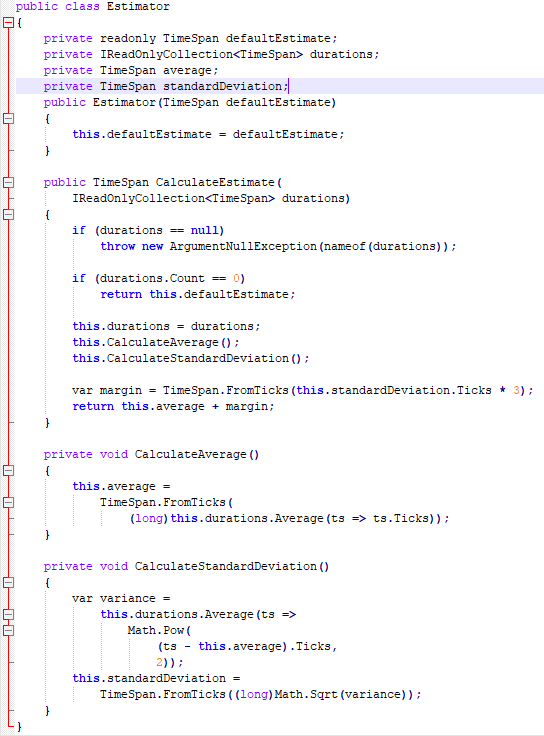
Lalu Buat Class Triangle.java dan Square.java meng-extends class Shape.



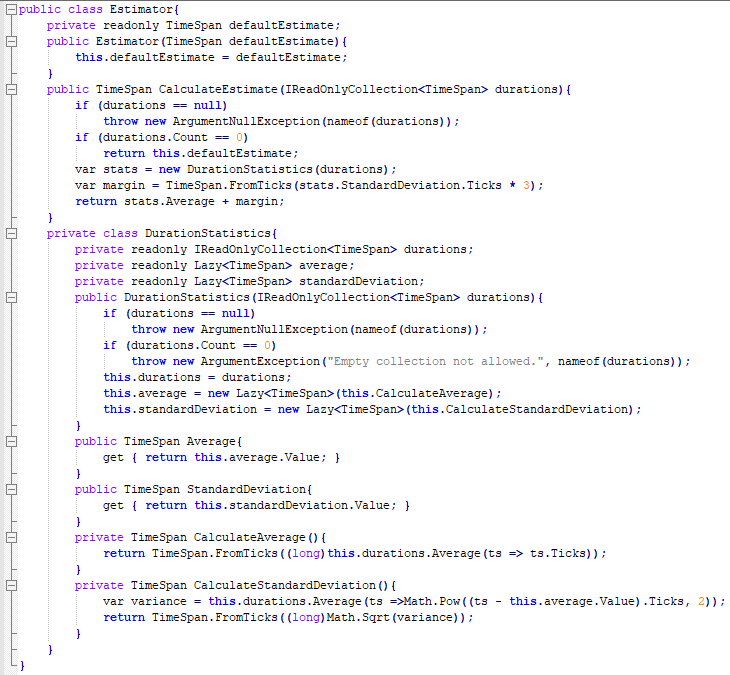
Setelah class PERSEGI dan SEGITIGA sudah terbentuk. Logic print dari ShapePrinter dan logic menghitung karakter dari CharNeededCounter dipindahkan ke masing-masing class.

1. **Temporary Field**

* Ketika fungsi selesai dijalankan, field ini tidak pernah lagi digunakan, mempertimbangkan apakah field yang cocok menempati class tersebut.



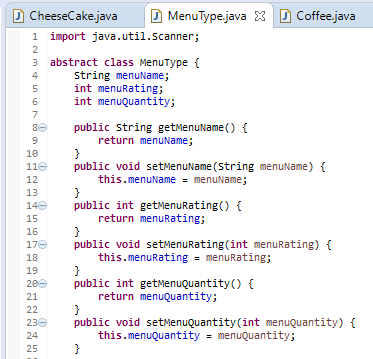
**SOLUSI Temporary Field Code Smell:**



**Semua fungsi berkaitan dengan Menghitung Durasi Statistik dan semua variabelnya kita usir ke class baru durationStatistic.java.**

1. **Refused Bequest**

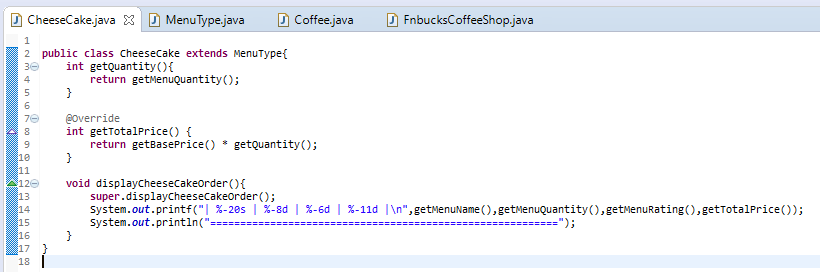
* Sebuah class turunan tidak memakai seluruh method hasil extendsnya.





Dalam Class MenuType.java, ada banyak function yang disediakan dari getter dan setter dari suatu vector. Kita menggunakannya sesuai kebutuhan kita.

**SOLUSI Refused Bequest Code Smell:**

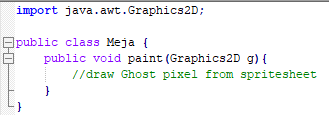


Oleh karena itu, diakali dengan cara melakukan override pada masing-masing fungsi yang tidak ingin digunakan, dan kita menghilangkan kinerjanya dengan cara menghapus pemanggilan super. Dan kasus ini hanya memanggil getBasePrice dan getQuantity serta getMenuQuantity saja.

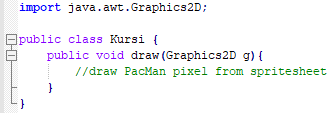
1. **Alternative Classes with Different Interfaces**

* Beberapa class yang memiliki tujuan yang sama namun bisa saja memiliki nama yang berbeda, atau bisa juga memiliki parameter yang serupa namun beda urutan, atau ada parameter yang satuan, ada yang berupa objek.

Contohnya, class Meja.java memiliki fungsi paint yang bertujuan menggambar Meja dari spritesheet yang tersedia.

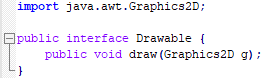


Di sisi lain, class Kursi.java memiliki fungsi draw yang bertujuan sama. Menggambar Kursi dari spritesheet yang tersedia.



**SOLUSI Alternative Classes with Different Interfaces Code Smell:**

Untuk contoh kasus ini, solusinya Extract Superclass. membuat interface Drawable.java yang akan di-implement oleh kedua class.



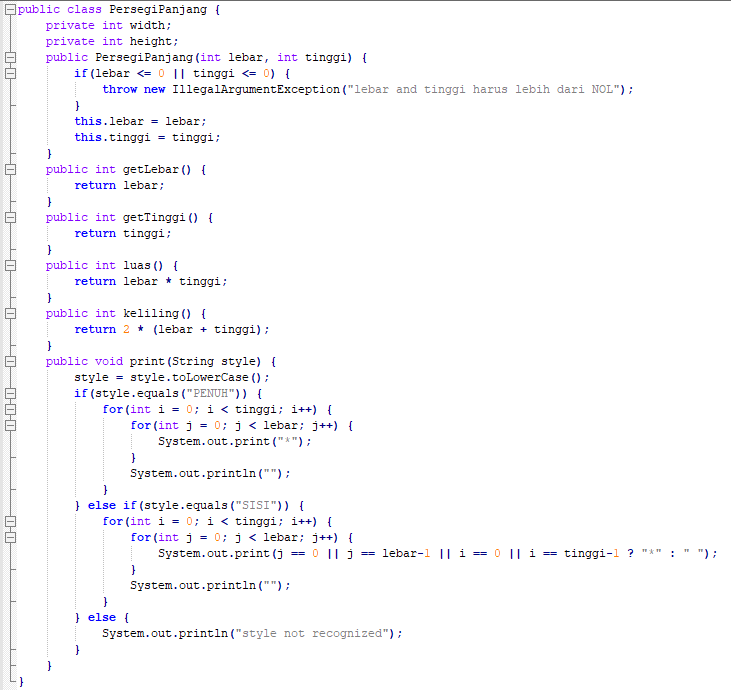
Pada class Kursi sebelumnya, nama fungsi adalah draw, sedangkan pada Meja nama fungsi adalah paint. Dipilih salah satu dari kedua sinonim tersebut yang akan dipakai. Misal diputuskan draw yang dipakai sebagai nama method di interface Drawable, maka dilakukan Rename Method pada fungsi paint di Meja.

1. **The Change Preventer**

* Semua smell di dalam grup ini berkaitan dengan code yang mempersulit ketika ingin melakukan perubahan atau penambahan fitur.

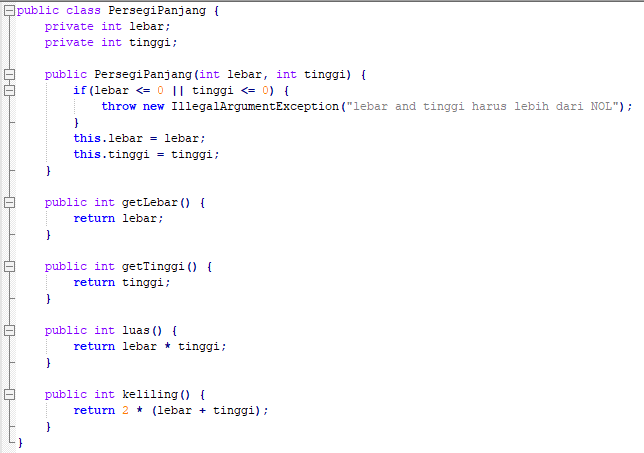
1. **Divergent Change**

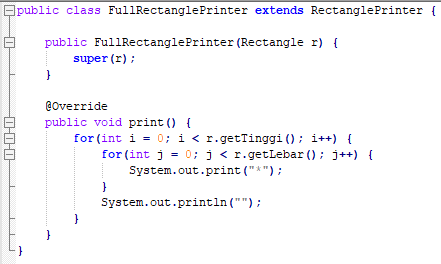
* Smell ini terjadi ketika sebuah class sering berubah untuk alasan yang berbeda-beda. Dengan kata lain, class ini melanggar Single Responsibility Principle.
* Perubahan ini kembali ke diskusi tim masing-masing ketika code design, atau menunggu gejala benar-benar jelas ketika terbukti benar class ini sering berubah untuk alasan yang berbeda.

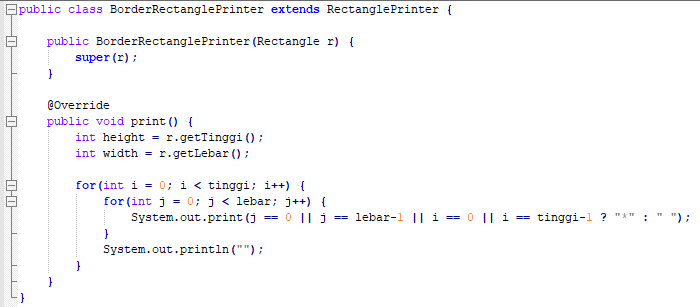


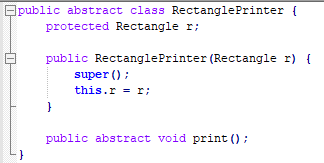
class PersegiPanjang terdapat field lebar dan tinggi. Terdapat method luas() dan keliling(). Dan juga terdapat method print(String style). Class ini memiliki dua tanggungjawab, yaitu menghitung luas dan keliling Persegi Panjang dan juga mengatur tampilan Persegi Panjang ke dalam console.

**SOLUSI Divergent Change Code Smell:**





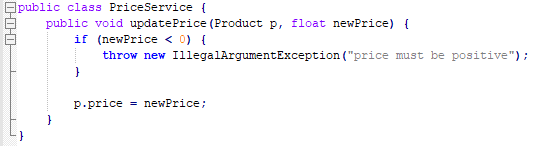


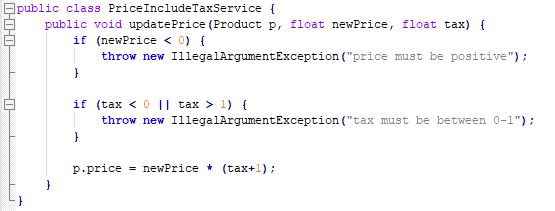


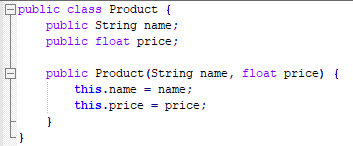
Pindahkan method print di class PersegiPanjang ke class yang baru. Karena di print juga terdapat smell primitive obsession, sekalian buat struktur baru menggunakan strategy design pattern.

1. **Shotgun Surgery**

* Smell ini terjadi ketika ingin mengganti atau menambahkan fitur ke dalam code, perlu mengganti bagian code yang tersebar di banyak class lain.
* Analoginya adalah bila seseorang ditembak dengan senjata api jenis shotgun, luka tembak akan menyebar di banyak tempat.

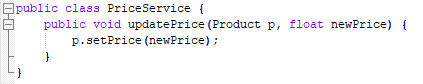


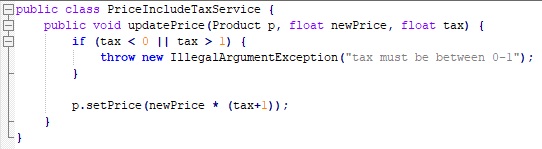


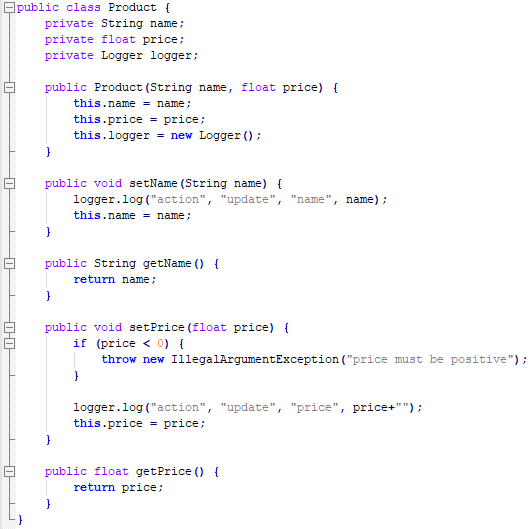


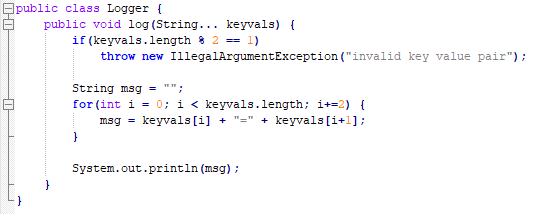
Terdapat code untuk mengubah price yang access modifier-nya public. Masing-masing fungsi update juga memiliki validasi price masing-masing. Misalkan ada fitur tambahan dimana setiap kali ada update value pada class Product, maka melakukan logging menggunakan class Logger.

**SOLUSI Shotgun Surgery Code Smell:**





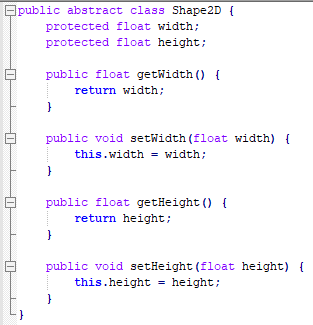




ubah access modifier price menjadi private. Code validasi price dipindahkan ke setPrice. Lalu fungsi setter ditambahkan logging. Sekarang, update value price sudah terpusat di dalam fungsi setPrice, sehingga bila suatu saat nanti ada perubahan mengenai update value price, cukup mengubah satu kali saja di satu tempat tanpa mengubah di seluruh class yang bersangkutan.

1. **Parallel Inheritance Hierarchies**

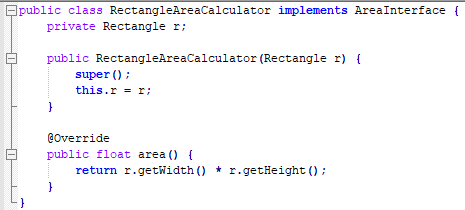
* Untuk menghadapi smell ini, perlu mempertimbangkan apakah cocok bila hirarki-hirarki ini digabung menjadi satu.
* Smell tidak perlu ditangani bila memang hirarki paralel ini disengaja untuk keperluan pemisahan konteks/tanggungjawab. Atau ketika digabung code malah menjadi lebih jorok.

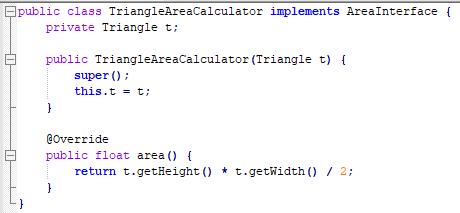








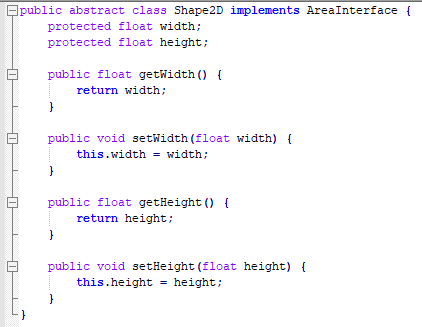


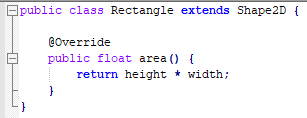


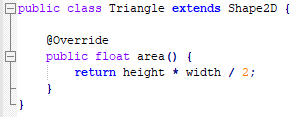
Shape2D selalu bersamaan dengan Interface Area. Bila ada Shape2D yang baru, misalnya Circle. maka Interface Area juga akan membuat anak baru CircleAreaCalculator.

**SOLUSI Parallel Inheritance Hierarchies Code Smell:**









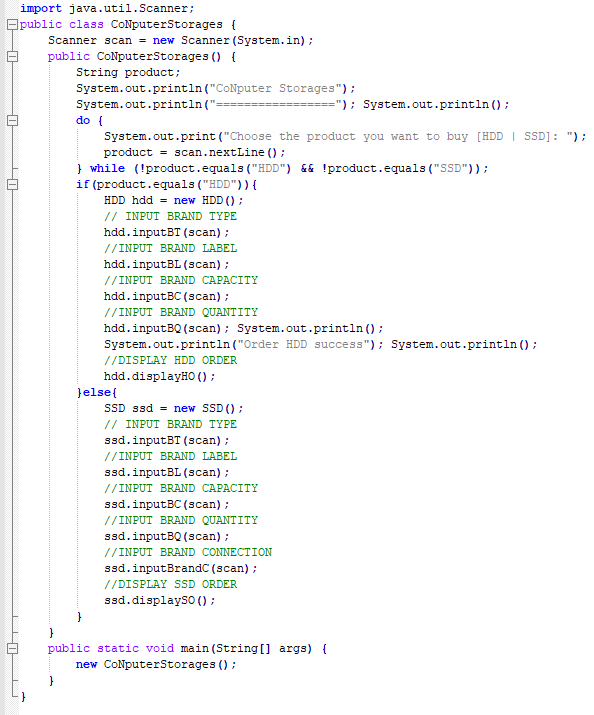
Memindahkan logika perhitungan area agar digabung saja ke dalam Shape2D karena keduanya sama sama menginplementasi Interface untuk menghitung suatu luas bangun datar dengan tipe bangun yang berbeda.

1. **The Dispensable**

* Semua smell di dalam grup ini berkaitan dengan bagian-bagian code kurang berguna dan bisa dibuang.

1. **Comments**

* Nama method atau atribut haruslah sudah self-explain tanpa harus diberi comment.
* Begitu juga logic flow di dalam fungsi. Bila logic flow perlu dijelaskan, dapat dipertimbangkan dan perlu di-extract.



**Contoh CoNpurerStorage.java, terdapat banyak komentar.**

Lebih parahnya, penamaan method function sangatlah buruk, seperti contoh “inputBT”, “inputBL” membuat seseorang memahami apa maksud dari penamaan tersebut dan akan membuat waktu menjadi terbuang untuk hal yang sebenarnya bisa dibuat lebih cepat.

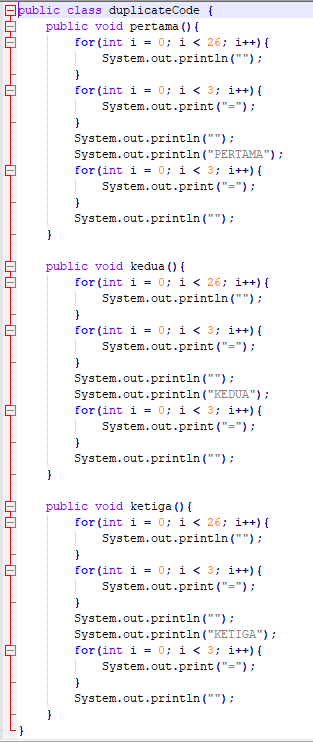
**SOLUSI Comments Code Smell:**



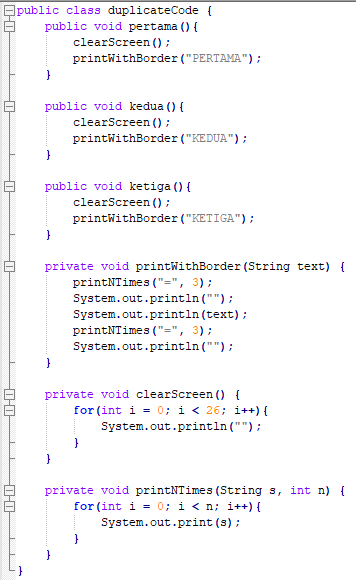
Dilakukan rename method pada hdd dan ssd sehinggal mempermudah serta mempercepat proses untuk memahami sehingga programmer bisa fokus ke hal lain yang lebih penting dibandingkan membaca lagi apa yang dimaksud dari method function ini.

1. **Duplicate Code**

* Terdapat bagian code yang sama atau mirip sehingga akan bisa membebani memori yang digunakan sehingga bisa memperlambat jalannya sistem.



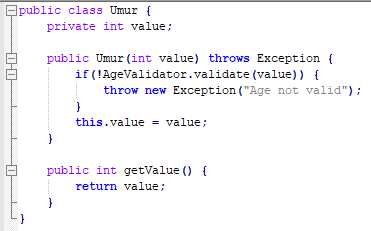
**SOLUSI Duplicate Code Smell:**

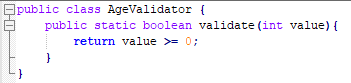


Terjadi extract method di banyak tempat. Selain itu, for yang serupa tetapi tidak sama juga diakali dengan cara di-extract namun ditambahi variabel sehingga bisa digunakan di ketiga fungsi tersebut.

1. **Lazy Class**

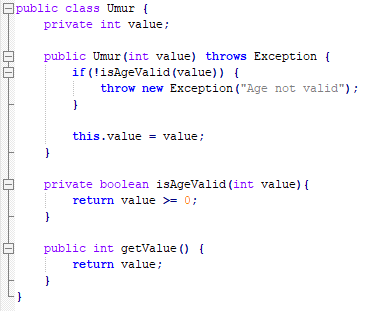
* kondisi dimana class memiliki fungsi yang minim, ekstrimnya hanya memiliki satu buah fungsi.





Contoh di AgeValidator.java. Class hanya memiliki satu buah fungsi untuk validasi umur.

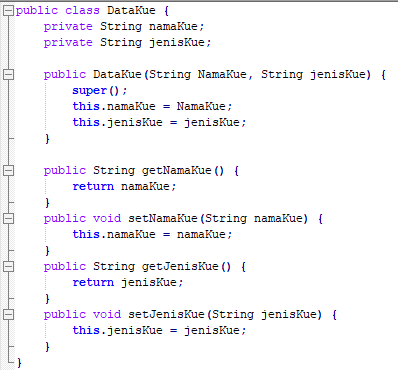
**SOLUSI Lazy Class Code Smell:**

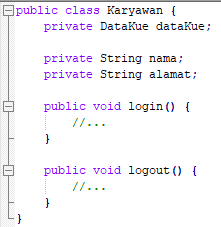


Fungsi validate dipindahkan ke class Umur.java kemudian diberi nama yang lebih sesuai untuk class Umur: isAgeValid. Setelah dipindahkan, class AgeValidator bisa dihapus.

1. **Data Class**

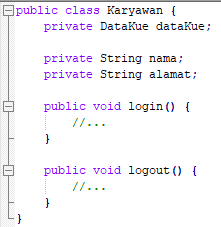
* Bila class hanya cuma sebagai dumb data holders, class bisa dipertimbangkan untuk dibuang dengan cara digabung dengan class lain.
* Dumb data holders berarti class ini hanya memiliki field dan fungsi setter getter saja.





Contoh kasus DataKue.java, class hanya berisi setter getter untuk field namaKue dan jenisKue.

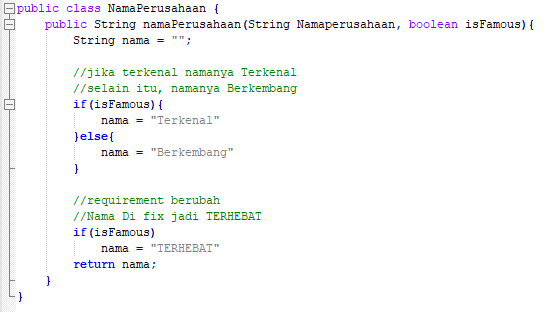
**SOLUSI Data Class Code Smell:**



Semua field dari DataKue dipindahkan ke Karyawan.java. Kemudian class DataKue dihapus.

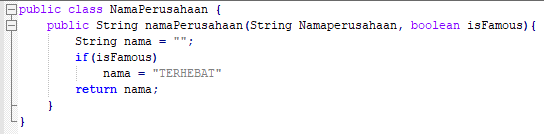
1. **Dead Code**

* Bagian code yang tidak akan tersentuh ketika sistem dijalankan. dead code biasanya akan diberi warning.
* Misalnya di Eclipse diberikan warning dengan simbol lampu kuning yang memberitahu line of code ini adalah dead code.



Awalnya requirement dari client adalah: namanya "Terkenal" bila Famous, jika tidak, namanya "Berkembang". Tiba-tiba, requirement dari client berubah. Client meminta Namanya menjadi “TERHEBAT”.

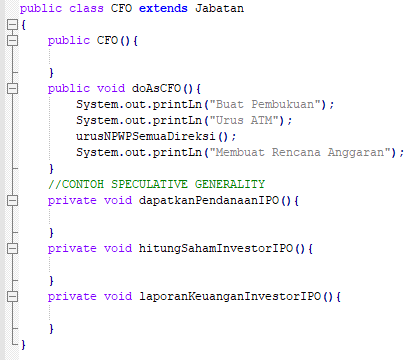
**SOLUSI Dead Code Smell:**



variabel nama akan selalu ditimpa di line of code selanjutnya. Sehingga bagian ini aman untuk dibuang dan hasilnya seperti diatas.

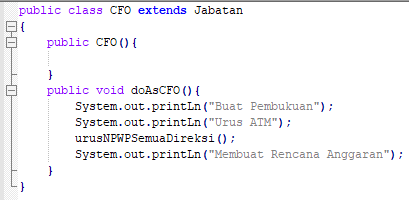
1. **Speculative Generality**

* Class, method, field, atau parameter yang sudah disiapkan walaupun belum dipakai. Bahkan sebenarnya tidak ada dalam requirement.
* Biasanya karena programmer mempunyai ide, berspekulasi, bahwa ini akan dibutuhkan. Namun ternyata setelah produk berjalan lama, spekulasinya ini tidak terbukti.



Contoh Kasusnya yaitu misal sebuah startup memiliki beberapa tahap pendanaan hingga akhirnya sampai pada tahap pendanaan IPO (Initial Public Offering). Tapi untuk sampai pada tahap itu, diperlukan waktu 5-10 tahun. Pada code diatas, terdapat Speculative Generality karena programmer menyiapkan fungsi yang tidak digunakan saat ini.

**SOLUSI Speculative Generality Code Smell:**



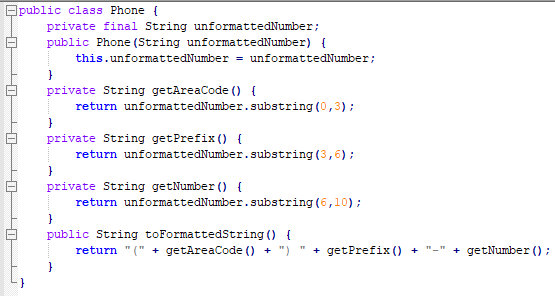
Pada code diatas, sudah Tidak terdapat Speculative Generality karena programmer menyiapkan fungsi yang digunakan saat ini. Alasannya untuk sampai pada tahap itu, diperlukan waktu 5-10 tahun.

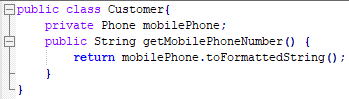
1. **The Couplers**

* Semua smell di dalam grup ini berkaitan dengan coupling yang tinggi.
* code yang baik memiliki coupling yang dibuat serendah mungkin antar modulnya, dan memiliki cohesion yang dibuat setinggi mungkin di dalam modulnya.

1. **Feature Envy**

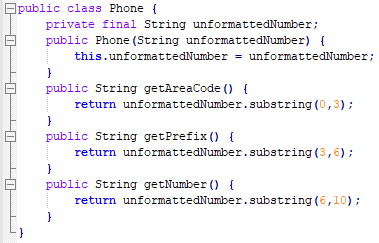
* Smell ini terjadi bila ada sebuah method yang lebih sering mengakses data class lain ketimbang class sendiri.
* Bila hal ini terjadi, harus dipikirkan bagaimana cara memindahkan method ini ke class lain tersebut.

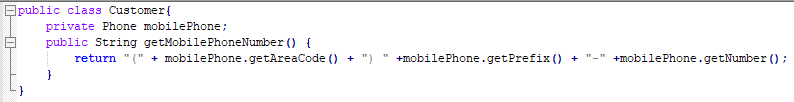




Class Customer bergantung dengan Class Phone untuk melakukan formatting. Sehingga jika terjadi sesuatu, akan membuat dampak yang besar jika tidak dibenarkan kebergantungan yang cukup sangat besar ini.

**SOLUSI Feature Envy Code Smell:**

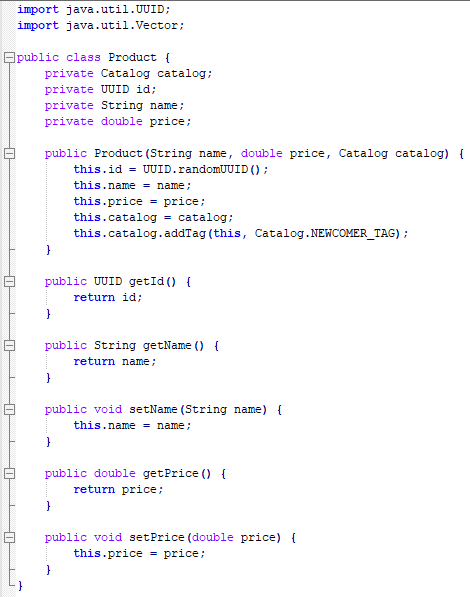


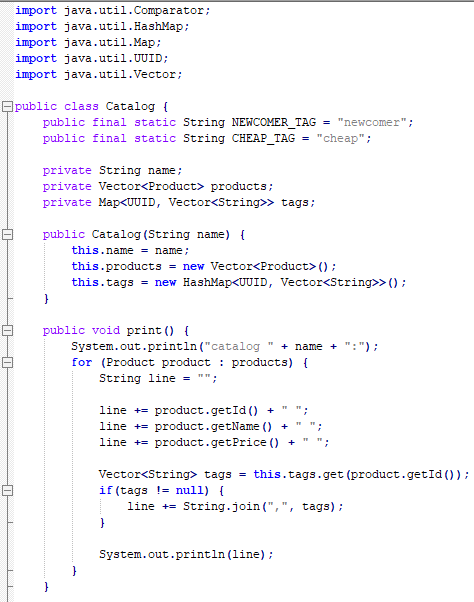


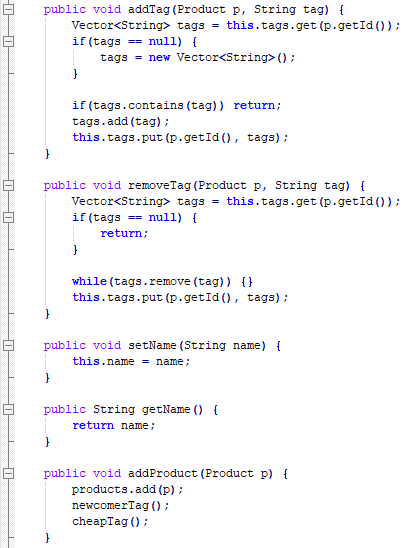
Class Customer dapat akses Method Class Phone untuk formatting. Sehingga jika terjadi Error, maka akan bisa dilacak dengan mudah karena kebergantungan nya sudah sangat dikurangi.

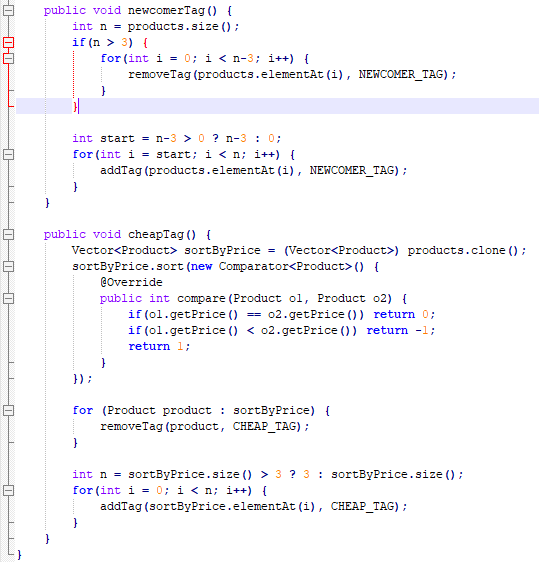
1. **Inappropriate Intimacy**

* Smell ini terjadi karena hubungan antar class yang terlalu intim, menyebabkan class yang satu mengeksploitasi internal field dan/atau method di class lain.
* Hal ini berbahaya karena class yang internal logic-nya diakses bisa mendapat perlakuan yang tidak terduga.



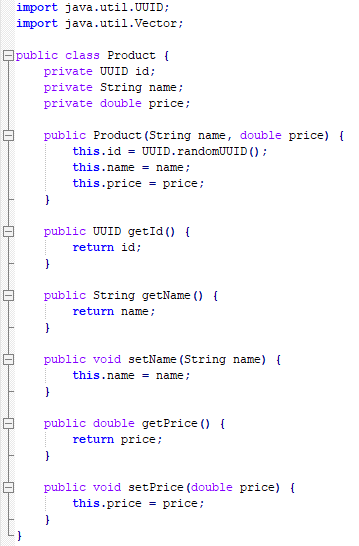




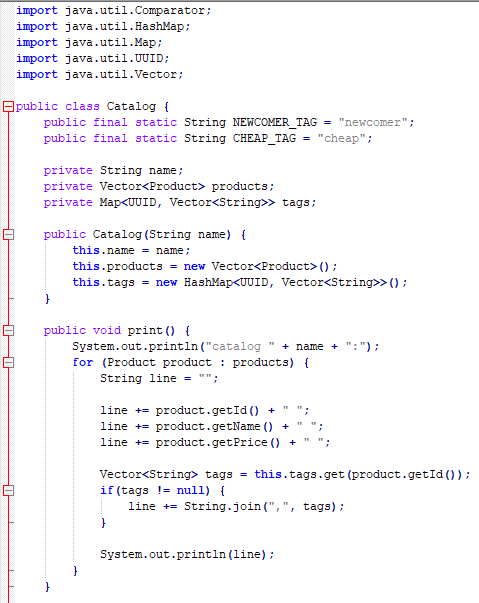


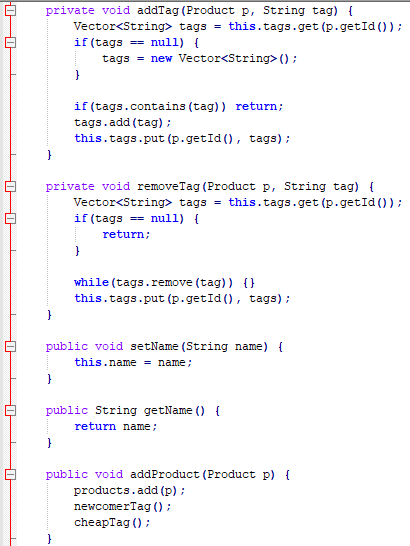
Pada constructor di Product.java, class Product mengetahui ada fungsi addTag di Catalog dan menambahkan tag-nya sendiri ke Catalog. Hal ini tentu berbahaya karena artinya, class Product bisa menambahkan tag apapun tanpa seizin Catalog.

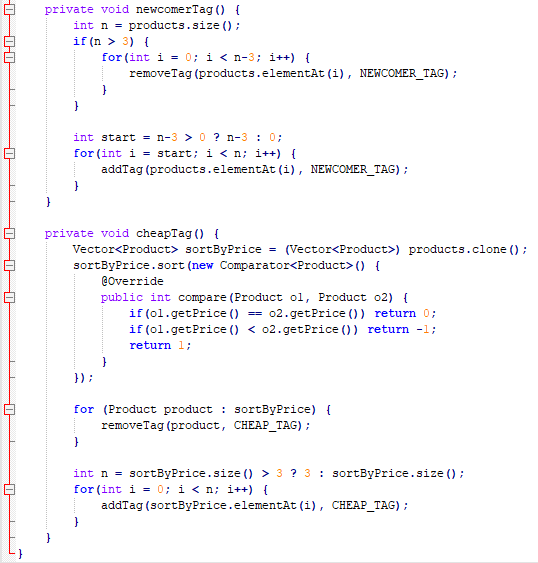
**SOLUSI Inappropriate Intimacy Code Smell:**



Catalog dan Product punya hubungan bidirectional yang tidak perlu, maka dilakukan Change Bidirectional Association to Unidirectional. Dengan cara menghapus field catalog di dalam Product.



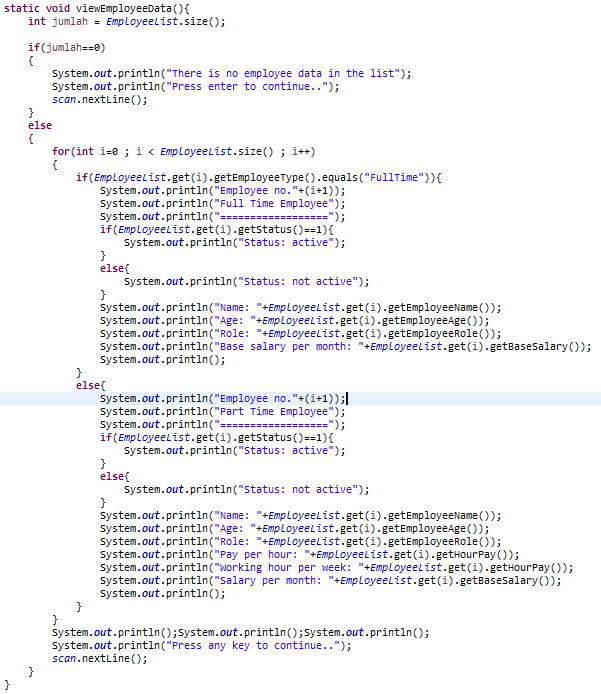




Semua method di class Catalog yang berhubungan dengan manipulasi tag access modifier-nya diubah menjadi private.

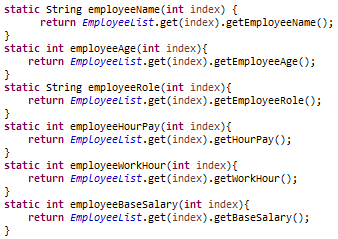
1. **Message Chains**

* Smell ini terjadi ketika ingin mengakses sebuah method, perlu dilakukan pemanggilan dari hasil return method lainnya sehingga membentuk rantai: obj.getDept().getSubDept().getHOD().getId();



Terdapat message chaining ketika ingin mengakses nama,umur, Jabatan, gaji per jam, total jam kerja per minggu, dan Gaji per bulan.

**SOLUSI Message Chains Code Smell:**



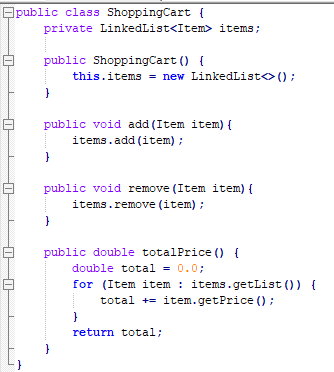
Dilakukan Hide Delegate. telah ditambahkan fungsi yang sebenarnya melakukan delegasi chaining. Hal ini dilakukan agar class tidak merasakan adanya chaining, disembunyikan di dalam sini.

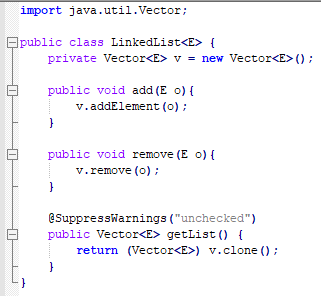


Hasilnya, seperti yang bisa dilihat diatas, message chain sudah tidak ada.

1. **Middle Man**

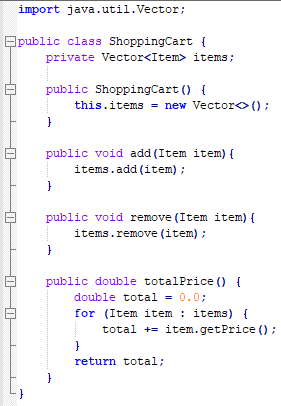
* Smell dimana class yang isinya hanya delegasi saja seperti Adapter, Proxy, Bridge, Facade atau Mediator, tidak ada behavior lain selain delegasi ke class lain.





ShoppingCart melakukan add pada LinkedList. Namun ternyata di dalam LinkedList, dia hanya delegasi ke java.util.Vector.

**SOLUSI Middle Man Code Smell:**



Hapus si Middle Man, yaitu LinkedList. Sekarang class ShoppingCart langsung mengakses java.util.Vector.