**1. Refactoring: Extract Method ✨**

Ini adalah refactoring pertama dan paling dasar yang bisa kita lakukan untuk merapikan kode yang terlalu panjang.

* Clue (Petunjuk/Code Smell 👃):

Metode main di file AplikasiResepJadul.java terlalu "gemuk" dan melakukan banyak hal. Blok if-else if untuk setiap pilihan menu (tambah, lihat, cari) membuat metode ini sangat panjang, sulit dibaca, dan sulit dipelihara. Jika Anda ingin mengubah cara kerja menu "Tambah Resep", Anda harus mencari dan mengedit di tengah-tengah metode yang besar.

* Bagian Spesifik yang Harus Direfactor:

Fokus pada seluruh isi dari setiap blok kondisional di dalam ***while (true)*** pada file ***BukuResepDigital.java***.

* + Blok if (pilihan.equals("1")) { ... }
  + Blok else if (pilihan.equals("2")) { ... }
  + Blok else if (pilihan.equals("3")) { ... }
* Solusi dan Penjelasan:

Kita "mengekstrak" atau mengeluarkan setiap blok logika tersebut ke dalam metodenya sendiri yang lebih kecil dan memiliki satu tujuan jelas.

* + Logika di dalam if (pilihan.equals("1")) dipindahkan ke metode baru bernama prosesTambahResep().
  + Logika di dalam else if (pilihan.equals("2")) dipindahkan ke metode prosesTampilkanSemuaResep().
  + Logika di dalam else if (pilihan.equals("3")) dipindahkan ke metode prosesCariResep().

Hasilnya, main method di file Main.java (setelah refactor) menjadi sangat bersih dan berfungsi seperti daftar isi, hanya memanggil metode yang sesuai berdasarkan input pengguna.

Java

// SEBELUM: main method terlalu panjang

if (pilihan.equals("1")) {

System.out.println("\n--- Menambah Resep Baru ---");

// ... (banyak baris kode lainnya) ...

}

// SESUDAH: main method menjadi ringkas

switch (pilihan) {

case "1": prosesTambahResep(); break; // Jauh lebih mudah dibaca!

}

**2. Refactoring: Extract Class 🏛️**

Ini adalah langkah besar untuk mengubah program dari gaya prosedural menjadi berorientasi objek.

* Clue (Petunjuk/Code Smell 👃):

Program kita tahu tentang konsep "resep", tetapi tidak ada "cetakan" atau **class** khusus untuk itu. Kita hanya menggunakan ***Map<String, Object>*** untuk merepresentasikan resep. Ini adalah code smell bernama Primitive Obsession, di mana kita menggunakan tipe data primitif/dasar untuk merepresentasikan ide yang kompleks. Selain itu, semua logika manajemen resep (menyimpan, mencari) tercampur di kelas utama.

* **Bagian Spesifik yang Harus Direfactor:**
  + Deklarasi ***static List<Map<String, Object>> bukuResep***. Ini adalah tanda bahwa ada kumpulan data yang perlu dikelola oleh kelasnya sendiri.
  + Setiap kali kita membuat ***new HashMap<String, Object>()*** untuk sebuah resep. Ini adalah tanda bahwa ada objek yang perlu dibuat dari kelasnya sendiri.
* Solusi dan Penjelasan:

Kita membuat dua kelas baru yang sangat penting:

* 1. ***Resep.java***: Kelas ini berfungsi sebagai "cetakan" untuk sebuah resep. Ia memiliki ***field*** yang jelas (judul, bahan, langkah) sehingga kita tidak lagi bergantung pada kunci String yang rawan salah ketik.
  2. ***BukuResep.java***: Kelas ini bertugas untuk mengelola koleksi objek ***Resep***. Semua logika seperti menambah resep baru dan mencari resep dipindahkan ke sini.

Dengan cara ini, kelas Main tidak perlu tahu bagaimana resep disimpan atau dikelola; ia hanya perlu "berbicara" dengan objek BukuResep.

**3. Refactoring: Encapsulate Field 🔒**

Setelah kita punya kelas baru, kita perlu melindungi datanya.

* Clue (Petunjuk/Code Smell 👃):

Pada kode awal, variabel ***bukuResep*** dideklarasikan sebagai ***static***. Ini menjadikannya variabel global yang bisa diakses dan diubah secara langsung dari mana saja. Ini sangat berbahaya karena tidak ada kontrol atas data tersebut.

* Bagian Spesifik yang Harus Direfactor:

Fokus pada deklarasi variabel koleksi resep:

***static List<Map<String, Object>> bukuResep = new ArrayList<>();***

* Solusi dan Penjelasan:

Setelah kita membuat kelas ***BukuResep***, kita memindahkan deklarasi koleksi resep ke dalamnya dan memberinya access modifier ***private***.

Java

// SEBELUM (di AplikasiResepJadul.java)

static List<Map<String, Object>> bukuResep = new ArrayList<>();

// SESUDAH (di BukuResep.java)

public class BukuResep {

private Map<String, Resep> daftarResep = new HashMap<>();

// ...

}

Sekarang, tidak ada kelas lain yang bisa mengakses daftarResep secara langsung. Jika kelas Main ingin menambahkan resep, ia *harus* melalui "pintu" yang sudah kita sediakan, yaitu metode publik tambahResep(). Ini membuat kode jauh lebih aman dan terstruktur.

**4. Refactoring: Move Method 🚚**

Memindahkan fungsi ke "rumah" yang lebih tepat.

* Clue (Petunjuk/Code Smell 👃):

Logika untuk menampilkan detail sebuah resep (mencetak judul, bahan, langkah dengan format tertentu) berada di dalam main method di AplikasiResepJadul.java. Seharusnya, kelas Main tidak perlu pusing memikirkan cara memformat tampilan resep. Objek Resep itu sendiri yang paling tahu cara menampilkan dirinya.

* Bagian Spesifik yang Harus Direfactor:

Seluruh blok kode di dalam yang tugasnya mencetak detail resep ke layar, dimulai dari 

* Solusi dan Penjelasan:

memindahkan seluruh blok kode pencetakan tersebut ke dalam sebuah metode baru bernama ***tampilkanDetail()*** di dalam kelas **Resep.java**.

Sekarang, dari kelas lain, kita hanya perlu memanggil resep.tampilkanDetail(). Jauh lebih bersih dan sesuai dengan prinsip bahwa objek bertanggung jawab atas dirinya sendiri.

**5. Refactoring: Rename... 🏷️**

Membuat nama lebih jelas dan sesuai standar.

* Clue (Petunjuk/Code Smell 👃):

Nama file/kelas utama adalah ***AplikasiResepJadul.java***. Nama ini memang deskriptif, tetapi dalam pengembangan aplikasi, kelas yang berisi ***main method*** sering kali diberi nama yang lebih umum dan standar seperti **Main** atau **App** untuk menandakan bahwa itu adalah titik masuk program.

* Bagian Spesifik yang Harus Direfactor:

Nama file dan deklarasi kelas: ***public class AplikasiResepJadul***.

* Solusi dan Penjelasan:

Kita mengganti nama kelas tersebut menjadi **Main**. Ini mungkin terlihat sepele, tetapi ini membantu pengembang lain untuk cepat mengerti struktur proyek Anda. **Main** secara universal dipahami sebagai tempat program dimulai.

Java

// SEBELUM

public class AplikasiResepJadul { ... }

// SESUDAH

public class Main { ... }

**6. Peningkatan Struktur (Hasil dari Refactoring Lainnya)**

* Clue (Petunjuk/Code Smell 👃):

Di kode awal, untuk mencari resep kita harus melakukan looping secara manual pada ArrayList. Ini adalah proses yang lambat (kompleksitas O(n)) dan kodenya harus ditulis berulang jika kita perlu mencari di tempat lain.

* Bagian Spesifik yang Terpengaruh:

Blok ***for (Map<String, Object> resep : bukuResep)*** di dalam ***BukuResepDigital.java***.

* Solusi dan Penjelasan:

Ini bukan satu teknik refactoring tunggal, melainkan manfaat besar yang kita dapatkan dari Extract Class (***BukuResep.java***). Karena sekarang kita punya kelas khusus untuk mengelola resep, kita bisa membuat keputusan cerdas tentang struktur data internalnya.

* + Kita mengganti ArrayList yang lambat untuk pencarian dengan HashMap.
  + HashMap menggunakan sistem key-value (dalam kasus kita, judul resep sebagai key). Ini memungkinkan kita menemukan resep secara instan tanpa perlu *looping* (kompleksitas O(1)).

Perubahan ini tersembunyi di dalam kelas BukuResep. Kelas Main tidak tahu dan tidak perlu tahu tentang perubahan ini. Ini adalah contoh bagus dari **enkapsulasi**, di mana kita bisa memperbaiki bagian internal sistem tanpa merusak bagian lainnya.