Nama : Fadhila Agil Permana NIM : 2211104006
Kelas : SE-06-01
Link Github : Repository Github

Refactoring Bad Smells

1. Long Method (`printInfo`)

Metode `printInfo` memiliki struktur panjang dengan banyak pengulangan kode (`System.out.println(...)`).
Solusi: Pisahkan logika ke metode kecil yang lebih modular.

---

2. Primitive Obsession (`genre`)

`Genre` menggunakan `int` untuk menyatakan tipe lagu dengan angka tetap.
Solusi: Ganti dengan `enum` agar lebih mudah dibaca dan dikelola.

```
public enum Genre {
    UNDEFINED, POP, ROCK, HIPHOP, RNB, JAZZ, INSTRUMENTALS, CLOWNCORE
}
```

---

3. Data Clump (`album` & `artist`)

`Album` dan `Artist` memiliki beberapa atribut yang berkelompok (`albumName`, `albumCoverURL`, `artistName`, dll.).
Solusi: Pisahkan ke dalam kelas terpisah (`Album` dan `Artist`).

```
public class Album {
    private String name;
    private String coverURL;
}

public class Artist {
    private String name;
    private String alias;
    private String imageURL;
}
```

---

4. Feature Envy (`setArtist` & `setAlbum`)

Metode `setArtist` dan `setAlbum` mengatur banyak atribut yang lebih cocok ditangani dalam kelas terpisah (`Album` dan `Artist`).

---

5. Null Reference Risk

Tidak ada inisialisasi default untuk `albumName`, `artistName`, dll.
Jika `printInfo` dipanggil sebelum `setArtist` atau `setAlbum`, bisa terjadi `NullPointerException`.
Solusi: Tambahkan inisialisasi default.

```
private String artistName = "";
private String albumName = "";
```

---

6. Magic Numbers (`detailLevel`)

`detailLevel` menggunakan angka (`0`, `1`, `2`, `3`) tanpa penjelasan yang eksplisit.
Solusi: Gunakan `enum` untuk meningkatkan keterbacaan.

```
public enum DetailLevel {
    SONG_ONLY, SONG_AND_ARTIST, SONG_AND_ALBUM, FULL_DETAILS
}
```

---

7. Perubahan Refactoring

a. Menggunakan `enum` untuk Genre

Sebelumnya: Genre direpresentasikan sebagai angka (`int`), seperti `1 = pop`, `2 = rock`, dst.
Masalah: Membuat kode sulit dibaca dan rentan terhadap kesalahan.
Sekarang: Menggunakan `enum` untuk merepresentasikan genre secara lebih jelas.

```
public enum Genre {
    UNDEFINED, POP, ROCK, HIP_HOP, RNB, JAZZ, INSTRUMENTALS, CLOWNCORE
}
```

Keuntungan:
- Lebih mudah dipahami, misalnya `Genre.POP` daripada `1`.

---

b. Memisahkan `Album` dan `Artist` ke Kelas Terpisah

Sebelumnya: Atribut album (`albumName`, `albumCoverURL`) dan artist (`artistName`, `artistAlias`, `artistImageURL`) langsung disimpan dalam kelas `Song`.
Masalah: Jika ingin menambahkan informasi tambahan pada album atau

artist, kode `Song` menjadi terlalu besar dan tidak terorganisir.
Sekarang: Dibuat kelas `Album` dan `Artist` sendiri.

```
public class Album {
    private String name;
    private String coverURL;
}

public class Artist {
    private String name;
    private String alias;
    private String imageURL;
}
```

Keuntungan:
- `Song` hanya menyimpan referensi ke objek `Album` dan `Artist`, sehingga lebih rapi.
- Memudahkan pengelolaan informasi album dan artist di masa depan.

---

c. Menggunakan `enum` untuk Detail Level

Sebelumnya: Detail level menggunakan angka (`int`), misalnya `0 = song info only`, `1 = song + artist info`, dst.
Masalah: Sulit diingat dan rawan kesalahan jika tidak ada dokumentasi.
Sekarang: Dibuat `enum` seperti ini:

```
public enum DetailLevel {
    SONG_ONLY, SONG_AND_ARTIST, SONG_AND_ALBUM, FULL_INFO
}
```

Keuntungan:
- Lebih jelas, misalnya `DetailLevel.SONG_AND_ARTIST` daripada `1`.

---

d. Memecah `printInfo` ke Metode Kecil

Sebelumnya: `printInfo()` memiliki banyak `if-else`, sehingga sulit dibaca dan dipelihara.
Sekarang: Dibagi menjadi beberapa metode kecil seperti:

```
private void printBasicInfo() { ... }
private void printArtistInfo() { ... }
private void printAlbumInfo() { ... }
```

Metode `printInfo()` hanya memanggil metode ini berdasarkan `DetailLevel`, sehingga lebih bersih dan mudah dipahami.

---

e. Menggunakan Null Check

Sebelumnya: Langsung mengakses atribut seperti `if (!artistName.equals(""))`, yang bisa menyebabkan error jika `artistName` bernilai `null`.
Sekarang: Menggunakan null check sebelum mengakses atribut:

```
if (artistName != null && !artistName.isEmpty()) { ... }
```

Keuntungan:
- Mengurangi risiko aplikasi crash karena `NullPointerException`.

Kode Sebelum Refactoring

```
// Kode yang lama
package assignment;

public class Song {

    private String id;
    private String title;
    private String releaseYear;
    private String musicFileURL;
    private int genre;

    private String albumName;
    private String albumCoverURL;

    private String artistName;
    private String artistAlias;
    private String artistImageURL;

    public Song(String id, String title, String releaseYear, String musicFileURL) {
        this.id = id;
        this.title = title;
        this.releaseYear = releaseYear;
        this.musicFileURL = musicFileURL;
    }

    public void setAlbum(String albumName, String albumCoverURL) {
        this.albumName = albumName;
        this.albumCoverURL = albumCoverURL;
    }

    public void setArtist(String artistName, String artistAlias, String artistImageURL) {
        this.artistName = artistName;
        this.artistAlias = artistAlias;
```

```
    this.artistImageURL = artistImageURL;
}

/**
 * Set the genre of this song
 *
 * 0 = undefined
 * 1 = pop
 * 2 = rock
 * 3 = hip hop
 * 4 = RnB
 * 5 = jazz
 * 6 = instrumentals
 * 7 = clowncore
 *
 * @param genre
 */
public void setGenre(int genre) {
    this.genre = genre;
}

/**
 * Print info of the song based on desired detail level
 *
 * 0 = song info only
 * 1 = song info and artist info
 * 2 = song info and album info
 * 3 = song, artist, and album info
 *
 * @param genre
 */
public void printInfo(int detailLevel) {
    if (detailLevel == 0) {
        System.out.println("song title: " + title);
        System.out.println("release year: " + releaseYear);
        if (genre > 0) {
            System.out.println("genre: " + genre);
        }
    } else if (detailLevel == 1) {
        System.out.println("song title: " + title);
        System.out.println("release year: " + releaseYear);
        if (genre > 0) {
            System.out.println("genre: " + genre);
        }
        if (!artistName.equals("")) {
            System.out.println("artist name: " + artistName);
```

```
            }
            if (!artistAlias.equals("")) {
                System.out.println("artist also known as: " + artistAlias);
            }
        } else if (detailLevel == 2) {
            System.out.println("song title: " + title);
            System.out.println("release year: " + releaseYear);
            if (genre > 0) {
                System.out.println("genre: " + genre);
            }
            if (!albumName.equals("")) {
                System.out.println("album title: " + albumName);
            }
        } else if (detailLevel == 3) {
            System.out.println("song title: " + title);
            System.out.println("release year: " + releaseYear);
            if (genre > 0) {
                System.out.println("genre: " + genre);
            }
            if (!artistName.equals("")) {
                System.out.println("artist name: " + artistName);
            }
            if (!artistAlias.equals("")) {
                System.out.println("artist also known as: " + artistAlias);
            }
            if (!albumName.equals("")) {
                System.out.println("album title: " + albumName);
            }
        }
    }

}
```

Kode Setelah Refactoring

```
// Kode Refactored

package assignment;

public enum Genre {
    UNDEFINED, POP, ROCK, HIPHOP, RNB, JAZZ, INSTRUMENTALS, CLOWNCORE;
}

class Album {
```

```
    private String name;
    private String coverURL;

    public Album(String name, String coverURL) {
        this.name = name;
        this.coverURL = coverURL;
    }

    public String getName() {
        return name;
    }
}

class Artist {
    private String name;
    private String alias;
    private String imageURL;

    public Artist(String name, String alias, String imageURL) {
        this.name = name;
        this.alias = alias;
        this.imageURL = imageURL;
    }

    public String getName() {
        return name;
    }

    public String getAlias() {
        return alias;
    }
}

enum DetailLevel {
    SONG_ONLY, SONG_AND_ARTIST, SONG_AND_ALBUM, FULL_DETAILS;
}

public class Song {
    private String id;
    private String title;
    private String releaseYear;
    private String musicFileURL;
    private Genre genre = Genre.UNDEFINED;
    private Album album;
    private Artist artist;
```

```java
public Song(String id, String title, String releaseYear, String musicFileURL) {
    this.id = id;
    this.title = title;
    this.releaseYear = releaseYear;
    this.musicFileURL = musicFileURL;
}

public void setAlbum(Album album) {
    this.album = album;
}

public void setArtist(Artist artist) {
    this.artist = artist;
}

public void setGenre(Genre genre) {
    this.genre = genre;
}

public void printInfo(DetailLevel detailLevel) {
    printSongInfo();
    if (detailLevel.ordinal() >= 1)
        printArtistInfo();
    if (detailLevel.ordinal() >= 2)
        printAlbumInfo();
}

private void printSongInfo() {
    System.out.println("Song Title: " + title);
    System.out.println("Release Year: " + releaseYear);
    if (genre != Genre.UNDEFINED) {
        System.out.println("Genre: " + genre);
    }
}

private void printArtistInfo() {
    if (artist != null) {
        System.out.println("Artist Name: " + artist.getName());
        if (!artist.getAlias().isEmpty()) {
            System.out.println("Also known as: " + artist.getAlias());
        }
    }
}

private void printAlbumInfo() {
    if (album != null) {
```

```
                System.out.println("Album Title: " + album.getName());
            }
        }
    }
```