

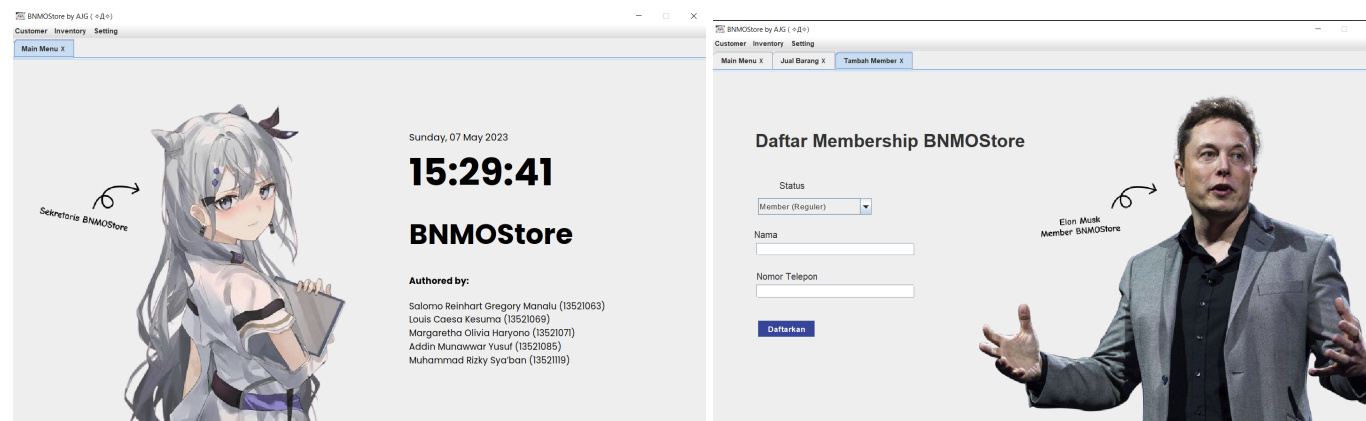
Kelompok : AJG

1. 13521063 / Salomo Reinhart Gregory Manalu
2. 13521069 / Louis Caesa Kesuma
3. 13521071 / Margaretha Olivia Haryono
4. 13521085 / Addin Munawwar Yusuf
5. 13521119 / Muhammad Rizky Sya'ban

Asisten Pembimbing : 13519064 / Aditya Bimawan

1. Deskripsi Umum Aplikasi

Aplikasi yang kami bangun bernama BNMOSTore, yang merupakan aplikasi sistem *Point of Sales* (POS) yang membantu sebuah toko dalam melakukan dan mencatat transaksi yang terkait dengan usaha mereka. Program ini memiliki fitur-fitur penting seperti manajemen inventaris barang serta manajemen transaksi. Terdapat juga fitur *membership* untuk memberikan *reward* kepada pelanggan yang setia dan fitur pembuatan laporan untuk memudahkan toko dalam melakukan evaluasi. Selain itu, program yang kami buat juga bersifat *extensible* karena mendukung plugin, sehingga pengguna dapat dengan mudah menambahkan fungsionalitas program sesuai dengan kebutuhan toko. Dengan fitur-fitur tersebut, program BNMOSTore dapat membantu toko untuk mempercepat proses transaksi dan memudahkan pengelolaan usaha.



Gambar 1.1. Tampilan Main Menu dan Halaman Pendaftaran Member BNMOSTore

2. Kakas GUI: Java Swing

Kakas GUI yang digunakan dalam pembuatan aplikasi BNMOSTore adalah **Java Swing**. Java Swing adalah sebuah library yang digunakan dalam pengembangan aplikasi desktop menggunakan bahasa pemrograman Java. Swing menyediakan komponen-komponen antarmuka grafis (GUI) dasar yang dapat digunakan untuk membangun aplikasi, seperti *window*, *button* label, dan lain-lain.

Dalam implementasinya, aplikasi memiliki sebuah *window* utama yang menggunakan komponen `JFrame` dari Java Swing. Window memiliki 2 komponen utama, yaitu kumpulan tab/halaman yang terbuka (menggunakan `JTabbedPane` dan `JPanel`), serta sebuah `JMenuBar` yang dapat digunakan untuk membuka tab baru ataupun mengubah pengaturan tertentu. Konten dari masing-masing page juga memanfaatkan komponen-komponen dari Java Swing, seperti:

- `JPanel`, digunakan sebagai *container* untuk mengelompokkan dan mengatur tata letak dari komponen-komponen Swing lainnya.
- `JLabel`, digunakan untuk menampilkan teks ataupun gambar.
- `JBUTTON`, digunakan sebagai tombol untuk menerima input klik dari pengguna dan melakukan respon berupa suatu aksi pada aplikasi.
- `JTextField`, digunakan untuk menerima input teks dari pengguna, seperti nama, deskripsi, atau data lainnya.
- `JComboBox`, digunakan untuk menerima pilihan dari pengguna yang dipilih dari sebuah kumpulan opsi dalam bentuk dropdown.
- Dan lain-lain.

Pada program main, Kelas Aplikasi (yang merupakan turunan dari `JFrame`) di-*construct*, dan di-*set* agar frame terlihat, sehingga aplikasi BNMOSTore dapat dijalankan.

3. Plugin & Class Loader

Pada aplikasi BNMOSTore, class loader digunakan untuk menambahkan fitur plugin dan menjalankan plugin tersebut saat runtime. Cara kerja dari class loader ini adalah dengan memanfaatkan **Java Reflection API**. Java Reflection API, memungkinkan kita untuk dapat menginspeksi, memanipulasi, hingga menjalankan kode Java secara dinamis saat runtime.

Class loader memanfaatkan hal ini dengan menerima .jar file saat runtime, kemudian menginspeksi isi dari jar file tersebut. Jika jar file tersebut memiliki kelas yang mengimplementasi BasePlugin (interface plugin dasar untuk aplikasi BNMOSTore), fungsi onLoad yang merupakan fungsi dari interface BasePlugin akan diinvokasi saat runtime.

Agar plugin dapat melakukan ekstensi terhadap aplikasi BNMOSTore, kami menyediakan 2 API yang dapat digunakan untuk melakukan aksi pada aplikasi, yaitu AppService (untuk melakukan aksi pada GUI) dan DataService (untuk melakukan aplikasi pada DataStore). API atau Interface ini akan dipass melalui fungsi onLoad, sehingga plugin dapat melakukan aksi sesuai dengan apa yang diinginkan. Berikut adalah desain interface BasePlugin.

```
package Plugins;

/* Imports */
import UI.IApp;
import DataStore.DataStore;

public interface BasePlugin {
    /**
     * Method that will run when the plugin is loaded
     * @param appService
     * @param dataService
     */
    public void onLoad(IApp appService, DataStore dataService);
}
```

Snapshot 3.1. Desain plugin dasar dari aplikasi

Plugin yang di-*load* pada aplikasi harus sudah dalam berbentuk jar file. Plugin ini di-*load* pada saat *runtime* dengan menggunakan suatu `ClassLoader`, yang memanfaatkan Java Reflection API, sehingga memungkinkan kelas baru dari JAR dapat dimasukkan ke aplikasi. Implementasi dari kelas tersebut secara garis besar adalah sebagai berikut.

```
Import ...

public class CustomClassLoader extends ClassLoader {

    private final String jarPath;

    public CustomClassLoader(String jarPath) {
        this.jarPath = jarPath;
    }

    protected List<Class<?>> findPluginClasses(String className) throws ClassNotFoundException {

        // Loading setiap data yang ada di JARFile, dan return kelas implementasi BasePlugin
        // Memanfaatkan java reflection API
    }

    public void load() {
        CustomClassLoader classLoader = new CustomClassLoader(this.jarPath);

        try {
            List<Class<?>> plugins = classLoader.findPluginClasses("Plugins.BasePlugin");

            for (Class<?> plugin : plugins) {
                // Invokasi onLoad untuk setiap kelas implementasi plugin
                Object pluginInstance = plugin.getDeclaredConstructor().newInstance();
                Class<?>[] parameterTypes = { IApp.class, DataStore.class };
                Method method = plugin.getMethod("onLoad", parameterTypes);
                IApp appService = App.getInstance();
                DataStore dataService = DataStore.getInstance();

                method.invoke(pluginInstance, appService, dataService);
            }
        }
    }
}
```

```

        JOptionPane.showMessageDialog(null, "Plugin loaded successfully!");
    }

    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

Snapshot 3.2. Kelas Loader untuk JAR File Plugin

Untuk implementasi plugin, plugin dapat memanfaatkan interface AppService untuk memodifikasi GUI maupun DataService untuk mengakses dan menulis ke Data Store. Salah satu implementasi sederhananya adalah sebagai berikut.

```

public class ChartPlugin1 implements BasePlugin {
    @Override
    public void onLoad(IApp appService, DataStore dataService) {
        JMenu menu = new JMenu("ChartPlugin1"); // Membuat menu baru
        JMenuItem menuItem = new JMenuItem("Statistik Toko");

        menuItem.addActionListener(e -> {
            appService.addTab("Statistik Toko", new ChartPlugin1Page(appService, dataService)); // Membuka page
        });

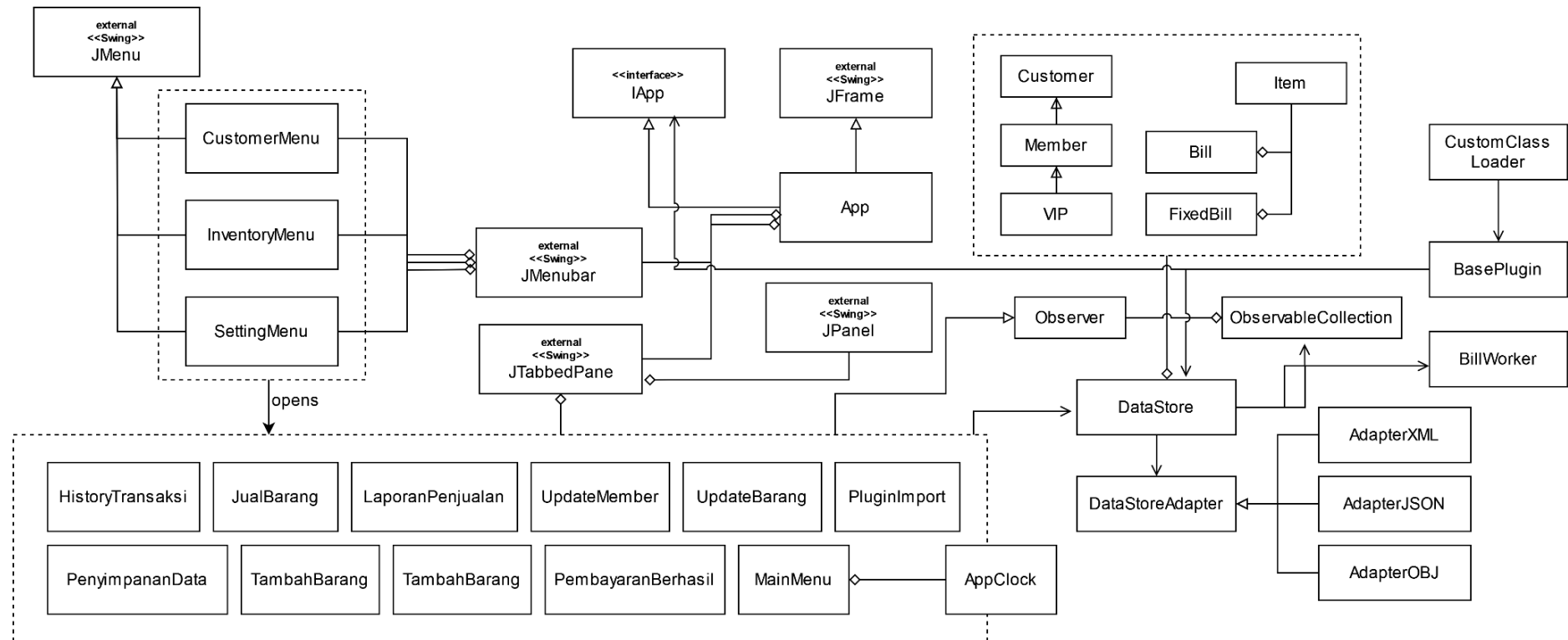
        menu.add(menuItem);
        appService.addMenu(menu);
    }
}

```

Snapshot 3.3. Contoh Implementasi Plugin Chart

Agar plugin dapat menggunakan kelas-kelas dari aplikasi BNMOStore, proyek dari plugin harus menambahkan BNMOStore ke dalam *dependency*-nya.

4. Class Diagram



*Gambar 4.1. Diagram Kelas Aplikasi BNMOStore (*beberapa kelas tidak di-include)*

Entry Point dari aplikasi ini adalah Kelas App. Kelas App memiliki dua komponen utama, yaitu MenuBar dan Panel Tab. Pada awalnya, Tab yang terbuka adalah Tab Kelas MainMenu. Menu yang terdapat di menu bar dapat digunakan untuk melakukan navigasi ataupun membuka tab baru. Hal tersebut dapat terjadi karena item yang terdapat di menu sudah di-*bind* dengan EventListener, sehingga ketika item di klik, EventListener akan membuka tab baru.

Setiap tab memiliki akses ke DataStore melalui instance yang sama. Hal ini dapat terjadi karena DataStore merupakan kelas singleton. Kelas DataStore memiliki kemampuan untuk menulis dan membaca data ke dalam file yang bersesuaian dengan adapter yang ada. Untuk menulis data, tab cukup memanggil method yang berawalan write dari DataStore. Sementara itu, untuk membaca, tab cukup memanggil getter dari records yang ada di DataStore. Selain itu, tab juga bisa menambahkan observer terhadap records yang ada di DataStore, sehingga tab selalu sadar terhadap perubahan yang ada di aplikasi.

DataStore juga tidak menulis data ke file secara langsung, melainkan melalui suatu adapter untuk extension file tertentu. DataStore juga menjalankan rutin BillWorker yang melakukan autosave setiap beberapa detik untuk record bill. Selain itu, plugin di load melalui CustomClassLoader yang akan mencari kelas di file jar yang mengimplementasi BasePlugin, dan memanggil method onLoad dari kelas tersebut. CustomClassLoader dipanggil pada tab PluginImport.

5. Konsep OOP

Berikut ini adalah konsep-konsep OOP yang digunakan pada aplikasi BNMOSTore.

5.1. Inheritance

Konsep Inheritance digunakan pada:

- Kelas VIP (src/main/java/Entity/VIP.java) merupakan anak dari Kelas Member (src/main/java/Entity/Member.java), Kelas Member merupakan anak dari Kelas Customer (src/main/Entity/Customer.java).
- Kelas-kelas Page (src/main/java/UI/Page/*.java) merupakan anak dari Kelas JPanel (javax.swing.JPanel).
- Kelas-kelas AppMenu (src/main/java/UI/Component/AppMenu/*.java) merupakan anak dari kelas JMenu (javax.swing.JMenu).
- Kelas App (src/main/UI/App.java) merupakan anak dari kelas JFrame (javax.swing.JFrame).
- Kelas CustomClassLoader (src/main/java/Plugins/CustomClassLoader.java) merupakan anak dari kelas ClassLoader (java.lang.ClassLoader).

5.2. Composition

Konsep Composition digunakan pada:

- Kelas `DataStore` (`src/main/java/DataStore/DataStore.java`) memiliki banyak kelas-kelas dari `Entity` (`src/main/java/Entity/*.java`) dan sebuah `DataStoreAdapter` (`src/main/java/DataStore/Adapter/DataStoreAdapter.java`).
- Kelas `FixedBill` dan `Bill` (`src/main/java/Entity/FixedBill | Bill.java`) memiliki banyak kelas `Item` (`src/main/java/Entity/Item.java`).
- Kelas `AppActionListener` (`src/main/UI/AppActionListener.java`) memiliki banyak kelas `JBUTTON` (`javax.swing.JButton`) dan tepat satu interface `IApp` (`src/main/java/UI/IApp.java`).
- Beberapa kelas `Page` (`src/main/java/UI/Page/*.java`) yang memiliki form, memiliki satu atau lebih kelas `JTextField` (`javax.swing.JTextField`) dan satu atau lebih kelas `JComboBox` (`javax.swing.JComboBox`).

5.3. Interface

Konsep Interface digunakan pada:

- Interface `IApp` (`src/main/UI/IApp.java`) untuk API untuk GUI pada aplikasi `BNMOSStore`.
- Interface `DataStoreAdapter` (`src/main/java/DataStore/Adapter/DataStoreAdapter.java`) untuk adapter pattern pada desain `DataStore`.
- Interface `Observer` (`src/main/java/Utils/Collection/Observer.java`) untuk observer pattern pada desain `ObservableCollection`.
- Interface `Runnable` (`java.lang.Runnable`) untuk mendefinisikan task untuk sebuah `Thread`, pada `AppClock` (`src/main/java/UI/Component/AppClock.java`) dan `BillWorker` (`src/main/java/DataStore/Thread/BillWorker.java`).
- Interface `JsonSerializer` (`src/main/java/`) untuk melakukan serialisasi objek java ke JSON dengan kondisi khusus.

5.4. Method Overriding dan Method Overloading

Konsep method overriding dan method overloading digunakan pada:

- Method overloading pada constructor kelas `Item`, `FixedBill`, dan `Customer` (`src/main/java/Entity/*.java`).
- Method overloading pada method `getActiveMembers` di `DataStore` (`src/main/java/DataStore/DataStore.java`).
- Method overriding pada setiap kelas yang mengimplementasi interface (pada bagian sebelumnya).

5.5. Polymorphism

Konsep polymorphism digunakan pada:

- Kelas `DataStoreAdapter` (`src/main/java/DataStore/Adapter/DataStoreAdapter.java`) pada `DataStore` (`src/main/java/DataStore/DataStore.java`) memanggil method `read` dan `write` yang sama, akan tetapi method implementasi yang dipanggil berbeda-beda (bisa menggunakan kelas `AdapterJSON`, `AdapterXML`, ataupun `AdapterOBJ` dari (`src/main/java/DataStore/Adapter/*.java`)).
- Method `update` pada kelas `Observer` (`src/main/java/Utils/Collection/Observer.java`) memanggil prosedur `update` yang berbeda-beda.
- `JMenuBar` pada prosedur `init` di Kelas `App` (`src/main/java/UI/App.java`) dapat menerima implementasi `JMenu` yang berbeda-beda (`CustomerMenu`, `InventoryMenu`, `SettingMenu` pada (`src/main/java/UI/Component/AppMenu/*.java`)).
- Method `load` pada kelas `CustomClassLoader` (`src/main/java/Plugins/CustomClassLoader.java`) menginvokasi method `onLoad` dari Kelas `BasePlugin` (`src/main/java/Plugins/BasePlugin.java`) dengan implementasi yang berbeda-beda.

5.6. Java API Collection

Java API Collection digunakan pada:

- Kelas `DataStore` (`src/main/java/DataStore/DataStore.java`) memiliki `List of Entity` (`src/main/java/Entity/*.java`).
- Method `getActiveMembers` memanfaatkan java API stream filter.
- Kelas `ObservableCollection` (`src/main/java/Utils/Collection/ObservableCollection.java`) memanfaatkan `ArrayList`.
- Kelas `FixedBill` dan `Bill` (`src/main/java/Entity/*.java`) memanfaatkan `LinkedList` (`java.util.LinkedList`). `LinkedList` dipilih karena `Bill` sering mengalami `insert` dan `remove`.
- Kelas `ChartPlugin2Page` (`plugins/implementation/chart2/src/main/java/ChartPlugin2Page.java`) menggunakan `HashMap` (`java.util.HashMap`) untuk meng-enumerasi nama kategori dan jumlah item terjual.

5.7. SOLID

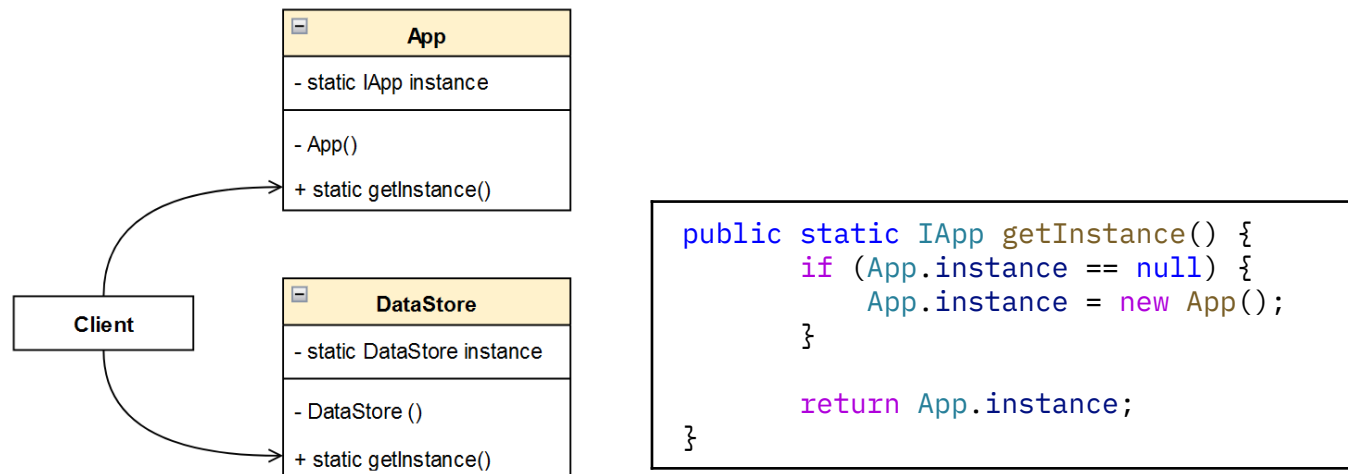
SOLID pada BNMOStore:

- S: `DataStore` (`src/main/java/DataStore/DataStore.java`) hanya mengurus bagian database saja, tidak mengatur GUI dari aplikasi.
- S: `Bill` (`src/main/java/Entity/Bill`) hanya mengurus bagian `Bill` saja, tidak mengatur GUI ataupun database dari aplikasi.
- S: `Item` (`src/main/java/Entity/Item`) hanya mengurus bagian `Item` saja, tidak mengatur GUI ataupun database dari aplikasi.
- O: Dalam menyediakan `BasePlugin` (`src/main/java/Plugins/BasePlugin`), aplikasi tidak mengizinkan adanya modifikasi secara `hardcode`, melainkan melakukan ekstensi melalui `service` ataupun `interface` yang ada.

- O: Dalam menyediakan DataService (src/main/java/DataStore/DataService), aplikasi hanya mengizinkan untuk melakukan ekstensi melalui interface atau service yang ada.
- O: Dalam menyediakan DataAdapter (src/main/java/DataStore/Adapter/DataStoreAdapter), aplikasi hanya mengizinkan untuk melakukan ekstensi melalui interface atau service yang ada.
- L: Kelas Member (src/main/java/Entity/Member.java) dapat mensubstitusikan Customer (src/main/java/Entity/Customer.java), begitu juga VIP ke Member (src/main/java/Entity/VIP.java).
- L: DataStoreAdapter (src/main/java/DataStore/Adapter/DataStoreAdapter.java) pada DataStore (src/main/java/DataStore/DataStore.java) dapat ditukar-tukar dengan kelas-kelas implementasinya (AdapterJSON, AdapterXML, dan AdapterOBJ pada (src/main/java/DataStore/Adapter/*.java)).
- L: BasePlugin (src/main/java/Plugins/BasePlugin) dapat ditukar-tukar dengan kelas-kelas implementasinya (ChartPlugin1, ChartPlugin2, SystemPlugin1, dan SystemPlugin2 pada (plugins\implementation)).
- I: Kelas Member (src/main/java/Entity/Member.java) memang menggunakan seluruh fungsionalitas Kelas Customer (src/main/java/Entity/Customer.java), begitu juga Kelas VIP (src/main/java/Entity/VIP.java) ke Kelas Member.
- I: interface DataStore Adapter hanya memiliki fungsionalitas untuk membaca/menulis data dari/ke data store.
- D: Kelas lain tidak dapat mengakses kelas App (src/main/java/UI/App.java) secara langsung melainkan melalui abstraksi interfacenya yaitu IApp (src/main/java/UI/IApp.java).
- D: Kelas DataStore menggunakan abstraksi dari AdapterJSON (src/main/java/DataStore/Adapter/AdapterJSON.java), yaitu interface DataStoreAdapter (src/main/java/DataStore/Adapter/DataStoreAdapter.java).

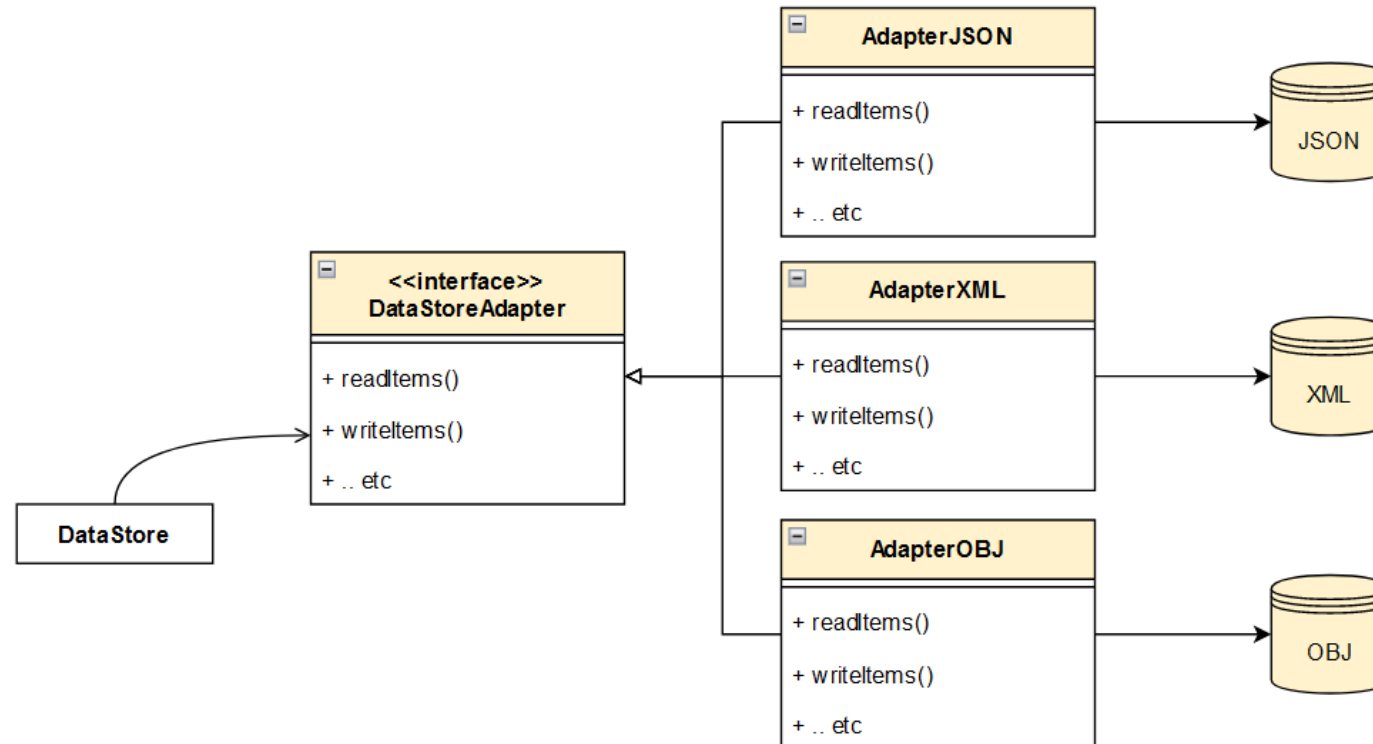
5.8. Design Pattern

Design Pattern yang Digunakan di BNMOStore:



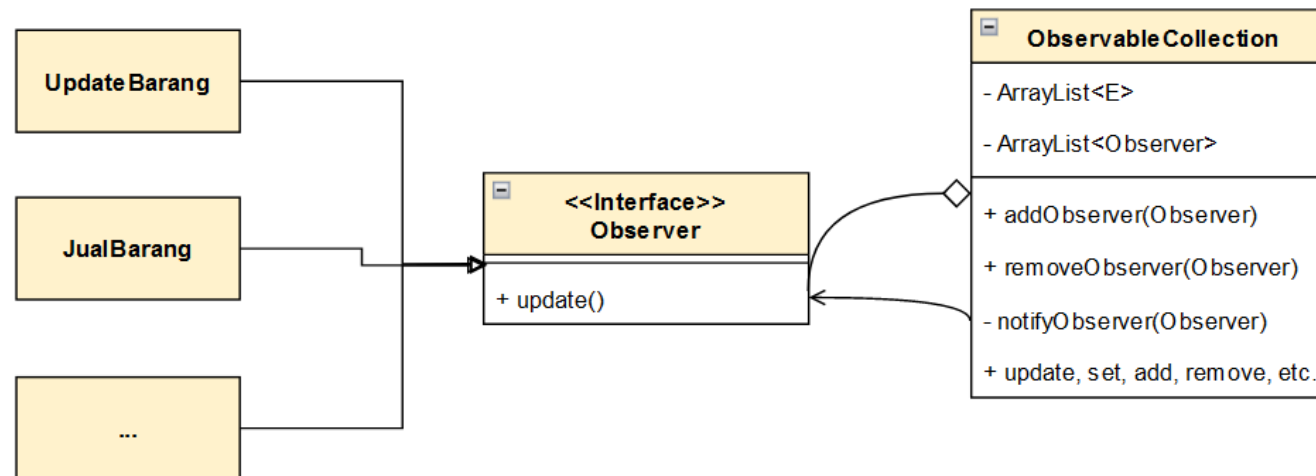
Gambar 5.8.1 Singleton Pattern pada Kelas App dan DataStore

- **Singleton Pattern:** Kelas App (src/main/java/UI/App.java) dan DataStore (src/main/java/DataStore/DataStore.java) sebagai kelas singleton, karena hanya boleh ada satu instance yang aktif dari kelas tersebut selama aplikasi berjalan.



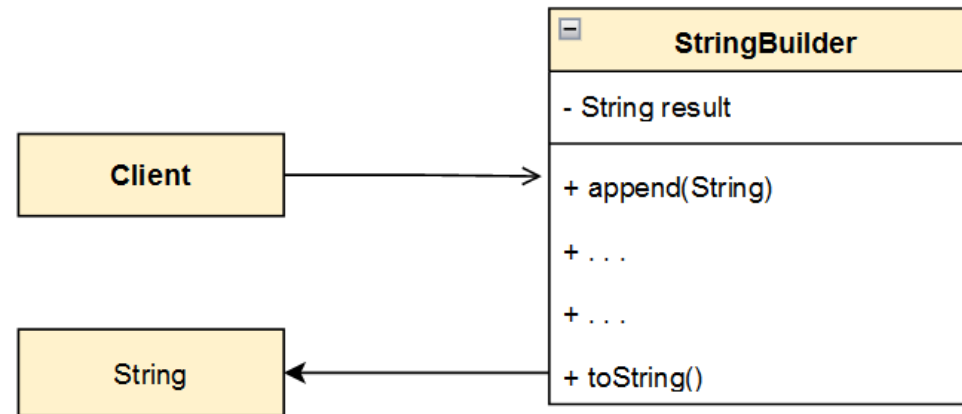
Gambar 5.8.2 Adapter Pattern pada Kelas DataStoreAdapter dan DataStore

- **Adapter Pattern:** Kelas DataStoreAdapter (src/main/java/DataStore/Adapter/DataStoreAdapter.java) sebagai kelas adapter, yang memiliki client DataStore (src/main/java/DataStore/DataStore.java). Digunakan agar objek ataupun kelas dari client dapat di-format ke atau konversi ke jenis-jenis yang berbeda dan dipakai sesuai kebutuhan.



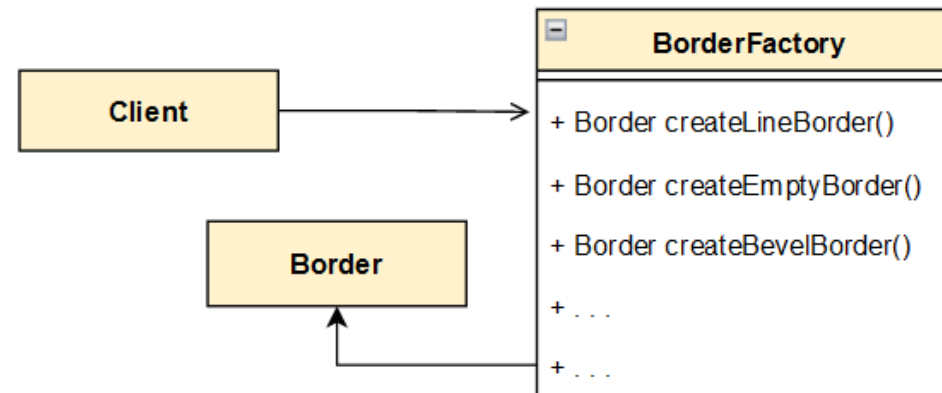
Gambar 5.8.3 Observer Pattern pada Kelas ObservableCollection and Observer

- **Observer Pattern:** Kelas ObservableCollection (src/main/java/Utils/Collection/ObservableCollection.java) sebagai kelas Publisher, dan kelas Observer (src/main/java/Utils/Collection/Observer.java) sebagai subscriber. Digunakan agar ketika terjadi perubahan pada suatu data, komponen GUI dapat merender ulang untuk menyesuaikan data terbaru.



Gambar 5.8.4 Builder Pattern pada Kelas String Builder

- **Builder Pattern:** Builder pattern digunakan pada aplikasi pada pembuatan string dengan menggunakan kelas bawaan java, yaitu StringBuilder. Pattern ini digunakan, khususnya pada string untuk mengurangi garbage collecting yang terjadi ketika kita menggunakan operator tambah pada string, karena operasi tersebut menyebabkan string di-reconstruct berulang kali. Maka dari itu, StringBuilder digunakan untuk mengurangi hal tersebut. Pada kelas lain, yaitu GSON (src/main/java/DataStore/Adapter/AdapterJSON.java) (Objek library untuk load JSON), Builder pattern digunakan untuk membuat kelas JSON, dengan tambahan *build* component, yaitu JsonMemberSerializer dan JsonMember Deserializer.



Gambar 5.8.5 Factory Pattern pada Kelas String Builder

- **Factory Pattern:** Factory pattern digunakan pada kelas BorderFactory dari package javax.swing. BorderFactory menyediakan implementasi-implementasi dari kelas Border yang dapat digunakan untuk menjadi Border pada komponen-komponen GUI yang ada di aplikasi.

5.9. Reflection

Java Reflection digunakan pada:

- Kelas CustomClassLoader (src/main/java/Plugins/CustomClassLoader.java) untuk me-load kelas-kelas dari JAR dan menjalankan onLoad dari kelas yang mengimplementasikan BasePlugin pada setiap methodnya.
- Kelas AdapterJSON (src/main/java/DataStore/Adapter/AdapterJSON.java) mengimplementasi Java Reflection (melalui com.google.gson.reflect.*) untuk resolving type ketika loading data dari json.

-

5.10. Threading

Threading digunakan pada:

- Kelas `AppClock` mengimplementasikan `Runnable` (`src/main/java/UI/Component/AppClock.java`) untuk menjalankan jam sepanjang aplikasi berjalan tanpa membloking proses lain pada aplikasi. Thread `AppClock` sendiri dijalankan pada `MainMenu` (`src/main/java/UI/Page/MainMenu.java`)
- Kelas `BillWorker` mengimplementasikan `Runnable` (`src/main/java/UI/DataStore/Thread/BillWorker.java`) untuk menjalankan autosave bill setiap beberapa detik sekali tanpa membloking proses lain pada aplikasi. Thread `BillWorker` sendiri dijalankan pada Kelas `DataStore` (`src/main/java/DataStore/DataStore.java`)
- Print laporan

6. Bonus Yang dikerjakan

6.1. Annotation

6.1.1. Lombok

Lombok digunakan pada:

- Semua entity class yaitu `Item`, `Member`, `VIP`, `Customer`, `Bill`, `FixedBill`.
- `Datastore Adapter`
- `Config`

6.1.2. Null Safety dengan menggunakan `org.jetbrains.annotations.NotNull`

Digunakan pada:

- Semua entity class yaitu `Item`, `Member`, `VIP`, `Customer`, `Bill`, `FixedBill`.

7. Pembagian Tugas

- 13521063 / Salomo Reinhart Gregory Manalu
 - Page Riwayat Transaksi
 - Page Laporan Penjualan
 - Plugins
 - Class Loader (Plugin Loader)
 - Print PDF Function
- 13521069 / Louis Caesa Kesuma
 - Page JualBarang
 - Page PluginImport
 - Page PembayaranBerhasil
 - Kelas Printer
 - Inisialisasi Entity
- 13521071 / Margaretha Olivia Haryono
 - Page Tambah Barang
 - Page Update Barang
 - Page Riwayat Transaksi
 - Page Laporan Penjualan
 - Menghubungkan Data Store dengan page barang dan member
- 13521085 / Addin Munawwar Yusuf
 - Plugins
 - Page Main Menu
 - Kelas DataStore
 - Interface DataStore Adapter
 - Adapter JSON dan OBJ
 - Kelas App
 - BillWorker
- 13521119 / Muhammad Rizky Sya'ban

- Page Tambah Member
- Page Update Member
- Adapter XML, JSON, OBJ
- Page Plugin
- Mekanisme Penyimpanan data serta paganya
- Change dir dan ext database
- Config DataStore

8. Foto Kelompok



LAMPIRAN

Kode Kelompok : AJG

Nama Kelompok : AkuJagonyaGolang

1. 13521063 / Salomo Reinhart Gregory Manalu

2. 13521069 / Louis Caesa Kesuma

3. 13521071 / Margaretha Olivia Haryono

4. 13521085 / Addin Munawwar Yusuf

5. 13521119 / Muhammad Rizky Sya'ban

Asisten Pembimbing : 13519064 / Aditya Bimawan

1. Konten Diskusi







1. Menunjukkan diagram
2. Program utama dijadikan 1 jar, lalu membuat 3 file jar untuk plugin (file jar bukan dari asisten)
3. Laporan penjualan isinya bebas, yang penting ditampilkan semua penjualannya (contoh: semua item di-list, trus ditulis berapa item sudah terjual). Kalo fixed bill cuma 1 aja. Mengolah semua fixed bill
4. IDE samain biar developnya lebih enak, kalo bisa UI sama pluginnya kerja bareng

2. Tindak Lanjut

1. Menentukan kelas-kelas yang akan diimplementasikan beserta atributnya
2. Menentukan *page-page* yang akan dibuat
3. Membagi tugas mengerjakan page
4. Setelah setiap page jadi, menyatukan page-page tersebut dalam App

3. Foto Dokumentasi

← → ↻ 🔒 meet.google.com/nyd-nkty-jdb 📺 🔗 ☆ 🗑️ S ⋮

 064 Aditya Bimawan	 13521069 Louis Caesa Kesuma	 13521119 Muhammad Rizky Sya'ban
 13521085 Addin Munawwar Yusuf	 13521071 Margaretha Olivia Haryono	 You

8:31 PM | Asistensi 1 Tubes OOP 2 🔇 🔕 📄 😊 ➕ 🖱️ ⋮ 📞

🔍 👤 💬 ⏏️ 🔒 6