APLIKASI TO DO LIST (DAFTAR TUGAS)

Dosen Pengampu:
Didik Kurniawan, S.Si., M.T.
Wahyu Aji Pulungan, S.T., M.T.

Kelompok 8:

FAJAR MAULANA	2307051006
PUTRI ALENA SARI	2307051024
CAHYA NAYLA DWI P.R.	2307051008
NUGRAHA ADITYA W.	2307051012

MANAJEMEN INFORMATIKA
UNIVERSITAS LAMPUNG

14 DESEMBER 2024

DAFTAR ISI

1. PENDAHULUAN

1.1 Latar Belakang

Dalam kehidupan sehari-hari menjadi mahasiswa, kami sering dihadapkan dalam aneka macam jadwal aktivitas dan tugas yang wajib diselesaikan. Terkadang, hal ini bisa mengakibatkan seorang merasa terbebani lantaran perlu mengingat banyak tugas dan informasi. Hal ini tentunya buruk bagi kesehatan mental, terutama bagi mereka yang wajib menyimpan aneka macam data pada waktu yang singkat. Belum lagi, masih ada mahasiswa atau individu yang mudah lupa atau kurang mempunyai konsentrasi pada jangka waktu yg lama. Mengingat aneka macam kasus ini, sangat krusial untuk mempunyai catatan.

Namun, situasi ini selalu memungkinkan kita untuk membawa buku catatan. Oleh karena itu, kami menentukan projek Aplikasi To Do List yaitu bentuk pelaksanaan catatan berbasis objek yg bisa membantu pengguna mencatat hal-hal krusial menjadi lebih efisien. Aplikasi To-Do List yang dibuat dengan bahasa Java adalah proyek yang memungkinkan pengguna untuk mengatur dan melacak tugas atau pekerjaan yang perlu diselesaikan. Aplikasi semacam ini umumnya mencakup fitur-fitur dasar seperti menambah tugas baru, mengedit tugas yang ada, menandai tugas sebagai selesai, dan menghapus tugas.

1.2 Tujuan

Berikut tujuan dalam pembuatan proyek aplikasi to do list:

1. Menciptakan alat yang membantu mahasiswa mengelola tugas dan waktu dengan lebih baik, sehingga mereka dapat lebih fokus dan produktif.
2. Mahasiswa dapat merancang aplikasi yang sesuai dengan kebutuhan mereka sendiri, menyesuaikan fitur dan fungsi untuk mendukung gaya belajar dan kerja mereka.
3. Mempraktikkan keterampilan pemrograman dengan proyek nyata membantu mahasiswa memahami konsep-konsep dasar seperti struktur data, algoritma, dan manajemen memori.

1.3 Manfaat

Adapun manfaat dari pembuatan proyek aplikasi to do list yaitu:

1. Aplikasi To-Do List membantu pengguna memprioritaskan tugas dan mengatur waktu, mengurangi stres dan meningkatkan efisiensi.

2. Mahasiswa dapat memperdalam pengetahuan mereka tentang Java dan pengembangan aplikasi, yang sangat berharga untuk masa depan karir mereka.
3. Menggunakan To-Do List secara teratur dapat membantu mahasiswa mengembangkan kebiasaan baik dalam pengelolaan waktu dan tugas.
4. Aplikasi ini dapat menjadi bagian dari portofolio mereka, menunjukkan keterampilan teknis dan kemampuan mereka dalam menyelesaikan proyek dari awal hingga akhir.

2. TINJAUAN PUSTAKA

2.1 Referensi Pemrograman Berorientasi Objek (PBO)

1. Belajar Pemrograman Berorientasi Objek dengan Java - Codepolitan

   Artikel ini membahas konsep dasar PBO seperti kelas, objek, enkapsulasi, pewarisan, dan polimorfisme dengan contoh nyata dalam bahasa Java.

2. Konsep Pemrograman Berorientasi Objek (PBO/OOP) - Serupa.id

   Artikel ini menjelaskan mengapa PBO penting dan bagaimana cara mengimplementasikannya dalam Java, termasuk analisis dan pemodelan berorientasi objek.

2.2 Referensi Penggunaan Java dalam Aplikasi To-Do List

Membuat Aplikasi Todo List (CRUD) dengan Java dan File - Petani Kode Tutorial ini membahas cara membuat aplikasi To-Do List berbasis teks dengan memanfaatkan file sebagai media penyimpanan.

3. IMPLEMENTASI

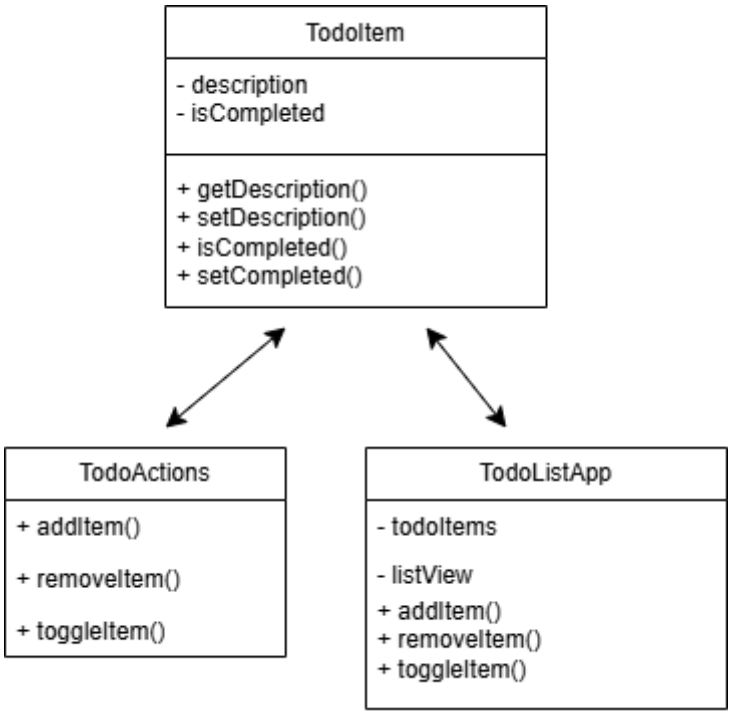
3.1 Deskripsi Project

Aplikasi TodoListApp adalah aplikasi GUI berbasis JavaFX yang memungkinkan pengguna untuk menambah, menghapus, dan mengubah status item dalam daftar tugas (To-Do List). Aplikasi ini menerapkan konsep-konsep pemrograman berorientasi objek seperti enkapsulasi, pewarisan, dan antarmuka.

3.2 Arsitektur Program

3.2.1 Struktur Program

Aplikasi ini menggunakan pendekatan berbasis objek dengan struktur berikut:



- Kelas TodoItem: Model data untuk setiap tugas, berisi atribut description dan isCompleted dengan getter dan setter.
- Antarmuka TodoActions: Mendefinisikan tindakan utama untuk daftar tugas seperti menambah, menghapus, dan mengganti status.
- Kelas Utama TodoListApp: Mengatur logika aplikasi dan antarmuka pengguna, mewarisi dari Application (JavaFX) dan mengimplementasikan antarmuka TodoActions.

3.2.2 Diagram Alur

a. Menambahkan Tugas:
   Pengguna memasukkan teks dalam TextField → Klik tombol Add → Data ditambahkan ke model todoItems dan diperbarui di tampilan listView.

b. Menghapus Tugas:
   Pengguna memilih tugas → Klik tombol Remove → Tugas dihapus dari model dan tampilan.

c. Mengubah Status:
   Pengguna memilih tugas → Klik tombol Toggle Complete → Status tugas diperbarui di model dan ditampilkan di tampilan.

3.3 Kode Program

Berikut adalah kode programnya.

Kode pada file TodoItem.java

```
// Mengimpor berbagai kelas yang diperlukan untuk membuat aplikasi GUI
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.stage.Stage;
import java.util.ArrayList;


// Encapsulation: Semua atribut private dengan getter dan setter
class TodoItem {
    private String description;
    private boolean isCompleted;


    public TodoItem(String description) {
```

```
        this.description = description;
        this.isCompleted = false;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public boolean isCompleted() {
        return isCompleted;
    }

    public void setCompleted(boolean completed) {
        isCompleted = completed;
    }
}

// Abstract Class / Interface: Menggunakan interface untuk tindakan TodoItem
interface TodoActions {
    void addItem(String description);
    void removeItem(int index);
    void toggleItem(int index);
}
```

Kode pada file TodoListApp.java

```
import java.util.ArrayList;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.scene.control.TextField;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

// Inheritance: Kelas utama TodoListApp mewarisi Application
public class TodoListApp extends Application implements TodoActions {

    private static int itemCount = 0; // Static Modifier: Counter untuk menghitung total item
    private final ArrayList<TodoItem> todoItems = new ArrayList<>();
    private final ListView<String> listView = new ListView<>();

    @Override
    public void start(Stage primaryStage) {
        // Layout utama
        BorderPane root = new BorderPane();
```

```
        root.setPadding(new Insets(10));

        // Header
        Label header = new Label("Todo List");
        header.setStyle("-fx-font-size: 24px; -fx-font-weight: bold; -fx-text-fill: #0066cc;");
        root.setTop(header);
        BorderPane.setMargin(header, new Insets(10, 0, 20, 0));

        // List View
        listView.setStyle("-fx-font-size: 14px;");
        root.setCenter(listView);

        // Input Field dan Tombol
        TextField inputField = new TextField();
        inputField.setPromptText("Enter a new task");
        inputField.setStyle("-fx-font-size: 14px;");

        Button addButton = new Button("Add");
        Button removeButton = new Button("Remove");
        Button toggleButton = new Button("Toggle Complete");

        addButton.setStyle("-fx-background-color: #009933; -fx-text-fill: white; -fx-font-size: 14px;");
        removeButton.setStyle("-fx-background-color: #cc0000; -fx-text-fill: white; -fx-font-size: 14px;");
        toggleButton.setStyle("-fx-background-color: #0066cc; -fx-text-fill: white; -fx-font-size: 14px;");
```

```
        HBox controls = new HBox(10, addButton, removeButton,
toggleButton);
        controls.setPadding(new Insets(10, 0, 0, 0));


        VBox inputSection = new VBox(10, inputField, controls);
        root.setBottom(inputSection);


        // Event Handlers
        addButton.setOnAction(e -> {
            String description = inputField.getText().trim();
            if (!description.isEmpty()) {
                addItem(description);
                inputField.clear();
            } else {
                showAlert("Error", "Todo item cannot be empty.",
Alert.AlertType.ERROR);
            }
        });


        removeButton.setOnAction(e -> {
            int selectedIndex =
listView.getSelectionModel().getSelectedIndex();
            if (selectedIndex != -1) {
                removeItem(selectedIndex);
            } else {
                showAlert("Error", "Please select an item to remove.",
Alert.AlertType.ERROR);
            }
        });
```

```
        toggleButton.setOnAction(e -> {
            int selectedIndex = 
listView.getSelectionModel().getSelectedIndex();
            if (selectedIndex != -1) {
                toggleItem(selectedIndex);
            } else {
                showAlert("Error", "Please select an item to toggle.", 
Alert.AlertType.ERROR);
            }
        });

        // Scene dan Stage
        Scene scene = new Scene(root, 400, 500);
        primaryStage.setTitle("Todo List App");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    // Static Method: Mengembalikan jumlah item
    public static int getItemCount() {
        return itemCount;
    }

    // Implementasi metode dari interface TodoActions
    @Override
    public void addItem(String description) {
        TodoItem newItem = new TodoItem(description);
        todoItems.add(newItem);
```

```
        listView.getItems().add(description);
        itemCount++;
    }

    @Override
    public void removeItem(int index) {
        todoItems.remove(index);
        listView.getItems().remove(index);
        itemCount--;
    }

    @Override
    public void toggleItem(int index) {
        TodoItem item = todoItems.get(index);
        item.setCompleted(!item.isCompleted());
        String updatedDescription = item.isCompleted() ? "[Completed] " + item.getDescription() : item.getDescription();
        listView.getItems().set(index, updatedDescription);
    }

    private void showAlert(String title, String message, Alert.AlertType alertType) {
        Alert alert = new Alert(alertType);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(message);
        alert.showAndWait();
    }
```

```
    public static void main(String[] args) {
        launch(args);
    }
}
```

4. PEMBAHASAN

Penerapan Konsep PBO

4.1 Encapsulation

Enkapsulasi adalah konsep yang menggabungkan data dan metode yang bekerja pada data tersebut menjadi satu unit, serta menyembunyikan detail internal dari objek. Ini memastikan bahwa data internal objek tidak dapat diakses secara langsung dari luar, melainkan melalui metode yang telah ditentukan. Enkapsulasi membantu menjaga integritas data dan mencegah modifikasi yang tidak diinginkan.

Contoh implementasinya dalam projek ini:

```
// Encapsulation: Semua atribut private dengan getter dan setter
class TodoItem {
    private String description;
    private boolean isCompleted;

    public TodoItem(String description) {
        this.description = description;
        this.isCompleted = false;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public boolean isCompleted() {
        return isCompleted;
    }
```

```
    public void setCompleted(boolean completed) {
        isCompleted = completed;
    }
}
```

Penjelasan Kode

1. Kelas TodoItem:
   - Kelas ini mewakili item dalam sebuah daftar tugas (To-Do List).

2. Atribut Private:
   - private String description: Menyimpan deskripsi dari tugas.
   - private boolean isCompleted: Menyimpan status apakah tugas sudah selesai atau belum.
   - Atribut ini dibuat private untuk melindungi data dan mengontrol akses melalui metode getter dan setter.

3. Konstruktor:
   - public TodoItem(String description): Konstruktor untuk membuat objek TodoItem dengan deskripsi yang diberikan. Status isCompleted diatur ke false secara default.

4. Getter dan Setter:
   - public String getDescription(): Mengembalikan deskripsi dari tugas.
   - public void setDescription(String description): Mengatur deskripsi dari tugas.
   - public boolean isCompleted(): Mengembalikan status apakah tugas sudah selesai.
   - public void setCompleted(boolean completed): Mengatur status apakah tugas sudah selesai.

4.2 Inheritance

Pewarisan (Inheritance) Pewarisan adalah mekanisme di mana sebuah kelas dapat mewarisi atribut dan metode dari kelas lain. Kelas yang mewarisi disebut subclass atau kelas turunan, sedangkan kelas yang diwarisi disebut superclass atau kelas induk. Pewarisan memungkinkan penggunaan kembali kode dan mempermudah pemeliharaan program.

Contoh implementasinya dalam program ini:

```
// Inheritance: Kelas utama TodoListApp mewarisi Application
```

```
public class TodoListApp extends Application implements
TodoActions {
   // ...
}
```

Penjelasan kode:

a. Kelas TodoListApp:

- Kelas ini adalah kelas utama yang mewarisi kelas Application dari JavaFX.
- public class TodoListApp extends Application menunjukkan bahwa TodoListApp adalah subclass dari Application.

b. Implements TodoActions:

- TodoListApp juga mengimplementasikan antarmuka TodoActions, yang berarti kelas ini harus menyediakan implementasi untuk semua metode yang didefinisikan dalam antarmuka tersebut.
- implements TodoActions menunjukkan bahwa TodoListApp berkomitmen untuk menyediakan logika untuk metode addItem, removeItem, dan toggleItem.

4.3 Polymorphism

Polimorfisme memungkinkan metode yang sama untuk digunakan pada objek yang berbeda. Ini berarti bahwa satu metode dapat memiliki banyak bentuk atau implementasi yang berbeda, tergantung pada objek yang memanggilnya. Polimorfisme memungkinkan fleksibilitas dan perluasan kode tanpa merusak kode yang ada.

Contoh implementasinya dalam program ini:

```
interface TodoActions {
   void addItem(String description);
   void removeItem(int index);
   void toggleItem(int index);
}

public class TodoListApp extends Application implements
TodoActions {
   @Override
   public void addItem(String description) {
      // Implementasi tambah item
   }

   @Override
```

```
    public void removeItem(int index) {
        // Implementasi hapus item
    }

    @Override
    public void toggleItem(int index) {
        // Implementasi toggle status item
    }
}
```

Penjelasan kode:

a. Interface TodoActions
   - Antarmuka TodoActions mendefinisikan tiga metode: addItem, removeItem, dan toggleItem.
   - Setiap kelas yang mengimplementasikan antarmuka ini harus menyediakan implementasi konkret untuk ketiga metode tersebut.
   - Antarmuka digunakan untuk mendefinisikan kontrak yang harus diikuti oleh kelas-kelas pengimplementasinya.

b. Class TodoListApp
   - Kelas TodoListApp memperluas (extends) kelas Application dari JavaFX, menjadikannya sebuah aplikasi JavaFX.
   - Kelas ini juga mengimplementasikan antarmuka TodoActions, yang berarti harus menyediakan implementasi untuk semua metode yang didefinisikan dalam antarmuka TodoActions.
   - Tiga metode yang diimplementasikan (addItem, removeItem, toggleItem) adalah implementasi konkret dari metode yang didefinisikan dalam antarmuka TodoActions.

4.4 Abstract Class atau Interface

Abstraksi adalah proses menyederhanakan kompleksitas dengan hanya menampilkan fitur penting dan menyembunyikan detail yang tidak relevan. Dalam PBO, abstraksi dicapai dengan menggunakan kelas dan objek untuk memodelkan entitas dunia nyata, sementara detail implementasi disembunyikan dari pengguna. Abstraksi membantu dalam memahami dan mengelola kompleksitas sistem perangkat lunak.

Contoh implementasinya dalam program ini:

```
// Abstract Class / Interface: Menggunakan interface untuk tindakan TodoItem
interface TodoActions {
```

```
    void addItem(String description);
    void removeItem(int index);
    void toggleItem(int index);
}
```

Penjelasan kode:
Interface ini memastikan bahwa kelas yang mengimplementasikannya (seperti TodoListApp) harus menyediakan implementasi spesifik untuk ketiga metode berikut.

- addItem(String description): Menambahkan item baru dengan deskripsi tertentu.
- removeItem(int index): Menghapus item berdasarkan indeksnya.
- toggleItem(int index): Mengubah status item (selesai atau belum selesai) berdasarkan indeks.

```
// Implementasi metode dari interface TodoActions
    @Override
    public void addItem(String description) {
        TodoItem newItem = new TodoItem(description);
        todoItems.add(newItem);
        listView.getItems().add(description);
        itemCount++;
    }

    @Override
    public void removeItem(int index) {
        todoItems.remove(index);
        listView.getItems().remove(index);
        itemCount--;
    }

    @Override
    public void toggleItem(int index) {
        TodoItem item = todoItems.get(index);
        item.setCompleted(!item.isCompleted());
        String updatedDescription = item.isCompleted() ? "[Completed] "
+ item.getDescription() : item.getDescription();
        listView.getItems().set(index, updatedDescription);
    }
```

```
    private void showAlert(String title, String message, Alert.AlertType
alertType) {
        Alert alert = new Alert(alertType);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(message);
        alert.showAndWait();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Penjelasan kode:

Implementasi Metode dari TodoActions:

a. addItem(String description): Membuat objek TodoItem baru dengan deskripsi. Menambahkan item ke daftar todoItems dan memperbarui tampilan daftar di GUI (listView). Menambah jumlah total item (itemCount).
b. removeItem(int index): Menghapus item dari daftar todoItems berdasarkan indeks. Menghapus elemen dari tampilan GUI (listView). Mengurangi jumlah total item (itemCount).
c. toggleItem(int index):Mengambil item berdasarkan indeks dan mengubah statusnya (completed). Memperbarui deskripsi tugas di tampilan GUI (listView) dengan menambahkan tanda [Completed] jika selesai.

Metode Tambahan:

d. showAlert: Menampilkan dialog pesan kepada pengguna (contoh: kesalahan atau informasi penting). Digunakan untuk memberikan umpan balik, misalnya ketika tugas tidak dapat ditambahkan atau dihapus.

Metode main:

e. Memulai aplikasi dengan memanggil launch(args), yang memanggil start(Stage) untuk menampilkan GUI.

4.5 Static Modifier

Static Modifier adalah salah satu modifier yang memungkinkan mengakses *Property* atau *Method* dari *Class* lain tanpa harus menginisialisasi/*instance Class* tersebut terlebih dahulu. Berikut contohnya:

```
private static int itemCount = 0; // Static Modifier: Counter untuk
menghitung total item
   private final ArrayList<TodoItem> todoItems = new ArrayList<>();
   private final ListView<String> listView = new ListView<>();
```

```
  // Static Method: Mengembalikan jumlah item
   public static int getItemCount() {
      return itemCount;
   }
```

Penjelasan kode:

a. 1rivate static int itemCount = 0;
   Variabel itemCount digunakan untuk menghitung jumlah total item dan bersifat static, sehingga nilainya bersama di seluruh objek kelas ini. Dimulai dari 0, akan bertambah atau berkurang seiring perubahan daftar item.

b. private final ArrayList<TodoItem> todoItems = new ArrayList<>();
   Variabel todoItems adalah daftar tugas berbentuk objek TodoItem. Hal ini bersifat final, sehingga referensi list ini tidak dapat diubah, tetapi isi list (item) dapat dimodifikasi.

c. private final ListView<String> listView = new ListView<>();
   listView adalah komponen grafis untuk menampilkan daftar item kepada pengguna. Digunakan dalam aplikasi GUI seperti JavaFX untuk merepresentasikan isi todoItems.

d. public static int getItemCount()
   Metode ini mengembalikan nilai dari itemCount dan bersifat static, sehingga dapat dipanggil tanpa membuat objek kelas.

4.6 Error/Exception Handling

Exception Handling adalah proses yang memungkinkan kita sebagai pengembang untuk menulis kode yang dapat menangani kesalahan yang mungkin muncul saat program dijalankan. Kesalahan ini dapat berupa masalah teknis, seperti kesalahan dalam pengaksesan file yang tidak ada atau masukan pengguna yang tidak valid.

```
// Event Handlers
      addButton.setOnAction(e -> {
         String description = inputField.getText().trim();
         if (!description.isEmpty()) {
```

```
            addItem(description);
            inputField.clear();
        } else {
            showAlert("Error", "Todo item cannot be empty.",
Alert.AlertType.ERROR);
        }
    });

    removeButton.setOnAction(e -> {
        int selectedIndex =
listView.getSelectionModel().getSelectedIndex();
        if (selectedIndex != -1) {
            removeItem(selectedIndex);
        } else {
            showAlert("Error", "Please select an item to remove.",
Alert.AlertType.ERROR);
        }
    });

    toggleButton.setOnAction(e -> {
        int selectedIndex =
listView.getSelectionModel().getSelectedIndex();
        if (selectedIndex != -1) {
            toggleItem(selectedIndex);
        } else {
            showAlert("Error", "Please select an item to toggle.",
Alert.AlertType.ERROR);
        }
    });
```

Penjelasan kode:

a. addButton.setOnAction(e -> {...})

- inputField.getText().trim(): Mengambil teks dari inputField (tempat pengguna mengetik) dan menghapus spasi di awal/akhir.
- if (!description.isEmpty()): Memeriksa apakah teks input tidak kosong.
- addItem(description): Menambahkan teks sebagai item baru ke daftar Todo.
- inputField.clear(): Menghapus isi inputField setelah item ditambahkan.
- showAlert(...): Menampilkan pesan error jika teks input kosong.

b. removeButton.setOnAction(e -> {...})

- listView.getSelectionModel().getSelectedIndex(): Mengambil indeks item yang dipilih pengguna dari daftar (listView).
- if (selectedIndex != -1): Memeriksa apakah ada item yang dipilih (indeks tidak -1).
- removeItem(selectedIndex): Menghapus item yang dipilih berdasarkan indeksnya.
- showAlert(...): Menampilkan pesan error jika tidak ada item yang dipilih.

c. toggleButton.setOnAction(e -> {...})
   - listView.getSelectionModel().getSelectedIndex():Sama seperti di atas, mengambil indeks item yang dipilih.
   - if (selectedIndex != -1): Memeriksa apakah ada item yang dipilih.
   - toggleItem(selectedIndex): Mengubah status item yang dipilih (misalnya, dari belum selesai menjadi selesai atau sebaliknya).
   - showAlert(...): Menampilkan pesan error jika tidak ada item yang dipilih.

d. setOnAction(e -> {...}): Menentukan aksi yang akan dijalankan ketika tombol diklik.
e. showAlert(String title, String message, Alert.AlertType type): Digunakan untuk menampilkan kotak dialog dengan pesan error.
f. Validasi Input: Semua tombol memeriksa apakah input valid (tidak kosong atau item terpilih) sebelum melakukan aksi untuk mencegah kesalahan.

5. UJI COBA DAN HASIL

Skenario 1 : Menambahkan Tugas

- Pengguna mengisi daftar tugas.
- Langkah pengujian : Pengguna memasukkan teks misalnya “Tugas PBO”, kemudian pengguna mengklik tombol add.

Skenario 2 : Menghapus Tugas

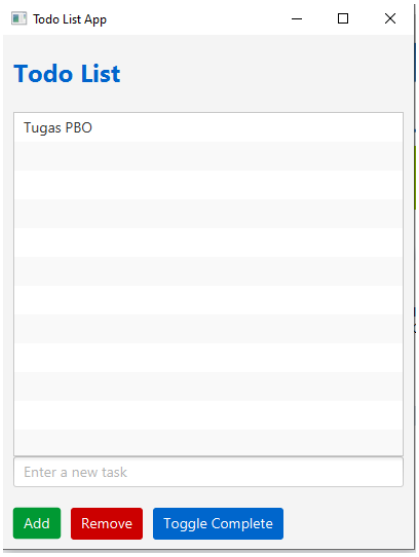
- Pengguna menghapus daftar tugas
- Langkah pengujian : Pengguna mengklik salah satu tugas pada kolom. Pengguna mengklik tombol delete

Skenario 3 : Menambahkan Tugas Completed

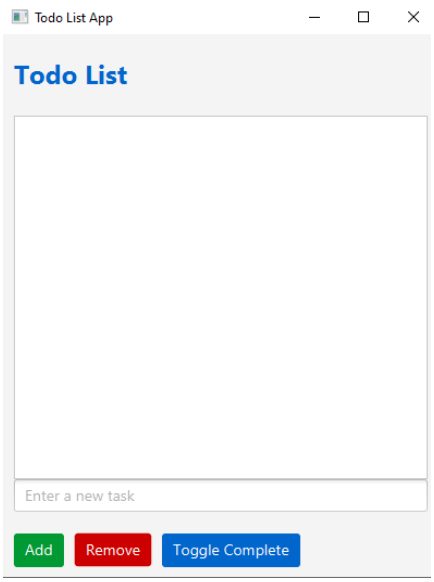
- Pengguna menghapus daftar tugas
- Langkah pengujian : Pengguna mengklik salah satu tugas pada kolom. Pengguna mengklik tombol Completed

5.1 Hasil yang Diharapkan

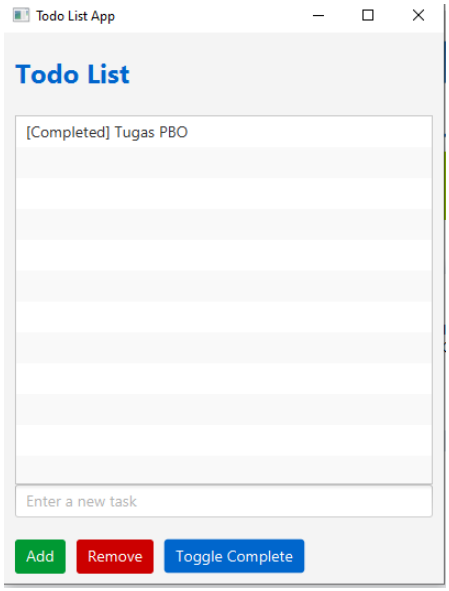
Skenario 1 : Item "Tugas PBO" ditambahkan ke daftar dengan status belum selesai.



Skenario 2: Item yang diklik delete akan terhapus dari tampilan aplikasi.



Skenario 3: Teks yang di klik completed akan menampilkan pesan completed disamping teks.



5.2 Masalah yang ditemui dan Solusinya

Pada saat menjalankan program berjalan dengan baik dan tidak ditemui masalah.

6. KESIMPULAN

Proyek To-Do List berbasis Java menggunakan paradigma Pemrograman Berorientasi Objek (PBO), dengan implementasi fitur dasar seperti menambah, menghapus, dan memperbarui status tugas. Konsep PBO seperti enkapsulasi, pewarisan, polimorfisme, antarmuka, dan static modifier telah diterapkan, yang membantu menjaga modularitas dan fleksibilitas kode. Aplikasi ini menggunakan GUI berbasis JavaFX, memberikan antarmuka pengguna yang intuitif dan fungsional.

7. SARAN

1. Pengembangan Fitur:

- Tambahkan fungsi penyimpanan data ke dalam file atau basis data agar daftar tugas tidak hilang saat aplikasi ditutup.
- Implementasikan pengingat (reminder) untuk tugas dengan tanggal jatuh tempo.
- Kembangkan fitur kategorisasi tugas agar pengguna dapat mengelompokkan tugas berdasarkan prioritas atau jenis aktivitas.

2. Optimasi Aplikasi:

- Gunakan exception handling yang lebih spesifik agar pengelolaan error lebih terarah, misalnya menggunakan try-catch untuk kesalahan spesifik seperti indeks keluar dari batas (index out of bounds).
- Perluas pengujian untuk memastikan aplikasi berjalan lancar di berbagai perangkat dan skenario.

3. Tampilan:

- Tingkatkan estetika GUI dengan menggunakan gaya yang lebih modern, misalnya dengan theme CSS untuk JavaFX.
- Berikan panduan kepada pengguna baru melalui pesan pop-up atau panduan dalam aplikasi.

8. DAFTAR PUSTAKA

Prasetya. (2024, Desember 18). *Belajar Pemrograman Berorientasi Objek (PBO): Pengertian dan Contohnya!* Retrieved from Codepolitan: https://www.codepolitan.com/blog/belajar-pemrograman-berorientasi-objek-pbo-pengertian-dan-contohnya/

Skodev. (2023, December 15). *APA ITU EXCEPTIONHANDLING PENGERTIAN DAN INFORMASI*. Retrieved from sko.dev: https://sko.dev/wiki/exception-handling-kesalahan-penanganan

Tau, S. (2020, May 3). *Apa itu Static Modifier, Final Modifier, dan Singleton Pattern?* Retrieved from Medium: https://gayuhnh.medium.com/apa-itu-static-modifier-final-modifier-dan-singleton-pattern-751dace814e6