Expense Tracker Application

Overview

The **Expense Tracker Application** is a JavaFX-based personal finance manager that enables users to track daily expenses, categorize them, and visualize their spending using charts. It supports different user roles (Normal and Premium), authentication, expense tracking, and theme toggling between dark and light modes. This application leverages Java concepts such as **Object-Oriented Programming**, **multi-threading**, **exception handling**, and **GUI programming**.

Features

1. **User Authentication**
   - Users can sign up with a **username** and **password**.
   - **Sign In** allows authenticated users to access their personalized expense tracking dashboard.
   - **User Types:**
     1. **Normal User**: Limited features and categories.
     2. **Premium User**: Access to advanced features, including the ability to add custom categories.
2. **Expense Tracking**
   - Users can add expenses with an **amount** and a **category**.
   - Categories include **Food**, **Travel**, **Entertainment**, and **Others** for normal users, while Premium users can add custom categories.
   - **Expenses are displayed** in a list view.
   - **Clear All**: Users can clear all expenses at any time.
3. **Data Visualization**
   - A **Bar Chart** is provided to visualize expenses by category.
   - The chart updates dynamically as new expenses are added.
4. **Dark Mode and Light Mode**
   - Users can toggle between **dark** and **light** themes, allowing a customizable experience.
5. **Sign Out Functionality**

   - Users can **sign out** at any time and return to the sign-in screen.

Java Concepts Covered

Java Data Types, Type Conversions, and Operators

1. Data Types
   Usage of primitive types like int, double, boolean, and String.

code

private final String username;

private final String password;

2. Type Conversions
   Example of String to double conversion in the addExpense method.

code

double amount = Double.parseDouble(amountField.getText());

3. Operators
   Arithmetic (+, *) operators are used for expense calculations, while logical operators handle user role checks.

Control Statements

- If-Else Statements
  Used for user authentication checks.
- Exception Handling (Try-Catch)
  For handling errors, such as invalid input data.
- Loops
  for and while loops are used to iterate over lists of expenses and other elements.

Arrays and Lists

1. ArrayLists
   Expenses are stored in ArrayList and ObservableList.

code

private final List<Expense> expenses;

return new ArrayList<>(items);

2. Thread Safety
   ArrayList is synchronized in Database and UserDatabase classes to ensure thread safety.

Classes & Methods

1. Entity Classes
   Multiple classes represent various entities, such as User, Expense, and UserDatabase.
2. Method Examples
   Methods handle user actions, such as adding expenses and toggling themes.

code

```
class Expense {
  private final double amount;
  private final String category;

  public Expense(double amount, String category) {
    this.amount = amount;
    this.category = category;
  }

  public double getAmount() { return amount; }
  public String getCategory() { return category; }
}
```

3. Abstract Class
   UserRole is an abstract class with subclasses NormalView and PremiumView.

code

```
abstract class UserRole {
  private final boolean isPremium;

  public UserRole(boolean isPremium) {
    this.isPremium = isPremium;
  }

  public boolean isPremium() { return isPremium; }
  public abstract void showView(Stage primaryStage);
}
```

Class Inheritance

- Inheritance Example
  UserRole is the base abstract class, with NormalView and PremiumView extending it to demonstrate inheritance.

Access Control, Static Keywords & Inner Classes

1. Access Control
   Various fields and methods are defined with access modifiers (private, public, protected).

<u>code</u>

```
private final String username;
private final String password;
private final List<Expense> expenses;
```

2. Static Keyword
   The UserDatabase is a static field to centralize user data, and a static toggle is used for the dark mode feature.

<u>code</u>

```
private static boolean isDarkMode = false;
```

3. Inner Classes
   UI components such as ThemeManager are designed as inner classes for encapsulation.

Interfaces & Abstract Classes

1. Interface
   UserAuthentication interface defines the contract for user authentication.
2. Abstract Class
   UserRole is an abstract class used as a base for different user roles.

code

```
class User extends UserRole implements UserAuthentication {
  // Implementation here
}
```

String Handling

- String Manipulations
  String objects handle user inputs, with equals() and contentEquals() used for comparisons.

code

```
this.username.equals(username) && this.password.equals(password);
```

Exception Handling

1. Try-Catch Blocks
   Used in methods like addExpense to handle exceptions (e.g., NumberFormatException for invalid data).
2. Custom Exceptions
   ThemeApplicationException handles theme toggle failures.

code

```
class ThemeApplicationException extends Exception {
  public ThemeApplicationException(String message) {
    super(message);
  }

  public ThemeApplicationException(String message, Throwable cause) {
    super(message, cause);
  }
}
```

Multithreaded Programming

- Multi-threading Example
  A separate thread (updateExpensesThread) updates the expense list every second, using Platform.runLater() to update the GUI from the background thread safely.

code

```
Thread updateExpensesThread = new Thread(() -> {
  while (true) {
    Platform.runLater(() -> loadExpenses(user));
    try {
      Thread.sleep(1000);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
});
```

Generics

- Generic Class
  Database<T> is a generic class, allowing for type-safe storage and retrieval of different object types.

JavaFX - GUI Programming

1. JavaFX Components
   The application uses JavaFX components such as Scene, Stage, TextField, Button, VBox, HBox, Alert, and BarChart.
2. Event Handling
   Event listeners are implemented for buttons like "Sign In", "Add Expense", and "Show Bar Chart".
3. Themes
   ThemeManager applies dark and light themes dynamically based on user preferences.

Usage

Steps to Use the Application

- **Launch the Application**: Upon running the application, the user will see a **sign-up screen**.
- **Sign Up**: If you are a new user, fill in your **username** and **password**, select either "Normal User" or "Premium User," and click the **Sign Up** button.
- **Sign In**: For returning users, click **Sign In** and enter your credentials.
- **Add Expenses**: Enter the **amount** and select or type the **category** to add an expense.
- **Clear Expenses**: Clear all expenses by clicking **Clear All**.
- **View Bar Chart**: Click **Show Bar Chart** to view a visual representation of your expenses.
- **Toggle Theme**: Use the **Toggle Theme** button to switch between dark and light modes.
- **Sign Out**: Click the **Sign Out** button to sign out.

How to Download and Run the Application

1. **Download the Source Code**
   Clone the repository or download the .zip file containing the project files.
2. **Dependencies**
   Ensure that JavaFX is set up in your project, as this application uses JavaFX for GUI development.
3. **Compile and Run**
   - Open the project in an IDE (Eclipse, IntelliJ, etc.) or compile it from the terminal.
   - **Terminal Commands:**

code

```
javac -cp path_to_javafx_libs *.java

java -cp .:path_to_javafx_libs javafxpackaged.Main
```

4. **Optional Theme Files**
   Make sure darkmode.css and lightmode.css are in your project folder to apply themes.

Conclusion

The Expense Tracker application demonstrates a well-structured Java project that integrates key Java programming concepts. It showcases object-oriented principles such as **inheritance**, **interfaces**, **exception handling**, **multi-threading**, and **GUI development using JavaFX.** This application serves as a practical tool for managing personal finances and provides a valuable learning experience in **Java development**.