Dokumentation (md)

Belegarbeit - Simple budget app

Name: Thi Phuöng Hien Tran
Matrikelnummer: 585489

1. Anwendungsfall

Es handelt sich um eine einfache Java-Anwendung, mit der Transaktionen mit den folgenden Eingabefeldern erfasst werden können: Datum, Name, Kategorie und Betrag. Zu den unterstützten Funktionen gehören Hinzufügen, Entfernen, Bearbeiten, Löschen, Sortieren, Suchen und Speichern, die im Folgenden erläutert werden:

- Alle erfassten Transaktionen ausdrucken
- Eine neue Transaktion hinzufügen
- Eine Transaktion löschen
- Alle Transaktionen löschen
- Eine Transaktion bearbeiten
- Alle Transaktionen nach Betrag, Datum, Namen oder Kategorie sortieren
- Alle Transaktionen nach Betrag, Datum, Namen oder Kategorie suchen
- Alle Kategorien und die für diese Kategorien ausgegebenen Beträge auflisten
- Alle Transaktionen in der csv-Datei "budget.csv" speichern, damit sie beim nächsten Start wieder abgerufen werden können

Um die Funktionalitäten der App zu veranschaulichen, wird die csv-Datei "budget.csv", die alle aufgezeichneten Transaktionen enthält, mit einigen Mustertransaktionen vorgeladen. Default-Kategorien werden auch aus der Datei "categories.csv" importiert.

2. Installationsanleitung

Option 1: Mit einer IDE ausführen

1. Die Datei `585489.zip` entpacken
2. Den Ordner `BudgetApp` mit einer IDE nach Wahl öffnen
3. Zu diesem Dateipfad navigieren `\BudgetApp\app\src\main\java\budgetapp\App.java` und `App.java` ausführen

Option 2: JAR-Datei mit dem Terminal ausführen

1. Die Datei `585489.zip` entpacken
2. Den Ordner `BudgetApp` öffnen
3. Terminal in diesem Ordner öffnen
4. Das Kommando `java -jar app.jar` ausführen

3. Benutzeranleitung

1. Programm starten

Beim Start der Anwendung werden Transaktionen aus der Datei `budget.csv` (im Stammverzeichnis) geladen. Änderungen an der Liste der Transaktionen könnten auch in dieser Datei gespeichert und beim nächsten Start

abgerufen werden. Hier ist der vorgelandene Inhalt dieser Datei:

```
2022-10-01,Rewe,Groceries,-12.45
2022-12-02,Kaufland,Groceries,-35.7
2022-09-03,Doener,Going out,-5.5
2022-12-12,zara: new jacket,Fashion,-79.99
2022-10-23,TK,Insurance,-117.8
2022-01-07,Julia rent october,Rent,-450.0
2022-06-19,prepaid S o2,Technology,-15.0
2022-03-25,netto: pastries,Groceries,-2.69
2020-05-11,savings,Savings,1000.0
2022-12-07,Flixbus,Travel,-9.99
2022-12-06,Flohmarkt lamp,Home,-10.0
2022-12-12,Weihnachtsmarkt,Going out,-26.5

```

Jede Zeile der csv-Datei ist wie folgt formatiert: Datum, Name, Kategorie, Betrag. Ausgaben werden als negative Werte und Einnahmen als positive Werte angegeben.

Standardkategorien werden aus der Datei `categories.csv` (im Stammverzeichnis) importiert:

```
Groceries
Rent
Utilities
Going out
Technology
Fashion
Travel
Medical

```

Die folgende Optionen werden gezeigt. Geben Sie die entsprechende Nummer ein, um den gewählten Befehl auszuführen.

```
-----------------------------------------------------------------------------------------
Console-Application: Budget App                                    Hien Tran - s0585489
-----------------------------------------------------------------------------------------
  0| Show all transactions
  1| Add a transaction
  2| Remove a transaction
  3| Clear all transactions
  4| Edit a transaction
  5| Sort transactions
  6| Search in transactions
  7| Show statistics
  8| Save transactions
  9| Exit program
-----------------------------------------------------------------------------------------
Select an option:
```

Eine ungültige Option führt zu einem Fehlermeldung und das Programm wird Sie auffordern, einen gültigen Wert einzugeben.

2. Alle Transaktionen anzeigen

Die erste Option [0] im Hauptmenü wählen. Alle erfassten Transaktionen und die Summe der Transaktionen werden ausgedruckt.

3. Transaktionen hinzufügen, löschen, und bearbeiten

- Die Option [1] der Hauptmenü wählen, um eine neue Transaktion hinzufügen. Sie werden dann aufgefordert, Datum, Name, Kategorie und Betrag der Transaktion einzugeben. Das Standarddatum ist das aktuelle

Systemdatum. Sie können auch eine neue Kategorie hinzufügen.
- Die Option [2] der Hauptmenü wählen, um eine Transaktion zu löschen. Sie können die zu löschende Transaktion durch Eingabe ihrer ID-Nummer auswählen.
- Die Option [3] der Hauptmenü wählen, um alle Transaktion zu löschen.
- Die Option [4] der Hauptmenü wählen, um eine Transaktion zu bearbeiten. Sie können die zu bearbeitende Transaktion durch Eingabe ihrer ID-Nummer auswählen. Sie werden dann aufgefordert, Datum, Name, Kategorie und Betrag der neuen Transaktion einzugeben.

4. Transaktionen sortieren

Die Option [5] der Hauptmenü wählen, um die Transaktionen zu sortieren. Die Transaktionen können nach Betrag [0], Datum [1], Namen [2], Kategorie [3] sortiert werden.

5. Transaktionen suchen

Die Option [6] der Hauptmenü wählen, um die Transaktionen zu sortieren. Die Transaktionen können nach Namen [0] (case insensitive), Kategorie [1], Datum [2] durchgesucht werden.

6. Transaktionen speichern und Programm beenden

Alle Transaktionen können in der csv-Datei "budget.csv" gespeichert werden, damit sie beim nächsten Start wieder abgerufen werden können.

Wenn Sie die Änderungen an der Transaktionsliste speichern möchten, können Sie die Option [7] des Hauptmenüs wählen. Der Inhalt der Datei "budget.csv" wird dann ebenfalls geändert.

Die Option [8] wählen, um das Programm zu beenden.

4. Datenstruktur und Datentypen

Für diese Anwendung wird die Datenstruktur einfach verketteten Liste gewählt. Sie erweitert die Interface List<Item> und hat die folgenden Methoden:

- `void add(Item data)`: Element am Ende der Liste hinzufügen
- `void add(Item data, int index)`: Hinzufügen eines Elements an einem bestimmten Index
- `void clear()`: Alle Elemente löschen
- `Item remove(int index)`: Entfernt Element an einem angegebenen Index und gibt das entfernte Element zurück
- `void set(int index, Item data)`: Element mit einem bestimmten Index auf ein anderes Element ersetzen
- `Item get(int index)`: Element an einem bestimmten Index abrufen
- `int size()`: Größe der verketteten Liste ermitteln
- `public void printAll()`: Alle Elemente der verketteten Liste drucken. Für diese spezifische Anwendung würde jedes Element mit dem folgenden Format gedruckt werden: `"%3d| %s\n", i, node.data"`, wobei `i` der Index des Elements und `node.data` die Daten des Elements in Form einer String sind.
- `boolean contains(Item data)`: Prüfen, ob die verkettete Liste das angegebene Element enthält
- `Node clone(Node head)`: Eine Kopie der verketteten Liste erstellen

Jede Transaktion wird als eine Instanz der Klasse `Transaction` gespeichert. Diese Klasse hat die privaten Variablen date (Type: LocalDate), name (Type: String), category (Type: String) und amount (Type: double). Der Konstruktor `Transaction(String date, String name, String category, double amount)` nimmt einen Datums-String (der dann mit der Klassenmethode `convertDate(String string)` in LocalDate umgewandelt wird), einen Name-String, einen Kategorie-String und einen Betrag-Double auf.

Die Transaktionen werden in einer `TransactionList` gespeichert, die von der Klasse `LinkedList` erbt. Zusätzlich zu den Methoden der Elternklasse ist die Klasse `TransactionList` auch für die Aufnahme von Benutzereingaben verantwortlich. Sie hat auch zusätzliche Methoden, um Transaktionen zu bearbeiten, Transaktionen zu importieren/exportieren (von und zu einer csv-Datei), die Transaktionen zu sortieren (mit dem Quicksort-Algorithmus), die Transaktionen zu suchen (mit einfacher linearer Suche). Eine Instanz dieser Klasse wird auch in einem speziellen Format gedruckt, um dem Anwendungsfall zu entsprechen.

5. Analyse der Laufzeitkomplexität der Sortier- und Suchalgorithmen

Sortieralgorithmus: Quicksort

```
private Node partitionLast(Node start, Node end, Comparator<Item> comparator) {
    if (start == end || start == null || end == null)  // 1 Operation
        return start;

    Node pivot_prev = start; // 1 Operation
    Node current = start;  // 1 Operation
    Item pivot = end.data;  // 1 Operation
    while (start != end) {                                  // |
        if (comparator.compare(start.data, pivot) < 0) {  // |
            pivot_prev = current;                           // |
            Item temp = current.data;                       // |
            current.data = start.data;                      // } n Operationen (Start -> End)
            start.data = temp;                              // |
            current = current.next;                         // |
        }                                                   // |
        start = start.next;                                 // |
    }

    Item temp = current.data; // 1 Operation
    current.data = pivot; // 1 Operation
    end.data = temp; // 1 Operation
    return pivot_prev;
}

public void quicksort(Node start, Node end, Comparator<Item> comparator) {
    if(start == null || start == end|| start == end.next ) // 1 Operation
        return;
    Node pivot_prev = partitionLast(start, end, comparator);  // n Operationen

    quicksort(start, pivot_prev, comparator);            // | linke Teilmenge sortieren --> n/2 Op

    if (pivot_prev != null && pivot_prev == start)       // |
        quicksort(pivot_prev.next, end, comparator);     // |
    else if (pivot_prev != null                          // } rechte Teilmenge sortieren --> n/2 Op
            && pivot_prev.next != null)                  // |
        quicksort(pivot_prev.next.next, end, comparator);// |
}
```

Jeder Partitionierungsvorgang (*partitionLast*) erfordert O(n) Operationen (ein Durchlauf auf dem Array). Im Durchschnitt unterteilt jede Partitionierung das Array in zwei Teile (was sich auf log n Operationen summiert). Insgesamt haben wir **O(n*log n)** Operationen.

Suchalgorithmus: Linearer Suchalgorithmus

```
public TransactionList searchByName() {
    String query = c.readString("Enter name to search: ");  // 1 Operation
    TransactionList result = new TransactionList();  // 1 Operation
```

```
    Node node = head;  // 1 Operation
    while (node.next != null) {                                                  // |
        if (node.data.getName().toLowerCase().contains(query.toLowerCase())) {  // |
            result.add(node.data);                                               // } n Operationen
            result.balance += node.data.getAmount();                             // |
        }                                                                        // |
        node = node.next;                                                        // |
    }
    if (node.data.getName().toLowerCase().contains(query.toLowerCase())) {       // |
        result.add(node.data);                                                   // } 1 Operation
        result.balance += node.data.getAmount();                                 // |
    }
    return result;
}
```

- Die Zeitkomplexität für die lineare Suche wird mit O(n) angegeben, da jedes Element in der Array nur einmal verglichen wird
- Im besten Fall ist die Komplexität O(1), wenn das Element beim ersten Index gefunden wird
- Im schlechtesten Fall ist die Komplexität O(n), wenn das Element am letzten Index gefunden wird oder das Element nicht im Array vorhanden ist.

6. Quellen

QuickSort on Singly Linked List. (2013, June 28). GeeksforGeeks. https://www.geeksforgeeks.org/quicksort-on-singly-linked-list/