SmartDivide

Michał Matejczuk
Felicja Warno
Wiktor Wierzchowski

Opisy klas i metod:

1. Klasa Member – jest obiektem osoby, która może w klasie Bill mieć zależności finansowe względem innych Memberów.
   1.1. Pola:
      1.1.1. String name – nazwa Membera (imię). Podawane jako argument w konstruktorze klasy.
2. Klasa Transaction – klasa reprezentująca transakcję finansową między dwoma Memberami.
   2.1. Pola:
      2.1.1. BigDecimal amount – wysokość transakcji (w zł)
      2.1.2. Member debtor – Member będący dłużnikiem, czyli otrzymujący pieniądze w transakcji
      2.1.3. Member creditor – Member będący wierzycielem, czyli dający pieniądze dłużnikowi.
      2.1.4. LocalDateTime time – czas transakcji
      2.1.5. String title – tytuł transakcji

      Wartości wszystkich pól przekazywane są w konstruktorze.

3. Klasa Bill – klasa przechowująca informacje o Memberach i transakcjach między nimi, rozwiązuje też problem najprostszego uregulowania rachunku między Memberami.
   3.1. Pola:
      3.1.1. String title – tytuł rachunku. Wartość podana w konstruktorze.
      3.1.2. int V – liczba wszystkich Memberów należących do rachunku.
      3.1.3. Map<Member, BigDecimal> debtList – mapa przechowująca łączny bilans transakcji dla każdego Membera w rachunku. Bilans oznacza sumę wysokości wszystkich długów i zawierzeń Membera (bilans = zawierzenia – długi).
      3.1.4. List<Member> members – lista wszystkich Memberów uczestniczących w rachunku.
      3.1.5. Map<Member, BigDecimal> negatives – mapa przechowująca Memberów z ujemnym bilansem transakcji i wartością bezwzględną tego bilansu.
      3.1.6. Map<Member, BigDecimal> positives - mapa przechowująca Memberów z dodatnim bilansem transakcji i wartością bezwzględną tego bilansu.
      3.1.7. List<Transaction> transactionHistory – lista wszystkich transakcji (Transaction) przeprowadzonych w ramach rachunku.
      3.1.8. Map<Member[], BigDecimal> solution – mapa przechowująca optymalne rozliczenie rachunku. Klucz Member[] jest dwuelementową listą Memberów, gdzie pierwszy to dłużnik drugi to wierzyciel, a wartość to wysokość długu, który należy uregulować.
      3.1.9. static int refoundCounter –
   3.2. Klasy wewnętrzne:
      3.2.1. ReturnTypeForSolveBill – typ, który zwraca metoda solveBill. Przechowuje pola:
         3.2.1.1. int count – liczba transakcji wykonanych dotychczas w metodzie solveBill.

3.2.1.2. Map<Member[], BigDecimal> transactions – mapa transakcji wykonanych dotychas w metodzie solveBill (taki sam format jak pole solution w klasie zewnętrznej).

Wartości wszystkich pól przekazywane są w konstruktorze.

3.3. Metody:

3.3.1. void addMember(Member a) – dodaje Membera z parametru do rachunku.

3.3.2. void addDebt(String debtTitle, BigDecimal amount, Member debtor, Member creditor) – dodaje dług do rachunku. Przyjmuje tytuł długu debtTitle, jego wysokość amount oraz instancje Member dłużnika i wierzyciela – debtor, creditor.

3.3.3. void addDebtForTime(String debtTitle, BigDecimal amount, Member debtor, Member creditor, LocalDateTime time) – jak wyżej; przyjmuje dodatkowo czas transakcji, który w metodzie addMember jest domyślnie ustawiony na now().

3.3.4. void createPosNegMaps() – buduje mapy negatives i positives na podstawie mapy debtList.

3.3.5. void minTransfers() – buduje mapę solution korzystając z createPosNegMaps() oraz solveBill().

3.3.6. void removeMember(Member selectedMember) – usuwa danego Membera rachunku.

3.3.7. ReturnTypeForSolveBill solveBill(Map<Member, BigDecimal> negatives, Map<Member, BigDecimal> positives) – główny algorytm znajdujący najmniejszą konieczną liczbę transakcji do uregulowania długu opisanego parametrami positives i negatives (w pierwszej iteracji to wartości pól o tej samej nazwie) oraz listę transakcji, które trzeba wykonać do uregulowania długu w najmniejszej liczbie transakcji. Działa na mapach positives i negatives, iteruje po wszystkich możliwych kombinacjach pierwszej transakcji między Memberem w positives i Memberem w negatives. Wysokość tej transakcji jest równa minimum z bilansu tych dwóch Memberów. Dzięki temu jeden z Memberów będzie już „rozliczony". Transakcję tę dodajemy do mapy transactions, która w takim samym formacie jak pole solution przechowuje mapę transakcji potrzebną do rozliczenia. Zmienna int count oznacza liczbę transakcji konieczną, aby dojść do tego miejsca w rekurencji. Usuwamy tego uczestnika z lokalnej mapy newPositives/newNegatives (w zależności, z której listy był rozliczony Member) a dla drugiego Membera w drugiej liście odpowiednio zmniejszamy bilans po czym wywołujemy solveBill() dla nowopowstałych list. Z otrzymanej przez rekurencję instancji ReturnTypeForSolveBill odczytujemy, ile transakcji potrzebowała dana ścieżka, aby otrzymać pełne rozliczenie. Porównujemy lokalne count z polem count ze zwróconej instancji ReturnTypeForSolveBill i jeśli zwrócona liczba jest mniejsza, aktualizujemy lokalne count i transactions. Na koniec zwracamy nową instancję ReturnTypeForSolveBill z parametrami count+1 (zwiększamy bo wykonaliśmy transakcję) i transactions.

3.3.8. void addGroupDebt(String debtTitle, BigDecimal amount, Member creditor, Member... debtors) – tworzy dług grupowy, gdzie jeden Member (creditor) jest wierzycielem dla jakiejś liczby dłużników (debtors). amount jest sumą tego długu (czyli ile zawierzył creditor. Funkcja dzieli kwotę amount po równo między dłużników i tworzy dla nich pojedyncze długi. Jeśli równy podział z dokładnością do groszy nie jest możliwy, dzielimy po równo największą podzielną kwotę mniejszą od amount, a następnie zwiększamy dług kilku uczestnikom o grosz tak, aby suma dodanych groszy była różnicą między amount a największą podzielną kwotą mniejszą od amount.

3.3.9. void mergeBills(Bill otherBill) – łączy instancję Bill, dla której wywołana została metoda z instancją Bill otherBill tak, aby ta instancja zaczęła przechowywać też wszystkich Memberów i ich zależności z otherBill.

3.3.10. void mergeBills(Bill otherBill) – łączy instancję Bill, dla której wywołana została metoda z instancją Bill otherBill tak, aby ta instancja zaczęła przechowywać też wszystkich Memberów i ich zależności z otherBill.

4. Klasa App - klasa główna w której następuje zapis i odczyt stanu aplikacji do/z pliku "savefile.txt" oraz zainicjowanie pierwszej sceny StartupScene.fxml

   4.1. Pola:

      4.1.1. Serialize reader - obiekt klasy Serialize używany do odczytu

      4.1.2. Serialize writer - obiekt klasy Serialize używany do zapisu

   4.2. Metody:

      4.2.1. void start() - inicjacja aplikacji

5. Klasa Holder - klasa przekazująca informacje pomiędzy scenami, przekazywana jako argument konstruktora kontrolera następnej sceny

   5.1. Pola:

      5.1.1. ArrayList<Bill> bills - pole przechowujące informacje o rachunkach w aplikacji

      5.1.2. ArrayList<Member> members - pole przechowujące informacje o osobach w aplikacji

   5.2. Metody:

      5.2.1. konstruktor Holder() - przy pierwszym uruchomieniu aplikacji lub usunięciu pliku "savefile.txt" następuje wywołanie tego konstruktora który tworzy kilka przykładowych obiektów

      5.2.2. konstruktor Holder(List<Bill> bills, List<Member> members) - używany przy wczytywaniu danych z pliku

      5.2.3. boolean checkBillExistence(String newBillTitle) - sprawdza czy istnieje konflikt nazw rachunków z nowym

      5.2.4. boolean checkMemberExistence(String newMemberName) - sprawdza czy istnieje konflikt imion osób z nową

      5.2.5. void addBill(Bill bill) - dodaje rachunek do listy rachunków

      5.2.6. void addMember(Member member) - dodaje osobę do listy osób

      5.2.7. ArrayList<Bill> getBills() - zwraca liste rachunków

      5.2.8. ArrayList<Member> getMembers() - zwraca liste osób

6. Klasa StartupSceneController - klasa kontroler dla sceny StartupScene.fxml obsługująca interface

   6.1. Pola:

      6.1.1. Holder holder - przechowuje holder

      6.1.2. Stage stage - pole do inicjacji sceny

      6.1.3. Scene scene - pole do inicjacji sceny

6.1.4. Parent root - pole do inicjacji sceny

6.1.5. Bill selectedBill - przechowuje rachunek wybrany przez użytkownika

6.1.6. ArrayList<Bill> selectedBills - przechowuje wszystkie rachunki które były wybrane

6.2. Metody:

6.2.1. konstruktor StartupSceneController(Holder holder) -

6.2.2. konstruktor StartupSceneController() - przypisuje polu holder nowy bezargumentowo zainicjowany holder

6.2.3. void initialize(URL url, ResourceBundle resourcebundle) - inicjuje interface sceny

6.2.4. void controllerAddBill() - dodaje rachunek

6.2.5. void controllerDeleteBill() - usuwa rachunek

6.2.6. void openBill(ActionEvent event) - przejście do sceny ScenaOgolna.fxml

6.2.7. void setHolder(Holder holder) - ustala nowy holder

6.2.8. void mergeSelectedBills - łączy wybrany rachunek z rachunkiem który wybrany był ostatnio przed nim

7. Klasa ScenaOgolnaController - klasa kontroler dla sceny ScenaOgolna.fxml obsługująca interface

7.1. Pola:

7.1.1. Holder holder - przechowuje holder

7.1.2. Bill bill - przechowuje który rachunek jest obecnie otwarty

7.1.3. Stage stage - pole do inicjacji sceny

7.1.4. Scene scene - pole do inicjacji sceny

7.1.5. Parent root - pole do inicjacji sceny

7.1.6. ArrayList<Member> selectedTransactionDebtors - przechowuje wszystkie osoby które były wybrane jako dłużnicy

7.1.7. Member selectedTransactionCreditor - przechowuje osobe wybraną jako wierzyciela

7.1.8. Transaction currentTransaction - przechowuje wybraną obecnie tranzakcje

7.2. Metody:

7.2.1. konstruktor ScenaOgolnaController(Holder holder, Bill selectedBill) - tworzy instacje konstruktora o zadanych polach

7.2.2. void initialize(URL url, ResourceBundle resourcebundle) - inicjuje interface sceny

7.2.3. void changedTransaction(Obsercable observable) - listener obsługujący zmiane wybranej transakcji

7.2.4. void displayTransactionDetails() - przedstawia informacje o tranzakcji

7.2.5. void changedCreditor - listener obsługujący zmiane wybranego wierzyciela

7.2.6. void changedDebtor - listener obsługujący zmiane wybranego dłużnika i dodający nowego do listy wszystkich wybranych

7.2.7. void closeBil - przejście do sceny StartupScene.fxml

7.2.8. void openMembersView - przejście do sceny ScenaDokladna.fxml

7.2.9. void changeBillName - zmienia nazwe rachunku

7.2.10. void displayBillName - wyświetla obecną nazwe rachunku

7.2.11. void controllerAddDebt - dodaje nową transakcję

8. Klasa ScenaDokladanController - klasa kontroler dla sceny ScenaDokladanController.fxml obsługująca interface

8.1. Pola:

8.1.1. Holder holder - przechowuje holder

8.1.2. Bill bill - przechowuje który rachunek jest obecnie otwarty

8.1.3. Stage stage - pole do inicjacji sceny

8.1.4. Scene scene - pole do inicjacji sceny

8.1.5. Parent root - pole do inicjacji sceny

8.1.6. Member selectedMember - wybrana przez użytkownika osoba z rachunku

8.1.7. Member selectedSavedMember - wybrana przez użytkownika osoba spośród zapisanych w holderze

8.1.8. Member[] selectedDebt - tablica osób względem których winna jest wybrana przez użytkownika osoba

8.2. Metody:

8.2.1. konstruktor ScenaDokladanController(Holder holder, Bill selectedBill) - tworzy instacje konstruktora o zadanych polach

8.2.2. void initialize(URL url, ResourceBundle resourcebundle) - inicjuje interface sceny

8.2.3. void changedDebt(Obsercable observable) - listener obsługujący zmiane pola selectedDebt

8.2.4. void updateDebtList() - aktualizuje przedstawiane informacje o użytkowniku

8.2.5. void changedMember - listener obsługujący zmiane wybranej do podglądu osoby

8.2.6. void changedSavedMember - listener obsługujący zmianej osoby z listy zapisanych

8.2.7. void addMember() - dodaje nowego członka do rachunku

8.2.8. void addSavedMember() - dodaje osobę zapamiętaną w holderze do obencego rachunku

8.2.9. void deleteMember() - usuwa członka rachunku, pod warunkiem że ma uregulwany bilans

8.2.10. void forgetMember() - usuwa członka z listy zapamiętanych pod warunkiem że nie jest członkiem żadnego rachunku

8.2.11. void deleteDebt() - reguluje wybraną należność

8.2.12. void switchToScenaOgolna - przejście do sceny ScenaOgolna.fxml

8.2.13. void displayBillName() - wyświetla nazwę obecnego rachunku

8.2.14. void changeMemberName() - zmienia nazwę wybranej obecnie zapisanej osoby