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

Wartości wszystkich pól przekazywane są w konstruktorze.

1. Klasa Bill – klasa przechowująca informacje o Memberach i transakcjach między nimi, rozwiązuje też problem najprostszego uregulowania rachunku między Memberami.
   1. Pola:
      1. String title – tytuł rachunku. Wartość podana w konstruktorze.
      2. int V – liczba wszystkich Memberów należących do rachunku.
      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).
      4. List<Member> members – lista wszystkich Memberów uczestniczących w rachunku.
      5. Map<Member, BigDecimal> negatives – mapa przechowująca Memberów z ujemnym bilansem transakcji i wartością bezwzględną tego bilansu.
      6. Map<Member, BigDecimal> positives - mapa przechowująca Memberów z dodatnim bilansem transakcji i wartością bezwzględną tego bilansu.
      7. List<Transaction> transactionHistory – lista wszystkich transakcji (Transaction) przeprowadzonych w ramach rachunku.
      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.
      9. static int refoundCounter –
   2. Klasy wewnętrzne:
      1. ReturnTypeForSolveBill – typ, który zwraca metoda solveBill. Przechowuje pola:
         1. int count – liczba transakcji wykonanych dotychczas w metodzie solveBill.
         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.

* 1. Metody:
     1. void addMember(Member a) – dodaje Membera z parametru do rachunku.
     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. 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().
     4. void createPosNegMaps() – buduje mapy negatives i positives na podstawie mapy debtList.
     5. void minTransfers() – buduje mapę solution korzystając z createPosNegMaps() oraz solveBill().
     6. void removeMember(Member selectedMember) – usuwa danego Membera rachunku.
     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.
     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.
     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.

1. Klasa Serialize – klasa służąca do zapisu i odczytu danych z pliku.
   1. Pola:
      1. List<Bill> bills – lista rachunków (Bill) do zapisu/odczytu
      2. List<Member> members – globalna lista Memberów do zapisu/odczytu

Wartości wszystkich pól przekazywane są w konstruktorze do zapisu.

* 1. Metody:
     1. Serialize(ArrayList<Bill> bills, ArrayList<Member> members) – konstruktor do zapisu, przyjmuje jako parametry wartości bills i members.
     2. Serialize (String file) – konstruktor do odczytu, przyjmuje nazwę pliku file i odczytuje z niego dane po czym przypisuje je do bills i members.
     3. Serialize readState(String file) – odczytuje dane z pliku o nazwie file i zwraca obiekt typu Serialize zawierający wszystkie dane z odczytu.
     4. void saveState(String file) – zapisuje dane programu do pliku o nazwie file.