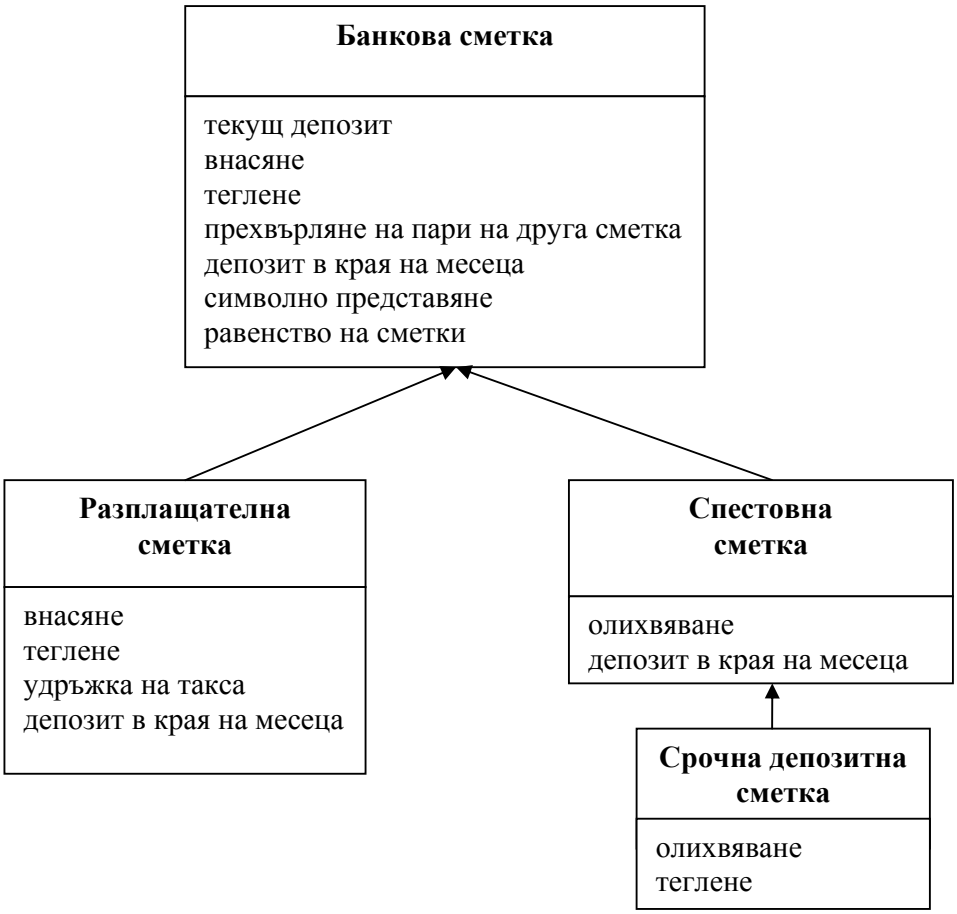
Упражнение №4
Класове и обекти

Разглеждаме банка, която предлага на своите клиенти три вида сметки:

1. Разплащателна сметка – без лихва, позволява малък брой банкови операции за месец без такса и удръжка на комисионна за всяка допълнителна банкова операция.
2. Спестовна сметка – месечно изчисляване на лихвата (лихвата се изчислява като се използва сумата от последния ден на месеца).
3. Срочна депозитна сметка – като спестовна сметка, но позволява парите да стоят в сметката за определен брой месеци и налага глоба за по-ранно изтегляне на пари.



I. Създайте клас **`BankAccount`** (**`Банкова сметка`**), който съдържа общите характеристики за банкови сметки: **`номер на сметка`** (тип **`int`**, първата сметка е с номер 1 и всяка следваща е с един номер по-голям от предшестващата), **`последен използван номер`** (тип **`int`**, статично поле, инициализирано с 0), **`име на притежателя на сметката`** (тип **`String`**) и **`депозит`** (тип **`double`**). Всички полета са с **`private`** достъп.

1. Добавете конструктор с един параметър – име на притежателя на сметка. В конструктора увеличете последния номер на сметка и го използвайте, за да инициализирате номера на сметката, установява името на притежателя на сметката и инициализирайте депозита с нула.
2. Добавете конструктор с два параметъра – име на притежателя и депозит, който извършва същите действия като конструктора с един параметър, но инициализира депозита с начална стойност.

3. Добавете метод за достъп **getBalance**, който реализира операцията **текущ депозит**, като връща текущия депозит от тип **double**.
4. Напишете метод **deposit** с параметър – пари за внасяне, който реализира операцията **внасяне на пари** по следния алгоритъм:

   ```
   депозит ← депозит + пари за внасяне
   ```

5. Включете метод **withdraw** с един параметър – пари за теглене, който реализира операцията **теглене на пари** по следния алгоритъм:

   ```
   ако количеството на парите за теглене е по-малко от депозита
      депозит ← депозит - пари за теглене
   ```

6. Включете метод **transfer** с два параметъра – банкова сметка-получател и количество пари за прехвърляне, който реализира операцията **прехвърляне на пари на друга сметка** по следния алгоритъм:

   ```
   изтегли количаството пари от текущата сметка
   внеси количеството пари в сметката-получател
   ```

7. Напишете метод **endOfMonth**, който изчислява **депозита в края на месеца**, като връща депозита (депозитът не се олихвява).
8. Предефинирайте метода **toString** на класа **Object**, който дава **символното представяне на банковата сметка** – името на притежателя и депозита.
9. Предефинирайте метода **equals** на класа **Object**, като разглеждайте равенство на две сметки спрямо техните депозити.

   ```
   public boolean equals(Object obj)
   ```

II. Създайте клас **CheckingAccount** (**Разплащателна сметка**) като наследник на базовия клас **BankAccount**, който не начислява лихва, позволява малък брой банкови операции за месец без такса и удържа комисионна за всяка допълнителна банкова операция. Използва допълнителни полета: **константен брой банкови операции за месец без такса** (тип **int**, инициализирано с 3), **константна комисионна за всяка допълнителна банкова операция** (тип **double**, инициализирано с 2.0) и **брояч на банковите операции** (тип **int**). Всички полета са с **private** достъп.

1. Добавете конструктор с два параметъра – име на притежателя и депозит, който установява името на притежателя и стойността на депозита и инициализира брояча на банковите операции в нула.
2. Предефинирайте метода **deposit**, като обновите брояча на банковите операции по следния алгоритъм:

   ```
   увеличете с 1 брояча на банкови операции
   внесете парите в сметката
   ```

3. Предефинирайте метода **withdraw**, като обновите брояча на банковите операции по следния алгоритъм:

   ```
   увеличете с 1 брояча на банкови операции
   изтеглете парите от сметката
   ```

4. Включете метод **deductFees**, който не връща стойност и реализира операцията **удръжка на комисионна** по следния алгоритъм:

   ```
   ако броячът на банкови операции надвиши броя на разрешените операции
      изчислете такса = комисионна .
                        (брояч на банкови операции – брой разрешени операции)
   установете в нула брояча на банковите операции
   ```

5. Предефинирайте метода **endOfMonth** на базовия клас **BankAccount**, който в края на месеца удържа комисионна по следния алгоритъм:

```
удържете комисионна
върнете текущия депозит
```

III. Създайте клас **SavingAccount** (**Спестовна сметка**) като наследник на базовия клас **BankAccount**, който използва допълнително поле **лихвен процент** (тип **double**, **private** достъп), за да начисли лихва.

1. Добавете конструктор с два параметъра, който установява името на притежателя на сметката и лихвения процент.
2. Включете метод **addInterest**, който не връща стойност и реализира операцията **олихвяване** по следния алгоритъм:

```
лихва = текущ депозит . лихвен процент / 100
внеси лихвата към сметката
```

3. Предефинирайте метода **endOfMonth** на базовия клас **BankAccount**, който в края на месеца начислява лихвата по следния алгоритъм:

```
олихвете сметката
върнете текущия депозит
```

4. Създайте клас **TimeDepositAccount** (**Срочна депозитна сметка**) като наследник на класа **SavingAccount**, който позволява парите да стоят в сметката за определен брой месеци и налага глоба за по-ранно теглене на пари. Използвайте допълнителните полета **срок на депозита** (брой месеци, тип **int**) и **константна глоба** при по-ранно теглене на пари (тип **double**, инициализирано с 20.0).
5. Добавете конструктор с три параметъра, който установява името на притежателя, лихвения процент и срока на депозита.
6. Предефинирайте метода **addInterest**, който следи срока на депозита по следния алгоритъм:

```
увеличете с 1 срок на депозит
олихвете сметката
```

7. Предефинирайте метода **withdraw**, който налага глоба при теглене на пари, ако срокът на депозита не е изтекъл, по следния алгоритъм:

```
ако срок на депозит > 0
   изтегли глобата от сметката
```

8. Създайте драйверен клас **Test**. Декларирайте три сметки от тип **BankAccount**. Създайте сметка за университета от тип **CheckingAccount** и за студенти от тип **SavingAccount** и **TimeDepositAccount**. От сметката на университета прехвърлете еднаква сума пари в студентските сметки и проверете дали депозитите в студентските сметки са еднакви. Изтеглете еднаква сума пари от студентските сметки и отново проверете равенството на техните депозити; изтеглете пари от университетската сметка. Внесете пари в трите сметки и изчислете сумата от депозитите в трите сметки в края на месеца.