Inlämningsuppgift 1

Inlämningsuppgifterna i denna kurs går ut på att ta fram ett enkelt system för en fiktiv bank. Varje inlämningsuppgift bygger vidare på den föregående så en bra grund ar viktig.

Banken har ett antal kunder och varje kund kan ha ett eller flera olika bankkonton. Banken erbjuder för tillfället endast konton av typen sparkonto med räntesats 1%.

I denna uppgift ska du börja med att skapa en klass som definierar sparkonto (SavingsAccount), en klass som definierar kund (Customer) samt en klass (BankLogic) som innehåller en lista med kunder.

OBS! Du får tillgång till en testklass (TestBank.java) som används för att testa dina klasser när du är klar med implementationen. Ändringar i denna testklass kan komma att göras under kursens gång så använda alltid den senaste versionen för att testa programmet innan du lämnar in.

I banken ska man kunna:

- få en lista med bankens kunder (för-, efternamn, personnummer)
- lägga till en ny kund med ett unikt personnummer
- ändra en kunds namn (personnummer ska inte kunna ändras)
- ta bort en befintlig kund, eventuellt befintliga konton avslutas
- skapa sparkonton till en befintlig kund, ett unikt kontonummer genereras (första kontot får nummer 1001, nästa 1002 osv.)
- konton ska även kunna avslutas, information om borttagning visas och kontot tas bort

För en viss kund ska man kunna utföra följande:

- se information om vald kund inklusive alla konton (kontonummer, saldo, kontotyp, räntesats)
- sätta in pengar på ett konto
- ta ut pengar från kontot (men bara om saldot täcker uttagsbeloppet)

Klassdesign

SavingsAccount

Börja med att implementera klassen ***SavingsAccount*** som ska hantera följande information: **saldo**, **räntesats**, **kontotyp** (Sparkonto) och **kontonummer** (det kan inte finnas flera konton med samma kontonummer).

Man ska till exempel kunna utföra transaktioner (insättning/uttag), hämta kontonummer, beräkna ränta (saldo * räntesats/100) samt presentera kontot (kontonummer, saldo, kontotyp, räntesats).

Skapa gärna upp en testklass och testa klassen och dess metoder innan du går vidare.

Customer

Klassen ***Customer*** ska hantera följande information: kundens **namn**, **personnummer** samt en lista med kundens alla **konton**. Om du känner att du vill lägga listan med konton i BankLogic istället är det okej men du ska vara medveten om att det blir en lite mer komplicerad lösning.

Man ska till exempel kunna ändra kundens namn samt hämta information om kunden (personnummer, för- och efternamn samt hämta information om kundens konton (kontonummer, saldo, kontotyp, räntesats)).

Dessutom ska man kunna hantera kundens konton, ifall du nu väljer att lägga listan med konton i klassen Customer.

Skapa upp en testklass och testa klassen och dess metoder innan du går vidare.

BankLogic

Klassen ***BankLogic*** ska innehålla en lista med alla inlagda **kunder**. Klassen ska innehålla ett antal publika metoder som hanterar kunder och dess konton. Du kommer troligtvis att skapa ett antal hjälpmetoder, privata metoder, men de publika metoderna finns definierade nedan.

public ArrayList<String> getAllCustomers()

- Returnerar en ArrayList<String> som innehåller en presentation av bankens alla kunder (namn och personnummer)

public boolean createCustomer(String name, long pNo)

- Skapar upp en ny kund med namnet name samt personnumer pNo, kunden skapas endast om det inte finns någon kund med personnummer pNo.
- Returnerar true om kund skapades annars returneras false.

public ArrayList<String> getCustomer(long pNo)

- Returnerar en ArrayList<String> som innehåller informationen om kunden inklusive dennes konton. Första platsen i listan är reserverad för kundens namn och personnummer sedan följer informationen om kundens konton.
- Returnerar null om kunden inte fanns

public boolean changeCustomerName(String name, long pNo)

- Byter namn på kund med personnummer pNo till name.
- Returnerar true om namnet ändrades annars returnerar false (om kunden inte fanns).

public ArrayList<String> deleteCustomer(long pNo)

- Tar bort kund med personnummer pNo ur banken, alla kundens eventuella konton tas också bort och resultatet returneras. Listan som returneras ska innehålla information om alla konton som togs bort, saldot som kunden får tillbaka samt vad räntan blev.
- Returnerar null om ingen kund togs bort

public int createSavingsAccount(long pNo)

- Skapar ett konto till kund med personnummer pNo.
- Returnerar kontonumret som det skapade kontot fick alternativt returneras -1 om inget konto skapades.

public String getAccount(long pNo, int accountId)

- Returnerar en String som innehåller presentation av kontot med kontonnummer accountId som tillhör kunden pNo (kontonummer, saldo, kontotyp, räntesats).
- Returnerar null om konto inte finns eller om det inte tillhör kunden

public boolean deposit(long pNo, int accountId, double amount)

- Gör en insättning på konto med kontonnummer accountId som tillhör kunden pNo.
- Returnerar true om det gick bra annars false.

public boolean withdraw(long pNo, int accountId, double amount)

- Gör ett uttag på konto med kontonnummer accountId som tillhör kunden pNo.
- Returnerar true om det gick bra annars false.

public String closeAccount(long pNr, int accountId)

- Avslutar ett konto med kontonnummer accountId som tillhör kunden pNo. När man avslutar ett konto skall räntan beräknas som saldo multiplicerat med ränta/100.
- OBS! Enda gången ränta läggs på är när kontot tas bort eftersom årsskiften inte hanteras i denna version av systemet.
- Presentation av kontots ska returneras inklusive räntan man får på pengarna.
- Returnerar null om inget konto togs bort

För godkänt (G) på denna inlämningsuppgift krävs:

- Inga större fel finns i din lösning.
- Att lösningen följer de objektorienterade principerna.
- Alla källkodsfiler ska vara **kommenterade** för att underlätta för handledaren som ska sätta sig in i din lösning. Alla filer ska dessutom innehålla ditt förnamn, efternamn och framför allt ditt **användarnamn**.

För väl godkänt (VG) på denna uppgift krävs dessutom:

- Att lösningen är genomtänkt och väl objektorienterad, tänk framför allt på:
  - Inkapsling av data
  - Relationerna mellan klasser
- En **rapport** där du **motiverar** dina val och **reflekterar** över din lösning så det framgår att du har en god förståelse av det du har gjort.
- Att källkoden följer kodningsregler och standard för dokumentation och kommentering. **Alla filer ska** dessutom innehålla ditt förnamn, efternamn och användarnamn.