Mittuniversitetet
MID SWEDEN UNIVERSITY

Lab 2 – Bankaffärer

Objektorienterad programmering i C++

Dynamisk minneshantering, smarta pekare, filhantering och samarbetande klasser med klassinstanser som datamedlemmar.

Lab 2 - Bankaffärer

Denna labuppgift introducerar samarbetande *klasser* och instanser av klasser, *objekt* i form av ett program för hantering av kunder och deras konton i den välkända *Ebberöds Bank*. För alla klasser gäller att klassdefinition ska finnas i en fil med namnet <klassnamn>.h och att klassens implementation ska finnas i en fil med namnet <klassnamn.cpp>. Du ska arbeta stegvis genom att skriva och testa varje klass för sig för att sedan integrera klasserna till en färdig lösning. Till varje klass ska du skriva ett testprogram som visar att den fungerar som det är tänkt. Lämpligt är att skapa ett speciellt projekt för utveckling/testning av varje enskild klass.
Kompilera och kör testprogrammet så fort du tillfört ny funktionalitet i klassen.

Steg1
Skriv klassen `Konto` som hanterar uppgifter om ett bankkonto. Datamedlemmar ska finnas för

- kontonummer
- aktuellt saldo
- beviljad kredit(default 0.00 kr)

Publika medlemsfunktioner ska finnas för att

- sätta in på kontot
- ta ut från kontot
- få uppgift om aktuell kredit
- ändra beviljad kredit
- få uppgift om aktuellt saldo
- få uppgift om disponibelt belopp
- få uppgift om kontonummer

Saldo är den faktiska kontoställningen. Om jag har en beviljad kredit så kan mitt saldo vara negativt.

Med *disponibelt belop*p menas hur mycket pengar som kan tas ut från kontot, eventuell kredit inräknad. Matematiskt kan det uttryckas som

disponibelt belopp = aktuellt saldo + beviljad kredit

Exempel:
Nisse öppnar ett konto med en beviljad kredit på 10000 kr. Han sätter in 3000 kr men nästa dag går bilen sönder och han måste ta ut 9000 kr för att byta övre ljuddämparlagret. Efter det är
saldot 3000 – 9000 = – 6000 kr och
disponibelt belopp – 6000 + 10000 = 4000 kr

Uttag ska naturligtvis tillåtas bara om täckning finns (kredit inräknad).

Skapa ett separat testprogram (ska ej redovisas).

Steg 2

Skriv klassen `BankKund` som hanterar konton för en viss kund i en bank. Klassen ska kunna hantera max 3 konton för en viss person (ingen kund kan ha fler än 3 konton i denna speciella bank).
Datamedlemmar (attribut):

- kundens efternamn och förnamn
- kundens personnummer
- någon typ av kontainer som innehåller tre smarta pekare till `Konto`-objekt . Motsvarande `Konto`-objekt ska skapas och förstöras dynamiskt efter behov.

Publika medlemsfunktioner ska finnas för att

- få uppgift om kundens namn
- få uppgift om kundens personnummer
- få uppgift om antalet konton för kunden
- få uppgift om kundens olika kontonummer
- få uppgift om saldo, kredit och tillgängligt belopp för ett visst kontonummer
- få uppgift om kundens totala tillgångar
- skapa nytt konto
- avsluta (ta bort) konto
- ta ut från ett visst konto
- sätta in på ett visst konto
- ändra kreditgräns
- skriva all kontoinformation till en för en viss kund till en fil
- läsa in all kontoinformation för en viss kund från en fil

Kundinformationen ska kunna sparas till en fil för att senare kunna läsas in. Namn på kontofilen för en viss kund ska vara kundens personnummer.knt, t.ex. `5302058291.knt`.

Utnyttja gärna boolska returvärden för att indikera om en viss operation kunde genomföras eller inte, t.ex. uttag från ett konto.

Steg 3
Skriv klassen `Bank`. Klassens uppgift är att kapsla in all hantering av bankkunder och konton. Alla operationer på dessa objekt ska göras via det publika gränssnittet i `Bank`.
Utöver de operationer som finns i BankKund ska man kunna få kundens 'kontobild', dvs. en samlad utskrift av kontonummer, saldo, kredit och tillgängligt belopp för varje konto och dessutom summeringar för samtliga saldon.
Ebberöds Bank har små resurser och kan bara betjäna en kund åt gången. Därför behövs endast ett Bankkund-objekt existera åt gången. Detta ska implementeras så att Bank som privat datamedlem innehåller en smart pekare till en BankKund. Denna pekare ska peka på det BankKund-objekt som f.n. hanteras. Detta objekt, "den aktuella kunden", skapas dynamiskt i Bank när det ska behandlas och alla data för kunden läses från motsvarande konto-fil. Alla transaktioner ska ske relativt den aktuella kunden. När den aktuella kunden är färdighanterad ska motsvarande kontofil uppdateras/skrivas om och BankKund-objektet förstöras.
Lösningen som ska redovisas är ett menystyrt program som instansierar ett Bank-objekt. Via detta objekt ska alla operationer på kunder och konton kunna utföras.

Krav på lösningen

- Vid programstart ska kunna välja mellan att
    - skapa och hantera ny kund, eller
    - att hantera en redan befintlig kund.

  Efter något av dessa steg ska det alltså finns en ”aktuell kund” som motsvaras av ett dynamiskt skapat BankKund-objekt som ägs av en smart pekare.
- Alla transaktioner och hantering av konton sker alltid mot den aktuella kunden.
- Programmet ska skapa och använda exakt ett Bank-objekt.
- För klasserna Konto, Bankkund och Bank gäller att de är ”tysta” klasser. Inga utskrifter eller inmatningar ska göras i dessa. All kommunikation ska ske genom parameteröverföringar och returvärden.
- All kommunikation med användaren ska ske från testprogrammet.
- Alla objekt som skapas dynamiskt ska deallokeras.

Redovisning
Zippad fil med alla källkod. Alla ingående filer ska inledas med en kommentardel med filnamn, vilken uppgift den har samt namnet på dig själv.