**H2**

**CASE**

**objektorienteret programmering**

**C#**

**Indledning:**

**Mål:**

* at kunne arbejde med objektorienteret programmering (OOP) i C# herunder klasser og metoder, analysere og oversætte virkelighed (OOA), benytte de datastrukturer som er relevante for løsning af opgaven, at kunne arbejde sammen om en programmeringsopgave.
* At opnå et bredt kendskab til C#
* Udvide kendskabet til og bruge database normalisering, SQL, relationer, tabeller etc.
* at kunne lave færdig statisk klasse med databasekommunikation der kan bruges på de følgende moduler.
* At kunne arbejde med integration mellem database SQL og C#
* at kunne planlægge og følge op på sit arbejde. At være bevidst omkring forskellige løsninger og deres fordele og ulemper. At kunne diskutere på et fagligt niveau.
* at kunne formidle sin viden mundtligt og visuelt

**Forudsætninger:**

Kendskab til SQL, design af database, tabeller og relationer, C# grundlæggende begreber og kodestrukturer og repræsentation (klasser, metoder, arv, indkapsling, lister, overload, delegates, override osv.)

**Arbejdsform:**

Gruppe (3-4 personer)

**Støtte og input undervejs:**

Vi vil når det er nødvendigt lave relevante oplæg til hvor i processen I nu er kommet. Ud over det skal I bruge tidligere kode I har lavet, Google og hinanden. Bagest er vedhæftet eksempler/kodestumper med basale databasefunktioner/oplag som I kan tage udgangspunkt i.

Vi starter med lidt repetition for at sikre at alle har de forudsætninger som vi kræver.

**Evaluering:**

I skal fremlægge jeres løsning for de andre i klassen, fredag i anden uge af forløbet.

**Opgaveoplæg/kravsspecifikation**

Vi har valgt at I skal arbejde med en bank, dens kunder og deres konti.

Der er følgende ønsker til systemet (kravsspecifikation):

1. Man skal kunne oprette en kunde, slette en kunde
2. Man skal kunne få vist en kundeoversigt (herunder en sorteret form efter efternavn eller kontonummer)
3. Man skal kunne få vist en kundes konti.
4. Man skal kunne oprette en konto og slette en konto.
5. Hver konto skal have en kontotype og en rentesats tilknyttet
6. En kunde kan godt have flere konti.
7. En konto kan kun være ejet af én kunde
8. Både kunder og konti skal have en oprettelsesdato.
9. Man skal kunne spørge om saldo på en kundekonto.
10. Man skal kunne hæve og indsætte penge på en konto.
11. Man skal kunne udskrive transaktioner der er knyttet til en bestemt konto, f.eks. transaktionsnummer, beløb, dato.

**TIP!** I skal arbejde objektorienteret, dvs. en kunde er et objekt, en konto er et objekt osv. Som regel siger man at navneord er objekter og udsagnsord giver os metoderne. (En menu kan også være et objekt, med metoder til at oprette nyt punkt etc.)

Al tilgang til databasen skal ske gennem en statisk klasse som kan håndtere databasekald som oprette, vise, (redigere) og slette poster (CRUD).

Det er muligt at udbygge dette system, så det til sidst minder om Danske Bank, Nordea el.lign. Det skal I ikke, men I skal tilstræbe enkelthed, så I selv og andre kan finde ud af at bruge jeres ”moduler”. Hvis I bliver ”hurtigt færdige” er der lidt ekstra ønsker til systemet. (se sidst i dette dokument)

I koder i C# Consolprogrammering. Hvis nogle af jer har Web/Windows forms erfaring må I gerne kode i det miljø. Selve databasen (etablering, tabeller, relationer, felter, navne definitioner) må gerne oprettes uden for C# miljøet i f.eks. SQL server management Studio. Her kan I også indlægge testdata. Databasen er af typen ”databasenavn”.mdf.

Forslag til overordnet arbejdsplan:

* Plan over hvad I skal lave de enkelte dage (10 dage i alt inkl. fremlæggelse)
* design og etablering af databasen med testdata.
* Kodning og test af statisk klasse til databaseadministration.
* Analyse (hvilke objekter og metoder skal benyttes) UML diagrammer. Hvilke klasser, hvilken beskyttelse (private osv.) Brug evt. https://www.gliffy.com/uses/uml-software/ til at tegne dit diagram.
* Hvilke skærmbilleder får vi brug for (brugergrænseflade), oversigtsdiagram
* Hvilke datastrukturer skal benyttes (f.eks. lister, arrays osv.)
* kodning i C#.
* Test og rettelser (kan systemet nemt udbygges med de foreliggende strukturer)
* Fremlæggelse (demo af system og forklaring af valgte løsninger). Brug diagrammer, C# kode, foto, video etc.

**Når du er færdig med alt og keder dig . . . .**

Hvad med:

* Sortering af kunder, konti, transaktioner på de enkelte konti.
* Klasse med fejlmeddelelser
* En form for søgefunktion (find kunde, find transaktion osv. Udskriv konti med overtræk)
* Paging (der vises f.eks. 25 poster ad gangen på én side og man kan så bladre til næste side og tilbage igen)
* Oprettelse af nye kontotyper (navn, rentesats)
* Kontrol af at man ikke uden videre kan slette en bruger/kunde hvortil der er knyttet en konto
* Brugernavn og password (logning af user med timestamp)

Ovenstående skal ikke nødvendigvis kodes, men du kan tænke over hvordan det evt. kunne implementeres.

**På de følgende sider er lidt kode til database.**

**Følgende namespaces er benyttet:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Data.SqlClient;

**CREATE**

static void addEmployee(string the\_fname, string the\_lname, string the\_job)

{

var connection = new SqlConnection(@"Data Source = (LocalDB)\MSSQLLocalDB; AttachDbFilename = 'C:\Users\ar\documents\visual studio 2015\Projects\DataBaseEksempel\DataBaseEksempel\Database1.mdf'; Integrated Security = True");

SqlCommand cmd;

connection.Open();

try

{

cmd = connection.CreateCommand();

cmd.CommandText = "INSERT INTO Employees1(fname, lname, job) values('" + the\_fname + "', '" + the\_lname + "', '" + the\_job + "');";

cmd.ExecuteNonQuery();

Console.WriteLine("Tilføjede " + the\_fname + " " + the\_lname + " til ansatte database.");

Console.ReadKey();

}

catch (Exception)

{

throw;

}

finally

{

if (connection.State == System.Data.ConnectionState.Open)

{

connection.Close();

}

}

}

**READ**

static void showData()

{

SqlConnection connection = new SqlConnection(@"Data Source = (LocalDB)\MSSQLLocalDB; AttachDbFilename = 'C:\Users\ar\documents\visual studio 2015\Projects\DataBaseEksempel\DataBaseEksempel\Database1.mdf'; Integrated Security = True") ;

{

connection.Open();

using (SqlCommand command = new SqlCommand("SELECT \* FROM Employees1", connection))

{

using (SqlDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

for (int i = 0; i < reader.FieldCount; i++)

{

Console.WriteLine(reader.GetValue(i));

}

Console.WriteLine();

}

}

}

}

}

**DELETE**

static void deleteEmployee(int i)

{

var connection = new SqlConnection(@"Data Source = (LocalDB)\MSSQLLocalDB; AttachDbFilename = 'C:\Users\ar\documents\visual studio 2015\Projects\DataBaseEksempel\DataBaseEksempel\Database1.mdf'; Integrated Security = True");

SqlCommand cmd;

connection.Close();

cmd = new SqlCommand("DELETE FROM Employees1 WHere id=@i", connection);

cmd.Parameters.Add("@i", System.Data.SqlDbType.Int);

cmd.Parameters["@i"].Value = i;

connection.Open();

int slettet = cmd.ExecuteNonQuery();

if (slettet>0)

{

Console.WriteLine("Slettet - TRYK enter.");

Console.ReadKey();

}

else

{

Console.WriteLine("Ikke fundet - TRYK enter");

Console.ReadKey();

}

connection.Close();

}