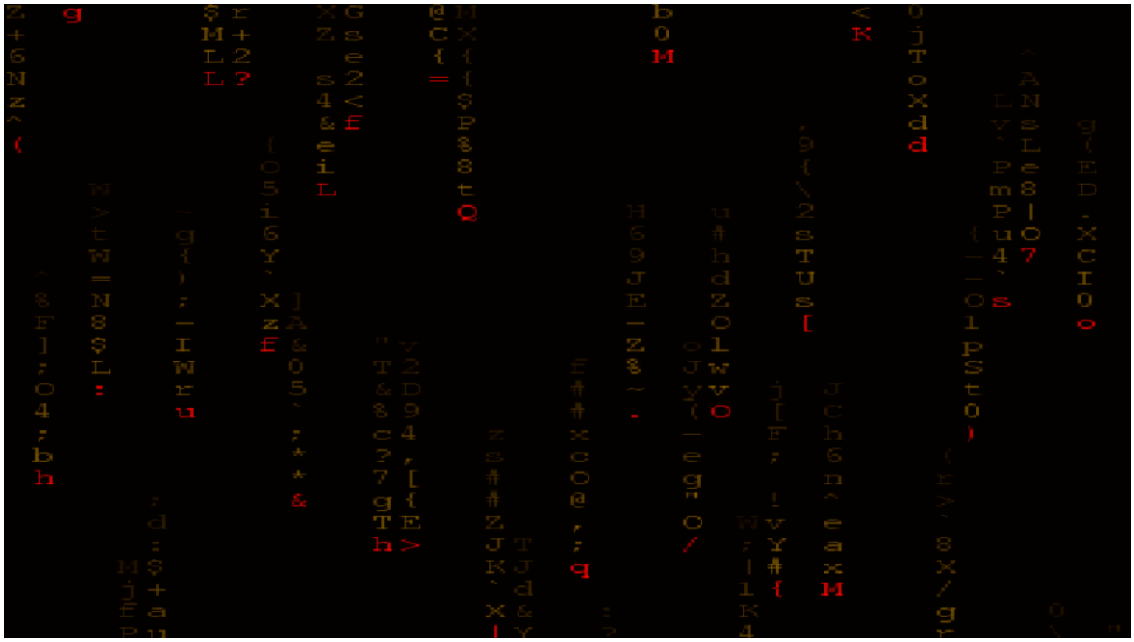
Opgavesamling til Softwarekonstruktion



Af Mads Mikkel Rasmussen

Senest ændret: 07.02.2023

Indholdsfortegnelse

DEL 0
GRUNDLÆGGENDE PROGRAMMERING

1 Øvelser i imperativ og procedural programmering

1.1 Kryptografi

Filer til disse opgaver findes her.

01 Dekryptér besked

I denne opgave skal du dekryptere en besked der er blevet opsnappet. Filen hedder EncryptedMessage001.txt. Det er ikke nødvendigt for dig at kende til afsenderen af beskeden (need-to-know-basis).

Der er følgende informationer om filens indhold:

- Består af hexadecimale tal, det må forventes at disse repræsenterer tekst.
- Det må forventes at beskeden kun indeholder bogstaver, ikke tal eller tegn.
- Der er indikationer på, at Ceaser's Cipher er anvendt som krypteringsalgoritme, med et primtal som shift.
- Typisk er indholdet i beskeder fra denne afsender et spørgsmål, et svar, en instruktion, et sted, et tidspunkt eller lignende.
- Der er indikationer på at det dekrypterede indhold også er krypto-kode. Det vil sige at det dekrypterede indhold kan være kodet også (måske endda i flere lag), men ikke nødvendigvis med en krypteringsalgoritme, men i stedet f.eks. en gåde, sætning/ord/tal i overført betydning og så videre.

Herudover vides intet om indholdet af beskeden.

Du skal løse følgende opgaver:

- Lav et program der kan dekryptere indholdet af beskeden. Du skal anvende best practice og være særligt opmærksom på at:
    - Anvende SoC (Separation of Concerns) således at én metode kun gør én ting, f.eks. at indlæse fra en fil og returnere en string med dette indhold, at vise en string i konsol vindue, at konvertere indholdet fra hex til ASCII osv.
    - Undersøge indlæsning fra en tekstfil, hvis du ikke allerede kan det, se f.eks. File klassen.
    - Undersøge character encodings, se f.eks. Encoding klassen.
    - Undersøge konvertering fra hex til en encoding, se f.eks. BitConverter klassen.
- Undersøg om du kan cracke det dekrypterede indhold.

DEL 1
OBJEKT ORIENTERET PROGRAMMERING

2 OOP: Klasser og Objekter

Se først denne video: madsmr – OOP Basics.

2.1 Gør som Mads gør i videoen

Lav en ny tom solution og navngiv den OOP. Lav dernæst et Console Application Project og kald det Basics. Lav Person klassen som i videoen og test den i Program klassen ved at lave et eller flere objekter af Person klassen, som i videoen.

2.2 Data og Funktionalitet

Som du kan se i videoen, er objekter til for at kunne opbevare data, samt at udføre funktionalitet på disse data. Du skal implementere følgende metoder i Person klassen og teste dem i Program klassen:

Metodens navn	**Returtype**	**Parametre**	**Funktionalitet**
GetInitials	string	ingen	De to første bogstaver i FirstName og LastName skal sættes sammen til en ny streng uden mellemrum. Eks.: KAHA for **Ka**rl **Ha**nsen.
GetAgeToday	int	ingen	Alderen i hele år beregnes ud fra dags dato.
IsOlderThan	bool	age (int)	Returnerer sandt, hvis personens alder er mere end værdien i parameteren *age*.
GetAgeAt	int	date (DateTime)	Returnerer personens alder i hele år, på tidspunktet for parameteren *date*.

2.3 Tilføjelse af en Property

Udvid Person klassen med:

- En property Height, der repræsenterer personens højde i centimeter.
- En property Weight, der repræsenterer personens vægt i kilogram.
- En metode GetBmi, der returnerer personens BMI tal. Der skal anvendes dansk voksent BMI for alle personer.

- En metode BmiDescription, der returnerer en streng, der angiver næringsmæssig status efter WHO's klassifikation.
- En metode der giver en narrativ beskrivelse af personen. Eks.: *Lille Rødhætte Jensen blev født den 31. februar 1808 og er i dag den 18. marts 2021 217 år gammel. Lille Rødhætte var 18 år i 1826, og har et BMI på 25…* osv.

2.4 En anden klasse

Lav en klasse Car, der består af:

- Make (mærket)
- Model (modellen)
- Fabrikationsdato
- Nypris
- Kørte kilometer

Husk at al kode skal være på engelsk. Du skal selv vælge datatype til hver property. Lav så følgende metoder:

- En metode der returnerer den anslåede værdi ved salg i dag. For hvert år taber bilen 15% i værdi hvert år de første fem år. Dernæst er værditabet 10% pr. år. Hvis bilen er en veteranbil (mere end 35 år gammel), stiger bilens værdi igen med 5% pr. år.
- Test din Car klasse ved at lave forskellige objekter af den.

EKSTRAOPGAVE 1: Salgsprisen hvis bilen skal sælges afhænger også af antallet af kørte kilometer. Hvis bilen gennemsnitligt har kørt mere end 25.000 km pr. år, reduceres salgsprisen yderligere med 7,5% pr. år.

EKSTRAOPGAVE 2: Tilføj nedenstående properties og lav en metode der narrativt beskriver bilen, som en slags salgsannonce:

- Farve
- Tophastighed
- Motorens ydelse i kilowatt
- Om det er en elbil, fossilbrændselsbil eller en hybridbil

2.5 ToString() metoden

3 OOP: Indkapsling og Tilstand

Se først denne video: madsmr – OOP encapsulation.

3.1 Indkapsl et fag

Et fag har disse egenskaber:

- Navn (f.eks. Oldgræsk filosofihistorie)
- Kode (f.eks. CJG-678)
- Underviser
- ECTS point (undersøg selv hvad det er)
- Startdato og slutdato
- Eksamensdato

Et fag har disse funktioner:

- Få varigheden
- Få dage til eksamen

Der gælder følgende regler for et fag:

- Fagkoden skal være tre store bogstaver, en bindestreg og tre tal, hvor det første tal ikke må være 0.
- Navnet skal være mindst fire karakterer langt
- Underviserens navn er et almindeligt navn
- ECTS point er i intervallet [0-10]
- Slutdato er altid efter startdato
- Eksamensdato er i intervallet [startdato; slutdato]

Du skal teste din klasse i tre scenarier:

1. Lav 3 objekter der har en gyldig tilstand.
2. Lav 3 objekter der ikke har en gyldig tilstand (overtræd én af reglerne). Undtagelser skal håndteres i en try-catch blok, og undtagelsens fejlmeddelelse skal udskrives til konsollen.
3. Mutér et objekt fra en gyldig tilstand til en ugyldig tilstand.

4 OOP: Constructors

En constructor har til formål at initialisere objektet til en gyldig starttilstand på baggrund af de værdier (argumenter) constructoren modtager. Hvis ikke det er muligt at initialisere objektet til en gyldig starttilstand, skal objektet slet ikke konstrueres og den kaldende kode skal i stedet få en undtagelse (exception).

4.1 Konstruer en lønseddel

Et lønseddelobjekt skal initialiseres med følgende argumenter:

- Lønperioden, dvs. den periode lønsedlen dækker over
- Skatteprocent
- Antal præsterede timer i perioden
- Timelønnen

Vent med at erklære constructoren til du har erklæret fields og properties og lavet indkapsling.

Erklær nu fields og properties der svarer til de fire nævnte ovenfor. Du skal anvende indkapsling med validering jævnfør reglerne og datatyperne nedenfor:

- Lønperiode: Her kan du enten vælge én parameter til start- hhv. slutdato, eller samle det i en tuppel. Husk minimum- og maksimumvalidering på hver dato. En lønperiode skal enten være 2 kalenderuger (14 dages lønning, starter altid på en mandag) eller 1 kalendermåned (månedslønning).
- Skatteprocent: double i intervallet [0.0; 1.0]
- Præsterede timer: decimal (så vi undgår afrundingsfejl) i intervallet [0; {12 t/dag x dage i lønperioden}]
- Timelønnen: decimal (så vi undgår afrundingsfejl) i intervallet [kr. 0,00; kr. 10.000,00]

Erklær og implementér nu constructoren.

Erklær nu tre metoder og implementér dem:

- Én metode der returnerer bruttolønnen (dvs. før skat trækkes)
- Én metode der returnerer nettolønnen (dvs. efter skat er trukket).
- Én metode der returnerer skattebeløbet.

Test nu følgende:

- At et objekt konstrueres til en gyldig starttilstand, når alle argumenter til constructoren indeholder valide værdier i henhold til reglerne.
- At objekt ikke kan konstrueres, hvis én eller flere af argumenterne til constructoren, ikke indeholder valide værdier.

4.2 Udvid med ID

Lav endnu en constructor (et constructor *overload*), hvor et id er et argument. Anvend constructor chaining med this keyword

5 OOP: Komposition og Aggregation

Komposition er et af de grundlæggende objekt orienterede principper. Kompositioner beskriver forhold mellem klasser, hvor den ene slags er en del af den anden slags. Vi kan beskrive det med sætninger hvor "har" eller "er en del af" indgår. Eksempler

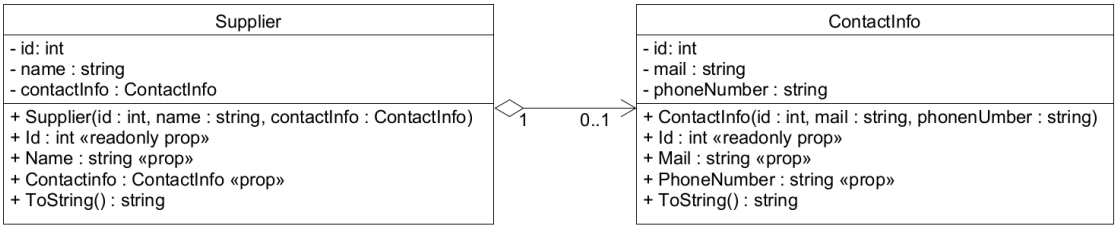
- "en vinge en del af en flyvemaskine" – og modsat "har en flyvemaskine vinger"
- "en kunde har kontaktoplysninger" – og modsat "en kontaktoplysning er en del af en kunde"

Ovenstående to eksempler illustrerer samtidig forskellen på komposition og aggregation. En flyvemaskine er ikke en flyvemaskine, når den ikke har vinger for så er den bare et skrog. Dette er komposition fordi begrebet flyvemaskinen ikke kan eksistere uden vinger. Derimod kan en kunde godt eksistere uden kontaktoplysninger. Dette kalder vi aggregation.

5.1 En leverandør har en adresse

Formålet med denne opgave er at du opnår viden og færdigheder i at implementere aggregation i én til én relationen.

Lad os betragte følgende UML klasse diagram:



Figur 1: UML diagram med én til én aggregation mellem Supplier og ContactInfo.

Note: stereotypen «prop» betyder en property. Stereotypen «readonly prop» betyder en property der ikke har en public setter. Denne teknik anvendes når man ikke vil tillade andre objekter at mutere det pågældende field.

Vi ser at de to klasser er relateret med aggregation ved at:

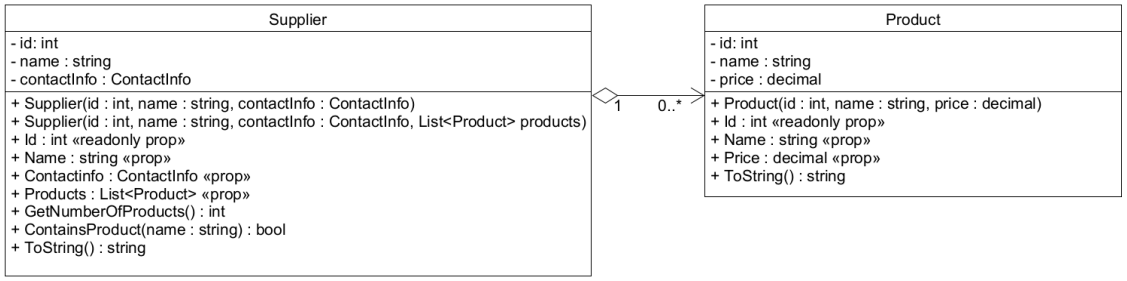
- Diamanten er hvid, ikke sort som ved komposition
- At det er muligt at mutere ContactInfo property på et Supplier objekt, fordi denne property ikke er readonly.

Vi ser også at multipliciteten er "én leverandør har nul til én kontaktoplysning". Det er det samme som at "én leverandør *kan have* en kontaktoplysning".

5.2 En leverandør har mange produkter

Formålet med denne opgave er at du oopnår viden og færdigheder i at implementere aggregation i én til mange relationen. Denne opgave forudsætter at du har løst ovenstående opgave.

Vi tilføjer nu en klasse Product, hvor relationen er én til mange:



Figur 2: UML klasse diagram med én til mange aggregation mellem Supplier og Product.

Vi ser at der er tale om aggregation (den hvide diamant) og at multipliciteten er én til mange. Vi kan derfor sikre os at vi er på rette spor ved at konstruere følgende sætninger der giver mening:

- Én leverandør kan have mange produkter.
- Ét produkt tilhører én leverandør

Bemærk den sidste sætning; Vi kan herudfra udlede, at det samme produkt ikke kan tilhøre mere end én leverandør.

Implementer ovenstående klassediagram i samme kode som forrige opgave. Bemærk den overloadede constructor i Supplier, samt de ekstra metoder.

5.3 Opsamlende opgave i grundlæggende OOP

Du skal implementere et forretningsdomæne i et klassebibliotek, og teste det. Du ser nederst i opgaven hvad du skal teste og hvad du skal implementere.

Til et online shoppingsystem er der følgende systembeskrivelse:

> *En kunde har en indkøbskurv hvor der kan være varer i. Det skal være muligt at se totalprisen, både med og uden moms, efterhånden som kunden tilføjer varer til kurven. En kunde er en person der på et tidspunkt har oprettet sig i systemet, med navn og e-mail.*

En rutineret Objekt Orienteret programmør, vil straks genkende at der er tre klasser: Customer, Basket og Produkt. Ud fra egen erfaring vil den rutinerede objekt orienterede programmør associere klasserne på følgende måde:

- En kunde har en indkøbskurv. En indkøbskurv er en del af en kunde.
- Der kan være nul til mange varer i indkøbskurven. En vare er en del af en indkøbskurv.

Din opgave er at konstruere forretningsdomænet, med følgende best-practice elementer:

- Der skal anvendes indkapsling med validering på alle klasser
- Der skal anvendes aggregation
- Intet objekt må instantieres (laves, konstrueres) i en ugyldig starttilstand. Test dette.
- Intet objekt må mutere til en ugyldig tilstand. Test dette.
- Alle klasser og deres members skal dokumenteres med summaries.

Hvis du har haft om Unit Testing skal du anvende dette, i stedet for et konsolprojekt.

Hint til aggregation: Du sikrer at et indkøbskurv-objekt ikke muterer til en ugyldig tilstand, når aggregationen til varer, er en property med en private setter.

Hint til summaries:

- *Klasser: beskriv hvad klassen repræsenterer i forretningsdomænet.*
- *Properties: start med sætningen "Gets og sets…"*
- *Fields/konstanter: beskriv hvad fieldet repræsenterer i forretningsdomænet*
- *Metoder: beskriv hvad hvilken funktion metoden opfylder i forretningsdomænet, husk også at beskrive eventuelle parametre og returværdien.*

5.4 Udvidelser til et system

Du skal implementere og teste følgende udvidelse af shoppingsystemet fra forrige opgave:

> *En kunde kan være enten en Basiskunde, Premiumkunde eller Guldkunde. Premium- og Guldkunder betaler hhv. kr. 150 og kr. 350 for medlemskab pr. måned. Basiskunder betaler 0 kr./md. Præmie- hhv.- Guldkunder får 2,45 % hhv. 6,25% rabat på alle køb.*

Du skal anvende enum til kundetypen. Start med at identificere alle navneord i ovenstående og overvej om det skal blive til klasser.

6 OOP: Arv og Polymorfi

Se først denne video: Inheritance.

Inden du går i gang med nedenstående opgaver skal du implementere de to klasser fra videoen. De næste fem opgaver anvender det du implementer.

Du skal teste hver opgave med Unit Testing hvis du har lært dette, ellers med et konsol projekt. Du skal teste for gyldig/ugyldig starttilstand og mutation. Der skal genereres en undtagelse hvis der er tale om ugyldig tilstand. Du skal også teste objekternes adfærd, dvs. kalde de metoder du har lavet og teste om de returnerer det forventede – og undtagelser ved ugyldig input.

Tip: Husk at tilpasse funktionaliteten i metoder/properties, efterhånden som du ændrer kodebasen. Hvis du har skrevet dine tests efter best-practice, vil disse hjælpe dig med at opdage mangler/uhensigstmæssigheder.

6.1 Nedarving – tilføjelse af klasser til hierarkiet

Du skal implementere sætningen "en kunde er en person". Du skal således lave en klasse Customer der har Person som base class. Udover egenskaberne fra Person klassen, har en Customer to egenskaber: datoen for hvornår personen blev kunde, samt et beløb for de samlede køb kunden har lavet siden personen blev kunde.

6.2 Nedarving – udvidelse af egenskaber i hierarkiet

Du skal tilføje en egenskab ved en sælger. En sælger har en beskrivelse der indikerer hvad sælgeren sælger.

Du skal tilføje to egenskaber ved en person: fødselsdato og benævnelse (title of courtesy).

6.3 Nedarving – udvidelse af klasser i hierarkiet

Du skal implementere sætningen "en sælger er en ansat, der er en person". En ansat har følgende egenskaber: stillingsbetegnelse, ansættelsesdato og årsløn. Tilføj også funktionalitet, der viser hvor mange procent af årslønnen, sælgeren har solgt for i år.

6.4 Nedarving – flere hierarkier

Et produkt består et navn, oprettelsesdato, pris og antal der skal sælges. En ydelse består af et navn, oprettelsesdato, pris, beskrivelse og løbetid fra starttidspunkt. Et produkt er en salgsenhed. En ydelse er en salgsenhed.

Implementer ovenstående hierarki.

6.5 Nedarving – anvendelse af aggregation i hierarkiet

Tilføj nu en klasse, der repræsenterer et salg fra en sælger til en køber. Et salg kan indeholde en eller flere salgsenheder. Et salg har en salgsdato og en totalpris.

Hint: Man kan godt lave en liste af salgsenheder, men tilføje objekter af klasser, der arver fra klassen der repræsenterer en salgsenhed.

6.6 Nedarving – abstrakte klasser og metoder

Se først denne video.

Du skal erklære din Person klasse abstrakt, samt tilpasse eventuelle tests der laver objekter af denne klasse. Overvej om der er andre klasser i dit forretningsdomæne, der også bør være abstrakte.

Tilføj nu en abstrakt metode i Person klassen. Metoden skal give en rating som en streng. For kunder skal der laves en stjerne rating, baseret på hvor meget kunden har købt for siden kunden blev oprettet. Der kan give 1 til 5 stjerner:

Stjernerating	**Gennemsnitligt køb pr. måned siden oprettelsen**
*	[0; 100]
**	]100; 250]
***	]250; 500]
****	]500; 1.000]
*****	]1.000;]

Tilsvarende for en sælger skal der laves en tekst beskrivelse, baseret på hvor meget sælger har solgt år-til-dato:

Tekstbeskrivelse	**År til dato salg**
X er en udmærket sælger	[0; 10.000]
X er en god sælger	]10.000; 250.000]
X er en dygtig sælger	]250.000; 1.000.000]
X er en fantastisk sælger	]1.000.000; 10.000.000]
X er en uovertruffen sælger	]10.000.000;]

Implementer ovenstående ved at override i hhv. sælger klassen og kunde klassen. Husk at teste.

6.7 Polymorfi – Best practice implementering af .NET interfaces og Object overrides

Formålet med denne opgave er, at du opnår rutine i at implementere centrale .NET interfaces og Object overrides, i henhold til best practice.

Du skal erklære en class Temperature, med to fields:

- degrees (decimal)

- scale (enum med grader celsius, fahrenheit og kelvin)

Overvej selv om der skal være mere end én constructor. Efter du har implementeret indkapsling, skal du implementere følgende interfaces i klassen:

- IComparable<T>
- IEquatable<T>
- IClonable

Viden:

- Best practice for Object.Equals() overrides: her, og her. Undersøg også den statiske metode Object.ReferenceEquals() på microsoft docs.
- Best practice for Object.ToString() her.
- Best practice for Object.GetHashCode() her. Forskel på .NET (Core) og .NET Framework her.
- Best practice for implementering af IEquatable<T> her
- Best practice for implementering af IClonable her. Undersøg selv hvad hhv. en shallow og en deep copy er. Undersøg også den statiske metode Object.MemberwiseClone() på microsoft docs.

Lav dernæst en void metode SetScale(scale.TemperaturSkala), der ændrer temperaturskalaen til det ønskede i argumentet – husk også at ændre værdien i degrees. Påvirker denne ændring dine andre metoder?

6.8 Statiske felter og metoder

Viden her.

Til forrige opgave skal du nu tilføje funktionalitet, sådan at man på ethvert Temperature objekt kan tilgå den højeste hhv. laveste temperaturværdi, der findes blandt alle objekter af klassen.

6.9 Operator overloading

Viden her. Implementer at temperatur objekter kan sammenlignes med hinanden for både lighed/ulighed og størrelse.

6.10 Polymorfi – IPayable

I denne opgave skal du opnå rutine med at implementere et interface. Interfacet implementeres af to klasser.

Viden: Se denne video og læs bloggen

Til opgaven er der et UML diagram, der åbnes med UMLet.

Forretningsdomænet

en sælger er en fastlønnet ansat, som er en ansat. En faktura har mange produkter. En faktura er et beløb og en ansats løn er et beløb.

Du skal her fokusere på GetPaymentAmount() og Earnings(). GetPaymentAmount() skal implementeres i både Employee og i Invoice. I Invoice skal GetPaymentAmount() returnere den totale pris for alle produkterne. I Employee skal GetPaymentAmount() returnere den løn der udbetales efter der er trukket 15% til pension. Lønnen beregnes for hver ansat type gennem Earnings() metoden. Bemærk på klassediagrammet at Earnings() overrides i hver ansat type, men at GetPaymentAmount() kun skal implementeres i Employee, men ikke i afledte klasser. Undlad at gå få meget i detaljen med indkapsling og validering her, og fokuser i stedet på at forstå polymorfi.

Tests

1. Test at Earnings() returnerer det forventede fra hver afledt instans af Employee.
2. Test at GetPaymentAmount() returnerer det forventede fra hver afledt instans af Employee. Hint: hver instans har datatypen Employee, hvor instansen initialiseres med et kald til den afledte klasses constructor.
3. Test udbetalinger ved at lave 5 forskellige instanser af afledte klasser af Employee, samt tre instanser af Invoice. Lav hertil en hjælpemetode i testklassen:
   GetTotalPaymentAmpunt(List<IPayable> iPayables) : decimal.

Forslag til opgaveløsning

1. Erklær alle typer
2. Erklær alles typers members.
3. Erklær nedarvinger fra andre klasser og implementeringer af interfaces.
4. Sørg for at properties indkapsler deres fields.
5. Lav constructor chaining.
6. Lav logikken i metoderne i Employee og Invoice.
7. Lav logikken i metoder i klasser der nedarver fra Employee, sammen med tests.

Tegn for færdiggørelse

De tre tests er korrekte og grønne.

Udvidelser

Du skal kun lave disse udvidelser, såfremt din lærer godkender det.

1. Implementer overrides fra Object efter best practice på alle klasser.
2. Implementer IEquatable<T> efter best practice på alle klasser.

DEL 2
APPLIKATIONER

7 Grafiske brugerflader

Du lærer hvordan man laver en grafisk brugerflade med funktionalitet, gennem en række mindre opgaver. Når du har løst alle opgaver korrekt, har du fået viden om og færdigheder i grundlæggende WPF controls med XAML og UI Designeren i Visual Studio.

Du kan med fordel lave ny tom solution i Visual Studio, og lave undermapper der matcher titlen på hvert afsnit i dette kapitel: Buttons, TextBox, CanvasImages osv. Til løsning af hver opgave kan du lave et nyt WPF Application project, og navngive det efter afsnit og nummer, eks. Buttons01, TextBox03 osv.

Husk at give et sigende navn til dine control variabler i XAML koden, gennem `x:name` attributten.

7.1 Buttons og TextBlocks

Disse opgaver omhandler knapper og tekst i WPF.

01 Klik her knappen

Brugerfladen skal indeholde en knap med teksten "Klik her". Programmet skal vise en MessageBox med beskeden "Du trykkede på knappen", når der trykkes på knappen.

02 Knappen skifter farve

Brugerfladen skal indeholde en knap. Når der trykkes på knappen, skal den skifte farve til grøn.

03 Tæl op

Brugerfladen skal indeholde en knap og et tekstfelt (en TextBlock) med tallet 0 til at starte med. Når der trykkes på knappen skal tallet tælle én op. Variablen til at indeholde værdien, skal være et field i code-behind klassen.

04 Tæl op og ned og sammen

Brugerfladen skal indeholde to knapper, én med en pil op (eller lignende) og én med pil ned (eller lignende). Knappen med pil op skal være vertikalt over den anden knap. Brugerfladen skal også indeholde to tekstfelter. Det ene tekstfelt skal indikere hvor mange gange der tilsammen er trykket på knapperne. Det andet tekstfelt skal indikere den nuværende værdi for klik på knapperne: trykkes på pil op knappen, tælles denne værdi én op – trykkes på pil ned knappen, tælles værdien én ned.

05 Fang mig spillet

Du skal lave et "Fang mig" spil. Brugerfladen skal være i fuldskærm (søgeterm: *WPF how to start window in full screen*). Brugerfladen skal indeholde én knap. Når brugeren klikker på knappen, skal knappen flytte sig et andet sted hen. Benyt Random til dette. En knaps position ændres ved Margin property. Lav et point system til brugeren, så brugeren bliver afhængig af spillet.

Bonus: Når brugeren efter noget tid har fanget knappen gentagne gange, skal tempoet sættes op sådan at knappen nu flytter sig et andet sted hen efter xx millisekunder (Dette kræver asynkron programmering). Lav et bedre point system til brugeren, så brugeren bliver endnu mere afhængig af spillet. Denne opgave er ret svær.

7.2 TextBox og TextChanged eventhandler

Disse opgaver omhandler tekstinput i WPF.

01 Indtast navn

Lav et program hvor brugeren skal indtaste sit navn i en TextBox og få vist følgende i en TextBlock, når brugeren har trykket på en knap: "Hej {indtastede navn} og velkommen til programmet." Man får fat i det indtastede ved at tilgå Text property på TextBox variablen i code-behind.

02 Indtast fornavn og efternavn

Lav et program hvor brugeren skal indtaste fornavn i én TextBox og efternavn i en anden TextBox. Brugeren skal få vist det fulde navn i en TextBlock, når brugeren klikker på en knap. Sørg for at `x:Name`'s er hhv. textBoxFirstname og textBoxLastname.

03 Replikator

Lav et program hvor dét brugeren skriver, altid er gengivet nøjagtigt i en TextBlock. Der skal ikke være nogen knap, men derimod skal eventhandleren TextChanged på TextBox'en laves. Dette gøres ved:

1. at markere TextBox'en i enten designeren eller XAML, og finde Properties vinduet.
2. Øverst i højre hjørne i Properties vinduet er der et lyn, tryk på det.
3. Nu fremkommer listen over alle events man kan lave eventHandlere til. Find TextChanged og skriv ud for det: TextBox_TextChanged og tryk ENTER.
4. Så er event handleren til TextChanged event lavet i code-behind filen.

Så når brugeren skriver/sletter et tegn i TextBox'en, afspejles det i TextBlock'en.

7.3 Canvas og billeder

01 Baggrundsbillede

Du skal lave et program hvor baggrunden af vinduet er et billede. Det bruger man Canvas controllen til. Så i stedet for Grid skal du nu have et Canvas. I Properties vinduet, tilføjer du et eller andet billede. Kør programmet.

02 Splash Screen

Nu laver vi en splash screen med login. Baggrunden skal igen være et eller andet billede. Nu skal der også være brugernavn, password og Log Ind knap. Brug en PasswordBox til password. For at fjerne titellinjen: WindowStyle="None", for at fjerne kanten: ResizeMode="NoResize". Brugeren skal kun kunne logge ind hvis brugernavn-password kombination er enten Ole, MinHundErSød eller Brian, Lastbil2006$Rød.

03 Billedgalleri

Lav et program hvor brugeren kan se ét billede ad gangen. Hvis brugeren trykker på Frem skifter billedet til næste billede i en mappe på din PC. Benyt eventuelt din Billeder mappe som programmets "bibliotek". Hvis brugeren trykker på tilbage knappen vises det foregående billede.

Bonus: Lad billederne køre i ring, således at hvis brugeren er på sidste billede, og trykker frem, så vises det første billede.

Bonus: Lad brugeren indtaste hvor lang tid man gider se på et billede, og automatisér visningen, dog stadig med mulighed for manuelt frem og tilbage skift.

7.4 Checkbox

01 Fun with CheckBox

Lav et program hvor brugeren kan afkrydse en CheckBox. CheckBox'en har mulighed for at have en tekst, og her skal der stå "Vis tekst" (Content property, ikke Text property). Hvis brugeren har afkrydset, skal teksten "Check!" vises i en TextBlock, ellers ikke. En CheckBox har to events du skal bruge, Checked og Unchecked. Disse events skal have EventHandlere.

02 Checked?

Lav et program hvor brugeren kan afkrydse en CheckBox, men nu med tre muligheder (IsThreeState). Når der er afkrydset vises teksten "Check!" i en TextBlock. Når der ikke er afkrydset vises "Uncheck!", ellers vises "Maybe?".

7.5 RadioButton

01 Rød, Gul, Grøn

Lav tre RadioButtons med disse valg: Rød, Gul og Grøn. Baggrunden på vinduet skal skifte farve efter valget. En RadioButton benytter Checked/Unchecked på samme måde som en CheckBox.

02 Visibility og IsEnabled

Lav tre RadioButton i en gruppe (sådan at de har samme GroupName i XAML) med valg: Rød, Gul og Grøn. Baggrunden på vinduet skal skifte farve efter valget. Lav nu tre RadioButtons mere til venstre for de tre. Disse RadioButtons skal have tre valg: Skjul, Inaktiv og Aktiv. Når Skjul er markeret skal de tre Rød, Gul Grøn Radio-Buttons skjules med deres Visibility property. Inaktiv/Aktiv indikerer om brugeren kan interagere med Rød, Gul, Grøn RadioButtons. Anvend her Enabled property.

7.6 ItemsControls

I denne opgave skal du træne ListBox, ComboBox og DataGrid. Der er alle ItemsControls. Du skal lave ét WPF Application project til alle delopgaverne.

Lav en ListBox til produkter hvor produktnavnet skal vises. Du skal lave mindst 10 produkter, fordelt på tre kategorier. Kategorierne er

- Drikkevarer
- Accessories
- Byggematerialer

Et produkt klasse består af id, navn, dato for oprettelse, beskrivelse, kategori, om produktet er udgået, enhedspris og antal på lager. Lav denne klasse med *autoimplemented properties* – det vil sige bare `{get; set;}` i stedet for indkapsling.

01 Vis alle produkter

Når programmet startes, skal alle produkter vises i ListBoxen, sorteret på navn.

Hints:

- Lav et field i MainWindow klassen du kalder dataSource. Dette field er en liste med produkterne. Dette gøres i stedet for at lave en database, som man ellers ville gøre i henhold til best practice.
- Anvend LINQ fra System.Linq namespace til at sorte listen, inden ListBox'en fyldes med produkt objekterne. Hvis du er i tvivl, så prøv at google "How to sort C# list with LINQ".
- Anvend `private void` metoder til de processer, du kan se vil blive gentaget. F.eks. en metode til at rydde og fylde ListBox'en, og en metode til at sortere efter navn. Husk at listen med produkter ikke selv må sorteres.

02 Tilføj produkt

Lav controls til at tilføje et nyt produkt. En Button med teksten "Tilføj" skal således tilføje det nye produkt til ListBox'en.

Hint: Anvend RadioButtons til valg af produktkategori.

03 ComboBox

Denne opgave forudsætter du har løst den forrige om ListBox, som du skal anvende her.

Lav en ComboBox til produktkategorien (som erstatter de RadioButtons du lavede tidligere), som brugeren skal bruge når der oprettes et nyt produkt.

04 DataGrid

Lav et DataGrid, der viser produkterne og deres egenskaber. Husk at gøre kolonnenavnene til dansk.

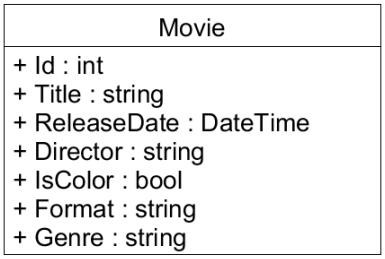
05 Rediger produkt

Tilføj funktionalitet til at redigere et i ListBox'en valgt produkt.

7.7 Master-Detail layout med Grid

Se først denne video, hvor master-detail layout princippet demonstreres.

I denne opgave skal du lave en brugerflade til håndtering af indtastning af film, samt visning af alle film med mulighed for søgning.



Figur 3: UML klassediagram der repræsenterer en film

Ovenstående diagram fortæller hvad Movie klassen skal indeholde, inklusive angivelse af datatyper. Lav blot din Movie class med public (+) auto-implemented properties, eksempel: `public int Id {get; set;}`.

Class MovieRepository

Denne klasse skal repræsentere din database, selvom der ikke er nogen:

```
class MovieRepository
  {
    List<Movie> movies = new List<Movie>
    {
      new Movie
      {
        Id = 1,
        Title = "Star Trek: Beyond",
        ReleaseDate = new DateTime(2016,07,22),
        Director = "Justin Lin",
        IsColor = true,
        Format = "2.35:1",
        Genre ="Sci-Fi"
      },

      new Movie
      {
        Id = 1,
        Title = "Star Wars: The Last Jedi",
        ReleaseDate = new DateTime(2017,12,15),
        Director = "Ryan Johnson",
        IsColor = true,
        Format = "2.39:1",
        Genre ="Epic Space Opera"
      }
    };

    public List<Movie> GetAll()
```

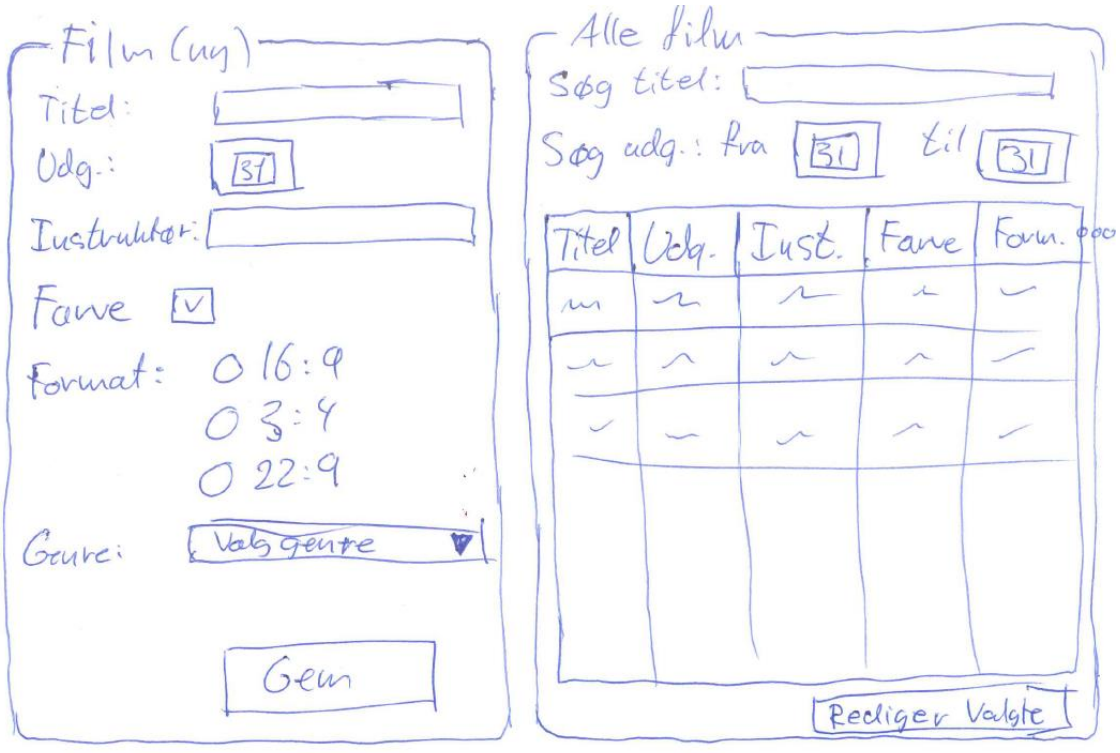
```
{
  return movies;
}

public void AddNew(Movie movie)
{
  movies.Add(movie);
}

public void Update(Movie movieToUpdate)
{
  foreach(Movie movie in movies)
  {
    if(movie.Id == movieToUpdate.Id)
    {
      movies.Remove(movie);
      movies.Add(movieToUpdate);
      break;
    }
  }
}
```

Klassen består af fire members: movie (field), GetAll (method), Add (method) og Update (method). Kopier klassen ind i din MovieRepository fil. Der er allerede to film i "databasen".

Brugerflade design



Figur 4: UI Mock-up af en filmhåndteringsapplikation.

Dette er et mock up af brugerfladen. Bemærk at master i dette tilfælde er til højre og details er til venstre. Du skal implementere designet. Husk i øvrigt at give x:Name's til de af dine controls.

Kasserne med 31 er DatePicker control. Du kan læse om hver control ved at google WPF Tutorial og udforske de links der fremkommer. Derudover bør du også finde disse controls på docs.microsoft.com, og sætte dig overordnet ind i dokumentationen for hver control. I øvrigt: der hvor der står Film (ny) og Alle Film, er en GroupControl.

Som du kan se ud fra designet, skal knappen Rediger Valgte indlæse den valgte film til controls til venstre, hvor brugerne så kan redigere i filmen, og gemme den.

7.8 .NET Maui

.NET Maui er kort og godt én UI kodebase til den samme applikation, der gør at applikationen kan køre på både Windows, android, iOS og MacCatalyst. Det er muligt at installere emulatorer til både android og iOS, samt at tilslutte devices med disse operativsystemer og afvikle app'en der. Det kan være besværligt at installere en emulator, og med mindre det fremgår af opgaven, kan du nøjes med at afvikle til Windows.

1 Introduktion

Se video 1-6 i denne dotnet serie med James Montemagno, og gør som James gør. Har du behov for ekstra viden har James også lavet en meget lang video hvor mange forskellige features af .NET Maui demonstreres.

8 Databaser

8.1 Northwind Queries

I denne opgave skal du lave en database og anvende scripts til at udvælge data og indsætte data.

Praktiske oplysninger

Du skal have Microsoft SQL Server Management Studio installeret på din PC. Du skal også på forhånd have kørt northwindScript.sql.

Se også w3schools og afprøv SELECT, INSERT, UPDATE, DELETE og JOIN.

SELECT

1. Find alle produkter der ikke længere føres.
2. Find alle leverandører fra Québec.
3. Find alle leverandører fra Tyskland og Frankrig.
4. Find alle leverandører der ikke har en hjemmeside.
5. Find alle leverandører fra europæiske lande, der har en hjemmeside.
6. Find alle ansatte hvis fornavn begynder med *M*.
7. Find alle ansatte hvis efternavn slutter på *an*.
8. Find alle kvindelige ansatte der ikke er læger (benyt en OR).
9. Find alle medarbejdere der er Sales Representative og kommer fra UK.
10. Find ud af hvor mange produkter der er.
11. Find gennemsnitsprisen for alle produkter.
12. Find antal produkter med en enhedspris over 20,00. Sorter efter dyreste.
13. Find de produkter der ikke er flere af, sorter alfabetisk.
14. Find alle de produkter der ikke er flere af, og som ikke er bestilt, men heller ikke udgået, sorter efter produktnavn, omvendt alfabetisk rækkefølge.
15. Find alle kunder der er enten er franske ejere eller britiske sælgere, sorter efter land, dernæst navn.
16. Find alle nord-, mellem-, og sydamerikanske kunder der ikke har en fax, sorter alfabetisk.

UPDATE

1. Opdater alle leverandørers fax til *no fax number*, hvis ikke de har et fax nummer. Gør det samme for alle kunder (Hint: adskil disse to opdateringer med ; og kør dem begge i samme transaktion).

2. Opdater genbestillingsmængden for alle ikke-udgåede produkter, hvis nuværende genbestillingsmængde er 0 og nuværende beholdning er mindre end 20, til 10.

3. Opdater alle spanske kunder med den korrekte region. Se spanske regioner på wikipedia og/eller google maps.

4. Simons bistro har ændret navn til Simons Vaffelhus og flyttet til Strandvejen 65, 7100 Vejle. Foretag opdateringen.

5. White Clover Markets er flyttet til 247 New Avenue, Chicago og har skiftet nummer til 555-20159. Foretag opdateringen.

6. Medarbejderen Janet er flyttet ind hos medarbejderen Andrew. Foretag opdateringen.

INSERT

1. Northwind har ansat Kim Larsen CPR: 190583-7856, Violvej 45, Sønderborg, tlf 75835264, pr. 1. januar i år. IT support gav ham tlf. ext. 0745. Der er ikke noget billede af Kim.

2. Northwind har hyret alle lærerne på din AspIT afdeling pr. næste måneds 1. Disse medarbejdere skal INSERT'es i det samme statement og ikke i hvert sit statement – altså ét INSERT med mange VALUES-rækker. Alle medarbejderne får AspITs hovednummer som tlf men hver sin ext. Du skal vælge AspIT afdelingernes adresser som adresser til medarbejderne – som skal være forskellige.

3. Der er et nyt produkt på banen: SuperDuperBeer. Tilføj dette produkt med passende valg af værdier til kolonnerne.

4. Der er en ny Leverandør: Campus Vejle. Find info om dem og tilføj leverandøren.

5. Campus Vejle er nu leverandør af SuperDuperBeer.

6. Der er en ny Shipper: Mærsk. Tilføj den.

JOINS

1. Find beskrivelserne for alle territorier og deres tilhørende regioner – hver kombination skal kun vises én gang (DISTINCT).

2. Find alle produkter der er drikkevarer og som ikke er udgået. Vis Produktnavn, pris og lagerbeholdning, sorter efter lagerbeholdning, dernæst produktnavn.

3. Find navnene på alle kunder der har placeret en ordre i februar 1997. Sorter efter ordredato, dernæst kundenavn.

4. Find navnene på alle kunderne og udskibningsdato for ordrer, der blev placeret i hele 2. kvartal 1997. Sorter efter nyeste ordredato, dernæst kundenavn.

5. Find de 9 leverandører der laver drikkevarer og vis kun disse 9 firmanavne, sorter efter navn.

6. Find fornavn og efternavn på alle medarbejdere, samt hvilken medarbejder der er deres chef (chefens fornavn og efternavn skal vises), sorter efter chefens efternavn. Denne opgave er svær, så brug ikke mere end 1 time på den.

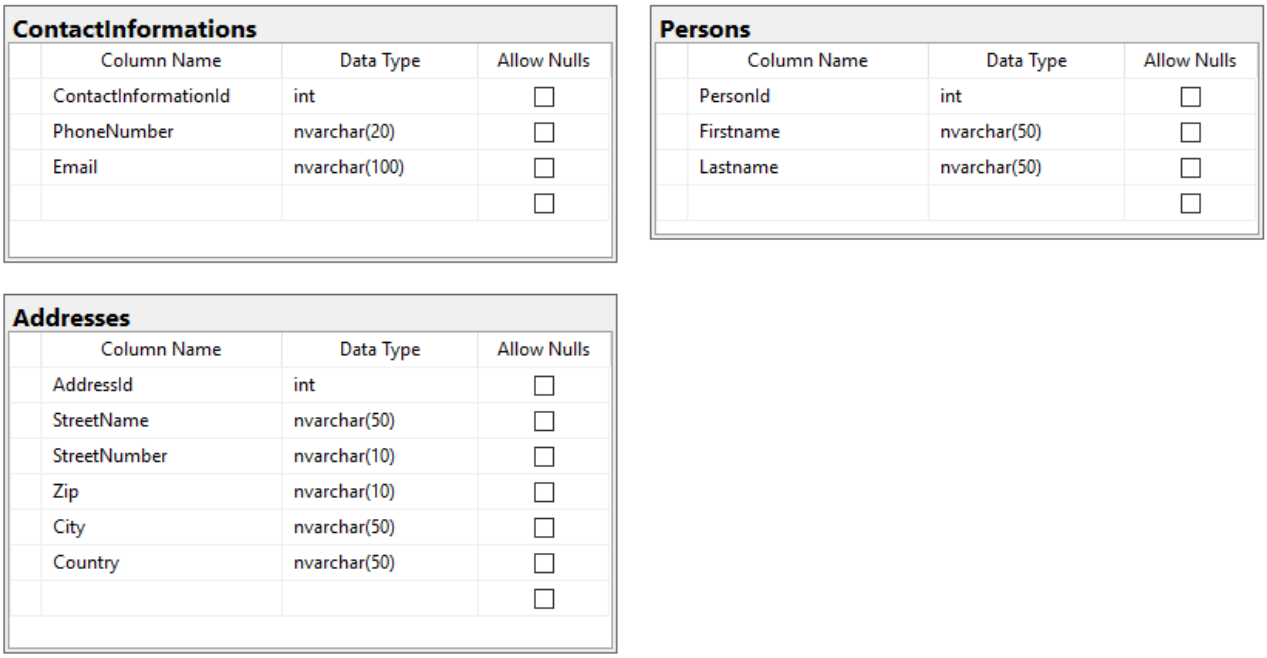
8.2 Personer og deres kontaktinformationer og adresser

I denne opgave skal du lave en database og anvende scripts til at tilføje, ændre og udvælge data.

Konstruer en database

Det første du skal er at lave en ny database. Åben SQL Server Management Studio, lav forbindelse til din lokale server (localdb)\mssqllocaldb. Højreklik på mappen Databasen i vinduet Object Explorer. Højreklik på Database og tilføj en ny database ved navn UniDB, og klik OK. Vi kan forestille os, at databasen skal anvendes af et universitet, til at holde styr på studerende.

UniDB databasen er nu at finde i mappen Databases i vinduet Object Explorer. Udvid på + til venstre for databasen og højreklik på Database Diagrams og vælg New Database Diagram. Der fremkommer muligvis en message box med at visse objekter bør installeres, her klikkes Yes. Der fremkommer en dialog Add Table, klik blot Cancel. Højreklik på designeren for at tilføje nye tabeller. Du skal lave nedenstående ER diagram:



ContactInformations

Column Name	Data Type	Allow Nulls
ContactInformationId	int	☐
PhoneNumber	nvarchar(20)	☐
Email	nvarchar(100)	☐
		☐

Persons

Column Name	Data Type	Allow Nulls
PersonId	int	☐
Firstname	nvarchar(50)	☐
Lastname	nvarchar(50)	☐
		☐

Addresses

Column Name	Data Type	Allow Nulls
AddressId	int	☐
StreetName	nvarchar(50)	☐
StreetNumber	nvarchar(10)	☐
Zip	nvarchar(10)	☐
City	nvarchar(50)	☐
Country	nvarchar(50)	☐
		☐

Figur 5: ER diagram af UniDB databasen uden relationer mellem tabellerne.

Bemærk, at der endnu ikke er lavet nøgler, dette er blot et udgangspunkt. Husk at gemme.

Viden om SQL

På w3schools.com får du den nødvendige viden om SQL sproget. Du skal følge:

1. SQL Tutorial (alle punkter).
2. SQL Database (alle punkter).
3. Bogmærk SQL Quick Ref (under SQL References).
4. Bogmærk T-SQL Data Types.
5. Gennemfør SQL Quiz.

Del 1: Relater dine tabeller

Du skal oprette primærnøgler og fremmednøgler i dit diagram. Se denne video først.

Sørg for at dine fremmednøglekolonner tillader NULL værdier, ellers bliver det besværligt at tilføje data.

Del 2: Tilføj data

Der skal oprettes 22 personer, der bor på 13 forskellige adresser der er i fem forskellige lande. Lav korrekte og troværdige data. Du skal benytte SQL INSERT statements i queries du eksekverer på databasen i Management Studio. Man højreklikker blot på databasen og vælger New Query. Derefter skriver man statement(s) og trykker på F5 for at eksekvere. NB: der er muligt at have mere end ét INSERT statement i et query script (faktisk op til 300). Husk du selv skal sørge for at relatere de 22 personer med deres egen kontaktinformation og adresse. Der gælder følgende: én person

har én kontaktinformation, én person bor på én adresse og én adresse kan have en til mange personer boende.

Del 3: Udvælg data

Du skal lave scripts til hver af de følgende punkter. Hvert script skal gemmes. Et script er en .sql fil (tekstfil). Et query er en .sql fil der eksekveres fra Mangament Studio.

1. Vælg alt fra hver tabel.
2. Vælg alle personers fornavn og efternavn fra Persons.
3. Valg alle e-mailadresser fra ContactInformation
4. Opdatere en adresse med nyt land
5. Finde alle personer født efter 2001
6. Finde alle adresser i DK
7. Finde alle personer med fornavn der starter med [A-J]
8. Finde alle personer der er født i et lige år
9. Finde alle personer hvis e-mail har et .com domæne
10. Finde alle personers fornavne og efternavne der har mobilnumre fra DK (Join)
11. Finde den/de adresse(r) hvor der bor flest
12. Find alle personers fornavne og efternavne, og deres e-mail og telefonnummer.
13. 12 nu med gadenavn, -nummer, postnummer og by.

8.3 Applikation med tilgang til database

Denne opgave baserer sig på den forrige. Du skal forvente at denne opgave strækker sig over mere end én dag.

Du skal lave et program der kan:

1. Vise alle personer, med deres kontaktinformationer og adresser.
2. Tilføje nye personer inklusive deres kontaktinformationer og adresser. Vær opmærksom på, at personen der tilføjes, kan bo på en adresse der allerede findes.
3. Opdatere oplysninger om en person, personens kontaktoplysninger, samt adresse.

Det er et krav at du:

- Laver en separat klasse til håndtering af tilgang til databasen
- Laver en grafisk brugerflade til applikationen med master-detail layout
- Laver klasser der repræsenterer forretningsdomænet, ved anvendelse af indkapsling og aggregation

8.4 Entity Framework – CRUD operationer

Denne opgave skal laves med .NET 6+, ikke med .NET Framework. Hvis ikke du har installeret Northwind databasen, så skal du gøre det – du finder scriptet i den første opgave til dette kapitel.

I denne opgave skal du lave CRUD operationer på Northwind databasen med Entity Framework (EF). EF er en ORM, Object Relational Mapper teknologi, der tillader at man anvender C# i stedet for SQL. Du vil se hvordan man opretter klasserne til Entities med en feature i EF, der hedder Scaffold-DbContext, og hvordan man laver CRUD operationer. Dernæst skal du opnå viden ved at gennemlæse en række tutorials. Husk at Entity Framework er et kompliceret og omfattende emne, og at du kun skal kunne det grundlæggende, som denne opgave giver dig mulighed for at lære.

1 Solution

Opret først en ny solution der hedder NorthwindEfExercise, tilføj et klassebibliotek der hedder Entities, og tilføj et konsol projekt der hedder ConsoleTests, og lad konsolprojektet have en reference til Entities.

- Lav konsol projektet til startup projekt, hvis ikke det allerede er det.
- Nullable disable i begge projekter (dobbeltklik på projektet i solution explorer)
- Build solution med F6.

2 Installation af EF NuGets

Der skal installeres en to NuGet pakker:

- Microsoft.EntityFrameworkCore.SqlServer
- Microsoft.EntityFrameworkCore.Tools

Følg denne tutorial, men installer pakkerne i begge projekter. Dette kan du gøre ved at højreklikke på solution > Manage NuGet Packages for Solution, i stedet for blot at højreklikke på konsol projektet, og så vælge begge projekter. Build på F6 efter hver pakke er installeret, og start med Microsoft.EntityFrameworkCore.SqlServer.

3 Scaffold klasserne til Entities fra en eksisterende database

Scaffold betyder at lave et stillads som udgangspunkt til videre konstruktion. Med udgangspunkt i denne tutorial, skal vi lave klasserne i Entities projektet. Der vises i førnævnte tutorial, hvordan man scaffolder til et konsol projekt, men vi skal scaffolde til et klassebibliotek. Derfor skal du nøjes med at læse førnævnte tutorial igennem, men ikke udføre den. Når du har gjort det, skal du følge nedenstående vejledning:

1. Åben Package Manager Console: Tools > NuGet Package Manager > Package manager Console. Dette åbner et vindue i VS. Sproget er PowerShell.
2. Ændr Default Project til ConsoleTest.
3. Find din connection string til Northwind databasen: View > Sql Server Object Explorer. Dette åbner et vindue til din lokale SQL Server. Navigér hen til din Northwind database og klik på udvid trekanten med musen. Klik så på F4. Her finder du din connection string, kopier den til udklipsholderen.

4. Kør nu følgende kommando i Package Manager Console:
```
Scaffold-DbContext "Data Source=(localdb)\MSSQLLocalDB;Initial
Catalog=Northwind;Integrated Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationInte
nt=ReadWrite;MultiSubnetFailover=False"
Microsoft.EntityFrameworkCore.SqlServer -Context NorthwindContext -
Project Entities -StartupProject ConsoleTest
```

Hvis det virker får du en gul advarsel du kan ignorere. Ellers er der røde fejl. Her er en beskrivelse af kommandoen:

- Første argument er connection string.
- Andet argument er den såkaldte provider der fortæller hvilken type server der skal laves transaktioner ved.
- Tredje argument er navnet på den klasse der arver fra DbContext.
- Fjerde argument er navnet på det projekt klasserne skal scaffoldes i.
- Femte argument er navnet på det (.exe) projekt, der skal udføre kommandoen.

Du kan altid slette klasserne og prøve igen. Bemærk at der er lavet en ekstra klasse, NorthwindContext. Den indeholder en masse DbSet<T> properties, der svarer til hver tabel i databasen. Det er disse properties man bruger, når man skal lave CRUD operationer. Dette demonstreres i det følgende, som du også selv skal lave.

4 Test at det virker

I program.cs kan du teste om alt virker ved:

```
using Entities;

NorthwindContext context = new();
var employees = context.Employees.ToList();
foreach (Employee employee in employees)
{
    Console.WriteLine($"{employee.FirstName} {employee.LastName}");
}
```

Selve kaldet til databasen sker i ToList() metoden.

5 Indsæt en række i en tabel

```
Supplier s = new()
{
    CompanyName = "MySoftware",
    ContactTitle = "Head of heads",
    ContactName = "Kevin Magnussen",
    Address = "Forløkken 47",
    PostalCode = "1024",
    City = "Bitstrup",
    Country = "Computerland"
};

context.Suppliers.Add(s);
context.SaveChanges();

foreach (Supplier supplier in context.Suppliers.ToList())
{
    Console.WriteLine($"{supplier.CompanyName}");
```

```
}
```

6 Opdater en række i en tabel

```
Supplier supplierToUpdate = context.Suppliers.SingleOrDefault(s =>
s.CompanyName == "MySoftware");
supplierToUpdate.CompanyName = "MEGA SOFT";
context.SaveChanges();

foreach (Supplier supplier in context.Suppliers.ToList())
{
    Console.WriteLine($"{supplier.CompanyName}");
}
```

7 Slet en række i en tabel

```
Supplier supplierToUpdate = context.Suppliers.SingleOrDefault(s =>
s.CompanyName == "MEGA SOFT");
context.Suppliers.Remove(supplierToUpdate);
context.SaveChanges();

foreach (Supplier supplier in context.Suppliers.ToList())
{
    Console.WriteLine($"{supplier.CompanyName}");
}
```

8 Gennemlæs tutorial

Du skal nu opnå større viden om hvordan EF virker. Derfor skal du gennemlæse disse tutorials, uden selv at udføre dem:

- EF Basics: læs alle underemner. Bemærk at vi her i det ovenstående har anvendt en anden tilgang (Code from Existing Database), end den der står i underemnet Basic Workflow.

Dernæst skal du gennemlæse følgende underemner til EF Core:

- DbContext
- First Application
- Querying
- Saving Data

9 Lav den første opgave i dette kapitel med EF

Ovenstående eksempler viser hvordan man anvender CRUD med EF. Du skal nu lave den første opgave i dette kapitel (Northwind Queries) med EF i stedet for SQL. Til JOIN opgaverne skal du anvende Include().

8.5 Repository og Unit of Work design pattern

I denne opgave lærer du om Separation of Concerns (SoC) i data access kode, der anvender EF. Du lærer om et design pattern der hedder Repository med Unit Of Work. Det er en best-practice tilgang til at lave SoC i data access kode med EF.

Du skal gennemlæse denne tutorial, men først skal du læse dette: Den tager udgangspunkt i en webapplikation med såkaldte controllers. Du behøver ikke at kende til controllers, og kan opfatte en controller som værende en klasse i et konsol projekt eller WPF applikation, der har ansvaret for at kalde data access koden. Først gennemgås det hvad et repository er og hvordan det anvendes. Ca. midtvejs bliver det interessant i afsnittet Implement a Generic Repository and a Unit of Work Class, samt de efterfølgende afsnit. Det er her du for alvor skal sætte dig ind i hvad der står. Bemærk også at der anvendes Generics, en del af det objekt orienterede princip polymorfi, og det er ikke anderledes end det du kender fra List<T> klassen.

1 Besvar følgende skriftligt

- Hvilket ansvar har et repository?
- Hvilket ansvar har en unit of work?
- Hvilket formål tjener det, at anvende repository og unit of work i data access?

2 Implementer det

Du skal lave en repository/unit of work løsning som har tilgang til Northwind databasen. Derfor skal der laves et klassebibliotek der hedder DataAccess, der har en reference til Entities projektet, som du lavede i forrige opgave. Løs følgende opgaver:

- Gør i første omgang sådan som det er gjort i førnævnte tutorial, men undlad at implementere Get metoden. Implementer i stedet en GetAll metode, uden parametre, der returnerer alle T. Du skal teste følgende med Unit Testing:
    - Test alle fire metoder i GenericRepository<T> for to udvalgte T i Entities.
    - Test at din Unit Of Work klasse rent faktisk ruller transaktionen tilbage, hvis der er sket en fejl. En transaktion skal her bestå af flere del-transaktioner, som f.eks. tilføjelse af en ny ansat, efterfulgt af opdateringen af et produkt. Du skal efterfølgende lave en del-transaktion med en fejl, eksempelvis ved at slette noget der ikke findes.

3 Udvid med derived repositories

Problemet med ovenstående GenericRepository<T> klasse er, at man altid skal have alle T fra databasen, før man begynder på filtrering og sortering. Det svarer til et SQL SELECT statement uden WHERE og ORDER BY. Derved har man dårlig performance over netværket. Imidlertid er det altid være en databases ansvar, at understøtte filtrering og sortering inden data returneres til applikationen. Derfor skal denne funktionalitet understøttes i Repository/Unit of Work design mønstret. Dette gøres ved at lave én Repository klasse for hver T, der nedarver fra GenericRepository<T>, eks.

```
public class EmployeeRepository : GenericRepository<Employee> { . . . }
```

Denne klasse kunne f.eks. have en metode, der returnerer alle ansatte i en bestemt afdeling (ikke at der findes afdelinger i Northwind, dette er blot et eksempel):

```
public List<Employee> GetAllEmployeesIn(Department department) { . . . }
```

Dine opgaver er:

- Lav et ProductRepository, der indeholder metoder til at få:
    - alle produkter der er i en bestemt kategori

    - alle produkter under eller over en bestemt pris
    - alle produkter der indgår i igangværende ordre
- Lave et OrderRepository, der indeholder metoder til at få:
    - alle ordrer fra en bestemt kunde
    - alle igangværende ordrer der kun indeholder produkter i en bestemt kategori
    - alle igangværende ordrer der indeholder mindst et produkt i en bestemt kategori
    - alle gennemførte ordrer hvis pristotal er i et bestemt interval

9 Webservices

Se først denne video og gør som Mads gør.

9.1 OpenWeatherMap API

Lav en applikation der viser vejret lige nu, for Aalborg, Aarhus, Vejle, Aabenraa og Høje Taastrup. Du skal anvende Weather API - OpenWeatherMap som remote endpoint. Anvend JSON som beskedformat. Når din applikation kan vise vejret lige nu for de fem byer, så tilføj en feature der viser vejret i morgen for hver af de fem byer.

9.2 Lav dit eget REST Web API

I denne opgave skal du konstruere et REST web API. Du skal også lære forskellen på synkron og asynkron programmering når kode skal afvikles på en web server. Nedenfor er tre videoer du skal se i denne rækkefølge:

- Viden om REST API: se denne video med Nathan Heckman.
- Demonstration af konstruktion af REST API i .NET 6: Se denne video med Patrick God.
- Demonstration af forskellen på synkron og asynkron afvikling af kode på en web server: se denne video af forfatteren.

Opgave: Byg et REST web API i .NET 6 til Northwind klient applikationer (som du ikke skal bygge i denne opgave) med JSON som dataformat, der opfylder følgende krav til klient applikationer:

1. Hent alle ordrer
2. Hent alle produkter med deres leverandører
3. Hent alle igangværende ordrer for en bestemt kunde
4. Hent alle produkter over eller under en bestemt pris
5. Gem ny kunde
6. Gem ny leverandør
7. Gem nyt produkt med eksisterende leverandør
8. Opdater kundeoplysninger
9. Slet leverandør

Du skal følge best-practice og derfor opdele ovenstående i relevante controllere. Hvis du har gennemført opgaven om Unit of Work, skal du anvende dette, i stedet for at bruge DbContext direkte fra controllerne. Du skal foretage manuelle tests med Swagger.

10 Unit Testing

Se denne video og gør som Mads gør.

10.1 Test of Time

Lav en class Time. Formålet med denne opgave er, at du implementerer en domæneklasse, som du så skal teste. Time klassen har til formål at repræsentere et tidspunkt, ligesom den indbyggede DateTime. Et Time objekt skal mindst have følgende

Fields: minute og hour.

Constructors: en uden parametre og en med minute og hour som parametre.

Properties: de to fields indkapsles og setterne kaster undtagelser, hvis value er ugyldig.

Methods: en ToString uden parametre der returnerer tiden som streng HH:mm. Klassen skal også override Object.Equals metoden.

Operators: +, -, >, <, >=, <=. Se denne video om operator overloading.

Tests

Du skal teste følgende:

1. At objekter initialiseres korrekt, og kaster undtagelser ved ugyldig initialisering
2. At objekter muteres korrekt gennem properties, og kaster undtagelser ved ugyldig mutation
3. At metoderne returnerer det forventelige, dvs. hvis tiden er 08:33 skal strengen fra ToString() være "08:33"
4. At hver af operatorerne virker efter hensigten

11 Systemkonstruktion

I dette kapitel omhandler opgaver konstruktionen af hele systemer. Det vil sige både frontend og backend. Opgaverne kan læses individuelt eller i teams. Formålet med disse opgaver er, at du oplever hvordan konstruktionen af systemer foregår i praksis. Uanset om opgaverne løses individuelt eller i teams, bør konstruktionen ske ved hjælp af et projektstyringsværktøj sammen med en valgt systemudviklingsmetode.

Hver opgave kan indledes med en elevatortale efterfulgt af en mere dybdegående systembeskrivelse. Til nogle af opgaverne vil være ekstra hjælpemidler såsom kravspecifikation og UML/ER diagrammer. Der kan også være hjælp i form af en eksisterende database og/eller kodebase.

11.1 ToDo

I denne opgave skal du konstruere et system der understøtter en ToDo liste app. Frontend er opgave 7.8. Backend skal konstrueres som opgave 9.2. Frontend skal kunne kalde backend. Der er derfor følgende delopgaver:

Lav en tom solution med enten:

1. Den nemme måde: Entities lib (Scaffolded med EF) + .NET Maui App project + Web API project hvor man kun anvender DbContext.
2. Best-practice-måden:
    a. Entities lib (Scaffolded med EF)
    b. Frontend: .NET Maui app + VievModels lib + Services lib (til at kalde serveren)
    c. Backend: Web API project + DataAccess project (med Repo/UoW med EF)

11.2 Event Collaboration Platform

1 Hjælpemidler

I denne opgave er hjælpemidlerne følgende:

1. eksisterende database
2. eksisterende kodebase på github
3. SCRUM anvendes som agil systemudviklingsmetode til teams

2 Elevatortalen

"ECP er et system til at arrangører og frivillige til forskellige arrangementer, kan samarbejde om planlægningen af gennemførelsen af et arrangement. Frivillige kan tilbyde deres hjælp til en arrangør, og en arrangør kan så udvælge hvilke frivillige man ønsker at have med til arrangementet. Der er en brugerflade hvor selve samarbejdet foregår, og det den samme for både frivillige og arrangører."

3 Systembeskrivelse

Systemet har to typer brugere: Arrangører og Frivillige.

Det typiske workflow for en arrangør: En arrangør opretter et arrangement med navn, sted, dato/tid og beskrivelse. En arrangør opretter dernæst en række forskellige opgaver der skal løses i forbindelse med arrangementet. Til hver opgave kan arrangøren ønske et antal frivillige. Med tiden tilmelder frivillige sig som kandidat til en eller flere af de opgaver arrangøren ønsker løst. Arrangøren udvælger bagefter de frivillige til den eller de opgaver arrangøren finder mest relevant, baseret på den frivilliges beskrivelse af sig selv, samt de opgaver den frivillige tidligere har været med til at løse i andre arrangementer.

Det typiske workflow for en frivillig: En frivillig gennemgår et valgt arrangement og deres opgaver og vælger hvilke opgaver den frivillige ønsker at hjælpe arrangøren med. Når en frivillig er valgt til et arrangement, kan den frivillige se det i appen.

En arrangør og en frivillig kan godt være den samme person.

4 Kravspecifikation

Funktionelle krav:

1. En arrangør skal kunne oprette, redigere og aflyse et arrangement.
2. En arrangør skal kunne se egne og andre arrangørers arrangementer, hver for sig.
3. En arrangør skal kunne oprette, redigere og slette de opgaver der knytter sig til et arrangement.
4. En arrangør skal kunne se en liste over eksisterende opgaver til et arrangement.
5. En arrangør skal kunne se alle frivillige kandidater til et arrangement.
6. En arrangør skal kunne se hvilke opgaver en bestemt frivillig har budt ind på, til et givent arrangement.
7. En arrangør skal kunne se hvilke frivillige der har budt ind på en bestemt opgave, til et givent arrangement.
8. En arrangør skal kunne tilknytte en frivillig kandidat til en opgave.
9. En arrangør skal kunne fjerne en frivillig kandidat fra en opgave.
10. En frivillig skal kunne redigere i sin beskrivelse af sig selv.
11. En frivillig skal kunne se alle arrangementer der søger frivillige.
12. En frivillig skal kunne se egen arrangement-historik med de opgaver den frivillige har været med til at løse.
13. En frivillig skal kunne kandidere sig selv til en eller flere opgaver til et arrangement.

Ikke-funktionelle krav:

1. Data gemmes i en Microsoft SQL database lokalt på hver udviklers maskine.
2. Der skal konstrueres et REST web API som backend.
3. Til frontend, skal der konstrueres én Android app som begge brugertyper anvender.
4. Al kode skal være i den samme solution.

DEL 3
ALGORITMER OG DATASTRUKTURER

12 Rekursion

12.1 Simpleste rekursion

Observer følgende C# kode:

```
int SimpleRecursion(int n)
{
    if(n == 0)  // Base case
        return -1;
    else        // Recursive case
        return SimpleRecursion(n - 1);
}
```

Kode 1: Eksempel på simpel rekursion i C#.

Resultatet når metoden kaldes med f.eks. `SimpleRecursion(3);` vil være -1. **Lav ovenstående program og efterprøv med F11 i Visual Studio**. På denne måde får du indblik i hvordan rekursionen foregår. Hold øje med værdien af parameteren n i Locals vinduet i Visual Studio, og også metodestakken i Call Stack vinduet, særligt når n får værdien `0`.

12.2 Fibonacci

Du har allerede viden om Fionacci sekvensen fra videoen. Her er pseudekode for algoritmen:

```
function Fibonacci(n)
    if n < 2
        return n
    else
        return Fibonacci(n-1) + Fibonacci(n-2)
```

Kode 2: Funktion til udregning af Fibonaccital i pseudokode.

Lav et program der udskriver sekvensen af Fibonacci tal for 2 til 20, f.eks. ved

```
procedure FibonacciSequence
    for n from 2 to 20
        print Fibonacci(n)
```

Kode 3: Procedure til udregning af Fibonaccisekvens i pseudokode.

Forskellen på en funktion og en procedure er at en funktion returnerer en værdi, hvorimod en procedure ikke returnerer en værdi. Når programmet er lavet, så dobbelttjek at sekvensen er korrekt.

Det viser sig, at når *n* bliver større så tager det længere og længere tid at køre programmet. **Prøv f.eks. med *n = 40***. Lad os undersøge hvordan tiden udvikler sig med *n*:

```
procedure FibonacciSequence
    for n from 2 to 40
```

```
        start timer
        result = Fibonacci(n)
        stop timer
        reset timer
    print n, result and elapsed time
```

Kode 4: Procedure til udregning af Fibonaccisekvens med timer i pseudokode.

Implementer ovenstående pseudokode. *Hint i C#: `System.Diagnostics.StopWatch`*.

Indtast *n* med den tilhørende tid i Ticks et Excel ark og lav en graf. Hvordan udvikler grafen sig: lineært, med potens, eksponentielt? - og hvorfor?

12.3 Fakultet

Lav et program der rekursivt beregner *n!* for *n* op til 14. Dit output skal se således ud:

```
1: 1
2: 2
3: 6
4: 24
5: 120
6: 720
7: 5040
8: 40320
9: 362880
10: 3628800
11: 39916800
12: 479001600
13: 1932053504
14: 1278945280
```

Kode 5: Output fra 14!.

Her er anvendt en iteration, der for hver *n* kalder fakultetfunktionen og udskriver til konsollen. Signaturen på fakultetsfunktionen er `int Faculty(int n)`. Observér 14. iteration, hvad sker der mon her? **Giv et bud og tilret koden, således output er korrekt for 21 iterationer**. *Hint: `System.Numerics`*.

12.4 Exceptions

Et objekt af `System.Exception` i .NET har en property `Inner`, også af typen `System.Exception`. Med andre ord *kan* en undtagelse have netop én indre undtagelse, og den undtagelse *kan* have en indre undtagelse, osv. **Du skal konstruere et .NET 5 program i et klasse bibliotek, der rekursivt kan generere en `IEnumerable<Exception>` der indeholder alle undtagelserne, ordnet med roden først**. Inden du programmerer, **skriv da pseudokode hvori du identificerer base case og recursive case**. **Du skal teste din løsning med en unit test, inden du laver en NuGet pakke som uploades til nuget.org** - den kunne f.eks. hedde `<ditProgrammørNavn>.ExceptionHandling`. På den måde kan du anvende denne pakke i alle dine fremtidige løsninger. Du finder information om hvordan man laver en NuGet pakke her.

12.5 Ekstraopgaver

A. Tilret opgave 3 således at der kan foretages flere tusinde iterationer med korrekt output. *Hint:* `System.Numerics`.
B. Lav iterative løsninger for både Fibunacci og Fakultet. Sammenlign tiden det tager at køre programmerne med med grafer i Excel, med de respektive rekursive løsninger og konkluder på det.

C. Roslyn compileren kompilerer C# kode. Undersøg om Roslyn optimerer for tail-recursion. Tail-recursion er rekursion hvor den sidste instruktion i metoden er det rekursive kald.

D. For at perspektivere emnet, læs dette blogindlæg.

E. Lav en rekursion uden base case, og find ud af hvor mange kald der er på stakken når den frygtede StackOverflowException udløses.

F. Lav Towers of Hanoi med grafisk brugergrænseflade.

13 Tids- og Pladskompleksitet

Følgende opgaver er givet som pseudokode. I hver opgave skal du:

- beregne O ud fra pseudokoden
- efterprøve med konkret kode
- lave en visuel graf over målingerne
- konkludere om din beregnede tidskompleksitet stemmer overens med dine målinger.

Husk at læse på dokumentationen for den/de anvendte operationer på den/de anvendte datastrukturer. Store O vil typisk fremgå af dokumentationen.

13.1 Sum af elementer

```
function Sum(array)
    total = 0
    for each element in array
        add to total
    return total
```

Kode 6: Output fra 14!.

13.2 Find

```
function Find(array, index)
    value = array[index]
    return value
```

Kode 7: Output fra 14!.

13.3 Indeholder

```
function Contains(array, value)
    for each element in array
        if element equals value
            return true
    return false
```

Kode 8: Output fra 14!.

13.4 Fjern

```
function RemoveAt(array, index)
```

Kode 9: Output fra 14!.

og igen for

```
function RemoveByValue(array, value)
```

Kode 10: Output fra 14!.

13.5 Matchende elementer

Denne funktion skal returnere de elementer der er ens i `array1` og `array2`:

```
function Matching(array1, array2)
    // Færdiggør selv pseudokoden inden trin 1-4
    return matchingElement
```

Kode 11: Output fra 14!.

13.6 Fakultet igen

Du har jo allerede skrevet koden fra opgave 12.3, så du kan springe implementeringen over:

```
function Factorial(n)
    if n equals 0
        return 1
    return n * Factorial(n - 1)
```

Kode 12: Output fra 14!.

13.7 Sortering

```
function Sort(array)
    for each element in array to n - 1
        for i in array to n - 1
            if array[i] > array[i + 1]
                Swap(array[i], array[i + 1]
```

Kode 13: Output fra 14!.

Skriv selv en pseudokode til `Swap`.

13.8 Ekstraopgaver

A. Efterprøv `Sort` metoden på .NET Array klassen: Denne metode benytter en hybrid sorterings algoritme. Det vil sige, at forskellige sorteringsalgoritmer anvendes efter hvor stor *n* er. Det er ikke nødvendigt for dig at kende de forskellige sorteringsalgoritmer. Du skal efterprøve de forskellige algoritmer ved at variere *n* og måle resultaterne.

B. Efterprøv følgende metoder for .Net `List<T>`: Add, Contains, Remove, Reverse og Sort om deres tidskompleksitet passer med dokumentationen.

14 Søgealgoritmer på Lineære Datastrukturer

Til resten af opgaverne i denne samling skal du oprette to biblioteker, *DataStructures* og *Algorithms*.

14.1 Er arrayet sorteret?

Lav en funktion i *Algorithms*, der afgør om et array er sorteret eller ej. Dette er en forudsætning for at kunne producere meningsfulde resultater fra binære søgninger, som næste opgave omhandler.

14.2 Binær søgning på sorteret array af unikke elementer

Implementer en binær søgealgoritme, for eksempel ved hjælp af følgende pseudokode:

```
// This assumes we are given a sorted array a of size n and a key x.
// Use integers left and right (initially set to 0 and n-1) and mid.
While left is less than right,
    set mid to the integer part of (left+right)/2, and
    if x is greater than a[mid],
        then set left to mid+1,
        otherwise set right to mid.
If a[left] is equal to x,
    then terminate returning left,
    otherwise terminate returning -1.
```

Kode 14: Uddrag fra "Lecture Notes for Data Structures and Algorithms" af Martín Escardó, Manfred Kerber og John Bullinaria, alle School of Computer Science, University of Birmingham.

Tillad kun algoritmen på arrays der er sorterede og kun har unikke elementer. Din algoritme skal være en funktion der modtager et array som argument, samt et tal (det der søges på). Funktionen skal findes i dit *Algorithms* bibliotek.

14.3 Lineær søgning og sammenligning med binær søgning

Lav endnu en funktion i *Algorithms*, der foretager en lineær søgning efter et element i et array og returnerer indekset – ligesom den binære søgning.

Redegør for tidskompleksiteten for både den lineære og den binære søgealgoritme, ved at lave en graf over måleresultaterne fra målinger af begge algoritmer. Sørg for at måle begge algoritmer henholdsvis når *n* er lille (<100) og når *n* er stor (>10.000.000.000). *Hint: for en (næsten) videnskabelig demonstration af målemetoder, så lad dig inspirere af dette program.*

Overvej resultaterne for lille *n* henholdsvis stor *n*.

14.4 Tag højde for dubletter

Fordi en binær søgning kræver at elementerne i arrayet er sorterede, og hvis der tillades dubletter, vil der blive fundet én af disse. Da arrayet er sorteret vil dubletterne altid være ved siden af hinanden. Tilføj funktioner, der finder den dublet der er længst til højre, henholdsvis den der er længst til venstre.

15 Sorteringsalgoritmer på Lineære Datastrukturer

I dette kapitel skal du implementere forskellige sorteringsalgoritmer.

15.1 Implementer 7 forskellige sorteringsalgoritmer

Du finder pseudokoden i Lecture Notes. For hver af disse sorteringsalgoritmer skal du gøre følgende:

1. Forklar hvert skridt i algoritmen i et tekstbehandlingsprogram
2. Implementer algoritmen i *Algorithms*
3. Lav funktionstest der afprøver om algoritmen virkelig sorterer input
4. Lav efterprøvning af algoritmens tidkompleksitet med visuelle grafer, hvor *n* varierer fra 10 til 1.000.000.000 i trin af 10, 100, 1.000, 10.000 osv.

Sorteringsalgoritmerne

A. Bubble Sort
B. Insertion Sort
C. Selection Sort
D. TreeSort
E. HeapSort
F. QuickSort
G. MergeSort

15.2 Konkluder på performance

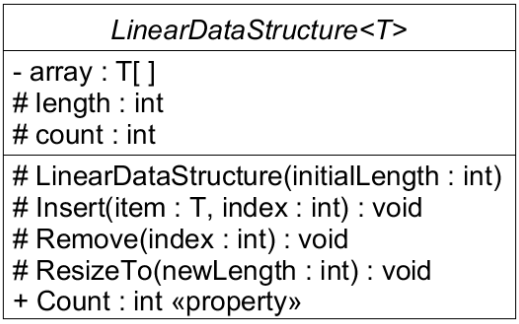
Hvilken algoritme har den bedste performance for et givent *n*? List dem alle i et tekstbehandlingsprogram.

15.3 Hybrid sorteringsalgoritme

Lav en funktion, der hedder HybridSort. Den skal anvende 2-4 af dine bedst performende sorteringsalgoritmer alt efter hvad *n* er. Efterprøv performance; er den bedre end de enkeltstående sorteringsalgoritmer?

16 Implementering af Lineære Datastrukturer

I stedet for at anvende de indbyggede lineære datastrukturer som stak, kø og linked lister, skal du lave dine egne. De forskellige datastrukturer har en række fællestræk, derfor kan du med fordel implementere følgende i *DataStructures*:



Figur 6: UML diagram der viser et forslag til en generisk abstrakt klasse LinearDataStructure, hvorudfra andre konkretiseringer af en lineær datastruktur, kan implementeres.

I .NET ville man således kun anvende `using System;`. Vi observerer at klassen har et internt array til at holde data, samt en række protected members. Et array kan ikke umiddelbart ændre størrelse, men eftersom vi ønsker en dynamisk længde, skal klassen understøtte at arrayet øger sin størrelse i takt med at der indsættes nye elementer. Men det behøver ikke at være en forøgning for hvert nyt element der tilføjes, det kan i stedet ske for hver 4. eller 10. element der tilføjes. Det kan også være arrayet performer bedre ved en generel formel som f.eks.: ny længde = gammel længde x faktor. Idéen med at lave størrelsen dynamisk, er at forøge performance for algoritmer der anvendes på arrayet. Overvej f.eks. hvad der ville ske hvis det interne array altid havde længden $\frac{2^{32}}{2} - 1$.

16.1 Implementering og test af LinearDataStructure<T>

Implementer klassen. Bemærk at ovenstående blot er et forslag, du bestemmer selv hvordan du vil gøre. Det er dog ikke tilladt at løse opgaven med kode, der anvender namespaces/biblioteker, der ikke er i den grundlæggende del af BCL. I .NET er den grundlæggende del af BCL namespace System.

Lav nu en klasse List<T> der arver fra LinearDataStructure. Test denne klasse med unit test.

16.2 Implementer en Stak

Elementerne i din stak skal være uafhængige af stakkens implementering. Det vil sige, at elementerne ikke må have funktionalitet, der relaterer elementet til et andet element i stakken. I C# kunne dette have være en property Next eller Previous.

Først skal du besvare følgende teoretiske opgaver:

1. Bogstav betyder Push() og * betyder Pop() i følgende sekvens, der starter på en tom stak:
   E A S * Y * Q U E * * * S T * * * I O * N * * *
   Hvad er pop sekvensen? Og hvor stor er stakken efter sekvensen?
2. Bogstav betyder Push() og * betyder Pop() i følgende sekvens, der starter på en tom stak:
   L A * * S T I * N * F I R * S T * * O U * T * * * * * *
   Hvad er pop sekvensen? Og hvor stor er stakken efter sekvensen?

Implementer nu en stak og test den. Hvad skal der ske, når man forsøger at poppe fra en tom stak? – null eller undtagelse?

Løs disse opgaver med stak/stakke:

1. Skriv et program, der læser en karaktersekvens og printer dem i omvendt rækkefølge.
2. Skriv et program der kan afgøre om en streng er et palindrom.
3. Løs Hanois tårne med din stak og rekursion – dog med den betingelse at skiverne på pind A skal over på pind C (og ikke blot B, som man ofte finder på internettet). T kan blot være en int, hvor et størrelsen af tallet angiver størrelsen af skiven. Forsøg først at løse problemet hvor du flytter flere skiver på én gang. Forsøg dernæst at løse problemet ved kun at flytte én skive ad gangen. Kan dette lade sig gøre for vilkårligt store *n*?

16.3 Implementer en Kø

Elementerne i din kø skal være uafhængige af stakkens implementering. Det vil sige, at elementerne ikke må have funktionalitet, der relaterer elementet til et andet element i køen. I C# kunne dette have være en property Next eller Previous.

Implementer en kø og test den.

Løs disse opgaver med en kø:

1. Lav en kø der indeholder karakterne 'F', 'I', 'L', 'O'. Få dem til at blive printet én ad gangen med 0,5 sek. intervaller, sådan som et banner i bunden af skærmen på TV2 News. Processen skal gentages indtil programmet lukkes.
2. Lav en funktion på klassen, der vender køen om. Gentag nr. 1.

16.4 Implementer en Linked List

Implementer en linked list og test den. Bemærk følgende:

- at den linkede liste skal være ensrettet (altså ikke være en doubly linked list)
- at elementerne i listen skal være uafhængige af listens implementering (på samme måde som stak og kø ovenover). Dette kræver at man laver en Node<T> klasse.

Lav et eksempel hvor du anvender funktionerne i din linkede liste og lav også et UML diagram over implementeringen.

16.5 Ekstraopgaver

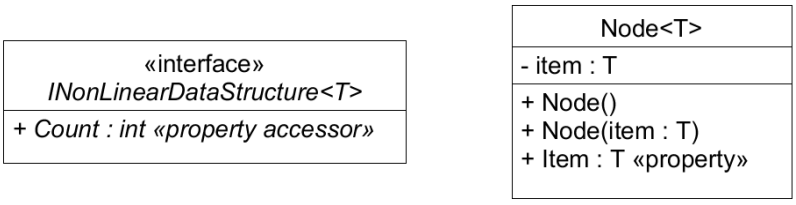
A. Lav en kø ud af to stakke.

B. (ret svær) Løs opgave 19.6 i "Bådbogen" (Visual C# How to Program Global Edition, 6e, Deitel & Deitel).

17 Implementering af Træer

17.1 Opret typer til senere brug

Det er smart at tænke lidt generaliserende, når man laver et bibliotek til mange forskellige datastrukturer, eksempelvis ved at lave grundlæggende typer til at starte med som man så kan genbruge til andre ting senere. Så i det følgende anvendes .NET 5 med C# 9 og UML, til at illustrere principper og teknikker. Det første man skal gøre når man laver sit eget bibliotek til datastrukturer, er at overveje hvilke interfaces man kan lave og hvad de skal indeholde. Det kunne for eksempel være disse to typer:



Figur 7: UML klassediagram med forslag til interface og base class.

Interfacet INonLinearDataStructure repræsenterer en ikke-lineær datastruktur, som for eksempel et binært træ, men det kan også være en række andre træer, og endog generaliseringen af træer, der kaldes grafer. Interfacet har én property der repræsenterer antallet af elementer i datastrukturen. Stereotypen property accessor, angiver, at der først og fremmest er tale om en property, men også at den ikke har en (public) set funktion. Node klassen repræsenterer en node/"knude" i datastrukturen, og den er generisk hvor T kan være en hvilken som helts type. Vi bemærker også, at disse to typer umiddelbart ikke har noget med hinanden at gøre, da der ikke er en association mellem dem.

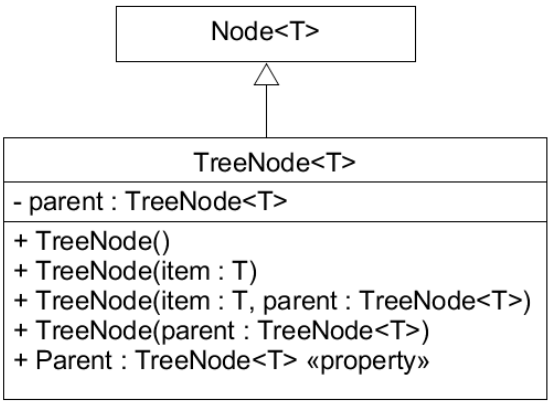
- Implementér de to typer.

17.2 Implementér typer til træer

Et træ er karakteriseret ved to ting:

- Et træ har altid en rod, *root*.
- Alle nodes, kan have en eller flere child nodes.
- Alle nodes, undtaget root, har netop én parent node.

Lad os omsætte dette i et UML diagram:



Figur 8: UML klasse diagram med forslag til hierarki.

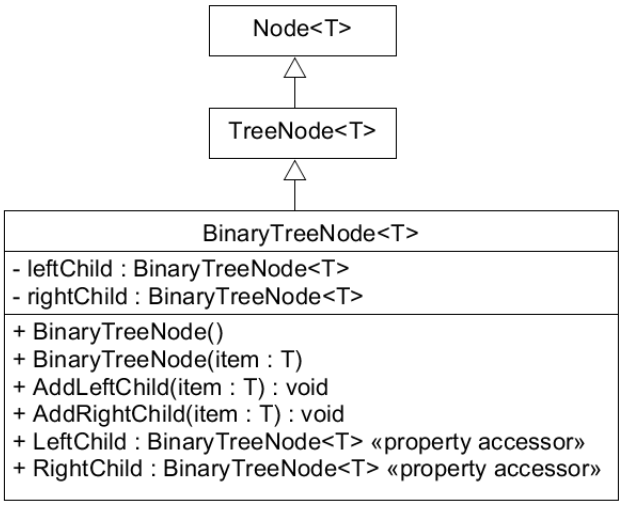
- Implementér TreeNode klassen.

17.3 Implementer et binært træ

Et binært træ er en specialisering af et træ. Den har følgende karakteristika:

- Et binært træ har altid en rod, *root*.
- Alle nodes, kan have en eller to child nodes.
- Alle nodes, undtaget root, har netop én parent node.

Bemærk hvordan det binære træ adskiller sig fra træer generelt. Vi kan skitsere en binære node med UML således:



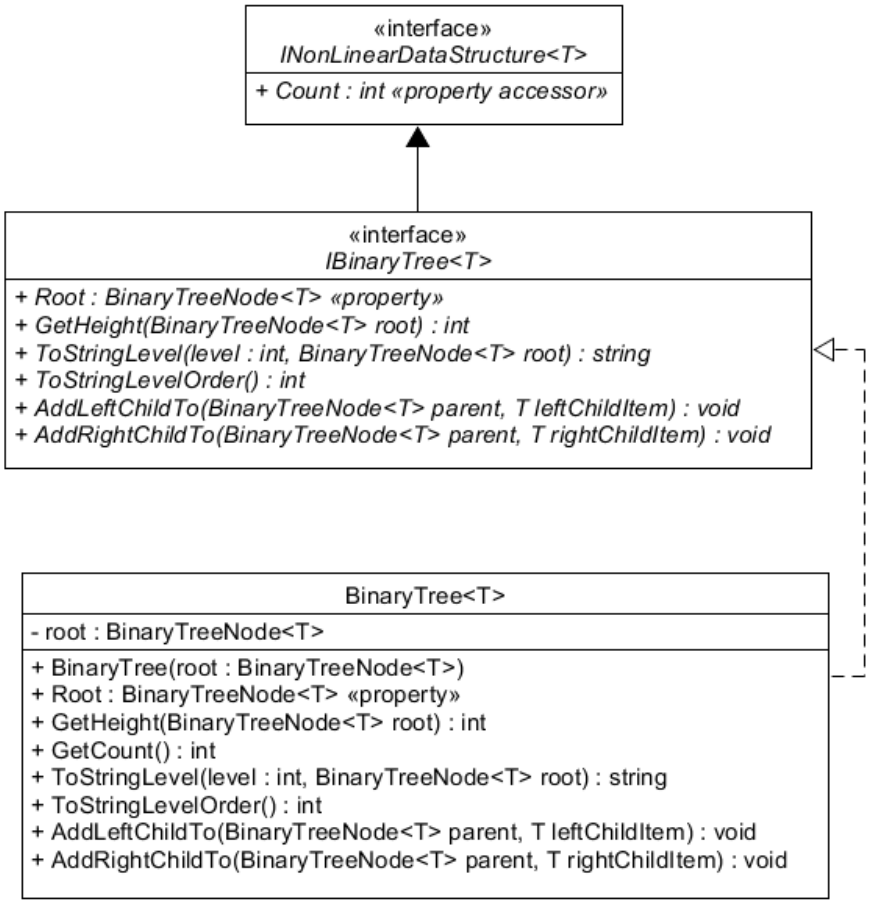
Figur 9: UML klasse diagram med forslag til hvordan BinaryTreeNode<T> arver fra TreeNode<T>.

- Implementér BinaryTreeNode klassen.

Bemærk AddLeftChild- og AddRightChild metoderne, i stedet for public settere til LeftChild og RightChild property. Det er et lille trick, idet de to metoder jo skal sætte parent for den nye child

node. Og det er jo ikke en property's ansvar. Bemærk også at metoderne ikke har BinaryTreeNode<T> som parametre, men i stedet blot T.

Lad os nu se på implementering af et binært træ. Først laves interfacet, og dernæst den konkrete type:



Figur 10: UML klassediagram med implementering af interface.

Du bør først implementere GetHeight, ToStringLevel og ToStringLevelOrder i næste kapitel.

18 Traversering i Træer

Fortsat fra kapitel 17.

18.1 Bredde først

Højden af træet er antallet af "levels" i træet. ToStringLevel er giver en streng repræsentation af ét bestemt level i træet. ToStringLevelOrder er en streng repræsentation af alle levels i træet, startende med root. Her er pseudokode til GetHeight:

```
Hvis root er null så returner 0.
Ellers
    Sæt højde af venstre undertræ til GetHeight(root.LeftChild).
    Sæt højde af højre undertræ til GetHeight(root.RightChild).
    Retruner højden af det højeste undertræ + 1.
```

Kode 15: Uddrag fra "Lecture Notes for Data Structures and Algorithms" af Martín Escardó, Manfred Kerber og John Bullinaria, alle School of Computer Science, University of Birmingham (oversat af forfatteren).

Ovenstående implementering af GetHeight, kaldes for en "bredde først" traverseringsalgoritme ("breadth-first"). Traversering betyder "gennemgå".

- Konstruer en bredde først algoritme, der giver strengrepræsentationen af et enkelt level i det binære træ, i metoden ToStringLevel. Lad ToStringLevelOrder kalde ToStringLevel iterativt.

18.2 Dybde først

Der findes tre former for dybde først traversering: inOrder, preOrder og postOrder.

- Implementer de tre algoritmer, således at en streng repræsentation af træet opnår ved kørsel af algoritmen – på samme vis som ToStringLevel.

19 Specialiserede Træer

19.1 Det binære søgetræ

Du har set to måde at bygge et binært træ på; enten at kalde Add metoderne på træet, eller at bygge nodes sammen, ved Add metoderne på nodes og så give root node som argument til constructoren i det binære træ. Hvad hvis man kunne lave en metode Insert, der sørger for at placere et element det korrekte sted, uden at den kaldende kode skal tage stilling til hvor det korrekte sted er? Dette kræver at objekterne af typen T der indsættes, skal kunne sammenlignes med hinanden. I .NET vil man lave et generisk constraint på sit træ, der kun tillader T hvis T implementerer IComparable<T>. En sådan Insert algoritme, der udnytter at objekter sammenlignes inden de indsættes på en bestemt plads træet, kan skrives således i pseudokode:

```
Tree procedure Insert(item)
    hvis root er null
        lad root indeholde item
    ellers
        root.Insert(item)

Node procedure Insert(item)
    hvis item er mindre end item i node
        hvis left er null
            lav en ny left child node med item som indhold
        ellers
            kald Insert(item) på left child node
    ellers hvis item er større end item i node
        hvis right er null
            lav en ny right child node med item som indhold
        ellers
            kald insert(item) på right child node
```

Kode 16: Pseudokode til indsættelse I det binære søgetræ.

Dette kalder man for et binært søgetræ og det skal du implementere. Din implementering bør specialisere BinaryTree og BinaryTreeNode.

- Implementer Insert i både det binære søgetræ og i den binære node.
- Test dit træ ved at indsætte værdier af f.eks. int.
- Prøv at printe dit træ og forklar hvordan træet udvikler sig når der indsættes en ny værdi.
- Udvid dit træ ved at tilføje en funktion InsertMany(lineær datastruktur).

19.2 Binære Heaps

Heap betyder bare *bunke*. En binær heap er en specialisering af det binære træ. Der findes to typer binære heaps, binær max heap og binær min heap. Mere viden findes her.

- Implementer MaxHeap og MinHeap heaps.
- Sørg for at disse funktioner er på hver heap: Insert, Extract, Search og Delete.

19.3 AVL træ

Et AVL træ er også en specialisering af et binært træ. Det besidder den særlige egenskab, at det kan balancere sig selv. Dette kræver en række operationer, der kaldes for rotationer. Mere info her, men find selv en video der viser balanceringen af et AVL træ.

- Implementer et AVL træ med rotationsfunktioner.

19.4 M-ary træ

Et binært træ er et eksempel på et træ, der altid har nul til to child nodes. Et m-ary træ, er en generalisering af et binært træ, hvor hver node kan have nul til *m* child nodes.

- Implementer et træ således at hver node kan have nul til mange child nodes. Hint: I stedet for LeftChild og RightChild i et binært træ, så lad en liste indeholde alle child nodes til en node.
- Implementer ToString funktion(er) til træet, både med bredde først og med de tre dybde først algoritmer.

19.5 Splay træ

Et såkaldt *splay* træ er et binært søgetræ, der selv sørger for at omarragere elementerne, alt efter hvor tit de bliver tilgået, således at det element der tilgås mest, bliver root. Implementer et splay træ.

20 Søgealgoritmer i Træer

20.1 Implementer følgende i dine binære træer fra opgaverne i kapitel 19:

- En funktion til at finde et element. Hint: anvend én af dybde først algoritmerne.
- Et binært træ kan implementeres som en indeksbaseret lineær datastruktur. Find selv viden og implementer det. Er der performance forskel til træimplementeringen når man søger?

21 Implementering af Hashtabeller

Hashtabeller er meget effektive i tid. Indsættelse og hentning af data i en hashtabel sker i konstant tid. Se denne video: HackerRank – DataStructures: HashTable og læs denne artikel: Wikipedia – Hash Table

En hashtabel kan implementeres som en lineær datastruktur, der indeholder værdierne til nøglerne. Selve nøglerne gemmes ikke. I den første opgave skal du implementere den simple hashtabel der ikke håndterer kollisioner. I den næste opgave skal du håndtere kollisioner med chaining. Men først lidt viden om hashfunktioner:

- Andrew Lock | .NET Escapades: Why is string.GetHashCode() different each time I run my program in .NET Core?
- Thomas Levesque's .NET Blog: Things every C# developer should know #1: Hash codes

Her er et bud på en nogenlunde okay hashfunktion der anvender modulus:

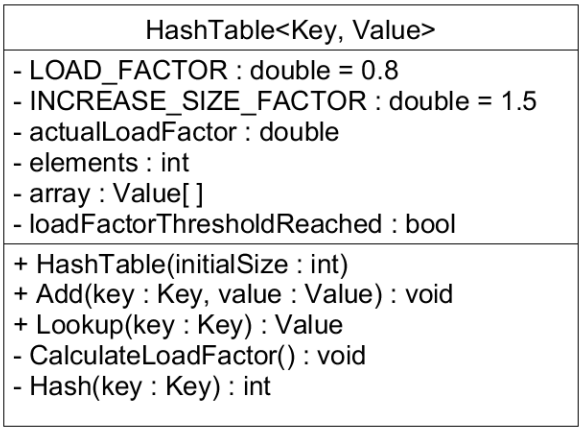
```
1       function Hash(k, m)
2           hashkode = hashkode af k
3           maske = right shift af 31 bits af hashkode
4           hashkode = hashkode XOR maske
5           hashkode = maske subtraheret fra hashkode
6           hashkode = hashkode modulus m
7           returner hashkode
```

Kode 17: Pseudokode med forslag til hashfunktion.

Her er k nøglen og m er antal buckets (typisk det samme som arrayets længde). På de fleste afviklingsplatforme vil linje 2 give et heltal der kan være negativt. Linje 3-5 konverter et eventuelt negativt tal til et positivt tal – det er jo nødvendigt da hashkoden anvendes som indeks i et array.

21.1 Den simple hashtabel uden håndtering af kollisioner

Lad os kigge nærmere på hvordan man kan implementere en hashtabel som lineær datastruktur uden håndtering af kollisioner. Betragt følgende klassediagram:



Figur 11: UML klassediagram med forslag til HashTable<Key, Value>

Konstanten LOAD_FACTOR beskriver en grænseværdi for hvornår arrayet skal forlænges, i dette tilfælde når det er 80% fyldt. Konstanten INCREASE_SIZE_FACTOR beskriver hvor meget arrayet skal forlænges med, når arrayets load factor grænseværdi nås ved indsættelse. Variablen actualLoadFactor beskriver hvor stor en del af arrayet der indeholder elementer, altså antallet af elementer delt med størrelsen på arrayet. Den tilhørende udregning efter hver indsættelse af et element, beregnes i metoden CalculateLoadFactor. Før hver indsættelse skal man teste om load factor grænseværdien er nået ved test af variablen loadFactorThresholdReached. Hvis ja, så skal arrayet forlænges med INCREASE_SIZE_FACTOR, inden hashkoden beregnes. Resultatet af denne beregning, bliver til det indeks værdien til nøglen indsættes på. Når man gerne vil hente en værdi der hører til en nøgle, beregnes hashkoden af nøglen, der jo er indekset af nøglens tilhørende værdi i arrayet.

- Implementer en hashtabel der ikke understøtter kollisioner

21.2 Hashtabel med håndtering af kollisioner ved sammenkædning

Hashtabellen fra forrige opgave kan ikke håndtere kollisioner, så med den forholdsvis dårlige hashfunktion og lav initialSize er kollisioner uundgåelige allerede efter få indsættelser af nøglers værdier.

Chaining (sammenkædning) gør, at flere værdier kan indsættes på samme indeks når der sker kollision, men samtidig skal nøglen også gemmes sammen med den værdi, da det ellers ikke er muligt at afgøre hvilken nøgle værdien tilhører. Altså skal der på hvert indeks være en dynamisk lineær datastruktur, der kan indeholde både nøgle og værdi. På den måde vil hashtabellen bestå af flest værdi-elementer, hvor længden af den dynamiske lineære datastruktur er 1, og få hvor længden er mere end 1. Der er klart, at på indeks hvor der er kollisioner, vil være en anden tidskompleksitet end konstant tid.

- Implementer en hashtabel der understøtter kollisioner med chaining

21.3 Hashtabel med sammenkædning og hoveder

Da en hashtabel der understøtter chaining har en dynamisk lineær datastruktur på hvert indeks, men kun få hvor længden af denne er større end 1, så lad os se på hvordan vi kan optimere dette. Den generiske type parameter Value, kan ændres til at være en BucketEntry<Key, Value> der har to values:

- Head (hovedet) af typen Value
- Chain af den dynamisk lineær datastruktur der indeholder objekter af typen Value

I forrige opgave var hvert indeks en dynamisk lineær datastruktur, også selvom der ikke var kollision. Vi optimerer nu, således at:

- En indsættelse af en nøgles værdi på et indeks hvor der ikke er andre værdier, bliver til Head
- En indsættelse af en nøgles værdi på et indeks hvor der er 1 eller flere værdier, bliver til indsættelse af både nøgle og værdi i en dynamisk lineær datastruktur.

21.4 Hashtabel med sammenkædning med træer

Hashtabellen kan optimeres endnu mere, hvis Value er af en type der kan sammenlignes med sig selv (I .NET skal denne type implementere IComparable<T>). Hvad nu hvis vi anvendte et træ i stedet for den dynamiske lineære datastruktur, når der sker kollisioner?

- Implementer et løsning hvor du erstatter den dynamiske lineære datastruktur med et træ
- Test performance forskellen og noter dine resultater

21.5 Ekstraopgaver

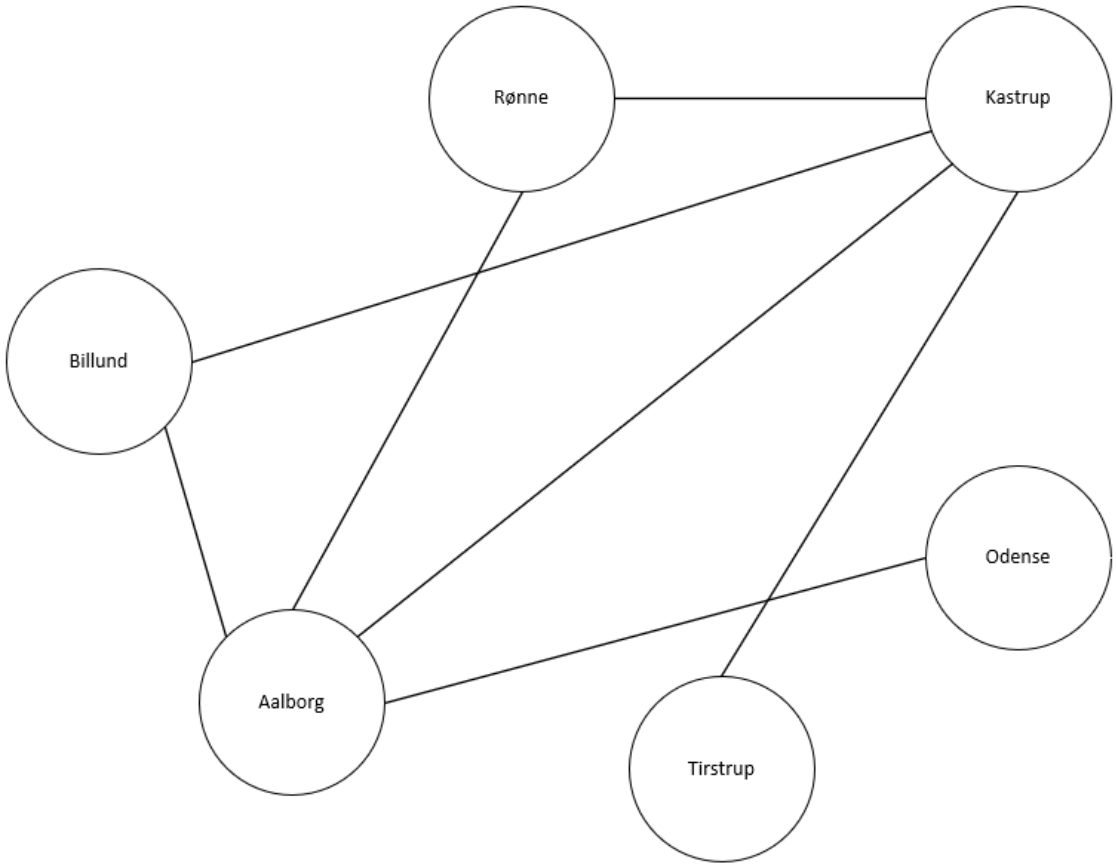
A. Tilføj funktionalitet til at slette elementer, og husk at afstemme den nye størrelse af træet til load factor.

B. Tilføj funktionalitet til at opdatere en value.

C. Tilføj muligheden for at klientkoden kan vælge mellem flere forskellige hashing algoritmer.

22 Implementering af Grafer

En graf er en datastruktur der kan repræsentere et netværk. Et netværk er f.eks. lufthavne og ruterne imellem, venner på Facebook eller et egentligt computernetværk af klienter forbundet gennem routere. Her visualiseres eksemplet med lufthavne og ruter:



Lufthavnene kaldes *knuder* (vertex, vertices) og ruterne mellem kaldes *kanter* (edges). Lad os tage et eksempel på en rute (path) gennem grafen: Man ønsker at flyve fra Billund til Rønne. Vi kan se at den nemmeste vej er via enten Kastrup eller Aalborg.

Lad os kigge på hvordan vi kan repræsentere dette i kode:

Vertex<T>
visited : bool # neighbours : List< Vertex<T> >
+ Vertex(item : T) + AddNeighbour(neighbour : Vertex<T>) : void + HasNeighbour(neighbour : Vertex<T>) : bool

Kanten repræsenteres ikke som konkrete objekter i dette tilfælde, men blot som nabo til en knude. I modsætning til træer, har grafer ikke en root node.

22.1 Implementering af graf som refererede knuder

- Implementer Vertex<T> og lad den arve fra Node<T>.
- Test Vertex<T> hvor T er en class Airport, og implementer ovenstående klassediagram. Undlad at lave traverseringsalgoritmer. Print grafen.

23 Algoritmer til Grafer

23.1 Traversering

Man traverserer grafer på samme vis som træer, altså med enten dybde-først eller bredde-først.

- Implementer en class GraphTraverser, der kan traversere en graf af Vertex<T> objekter med både bredde- og dybde-først. Traverseringen skal returnere en streng med alle besøgte knuders interne værdier for T.

23.2 Kanter med vægt

Objekter der repræsenterer kanterne, bliver vigtige når kanterne har en vægt. En vægt på en kant er f.eks. rejsetiden eller prisen osv.

- Implementer kanter således at TEdge indgår i typeparameterlisten. TEdge bør implementere IComparable<TEdge> og den konkrete afviklingstype bør understøtte aritmetiske operationer – dette bliver vigtigt i næste afsnit.
- Test hvor du angiver vægte til hver kant og print med traversering.

23.3 Dijkstras algoritme

Viden: Computer Science – Graph Data Structure 4. Dijkstra's Shortest Path Algorithm

Opret en klasse Graph<TVertex, TEdge>. Denne klasses ansvar er at udføre operationer på de enkelte knuder i grafen, og bør man bør derfor kunne tilføje knuder (med kantvægte) og finde den korteste vej mellem to knuder.

DEL 4
MODERNE TEKNOLOGIER

24 ASP.NET Core MVC med Dependency Injection og Inversion of Control

Se denne video og gør som der bliver gjort i videoen:

IAmTimCorey: Introduction to ASP.NET Core MVC in C# plus LOTS of Tips

24.1 Products and Suppliers website

Denne opgave har til formål, at du opnår mere rutine i at bygge et website med ASP.NET Core.

Du skal anvende Northwind databasen som datakilde, og Entity Framework Core som ORM. Sørg for at anvende Scaffold-Db til at få EF til at generere entiteterne.

Krav

I et ASP.NET Core MVC project med Entity Framework kan man få autogenereret de nødvendige websider og tilhørende funktioner til de forskellige entiteter. Du skal sikre at en bruger kan:

- Se alle produkter
- Opdatere et produkt
- Indtaste et nyt produkt
- Slette et produkt

Da der er en relation mellem produkter og leverandører i databasen, du samtidig også sørge for at en bruger kan:

- Se alle leverandører
- Opdatere data på en leverandør
- Indtaste en ny leverandør
- Slette en leverandør, og dens tilhørende produkter (EF Cascade Delete)

24.2 Social Media Authentication med ASP.NET Core

Fokus I denne opgave er at lære hvordan man anvender autentifikation fra eksterne udbydere, særlig sociale medier, såsom Google, Facebook, LinkedIn osv.

Du skal lave en webapplikation (ASP.NET Core MVC) hvor man som bruger kan lave opslag som andre brugere kan kommentere på – med andre ord et socialt medie. En bruger skal oprette sig enten med mail og password eller med en af mindst 3 forskellige eksterne udbydere af autentifikation. Du finder viden på MS Docs her, og en god DZone artikel om emnet her.

Websitet skal køre på din egen PC og det skal databasen også.

Ekstra opgave:

Eksponer et REST API med lidt forskellig data om sitet, eksempelvis antal opslag på sitet på en bestemt dag osv. Men sørg for at klienten har et gyldigt token, for at kunne tilgå API'et.

25 Microservices

Viden om microservices: Edurika – What are miroservices.

Gør som Ion gør Cognizant Softvision: Building a gRPC service with .NET Core.

25.1 Konstruer din egen microservice

Din microservice skal have Northwind databasen som datakilde. Din microservice skal eksponere funktionalitet svarende til kravene i opgave 19.1. Bemærk, at klienten således skal kunne oprette en kanal til serveren med de data der skal tilføjes, opdateres eller slettes.

25.2 Opbyg en microservice arkitektur

Idéen er, at din server har tre microservies:

- En gateway
- En auth service (gør det simpelt)
- En data service (Northwind data)

Gatewayen er den eneste microservice der er eksponeret eksternt. Alle requests routes gennem auth servicen, inden der tillades interaktion med data servicen. Prøv at konstruere denne arkitektur, og bagefter prøv at refactor dit website fra forrige kapitel til at anvende din microservice arkitektur.

DEL 5
Kompetencetests

26 Midtvejstest i Imperativ Programmering

Formålet med denne test er at afdække dit niveau i imperativ programmering. Imperativ programmering dækker over emnerne:

- Variable og Datatyper
- Kontrolstrukturer (selektion og iteration)
- Lineære Datastrukturer
- Programkonstruktion

Testen må højst tage 4 timer og 30 minutter.

Testen består af tre dele:

- Videndelen: Her skal du besvare spørgsmålene i et Word dokument, således at du demonstrer din viden i imperativ programmering.
- Færdighedsdelen: Her skal du løse mindre opgaver, således du kan demonstrere dine færdigheder inden for enkeltstående elementer af imperativ programmering.
- Kompetencedel: Her skal du løse en lidt større opgave, således du demonstrer at du kan anvende din viden og dine færdigheder i samspil.

VIGTIGT: Du skal sørge for at være meget omhyggelig, præcis og grundig i dine besvarelser; I videndelen lægges der vægt på at du anvender fagtermer, og kun besvarer det stillede spørgsmål. I både færdigheds- og kompetencedelen lægges der vægt på at du kun skriver den nødvendige kode der løser opgaven, at du imødekommer opgavens krav præcist, samt at du udviser god kodeskik i forhold til navngivning, formatering og logik.

Bedømmelsesmetoden er som følger: hvert korrekt svar på et spørgsmål/opgave giver 1 point. Et delvist korrekt svar giver 0,5 point. Et ikke korrekt eller ikke besvaret spørgsmål/opgave giver 0 point. Det er beskrevet i forrige afsnit hvad kriteriet er for at en besvarelse vurderes som korrekt. Der indgår desuden følgende vægtning af de tre dele i testens samlede resultat: videndel: 25%, færdighedsdel: 25% og kompetencedel: 50%.

Dit Word dokument og din Visual Studio solution skal pakkes i en .zip, og sendes til bedømmelse.

26.1 Videndel

Du skal besvare følgende spørgsmål skriftligt i et word dokument:

1. Hvad er forskellen på en variabel og en datatype?

2. Hvad er den generelle syntaks for erklæring af en variabel?

3. Er det muligt for en variabel af datatypen int, at indeholde følgende værdi: 0 ?

4. Er det muligt for en variabel af datatypen bool, at indeholde følgende værdi: perhaps ?

5. Angiv samtlige mulige værdier en variabel af datatypen bool, kan antage.

6. Er det muligt at konvertere en variabel af datatypen byte til datatypen int? Er det omvendte også muligt? Beskriv hvorfor det er muligt/ikke muligt.

7. Hvilken metode skal kaldes på Console klassen, når man ønsker at indlæse indtastninger fra tastaturet?

8. Hvilken metode skal kaldes på Console klassen, når man ønsker at udskrive tekst til konsol vinduet?

9. Hvilke(t) statement(s) kan anvendes, når man ønsker at bestemte andre statements eksekveres, mens igen andre ikke gør – på baggrund af en logisk betingelse?

10. Hvilke(t) statement(s) kan anvendes, når man ønsker at gentage et eller flere andre statements, indtil en logisk betingelse er opfyldt?

11. Angiv en syntaks til initialisering af en lineær datastruktur der kan indeholde præcis 100 heltal.

12. Angiv en syntaks for en iteration over en lineær datastruktur, der til konsollen udskriver alle værdier i den lineære datastruktur.

26.2 Færdighedsdel

I denne opgave må kun `using System;` være i kodefilen, andre using's skal slettes. Hver opgave skal løses i sit eget konsol projekt:

1. Skriv et program der viser følgende tekst i konsollen: **C# er et programmeringssprog**

2. Skriv et program der først viser følgende tekst til brugeren: **Hej bruger. Indtast dit navn her:** Efter brugeren har indtastet sit navn, skal brugeren få vist følgende tekst: **Velkommen, <navn>!**

3. Skriv et program hvor brugeren skal indtaste en værdi i intervallet [1;10]. Brugeren skal få en tekstbesked, der indikerer om den indtastede værdi var over 5, mindre end 5 eller præcis 5.

4. Skriv et program hvor brugeren skal indtaste tre tal. Brugeren skal få vist hhv. additionen, subtraktionen og multiplikationen af de tre tal.

5. Skriv et program der udregner momsbeløbet (25%) af en af brugeren indtastet pris. Prisen skal bestå af kroner og øre.

6. Skriv et program der udregner totalprisen for to varer som brugeren har indtastet (navn på varerne og varernes pris), samt discountprisen. Discountprisen udregnes ved at give 10% rabat på den første vare, og 20 procent rabat på den anden vare.

7. Skriv et program der har en lineær datastruktur der indeholder disse fem heltal: 7, 15, 22, 38, 46. Programmet skal ved iteration udskrive følgende til konsollen:

- Tallene.
- Den negative sum af alle elementerne ved bagvendt iteration – det vil sige elementerne fratrækkes hinanden, og iterationen starter bagfra i den lineære datastruktur.
- Summen alle af alle ulige tal ved anvendelse af modulo operatoren.

26.3 Kompetencedel

I denne opgave må kun `using System;` og `using System.Collections.Generic;` være i kodefilen, andre using's skal slettes. Du skal anvende ét konsol projekt til at løse opgaven, og du må kun anvende Main metoden – det vil sige at du ikke må lave andre metoder. Skriv et program der opfylder følgende krav:

1. Brugeren skal præsenteres for en menu med tre valgmuligheder:

- Indtast varer
- Udregn total
- Afslut

2. Ved ikke-gyldigt valg, skal brugeren først mødes af en fejlbesked, og herefter kunne forsøge igen og indtil et gyldigt valg er indtastet.

3. Ved valg af Indtast varer, skal brugeren kunne indtaste en vare og dens pris. Brugeren skal have muligheden for at vende tilbage til den første menu, når brugeren ikke har flere varer at indtaste.

4. Ved valg af Udregn total, skal brugeren kunne se samtlige indtastede varer og deres priser, samt den udregnede sum af alle varer priser. Brugeren skal have muligheden for at vende tilbage til den første menu.

5. Ved valg af Afslut, skal brugeren mødes af en farvel besked.

6. Du skal anvende semantisk beskrivende navngivning af variable, overholde god formatering og i øvrigt overholde best practice.

27 Kompetencetest i Grundlæggende Programmering

Formålet med denne test er at afdække dit niveau i grundlæggende programmering. Dette bygger ovenpå imperativ programmering. Testen må højst tage 4 timer og 30 minutter. Testen består af tre dele:

- Videndelen: Her skal du besvare spørgsmålene i et Word dokument, således at du demonstrer din viden i faget.
- Færdighedsdelen: Her skal du løse mindre opgaver, således du kan demonstrere dine færdigheder inden for enkeltstående elementer af fagets mål.
- Kompetencedel: Her skal du løse en lidt større opgave, således du demonstrer at du kan anvende din viden og dine færdigheder i samspil.

VIGTIGT: Du skal sørge for at være meget omhyggelig, præcis og grundig i dine besvarelser; I videndelen lægges der vægt på at du anvender fagtermer, og kun besvarer det stillede spørgsmål. I både færdigheds- og kompetencedelen lægges der vægt på at du kun skriver den nødvendige kode der løser opgaven, at du imødekommer opgavens krav præcist, samt at du udviser god kodeskik i forhold til navngivning, formatering og logik.

Bedømmelsesmetoden er som følger: hvert korrekt svar på et spørgsmål/opgave giver 2 point. Et delvist korrekt svar giver 1 point. Et ikke korrekt eller ikke besvaret spørgsmål/opgave giver 0 point. Det er beskrevet i forrige afsnit hvad kriteriet er for at en besvarelse vurderes som korrekt. Der indgår desuden følgende vægtning af de tre dele i testens samlede resultat: videndel: 25%, færdighedsdel: 25% og kompetencedel: 50%.

Dit Word dokument og din Visual Studio solution skal pakkes i en .zip, og sendes til bedømmelse.

27.1 Videndel

Du skal besvare følgende spørgsmål skriftligt i et word dokument:

1. Skriv hvordan man tilføjer flere projects til en solution.

2. Hvilken filendelse har C# filer? Er disse tekstfiler eller er de kompileret kode (binære filer)?

3. Gengiv syntaksen for erklæring af en variabel af en C# indbygget datatype, hhv. med og uden initialisering. Gengiv også syntaksen for overskrivning af en eksisterende værdi i en variabel.

4. Hvilken datatype ville du vælge til at repræsentere en persons navn?

5. Hvilken datatype ville du vælge til at repræsentere hvor mange varer en kunde har i sin indkøbskurv?

6. Skriv din forståelse af, hvad en variabel er.

7. Angiv værdien af variablen x: int a = 6; int b = 10; int x = (a-b)*(a+b)/2

8. Er dette korrekt syntaks inde i en Main metode? Hvis ikke så begrund dit svar. for(i = 0; i < 10; i++) { i += i – 1; }

9. Er dette korrekt syntaks inde i en Main metode? Hvis ikke så begrund dit svar. string correctInput = false; if(!correctInput) { Console.Write(correctInput + Du indtastede noget forkert.); }

10. Er dette korrekt syntaks inde i en Main metode? Hvis ikke så begrund dit svar. int i = Console.ReadLine(); Console.WriteLine(i);

11. Hvad er forskellen på kompileren og afviklingsmiljøet (a.k.a. runtime environment)?

27.2 Færdighedsdel

Hver opgave skal løses i sit eget konsol projekt:

1. Skriv et program hvor brugeren indtaster sit fornavn og efternavn separat. Brugeren skal få vist: ”Du hedder fornavn efternavn”.

2. Skriv et lille program der viser de første ti tal i 7 tabellen. Du skal benytte en for-løkke.

3. Skriv et lille program der går ud på at brugeren skal gætte et af programmet tilfældigt genereret tal mellem 1 og 10. Hvis man gætter forkert skal brugeren få vist en fejlbesked. Programmet skal afsluttes når brugeren har gættet rigtigt. Du skal benytte en while løkke.

4. Skriv et lille program hvor man fra Main metoden kalder en anden metode som har til ansvar at beregne additionen, subtraktionen, multiplikationen, divisionen og modulus af to tal som brugeren har indtastet. Disse resultater skal vises til brugeren på formen "regneart: resultat", med hver regneart på en ny linje.

5. Skriv et program der kan gemme hvad brugeren indtaster i en tekstfil. Der skal være en menu hvor brugeren kan vælge mellem at se alt indhold i tekstfilen, eller at skrive noget som skal gemmes i tekstfilen, eller afslutte programmet. Brugeren skal ved et af de første menuvalg, kunne vende tilbage til hovedmenuen. Der skal være tre metoder ud over Main metoden: DisplayMenu, DisplayTextFileContent og AddContentToTextFile. Du skal selv afgøre returtyper og parametre efter best practice.

27.3 Kompetencedel

Du skal anvende netop ét konsol projekt til at løse opgaven. Hvis du er blevet undervist i klasser og objekter, skal du anvende disse til at løse opgaven, ellers skal du løse opgaven uden.

Skriv et system til håndtering af film. En film består af titel, udgivelsesår, instruktør og filmselskab. Der skal være mindst to film i filen: Star Trek Beyond og Star Wars Rise of Skywalker.

Du skal opfylde følgende krav:

1. Brugeren skal præsenteres for en menu med valgmuligheder for at gemme en ny film, se alle film og søge på en films titel, samt at afslutte programmet.

2. Man skal kunne gemme ny-indtastede film i en tekstfil.

3. Man skal kunne se alle de film som er gemt i tekstfilen.

4. Man skal kunne søge på titel og få vist den eller de film der matcher søgekriteriet.

5. Du skal overholde god skik for navngivning og formatering.

6. Du skal opdele funktionaliteten i programmet efter princippet om Separation of Concerns.