Eksempel Repetition Sikker kode Opgaver Usikker kode

Windows Forms

Grundlæggende programmering

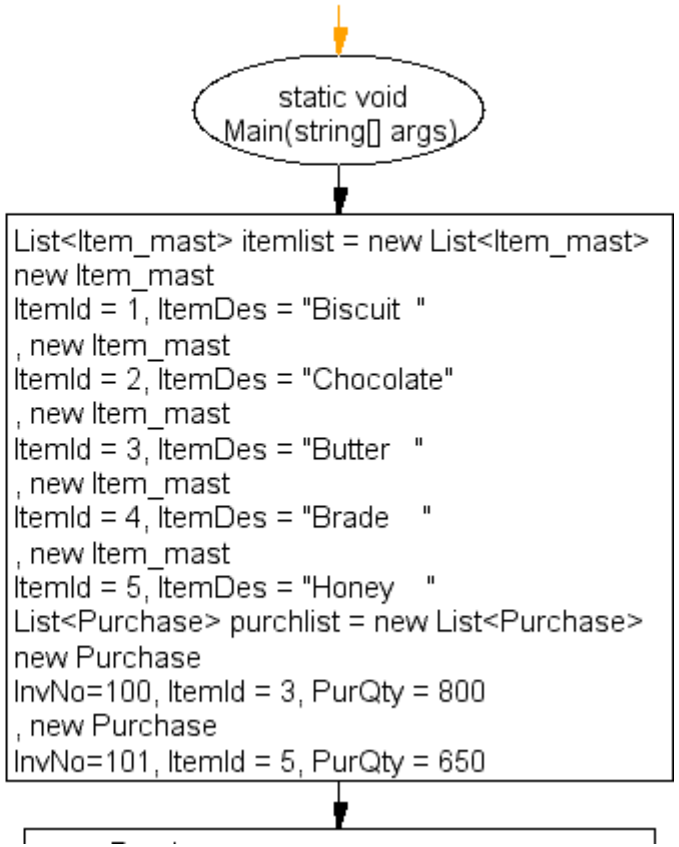
Lektion 7

Eksempel

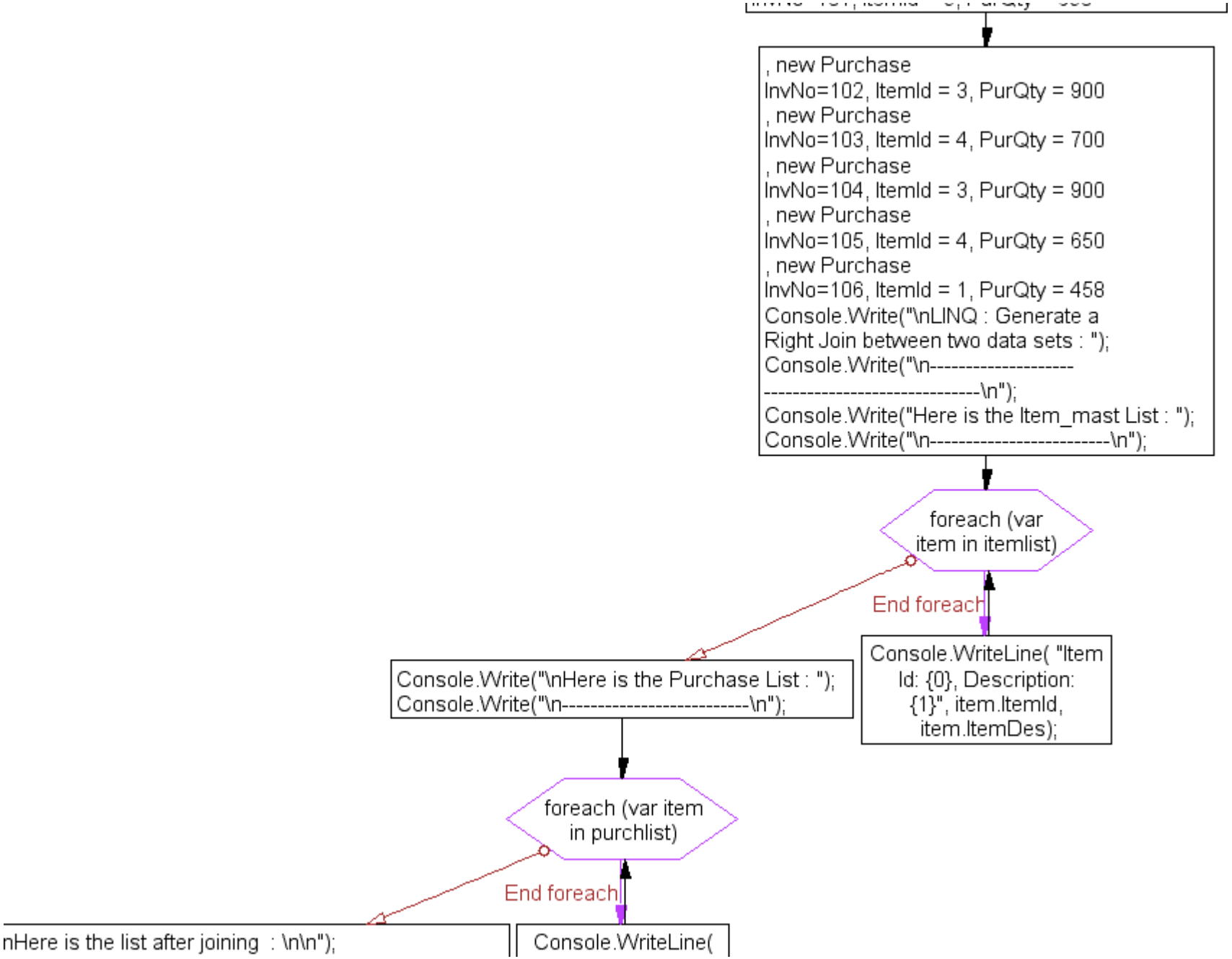
Hent lesson7example.cs fra Fronter

Eksempel

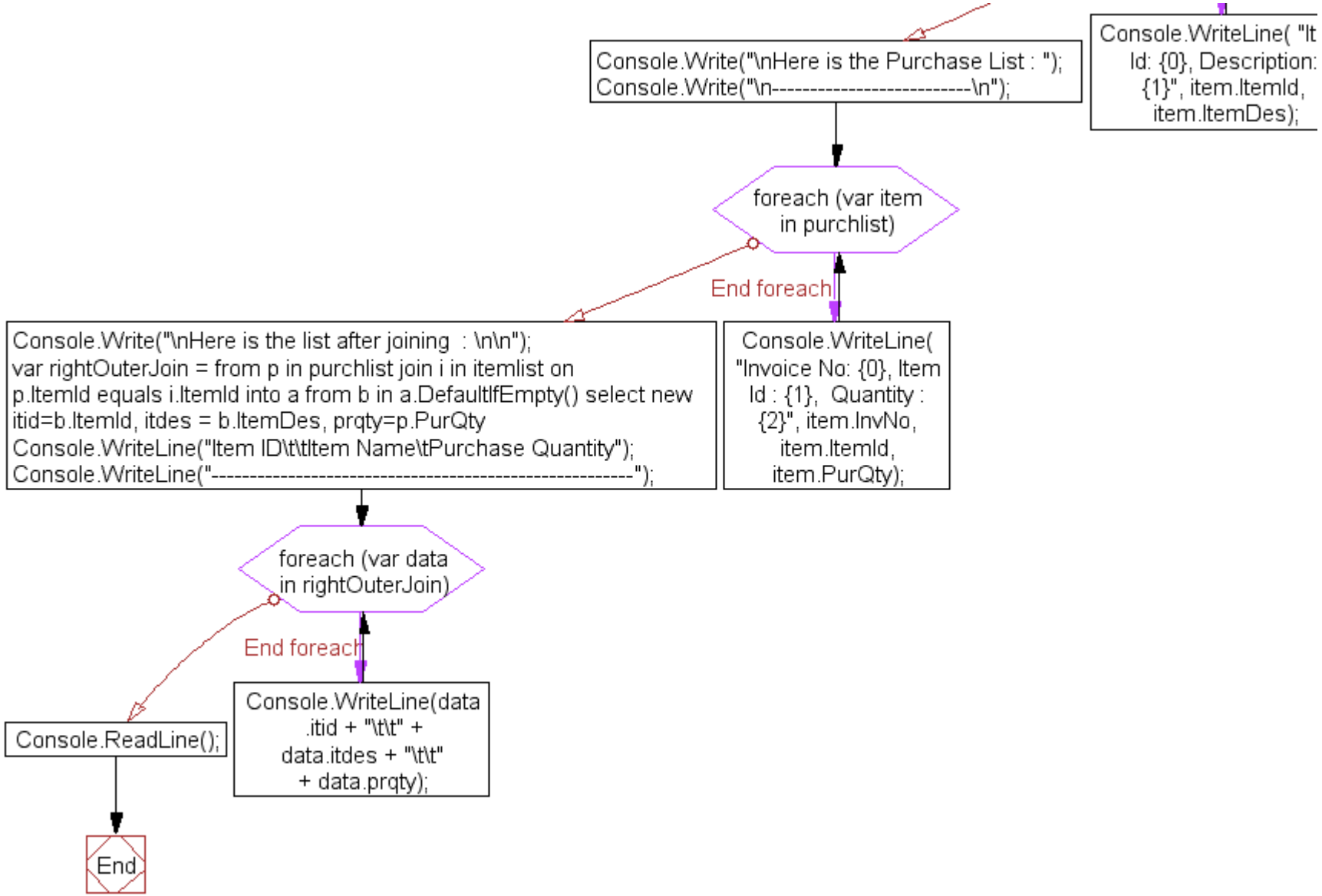
- Joins er en af de essentielle database operationer, hvor man kombinerer to tabeller.
  - Om man putter den ene tabel "foran" eller "bagved" den anden resulterer i left eller right joins.



Eksempel


, new Purchase
InvNo=102, ItemId = 3, PurQty = 900
, new Purchase
InvNo=103, ItemId = 4, PurQty = 700
, new Purchase
InvNo=104, ItemId = 3, PurQty = 900
, new Purchase
InvNo=105, ItemId = 4, PurQty = 650
, new Purchase
InvNo=106, ItemId = 1, PurQty = 458
Console.Write("\nLINQ : Generate a
Right Join between two data sets : ");
Console.Write("\n----------
----------\n");
Console.Write("Here is the Item_mast List : ");
Console.Write("\n----------\n");
foreach (var
item in itemlist)
End foreach
Console.WriteLine("Item
Id: {0}, Description:
{1}", item.ItemId,
item.ItemDes);
Console.Write("\nHere is the Purchase List : ");
Console.Write("\n----------\n");
foreach (var item
in purchlist)
End foreach
nHere is the list after joining : \n\n");
Console.WriteLine(

Eksempel


Console.Write("\nHere is the Purchase List : ");
Console.Write("\n----------\n");
Console.WriteLine("It
Id: {0}, Description:
{1}", item.ItemId,
item.ItemDes);
foreach (var item
in purchlist)
End foreach
Console.Write("\nHere is the list after joining : \n\n");
var rightOuterJoin = from p in purchlist join i in itemlist on
p.ItemId equals i.ItemId into a from b in a.DefaultIfEmpty() select new
itid=b.ItemId, itdes = b.ItemDes, prqty=p.PurQty
Console.WriteLine("Item ID\t\tItem Name\tPurchase Quantity");
Console.WriteLine("----------");
Console.WriteLine(
"Invoice No: {0}, Item
Id : {1}, Quantity :
{2}", item.InvNo,
item.ItemId,
item.PurQty);
foreach (var data
in rightOuterJoin)
End foreach
Console.WriteLine(data
.itid + "\t\t" +
data.itdes + "\t\t"
+ data.prqty);
Console.ReadLine();
End

Sikker kode

Hvordan kan ens kode kaldes sikker

Sikker kode

- Evidensbaseret sikkerhedspolitik og kode adgang sikkerhed sørger for meget kraftige, eksplicitte mekanismer til at implementere sikkerhed.
- Den meste programkode kan blot bruge infrastrukturen fra .NET Framework. I nogle tilfælde er yderligere applikationsspecifik sikkerhed, der kræves bygget enten ved at udvide sikkerheds-systemet eller ved hjælp af nye ad hoc-metoder, dog nødvendig.
- Ved brug af .NET Framework-håndhæves tilladelser og andre håndhævelser i din kode, skal du opføre barrierer for at forhindre skadelig kode i at opnå oplysninger, som du ikke ønsker, at den skal have eller udfører andre uønskede handlinger.
- Desuden skal du finde en balance mellem sikkerhed og brugervenlighed i alle de forventede scenarier med betroet kode.
- De næste slides vil gennemgå forskellige måder kode kan være konstrueret til at arbejde med sikkerhedssystemet på.

Sikker kode
Security-Neutral Code

- Security-neutral kode gør ikke noget eksplicit med sikkerheds-systemet.
- Den kører med de tilladelser den måtte modtage.
- Selvom applikationer, der undlader at fange sikkerheds-undtagelser forbundet med beskyttede operationer (såsom at bruge filer, netværk, og så videre) kan resultere i en ikke-afviklet undtagelse, vil sikkerheds-neutral kode stadig udnytte .NET Framework sikkerhedsteknologierne.

Sikker kode
Kode der ikke er genbrugelig

- Hvis din kode er en del af et program, der ikke bliver kaldt af anden kode, er sikkerhed enkel og speciel kodning er måske ikke være påkrævet.
- Husk dog at skadelig kode kan kalde din kode.
- Mens kode adgang sikkerhed kan stoppe skadelig kode i at få adgang til ressourcer, kan en sådan kode stadig læse værdier af dine fields eller egenskaber, der kan indeholde følsomme oplysninger.
- Desuden, hvis din kode accepterer brugerinput fra internettet eller andre upålidelige kilder, skal du være forsigtig med ondsindet input.

Sikker kode
Biblioteks kode, der udsætter beskyttede ressourcer

- Dette er den mest kraftfulde og dermed potentielt farlige (hvis det gøres forkert) tilgang til sikker kodning: Dit bibliotek fungerer som en grænseflade til anden kode for at få adgang til visse ressourcer, der ellers ikke tilgængelige, ligesom klasserne af .NET Framework håndhæver tilladelser for de ressourcer, de bruger.
- Uanset hvor du blotlægger en ressource, skal din kode først kræve hensigtsmæssig tilladelse til ressourcen (dvs., skal det foretage en sikkerhedskontrol) og derefter typisk udnytte sine rettigheder til at udføre selve operationen.

Usikker kode

Hvordan kan ens kode kaldes usikker

Usikker kode

- Hvis du vil ud over den sikkerhed som C# i sig selv tilbyder kan man gøre det på tre måder:
  - Platform Invoke (P/Invoke)
    - Går gennem APIer er blotlægges af ikke-administrerede DLLer.
  - Usikker kode
    - Tillader adgang til memory pointers og adresser.
  - Windows Runtime (WinRT) API
    - Kun tilgængelig i Windows 8 eller nyere.
    - Lader en se mere af operativsystemets funktioner.

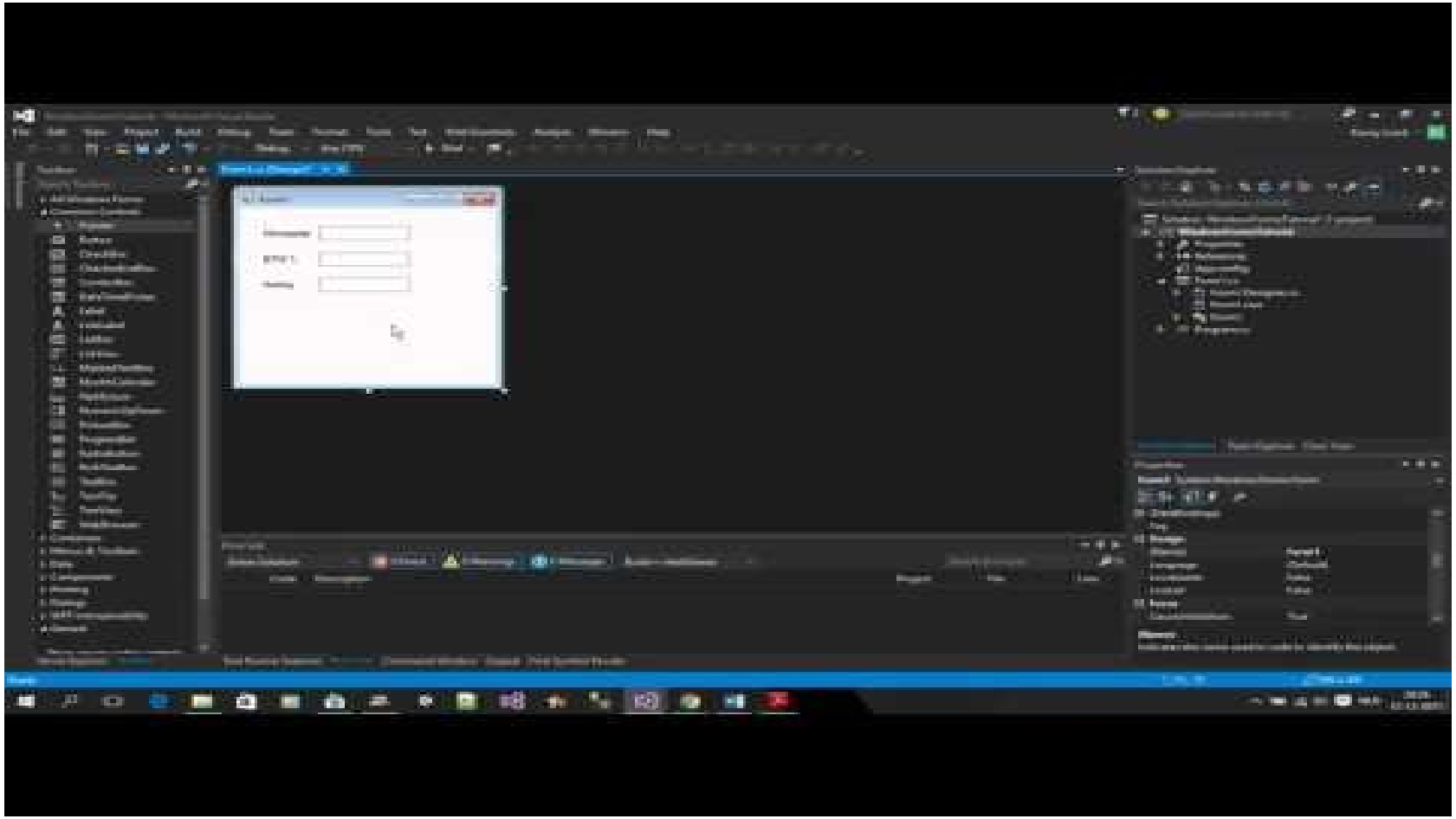
Usikker kode

- Hvis du vil ud over den sikkerhed som C# i sig selv tilbyder kan man gøre det på tre måder:
  - Platform Invoke (P/Invoke)
    - Går gennem APIer er blotlægges af ikke-administrerede DLLer.
  - Usikker kode
    - Tillader adgang til memory pointers og adresser.
  - Windows Runtime (WinRT) API
    - Kun tilgængelig i Windows 8 eller nyere.
    - Lader en se mere af operativsystemets funktioner.
- Ikke noget jeg vil anbefale i de fleste tilfælde, men det kan lade sig gøre.

Windows Forms

Grafiske programmer til Windows

Windows Forms



Repetition

Hvad vi har gennemgået i løbet af kurset

Repetition UML

Hvad er UML

- **UML er et sprog**
- Regler for, hvordan elementer er sat sammen
- Regler for organisationen
- UML vise hvordan elementer forholder sig til hinanden
- Kan både anvendes i software værktøjer, på white boards og på papir

Der er grundlæggende **to slags diagrammer**

- Adfærds diagrammer
  - Krav, drift, indre tilstande
- Struktur diagrammer
  - Fysisk organisation

Repetition UML

Adfærds diagrammer

- Use case diagram
  - Funktionelle krav til et system
  - Hvad et system skal gøre
  - Gør det muligt for den der laver modellen at fokusere på brugerens behov snarere end detaljer i produktionen
- Aktivitet diagram
  - Vis strømmen fra en adfærd eller aktivitet, til den næste
  - Med udtryksfuld end en klassisk flowchart

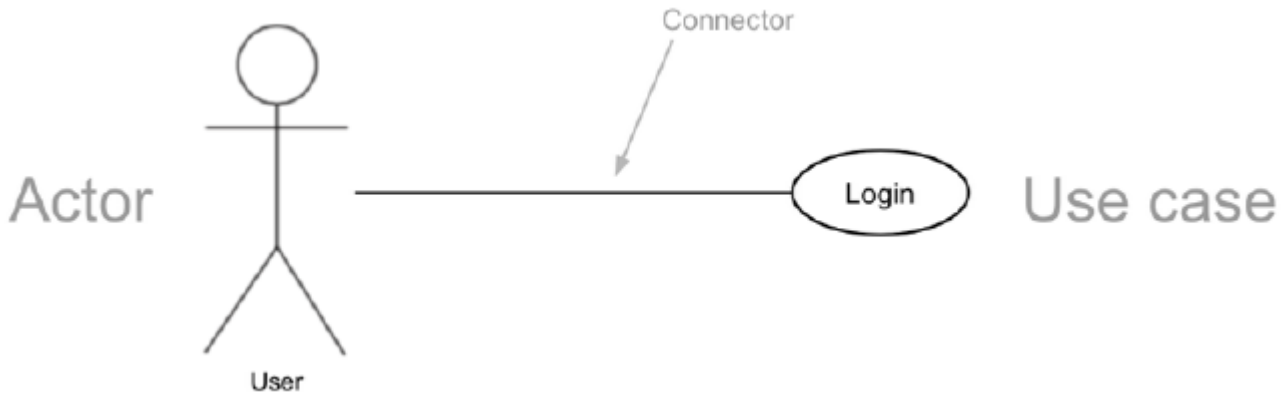
Struktur diagrammer

- Klasse diagrammer
  - Viser enheder i et system og forholdet mellem dem
  - Kan være detaljeret og generere kildekode eller simple skitser

Repetition
UML
Adfærds diagram

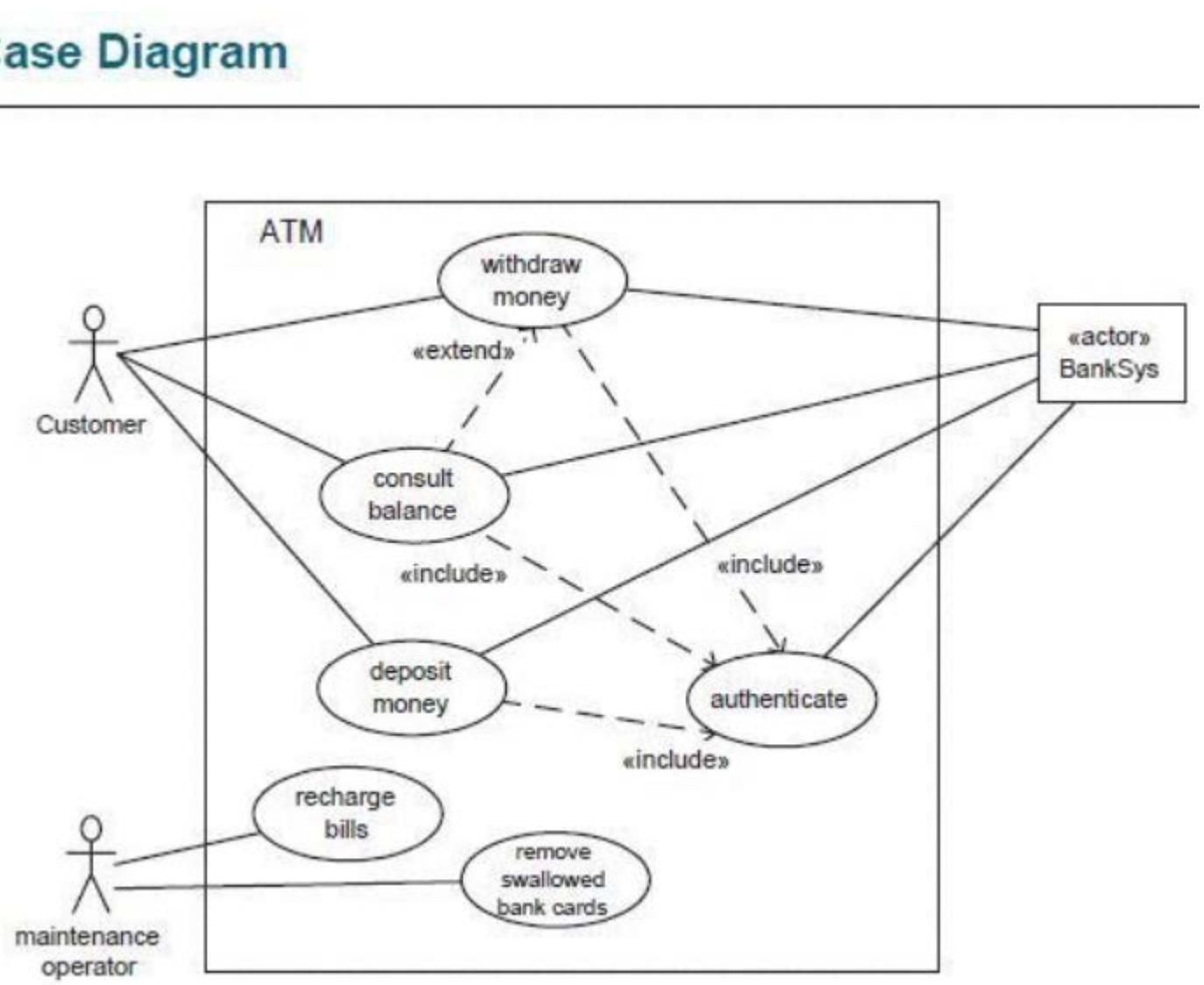
Use case diagrammer

- Grafisk oversigt over en eller flere aktørers involvering i et system.



Repetition UML

Adfærds diagram



Repetition UML
Adfærds diagram

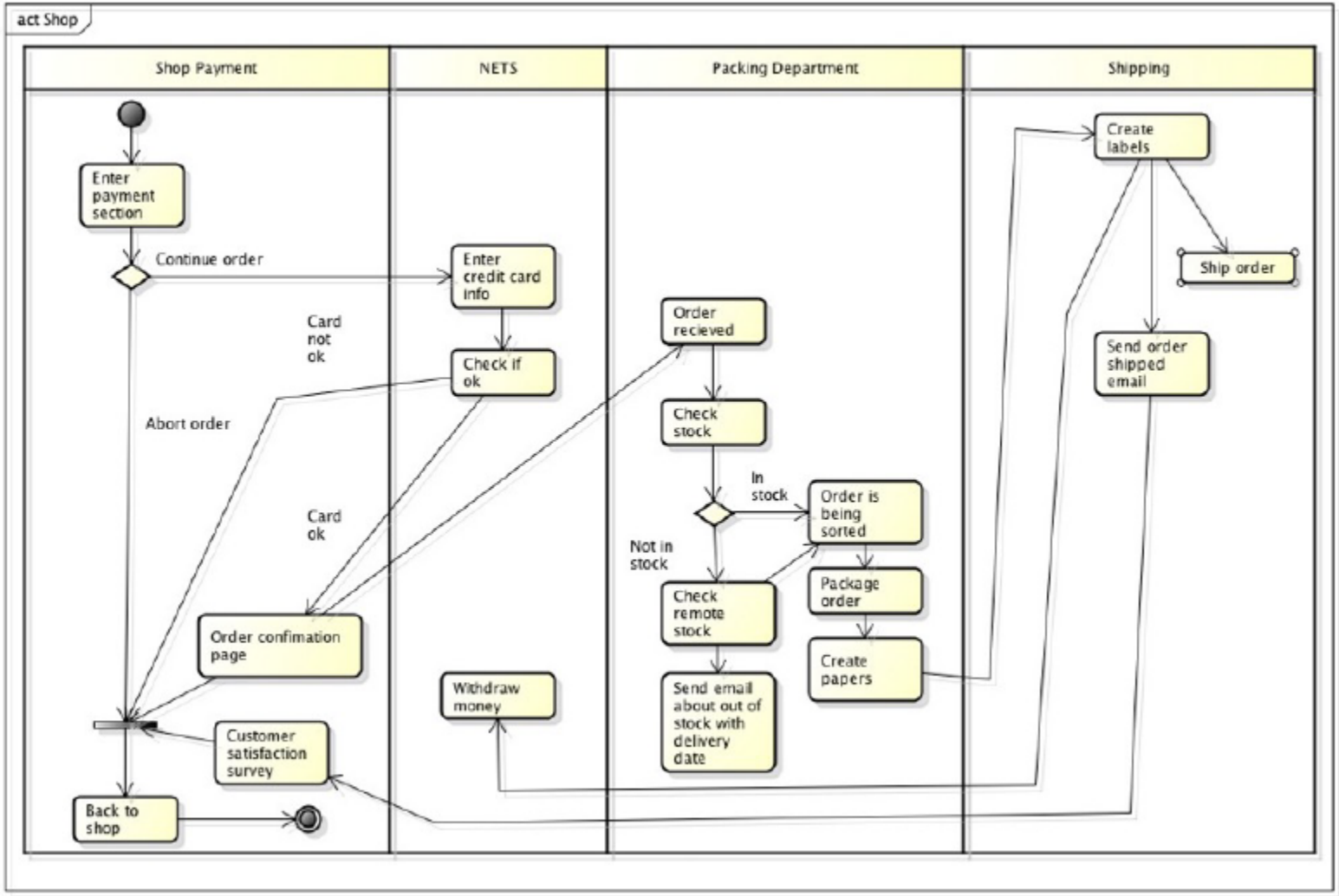
Aktivitets diagrammer

- Opdelt i opgaver af lodrette "kasser" ved siden af hinanden
- Afrundede rektangler = handlinger
- Diamanter = beslutninger
- Barer = splitter eller sammenføjede aktiviteter
- Sort cirkel = start workflow (oprindelige tilstand)
- Omkranset sort cirkel = ende af flow (endelige tilstand)

Repetition UML

Adfærds diagram

Aktivitets diagrammer



Repetition
UML
Struktur diagram

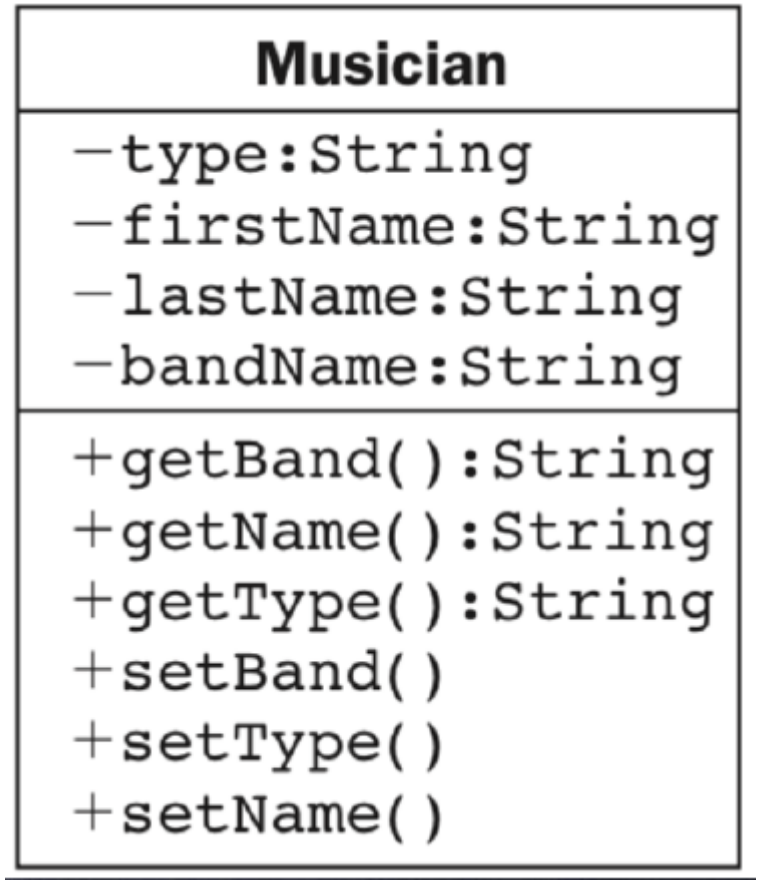
Klasse diagrammer

- En klasse repræsenterer en gruppe af ting, der har fælles tilstand og adfærd
- En klasse er repræsenteret af en rektangulær kasse opdelt i rum
  - Første rum indeholder navnet på klassen
  - Det andet er attributter
  - Det tredje er operationer
- Et klasse diagram bruges især til at udlægge variabler og

Repetition
UML
Struktur diagram

Klasse diagrammer

- Klasse navn
- Egenskaber
  - (+ / - / #)
    - offentlig / privat / beskyttet
  - :type
- Operationer ()
  - (+ / - / #)
    - offentlig / privat / beskyttet
  - :return



Repetition Grundlæggende begreber

Indhold i et C# program

Et C # program består af følgende dele:

- Namespace deklaration
- En klasse
- Klasse metoder
- Klasse attributter
- En Main metode
- Statements og Expressions
- Kommentarer

Repetition Grundlæggende begreber

Indhold i et C# program

Noter omkring C#

- C # er case sensitive.
- Alle statements og udtryk skal afsluttes med et semikolon (;).
- Udførelsen af programmet starter ved Main metoden.
- I modsætning til Java må programmets filnavn være forskelligt fra klasse navnet.

Repetition
Data typer

Variablerne i C# er inddelt i tre typer

- Værdi typer / Value types
- Reference typer / Reference types
- Markør typer / Pointer types

Repetition
Data typer
Værdi typer

Value types variabler kan direkte få en værdi. De er afledt af klassen class **System.ValueType**.

Det kan være data som i**nt, char** og **float**, der gemmer tal, bogstaver og komma tal respektivt.

Når man deklarerer en **int** type allokerer systemet automatisk hukommelse til at gemme værdien

Repetition
Data typer
Reference typer

En reference type indeholder ikke faktisk data gemt i en variabel men indeholder en reference til variablerne.

De refererer altså til en plads i hukommelsen. Således kan flere variabler af reference typerne referere til sammes plads i hukommelsen. Hvis dataen på denne plads i hukommelsen ændres af en af variablerne vil de andre automatisk ændre indhold.
Et eksempel på **built-in** reference typer er: **object**, **dynamic** og **string**.

Repetition
Data typer
Markør typer

Markør/pointer type variabler gemmer hukommelses adressen for en anden type. Pointers i C# har de samme muligheder som pointers i C eller C++.

```
Int *myVariable;
```

Udtrykket *myVariable henviser til int variablen der findes på adressen indeholdt i myVariable.

Pointers kan dog let lede til usikker kode som nævnt tidligere.

Repetition Operatorer

Operatorer bruges til at udføre matematiske eller logiske operationer på værdier
(eller variabler) kaldet operander for at producere en ny værdi, kaldet resultatet.

```
int difference = 4 – 2;
```

Her er – operatoren og resultatet 2 gemmes i integer data typen difference.

Operatorer er generelt inddelt i tre kategorier

- Unary
- Binary
- Tertiary

svarende til antallet af operander (en, to, og tre henholdsvis).

Repetition Operatorer Unære

Plus og minus unære operatorer (+, -)

Hvis man har brug for at ændre en værdi fra positiv til negativ (eller omvendt) kan C# gøre det med den unære operator –

Den undære operator + har sjældent en effekt og er mest med for god ordens skyld.

Repetition Operatorer Binære

Aritmetiske binære operatorer (+, -, *, /, %)

I eksemplet bliver divisionen og resten operationerne gennemført før opgaverne. Den rækkefølge, som de operatorerne udføres ier bestemt af deres forrang (precedence) og associativitet (associavity).

Rangen for operatørerne hidtil anvendt er_

1. *, / og % har højeste prioritet.
2. + og - har lavere prioritet.
3. = har den laveste rang af disse seks operatører.

Man kan også benytte + til at sætte andet end tal sammen, f.eks. flere strenge som vi har gjort tidligere.

Repetition Operatorer Binære

Compound Assignment Operators (+=, -=, *=, /=, %=)

Compound assignment operators kombinerer standard binære operator beregninger med en assignment operator.

Således giver

```
int x = 123;
x = x + 2;
```

og

```
int x = 123;
x += 2;
```

det samme.

Der findes talrige andre ”compund assignment" operatører til at give lignende funktionalitet. Du kan således også bruge assignment operatoren sammen med subtraktion, multiplikation, division, og resten (remainder) operatører.

Repetition
Operatorer
Flow kontrol

Flow kontrol

Selv i simple programmer kan det være nødvendigt at styre rækkefølgen af udførelse af programmet, ikke blot fra start til slut

Al C# kode I har set hidtil har haft én ting til fælles. I hvert tilfælde er programmet udført fra den ene linje til den næste fra top til bund i rækkefølge, intet mangler. Hvis alle programmer arbejdede som dette, så ville man være meget begrænset i hvad man kunne gøre.

For at opnå dette benytter man forgrening og looping.
Forgrening udfører kode betinget, afhængigt af resultatet af en evaluering, som "only execute this code if the variable myVal is less than 10."
Looping udfører gentagne gange de samme udsagn, enten et bestemt antal gange, eller indtil en test tilstand har været nået.

Repetition
Operatorer
Flow kontrol

if statements

Et if statement er et af de mest udbredte i C#. Det evaluerer et **boolsk udtryk** (**Boolean expression**), et udtryk der resulterer i enten sandt eller) der kaldes **condition**.
Hvis tilstanden er sand så udføres **consequence statement**.

Et if statement kan eventuelt have en else clause der indeholder et **alternativt statement** der udføres hvis condition er falsk.

```
if (condition)
consequence-statement
else
alternative-statement
```

Repetition Løkker

Løkker

- En løkke er når en blok kode køres flere gange.
- Termen loop body henviser til et statement, typisk en kode blok, der køres i et loop indtil afslutnings-kravet er mødt.

Forskellige former for løkker

- Man bruger while til at gentage (iterate) så længe kravet er true.
- En for løkke bruges mest hensigtsmæssigt, når antallet af gentagelser er kendt, såsom når der tælles fra o til n.
- En do/while ligner en while-løkke, men den vil altid udføre løkken kroppen mindst én gang.

Repetition
Løkker
While

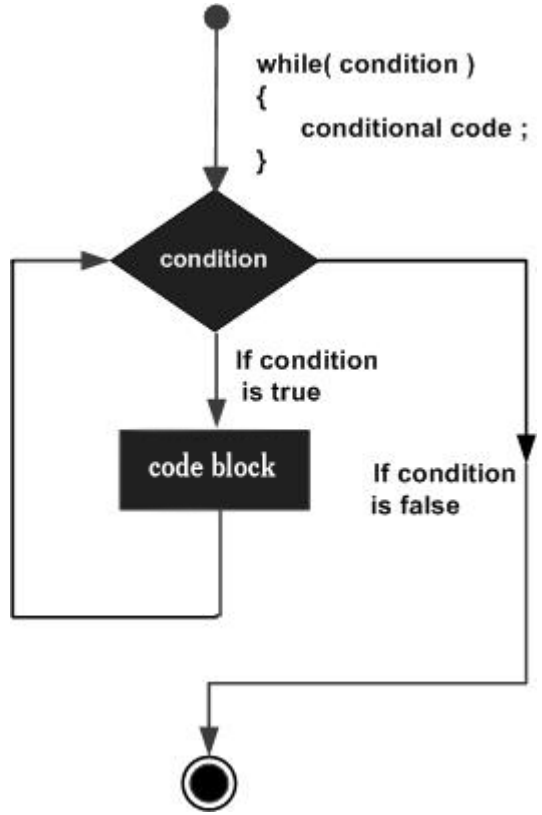
While løkker

While løkken er det enkleste betingede loop. Den almindelige form af while-sætningen er:

while (*condition*)

statement

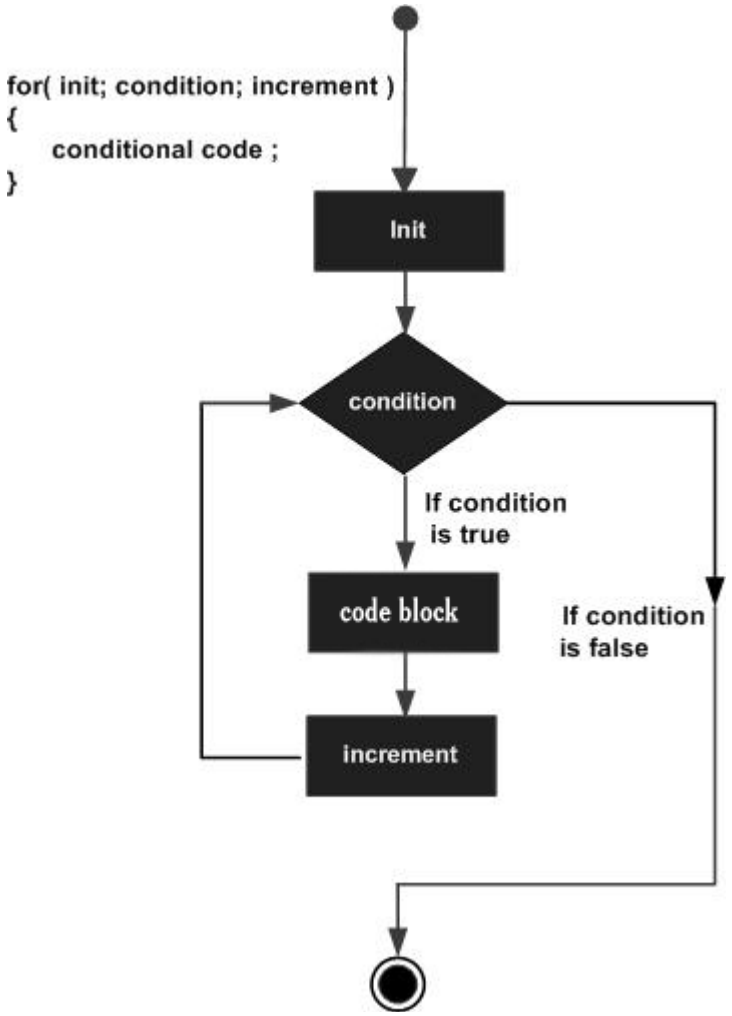
Loopet kører det statement, der danner kroppen i udtrykket, rundt så længe at conditionen (der *skal* være boolean) er true. Hvis den bliver *false* dropper udførelsen af koden kroppen og går videre til koden efter loop statementet.



Repetition
Løkker
For

For løkker

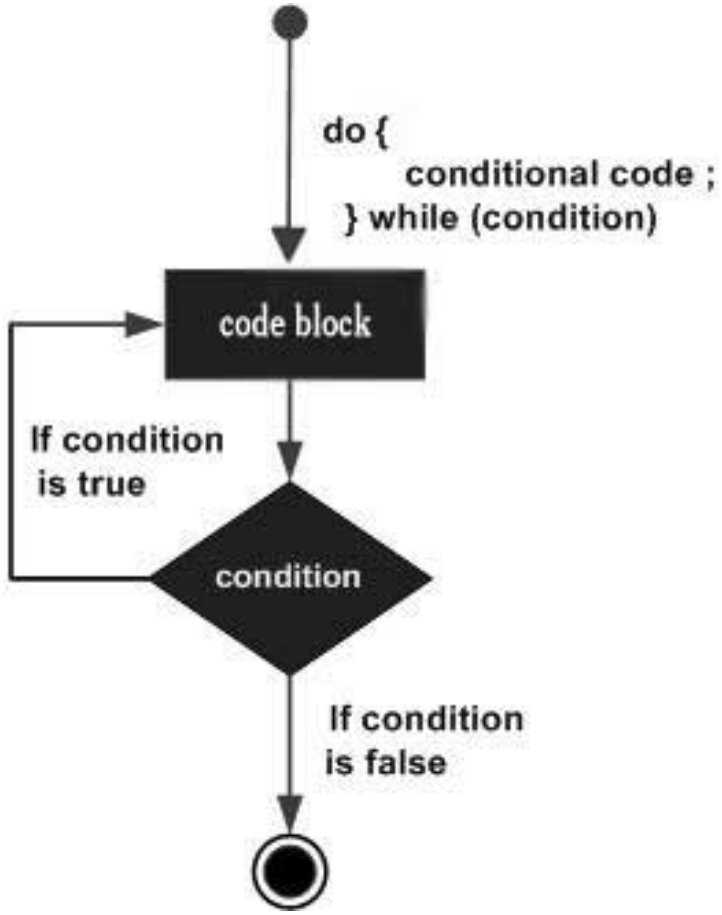
- En for-løkke gentager en kode blok indtil en bestemt betingelse er nået.
- I forhold til while løkken har for-løkken indbygget syntaks for initialisering, forøgelse og afprøvning af værdien af en tæller, kaldet loop variablen.
- Fordi der er en specifik placering i loopets syntaks for en tilvækst operation, anvendes increment og decrement operatører ofte som en del af en for-løkke.



Repetition
Løkker
Do… while

Do… while løkker

- Do / while-løkken er meget lig while løkken, bortset fra at do / while løkken foretrækkes, når antallet af gentagelser er fra 1 til n, og n er ikke er kendt når iterationen begynder.
  - Dette opstår ofte når man spørger en bruger efter input.



Repetition Metoder

- En metode er en måde at samle en sekvens af statements der udfører en bestemt handling eller beregner et bestemt resultat.
  - Dette giver større struktur og organisation for de statements, der opbygger et program.
- Hvert C # program har mindst en klasse med en metode kaldet Main.
- For at benytte en metode skal man:
  - Definere metoden.
  - Kalde metoden.
- Vi har brugt metoder hidtil, primært main() metoden, hvori hele vores programmer lå, men med mere komplekse programmer kan det være praktisk eller nødvendigt at benytte flere metoder.

Repetition Metoder

- En metodes grund struktur er

```
<Access Specifier> <Return Type> <Method Name>(Parameter List)
{
  Method Body
}
```

- **Access Specifier** (scope): Bestemmer synligheden af en variabel eller en metode for andre klasser.
- **Return type**: En metode kan returnere en værdi. Hvis metoden ikke returnerer nogen værdier, så er retur typen ugyldig (void).
- **Method name**: Metode navn er et entydigt id, og det er følsomt overfor store og små bogstaver. Det kan ikke være det samme som en anden identifikator erklæret i klassen.
- **Parameter list** (parametre og argumenter): Lukket inde mellem parenteser finder man de parametre, der anvendes til at sende og modtage data fra metoden. Parameterlisten refererer til typen, rækkefølgen og antallet af parametre i metoden. Parametre er valgfri, det vil sige at en metode ikke behøves have nogen parametre.
- **Method body**: Metodens krop indeholder det sæt af instruktioner, der er nødvendige for at gennemføre den ønskede aktivitet.

Repetition Klasser

Klasser og objekter og objekter lavet ud fra dem er hvad der gør C# til et objekt orienteret programmerings sprog.

- Når man definerer en klasse definere en man en plan for en datatype.
  - Dette definerer faktisk ikke nogen data, men det definerer, hvad klasse navnet betyder. Det vil sige, hvilket objekt klassen består af og hvilke operationer der kan udføres på det pågældende objekt.
  - Objekter er instanser af en klasse. De metoder og variabler, der udgør en klasse, kaldes medlemmer af klassen.
- En klasse er en skabelon for hvordan et objekt vil se ud på instantiering tidspunktet. Et objekt er derfor en instans af en klasse.
  - Det at lave et objekt fra en klasse kaldes derfor instantiation.

Repetition
Klasser
Access modifiers

Yderligere indkapsling

Access modifiers identificerer niveauet for indkapsling forbundet med medlemmet (member) de hører til.

De findes fem access modifiers:

- Public
- Protected
- Internal
- Protected internal
- Private

Repetition
Klasser
Access modifiers

Public

Hvis et member eller en type har public access modifier har det det mest eftergivende adgangsniveau. Der er ingen restriktioner på adgang til public members.

Protected

En beskyttet medlem er tilgængeligt inden for sin klasse og ved afledte klasse instanser.

Internal

Adgangen er begrænset til det aktuelle assembly.
Internal er default hvis der ikke er angivet nogen access modifier.

Repetition
Klasser
Access modifiers

Protected internal

Adgangen er begrænset til det aktuelle assembly eller typer afledt af den indeholdende klasse.

Private

Private access er det mindst eftergivende adgangsniveau. Private medlemmer er kun tilgængelige i kroppen af klassen eller struct, hvor de er blevet erklæret.

Repetition
Klasser
Structs

Structs er defineret af struct nøgleordet, for eksempel:

```
public struct PostalAddress
{
   // Fields, properties, methods and events go here...
}
```

Structs deler det meste af syntaks med klasser, selv om structs er mere begrænset end klasser:

- Inden for en struct erklæring, kan felterne ikke initialiseres, medmindre de angives som const eller static.
- En struct kan ikke erklære en standard-constructor (en konstruktør uden parametre) eller en destructor.
- Structs kopieres ved assignment. Når en struct tildeles en ny variabel bliver al data kopieret, og enhver ændring til den nye kopi ændrer ikke data for den originale kopi. Det er vigtigt at huske, når du arbejder med samlinger af værdi typer såsom Dictionary <string, myStruct>.
- Structs er værdi typer og klasser er referencetyper.

Repetition
Klasser
Structs

- I modsætning til klasser kan structs instantieres uden brug af en ny operator.
- Structs kan erklære konstruktører, der har parametre.
- En struct kan ikke arve fra en anden struct eller klasse, og den kan ikke danne basis for en klasse. Alle structs arver direkte fra System.ValueType, som arver fra System.Object.
- En struct kan implementere grænseflader.
- En struct kan bruges som en nullable type og kan tildeles en null-værdi.

Repetition
Arv

En af de vigtigste begreber i objektorienteret programmering er arv. Arv giver os mulighed for at definere en klasse i form af en anden klasse, hvilket gør det lettere at oprette og vedligeholde en applikation. Dette giver også mulighed for at genbruge kodens funktionalitet og fremskynder dermed implementeringstid.

Når man opretter en klasse, i stedet for at skrive helt nye data medlemmer og medlems funktioner, kan man udpege, at den nye klasse skal arve medlemmerne fra en eksisterende klasse. Denne eksisterende klasse kaldes base klasse eller forælder, og den nye klasse betegnes som den afledte klasse eller barn.

Ideen om arv implementerer IS-A relationer. F.eks er pattedyret IS-A dyr, hund IS-A pattedyr dermed er hund IS-A dyr også, og så videre.

Repetition
Arv
Polymorfisme

Repetition Interfaces

- Hvor klasser definerer *hvordan* noget skal udføres sørger interfaces for *hvad* der skal udføres.
- Interfaces definerer properties, methods og events, der er medlemmer af interfacet.
- Interfaces indeholder kun deklarationen af medlemmerne. Det er den afledte klasses ansvar at definere medlemmerne.
- Interfacet hjælper ofte med at give en standard struktur, som de afledte klasser vil følge.
- Abstrakte klasser har til en vis grad samme formål, men de bliver primært brugt når der kun skal erklæres få metoder af base-klassen og de underliggende klasser implementerer funktionerne.

Repetition Interfaces

Repetition Enums

- En enumeration er et sæt navngivne integer konstanter.
- En enumerated type declares med enum keyword.
- C# enumerations er en værdi data type. Enumeration indeholder altså dens egne værdier og kan ikke arve eller videresende arv.
- Enum declares således

```
enum <enum_name>
{
   enumeration list
};
```

  - *enum_name* angiver enumeration typens navn.
  - *enumeration list* er en komma-separeret liste af identifiers.

- Hver af symbolerne i enumeration listen står for en heltalsværdi, en større end symbolet der er angivet foran. Som standard er værdien af det første enumeration symbol o.

```
enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };
```

Repetition
Exception handling

- En undtagelse (exception) er et problem, der opstår under udførelsen af et program.
- I C# er en undtagelse er en reaktion på en usædvanlig hændelse, der opstår, mens et program kører, f.eks. et forsøg på at dividere med nul.
- Exceptions sørger for en måde at transfer kontrol fra en del af et program til en anden.
- C# exception handling bygger på fire keywords: **try**, **catch**, **finally** og **throw**.
  - **try**: En try blok identificerer en blok af kode, for hvilken særlige undtagelser er aktiveret. Det efterfølges af en eller flere catch blokke.
  - **catch**: Et program fanger en undtagelse med en exception handler det sted i programmet, hvor du ønsker at håndtere problemet. Catch keyword angiver fangst af en undtagelse.
  - **finally**: Finally blokken anvendes til at udføre et givet sæt udsagn, om en undtagelse er thrown eller ikke thrown. For eksempel, hvis du åbner en fil, skal det være lukket om en undtagelse hæves eller ej.
  - **throw**: Et program kaster en undtagelse, når et problem dukker op. Dette gøres ved hjælp af throw keyword.

Repetition
Exception handling
Syntaks

Syntaks

- Hvis en blok udløser en exception fanger en metode en exception gennem en kombination af try og catch keywords.
- En try/catch blok placeres omkring koden, der kan generere en exception. Kode i en try/catch omtales som protected (beskyttet) kode.
- Du kan liste flere catch statements til at fange forskellige typer af undtagelser i tilfælde af at en blok trigger mere end en enkelt undtagelse i forskellige situationer.

```
try
{
   // statements causing exception
}
catch( ExceptionName e1 )
{
   // error handling code
}
catch( ExceptionName e2 )
{
   // error handling code
}
catch( ExceptionName eN )
{
   // error handling code
}
finally
{
   // statements to be executed
}
```

Repetition Generics

- Generics gør det muligt at forsinke specifikation af datatypen for program elementer i en klasse eller en metode, indtil de faktisk bliver brugt i programmet.
  - Med generics kan man altså skrive en klasse eller metode, der kan arbejde med alle datatyper.
- Man skriver specifikationerne for klassen eller den metode, med alternative parametre for datatyper.
  - Når compileren møder en constructor for klassen eller et funktionskald for metoden, genererer den kode til at håndtere den specifikke datatype.

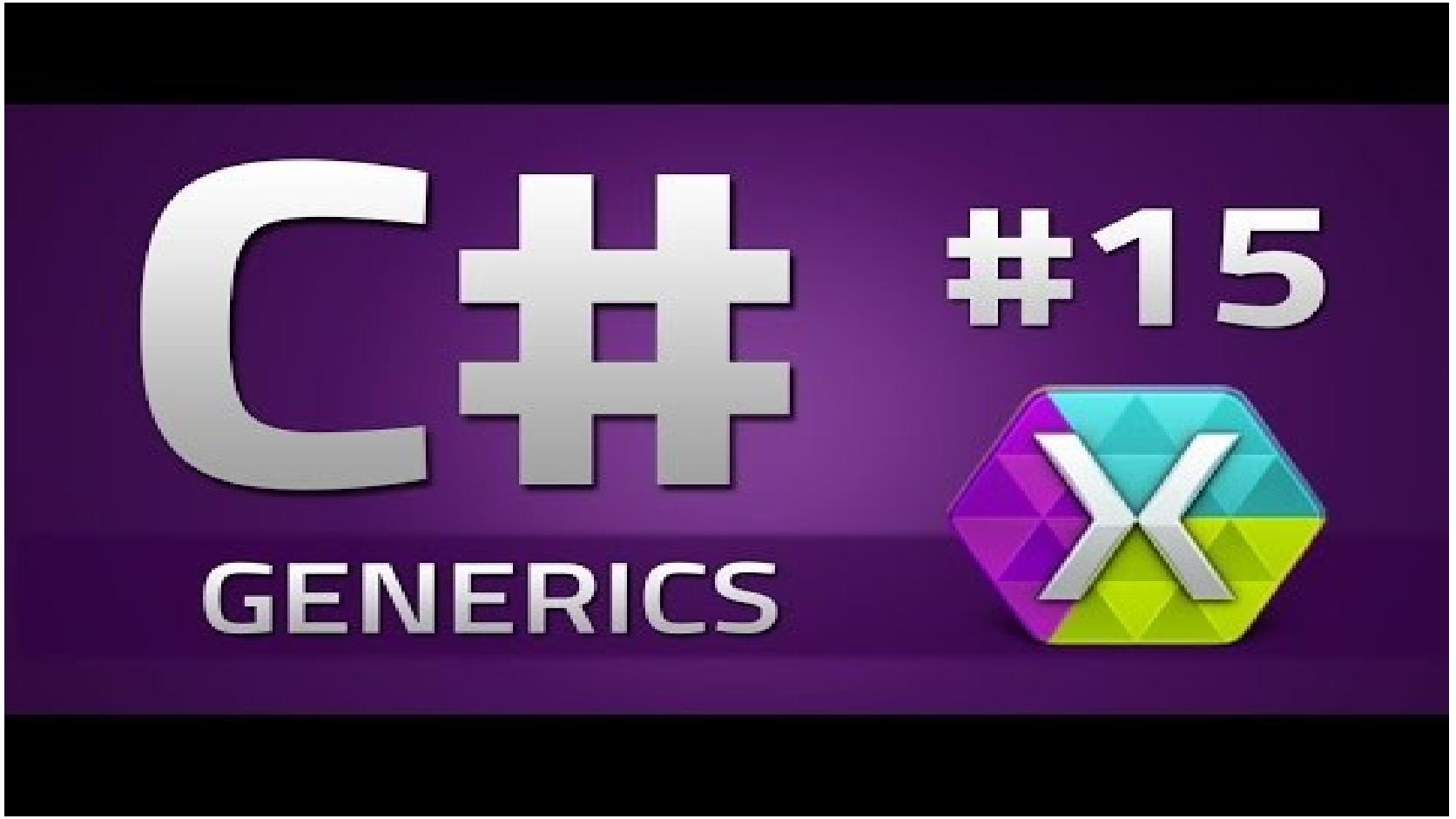
Repetition
Generics
Egenskaber

Generics egenskaber

Generics kan forbedre ens programmer på følgende måder:

- De hjælper med at maksimere genbrug af kode, type sikkerhed og ydeevne.
- Du kan oprette generic collection klasser.
  - .NET Framework klasse bibliotek indeholder en række nye generic collection classes i System.Collections.Generic namespace. Man kan bruge disse generic collection classes i stedet for collection classes i System.Collections namespace.
- Man kan lave egne generic interfaces, classes, methods, events og delegates.
- Man kan lave generic classes tvunget til at give adgang til metoder for bestemte datatyper.
- Man kan få information om de typer, der anvendes i en generic datatype ved run-time ved hjælp af reflection.

Repetition Generics



Repetition
Lambda expressions

- Et lambda udtryk er en anonym funktion, som kan bruges til at oprette delegates eller udtryk træ typer.
- Ved at bruge lambda udtryk kan man skrive lokale funktioner, der kan overføres som argumenter eller returneres som værdien af funktionskald.
- Lambda udtryk er særligt nyttigt for at skrive LINQ query udtryk.
- For at oprette et lambda udtryk angiver man inputparametre (hvis nogen) på venstre side af lambda operatoren =>, og man sætter udtryk eller udsagn blokken på den anden side
  - Lambda udtrykket `x => x * x` angiver en parameter, x og giver den værdien af x potens. Du kan tildele dette udtryk til en delegeret type.

Repetition
Lambda expressions

Getting Started with

Lambda Expressions

//c0deporn;

Repetition Events

- Events er brugerhandlinger såsom tastetryk, klik, musebevægelser, etc., eller nogle hændelser såsom system genererede meddelelser.
- Applikationer er nødt til at reagere på begivenheder, når de opstår. For eksempel interrupts.
- Events anvendes til inter-proces kommunikation.
- Begivenhederne erklæres og udføres i en klasse og forbindes med event handlers der bruger delegates inden for samme klasse eller en anden klasse.
- Klassen, der indeholder eventet, bruges til at udgive eventet.
  - Dette kaldes udgiver (publisher) klassen.
  - Andre klasser, der accepterer eventet, kaldes abonnent (subscriber) klassen.
- Derfor kaldes det at begivenheder bruger en udgiver-abonnent model.

Repetition
Events
Delegates

Repetition Filer

- En fil er en samling gemt på en disk med et specifikt navn og mappe sti. Når filen åbnes til læsning eller skrivning bliver den en stream.
- En stream er grundlæggende sekvensen af bytes der går gennem kommunikations stien. Der er to hoved streams: **input stream** og the output stream.
  - Input stream bruges til at læse data fra filen (read operation)
  - og output stream bruges til at skrive til filen (write operation).

Repetition
Fil systemet
FileStream klassen

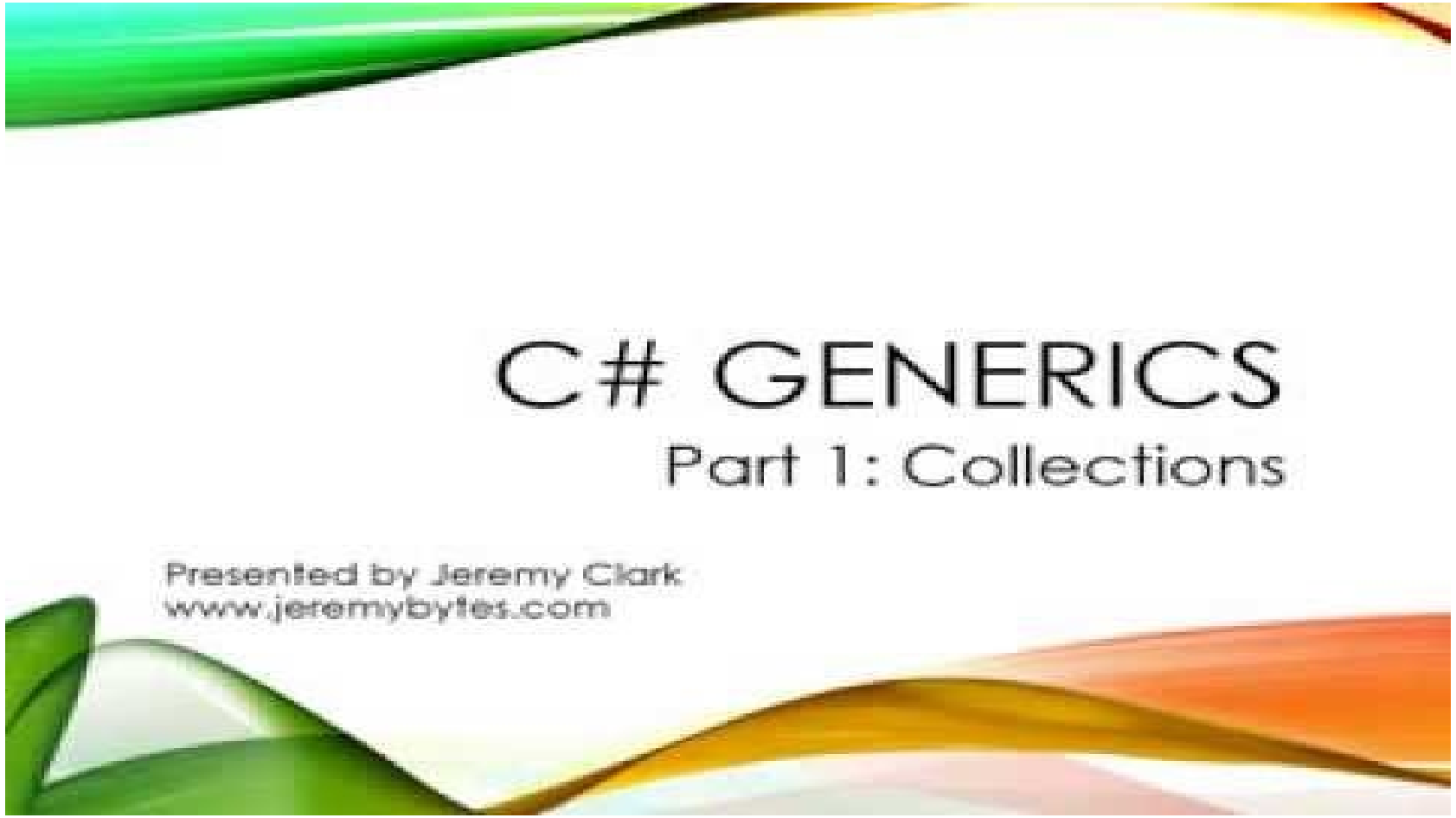
- FileStream klassen i System.IO namespace hjælper med at skrive fra, skrive til og lukke filer. Klassen er afledt af den abstrakte klasse Stream.
- For at lave en ny fil eller åbne en eksisterende fil skal man have et **FileStream** objekt.

```
FileStream <object_name> = new FileStream( <file_name>,
<FileMode Enumerator>, <FileAccess Enumerator>,
<FileShare Enumerator>);
```

- Her laves et FileStream objekt F til at læse filen sample.txt

```
FileStream F = new FileStream("sample.txt", FileMode.Open,
FileAccess.Read, FileShare.Read);
```

Repetition Collections



Repetition LINQ

- Akronymet LINQ star for Language INtegrated Query.
- Gennem LINQ queries får man let data adgang til in-memory objekter, databaser, XML documenter og mere til.
- LINQ blev indført i Visual Studio 2008 og er designet af Anders Hejlsberg.
- LINQ tillader en at skrive queries selv uden kendskab til query sprog som SQL, XML osv.
- LINQ queries kan skrives til forskellige datatyper.
- Der er to syntakser I LINQ.
  - Lamda (Method) Syntax

```
var longWords = words.Where( w => w.length > 10);
```

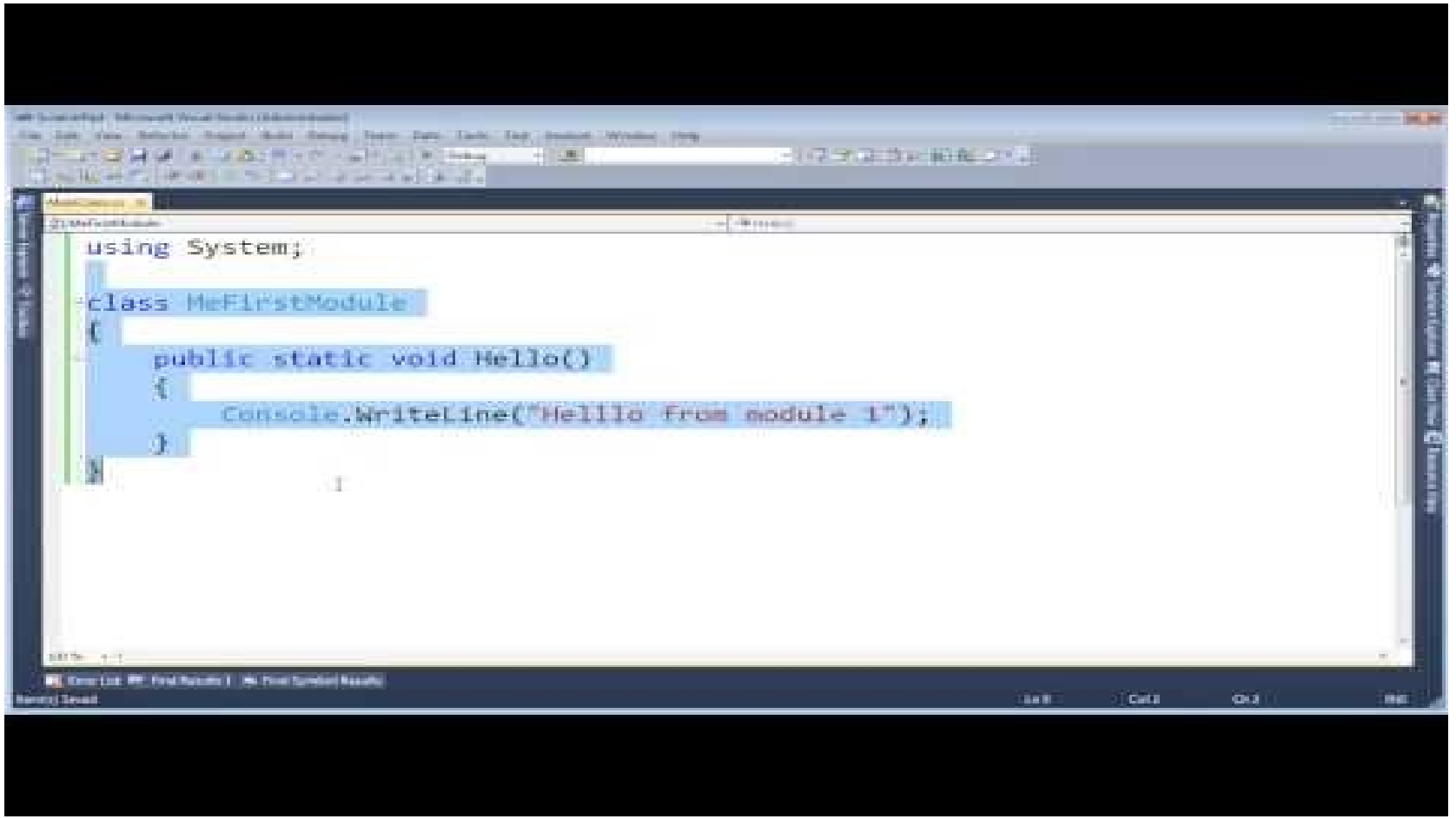
  - Query (Comprehension) Syntax

```
var longwords = from w in words where w.length > 10;
```

Repetition Assemblies

- Et assembly er en fil, der er automatisk genereres af compileren efter en vellykket samling af hver NET applikation.
- Det kan enten være et Dynamic Link Library eller en eksekverbar fil.
- Det genereres kun en gang for et program og ved hver efterfølgende kompilering bliver assembly opdateret.
- Hele processen vil køre i baggrunden af ens applikation.
- Et assembly indeholder Intermediate Language (IL) kode, som svarer til Java byte kode.
- I .NET sproget består det af metadata. Metadata enumerates features i hver "type" inde i assembliet eller binary.

Repetition Assemblies



Repetition Attributes

- En attribut er en deklarativ tag, der bruges til at formidle information til runtime om adfærd af forskellige elementer som klasser, metoder, strukturer, enumerators, samlinger etc. i et program.
- Man kan tilføje deklarativ information til et program ved hjælp af en attribut.
- Et deklarativt tag er vist ved firkantede ([]) parenteser placeret over elementet det anvendes til.
- Attributter bruges til at tilføje metadata, såsom compiler instruktion og andre oplysninger, såsom kommentarer, beskrivelse, metoder og klasser til et program.
- .NET Frameworket indeholder to typer af attributter: de foruddefinerede attributter (pre-defined) og specialbyggede attributter (custom built).

Repetition Attributes AttributeUsage Obsolete

- Denne foruddefineret attribut markerer en program enhed, som ikke bør anvendes.
- Det giver dig mulighed for at informere compileren om at kassere et bestemt mål element.
  - For eksempel, når der anvendes en ny metode i en klasse, og hvis du stadig ønsker at beholde den gamle metode i klassen, kan du markere den som forældet ved at vise en meddelelse om at den nye metode bør anvendes i stedet for den gamle metode.
- Syntaks

```
[Obsolete( message )]
[Obsolete( message, iserror )]
```

- Hvor
  - Parameteren *message* er en streng, der beskriver grunden til at elementet er forældet og hvad der skal bruges i stedet.
  - Parameteren *iserror* er en boolesk værdi. Hvis dens værdi er sand, skal compileren behandle brugen af elementet som en fejl. Standardværdien er falsk (compiler genererer en advarsel).

Repetition
Attributes
AttributeUsage
Obsolete

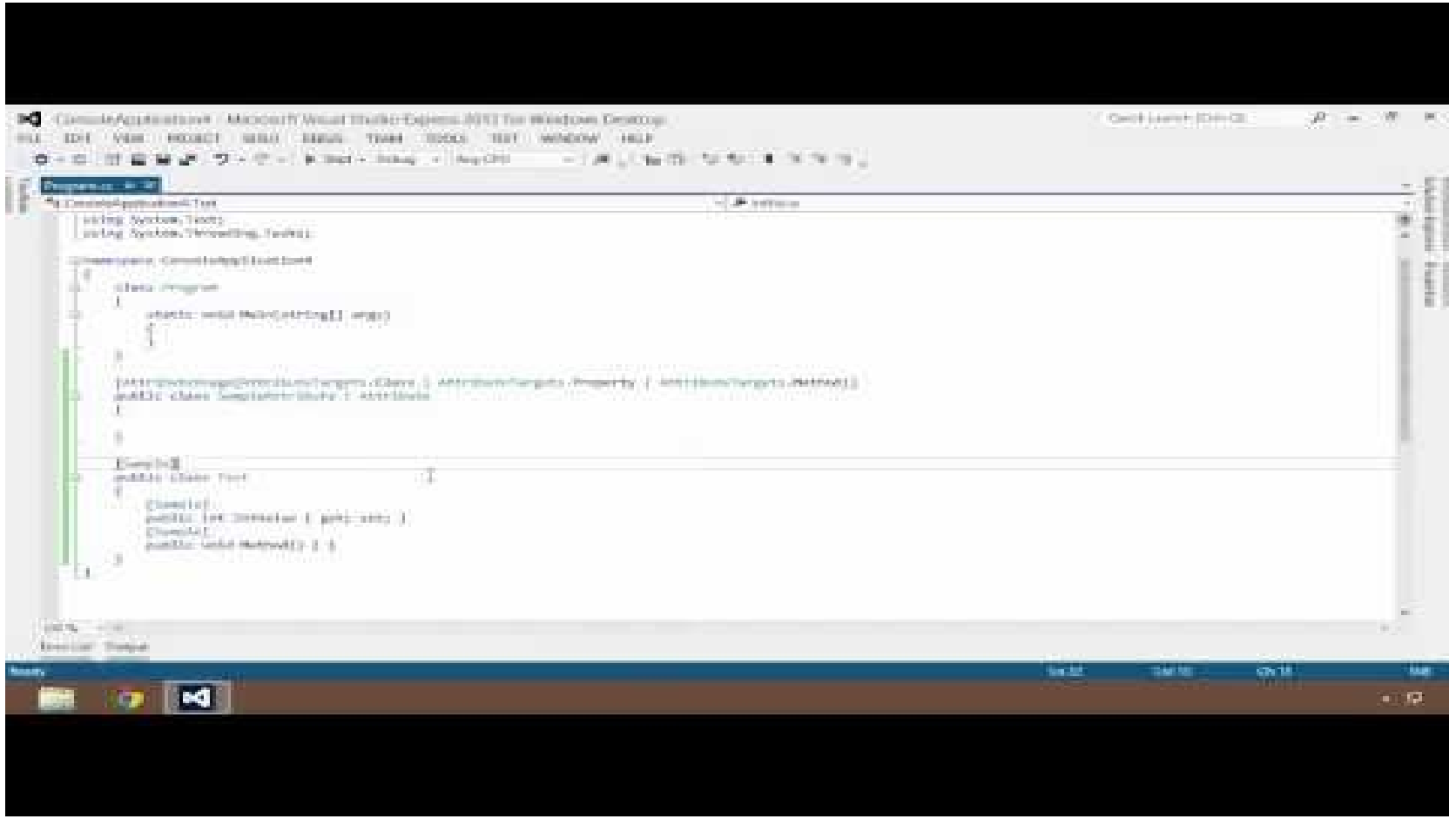
```
1   using System;
2
    0 references
3   public class MyClass
4   {
5       [Obsolete("Don't use OldMethod, use NewMethod instead", true)]
        1 reference
6       static void OldMethod()
7       {
8           Console.WriteLine("It is the old method");
9       }
        0 references
10      static void NewMethod()
11      {
12          Console.WriteLine("It is the new method");
13      }
        0 references
14      public static void Main()
15      {
16          OldMethod();
17      }
18  }
```

Repetition Attributes
Custom Attributes

- .NET frameworket tillader oprettelse af brugerdefinerede attributter, der kan bruges til at lagre deklarative oplysninger og kan hentes ved run-time. Denne information kan relateres til ethvert mål element afhængigt af designkriterier og program behov.
- Oprettelse og brug af brugerdefinerede attributter involverer fire trin:
  - Erklære en brugerdefineret attribut
  - Konstruktion af den brugerdefinerede attribut
  - Tilføj den brugerdefinerede attribut på et mål program element
  - Adgang attributter gennem refleksion
- Det sidste trin omfatter det at skrive et simpelt program til at læse igennem metadata til at finde forskellige notationer.
  Metadata er data om data eller oplysninger, der anvendes til at beskrive andre data. Dette program bør bruge refleksioner til at få adgang attributter ved runtime.

Repetition Attributes

Custom Attributes



Repetition Multithreading

- En tråd er defineret som udførelsesvejen af et program.
- Hver tråd definerer en unik strøm af kontrol.
  - Hvis en applikation indebærer komplicerede og tidskrævende operationer, så er det ofte nyttigt at indstille forskellige udførelses stier eller tråde, hvor hver tråd udfører et bestemt job.
- En tråd kaldes også en letvægts proces (lightweight process).
- Et almindeligt eksempel på brug af tråde er gennem-førelsen af parallel programmering af moderne operativsystemer.
- Anvendelse af tråde sparer spild af CPU cyklus og øge effektiviteten af en applikation.

Repetition
Debugging i VS

Opgave

Jeg står til rådighed til svar på spørgsmål og forklaring af fejl

Opgave

OPGAVE

- Tag prøverne inde på https://mva.microsoft.com/en-US/training-courses/c-fundamentals-for-absolute-beginners-16169 fra en ende af.
- I behøves ikke se videoerne, men brug dem endelig som reference / genopfriskning.

- Skriv på jeres eksamens rapport eller program.

Kilder

Materiale benyttet i denne lektion
Noget af det er udover pensum-listen!

Kilder

Eksempel

- http://www.w3resource.com/csharp-exercises/linq/csharp-linq-exercise-27.php

Sikker kode

- https://msdn.microsoft.com/en-us/library/d55zzx87(v=vs.90).aspx
- https://msdn.microsoft.com/en-us/library/8a3x2b7f(v=vs.90).aspx

Kilder

Windows Forms

- https://youtu.be/dbKHYDjijNo
- http://csharp.net-informations.com/gui/cs_forms.htm
- http://www.homeandlearn.co.uk/csharp/csharp_s1p5.html
- http://www.c-sharpcorner.com/article/create-basic-calculator-using-windows-forms-and-c-sharp/
- http://www.ittutorialswithexample.com/2015/01/simple-windows-form-login-application-in-csharp.html

Repetition

- https://youtu.be/spYyUBCkM70 (Polymorfisme)