**Beskrivelse af opgaven:**

Lav et C# applikation til at oprette en eller flere liste-struktur der kan indeholde objekter af en eller flere klasse datastruktur.

Data på en given person kan angives ved navn, alder, bopal, postnummer, job-og-løn, telefon nummer eller skole/universitet mm. Personen kan i denne opgave enten være studerende eller være i et job.

Derfor lav en base klasse der hedder Person, hvor afledte klasser Student og Employee arver fra Person. De skal også arve fra en interface som gøre man kan lave en Compare af to objekter. Dette bruges til sortering.

NamespaceXX

{

Public abstract class Person { //Abstract class to force possible other programmers to follow my design.

* //Der bruges propperties til alle felter, da ingen umiddelbart kræver noget særligt vedr. readonly eller lign.
  + Navn – Prop?
  + Alder - Prop? ->Fødselsdato? – metode til alder?
  + Adresse - Prop?
  + Postnummer - Prop? ->Indput validering på ”XXXX”
  + Telefonnr. - Prop?

}

}

NamespaceXX

{

Public class Student : Person, IComparable

{

* Skole/universitet - Prop? -> //valgt en Enum, for at bruge det samt dropdown i opgaven
* IComparable skal implementers

}

Employee : Person, IComparable

* Job -> prop
* Løn? - > prop -> lav til nullable? Så man laver til ikke obligatorisk felt! //skriv at det giver note boxing/unboxing.
* IComparable skal implementeres

Andre Klasser?

Kartotek(Archives)

List<Students> Students = new List<Students>();

List<Employees> Employees = new List<Employees>();

List<Persons> Persons = Students + Employees;

Diverse Metoder

(list har en Add(), AddRange(), Remove(), RemoveAt(), IndexOf(), Contains(), Count()

RemoveAt fjerner på et given index . Det skal nok bruges når man sletter.

Kan man i main metoden, oprette en kartotekklasse først?

Udskriv klasse skal oprettes med et interface. Interface FileWrite har udskriv som udskriver til Fil – Se Mosh video.

Programmet skal stille et **menu** til rådighed hvor brugeren kan få lov til at

·         indtaste en ny forekomst, opdatere en eksisterende,

Metode til at oprette (Check på om person findes via telefonnr, hvis den gør skal kunden overskrives (besked i log om overskrivelse/opdatering) ellers skal den oprettes.)

Int phonenumber = phonenumber

For (var I; i<persons.count; i++ {

If (phonenumber == Persons[i].phonenumber)

Bla bla

} else {

Var Person = New Persons (xxx,xxx,x,xx,x).

Add.list (Person)

}

·         delete eksisterende forekomst,

Metode til at slette en personklasse (der skal noget unit test på – lav noget søgning på personen, hvis den findes kan den slettes ellers return en besked om at personen ikke findes. – Exception handling hvor den vises i et advarselsfelt.

Public void DeletePersonFromArchives (int phonenumber){

For (var I; i<persons.count; i++ {

If (phonenumber == Persons[i].phonenumber)

RemoveAt(indexOf(i));

} else {

Throw PersonException(“There is no person with that phone number.”)

}

}

·         vise antal personer i listen, antal studerende og antal personer i job

Metode til returnere antallet af (paramter) personer i listen – evt. vist alt på en gang eller afhængig af hvad der ønskes?

·         vise personer med minimum og maximum alder eller med minimum og maximum løn.

Metode til returnering af liste (parameter) –så samme metode bruges til alle - udvid evt. med params så den kan bruges til flere type søgninger

·         give mulighed for at kunne finde en person udfra et telefon nummer.

Metode: Metode til returnering af liste (evt. samme metode som ovenfor bruges til alle )- udvid evt. med params så den kan bruges til flere type søgninger

·         håndtere exceptions og andre fejl på en passende måde

Lav exceptionklasse – måske en på klasse og en på kartotek og vis fejlene i en log måske på skærmen. Brug evt. samme log (med timestamp?) til at vise at en person bliver oprettet / Slettet.

Log Klasse? Singleton klasse. Måske et Interface! Så den bare tager mod en besked og det så er Log klassen der bestemmer hvordan data skal behandles.

·         generering af brugervenlige fejlmeddelelser,

Løses via exception og Log klassen

·         mulighed for at gentaste fejlbehæftede felt.

Try catch på input med indikator på om et felt er forkert udfyldt. Try catch på hvert input, hvor catch kan vise tekst og rødt tegn ud fra hver boks.

·         kunne sortere listen, eksempelvis efter navn i faldende eller stigende orden eller begge muligheder.

Hvordan skal listen vises på GUI? – hvordan sorteres den gu imæssigt?

Man kan bruge ”GRID” – Se doku om grid

* udskrive listen, enten studenter-listen eller employee-listen. (evt. valgmulighed om fil eller info på skærm)

Output file - Dan en fil med data. Få en Path (https://www.tutorialspoint.com/csharp/csharp\_file\_io.htm)

Du vælger selv om det skal være en console applikation eller en GUI, da her er der lagt vægt på at man skal kunne anvende klasser, interfaces, arving, liste, sortering, mm. Dvs. de mest vigtige begreber inden for OOP.

Det skal i hvert fald som minimum se sådan ud (hvis det er console, og hvis det er GUI så er det jer der bestemmer hvordan skal det se ud):

 Eksampel:

Operation Menu:

1. Tilføj eller Opdater ny person data. Applikationen skal tjekke om personen allerede findes eller ej. Brug telefonnummer som unik nøgle. Hvis den findes, opdater personen.   
  
2. Fjern en person fra listen (kom også med en brugervenlig meddelelse i tilfælde af om elementet findes eller ej)  
  
3. Vis antal personer I listen, antal studerende og antal personer i job  
  
4. Vis personer med Min og Max alder, eller løn  
  
5. Find en person  
  
6. Sorter listen og skriv ud på skærmen.  
  
7. Afslut programmet

%2F%2F Cool Class Diagram, [Person|\_name:string;\_age:int;\_adresse:string;\_postNumber:int;\_phoneNumber:int|CompareTo();SortAgeAscending()]^-[Student|\_School:School],[Person]^-[Employee|\_job:string;\_salary:double|SortSalaryDescending(); SortSalaryAscending()]

[<<IComparable>>;CompareTo]^-.-[Person]

[Archive|Students:list;Employees:list;Persons:list;|CreateOrAlterStudent();CreateOrAlterEmployee();DeletePerson();ShowNumberOfPersons();ShowPersonsWithAge();ShowPersonsWithSalary();FindPersonWithPhoneNumber()]++-\*[Student]

[Archive]++-\*[Employee]

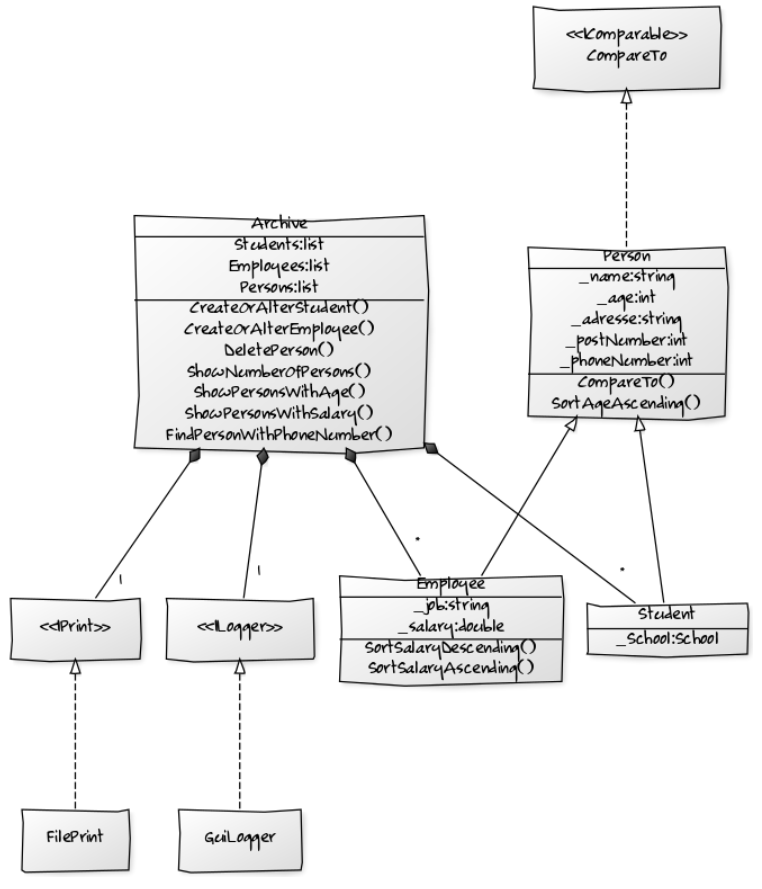
[Archive]++-1[<<ILogger>>]

[Archive]++-1[<<IPrint>>]

[<<ILogger>>]^-.-[GuiLogger]

[<<IPrint>>]^-.-[FilePrint]

Lav logger delen med composition mv. inkl. Enum til message type.



Design kan ses af ovenstående diagram, Hvor der laves en abstrakt person klasse, hvor der er to arvede klasser. Person klassen er lavet abstract da opgaveteksten går meget på at man skal være enten medarbejder eller studerende og det ikke er meningen at der skal være en instans af person klassen.

Både Student og Employee har ekstra data og implementere interfacet IComparable, da de to objekter skal tilføjes lister og senere sorteres og her skal listen vide hvornår en studerende er lig, mindre eller større end anden studerende eller employee.

Jeg har valgt at lave 3 lister. En med students, en med employees og en der ligger dem sammen. Jeg tænkte oprindeligt, at have en liste og så lave en for løkke og caste til student/employee (AS operatoren) for så at validere om casting er succesfuldt, så jeg ved om det er en studerende eller ej – Men endte med at konkludere at det blev for beregningstungt fremfor ”bare” at lave en type-safe liste der kun holdte det ene eller det andet og så en generisk objekt liste til begge dele, når alle værdier skal bruges.

I min Gui vil der i oprettelse være en radio button som bliver oversat til boolean værdi isStudent, så der bliver kaldt den korrekte CreateOrAlter metode – Jeg valgte 2 metoder, da de to konstruktors skulle bruge forskelligt input. Kiggede oprindeligt på at lave en metode, med optional parameters – men da jeg gerne vil lave salary nullable, da der er mange, der ikke vil opgive løn, var det lidt modstridende.

For at udvide brugen af interfaces har jeg valgt at uddelegere udskrift funktionen til et interface, hvor jeg har oprettet en klasse der implementerer interfaces der udskriver til en fil. Lidt i samme stil har jeg lavet en logning, hvor logging sker til skærmen via interface. Det sker via dependecy injection i Program CS, så det vil være let at lave en ny klasse der eksempelvis logger til en fil i stedet og bruger denne i program.cs i stedet og voila har man nu sin logging i en fil.

Man kunne også gøre begge dele – men her skulle man så lave en liste af logninger og hvor de to blev tilføjet og så skulle de metode kald der bruge metoden have en foreach item i listen .log , så der blev send besked til båd fil og skræm at der skulle lave en logging registrering.

Exception class og unit tests.

Opgaver

1. lav Create/alter x 2 færdig.
2. Implementer Icomparable (vent med ICompare() – grid laver evt. Sorteringen)
3. Lav de andre metoder
4. Lav Gui – inkl. Navne på bokse mv.
5. Connecte guid elementer med backend
6. Lav Logging (ændre UML til at indholde en logger)
7. Lav Udskriv
8. Lav exception handling
9. Lav Unit test på de 5 metode kald.

Add. 1.

Tag baseret på bool skal listen bestemmes om den ene eller anden metode bliver kaldt. Check på om kunden findes i listen kan måske laves som separat metode.

Add 2.

Mulighed for telefonnnr.

public class Student : IComparable<Person>

{

[...]

public int CompareTo(Person obj)

{

if (this.PhoneNumber > obj.PhoneNumber) return -1;

if (this.PhoneNumber == obj. PhoneNumber) return 0;

return 1;

}

}

Hvis man burger navn og string metoden til sort

// Implement IComparable CompareTo method - provide default sort order.  
int IComparable.CompareTo(object obj)  
{  
 person a=(person)obj;  
 return String.Compare(this.name,a.name);  
  
}

//BRUG ICOMPARER I stedet. Så kan man sorter på mere.

//navn, age, adresse, postnumber, phonenumber

<https://support.microsoft.com/da-dk/help/320727/how-to-use-the-icomparable-and-icomparer-interfaces-in-visual-c>

The role of IComparer is to provide additional comparison mechanisms. For example, you may want to provide ordering of your class on several fields or properties, ascending and descending order on the same field, or both.   
  
Using IComparer is a two-step process. First, declare a class that implements IComparer, and then implement the Compare method:

private class sortYearAscendingHelper : IComparer  
{  
 int IComparer.Compare(object a, object b)  
 {  
 car c1=(car)a;  
 car c2=(car)b;  
 if (c1.year > c2.year)  
 return 1;  
 if (c1.year < c2.year)  
 return -1;  
 else  
 return 0;  
 }  
}

Note that the IComparer.Compare method requires a tertiary comparison. 1, 0, or -1 is returned depending on whether one value is greater than, equal to, or less than the other. The sort order (ascending or descending) can be changed by switching the logical operators in this method.  
  
The second step is to declare a method that returns an instance of your IComparer object:

public static IComparer sortYearAscending()  
{   
 return (IComparer) new sortYearAscendingHelper();  
}

In this example, the object is used as the second argument when you call the overloaded Array.Sort method that accepts IComparer. The use of IComparer is not limited to arrays. It is accepted as an argument in a number of different collection and control classes.

private class sortYearAscendingHelper : IComparer  
{  
 int IComparer.Compare(object a, object b) {  
 person a=(person)a;  
 person b=(person)b;

//lav evt. If(a!=null && b !=null)  
 if (a.name > b.name)  
 return 1;  
 if (c1.year < c2.year)  
 return -1;  
 else  
 return 0;  
 }  
}

GridView:

Data grid view controller from toolbox

Hvis man dobbeltklikker på form (menubaren, så kommer der en form\_load metode, hvor man kan sætte datasource er lig med listen – se ndenstående

I Form1() kan man også sætte datagrid lig resultlist – så burde den blive popupliceret med det samme form 1 loader.

Lav en resultatliste – start med at vise alle personer

List<persons> resultList = new list<persons>();

Sæt datasource til den liste man lige har lavet.

Når man så trykker på en knap der skal vise en liste så skal resultatliste pege den det man ønsker at vise.

dataGridView1.Datasource = resultList;

Table kolonne navn kommer fra person klasse navn.

Skal der laves en for personer, student og/eller employee?

Behøves ikke at rette – burde tilpasses automatisk

Add. 3

//Fjern en person fra listen (kom også med en brugervenlig meddelelse i tilfælde af om elementet findes eller ej)

Pulic void DeletePerson(int phoneNumber){

for (i = 1; I < persons.count; i++)

{

If(persons[i].phoneNumber == phoneNumber){

//Log udskriv data om at person.navn slettes fra listen

Persons.Delete(persons[i])

}

}

}

//Vis antal personer I listen

Public int ShowNumberOfPersons(){

Return persons.count;

}

//overload af ShowNumberOfPersons til at hvis en specific liste

//Vis antal personer studerende og antal personer i job

Public int ShowNumberOfPersons (boolean showStudents){

ShowStudents ? Return students.count : return employees.count;

}

Vis personer med Min/max alder

Public list<persons> ShowPersonsWithAge(Boolean lowestAge = false){

List<persons> minAgeList = new list<persons>

List<persons> maxAgeList = new list<persons>

Int minAge;

Int maxAge;

Persons.sort();

//da listen er sorteret efter alle må min og max være de to yderse værdier.

if ((persons[0].age< persons[persons.count - 1].age)

{

minAge = persons[i].age

maxAge = persons[persons.count - 1] )

} else

{

maxAge = persons[i].age

minAge = persons[persons.count - 1] )

}

//hvis en person på listen har samme alder som min eller max alder bliver de tilføjet den respective liste .

for (i = 1; I < persons.count; i++)

{

if (persons[i].age= minAge) {

minAgeList.Add(persons[i])

}

if (persons[i].age= maxAge) {

maxAgeList.Add(persons[i])

}

}

//returnere den liste der bliver bedt om.

(lowestAge) ? minAgeList : maxAgeList;

}

Vis personer med Min og Max løn

Public list<employees> ShowEmployeesWithSalary(Boolean minSalary){

List<employees> minSalaryList = new list<employees>

List<employees> maxSalaryList = new list<employees>

Double minSalary;

Double maxSalary

Employees.sort(); //efter salary

//da listen er sorteret efter løn må min og max være de to yderse værdier.

//hvis listen er sorteret stigende/falden kunne man bare min = [0], max = count -1

if ((employees[0].salary< employees[employees.count - 1].salary)

{

minSalary = persons[i].age

maxAge = persons[persons.count - 1] )

} else

{

maxAge = persons[i].age

minAge = persons[persons.count - 1] )

}

//hvis en person på listen har samme løn som min eller max alder bliver de tilføjet den respective liste .

//hvis jeg ved at listen er sorteret stigende kunne jeg lave while løn = minLøn og spare tid.

for (i = 1; I < employees.count; i++)

{

if (employees[i].salary= minSalary) {

minSalaryList.Add(employees [i])

}

if (employees [i]. salary = maxSalary) {

maxSalaryList.Add(employees [i])

}

}

//returnere den liste der bliver bedt om.

(lowestAge) ? minAgeList : maxAgeList;

}

Find en person

Public person FindPersonWith(int phoneNumber)

{

for (i = 1; I < persons.count; i++)

{

If(persons[i].phoneNumber == phoneNumber)

{

Return persons[i]

}

}

Throw exception(“person not found”)

}

Add. 6. Logging

//Interface

Namespace xxxx

{

Public interface ILogger{

void LogError(string message)

void LogInfo(string message)

}

}

//class der bruger Interface //dependency injection

Namespace xxxx

{

Public class Archive

{

Private readonly Ilogger \_logger;

Public Archive (ILogger logger)

{

\_logger = logger;

}

Public CreateOrAlter (dsdfsfd ds fsd fdsf)

{

\_logger.LogInfo(”Studerende oprettet kl. ” + DateTime.Now);

}

Public DeleteXXXX (dsdfsfd ds fsd fdsf)

{

\_logger.LogInfo(”xx blev slettet kl. ” + DateTime.Now);

}

}

}

//class der implementere interface

Public class GuiLogger : ILogger

{

public void LogError(string message)

{

Ændre tekst farven til Rød.

Text.felt = sgdfs

}

void LogInfo(string message)

{

Text.felt = sgdfs

}

}

// I program.cs hvor der oprettes Archive skal der indsættes et interface //skriv kommentar.

Archvie archive = new Archive(new GuiLogger());

//Hvis man I stedet burger FileLogger ser det således ud:

Archvie archive = new Archive(new FileLogger(“C:\\log.txt”));

Add. 7. Udskriv - //interfaces.

Udskriv til en Fil:

Namespace xxxx

{

Public interface IPrint

{

void Print(string message)

}

}

//class der bruger Interface //dependency injection

Namespace xxxx

{

Public class Archive

{

Private readonly IPrint \_print;

Public Archive (IPrint print)

{

\_print = print;

}

Public ShowListxxxxx (dsdfsfd ds fsd fdsf)

{

\_print.Print( resultat listen );

}

}

}

Public class FilePrint: IPrint

{

private readonly string \_path;

public FilePrint(string path)

{

\_path = path;

}

public void Print(string message)

{

Using (var streamWriter = new StreamWriter(\_path, true))

{

streamWriter.WriteLine(message);

}

}

// I program.cs hvor der oprettes Archive skal der indsættes et interface //skriv kommentar.

Archvie archive = new Archive(new GuiLogger(), new FilePrint(“C:\\log.txt”));

Add. 8.

Lav de tre exceptions konstruktors – lav try catch nogle steder og throw diverse fejl med beskeder. Genbrug evt. fra Pension eller Pizza exception.

Add. 9.

{

[TestClass]

public class ArchiveTests (name of class+Tests)

{

[TestMethod]

public void MetodeNavn\_Scenarie\_ExpectedBehavior

{

//Arrange

var object = new xxxx();

//Act

var result = object.metode(parametre der passer med scenarie);

//Assert

Assert.isTrue(result)

}

}

}