DNP Assignment – Part 1  
Blazor

In this assignment you will create a Blazor-server web page to manage data and view statistics about families.

There are two approaches, described below. You decide where to start. If you do approach #1 first, you can still expand to approach #2 afterwards. The second is a lot more comprehensive than the first.

A family consists of an address, a number of adults, a number of children, and a number of pets.

# Approach 1 – Managing Adults

In the first approach, you will only work with Adults.

The below, figure 1, is the Domain Model related to the Adult:

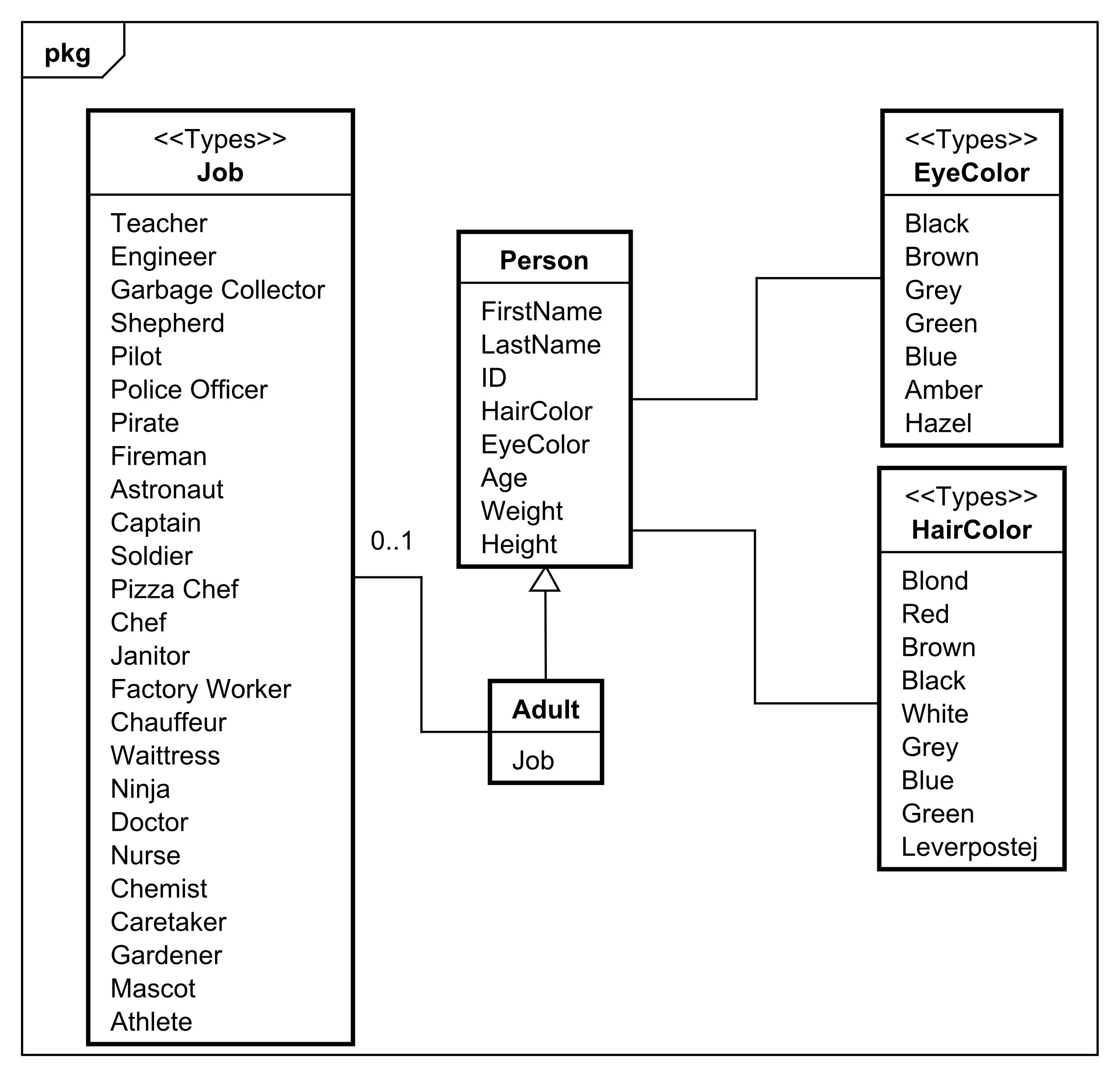


Figure 1, domain model, isolated around Adult

The Job, EyeColor, HairColor are actually just strings, but in the diagram above they are marked as *Types* to indicate it is a limited list of possible values. I.e., a Person’s eye color can only be one of the colors shown above. The same is true for hair color. For Jobs, however, there can be more than what is shown, there is no limit here.

Converting this to C# you will get an Adult class, which extends a Person class, see figure 2.

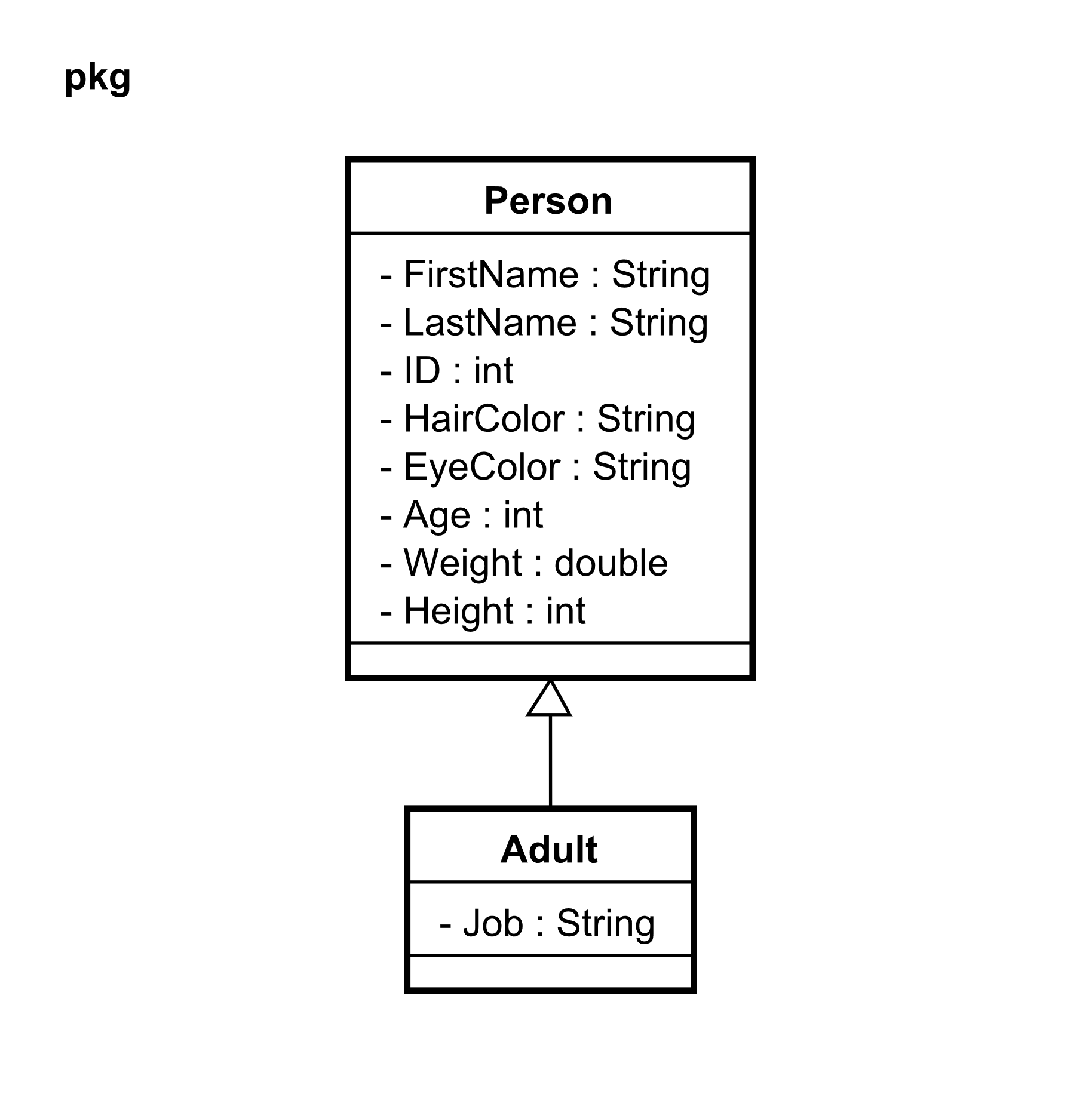


Figure 2, Class diagram for an Adult

If you pick this approach, you’ll just be working with a collection of Adult objects.

The Adult/Person objects contain a couple of validation attributes, take a look at the classes.

# Approach 2 – Managing Families

In this approach, you get the entire Family. The Domain Model is shown below in figure 3:

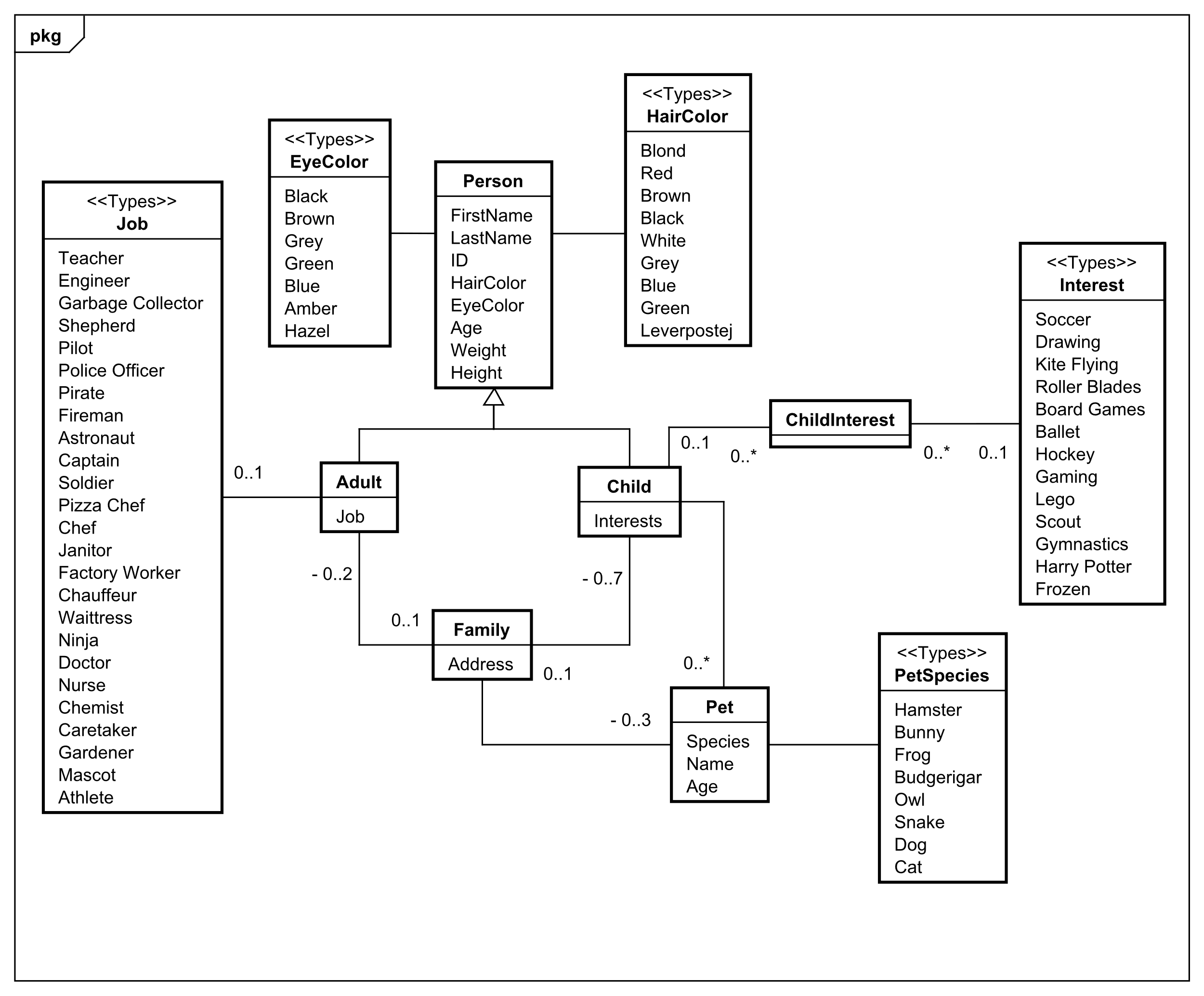


Figure , Family domain model

Again, there are a couple of entities marked with <<Types>>, used to indicate that there is a finite list of options (Interest is similar to Job, you may add functionality to add new Interests, if you wish).

An adult optionally has a Job. The attributes inherited from Person are supposed to be mandatory. There can be more job types than shown above.

A child optionally has zero or more ChildInterests. A ChildInterest entity consists of a Child and an Interest, a ChildId and an InterestId. This setup is a bit strange, explanation to follow:

Normally this would be represented by the Child having a List<string> of Interests, but a detail about Entity Framework Core, which we will discuss later in the course, requires that we create this extra entity, ChildInterest. It is similar to the join table used for many-to-many relationships in relational databases. Given that we will be using a database later on, we have to model the C# models accordingly. This also provides you with an example.

The result is that in C#-objects a Child now have a List of ChildInterest, and each ChildInterest has an InterestId, which just has a *string*. Other fields are needed by the database later, and will be added in later assignments.

This also means, the Interest type is not used yet.

You may notice, we could have made a many-to-many relationship between Job and Adult, too, but to simplify things, we will violate normalization here.

**Family Structure**

A Family consists of 0..2 Adults, and 0..7 Children. A Family can have Pets. And Children can have personal Pets.

Regarding the address of the Family, in the class diagram this consists of a StreetName and HouseNumber, the combination of which is a unique key, i.e. no two Families can have the same StreetName *and* HouseNumber.

The class diagram corresponding to the above domain model can be seen in figure 4 below:

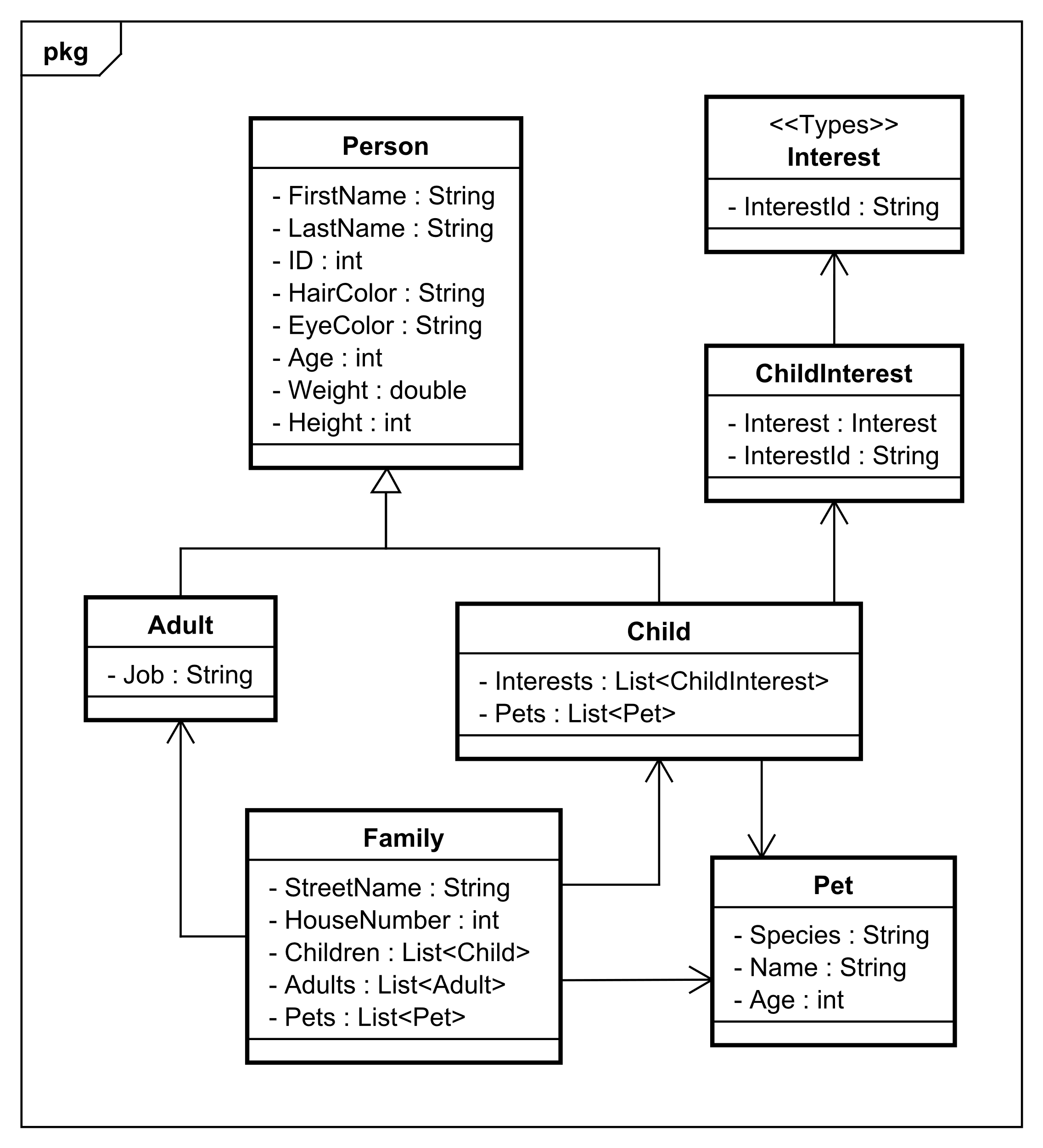


Figure 4, class diagram for Family

**Start small**

You don’t have to handle the entire model above right away.

You can start with CRUD-operations for Family and Adult, and just ignore other information in Family like children and pets. E.g. comment these fields out, or just ignore them.

You could then expand with either Family Pet or children. For children, you can ignore the interests/pets initially, and expand later. Again, this just means leaving certain field variables empty, and not displaying them in the GUI.

So, expand step by step or until you feel you have enough.

# What is handed out

**Domain classes**

You will be given the C# domain classes for the above, the entire thing. If you go for approach #1 you can just ignore and not use most of the classes.

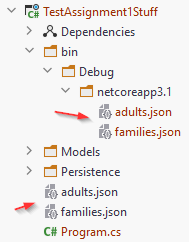
**File storage**

You are given a class, FileContext, which can load from and save to a file. It uses JSON format.

**Files**

Two files are used, one to store a list of Families, and one for a list of Persons. They both come with a bunch of randomly generated data to get you started.

They are not used together, so use Adults list for approach #1, and Family list for approach #2. You should probably change the path of the files to match your location, or place the files accordingly. It seems to vary, where you have to place the files:



**Modifications**

You may change the class if needed, e.g. comment out in the SaveChanges the part you’re not using. But other than that you most likely need not change anything.   
The idea is that you write an adapter around it, i.e. you create your own class with functionality like:

* Filtering/searching
* Managing Ids, so they remain unique
* Updating, Adding, Deleting.
* What ever else you need.

The FileContext::SaveChanges method will write both lists to the file, overwriting the existing data.

# What you need to do

You must create a Blazor web page to view, edit, add, remove data. You must use server-Blazor.

**Minimum requirements:**

1. Be able to add an Adult to persistent storage (i.e. the file, through the FileContext class).
2. Be able to search for Adults(s) by name or other criteria. The result must be displayed on a web page.
3. There must be multiple pages, i.e. one for adding, one for viewing.
4. There must be some kind of log in system, which will restrict access to certain parts of the web page, e.g. only logged in users may add Persons.   
   You can create your own User class, and store in a file similarly to the Adults and Families.   
   You may hardcode a couple of users, if needed. Or create a registration page.
5. Remember relevant interface separations, like you were taught last semester. This will help you in the following assignments.

Below are ideas for further development:

* Show data in various charts. E.g. a distribution of the ages of all Persons. Or a pie chart for eye color distribution. Or other kinds of statistics using various types of charts.   
  Find charts here: [Blazor charts](https://blazor.syncfusion.com/demos/)
* More elaborate user privileges, using various Claims.
* More elaborate search functionality.
* Filter functionality, e.g. ”show all persons between ages 35-37”, or ”show all persons with green eyes”, or ”show all families with more than 3 children”, etc.
* Edit functionality for Persons/Adults/Families.
* Switch to Family data, and make functionality for Create, Read, Update, Delete.
* Show various Family statistics, e.g. distribution of number of children, or number of single parents.
* Use your own imagination.

# Format

You may work on this assignment individually or in group. This decision can be changed for the next assignment, so you’re not stuck with your choice.

**The assignment must be handed in and approved in order to go to the exam.**

**What to hand in:** Link to short video demonstration (~2 minutes), and a link to your github with the project.

**Deadline**: See itslearning.