Advanced Programming Assignment 1 Observations
Rudi Basiran (s3665980)

The weekend after we got the assignment, each of us went away to think about how to tackle the problem.

Sherri shared that she discovered that ArrayList could be a way to keep all the friends. Based on that, I explored using:

```
ArrayList<String[]> rowList = new ArrayList<String[]>();
```

with the following initialization:

```
rowList.add(new String[] { "Rudi", "Married To", "Ahysa" });
rowList.add(new String[] { "Ahysa", "Married To", "Rudi" });
```

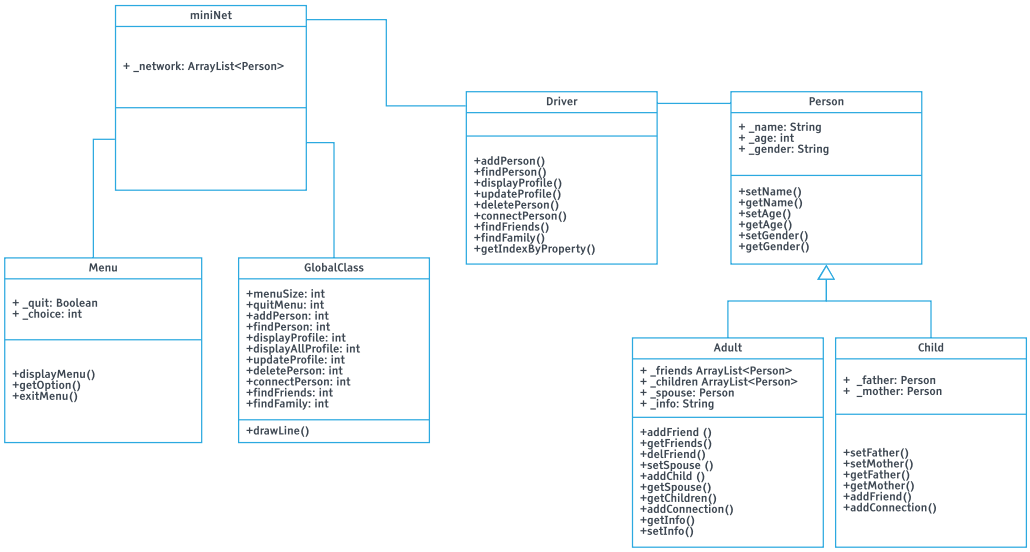
This search method was used:

```
if ((rowList.get(i)[searchParam1] == searchList.get(searchParam1)
&& rowList.get(i)[searchParam2] == searchList.get(searchParam2)
&& rowList.get(i)[searchParam3] == searchList.get(searchParam3)) ||

(rowList.get(i)[searchParam1] == searchList.get(searchParam3)
&& rowList.get(i)[searchParam2] == searchList.get(searchParam2)
&& rowList.get(i)[searchParam3] == searchList.get(searchParam1)))
```

[I did not realize at that time that this 3-dimensional array became the blue-print eventually of the Relationship Class]

The initial draft model had Person class extended out to Adult and Child. Each Person had an ArrayList of Person (_friends) along with a variable Person _spouse. Each Child too had an ArrayList of Person (_friends) along with variables Person _father and _mother.



However, as we got into coding, it became very tedious to maintain these relationships as for each person we added as a friend, we had to do this in the Driver class:

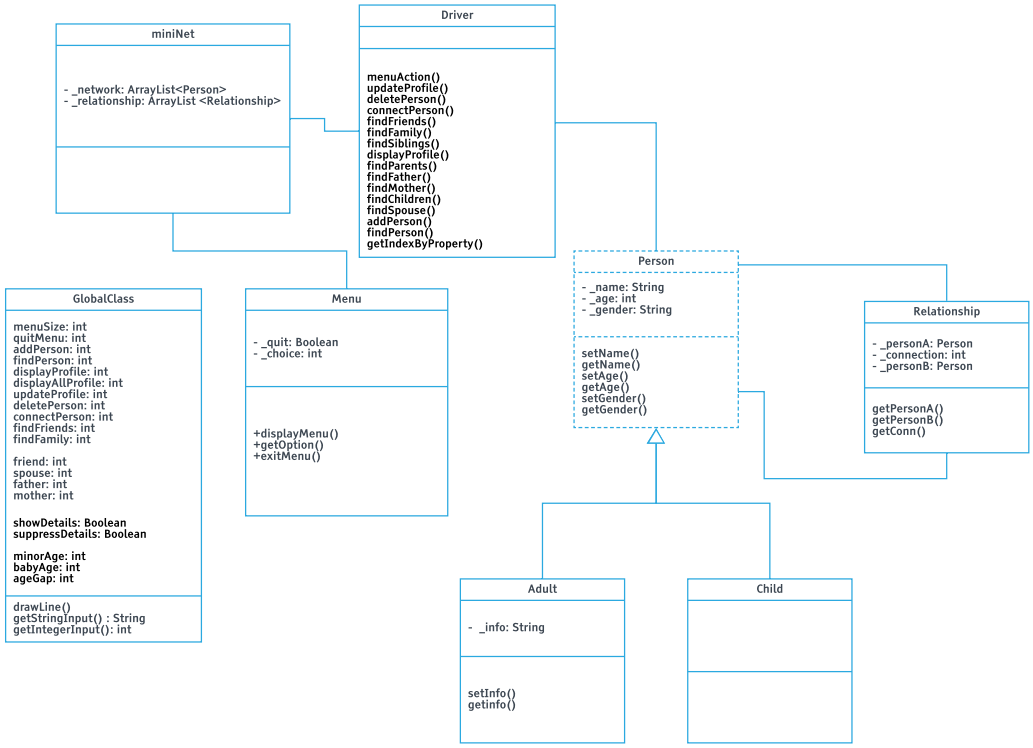
```
Public void addConnection (Person a, Person b)
{
	_friends.add(a);
	a.addFriend(this);
}

public void addFriend(Person p) {
	_friends.add(p);
}
```

To remove, we had to take action on both persons as well.

The setFather() & setMother() were alright to implement but if we were to delete a Person, that would mean having to write separate methods to delFather() and delMother().

It was at this point that we decided to embark on a fresh idea – to make use of a Relationship class.



That made things a lot easier as adding and deleting a relationship was simply done using relationship.remove(new Relationship(Person p, relationship_type, Person q).

The design is basically a “has-a” relationship where a Person has someone else related to him/her be it in the role of a friend, spouse, father or mother.

GlobalClass was created as we kept repeating certain variables, so it made sense to just make them global; these reduced mistakes whenever we have to change items across the board like “Friend” to “friend” and it also made control of the menu/functions easier instead of passing around meaningless integers or complicated strings. The values of minorAge, babayAge and ageGap can easily be changed, if the rules change, at 1 place instead of hunting everywhere for the initial setting of 16, 2 and 3.

getIntegerInput and getStringInput were also made into global functions so that we did not have to keep on doing a Scanner sc = new Scanner(System.in) each time we had to solicit input from the user; the integer validation is just done once instead of all over the place.

As Sherri mentioned in her observations, once we figured out how to find a particular object be it an Adult, Child or Relationship and its attributes, the specifications were fairly easy to manage. However, the longer we took to code, the more what-if’s we came out with and thankfully, we managed to take care of them all.

Listed below as well as in the README.md are the various scenarios that we thought of. If we had tackled deleteConnection(), there would probably be many more to think of.

An additional feature that we thought of was to have gender as part of a Person; that way, it's easier to allocate/validate the correct gender for the correct father/mother relationship. We did not check for the gender of spouses though and we did not "upgrade" a Child to an Adult or "downgrade" an Adult to a Child whenever age is changed as the specifications did not state what happens when these updates are made by the user; we just made sure that the requirements stated are met like for example if a 7-year-old Child (with an 8-year-old Friend) advanced in years to 15, an error message would pop up to show the ageGap (3 year) limitation.

We also made use of real-world people and relationships; it makes it easier to test rather than having fictitious characters with imaginary linkages.

We also did not automatically add connections; e.g., a newly-added spouse connection does not trigger off the children connections to the new father or mother. While we did initially cater for it, it led to too many checks for every role and it just got out of control, so we took it out totally.

The exceptions class was not extended as exception handling can be done using error messages and thorough logic check for this assignment; each object created or deleted was handled properly via the existing ArrayList add/remove methods used.

The toughest specification was to find out whether someone is in the same family or not, i.e., is Child A a sibling of Child B. Luckily the Relationship class managed to capture it properly and it was just a matter of slow, logical thinking to figure it out.

Lastly, what should have taken perhaps 10-15 hours of coding took longer because of the numerous scenarios that we thought of. This mimicked the real-world where given requirements are pretty basic at first and it's only upon being examined from every angle that more refined adjustments have to be made and the code/design fine-tuned accordingly. In the process, a lot of knowledge was gained learning about the idiosyncrasies of ArrayList's add, remove and contain.

List of Probable Scenarios

- user enters invalid menu option (int): passed
- user enters invalid menu option (string): passed

- addPerson
    - test add: passed
    - test add same person: passed, prevented

- findPerson
    - test person does not exist: passed
    - test person exist: passed

- displayProfile
    - test person does not exist: passed
    - test person exist: passed

- displayAllProfile

- updateProfile
    - test person does not exist: passed
    - test person exist: passed
    - test adult with spouse change age: passed
    - test adult with spouse change age: passed

    - test child with parents change age: passed
    - test child change age who has friends within age-gap: passed
    - test child change age to baby (with friends): passed
    - test child change age to baby (without friends): passed

- deletePerson
    - test person does not exist: passed
    - test person exist: passed
    - test delete person with spouse: passed
    - test delete person with friends: passed
    - test delete child: passed

- connectPerson
    - test both person exist/not exists combination: passed
    - test both person exist: passed
    - test connect friends (no connections yet): passed
    - test connect spouse (1 not adult): passed
    - test connect spouse (no spouse yet, both adults): passed
    - test connect spouse (spouse already exists): passed
    - test connect relationship already exists: passed
    - test connect relationship already exists (other way): passed
    - test child connect friend (adult): passed
    - test child connect friend (child inside family): passed
    - test child connect friend (child outside family): passed
    - test child connect friend (child not within age-range): passed
    - test child connect friend (child within age-range): passed
    - test child below 2 connect friend: passed
    - test child connect adult as parent: passed, should be parent connect child
    - test adult connect child (father already exists): passed
    - test adult connect child (mother already exists): passed
    - test adult connect child (parent does not exist): passed
    - test adult connect child (correct gender): passed

- findFriends (input person A, input person B)
    - test both person exist/not exists combination: passed
    - test both person exist: passed
    - test both persons are friends: passed
    - test both persons are not friends: passed

- findFamily
    - test person does not exist: passed
    - test person exist: passed

- return to menu once any option selected: passed