Programming Challenge: Secret Santa

There are many many different ways to solve this problem. What we are looking for is simply good code and good architecture. It is up to you to decide what that is.

We want your solution to be simple enough to complete in a few hours (it doesn't necessarily have to be "perfect" - if there is such a thing). Find a good medium, write the best code you can. If you don't have time to finish - send what you have, just explain what you planned on doing.

If you get done and decide you could have done something better, feel free to add commentary on what you would have done differently if you were to do it again. Any commentary is welcomed (it will help us understand your thought process).

Part one:

Imagine that every year your extended family does a "Secret Santa" gift exchange. For this gift exchange, each person draws another person at random and then gets a gift for them. Write a program that will choose a Secret Santa for everyone given a list of all the members of your extended family. Obviously, a person cannot be their own Secret Santa.

Part two:

After the third year of having the Secret Santa gift exchange, you've heard complaints of having the same Secret Santa year after year. Modify your program so that a family member can only have the same Secret Santa once every 3 years.

Part three:

As your extended family has grown, members have gotten married and/or had children. Families usually get gifts for members of their immediate family, so it doesn't make a lot of sense for anyone to be a Secret Santa for a member of their immediate family (spouse, parents, or children). Modify your program to take this constraint into consideration when choosing Secret Santas.

Directions:

1. Please create a repository on gitHub (or other online repository) for this exercise that you can share the access with us.
2. Please complete the three parts sequentially. After each part, please make a tag (e.g. 'Secret Santa Part 1')

Some things to keep in mind:

- Your code should be easily readable, commented, and maintainable
- Include unit-tests to validate your application's functionality (edge cases, normal operation, etc)
- If you need to store data, using an in-memory "database" is OK (simple Collections are fine). You do not need a datastore, but think about how it would be modeled in a datastore if you were to transition from your application's in-memory datastore to a persistent datastore.
- Your application may be viewed by many family members at the same time. Even though your application is single-threaded, you may want to consider what would happen with concurrent access.
- Your family may want to impose additional constraints on how Secret Santas are chosen
- You do not need a UI, just unit tests

Extra Credit:

1. Build a UI for your application
2. Deploy your application as a webapp to Heroku (or as an iOS/Android app with UI)
3. Switch your application's in-memory datastore to a persistent datastore