Individual Assignment Specifications

With the new topics we have studied since the original specifications, we realized that there are a number of elements in our design which could be improved:

1. **Requirement: Appropriate data structures.** Arrays are not the optimal data structure for arbitrary length, changing lists. We should use an appropriate data structure.
2. **Requirement: Family relationship consistency**. Our current implementation of family relationships doesn't ensure consistency. Imagine we have two siblings, Peace and Sam. How should we create and store the relationship? Consider the following:
    - ❖ We could create a FamilyRelation object *relationshipPeaceSam* with Peace as personOne and Sam as personTwo and store the object relationshipPeaceSam in Peace's familyConnections and also in Sam's familyConnections.
    - ❖ We could create a FamilyRelation object *relationshipPeaceSam* with Peace as personOne and Sam as personTwo and store the object relationshipPeaceSam in Peace's familyConnections, and create a FamilyRelation object *relationshipSamPeace* with Sam as personOne and Peace as personTwo, and store the object relationshipSamPeace in Sam's familyConnections.
    - ❖ We could even create a FamilyRelation object *relationshipPeaceSam1* with Peace as personOne and Sam as personTwo and store the object in Peace's familyConnections, then copy it to *relationshipPeaceSam2* and store this object in Sam's familyConnections.

    Our new requirements for family relationships are:
    - We need to ensure that our implementation enforces a two-sided relationship. This means that altering or deleting a relationship affects both DisasterVictims.
    - We need to ensure that it isn't possible to enter duplicate data. In the above illustrations, *relationshipPeaceSam* and *relationshipSamPeace* contain duplicate information because the assignation as personOne or personTwo is an arbitrary implementation detail.
    - We need to ensure that it isn't possible to store only part of a series of relationships. For example, if Peace and Sam also have a sibling Diamond, we should never have a situation where Peace and Sam's relationship, and Peace and Diamond's relationship are present, but Diamond and Sam's is not.
3. **Requirement: Supply consistency.** We need to ensure that when a Supply is allocated to a DisasterVictim, it is removed from the available supplies at a Location.
4. **Requirement: Use database.** We should use a database to store information which is shared between locations, such as inquiries by outside parties.
    - Use the SQL file `project.sql` (to be provided). The starting data in the database may be modified by TAs in marking.
5. **Requirement: Multiple interactions with inquirer.** We should maintain a log of interactions with the same inquirer instead of creating a new Inquirer object for each instance.

6. **Requirement: Interface to enter DisasterVictims.** We need an interface to our application which allows relief workers to enter information about DisasterVictims, their relationships, and their medical records for a particular stored location. This can be GUI or it can take terminal input.
7. **Requirement: Interface to log inquirer queries.** We need an interface to our application for relief services which allows them to log inquirer queries, and to search in order to potentially link to DisasterVictims. This interface can be GUI or take terminal input.
    - Search should be based on a part of a name, regardless of whether it is upper or lower case. For example, if I search for 'Pra' I want to see 'Praveen' and 'Oprah'.
    - We will want a way to determine whether we are entering the application as a central or location-based relief worker. You might use a command-line flag to control the mode, or have more than one main.

After considering the technical shortcomings of our application, we showed the prototype to someone working in the field and received a lot of valuable feedback about our assumptions. The biggest concern that they had was that our software was developed in a Canadian context, and doesn't consider how it might be used in other countries.

"Take a look at two open source projects for medical records, OpenEMR and OpenMRS. This will explain some of the complexities that you've overlooked. For example, you assume that someone will know their birthdate. You made the field optional, but what if you have a parent and a child with identical names - you will want a way to keep track of their approximate ages, even if you cannot store exact birthdays."

Based on this input we added the requirement:

8. **Requirement: Age or birthdate.** It must be possible to store a person's approximate age or their birthdate (these two fields cannot both be set).

Another person who reviewed our system raised a question about our use of female/male/other for gender. They pointed out that people don't usually identify as "other". They pointed out that the gender a person identifies with could be drawn from their cultural background, for example "hijra", "ubhatobyañjanaka", or "pandaka". In Canada, people might identify as "non-binary", "two-spirit" or "gender queer", although the 2021 census categorized all people who reported a gender other than "man" or "woman" as "non-binary".

We added this requirement:

9. **Requirement: Gender options from file.** The gender options available should be populated from a text-based configuration file. Our options should be limited to those provided in the file. This allows us to ensure a culturally relevant list of terms is available, while still preventing data from being corrupted by invalid typos like "womn".
    - We have provided a file, `GenderOptions.txt`. Use the same name and format as the example although you may modify contents for testing. In grading, this file might be swapped out with another file with different options. Make sure you consider that UNIX- or Windows-style line endings might be used.

Yet another concern was our lack of ability to store information about any dietary restrictions, be they religious, medical, or based on ethical considerations.

10. **Requirement: Dietary restrictions.** We must be able to keep track of dietary restrictions for DisasterVictims. One person could have multiple dietary restrictions, such as "diabetic" and "kosher". Initially, we will only be able to offer:
    - AVML - Asian vegetarian meal
    - DBML - Diabetic meal
    - GFML - Gluten intolerant meal
    - KSML - Kosher meal
    - LSML - Low salt meal
    - MOML - Muslim meal
    - PFML - Peanut-free meal
    - VGML - Vegan meal
    - VJML - Vegetarian Jain meal

    Tip: Consider using an enumeration to implement this requirement.