The CS Majors

Adam Davis (gte459u)
Namit Bhatia (gte276u)
Nimish Gautam (gte671u)

CS Project: Milestone 2
The CS Majors

Adam Davis  gte459u
Nimish Gautam gte671u
Namit Bhatia  gte276u

CRC Analysis of objects:
The classes considered were Person, Family, Marriage, Population, and Query. The class that was rejected was Family. We rejected it because we decided to use the Person and Marriage classes to hold all of the information that would have been in Family, therefore rendering Family useless.

Two scenarios:
1.) A Person is added to the world. Person is responsible for checking to see if the class attribute population has been initialized yet. If it isn’t, Person must create an instance the Population class for the population attribute. Next, the Person class must add itself to the Population object and give the Query class a reference to the Population object.
2.) When a relationship is defined for the a Person object, Person must modify its own attribute corresponding to this, then it must see if there is an existing Family object that includes the relative just referred to. If there is, the Person adds itself to that Family in the proper fashion, otherwise, the Person creates a new Family instance, which is given to the Population object, and adds itself and its relative to that new Family instance in the proper fashion.

Documentation of a test plan based on the Equivalence Partitioning approach.

In equivalence partitioning, the possible cases of input are separated into different categories which are then represented by specific test cases.

To test the Person class, we must think of all the possible categories of input that might show that there are errors in the code. These will be the equivalence partitions of our test plan. There are three main types of errors that the Person class needs to check for in its information: missing vital information, mirror and transitive information, and contradictions and conflicts. To test these three responsibility categories of Person, we will make tests to test if Person reports each type of missing information, connects or reports each type of mirror and transitive information, and identifies each type of contradiction. The types of each category are listed below:

1.) Missing
   a.) Personal: Name, birthdate and location, deathdate and location.
   b.) Familial: Children with no spouse, missing parents.

2.) Mirror of Transitive
   a.) Marriage and divorce dates.
   b.) Sibling relationships.
   c.) Parents of siblings will probably be the same.

3.) Contradictions
a.) Death of the Person should occur after birth, marriages, divorces, and birth dates of children (fathers can’t die more than nine months before their children are born).

b.) Birth of the Person should occur before marriages and birth dates of children.

c.) Marriage dates should occur before divorce dates of the same marriage.

Every case mentioned above will be considered an equivalence partition and a test case will be made for it which represents that partition. For instance, point 3b contains two different partitions, birth should occur before marriage and birth should occur before children are born. For each of these partitions we will make a case where a Person class is given information that breaks this rule and then run “PrintIt” on <Person> check to see if the code accounts for that rule and prints the proper response. If it doesn’t, we will consider that test failed.

To test the Query class we must think of all the possible types of queries that we can make, make that query given a database of existing people, and check to see if the result is correct based on the requirements of the query. Below are the types of queries that we need to test:

1.) Queries based on relationships.
   a.) Name: givenName, surName.
   b.) Gender: isMale, isFemale.
   c.) Misc: hasAlias, hasSibling: person, hasChild: person, hasParent: person, married: person, born: date, bornIn: ‘location’, died: date, diedIn: ‘location’, hasInfo: x as: y.

2.) Queries based on global information.
   a.) livedIn: ‘place’ (should return all people who ever lived in that place).
   b.) livedOn: date (should return all people who lived on that date).
   c.) generalSearch: ‘string’ (should return all people who have that string in any information corresponding to them).

Description of Each Class and Methods

- **Person:**
  - Holds information about a Person and all of their relationships with other persons.
  - **Methods:**
    - check – Checks the validity of every person in this person’s family tree and prints out a string reporting what is missing, what information was mirrored, and what is conflicting.
    - isMale – Sets this person’s gender to be male.
    - isFemale – Sets this person’s gender to be female.
    - givenName: <string> -- Modifier for given name.
    - surName: <string> -- Modifier for surname.
    - addAlias: <string> -- Adds an alias to the person’s list (OrderedCollection) of aliases.
- **hasChild**: `<person>` -- Adds a child to the person’s list of children (OrderedCollection).
- **hasChild**: `<person>` with: `<person>` -- Same as above except adds the second person to a list of would-be parents called *spouses*.
- **hasFather**: `<person>` -- Sets the person’s father.
- **hasMother**: `<person>` -- Sets the person’s mother.
- **hasSibling**: `<person>` -- Adds a sibling to the person’s list of siblings.
- **married**: `<person>` -- Add a new marriage object to the person’s list of marriages.
- **married**: `<person>` on: `<date>` -- Same as above except includes date of marriage.
- **married**: `<person>` on: `<date>` divorced: `<date>` -- Same as above except includes date of marriage and divorce.
- **record**: `<record>` as: `<info>` -- Add this record “definition” to the Dictionary called *recordMisc*.
- **born**: `<date>` location: `<place>` -- Modifier for birthTime and birthPlace.
- **died**: `<date>` location: `<place>` -- Modifier for deathTime and deathPlace.
- **allRelatives** – Returns a collection of all the people related to this person (including the object that calls it).

**Marriage**:
- Holds the information about a marriage: date of marriage, date of divorce (if any), and the Person who is the spouse.

**Population**:
- This class contains two lists of objects (which are implemented as OrderedCollections), Persons and Families.
  - **Methods**:
    - **People**: Assessor for class (global) variable People.
    - **addPerson**: Adds a person to the collection, People.
    - **printPeople**: Prints out the names of all the people for testing purposes.
    - **resetPeople**: Sets the People variable to nil for testing purposes.

**Query**:
- This class is responsible for searching the entire population of people for certain information and returning the corresponding people in a list. All of its methods return a list of Person objects in the form of an OrderedCollection.
  - **Methods**:
    - **givenName**: `<string>` -- Returns all people who have that given name.
    - **surName**: `<string>` -- Returns all people who have that surname.
    - **isMale** -- Returns all people who are male.
    - **isFemale** -- Returns all people who are female.
    - **hasChild**: `<person>` -- Returns all people whose child this is.
    - **hasChild**: `<person>` with: `<person>` -- Returns all people who have had that child with that person.
- **hasParent: <person>** -- Returns all persons that have that person as a father or mother.
- **hasSibling: <person>** -- Returns all people who have that sibling.
- **married: <person>** -- Returns all people who married that person.
- **married: <person> on: <date>** -- Same as above except includes date of marriage.
- **born: <date>** -- Returns all people who were born on that date.
- **died: <date>** -- Returns all people who died on that date.
- **bornIn: <place>** -- Returns all people who were born in that place.
- **livedIn: <place>** -- Returns all people who lived in that place at anytime in their life. This is accomplished by searching the birth place, death place and general information.
- **livedOn: <date>** -- Returns all people who lived on this date. This is accomplished by making sure that their birth date is before than this date and their death date is after it.
- **generalSearch: <string>** -- Returns all people who have that string existing anywhere in their general information records.