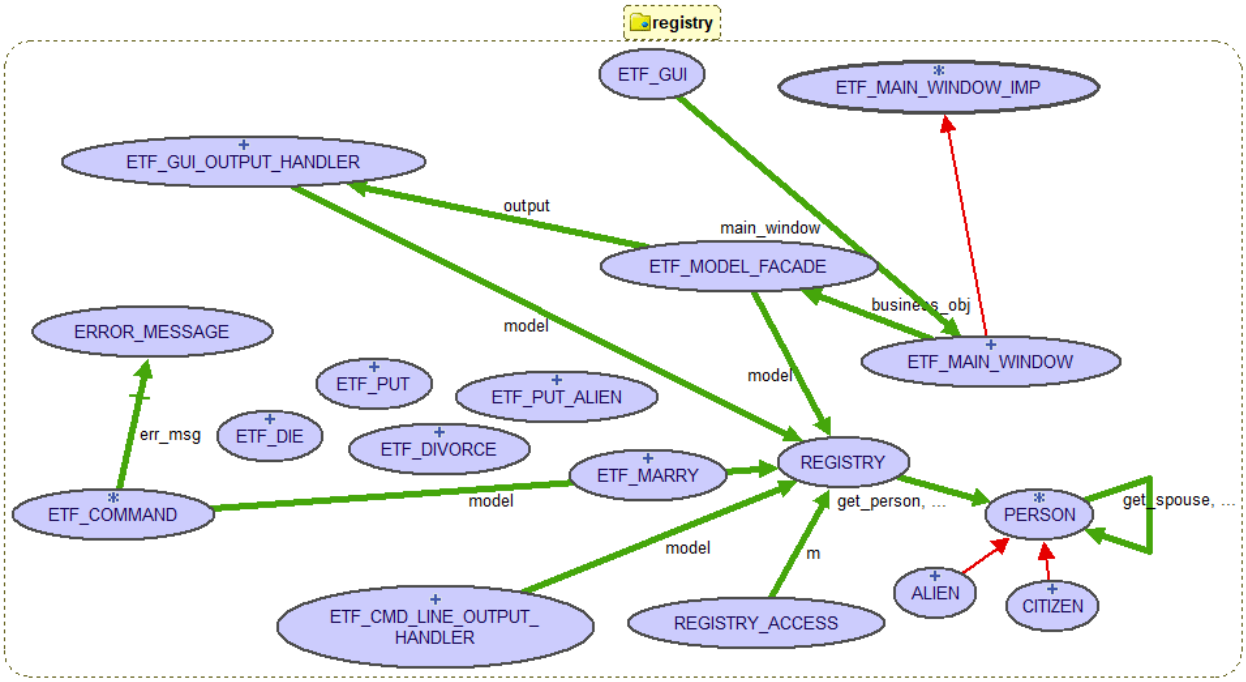
Joshua Phillip – EECS 3311 – Lab3 Report – 207961907 Prism: jep4444

I hereby declare that the work done in this lab is my own work and is not a duplicate of anyone else's work.

For me, the hardest part of the lab was filtering through the documentation and deciphering what we were supposed to do in the lab. I also had some trouble with the ETF GUI in Windows crashing due to use of void leading me to abandon void in my design(these problems did not persist when testing on CentOS but since I did most of my work on Windows, it seemed easier to develop a design that worked there). As for time spent on the lab, I lost track but likely 10+ between all the different parts, debugging, testing, etc.

A side note, I included the Person contract on top of Registry because I wanted to include the invariants that dictated what constitues a valid person in my design. The pre-conditions of the various methods in Registry are there to ensure correctness of the person invariants. I realise this puts my report at 4 pages, you can ignore the additon if you wish.



If we wanted to extend the business logic to allow for various types of non-citizens, we could add a feature to the Alien class that deals with citizenship status. From there we could add that field to the constructor to list one of the options (visitor, landed immigrant, refugee, etc.). Better yet, Alien could be made deferred and we could have subclasses for each that inherit from Alien and add the additional features in their own constructors.

```
class interface
	REGISTRY

create {REGISTRY_ACCESS}
	make

feature -- model operations

	reset
			-- Reset model state.

feature -- Model Commands

	put (id: INTEGER_64; name: STRING_8; dob: TUPLE [d: INTEGER_64; m: INTEGER_64; y: INTEGER_64])
			--adds a citizen to the registry
		require
			err_id_nonpositive: id > 0
			err_id_taken: not valid_id (id)
			err_name_start: (not name.is_empty) and then name [1].is_alpha
			err_invalid_date: is_valid_date (dob.d, dob.m, dob.y)
		ensure
			correct_size: persons.count = (old persons.count + 1)
			new_key_inserted: persons.has_key (id) /= old persons.has (id)

	put_alien (id: INTEGER_64; name: STRING_8;
	dob: TUPLE [d: INTEGER_64; m: INTEGER_64; y: INTEGER_64]; country: STRING_8)
			--adds a non-citizen to the registry
		require
			err_id_nonpositive: id > 0
			err_id_taken: not valid_id (id)
			err_name_start: (not name.is_empty) and then name [1].is_alpha
			err_invalid_date: is_valid_date (dob.d, dob.m, dob.y)
			err_country_start: (not country.is_empty) and then country [1].is_alpha
		ensure
			correct_size: persons.count = (old persons.count + 1)
			new_key_inserted: persons.has_key (id) /= old persons.has_key (id)
```

```
	marry (id1, id2: INTEGER_64; date: TUPLE [d: INTEGER_64; m: INTEGER_64; y: INTEGER_64])
			--marries two people in the registry
		require
			err_id_same: id1 /= id2
			err_id_nonpositive: id1 > 0 and then id2 > 0
			err_invalid_date: is_valid_date (date.d, date.m, date.y)
			err_id_unused: valid_id (id1) and then valid_id (id2)
			err_marry: not (get_person (id1).is_married or else
			get_person(id2).is_married)
			err_dead: get_person (id1).is_alive and then get_person (id2).is_alive
			err_under_18: (get_person (id1).get_married_age (date) > 17) and then
			(get_person (id2).get_married_age (date) > 17)
		ensure
			spouse1_is_person2: get_person (id1).get_spouse = get_person (id2)
			spouse2_is_person1: get_person (id2).get_spouse = get_person (id1)

	divorce (id1, id2: INTEGER_64)
			--divorces two people in the registry
		require
			err_id_same: id1 /= id2
			err_id_nonpositive: id1 > 0 and then id2 > 0
			err_id_unused: valid_id (id1) and then valid_id (id2)
			err_divorce: get_person (id1).get_spouse = get_person (id2) and then
			get_person (id2).get_spouse = get_person (id1)
		ensure
			person1_is_single: get_person (id1).get_spouse = get_person (id1)
			person2_is_single: get_person (id2).get_spouse = get_person (id2)

	die (id: INTEGER_64)
			--sets a person in the registry to deceased
		require
			err_id_nonpositive: id > 0
			err_id_unused: valid_id (id)
			err_dead_already: get_person (id).is_alive
		ensure
			not_alive: not get_person (id).is_alive
			spouse_not_married: not (old get_person (id)).get_spouse.is_married

	set_err_message (s: STRING_8)
			--sets the error message

feature -- Model Queries

	get_list: SORTED_TWO_WAY_LIST [PERSON]
			--returns the registry as a sorted list
		ensure
			sizes_match: persons.count = Result.count
			registry_unchanged: persons ~ old persons

	is_valid_date (date: TUPLE [d: INTEGER_64; m: INTEGER_64; y: INTEGER_64]): BOOLEAN
			--checks whether a date as a tuple is valid

	get_person (id: INTEGER_64): PERSON
			--checks,given an id, that the person is attached and returns the person
		ensure
			person_unchanged: persons.at (id) ~ old persons.at (id)

	valid_id (id: INTEGER_64): BOOLEAN
			--checks if the id is in the registry
		ensure
			person_unchanged: persons.at (id) ~ old persons.at (id)

feature -- queries

	out: STRING_8
			--sets the output for ETF

invariant
		persons.count >= 0

end -- class REGISTRYdeferred class interface
```

```
        PERSON

feature -- queries

        get_id: INTEGER_32
                        --returns the ID of the person

        get_name: STRING_8
                        --returns the name of the person

        get_country: STRING_8
                        --returns the nationality of the person

        date_to_string (date: TUPLE [d: INTEGER_64; m: INTEGER_64; y: INTEGER_64]): STRING_8
                        --converts a date as a tuple to a string

        get_dob: STRING_8
                        -- returns the birthday of the perspn

        get_married_since: STRING_8
                        --returns the wedding date of the person (returns 00-00-0000 for unmarried)

        get_spouse: PERSON
                        --returns the spouse of the person (returns self if unmarried)

        is_married: BOOLEAN
                        --returns the marriage status of the person

        is_alive: BOOLEAN
                        --returns whether the person is alive

        get_married_age (date: TUPLE [d: INTEGER_64; m: INTEGER_64; y: INTEGER_64]): INTEGER_64
                        --returns age at date of marriage(returns a negative value if unmarried)

        is_less alias "<" (other: like Current): BOOLEAN
                        --used to sort users, first by name, then by id

feature --commands

        set_married (p: PERSON; date: TUPLE [d: INTEGER_64; m: INTEGER_64; y: INTEGER_64])
                        --sets the persons spouse and date of marriage

        set_divorced
                        --sets the person to single and sets marriage date to 0000-00-00

        set_death
                        --sets the person to deceased

invariant
        spouse_is_alive: Current.get_spouse.is_alive = Current.is_alive
        married_person_is_alive: Current.is_married = Current.is_alive or else not Current.is_married
        married_person_over_18: Current.is_married = (Current.get_married_age (married_since) >= 18)
        positive_id: Current.get_id > 0
        valid_name: not (Current.get_name.is_empty) and then name [1].is_alpha
        valid_country: not (Current.get_country.is_empty) and then name [1].is_alpha

end -- class PERSON
```