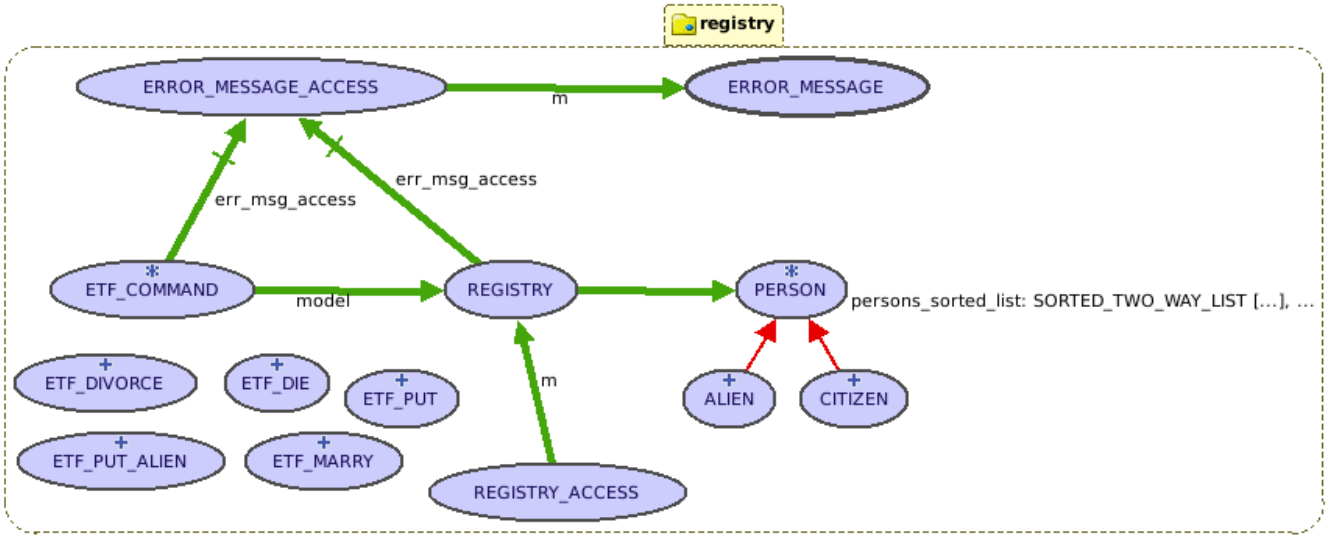
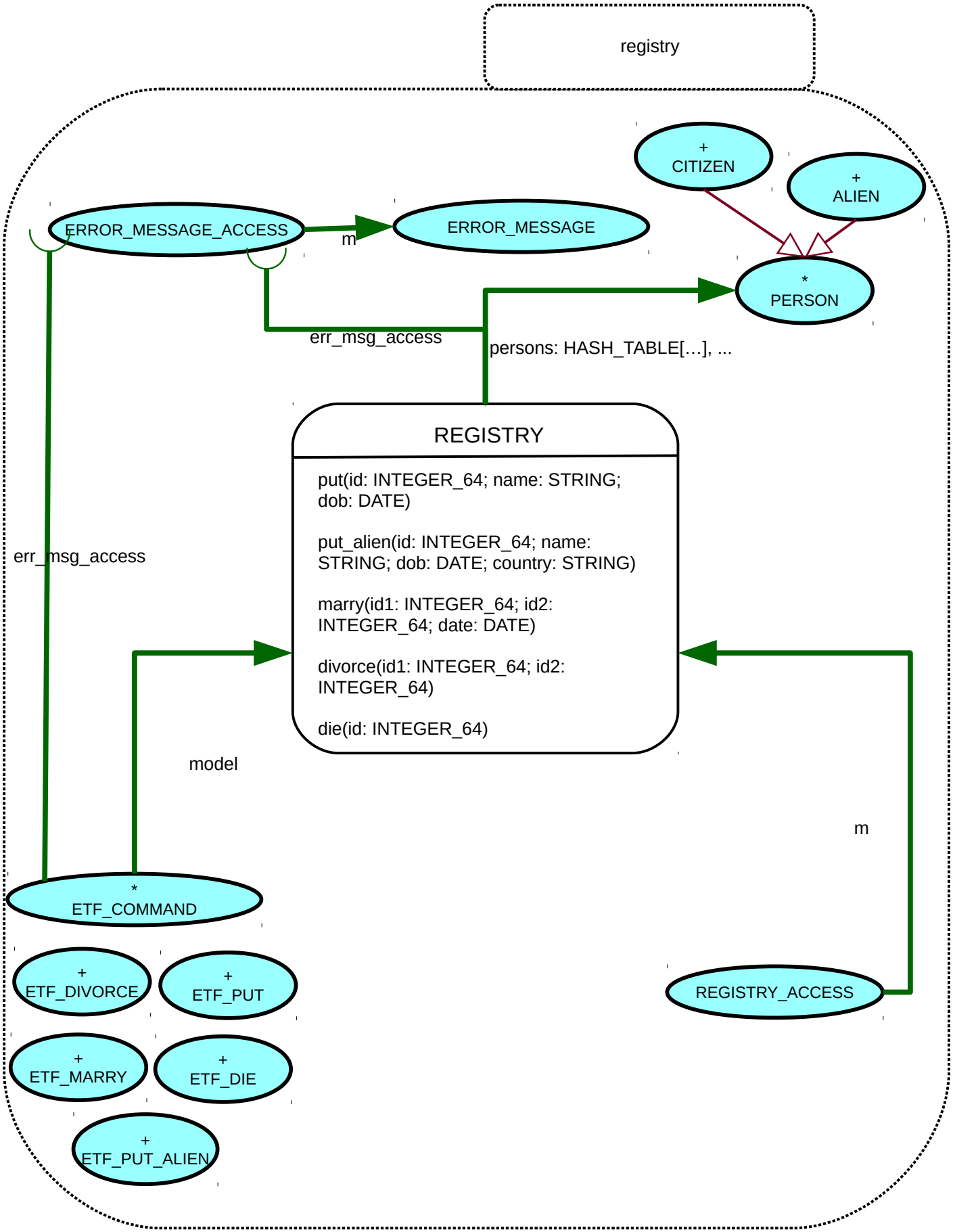
EECS3311-Lab3-Report

1.
Name: Mikhail Gindin
Login: mikegin

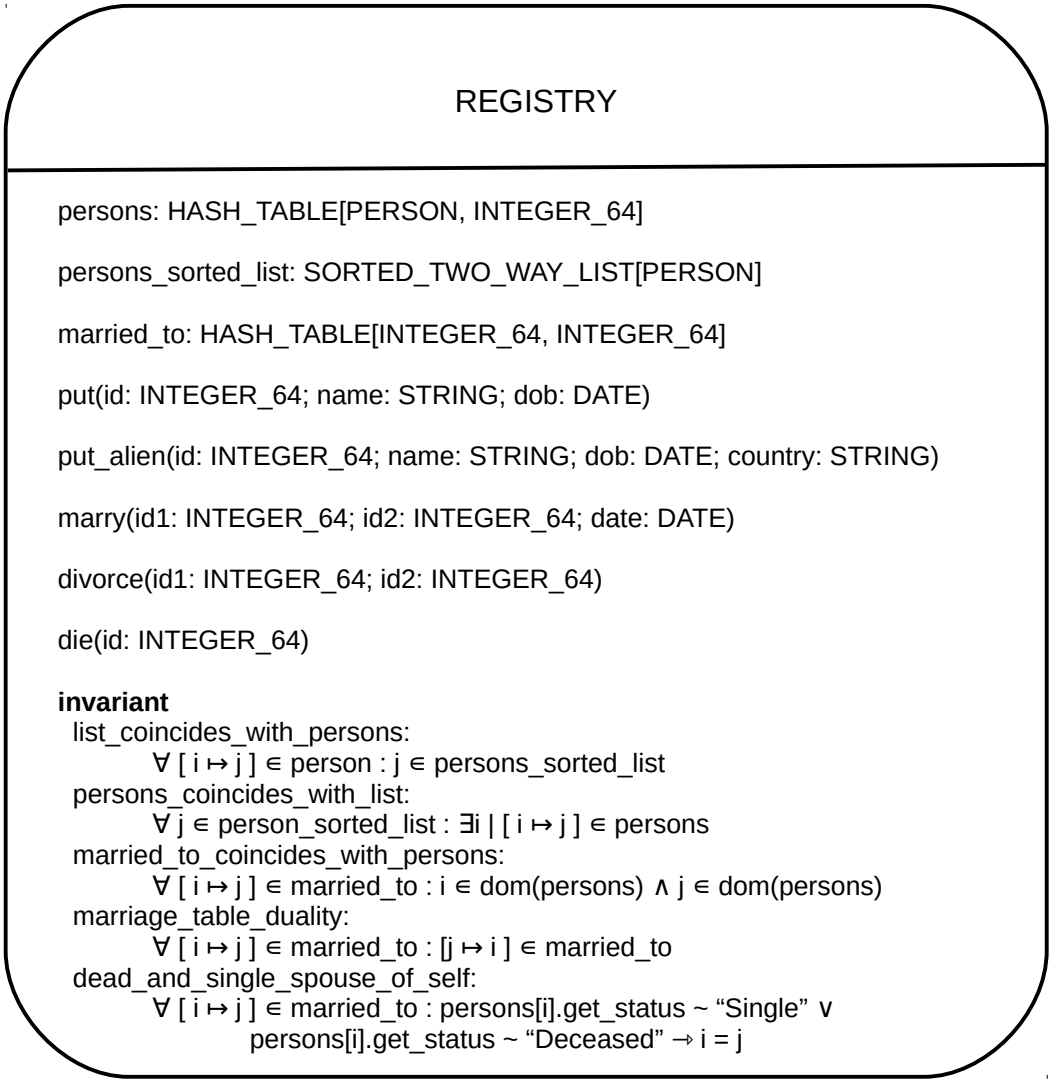
2. I completed the whole lab: YES

3.




registry
+
CITIZEN
+
ALIEN
ERROR_MESSAGE_ACCESS
m
ERROR_MESSAGE
*
PERSON
err_msg_access
persons: HASH_TABLE[…], ...
REGISTRY
put(id: INTEGER_64; name: STRING;
dob: DATE)
put_alien(id: INTEGER_64; name:
STRING; dob: DATE; country: STRING)
marry(id1: INTEGER_64; id2:
INTEGER_64; date: DATE)
divorce(id1: INTEGER_64; id2:
INTEGER_64)
die(id: INTEGER_64)
err_msg_access
model
m
*
ETF_COMMAND
+
ETF_DIVORCE
+
ETF_PUT
+
ETF_MARRY
+
ETF_DIE
+
ETF_PUT_ALIEN
REGISTRY_ACCESS

4.


REGISTRY
persons: HASH_TABLE[PERSON, INTEGER_64]
persons_sorted_list: SORTED_TWO_WAY_LIST[PERSON]
married_to: HASH_TABLE[INTEGER_64, INTEGER_64]
put(id: INTEGER_64; name: STRING; dob: DATE)
put_alien(id: INTEGER_64; name: STRING; dob: DATE; country: STRING)
marry(id1: INTEGER_64; id2: INTEGER_64; date: DATE)
divorce(id1: INTEGER_64; id2: INTEGER_64)
die(id: INTEGER_64)
invariant
list_coincides_with_persons:
∀ [i ↦ j] ∈ person : j ∈ persons_sorted_list
persons_coincides_with_list:
∀ j ∈ person_sorted_list : ∃i | [i ↦ j] ∈ persons
married_to_coincides_with_persons:
∀ [i ↦ j] ∈ married_to : i ∈ dom(persons) ∧ j ∈ dom(persons)
marriage_table_duality:
∀ [i ↦ j] ∈ married_to : [j ↦ i] ∈ married_to
dead_and_single_spouse_of_self:
∀ [i ↦ j] ∈ married_to : persons[i].get_status ~ “Single” ∨
persons[i].get_status ~ “Deceased” ⇻ i = j

Information Hiding

a) I have done a below average job of information hiding for this class.

b) Attributes like persons, persons_sorted_list, and married_to are all public and can easily be accessed by other classes. This can be changed by changing these attributes to be exported to {NONE} and creating public get queries like are_married or person_exists in REGISTRY that have access to needed information in these private attributes.

Single Responsibility Principle

a) I have done an average job of applying the SRP to this class.

b) This classes sole responsibility is keeping a registry of people (PERSON objects), and storing basic mapping information about who is married to whom. No other class in the program does this. However, if the requirements change for marriages and they become very complex, it would then would have been wiser to separate the marriage information into another class that would deal with that responsibility separately. Also output handling is very static and is tailored in the class to what is expected in the current UI. However if things change the classes own output would have to be formatted. Thus is would have been wiser to also have an output handler class to handle the outputs that tailor to each specific UI to this program.

5.

```
note
	description: "A default business model."
	author: "Jackie Wang"
	date: "$Date$"
	revision: "$Revision$"

class interface
	REGISTRY

create {REGISTRY_ACCESS}
	make

feature -- model attributes

	persons: HASH_TABLE [PERSON, INTEGER_64]

	persons_sorted_list: SORTED_TWO_WAY_LIST [PERSON]

	married_to: HASH_TABLE [INTEGER_64, INTEGER_64]

	message: STRING_8

	err_msg_access: ERROR_MESSAGE_ACCESS

feature -- model operations

	reset
			-- Reset model state.

	set_message (msg: STRING_8)

	put (id: INTEGER_64; name: STRING_8; dob: DATE)
		require
			positive_id: check_id_positive (id)
```

```
                id_not_take: not persons.has (id)
                valid_name: not name.is_empty and then name.item (1).is_alpha
                valid_date: is_valid_date2 (dob)
        ensure
                registered: attached persons [id] as p and then (p.get_name ~ name and
                        p.get_citizenship ~ "Canada" and p.get_dob ~ dob and p.get_id = id and
                        p.get_status ~ "Single")
                increased_person_count: persons.count = old persons.count + 1
                others_unchanged_in_table: persons_unchanged_other_than (id, old
                        persons.deep_twin)
                increased_person_sorted_list_count: persons_sorted_list.count = old
                        persons_sorted_list.count + 1
                others_unchanged_in_list: persons_sorted_list_unchanged_other_than (id, old
                        persons_sorted_list.deep_twin)
                increased_married_to_count: married_to.count = old married_to.count + 1

put_alien (id: INTEGER_64; name: STRING_8; dob: DATE; country: STRING_8)
        require
                positive_id: check_id_positive (id)
                id_not_take: not persons.has (id)
                valid_name: not name.is_empty and then name.item (1).is_alpha
                valid_date: is_valid_date2 (dob)
                valid_country: not country.is_empty and then country.item (1).is_alpha
        ensure
                registered: attached persons [id] as p and then (p.get_name ~ name and
                        p.get_citizenship ~ country and p.get_dob ~ dob and p.get_id = id and
                        p.get_status ~ "Single")
                increased_person_count: persons.count = old persons.count + 1
                others_unchanged_in_table: persons_unchanged_other_than (id, old
                        persons.deep_twin)
                increased_person_sorted_list_count: persons_sorted_list.count = old
                        persons_sorted_list.count + 1
                others_unchanged_in_list: persons_sorted_list_unchanged_other_than (id, old
                        persons_sorted_list.deep_twin)
                increased_married_to_count: married_to.count = old married_to.count + 1

marry (id1: INTEGER_64; id2: INTEGER_64; date: DATE)
        require
                different_ids: id1 /= id2
                positive_ids: check_id_positive (id1) and check_id_positive (id2)
                valid_date: is_valid_date2 (date)
                persons_exist: persons.has (id1) and persons.has (id2)
                valid_marriage: valid_marriage (id1, id2, date)
        ensure
                are_married: married_to [id1] = id2 and married_to [id2] = id1
                correct_status: (attached persons [id1] as p1 and attached persons [id2] as p2)
                        and then (p1.get_status /~ "Single" and p1.get_status /~ "Deceased" and
                        p2.get_status /~ "Single" and p2.get_status /~ "Deceased")

divorce (id1: INTEGER_64; id2: INTEGER_64)
        require
                different_ids: id1 /= id2
                positive_ids: check_id_positive (id1) and check_id_positive (id2)
                persons_exist: persons.has (id1) and persons.has (id2)
                persons_are_married: married_to [id1] = id2
        ensure
                are_divorced: married_to [id1] = id1 and married_to [id2] = id2
                correct_status: (attached persons [id1] as p1 and attached persons [id2] as p2)
                        and then (p1.get_status ~ "Single" and p2.get_status ~ "Single")

die (id: INTEGER_64)
        require
                positive_id: check_id_positive (id)
                person_exists: persons.has (id)
                not_dead: not is_dead (id)
        ensure
                person_count_unchanged: persons.count = old persons.count
                persons_sorted_list_count_unchanged: persons_sorted_list.count = old
```

```
                        persons_sorted_list.count
                    others_unchanged_in_table: persons_unchanged_other_than (id, old
                        persons.deep_twin)
                    others_unchanged_in_list: persons_sorted_list_unchanged_other_than (id, old
                        persons_sorted_list.deep_twin)
                    person_is_dead: is_dead (id)
                    spouse_is_single_xor_self_is_dead: attached persons [old married_to [id]] as s
                            and then (s.get_status ~ "Single" xor s.get_status ~ "Deceased")

feature -- queries

        persons_unchanged_other_than (id: INTEGER_64; old_persons: like persons): BOOLEAN
                    -- Are persons other than `person[id]' unchanged?
                ensure
                        Result = across
                                persons as p
                        all
                                p.key /= id implies old_persons [p.key] ~ persons [p.key]
                        end

        persons_sorted_list_unchanged_other_than (id: INTEGER_64; old_persons_sorted_list: like
                persons_sorted_list): BOOLEAN
                        -- Are persons_sorted_list unchanged other than the the addition of
                                `persons[id]'?
                ensure
                        Result = across
                                persons_sorted_list as p
                        all
                                p.item.get_id /= id implies old_persons_sorted_list.has (p.item)
                        end

        check_id_positive (id: INTEGER_64): BOOLEAN

        is_valid_date (dobastuple: TUPLE [d: INTEGER_64; m: INTEGER_64; y: INTEGER_64]): BOOLEAN

        is_valid_date2 (dob: DATE): BOOLEAN

        is_dead (id: INTEGER_64): BOOLEAN

        valid_marriage (id1: INTEGER_64; id2: INTEGER_64; m_date: DATE): BOOLEAN

        out: STRING_8
                    -- New string containing terse printable representation
                    -- of current object

invariant
        list_coincides_with_persons: across
                    persons as p
                all
                    persons_sorted_list.has (p.item)
                end
        persons_coincides_with_list: across
                    persons_sorted_list as p
                all
                    attached persons [p.item.get_id] and then persons [p.item.get_id] ~ p.item
                end
        married_to_coincides_with_persons: across
                    married_to as m
                all
                    persons.has_key (m.key) and persons.has (m.item)
                end
        marriage_table_duality: across
                    married_to as l
                all
                    l.key = married_to [married_to [l.key]]
                end
        dead_and_single_spouse_of_self: across
```

```
                persons as p
        all
                (p.item.get_status ~ "Single" or p.item.get_status ~ "Deceased") implies
                        married_to [p.item.get_id] = p.item.get_id
        end

end -- class REGISTRY
```