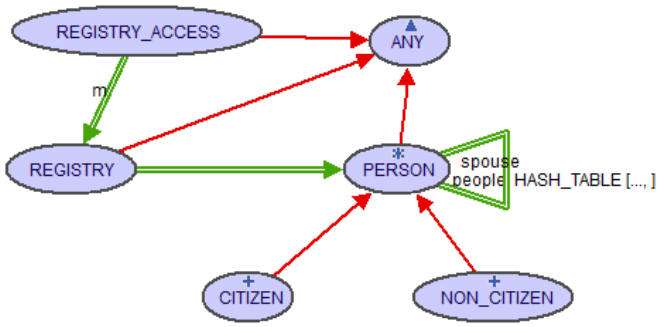
Name: Tao Wang Student#: 214672547 Prism Login:kevin89

I guarantee that I finish all the work individually

The hardest part of this lab is to familiarize myself using ETF

The time I spent on this lab is approximately 11 hours



For extending business logic to various non-citizens, I would use inheritance. For example, create VISTOR class, then make it inherit from NON_CITIZEN class.

```
**************************************************************************************
note
        description: "A person model."
        author: ""
        date: "$Date$"
        revision: "$Revision$"

deferred class interface
        PERSON

feature -- atrributs

        name: STRING_8

        dob: DATE
                        -- date of birth

        country: STRING_8

        alive: BOOLEAN
                        -- dead or alive

        spouse: detachable PERSON

        dom: detachable DATE
                        -- date of marriage

feature -- queries

        is_alive: BOOLEAN
                        -- is alive?

        can_marry (other: PERSON; y: INTEGER_32; m: INTEGER_32; d: INTEGER_32): BOOLEAN
                        -- can marry on a specified day?
                require
                        other_is_not_void: other /= Void

        is_single: BOOLEAN
                        -- is single?

feature -- commands

        marry (other: PERSON; y: INTEGER_32; m: INTEGER_32; d: INTEGER_32)
                        -- marry the other person
                require
                        other_exits: other /= Void
                        both_alive: is_alive and other.is_alive
                        can_marry: can_marry (other, y, m, d)
                ensure
                        spouse_and_dom_attached: attached spouse and attached dom

        divorce
                        -- end the marriage
```

```
                require
                        is_married: not is_single
                ensure
                        is_single: is_single
                        old_spouse_exists: attached (old spouse) as os
                        old_spouse_is_single: os.is_single

        die
                        -- R.I.P
                ensure
                        not_alive: alive = False
                        is_single: is_single

        set_name (n: STRING_8)
                        -- Set the person's name
                require
                        name_is_not_void_and_empty: n /= Void and not n.is_empty
                ensure
                                name ~ n

        set_dob (y, m, d: INTEGER_32)
                        -- set the date of birth
                ensure
                        dob_is_set_correctly: dob.is_correct_date (y, m, d)

        set_country (c: STRING_8)
                        -- set the citizen info
                require
                        c_is_not_void_and_empty: c /= Void and not c.is_empty

        set_alive
                        -- alive!

invariant
        name_is_not_void_and_empty: name /= Void and not name.is_empty
        date_of_birth_is_not_void: dob /= Void
        country_is_not_void_and_empty: country /= Void and not country.is_empty
        if_dead_not_married: not alive implies spouse = Void and dom = Void
        married: (attached spouse as s and attached dom as d) implies s.alive and Current.alive and s.spouse =
Current and Current.dom ~ s.dom

end -- class PERSON

**********************************************************************************************

note
        description: "A citizen model."
        author: ""
        date: "$Date$"
        revision: "$Revision$"

class interface
        CITIZEN

create
        make

end -- class CITIZEN

**********************************************************************************************

note
        description: "A non-citizen model."
        author: ""
        date: "$Date$"
        revision: "$Revision$"

class interface
        NON_CITIZEN

create
        make

end -- class NON_CITIZEN

**********************************************************************************************

note
        description: "A registry model."
        author:
        date: "$Date$"
        revision: "$Revision$"

class interface
        REGISTRY
```

```
create {REGISTRY_ACCESS, REGISTRY_TESTS}
        make

feature -- attributs

        people: HASH_TABLE [PERSON, INTEGER_32]

        error_message: STRING_8

feature -- commands

        put (id: INTEGER_32; name: STRING_8; y: INTEGER_32; m: INTEGER_32; d: INTEGER_32)
                        -- Add one citizen into the registry
                require
                        passed_id_is_not_used: not people.has (id)
                ensure
                        id_added_successfully: people.has (id)
                        person_added_successfully: attached people [id] as p
                        person_set_correctly: p.name ~ name and p.dob ~ create {DATE}.make (y, m, d)

        put_alien (id: INTEGER_32; name: STRING_8; country: STRING_8; y: INTEGER_32; m: INTEGER_32; d:
INTEGER_32)
                        -- Add one non-citizen into the registry
                require
                        passed_id_is_not_used: not people.has (id)
                ensure
                        id_added_successfully: people.has (id)
                        person_added_successfully: attached people [id] as p
                        person_set_correctly: p.name ~ name and p.dob ~ create {DATE}.make (y, m, d) and
p.country ~ country

        marry (id1: INTEGER_32; id2: INTEGER_32; y: INTEGER_32; m: INTEGER_32; d: INTEGER_32)
                        -- the person with id1 marries the person with id2
                require
                        id1_can_marry_id2_on_passed_date: marriageable (id1, id2, y, m, d)
                ensure
                                attached people [id1] as p1
                                attached people [id2] as p2
                        married: p1.spouse = p2 and p2.spouse = p1 and p1.dom ~ p2.dom

        divorce (id1: INTEGER_32; id2: INTEGER_32)
                        -- divorce id1 and id2
                require
                        id1_id2_can_divorce: divorceable (id1, id2)
                ensure
                                attached people [id1] as p1
                                attached people [id2] as p2
                        divorced: p1.is_single and p2.is_single

        die (id: INTEGER_32)
                require
                        id_nonnegative_and_used_and_is_alive: dieable (id)
                ensure
                                attached people [id] as p
                        id_is_dead: not p.is_alive

        set_error_message (m: STRING_8)
                require
                        not_void_and_empty: m /= Void and not m.is_empty

feature -- model operations

        default_update
                        -- Perform update to the model state.

        reset
                        -- Reset model state.

feature -- preconditions

        marriageable (id1: INTEGER_32; id2: INTEGER_32; y: INTEGER_32; m: INTEGER_32; d: INTEGER_32): BOOLEAN
                        -- Whether two people with passed ids can marry each other

        divorceable (id1: INTEGER_32; id2: INTEGER_32): BOOLEAN
                        -- Whether id1 and id2 can be divorced

        dieable (id: INTEGER_32): BOOLEAN

feature -- queries

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

end -- class REGISTRY
```