Lab 3 - Registry

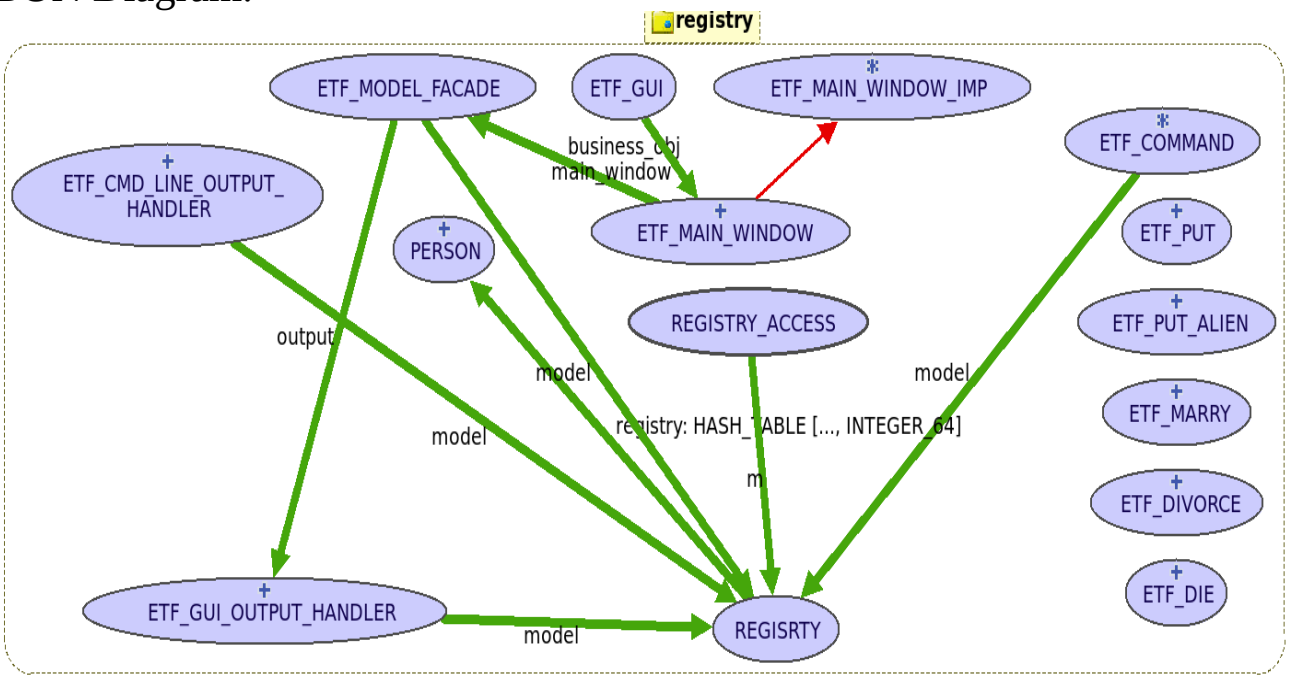
EECS 3311 – Software Design.

Rajkumar Balakrishnan Lakshmi - 213141197

20/02/18

I acknowledge that all the work done for this lab was my individual work. The hardest part of this lab was to get used to the etf and also learning how to generate and create bon diagram. My model can be easily extended and new classes like visitors, international students, etc. can be added to the business logic. All that has to be done in that new class is that it inherits from the class PERSON and any additional required attributes and their setter methods needs to be added to it and the rest is taken care by the registry and person classes, which basically registers new entry to the registry hash table. I have attached the text view of the PERSON and REGISTRY classes with this report.

BON Diagram:



Automatic generation produced by ISE Eiffel

Classes Clusters Cluster hierarchy Chart Relations Text Go to: regisrty

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

class
	REGISRTY

inherit
	ANY
		redefine
			out
		end

create {REGISTRY_ACCESS, ES_TEST}
	make

feature {NONE} -- Initialization

	make
			-- Initialization for Current.
		do
			create registry.make (10)
			registry.compare_objects
		end

feature -- model attributes

	registry: HASH_TABLE [PERSON, INTEGER_64]

feature -- model operations

	default_update
			-- Perform update to the model state.
		do
		end

	reset
			-- Reset model state.
		do
			make
		end

feature {ETF_COMMAND} -- Report

	error_msg: STRING_8
		attribute
			create Result.make_empty
			Result := "ok"
		end

	err_id_nonpositive: STRING_8
		attribute
			Result := "id must be positive"
		end

	err_id_unused: STRING_8
		attribute
			Result := "id not identified with a person in database"
		end

	err_id_same: STRING_8
		attribute
			Result := "ids must be different"
		end

	err_id_taken: STRING_8
		attribute
			Result := "id already taken"
		end

	err_name_start: STRING_8
		attribute
			Result := "name must start with A-Z or a-z"
		end

	err_country_start: STRING_8
		attribute
			Result := "country must start with A-Z or a-z"
		end

	err_invalid_date: STRING_8
		attribute
			Result := "not a valid date in 1900..3000"
		end

	err_marry: STRING_8
		attribute
			Result := "proposed marriage invalid"
		end

	err_divorce: STRING_8
		attribute
			Result := "these are not married"
		end

	err_dead_already: STRING_8
		attribute
			Result := "person with that id already dead"
		end

	err_success: STRING_8
		attribute
			Result := "ok"
		end

	set_err (err: STRING_8)
		do
			error_msg := err
		end

feature -- myfeature

	put (id: INTEGER_64; name1: STRING_8; dob: TUPLE [d: INTEGER_64; m: INTEGER_64; y: INTEGER_64])
		require
			id_greater_than_zero: id > 0
			id_not_taken_already: not registry.has (id)
```

```
                name_start_proper: name1.at (1).is_alpha
                check_date_proper: dob.d <= 31 and dob.d >= 1 and dob.m <= 12 and dob.m >= 1 and dob.y <= 3000 and dob.y >= 1900
                check_no_of_days: is_day_ok (dob)
        local
                p: PERSON
        do
                create p.make
                p.set_name (name1)
                p.set_id (id)
                p.set_dob (dob)
                p.set_country ("Canada")
                p.set_status ("Single")
                registry.extend (p, id)
        end

put_alien (id: INTEGER_64; name1: STRING_8; dob: TUPLE [d: INTEGER_64; m: INTEGER_64; y: INTEGER_64]; country: STRING_8)
        require
                id_greater_than_zero: id > 0
                id_not_taken_already: not registry.has (id)
                name_start_proper: name1.at (1).is_alpha
                country_start_proper: country.at (1).is_alpha
                check_date_proper: dob.d <= 31 and dob.d >= 1 and dob.m <= 12 and dob.m >= 1 and dob.y <= 3000 and dob.y >= 1900
                check_no_of_days: is_day_ok (dob)
        local
                r: PERSON
        do
                create r.make
                r.set_name (name1)
                r.set_id (id)
                r.set_dob (dob)
                r.set_country (country)
                r.set_status ("Single")
                registry.extend (r, id)
        end

marry (id1: INTEGER_64; id2: INTEGER_64; date: TUPLE [d: INTEGER_64; m: INTEGER_64; y: INTEGER_64])
        require
                id_greater_than_zero: id1 > 0 and id2 > 0
                id_must_be_different: id1 /~ id2
                id_not_taken_already: registry.has (id1) and registry.has (id2)
                legal_age_of_pair: is_legal_age (id1, id2)
                marriage_is_valid: is_marriage_valid (id1, id2)
                check_date_proper: date.d <= 31 and date.d >= 1 and date.m <= 12 and date.m >= 1 and date.y <= 3000 and date.y >= 1900
                check_no_of_days: is_day_ok (date)
        local
                person1: PERSON
                person2: PERSON
                mydate: STRING_8
        do
                person1 := registry.item (id1)
                person2 := registry.item (id2)
                check
                        attached person1 as p1 and attached person2 as p2
                then
                        create mydate.make_empty
                        if date.m < 10 and date.d < 10 then
                                mydate.append ("[" + date.y.out + "-0" + date.m.out + "-0" + date.d.out + "]")
                        else
                                if date.d < 10 then
                                        mydate.append ("[" + date.y.out + "-" + date.m.out + "-0" + date.d.out + "]")
                                else
                                        if date.m < 10 then
                                                mydate.append ("[" + date.y.out + "-0" + date.m.out + "-" + date.d.out + "]")
                                        else
                                                mydate.append ("[" + date.y.out + "-" + date.m.out + "-" + date.d.out + "]")
                                        end
                                end
                        end
                        p1.set_spouse_id (id2)
                        p2.set_spouse_id (id1)
                        p1.set_status ("Spouse: " + p2.name + "," + id2.out + "," + mydate)
                        p2.set_status ("Spouse: " + p1.name + "," + id1.out + "," + mydate)
                        p1.set_spouse_name (p2.name)
                        p2.set_spouse_name (p1.name)
                        p1.set_marry_date (date)
                        p2.set_marry_date (date)
                end
        end

divorce (a_id1: INTEGER_64; a_id2: INTEGER_64)
        require
                id_greater_than_zero: a_id1 > 0 and a_id2 > 0
                id_must_be_different: a_id1 /~ a_id2
                id_not_taken_already: registry.has (a_id1) and registry.has (a_id2)
                divorce_is_valid: is_divorce_valid (a_id1, a_id2)
        local
                person1: PERSON
                person2: PERSON
                mydate: TUPLE [d: INTEGER_64; m: INTEGER_64; y: INTEGER_64]
        do
                person1 := registry.item (a_id1)
                person2 := registry.item (a_id2)
                create mydate.default_create
                check
                        attached person1 as p1 and attached person2 as p2
                then
                        p1.set_status ("Single")
                        p1.set_spouse_id (0)
                        p1.set_spouse_name ("")
                        p1.set_marry_date (mydate)
                        p2.set_status ("Single")
                        p2.set_spouse_id (0)
                        p2.set_spouse_name ("")
                        p2.set_marry_date (mydate)
                end
        end

die (id: INTEGER_64)
        require
                id_greater_than_zero: id > 0
                id_not_taken_already: registry.has (id)
                check_person_alive: is_person_dead (id)
        local
                person: PERSON
                person2: PERSON
                spid: INTEGER_64
                mydate: TUPLE [d: INTEGER_64; m: INTEGER_64; y: INTEGER_64]
        do
                person := registry.item (id)
                create mydate.default_create
                check
                        attached person as p
                then
                        if p.status /~ "Single" then
                                spid := p.spouse_id
                                person2 := registry.item (spid)
                                check
                                        attached person2 as p2
```

```
                                        then
                                                p2.set_status ("Single")
                                                p2.set_spouse_id (0)
                                                p2.set_spouse_name ("")
                                                p2.set_marry_date (mydate)
                                        end
                                end
                                p.set_status ("Deceased")
                        end
                end

        is_legal_age (id1: INTEGER_64; id2: INTEGER_64): BOOLEAN
                local
                        person1: PERSON
                        person2: PERSON
                        current_date: DATE_TIME
                do
                        person1 := registry.item (id1)
                        person2 := registry.item (id2)
                        check
                                attached person1 as p1 and attached person2 as p2
                        then
                                create current_date.make_now
                                if (current_date.year.to_integer_64 - p1.dob.y) < 18 or (current_date.year.to_integer_64 - p2.dob.y) < 18 then
                                        Result := False
                                else
                                        Result := True
                                end
                        end
                end

        is_person_dead (id1: INTEGER_64): BOOLEAN
                local
                        person1: PERSON
                do
                        person1 := registry.item (id1)
                        check
                                attached person1 as p1
                        then
                                if p1.status ~ "Deceased" then
                                        Result := False
                                else
                                        Result := True
                                end
                        end
                end

        is_marriage_valid (id1: INTEGER_64; id2: INTEGER_64): BOOLEAN
                local
                        person1: PERSON
                        person2: PERSON
                do
                        person1 := registry.item (id1)
                        person2 := registry.item (id2)
                        check
                                attached person1 as p1 and attached person2 as p2
                        then
                                if (p1.country /~ "Canada" and p2.country /~ "Canada") or (p1.status ~ "Deceased" or p2.status ~ "Deceased" or p1.status /~ "Single" or p2.status /~ "Single") then
                                        Result := False
                                else
                                        Result := True
                                end
                        end
                end

        is_divorce_valid (id1: INTEGER_64; id2: INTEGER_64): BOOLEAN
                local
                        person1: PERSON
                        person2: PERSON
                do
                        person1 := registry.item (id1)
                        person2 := registry.item (id2)
                        check
                                attached person1 as p1 and attached person2 as p2
                        then
                                if p1.status ~ "Single" or p1.status ~ "Deceased" or p1.spouse_id /~ id2 or p2.spouse_id /~ id1 or p2.status ~ "Single" or p2.status ~ "Deceased" then
                                        Result := False
                                else
                                        Result := True
                                end
                        end
                end

        is_day_ok (date: TUPLE [d: INTEGER_64; m: INTEGER_64; y: INTEGER_64]): BOOLEAN
                do
                        if (date.m ~ 1 or date.m ~ 3 or date.m ~ 5 or date.m ~ 7 or date.m ~ 8 or date.m ~ 10 or date.m ~ 12) and (date.d > 31 or date.d < 1) then
                                Result := False
                        else
                                if (date.m ~ 4 or date.m ~ 6 or date.m ~ 9 or date.m ~ 11) and (date.d >= 31 or date.d < 1) then
                                        Result := False
                                else
                                        if (date.m ~ 2) and is_leap_year (date.y) and (date.d > 29 or date.d < 1) then
                                                Result := False
                                        else
                                                if (date.m ~ 2) and not is_leap_year (date.y) and (date.d >= 29 or date.d < 1) then
                                                        Result := False
                                                else
                                                        Result := True
                                                end
                                        end
                                end
                        end
                end

        is_leap_year (y: INTEGER_64): BOOLEAN
                require
                        y > 1584
                do
                        if (y \\ 4) = 0 and ((y \\ 100) /= 0 or (y \\ 400) = 0) then
                                Result := True
                        end
                ensure
                        Result = (mod (y, 4) = 0 and mod (y, 400) /= 100 and mod (y, 400) /= 200 and mod (y, 400) /= 300)
                end

        mod (a, b: INTEGER_64): INTEGER_64
                do
                        Result := a \\ b
                end

feature -- queries

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

```
                id: INTEGER_64
                dob: TUPLE [d: INTEGER_64; m: INTEGER_64; y: INTEGER_64]
                country: STRING_8
                status: STRING_8
                date: STRING_8
                sorted_persons: SORTED_TWO_WAY_LIST [PERSON]
        do
                create sorted_persons.make
                sorted_persons.compare_objects
                across
                        registry as r
                loop
                        sorted_persons.extend (r.item)
                end
                create Result.make_from_string ("  ")
                Result.append (error_msg)
                across
                        sorted_persons as c
                loop
                        pname := c.item.name
                        id := c.item.id
                        dob := c.item.dob
                        country := c.item.country
                        status := c.item.status
                        create date.make_empty
                        check
                                attached dob as d
                        then
                                if d.m < 10 and d.d < 10 then
                                        date.append (d.y.out + "-0" + d.m.out + "-0" + d.d.out)
                                else
                                        if d.d < 10 then
                                                date.append (d.y.out + "-" + d.m.out + "-0" + d.d.out)
                                        else
                                                if d.m < 10 then
                                                        date.append (d.y.out + "-0" + d.m.out + "-" + d.d.out)
                                                else
                                                        date.append (d.y.out + "-" + d.m.out + "-" + d.d.out)
                                                end
                                        end
                                end
                        end
                        Result.append ("%N" + "  " + pname + "; ID: " + id.out + "; Born: " + date + "; Citizen: " + country + "; " + status)
                end
        end

end -- class REGISRTY
```

Classes Clusters Cluster hierarchy Chart Relations Text Go to: regisrty

Automatic generation produced by ISE Eiffel


```
note
        description: "Summary description for {REGISTER}."
        author: ""
        date: "$Date$"
        revision: "$Revision$"

class
        PERSON

inherit
        COMPARABLE
                redefine
                        is_less
                end

create {REGISRTY, ES_TEST}
        make

feature {NONE} -- Initialization

        make
                        -- Initialization for Current.
                do
                        create name.make_empty
                        create dob.default_create
                        create country.make_empty
                        create status.make_empty
                        create spouse_name.make_empty
                        create spouse_id.default_create
                        create marry_date.default_create
                end

feature -- model attributes

        name: STRING_8

        id: INTEGER_64

        dob: TUPLE [d: INTEGER_64; m: INTEGER_64; y: INTEGER_64]

        country: STRING_8

        status: STRING_8

        spouse_name: STRING_8

        spouse_id: INTEGER_64

        marry_date: TUPLE [d: INTEGER_64; m: INTEGER_64; y: INTEGER_64]
```

```
feature

	set_name (n: STRING_8)
		do
			name := n
		end

	set_id (i: INTEGER_64)
		do
			id := i
		end

	set_country (c: STRING_8)
		do
			country := c
		end

	set_status (st: STRING_8)
		do
			status := st
		end

	set_dob (d: TUPLE [d: INTEGER_64; m: INTEGER_64; y: INTEGER_64])
		do
			dob := d
		end

	set_spouse_name (spn: STRING_8)
		do
			spouse_name := spn
		end

	set_spouse_id (spid: INTEGER_64)
		do
			spouse_id := spid
		end

	set_marry_date (d: TUPLE [d: INTEGER_64; m: INTEGER_64; y: INTEGER_64])
		do
			marry_date := d
		end

	is_less alias "<" (other: like Current): BOOLEAN
			-- Is current object less than other?
		do
			if Current.name /~ other.name then
				Result := Current.name < other.name
			else
				Result := Current.id < other.id
			end
		end

invariant
	name_country_is_not_void: name /= Void and country /= Void
	dob_is_not_void: dob /= Void

end -- class PERSON
```

Classes Clusters Cluster Chart Relations Text Go to:

hierarchy

person

-- Generated by ISE Eiffel --
For more details: www.eiffel.com