Q5 Memoria

Tags

Q5: Ejercicio del Tema 3

Álvaro Valencia Villalón

Alba de la Torre Segato

Pablo Astudillo Fraga

Carla Serracant Guevara

Pablo Alarcón Carrión

Contexto:

En este ejercicio nos piden que a partir de un diagrama de clases de una comunidad en la que residen varias personas, implementemos varias restricciones y operaciones para modelar eventos importantes en la vida de los mismos. Dichas personas nacerán, fallecerán, se casarán y podrán divorciarse o enviudarse. El paso del tiempo se modela con un reloj que es común a todas las comunidades. Es importante notar que una persona solo podrá pertenecer a una unica comunidad.

Diagramas:

Use

Diagrama de clase

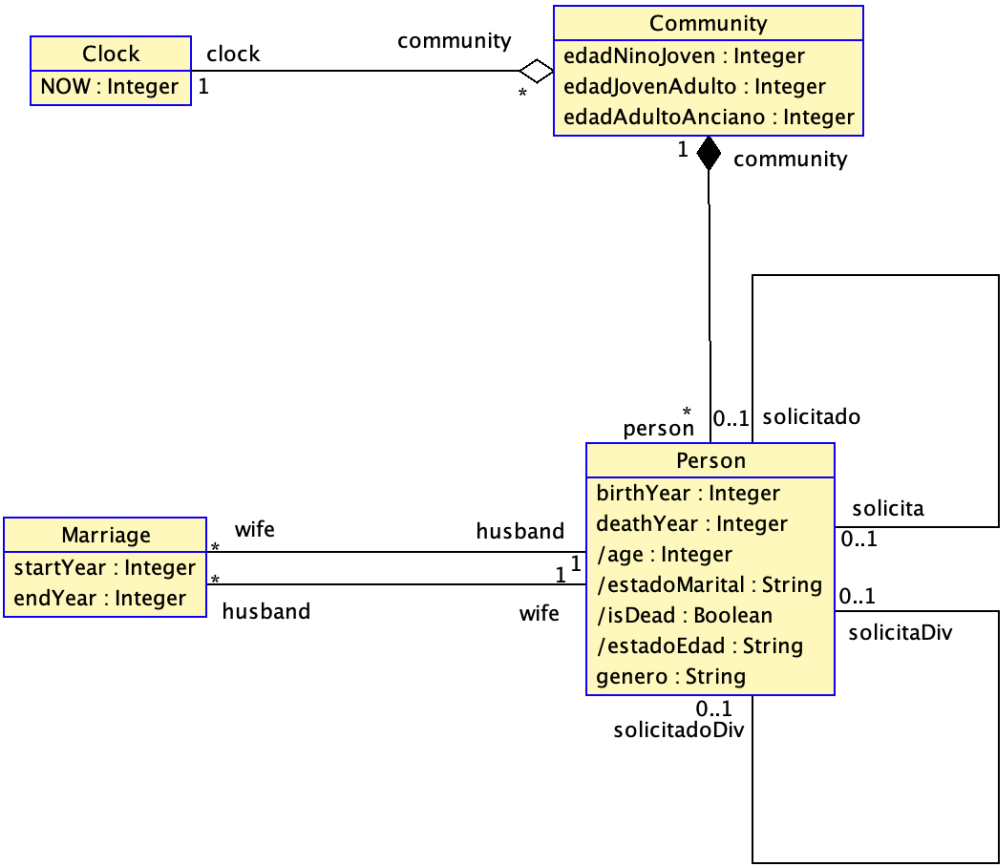
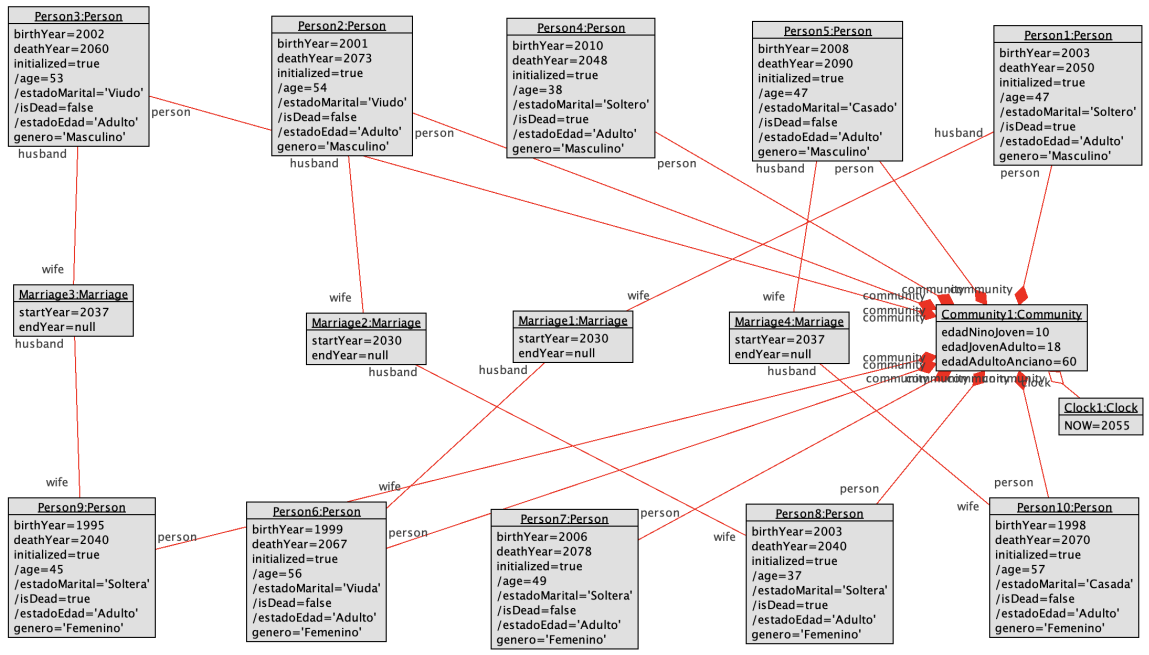


Diagrama de objetos



Papyrus

Diagrama de clase

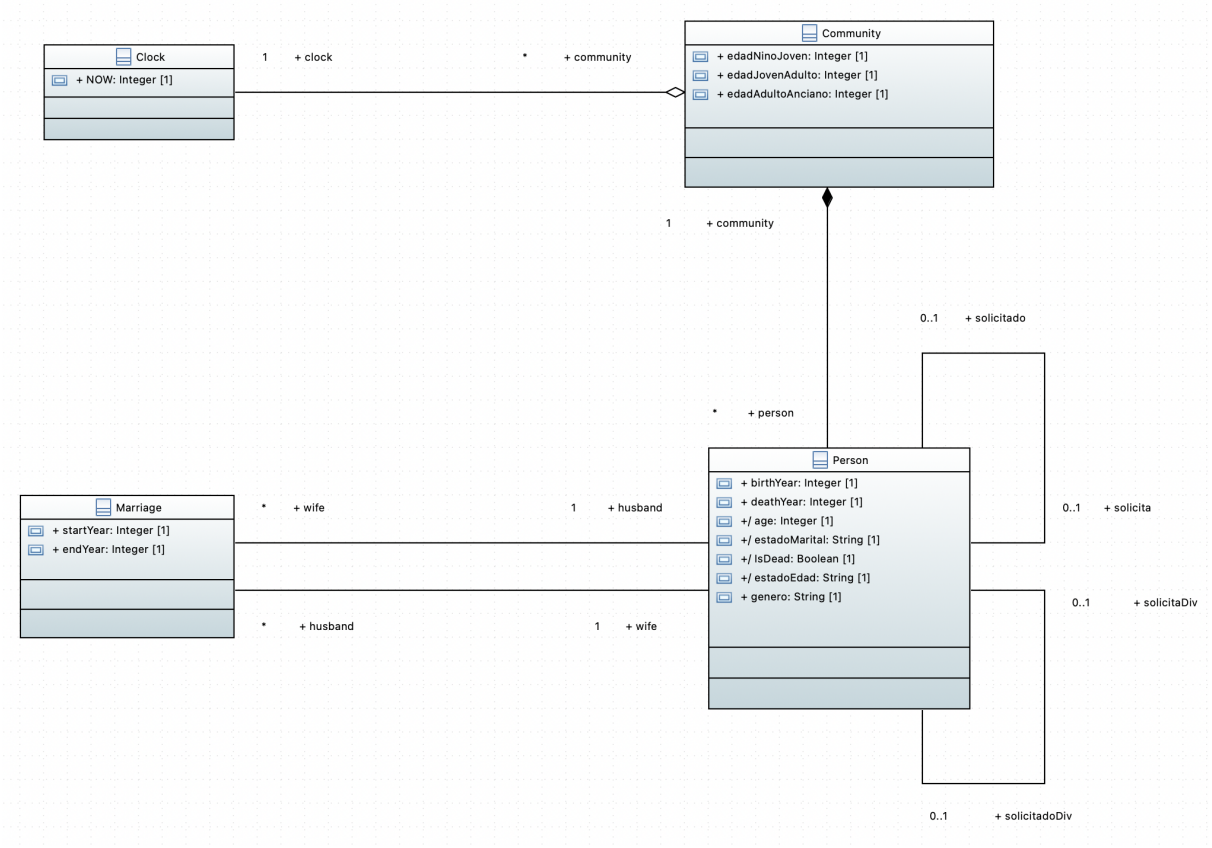
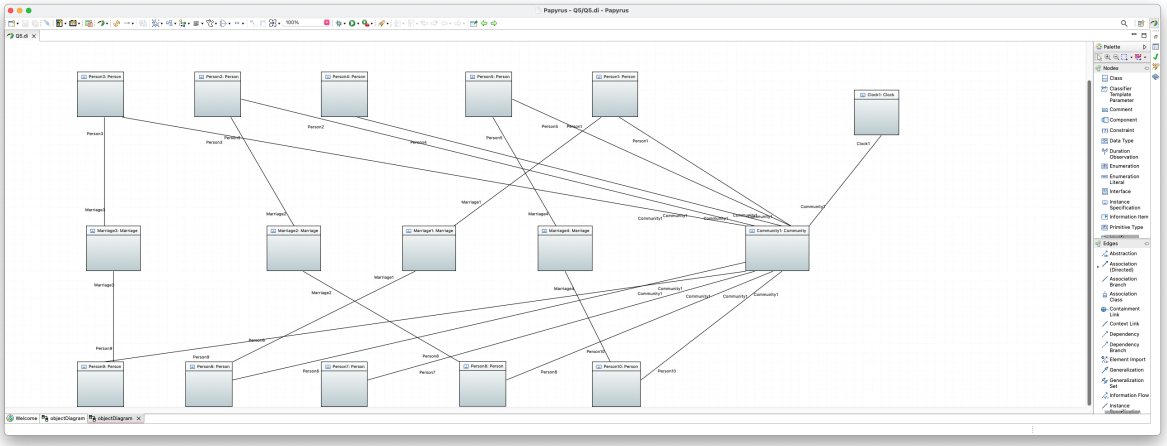


Diagrama de objetos



Visual Paradigm

Diagrama de clase

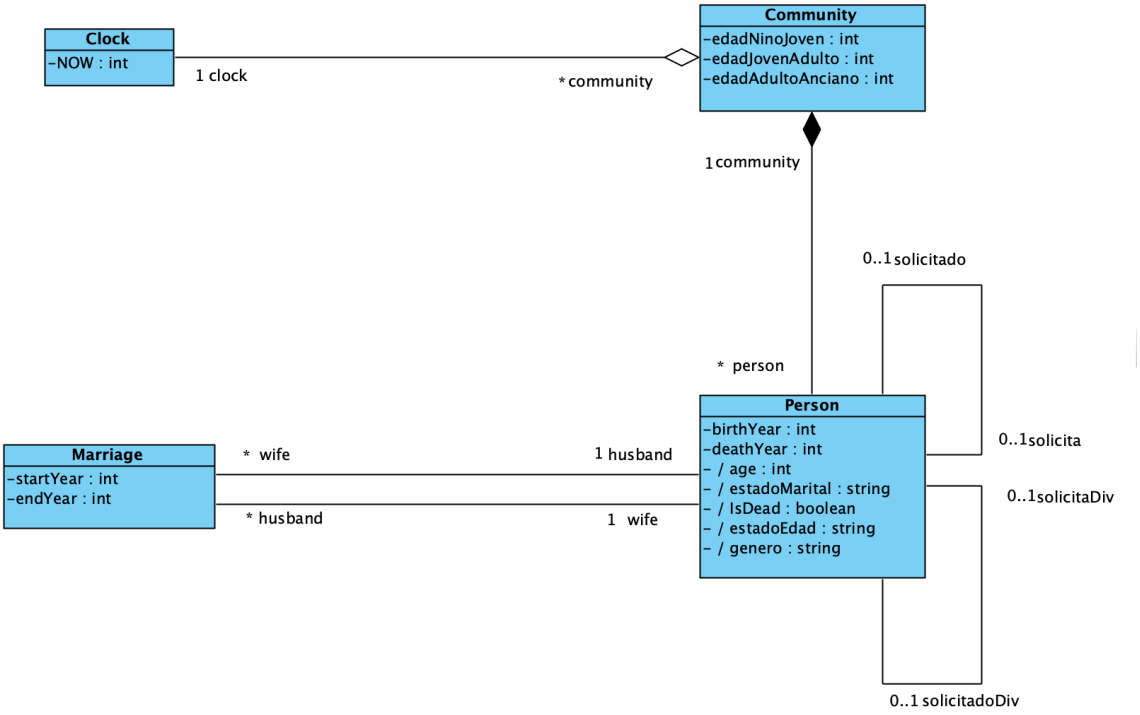
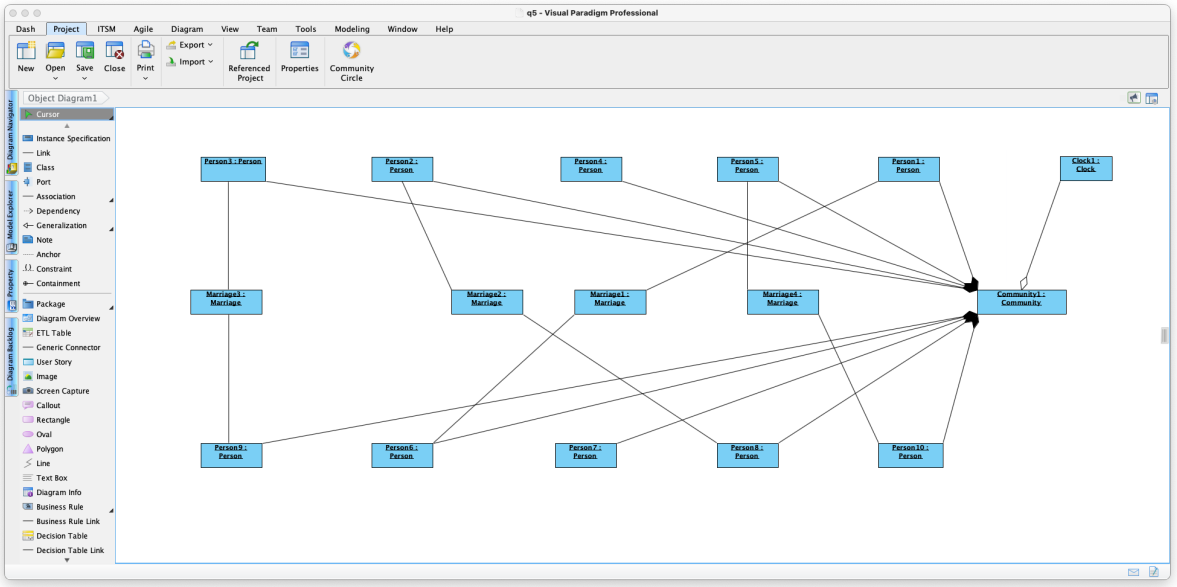


Diagrama de objetos



Explicación:

Clases:

Tenemos las siguientes clases en el modelo:

Clock:

La clase clock representa, como bien dice el nombre, un reloj. Esta clase modela el paso del tiempo en las comunidades. Servirá para determinar eventos importantes como el nacimiento y fallecimiento de una persona, además de su edad, posibles matrimonios, etc.

Tiene un atributo:

NOW: representa el momento actual

Community:

La clase community representa una comunidad, compuesta por personas.

Marriage:

La clase marriage representa un matrimonio que, en nuestro caso, será entre una mujer y un hombre.

Tiene dos atributos:

startYear: año de comienzo del matrimonio. Este deberá ser menor al del año de finalización.

endYear: año de finalización del matrimonio. La finalización de un matrimonio podrá deberse a varios factores, como el fallecimiento de una de las dos partes involucradas o un divorcio.

Person

La clase person representa una persona que pertenece a una comunidad.

Tiene cuatro atributos:

birthYear: año de nacimiento.

deathYear: año de fallecimiento.

age: edad de la persona.

isDead: booleano que indica true si la persona ha fallecido y false en caso contrario.

Relaciones

A continuación, vamos a explicar las relaciones implementadas entre las clases. Para ello usaremos las restricciones tal y como las vemos en el archivo .use.

association marryWifeHusband:

Es una relación entre la clase Person y Marriage. Indica un matrimonio ente una mujer y un hombre, en la que la mujer jugará el rol de persona.

La multiplicidad de esta relación es de 1 a *, 1 ya que un matrimonio no puede estar compuesto por más de una mujer (esto es así para asegurar la monogamia en la relación) y * porque una mujer puede haber tenido entre 0 y varios matrimonios.

association marryHusbandWife:

Es una relación entre la clase Person y Marriage. Indica un matrimonio ente una mujer y un hombre, en la que el hombre jugará el rol de persona.

La multiplicidad funcionará igual que en el caso de la mujer, siendo este de 1 a *.

association casa:

Esta relación es de Person a Person. Representa la solicitud de una persona a otra para casarse con ella.

La multiplicidad es 0..1 a 0..1, ya que una persona puede o no pedirle matrimonio a la otra.

association divorcio:

Esta relación, al igual que casa, es de Person a Person y representa la solicitud de divorcio de una persona a otra.

La multiplicidad es 0..1 a 0..1 por la misma razón, una persona puede o no pedirle el divorcio a otra.

composition perteneceAComunidad:

Esta relación de composición entre Person y Comunidad representa la pertenencia de una persona a una comunidad. Es una composición en vez de una agregación ya que, como hemos comentado en el contexto de la memoria, una persona solo puede pertenecer a una única comunidad.

La multiplicidad es 1 a *, 1 porque una persona debe pertenecer a mínimo y máximo una comunidad y * porque una comunidad está compuesta por 0, 1 o varias personas.

aggregation tiempo:

La relación de agregación tiempo entre Community y Clock representa el paso del tiempo. Es de agregación porque el reloj se comparte entre comunidades.

La multiplicidad es 1 a * porque una comunidad tendrá un único reloj y el reloj pertenece a 0,1 o varias comunidades.

Restricciones aléticas

Una persona no puede fallecer antes de nacer.

```
context Person inv personaNoPuedeMorirAntesDeNacer:
    self.deathYear <> null and self.birthYear <> null implies
        self.deathYear >= self.birthYear
```

Un matrimonio no puede acabar antes de empezar.

```
context Marriage inv matrimonioNoPuedeAcabarAntesDeEmpezar:
    self.endYear<>null implies
        self.endYear >= self.startYear
```

Los muertos no pueden estar casados.

```
context Person inv personaMuertaNoPuedeEstarCasada:
    self.isDead = true implies self.estadoMarital = 'Soltere'
```

No se puede nacer en el futuro: Clock.NOW tiene que ser mayor o igual que cualquier fecha de nacimiento.

```
context Community inv noSePuedeNacerEnElFuturo:
    self.clock.NOW <> null implies
        self.person -> forAll(p | p.birthYear <> null implies
        p.birthYear <= self.clock.NOW)
```

No puede haber matrimonios futuros (“acordados”): Clock.NOW tiene que ser siempre mayor o igual que cualquier fecha de comienzo de un matrimonio.

```
context Community inv noHayMatrimoniosAcordados:
    self.clock <> null implies
        (self.person.husband<> null implies
        self.person.husband -> forAll(h | h.startYear <= self.clock.NOW)) and
        (self.person.wife <> null implies
        self.person.wife -> forAll(w | w.startYear <= self.clock.NOW))
```

No se puede divorciar una persona de alguien con quien no esté actualmente casado.

```
context Person inv noPuedesDivorciarteSiNoEstasCasado:
    self.solicitaDiv <> null implies
    (self.wife.wife -> includes(self) or
    self.husband.husband -> includes(self))
```

Restricciones deónticas

No puede haber más que un reloj en toda la aplicación (y por tanto compartido por todas las comunidades)

```
context Clock inv NumReloj:
Clock.allInstances() -> size = 1
```

Una persona no puede estar casada consigo misma.

```
context Person inv autoMatrimonio:
not self.wife -> isEmpty() implies
self.wife -> select(m | m.wife = self) -> isEmpty()
```

Monogamia: Una persona no puede tener más de un matrimonio activo en un momento dado.

```
context Person inv monogamia:
  self.wife -> union(self.husband) -> forAll(m1, m2 |
    m1 <> m2 implies (
     (m1.endYear < m2.startYear and m1.startYear < m2.startYear) or
        (m1.endYear > m2.startYear and m1.startYear > m2.startYear)
    )
  )
```

Hacemos un set de todos los matrimonios, y cogemos todas las combinaciones con el forAll. Si no son el mismo matrimonio tiene que cumplir esas condiciones.

Los niños no pueden estar casados.

```
context Person inv ninoSoltero:
self.age < self.community.edadNinoJoven implies
self.wife -> isEmpty() and self.husband -> isEmpty()
```

No se permite ni la eutanasia ni el suicidio.

```
context Person inv muerteNatural:
self.age < self.community.edadJovenAdulto implies
isDead = false
```

La edad en las que las personas cambian de niño a joven, de joven a adulto o de adulto a anciano puede depender de la comunidad a la que pertenezca la persona.

(Especificado en el diagrama de clases)

Restricciones adicionales:

Una persona solo puede solicitar matrimonio a otra si no está casada:

```
context Person inv solicitaMatrimonioSiNoEstaCasada:
self.solicita->notEmpty() implies self.solicita.isMarried() = false
```

El año de muerte de una persona no puede ser menor que el año de su nacimiento

```
context Person inv fechaMuerteNoPuedeSerMayorAlAnoDeNacimiento:
self.deathYear<> null implies self.deathYear > self.birthYear
```

La edad no puede ser null

```
context Person inv edadNoPuedeSerNull:
self.age <> null
```

El genero no puede ser null

```
context Person inv generoNoPuedeSerNull:
self.genero <> null
```

El genero tiene que ser 'Masculino' o 'Femenino'

```
context Person inv generoTieneQueSerUnoDeLosDos:
self.genero <> null implies (self.genero='Masculino' or self.genero='Femenino')
```

Queries

isMarried(): Boolean que devuelve si una persona está actualmente casada o no

```
isMarried():
    Boolean =
    self.husband -> exists(m | m.startYear <> null and m.husband -> size() >= 1) or
    self.wife -> exists(m | m.startYear <> null and m.wife -> size() >= 1) --creo que esta ultima linea se podria quitar
```

marriages(): Integer que devuelve el número de matrimonios que ha tenido una persona a lo largo de su vida (incluido el actual, si es que estás casado).

```
marriages():
    Integer = self.husband -> size() + self.wife -> size()
```

Operaciones

Para modelar el comportamiento de las personas en una comunidad a lo largo de su vida implementaremos las siguientes operaciones.

De la clase Person:

cumpleAnos(): Suma 1 a la edad de una persona. La precondición es que la persona no esté muerta.

```
cumpleAnos()
    begin
        self.age := self.age + 1;
    end
    pre: self.isDead = false
```

Solo se suman años a los vivos

casarseConMarido(marido : Person): Inserta en la relación marryWifeHusband y marryHusbandWife self y la persona marido. Elimina la relación casa, que se refiere a la solicitud de matrimonio. La precondición es que al menos uno de las dos personas en la relación debe haber solicitado el matrimonio y que ninguno de los dos puede estar muerto. La post condición es que la solicitud ya no existirá, sustituyendo por matrimonio.

```
casarseConMarido(marido : Person)
    begin
    declare m : Marriage;
    m := new Marriage;
    insert(self, m) into marryWifeHusband;
    insert(marido, m) into marryHusbandWife;
    marido.estadoMarital := 'Casado';
```

```
    self.estadoMarital := 'Casado';
    m.startYear := self.community.clock.NOW;
    m.endYear := null;

    if (not self.solicitado -> isEmpty()) then
            delete(self,marido) from casa;
    end;

        if (not self.solicita -> isEmpty()) then
            delete(marido,self) from casa;
    end;

    end
    pre: (self.solicita <> null and marido.solicitado <> null)
    or (self.solicitado <> null and marido.solicita <> null)
    and self.isDead=false and marido.isDead=false
    post: not (self.solicita <> null and marido.solicitado <> null)
    or not (self.solicitado <> null and marido.solicita <> null)
```

casarseConMujer(mujer : Person)

```
casarseConMujer(mujer : Person)
    begin
    declare m : Marriage;
    m := new Marriage;
    insert(self, m) into marryHusbandWife;
    insert(mujer, m) into marryWifeHusband;
    mujer.estadoMarital := 'Casado';
    self.estadoMarital := 'Casado';
    m.startYear := self.community.clock.NOW;
    m.endYear := null;

    if (not self.solicitado -> isEmpty()) then
            delete(self,mujer) from casa;
    end;

        if (not self.solicita -> isEmpty()) then
            delete(mujer,self) from casa;
    end;

    end
    pre: (self.solicita <> null and mujer.solicitado <> null)
    or (self.solicitado <> null and mujer.solicita <> null)
    and self.isDead=false and mujer.isDead=false
    post: not (self.solicita <> null and mujer.solicitado <> null)
    or not (self.solicitado <> null and mujer.solicita <> null)
```

fallecer(): Fallecer cambia el estado marital de cualquier persona a soltero/a, independientemente de su estado anterior. Si la persona estaba casada, se cambiará el estado de su pareja a viuda/o. También se borrarán las posibles solicitudes de matrimonio y divorcio que pueda tener la persona, ya que una vez fallecida no podrá casarse ni divorciarse.

```
fallecer()
    begin

    if (self.genero='Femenino') then
        self.estadoMarital := 'Soltera';
    else
        self.estadoMarital := 'Soltero';
    end;

    self.isDead := true;

    self.deathYear := self.community.clock.NOW;

    if (self.isMarried()) then      --Comprobamos, para finalizar el matrimonio
        if(not self.wife -> isEmpty()) then    --Si tiene mujer
            self.wife -> select(m | m.endYear = null)
            -> asSequence -> first().wife.estadoMarital := 'Viuda';
```

```
            self.wife -> select(m | m.endYear = null)
            -> asSequence -> first().endYear := self.community.clock.NOW;
        else                                    --Si tiene marido
            self.husband -> select(m | m.endYear = null)
            -> asSequence -> first().husband.estadoMarital := 'Viudo';
            self.husband -> select(m | m.endYear = null)
            -> asSequence -> first().endYear := self.community.clock.NOW;
        end;
    else                                    --Borramos las posibles relaciones
        if (not self.solicitado -> isEmpty()) then
                delete (self, self.solicitado) from casa;
        end;
        if (not self.solicita -> isEmpty()) then
                delete (self.solicita, self) from casa;
        end;
        if (not self.solicitadoDiv -> isEmpty()) then
            delete (self, self.solicitadoDiv) from divorcio;
        end;
        if (not self.solicitaDiv -> isEmpty()) then
            delete (self.solicitaDiv, self) from divorcio;
        end;
    end;

    end
    pre: self.isDead = false
    post: self.isDead = true
```

proponerDivorcio(persona : Person): Inserta en la relación divorcio a ambas personas en la relación. La precondición es que la pareja de la persona debe existir (se comprueba para ambos géneros) que la solicitud de divorcio es unidireccional, es decir, una persona solo puede pedir el divorcio a la otra si la otra no ha pedido el divorcio anteriormente y que ninguno de los dos puede haber fallecido.

```
proponerDivorcio(persona : Person)
    begin
        insert(self,persona) into divorcio
    end
    pre: (self.husband.exists(p|p.husband=persona)
        implies persona.wife.exists(p|p.wife=self)) and
        (self.wife.exists(p|p.wife  =persona)
        implies persona.husband.exists(p|p.husband=self)) and
        not self.solicitadoDiv->includes(persona) and
        not persona.solicitaDiv->includes(self) and
        self.isDead=false and persona.isDead=false
    post: self.solicitadoDiv->includes(persona) and
    persona.solicitaDiv->includes(self)
```

proponerMatrimonio(persona : Person)

```
proponerMatrimonio(persona : Person)
    begin
        insert(self,persona) into casa
    end

    pre: self.isDead=false and persona.isDead=false and
    self.solicitado <> persona and persona.solicitado <> self

    post: self.solicitado->includes(persona) and
    persona.solicita->includes(self)
```

casar(persona : Person): La precondición es que debe existir una solicitud de matrimonio de self a la otra persona, además de comprobar que ninguna de las dos personas está ya casado. La post condición es que la solicitud se elimina una vez aceptado el matrimonio.

```
casar(persona : Person)
    begin
```

```
        if(self.genero='Femenino' and persona.isDead=false) then
            self.casarseConMarido(persona);
        else
            if (self.genero='Masculino' and persona.isDead=false) then
            self.casarseConMujer(persona);
            end;
        end;
    end
    pre: self.solicita->includes(persona) and
    persona.solicitado->includes(self) and (self.genero <> persona.genero)
        and self.marriages() = 0 and persona.marriages() = 0
    post: not self.solicita->includes(persona) and
    not persona.solicitado->includes(self)
```

rechazarCasar(persona : Person): La precondición es que la solicitud de matrimonio debe existir y la post condición es que dejará de existir una vez rechazado.

```
rechazarCasar(persona : Person)
    begin
        delete(persona,self) from casa;
    end
    pre: self.solicita->includes(persona) and
    persona.solicitado->includes(self)
    post: not self.solicita->includes(persona) and
    not persona.solicitado->includes(self)
```

rechazarDivorciar(persona: Person): Funciona de la misma manera que rechazarCasar, solo que se refiere a un divorcio.

```
rechazarDivorciar(persona: Person)
    begin
        delete(persona,self) from divorcio
    end
    pre: self.solicitaDiv->includes(persona) and
    persona.solicitadoDiv->includes(self)
    post: not self.solicitaDiv->includes(persona) and
    not persona.solicitadoDiv->includes(self)
```

divorciar(persona : Person): Declaramos un matrimonio m en el que los participantes sean self y la pareja de self. Se elimina m de las relaciones marryHusbandWife y marryWifeHusband, se elimina la petición de divorcio y se destruye la variable m.

La precondición será que debe existir una solicitud de divorcio anteriormente y que dichas personas estan casadas el uno con el otro.

```
divorciar(persona : Person)
    begin
    declare m : Marriage;
        if (self.genero='Femenino') then
            m :=self.husband->select(p|p.husband=persona)->asSequence()->first();
            delete (persona,m) from marryHusbandWife;
            delete (self,m) from marryWifeHusband;
            delete(persona,self) from divorcio;
            destroy(m)
        else
            if (self.genero='Masculino') then
                m:=self.wife->select(p|p.wife=persona)->asSequence()->first();
                delete(persona,m) from marryWifeHusband;
                delete(self,m) from marryHusbandWife;
                delete(persona,self) from divorcio;
                destroy(m);
            end;
        end;
    end
    pre: ((self.solicitaDiv->includes(persona) and
```

```
          persona.solicitadoDiv->includes(self)) or
          (self.solicitaDiv->includes(self) and
          persona.solicitadoDiv->includes(persona))) and
          ((self.husband.exists(p|p.husband=persona) and
          persona.wife.exists(p|p.wife  =self)) or
          (self.wife.exists(p|p.wife=persona) and
          persona.husband.exists(p|p.husband=self)))
          and self.genero <> persona.genero
    post: not self.solicitaDiv->includes(persona) and
          not persona.solicitadoDiv->includes(self)
```

initialize()

```
initialize()
begin
    self.initialized := true
end
pre: not self.initialized
post: self.initialized
```

De la clase Clock:

anoNuevo()

```
anoNuevo()
    begin
        self.NOW := self.NOW + 1;
         for p in Person.allInstances() do
            if (p.isDead <> true) then
                p.cumpleAnos();
            end
        end
    end
end
```

Si no hacemos la comprobación if, USE parará la ejecución de la operación si encuentra a alguien muerto porque no se cumple la precondición.

StateMachine

```
statemachines

    psm CicloDeLaVida
        states
            s: initial
            Init
            Nino [self.age < self.community.edadNinoJoven and
            self.initialized and not self.isDead]
            Joven [self.age >= self.community.edadNinoJoven and
            self.age < self.community.edadJovenAdulto
            and not self.isDead]
            Adulto [self.age >= self.community.edadJovenAdulto and
            self.age < self.community.edadAdultoAnciano
            and self.initialized and not self.isDead]
            Anciano [self.age >= self.community.edadAdultoAnciano and
            not self.isDead and self.initialized]
            Muerte : final [self.isDead]
        transitions
            s->Init
            Init->Nino {[self.age < self.community.edadNinoJoven and
            not self.initialized] initialize()}
            Init->Joven {[self.age >= self.community.edadNinoJoven and
            self.age < self.community.edadJovenAdulto
            and not self.initialized] initialize()}
            Init->Adulto {[self.age >= self.community.edadJovenAdulto and
            self.age < self.community.edadAdultoAnciano
            and not self.initialized] initialize()}
```

```
            Init->Anciano {[self.age >= self.community.edadAdultoAnciano and
            not self.initialized] initialize()}

            Nino->Nino {[self.age < self.community.edadNinoJoven]
            cumpleAnos()}
            Nino->Joven {[self.age + 1 >= self.community.edadNinoJoven]
            cumpleAnos()}
            Joven->Joven {[self.age < self.community.edadJovenAdulto]
            cumpleAnos()}
            Joven->Adulto {[self.age + 1 >= self.community.edadJovenAdulto]
            cumpleAnos()}
            Adulto->Adulto {[self.age < self.community.edadAdultoAnciano]
            cumpleAnos()}
            Adulto->Anciano {[self.age + 1 >= self.community.edadAdultoAnciano]
            cumpleAnos()}
            Anciano->Anciano {[not self.isDead and
            (self.deathYear - self.community.clock.NOW) > 0] cumpleAnos()}

            Nino->Muerte {[not self.isDead] fallecer()}
            Joven->Muerte {[not self.isDead] fallecer()}
            Adulto->Muerte {[not self.isDead] fallecer()}
            Anciano->Muerte {[not self.isDead] fallecer()}
    end
```

En la maquina de estados tendremos los siguientes componentes:

Estados:

Declaramos los estados niño, joven, adulto y anciano, asegurandonos de que su edad queda dentro de los rangos establecidos por la comunidad.

Transiciones:

Las transiciones de un grupo de edad al siguiente se harán cuando la persona cumple años. Cuando ejecutamos la operación cumpleAnos(), comprobaremos que la edad a la que entra la persona sigue estando en el grupo de edad al que pertenece actualmente, en caso contrario, transicionará al siguiente grupo de edad.

Si ejecutamos la operación fallecer(), la persona pasará de cualquier grupo de edad en el que se encuentra al estado “Muerte”.