2. Práctica: Integración de reglas y objetos

3.1. Objetivo de la práctica

El objetivo de esta práctica es utilizar distintos esquemas de representación (reglas y objetos) e integrarlos.

3.2. Representación de relaciones familiares y herencias

Escribir un progrrama **CLIPS** con las partes propuestas para representar la siguiente información referente a una persona: `nombre`, `nombreyapellido`, `conyuge`, `padres`, `hijos`, y si está `vivo` o no. En caso de que no este vivo el atributo herederos contiene la lista de `herederos` de la persona fallecida.

NOTA: Supondremos que el slot `nombre` contiene un símbolo que identifica de manera unívoca a la instancia. Nombreyapellido es una cadena con el nombre y primer apellido. Los slots que hacen referencia a personas contienen los simbolos que las identifican. Algunas funciones CLIPS que pueden ser útiles para el ejercicio están documentadas al final de la práctica.

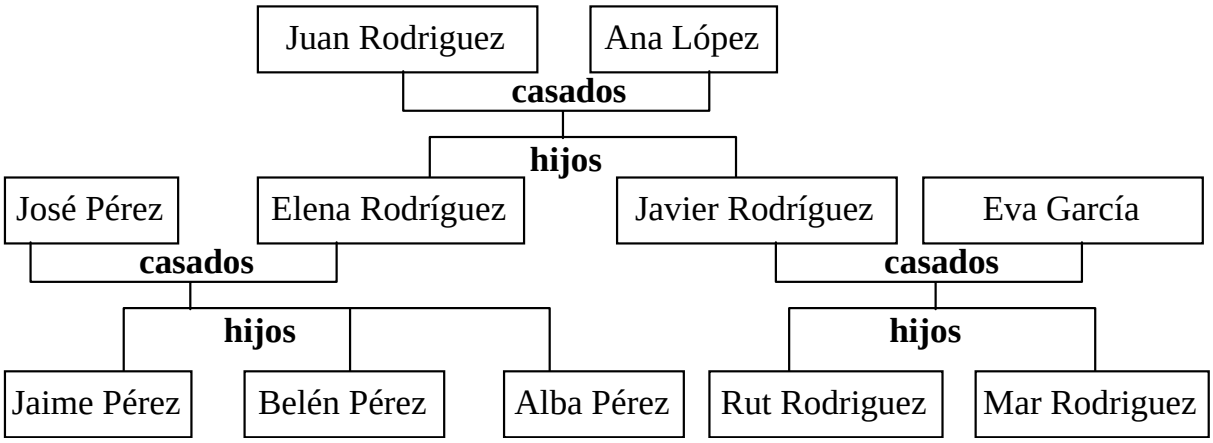
Dadas las siguientes declaraciones se pide

```
(defclass persona (is-a USER)
  (role concrete)
 (pattern-match reactive)
 (slot nombre
      (type SYMBOL)
      (create-accessor read-write))
 (slot nombreyapellido
      (type STRING)
      (create-accessor read-write))
 (slot conyuge
      (type SYMBOL)
      (create-accessor read-write))
 (multislot padres
      (create-accessor read-write))
 (multislot hijos
      (create-accessor read))
 (slot vivo
      (allowed-symbols si no)
      (create-accessor read-write))
 (multislot herederos
      (create-accessor read)))
```

```
(definstances Inicializa
  (Juan of persona (nombre Juan) (nombreyapellido "Juan Rodriguez"))
```

```
(Ana of persona (nombre Ana) (nombreyapellido "Ana Lopez"))
(Jose of persona (nombre Jose) (nombreyapellido "Jose Perez"))
(Elena of persona (nombre Elena) (nombreyapellido "Elena Rodriguez"))
(Javier of persona (nombre Javier) (nombreyapellido "Javier Rodriguez"))
(Eva of persona (nombre Eva) (nombreyapellido "Eva Garcia"))
(Jaime of persona (nombre Jaime) (nombreyapellido "Jaime Perez"))
(Belen of persona (nombre Belen) (nombreyapellido "Belen Perez"))
(Alba of persona (nombre Alba) (nombreyapellido "Alba Perez"))
(Rut of persona (nombre Rut) (nombreyapellido "Rut Rodriguez"))
(Mar of persona (nombre Mar) (nombreyapellido "Mar Rodriguez")))
```

Primera Parte: Se deben definir las relaciones iniciales entre los miembros de la familia que son las siguientes:



Para establecer las relaciones iniciales utilizaremos la siguiente función:

```
(deffunction inicializa-relaciones ()
     (+ [Juan] [Ana])
     (+ [Jose] [Elena])
     (+ [Javier] [Eva])
     (send [Juan] agnade-hijos (create$ Javier Elena))
     (send [Jose] agnade-hijos (create$ Jaime Belen Alba))
     (send [Javier] agnade-hijos (create$ Rut Mar)))
```

Definir la función genérica `+` que recibe dos personas y las casa.

Definir el método `agnade-hijos` así como el demonio que incluya los hijos en la lista de hijos del conyuge correspondiente y actualize la lista de padres en los hijos.

Segunda Parte:

1. Define el método de `persona agnade-heredero` que recibe un heredero y lo incluye en el atributo herederos. En caso de que el argumento enviado esté ya en la lista o sea el propio fallecido no se incluirá.

2. Dada la siguiente función genérica `fallecimiento`

```
(defmethod fallecimiento ((?muerto persona))
 (send ?muerto put-vivo no)
 (assert (herederos-de (send ?muerto get-nombre)))
 (run))
```

Escribir las reglas que calculan el valor del atributo `herederos` de la persona fallecida de acuerdo a las siguientes circunstancias:

1. Si una persona tiene cónyuge que está vivo, entre sus herederos estará el cónyuge.
2. Si una persona tiene hijos que están vivos, éstos estarán entre sus herederos.
3. Si una persona tiene algún hijo no vivo, los hijos vivos de ese hijo fallecido estarán entre sus herederos.
4. Si una persona no tiene hijos, ni cónyuge ni padres vivos, sus herederos serán los herederos de sus padres.

3. Despues de ejecutar la siguiente función `fallecimientos`:

```
(deffunction fallecimientos ()
(reset)
(inicializa-relaciones)
(fallecimiento [Elena])
(fallecimiento [Juan])
(fallecimiento [Jose])
(fallecimiento [Ana])
(fallecimiento [Eva])
(fallecimiento [Rut])
(fallecimiento [Mar])
(fallecimiento [Javier]))
```

Comprueba lo que devuelve `(send [Javier] get-herederos)` y si es correcto de acuerdo a las reglas y a las relaciones representadas.

Anexo: Funciones CLIPS que te pueden ser de utilidad para la práctica

`* (`**`instance-name-to-symbol`** `<instance-name-expression>)`
Devuelve un símbolo que es igual al nombre de la instancia que recibe como argumento.
Ejemplo:

```
 CLIPS> (instance-name-to-symbol [Juan])
Juan
```

`* (`**`symbol-to-instance-name`** `<instance-name-expression>)`

Devuelve el nombre de la instancia equivalente al símbolo que recibe como argumento:
Ejemplo:

```
CLIPS> (symbol-to-instance-name Juan)
[Juan]
```

* (**create$** <expresion>*)
Devuelve un valor "multifield" (lista, o valor de un multislot)
Ejemplo:

```
CLIPS> (create$ hola adios buenas)
(hola adios buenas)
CLIPS> (create$ (+ 3 4) (* 2 3) (/ 8 4))
(7 6 2.0)
```

* (**member$** <sigled-field-expresion> <multifield-expresion>)
Dice si un elemento pertenece a una lista. Si está en la lista devuelve la posición que ocupa
Ejemplo:

```
CLIPS> (member$ azul (create$ rojo 3 "texto" 8.7 azul))
5
CLIPS> (member$ azul (create$ rojo 3 "texto" 8.7 ))
FALSE
```

* (**insert$** <multifield-expresion> <integer-expresion>
<sigle-or-multifield-expresion>+)
Inserta una serie de uno o varios valores en una localización especificada.
Ejemplo:

```
CLIPS> (insert$ (create$ a b c d) 1 x)
(x a b c d)
CLIPS> (insert$ (create$ a b c d) 4 y z)
(a b c y z d)
CLIPS> (insert$ (create$ a b c d) 5 (create$ q r))
(a b c d q r)
```