¿CÓMO OBTENER UNA LECCIÓN?

Una vez que tengamos LOOP instalado y configurado para poder empezar a trabajar debemos crear o importar una **Lección**.

La Lección es un ambiente que tiene objetivos preestablecidos

Entendemos por Ambiente al lugar donde viven e interactúan los objetos

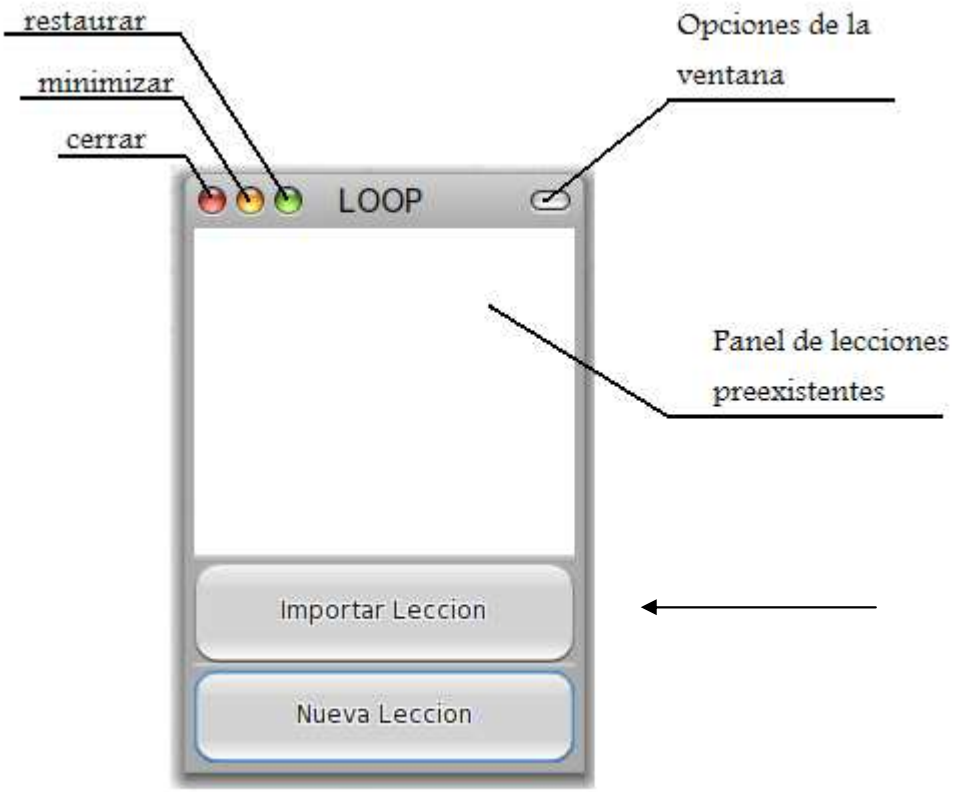
Como las lecciones pueden ser creadas o importadas se debe hacer una distintición.

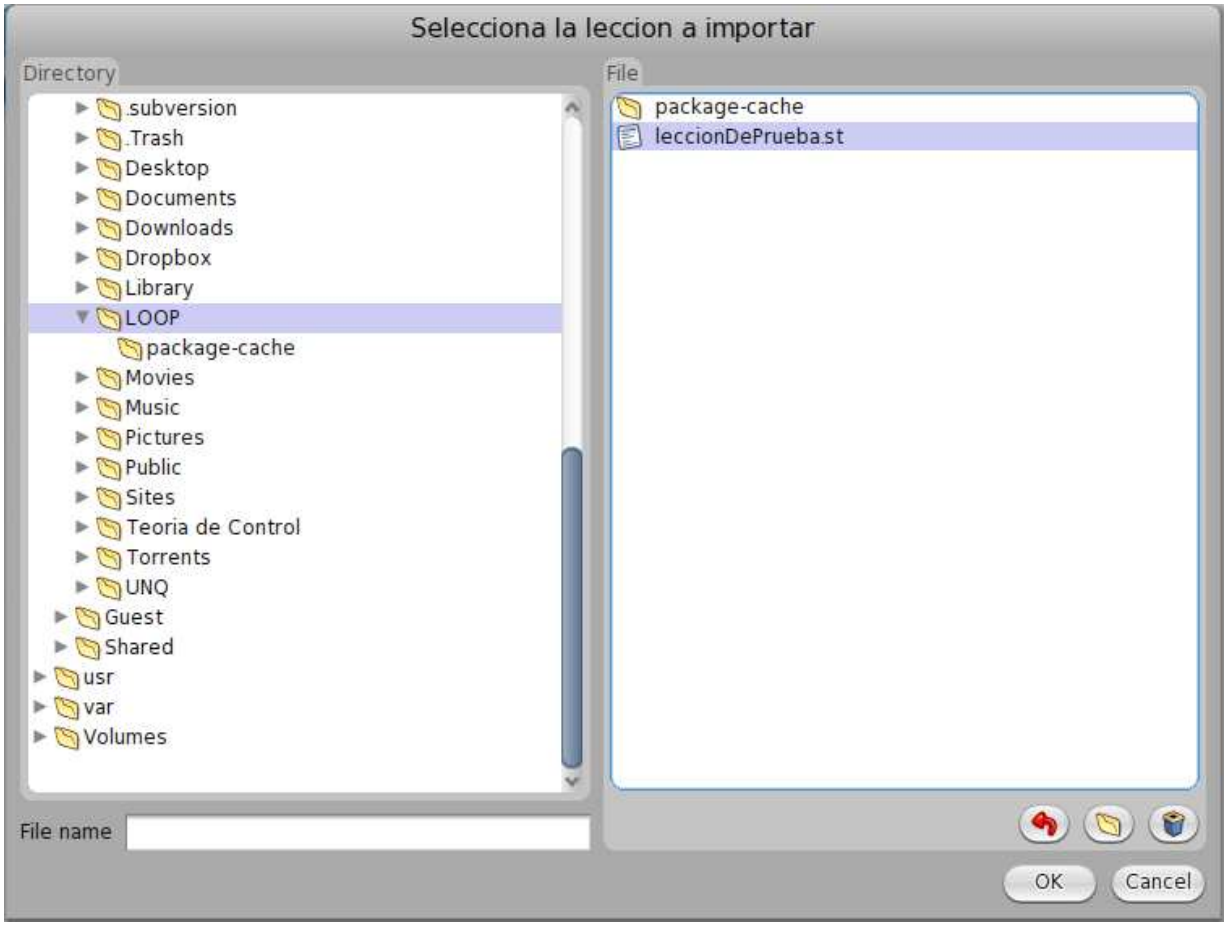
- En el caso de crear una lección su objetivo puede ser directamente realizar una práctica libre.
- En el caso de importar una lección pre-armada se puede interactuar con los objetos definidos en ella y de tener otras indicaciones se puede seguir la lección en modo paso a paso.

Para realizar estas acciones debemos utilizar el **Lesson Browser** que se accede haciendo click en el Loop Browser en WorldMenu.

IMPORTACIÓN

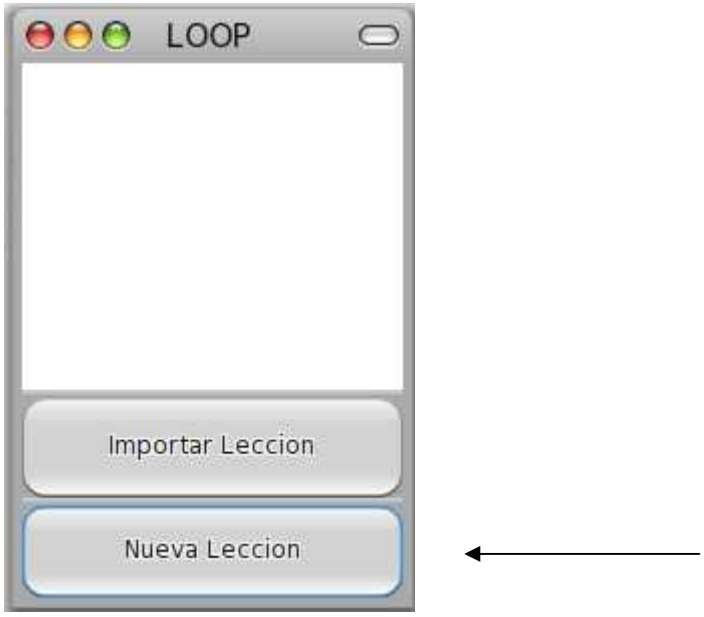
Para importar una lección debemos seleccionar la primera opción (**Importar Lección**) y se nos abrirá un explorador de directorios con el cual podemos buscar la lección a cargar



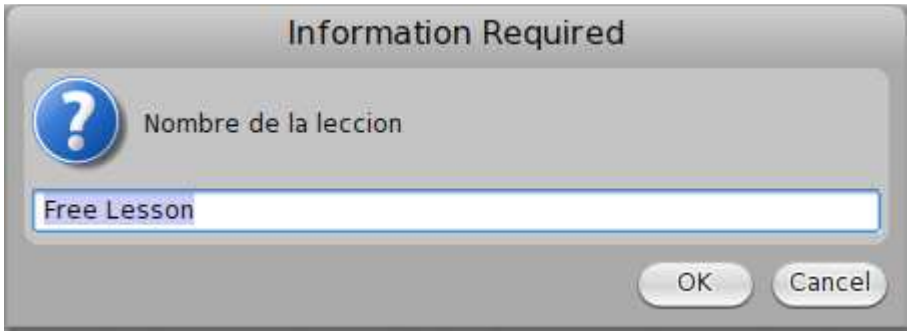


Una vez seleccionada debemos presionar **Ok**

CREACIÓN

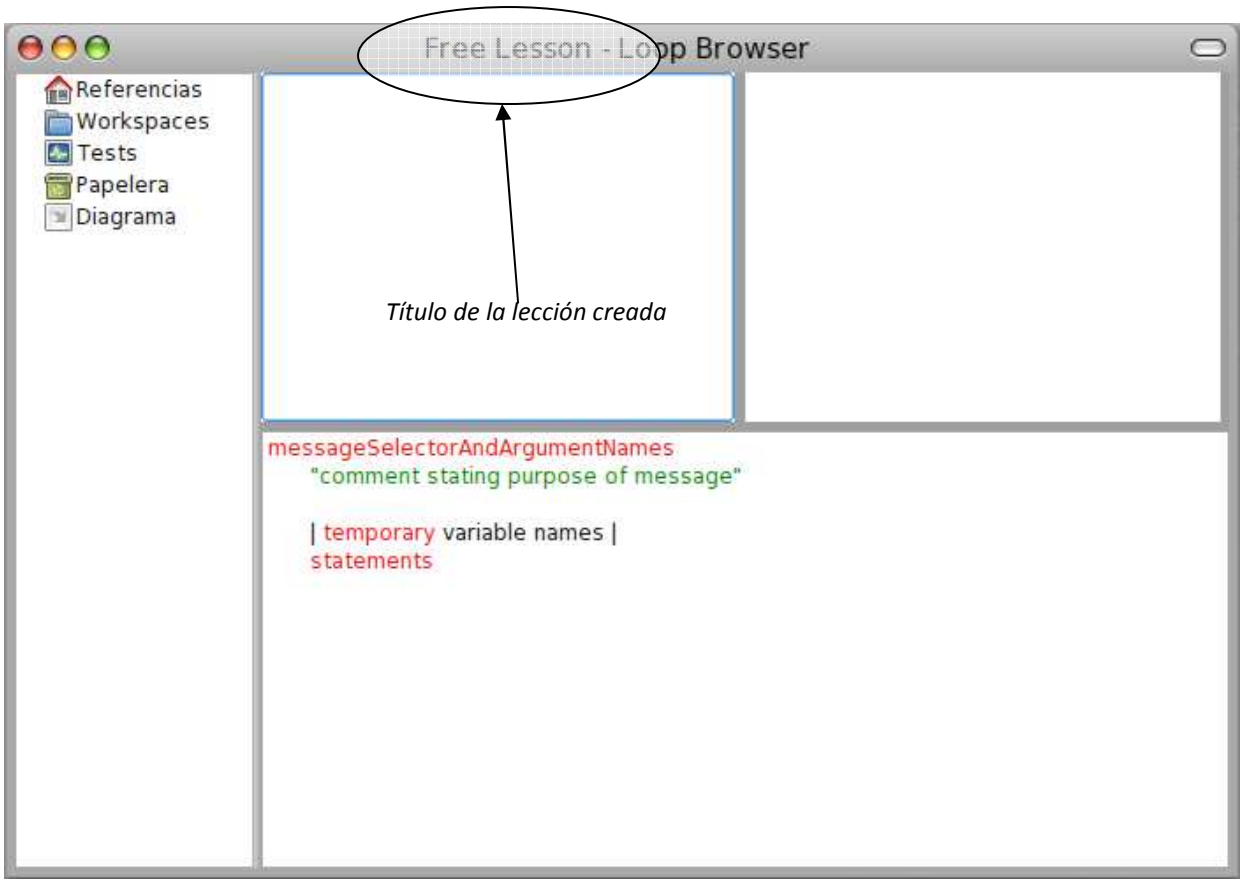


Para crear una nueva lección debemos seleccionar la segunda opción (**Nueva Lección**) donde nos aparecerá la siguiente pantalla



Se debe escribir un nombre para la lección y luego presionar **Ok**

Con esto tenemos una lección lista para empezar a trabajar.



EXPORTACIÓN

Una vez que tengamos creadas lecciones podemos exportarlas desde el **Lesson Browser**

Además de exportar la lección podemos:

- Abrirla
- Eliminarla
- Renombrarla

LOOP BROWSER

CONCEPTOS IMPORTANTES

> ***Objeto:*** *representación computacional de un ente que exhibe comportamiento, dicho comportamiento está representado en los métodos y se exhibe a través de los mensajes que entiende el objeto.*

Atributo: es una variable que le pertenece a un objeto

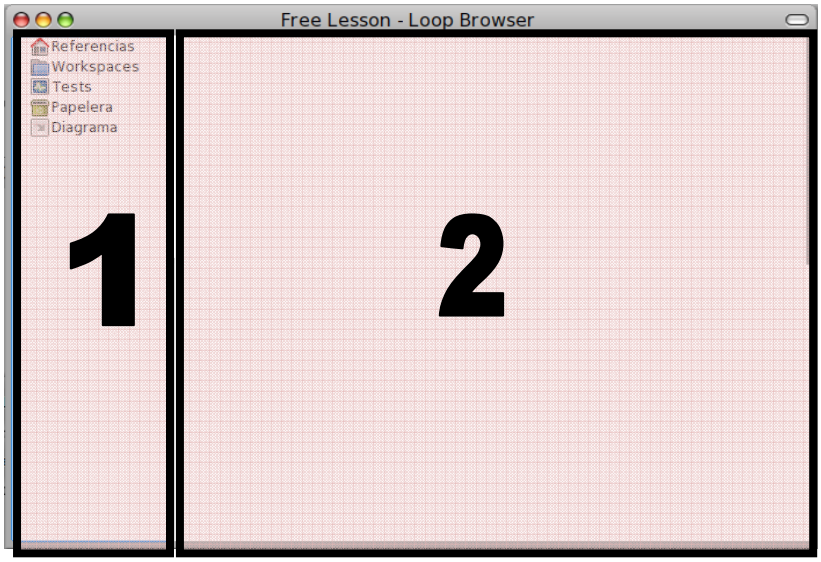
Variable: es una referencia a un objeto

Envío de Mensaje: es la única operación que podemos hacer para que un objeto realice una acción

Método (comportamiento): código que se ejecuta cuando se le envía un mensaje determinado a un objeto

VISIÓN GENERAL

Se pueden distinguir 2 grandes paneles en el **Loop Browser**



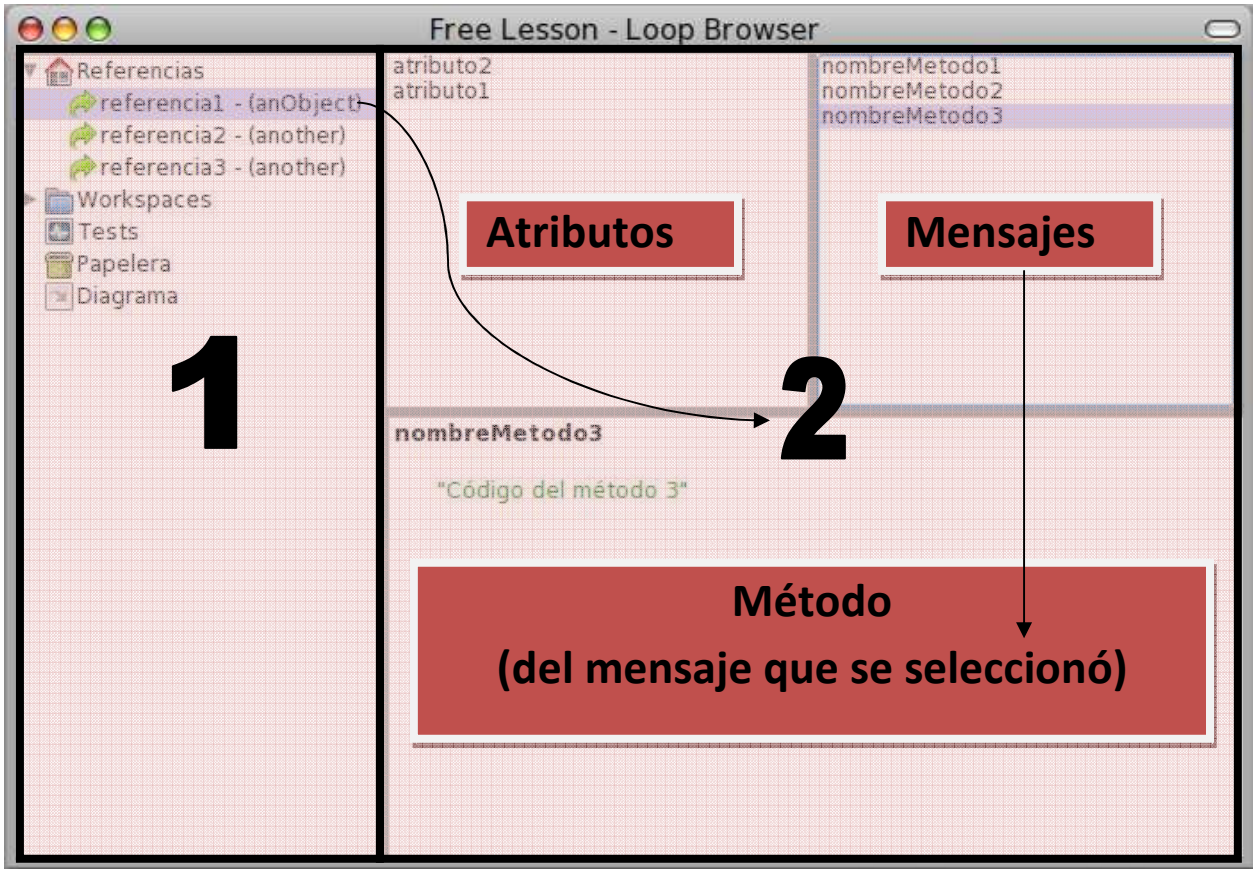
- Primer Panel:
  En esté panel aparecen en forma de árbol las actividades más comunes que se pueden realizar

  a) Referencias
  b) Workspace
  c) Tests
  d) Papelera
  e) Diagrama

- Segundo Panel:
  En este panel se representará información relacionada con el primer panel (por defecto este panel no se modificará).

LOOP BROWSER - REFERENCIAS

Cuando se selecciona una **referencia** aparecera en el segundo panel las características del objeto referenciado[1]
Todo objeto tiene:



- Estado interno: representado por los **atributos** que tiene el objeto

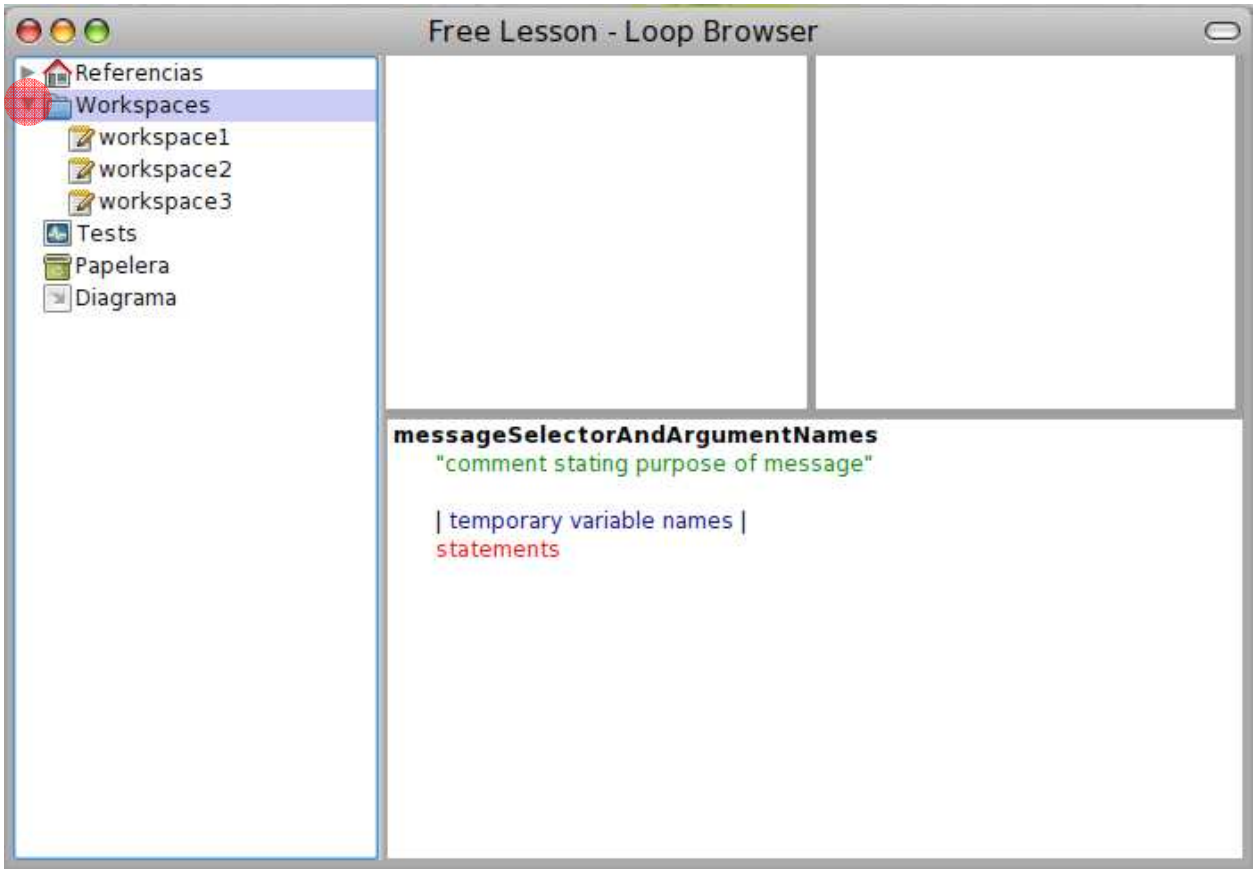
[1] En este ejemplo el objeto referenciado tiene como *displayString* anObject

- Comportamiento: representado por los **métodos** que se ejecutarán cuando se le envíe un determinado **mensaje** al objeto

Se explica en detalle el uso de referencias y objetos en Trabajando en un Ambiente de Objetos

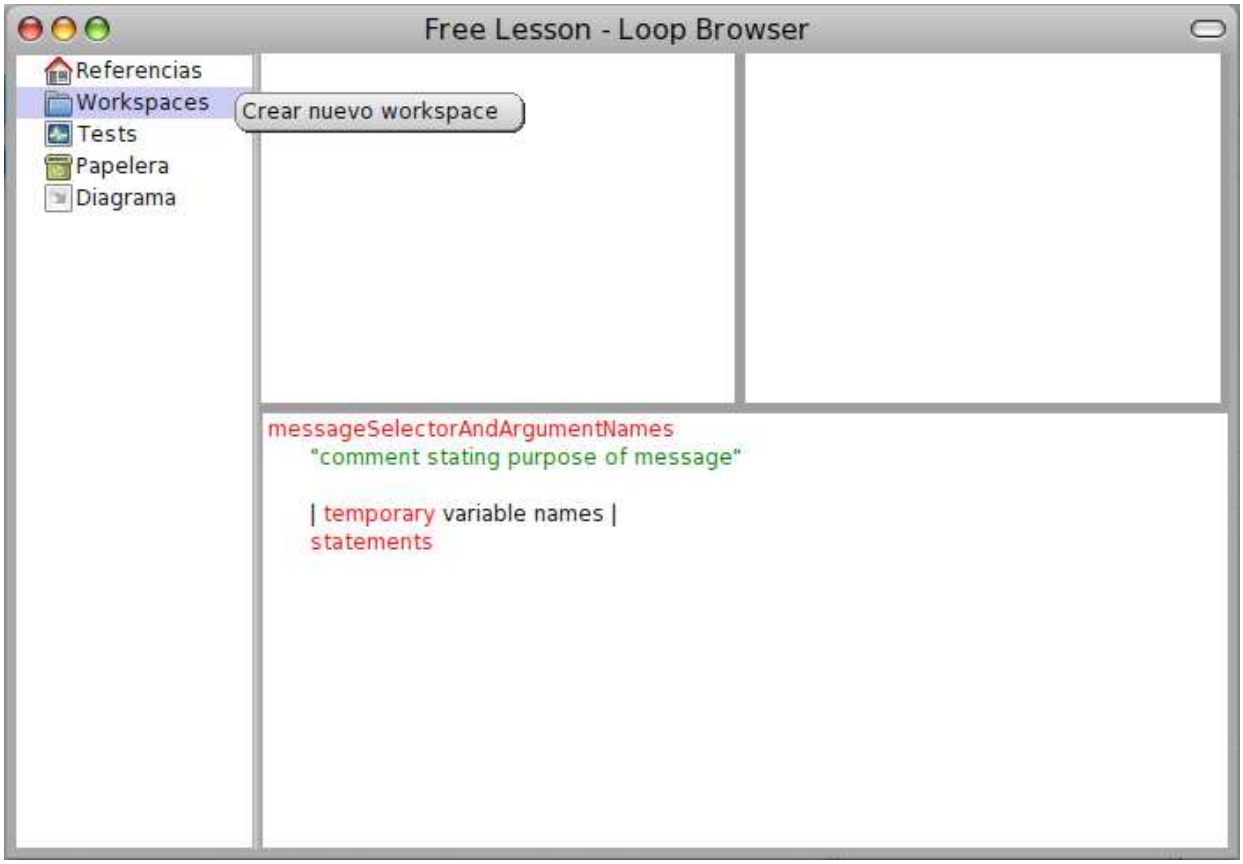
LOOP BROWSER - WORKSPACES

Debajo de la carpeta Workspaces van a aparecer todos los workspaces creados.

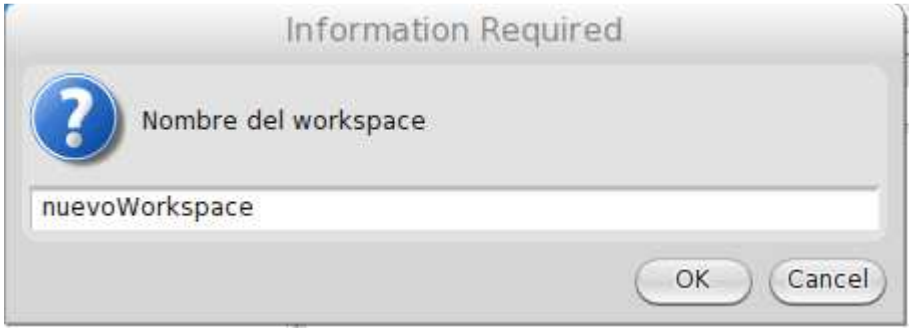


Se debe hacer click en el *triángulo* indicado en la imagen para abrir la lista de workspaces.

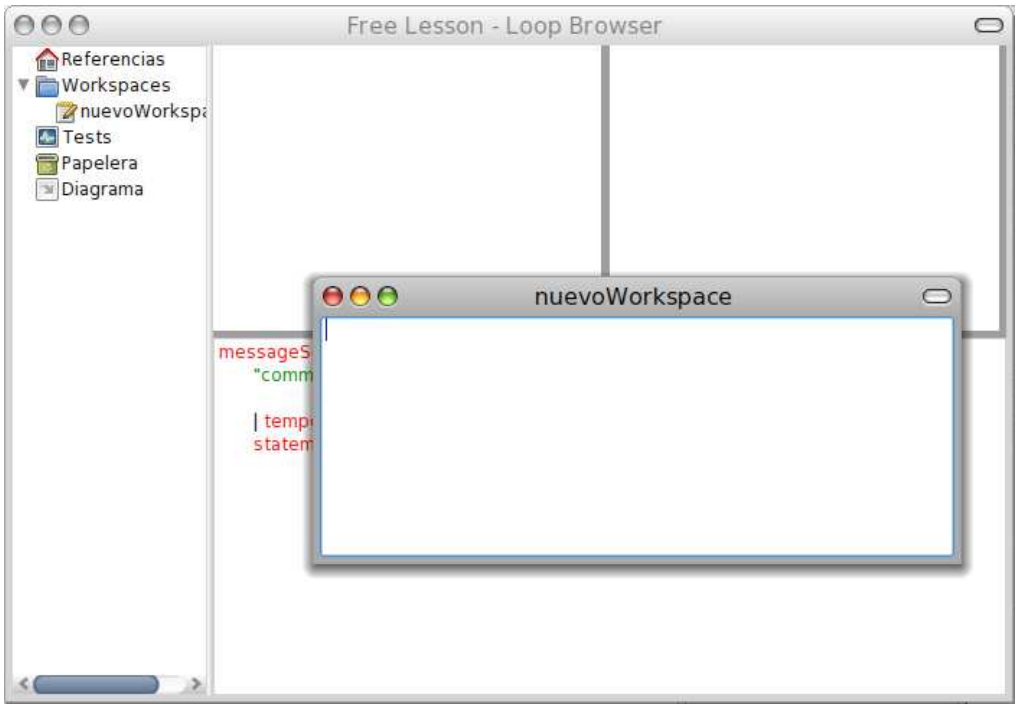
Una vez que hayamos creado objetos y previamente hayamos definido los nombres de las referencias hacia ellos, para que el usuario pueda interactuar con los objetos debe crear un Workspace.



Vamos al primer panel y haciendo botón derecho sobre Workspaces seleccionamos Crear nuevo workspace.

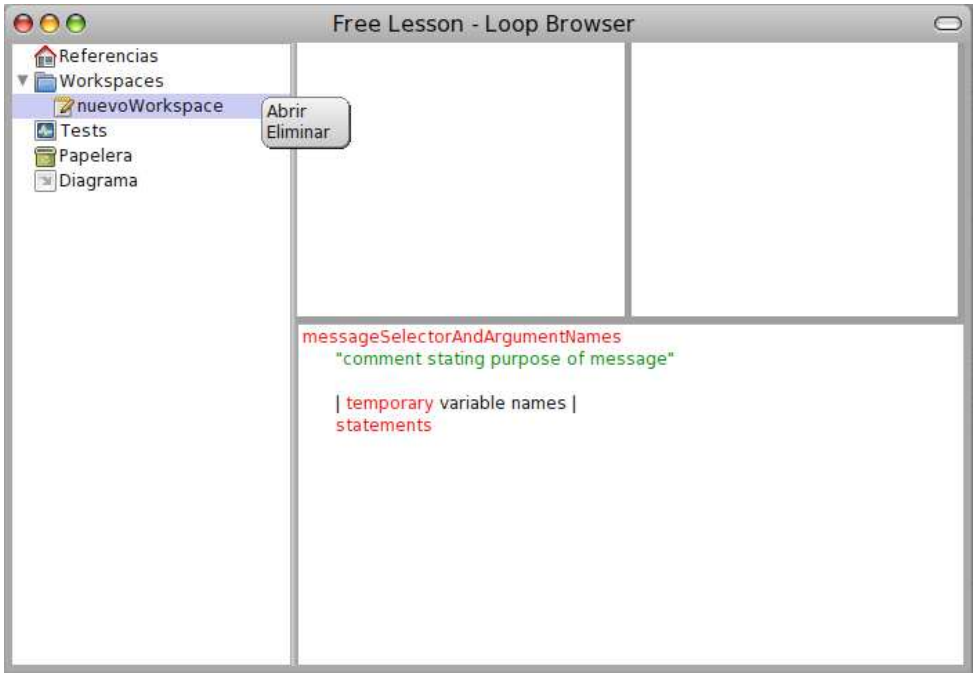


Después de elegir el nombre y darle Ok, vemos que aparece nuestro *nuevoWorkspace* en el primer panel, también se nos abre una ventana que es el nuevo workspace creado.

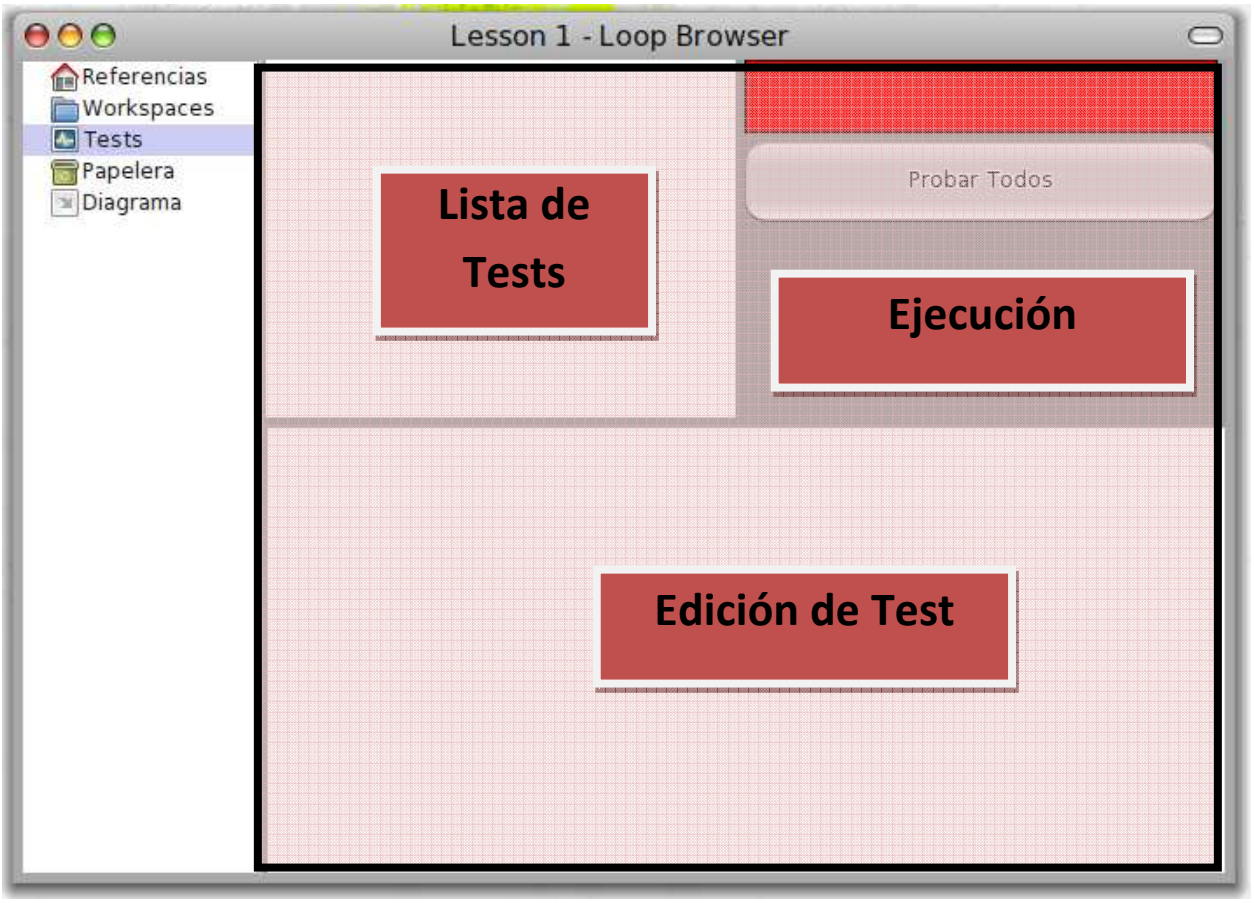


Después de cerrar un workspace podemos volver a abrirlo y también podemos eliminarlo.

Ambas opciones están disponibles haciendo botón derecho sobre el workspace deseado en el primer panel como se indica en la imagen.



LOOP BROWSER - TESTS



En el segundo panel podemos observar

1. **Lista de tests**:
   se verán todos los test creados hasta el momento con un icono que identifica su estado
   - Rojo
   - Amarillo
   - Verde

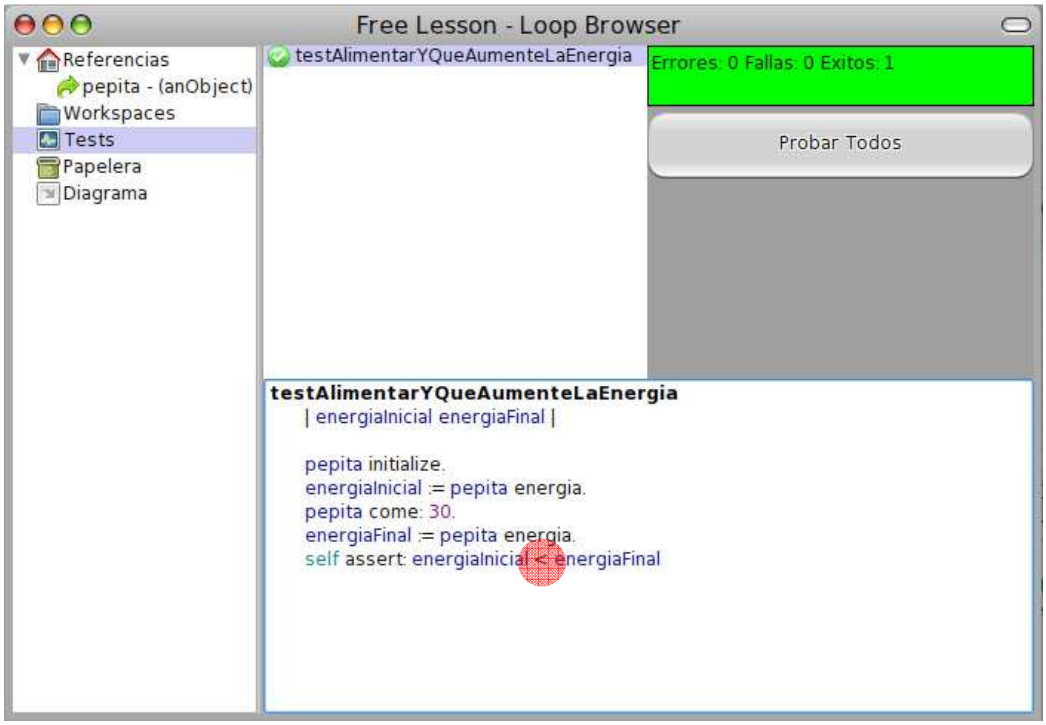
2. **Ejecución**:
   se verá un resumen de la ejecución de todos los test luego de hacer click en el botón "*Probar Todos*"

3. **Edición**:
   cuando se desea crear un nuevo test se hará en está sección.

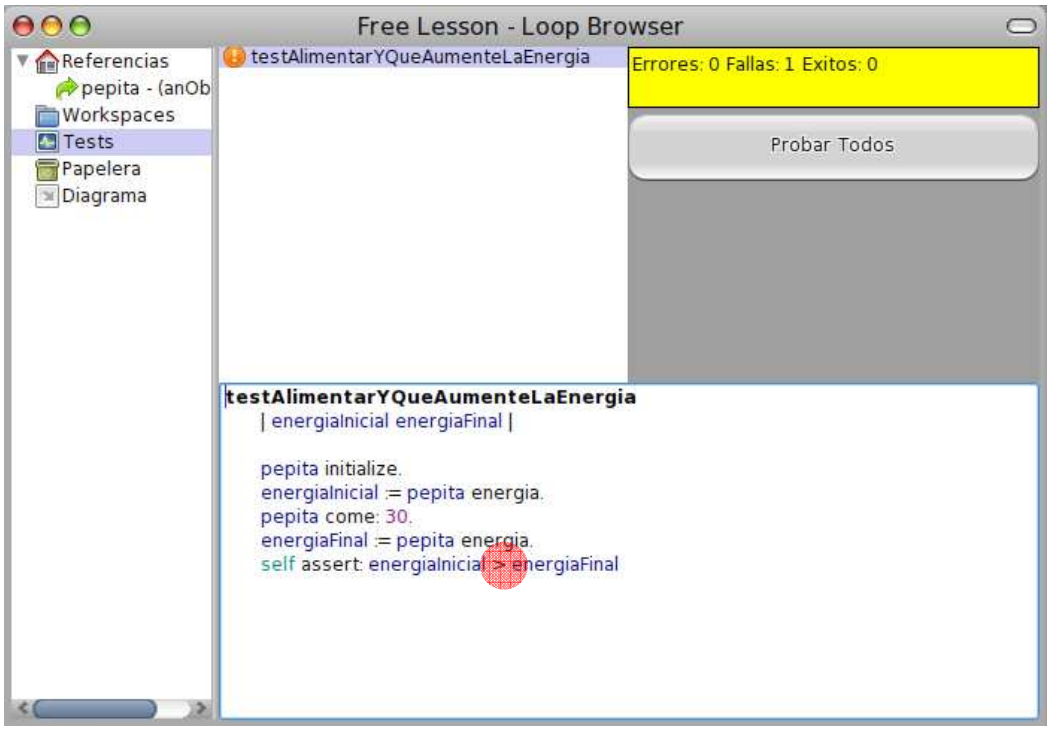
Un test constará de

- Nombre
- Código (donde se usan mensajes como assert:, deny:, should:, etc.)

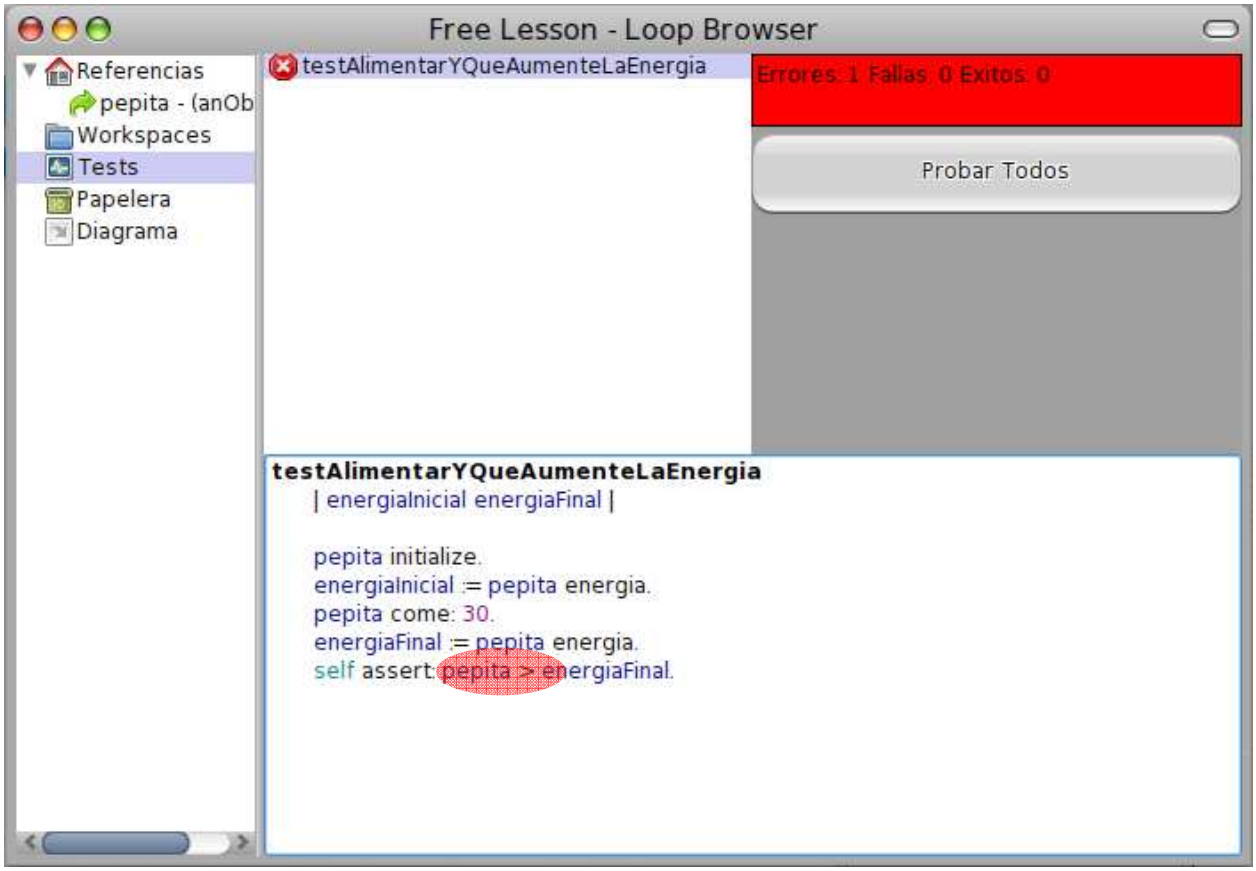
Ejemplo 1: Test que se ejecuta correctamente y da el resultado esperado



Ejemplo 2: Test que se ejecuta correctamente y no da el resultado esperado



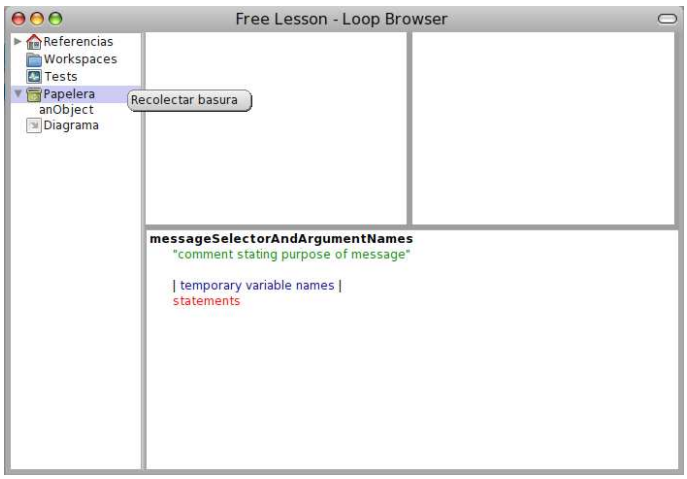
Ejemplo 3: Teste que no se ejecuta correctamente



Nota: *el objeto referenciado por la variable pepita no entiende el mensaje > (mayor)*

LOOP BROWSER - PAPELERA

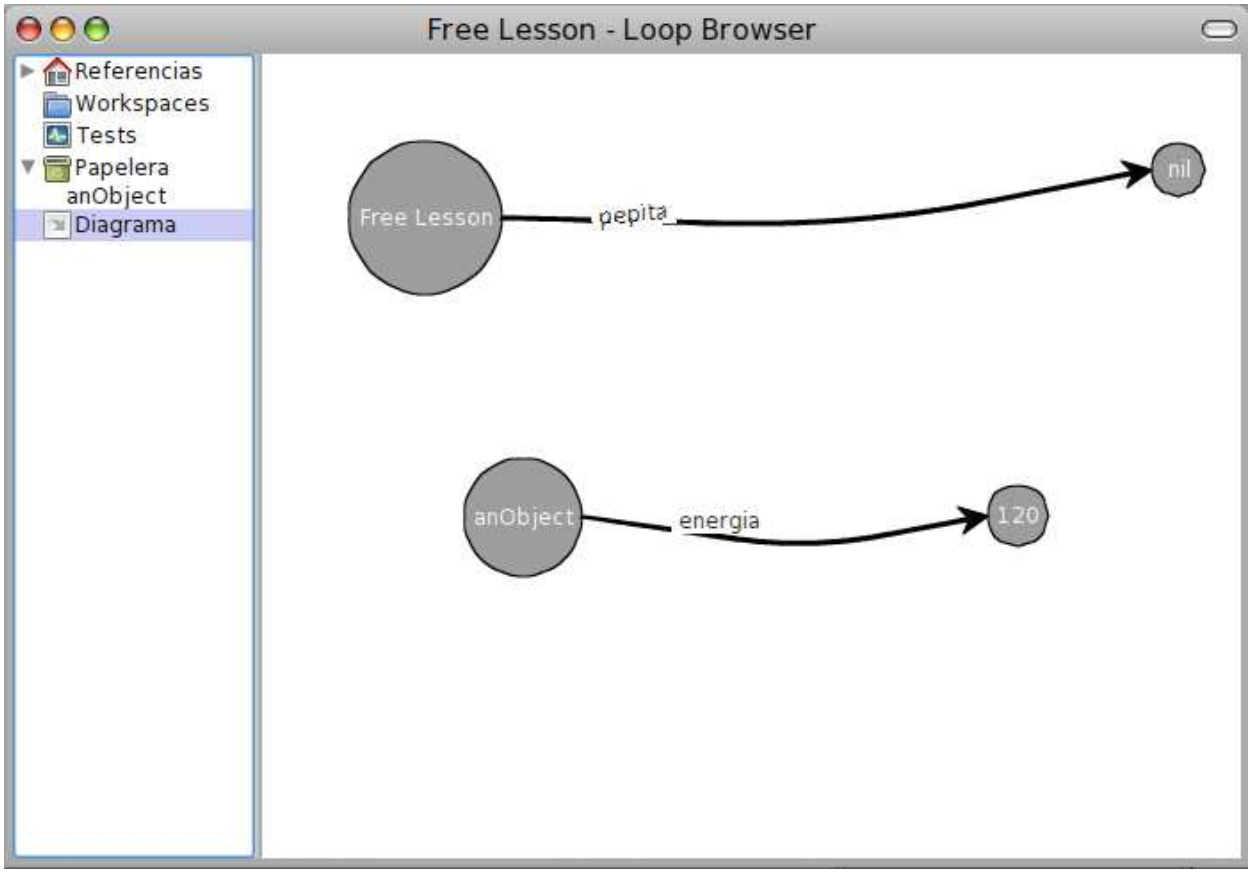
En la opción **Papelera** podemos eliminar los objetos que no son referenciados desde ningún otro objeto.



Para más información ver los ejemplos sobre [Garbage Collector](#).

LOOP BROWSER - DIAGRAMA

Por último, tenemos la opción **Diagrama**



En el segundo panel aparece un diagrama que representa las relaciones entre los objetos de la lección.

En el diagrama cada objeto está representado por un círculo y cada referencia (o variable) está representada por una flecha.

TRABAJANDO EN UN AMBIENTE DE OBJETOS

Una vez que estamos en el LOOP Browser necesitamos interactuar con distintos objetos.

Existen 2 tipos de objetos

- Los que ya vienen con LOOP (los números, los strings, los booleanos, etc.)
- Los que se crean en cada lección

Todo en LOOP es un objeto y para que un objeto le pueda hablar a otro necesita conocerlo de alguna manera.

¿De qué forma conoce la lección (e.g. Lección1) a un nuevo objeto? A través de una referencia.

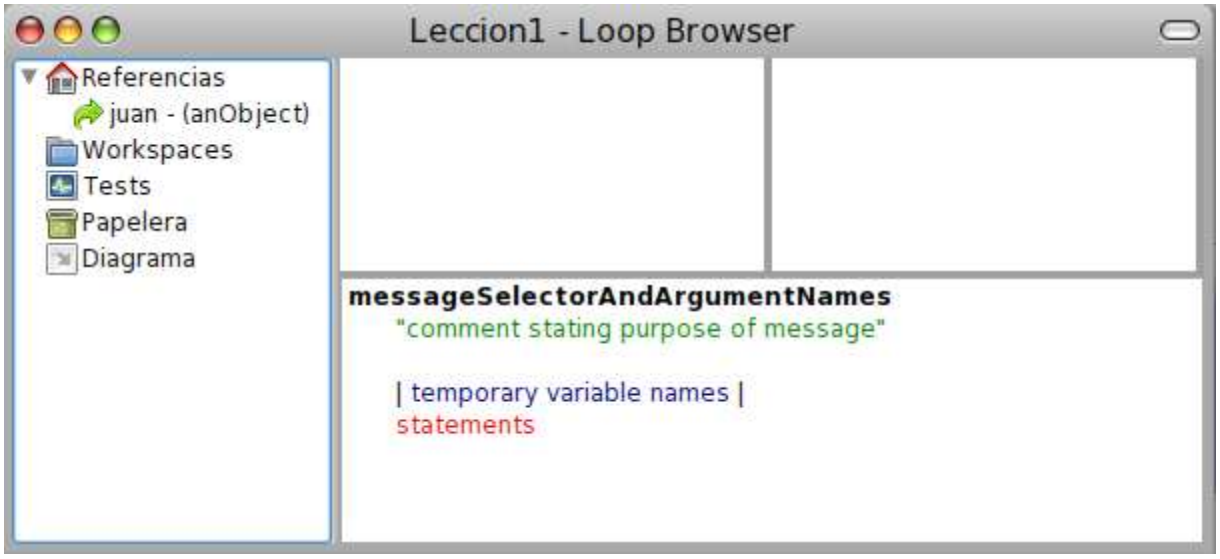
CREACIÓN DE OBJETOS

La creación de un objeto en un lección tiene 2 pasos

1. Crear el objeto
2. Crear una referencia hacia ese nuevo objeto

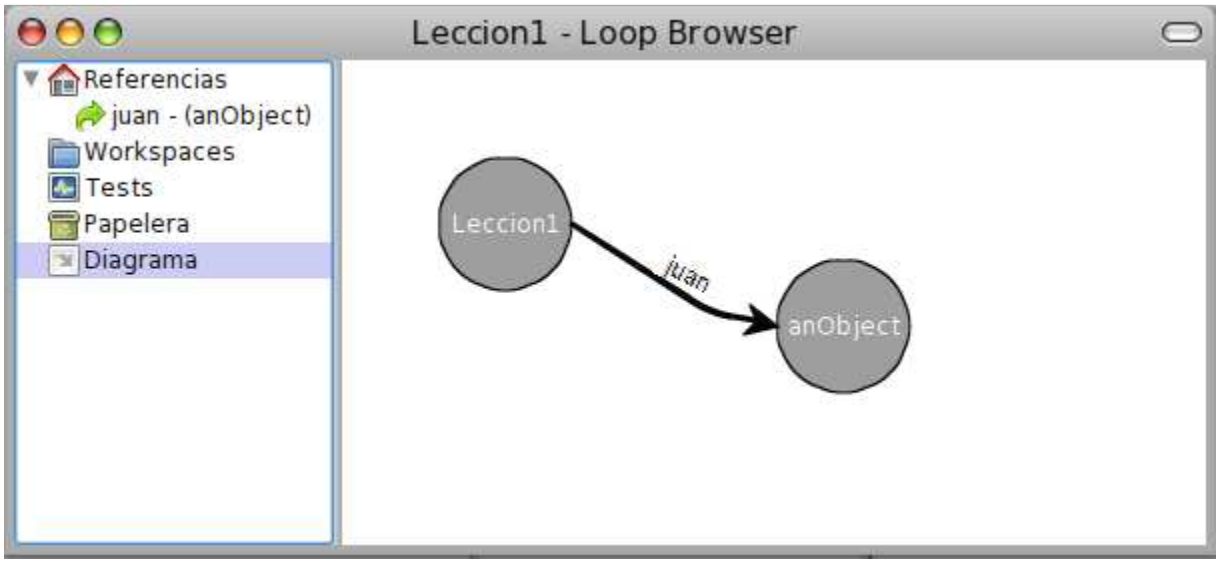
Estos dos pasos se hacen en simultáneo yendo al *Primer Panel*, haciendo botón derecho sobre **Referencias** y luego seleccionando **Crear nuevo objeto**

Importante: lo que escribimos no es el nombre del objeto, es el nombre de la referencia hacia ese nuevo objeto. Por eso si nos fijamos en el diagrama, adentro del círculo dice anObject y en la flecha que sale del título de la lección dice el nombre de la referencia.



*En nuestro ejemplo llamamos a la referencia **juan***

Para visualizar como está el ambiente podemos hacer click en **Diagrama**



Recordemos que en el diagrama cada objeto está representado por un círculo y cada referencia (o variable) está representada por una flecha.

Mensajes y Métodos

Si queremos interactuar con los objetos que creamos debemos abrir un Workspace (Ver LOOP Browser – Workspaces).

Para envíarle un mensaje a un objeto de la lección debemos escribir en el workspace

```
nombreDeLaReferencia<ESPACIO>mensaje<PUNTO>
```
[2]

Un mensaje está compuesto por 2 elementos

- Su nombre (o selector)
- Los parámetros que recibe (un mensaje puede no recibir parámetros – los parámetros son objetos)

> *Cuando se le envía un mensaje a un objeto se va a ejecutar un método con el mismo selector, a este mecanismo se lo conoce como Method-Lookup. De no encontrarse un método con el mismo nombre se producirá un error.*

Por ejemplos, en este momento el objeto referenciado por la variable **juan** no entiende el mensaje **#haceTuGracia** (ya que no escribimos ningún método llamado así).

Para probar esto vamos a un workspace (e.g. workspace1) y escribimos

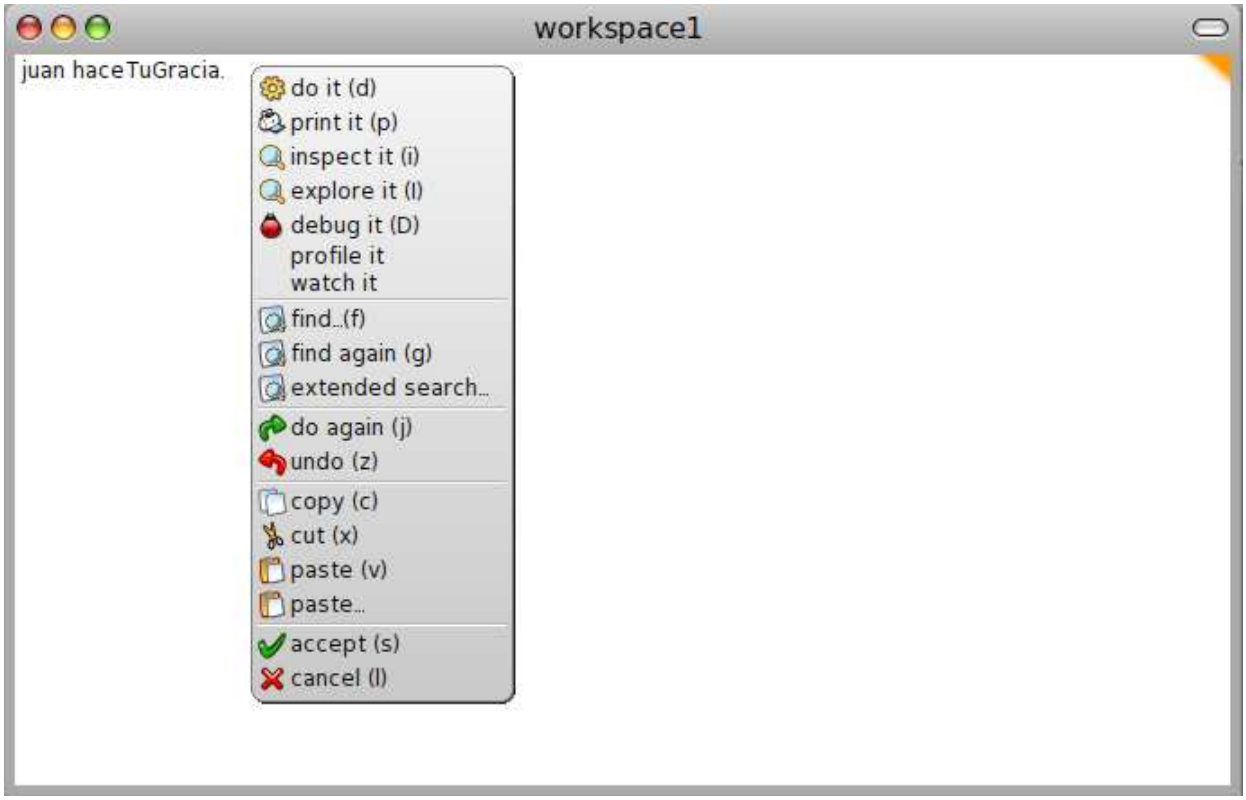
```
juan haceTuGracia.
```

Luego ejecutamos ese código haciendo botón derecho al final de la línea seleccionando *Do It* o sin utilizar el mouse presionando las teclas CMD[3]+d.

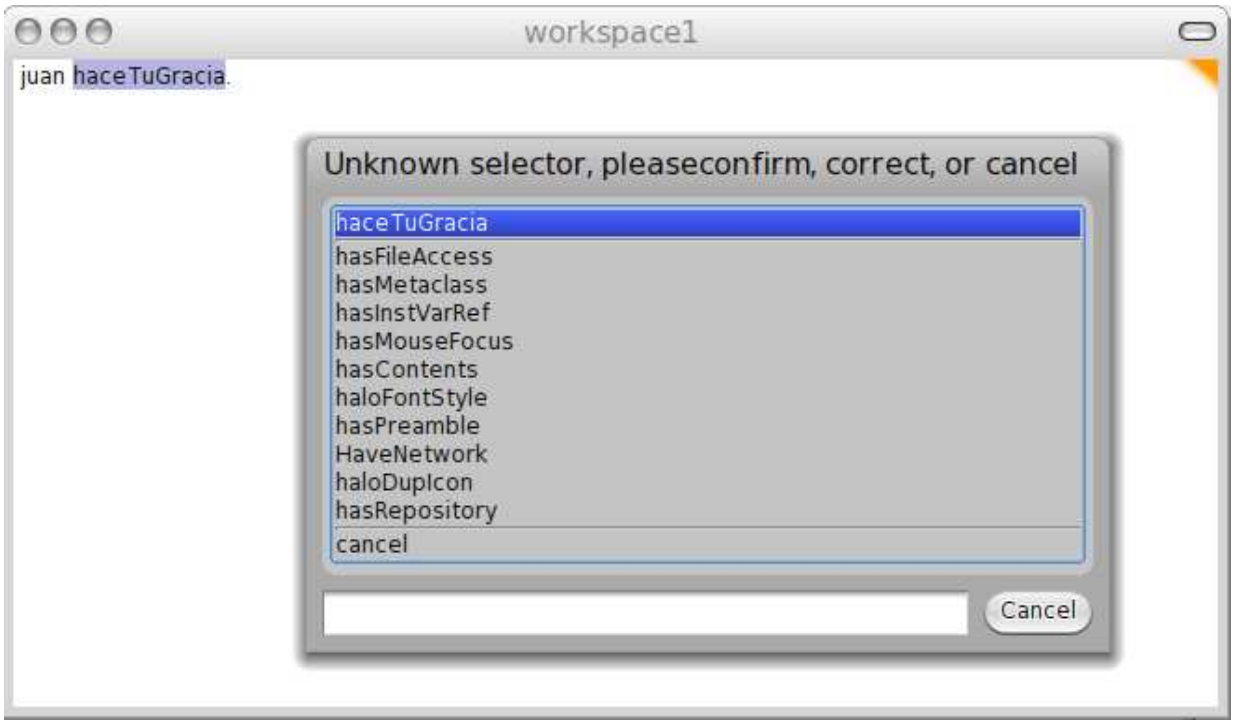
[2] Está sintaxis nace en el lenguaje Smalltalk, LOOP es configurable para tener sintaxis C-Like

```
nombreDeLaReferencia<PUNTO>mensaje<PUNTO Y COMA>
```

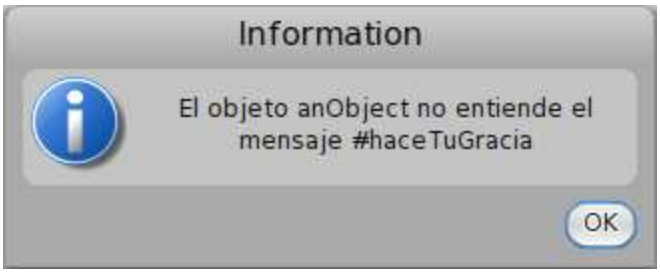
[3] CMD es la tecla Command utilizada por las computadoras Apple®. Cuando usemos esta notación puede reemplazarse por ALT o CTRL. Ejemplo CMD+S puede ser CTRL+S o ALT+S (dependiendo el caso)



Lo primero que nos dice LOOP es que nunca confirmemos, corrijamos o cancelemos ese selector porque es la primera vez que lo escribimos. Confirmamos haciendo click en el selector **#haceTuGracia**.



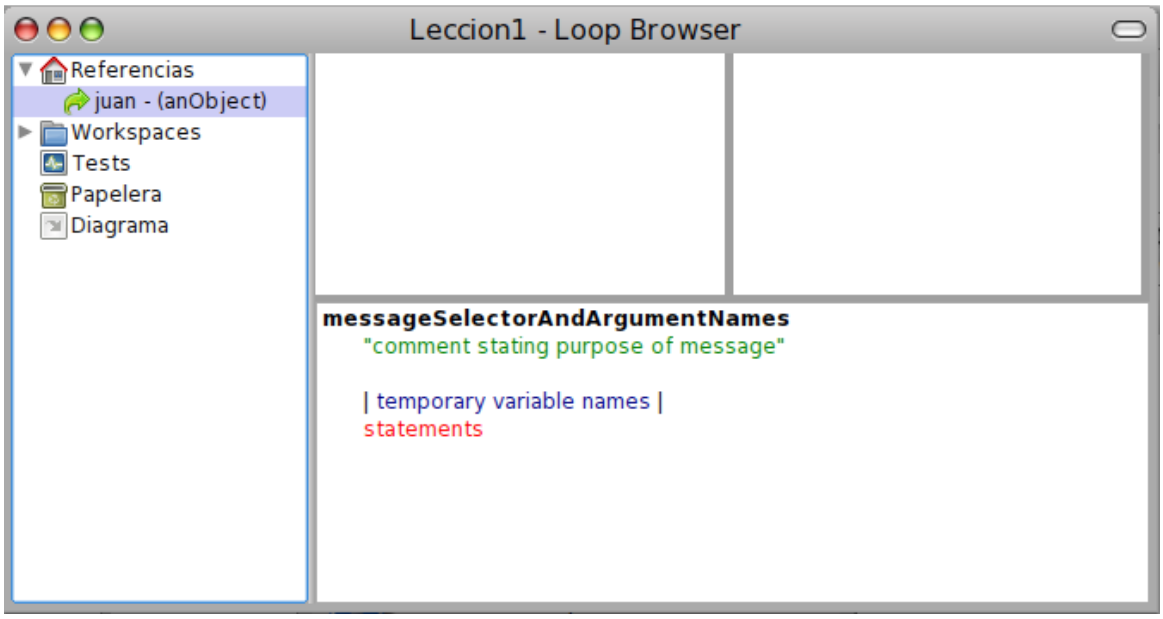
Una vez hecho esto, LOOP nos informa lo que esperábamos



CREACIÓN DE MÉTODOS

Lo que debemos hacer es agregar un método llamada **#haceTuGracia**.

Para hacer eso nos posicionamos sobre la referencia **juan**



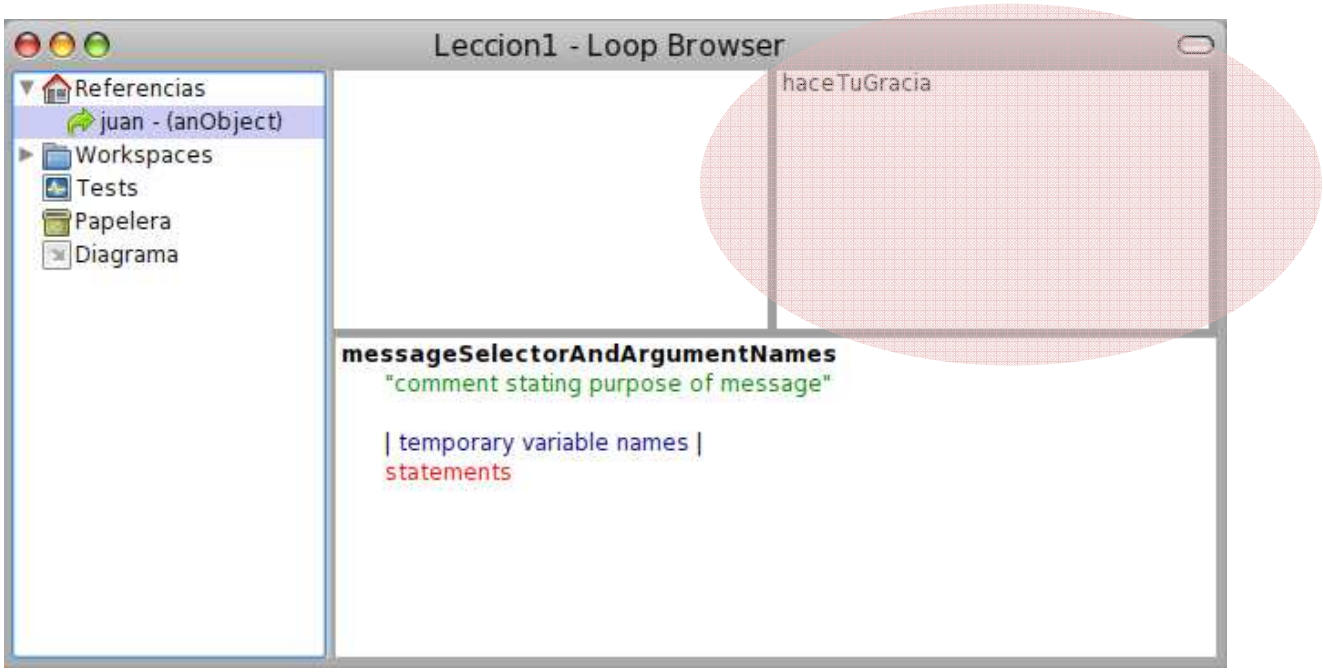
Aparece una plantilla (o template) que debemos reemplazar por el método que queremos agregar.

```
messageSelectorAndArgumentNames
  “comment stating purpose of message”
  | temporary variable names |
  statements
```

En nuestro caso queremos que el selector sea **#haceTuGracia**, y no tiene parámetros.
Para simplificar el ejemplo lo único que hace este método es retornar[4] el objeto 8.

[4] En LOOP el return se escribe ^ (que tiene su origen en el símbolo ↑)

Una vez que escribimos el método debemos guardarlo, haciendo botón derecho Aceptar o presionando las teclas CMD+s.



Para ver el método debemos hacer click en la lista de métodos que aparece a arriba a la derecha.

SINTAXIS

La sintaxis de LOOP es minimalista, ya que solo se basa en los conceptos del paradigma de objetos.

Nos va a interesar

- Enviar mensajes
- Que los mensajes retornen algún objeto en particular (solo 1 objeto)
- Poder decirle a una variable que referencie a un objeto determinado

Esto se hace de la siguiente manera

- `Objeto<ESPACIO>Mensaje<PUNTO>`
- `^Objeto`
- `Variable := Objeto`[5]

Donde dice `Objeto` nos referimos a un objeto o a algo que potencialmente termine siendo un objeto (i.e. el envío de un mensaje).

[5] En LOOP la sintaxis para la asignación es := y no debe confundirse con el mensaje = (igual)

Teniendo en cuenta esto

Código	*Objeto Receptor del Mensaje*	*Mensaje*	*Selector del mensaje*	*Parámetros del mensaje*	*Objeto Resultado*
2 + 3	2	+3	+	3	5
3 factorial	3	factorial	factorial	No tiene	6
6 between: 5 and: 8	6	between: 5 and: 8	between:and:	5 8	true
'hola' size	'hola'	size	size	No tiene	4
('hola' size) between: 5 and: 8	4 (el resultado de enviar size a 'hola')	between: 5 and: 8	between:and:	5 8	false

Dependiendo de la sintaxis seleccionada, lo que en un lenguaje con sintaxis C-Like[6] sería:

```
3.betweenAnd(5,8);
```

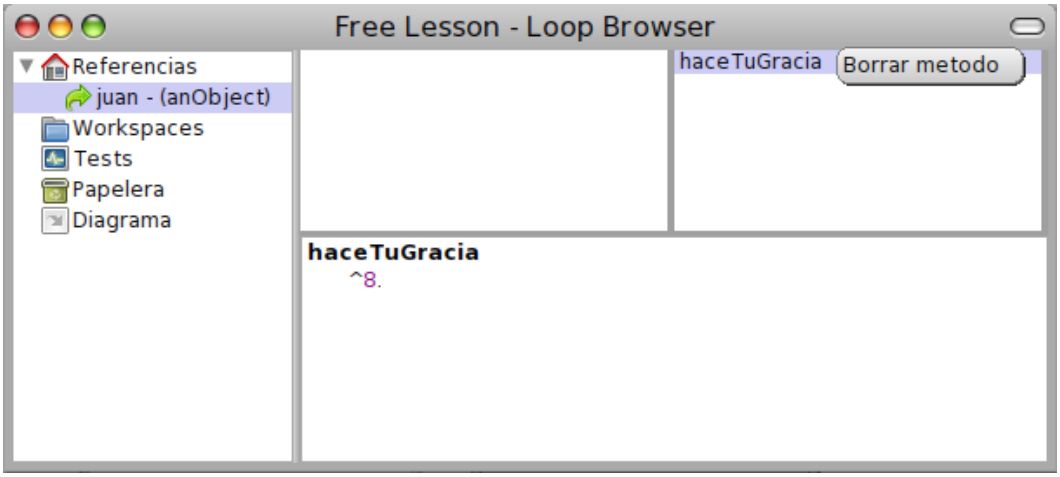
en LOOP (con la sintaxis por defecto) se escribe:

```
3 between: 5 and: 8.
```

Los parámetros están intercalados en el nombre del mensaje (selector) lo que mejora la lectura.

ELIMINACIÓN DE MÉTODOS

Para eliminar un método lo único que tenemos que hacer es posicionarnos en el método que deseamos borrar en la lista de métodos y hacer botón derecho **Borrar método**



ATRIBUTOS

[6] Los lenguajes con sintaxis C-Like tienen una sintaxis similar al lenguaje C

> *Un programa dentro del paradigma de objetos es un conjunto de objetos que interactúan entre sí envíandose mensajes*

Para que los objetos interactúen entre sí se deben conocer de alguna forma.

Un objeto A conoce a un objeto B si

- B es uno de los objetos que ya vienen con LOOP (números, strings, booleanos, etc.)
  A estos objetos los vamos a llamar **literales**
- B es un **parámetro** de un mensaje que se le envío a A
- B es un objeto que A conoce a través un **atributo**

Un **atributo** es un variable definida en un objeto, a través de dicha variable (o referencia) un objeto puede conocer a otro y mantener ese conocimiento a través del tiempo.

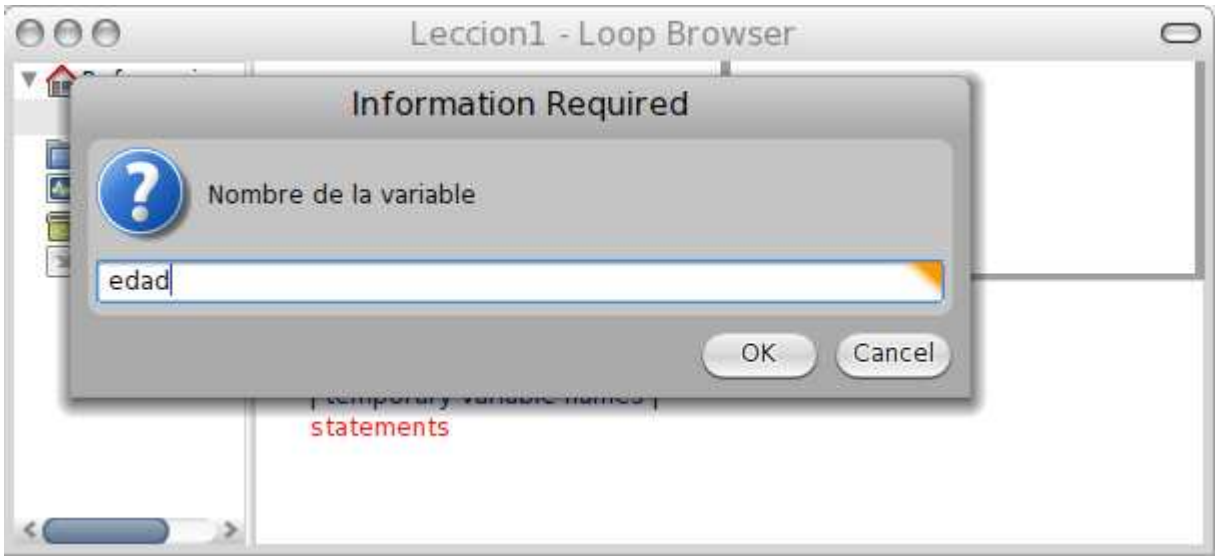
Ejemplo:

Si queremos que el objeto referenciado por la variable **juan** conozca su *edad* a través del tiempo necesitamos que ese objeto tenga un atributo *edad*.

CREACIÓN

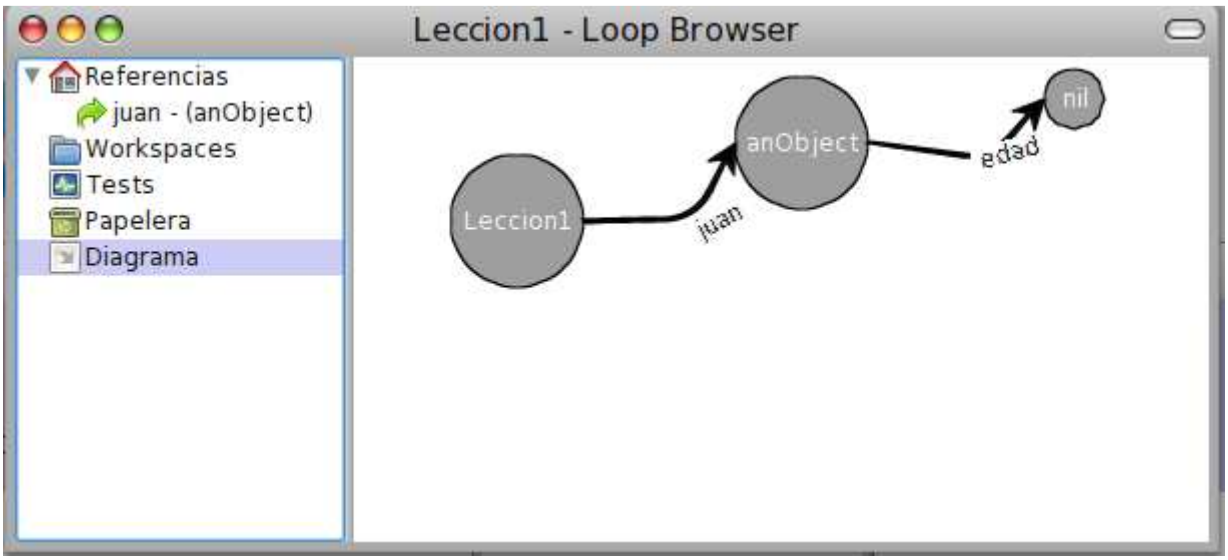
Para hacer eso nos paramos en la referencia **juan** y hacemos botón derecho

Elegimos el nombre de la variable y le damos **Ok**



> *Una variable siempre referencia a uno y solo a un objeto. Si no se especifica a que objeto apunta una variable dicha variabla* ***apuntará al objeto nil***

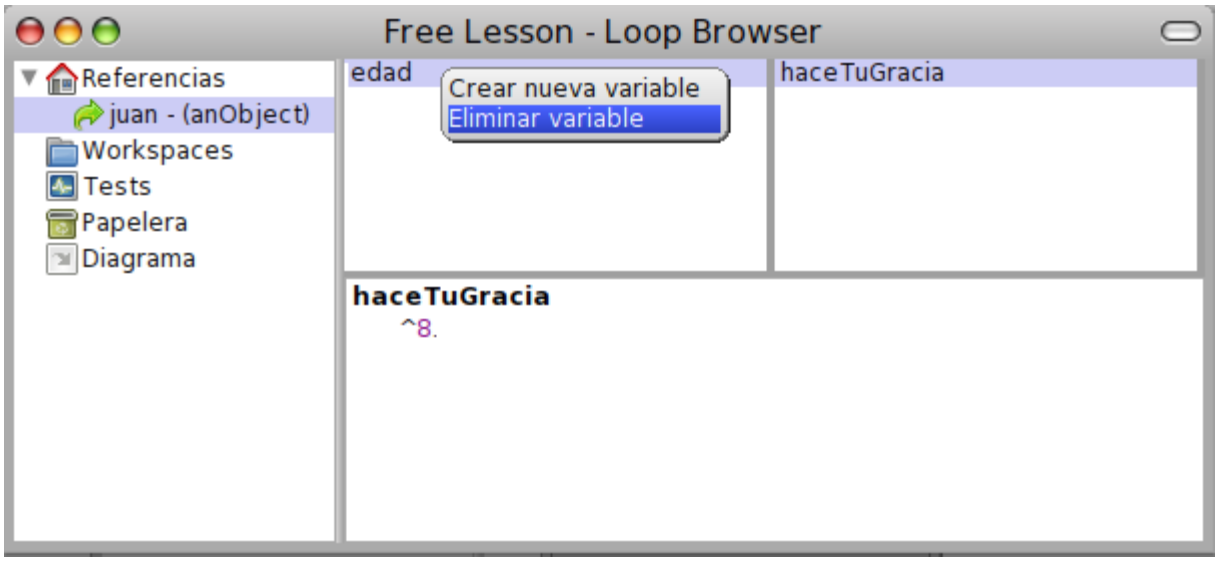
Para ver que ocurrío luego de agregar el atributo **edad** podemos ver el Diagrama



Actualmente la variable edad apunta a nil.

ELIMINACIÓN

Para eliminar un atributo debemos posicionarnos sobre el atributo a eliminar y hacer botón derecho **Eliminar variable**



ACCESSORS[7]

Si desde un workspace queremos conocer la edad de anObject no podemos escribir solo edad, porque ese atributo pertene a anObject. Lo único que podemos hacer es enviarle un mensaje a anObject y que el se encargue de conseguir el objeto deseado.

```
juan edad.
```

El método debería tener la siguiente implementación[8]

```
edad
  ^edad.
```

Desde un workspace podemos preguntar si la edad de anObject es mayor que 18

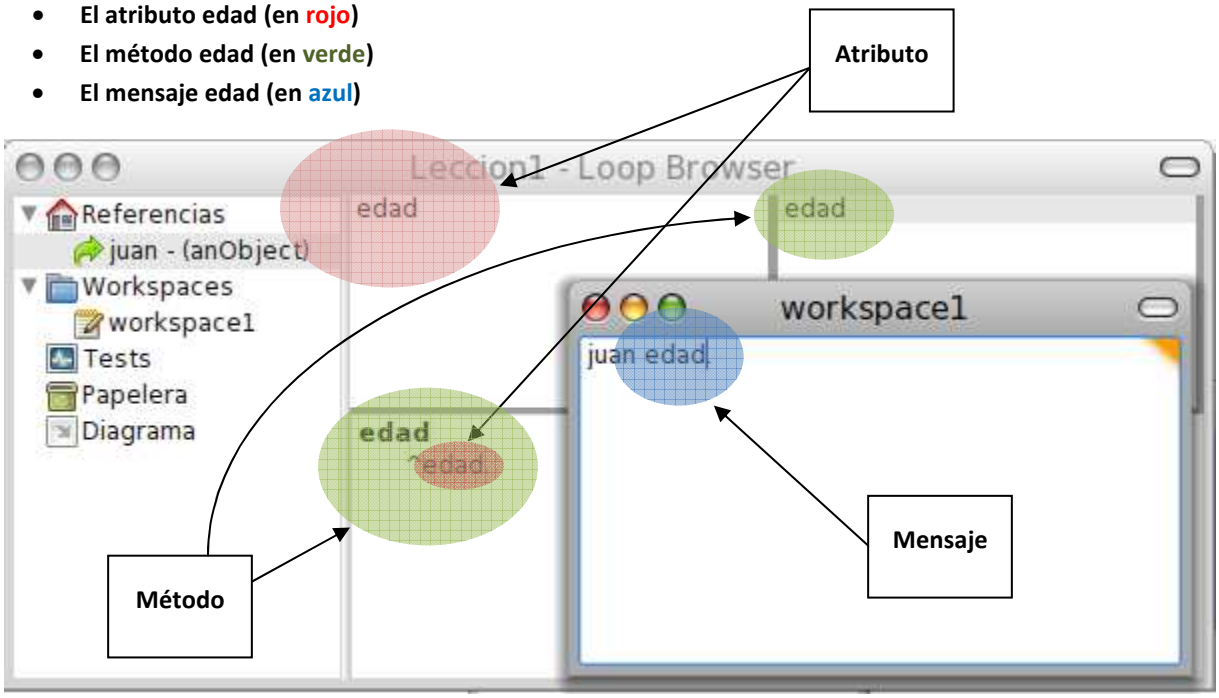
```
juan edad > 18.
```

Enviar el mensaje edad nos retorna el objeto `nil` y `nil` no entiende el mensaje > (mayor) lo que generará un error.

[7] Un accessor es un **getter** o un **setter** (ver más abajo)
[8] La implementación del método **edad** es solo retornar el valor de la variable `edad`. A estos métodos se los llama **getter**

Importante: ahora tenemos 3 cosas que se llaman edad

- **El atributo edad (en rojo)**
- **El método edad (en verde)**
- **El mensaje edad (en azul)**



¿Cómo podemos hacer para que el atributo edad apunte al objeto 24?

El único que puede acceder a los atributos (variables) de un objeto es ese mismo objeto y la única forma de pedirle a un objeto que realice una acción es enviándole un mensaje.

Entonces, tenemos que enviarle un mensaje al objeto referenciado por la variable juan, por ejemplo **initialize**[9].

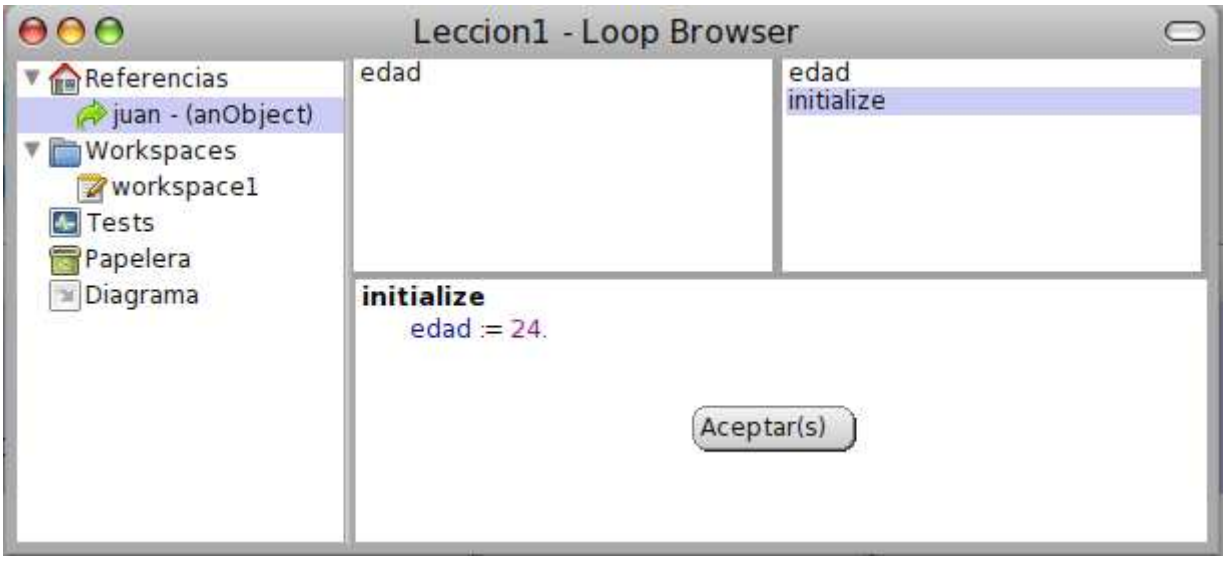
Para que entienda este **mensaje** debemos escribir un **método**.

Hacemos click en la referencia juan y agregamos el siguiente método

```
initialize
   edad := 24.
```

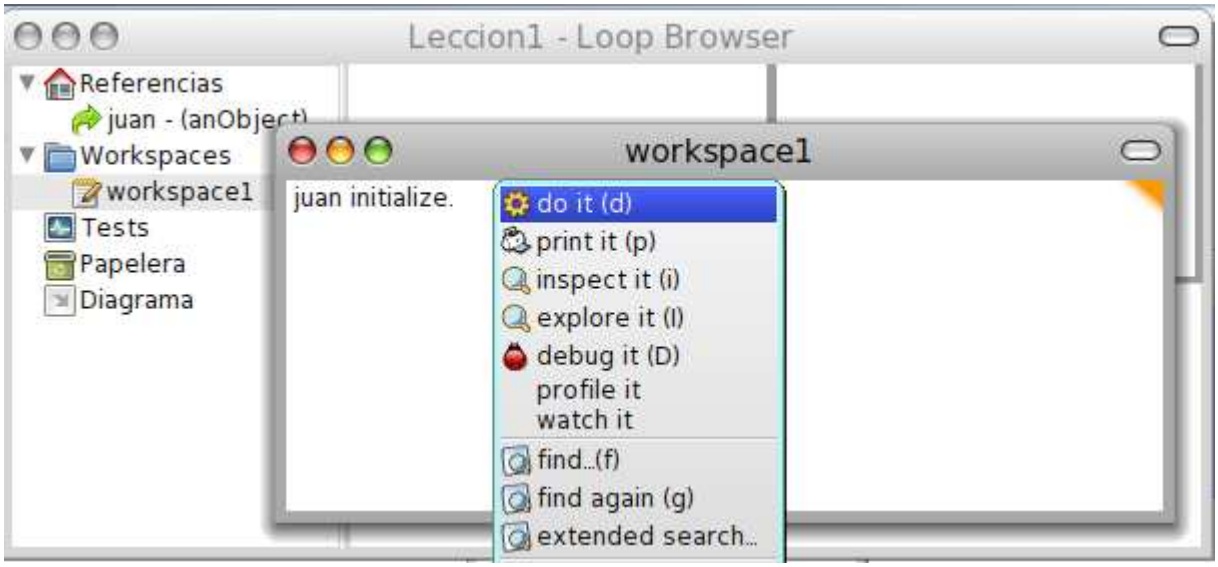
Dentro del método **initialize** estamos realizando una asignación de la variable edad (conocida desde el objeto en el que estamos posicionados) hacia el objeto 24 (objeto conocido porque es un literal).

[9] Por convención vamos a usar este selector para el mensaje por el cual un objeto se inicializa (le asigna a sus atributos los objetos que cree conveniente al momento de ser creado por primera vez)



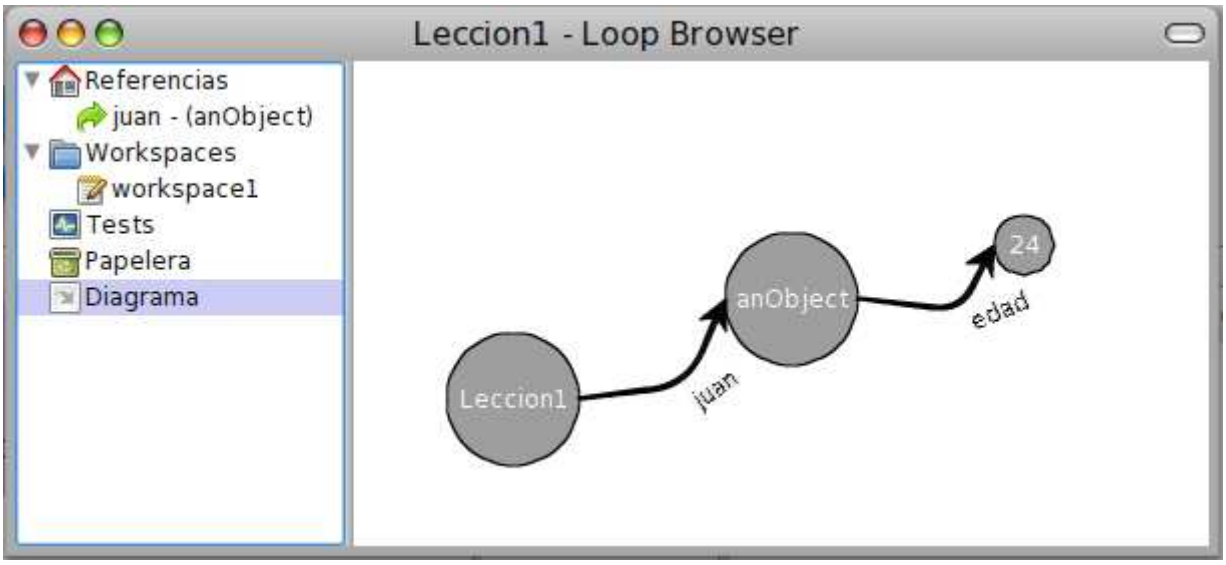
Recordamos guardar el método haciendo botón derecho Aceptar o CMD+S

Una vez guardado el método podemos verlo en la lista de métodos y ya podemos enviarle un mensaje con ese selector desde un workspace.



Haciendo botón derecho Do It o CMD+D ejecutamos el código.

Para ver cómo quedo nuestro ambiente luego de enviar este mensaje vemos el **Diagrama**.



Tenemos las siguientes preguntas

1. ¿Cómo podemos hacer para que edad deje de apuntar al 24 y después apunte al 25, y después al 26 y así ...?
2. ¿Cómo podemos hacer para que anObject cumpla años?

Cuando pensamos con las ideas del paradigma de objetos nos interesa pensar en términos de la segunda pregunta.

No me interesa cómo hace el objeto internamente para guardar su edad, yo lo que quiero es que cumpla años.

Queremos que anObject entienda el mensaje **cumpliAños** y para que esto funcione debemos escribir un método con el mismo selector.

```
cumpliAños
    edad := edad + 1.
```

Todo lo que está a la derecha de la asignación (`edad + 1`) tiene que cumplir con la sintaxis **objeto<ESPACIO>mensaje<PUNTO>** por ende **no** nos interesa que edad sea una variable sino que apunta al objeto 24.

En cambio, lo que está a la izquierda de la asignación (*edad*) tiene que ser una variable, ahí vemos a *edad* como una referencia y no como al objeto al que hace referencia.

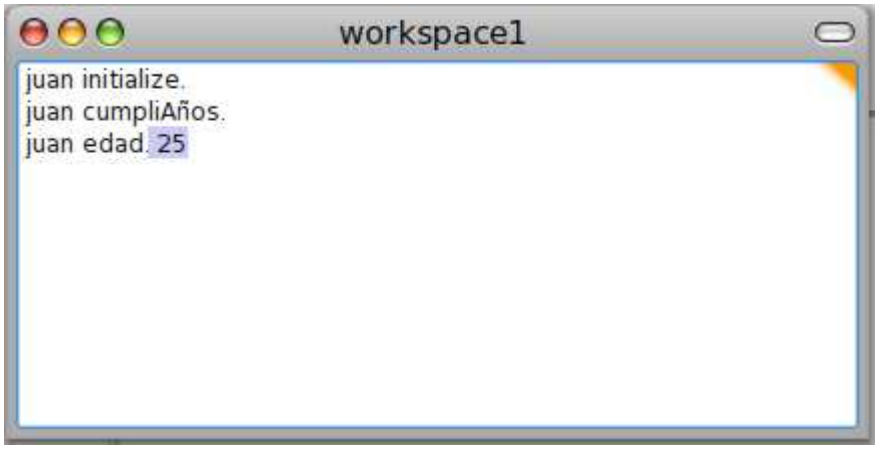
Si en workspace hacemos

```
juan initialize.
juan cumpliAños.
   juan edad.
```

Si seleccionamos todo con el mouse y lo ejecutamos (botón derecho *Do It* o CMD+D) no vamos a ver nada, cuando lo normal sería esperar como respuesta el objeto 25.

Para lograr esto no solo queremos ejecutar las líneas seleccionados sino **mostrar por pantalla el objeto resultado** de la última línea seleccionada, para ello debemos hacer botón derecho *Print It* o CMD+P.

Al hacer esto obtenemos



CLONACIÓN

Hoy tenemos a un objeto referenciado por la variable juan que entiende los mensajes

- initialize
- edad
- cumpliAños

Además este objeto tiene un atributo llamado edad.

¿Qué pasa si queremos tener otro objeto (referenciado por la variable juana) entienda los mismos mensajes?

Una solución es crear un nuevo objeto, definir 3 métodos y agregarle un atributo.

Pero, ¿si necesito una cantidad importante de objetos que tengan este comportamiento?
No parece una solución viable estar haciendo copy & paste una y otra vez.

Más importante aún, si el día de mañana cambia la forma en que se representa internamente la edad (por ejemplo en vez de guardarse los años se conoce el año actual y se le resta el año de nacimiento), quiero que ese cambio se vea reflejado en juan, en juana y en todos estos objetos que yo dije que tenían "el mismo comportamiento".

> ***Once & Only Once**: la lógica (e.g. los métodos) no debe estar repetida*

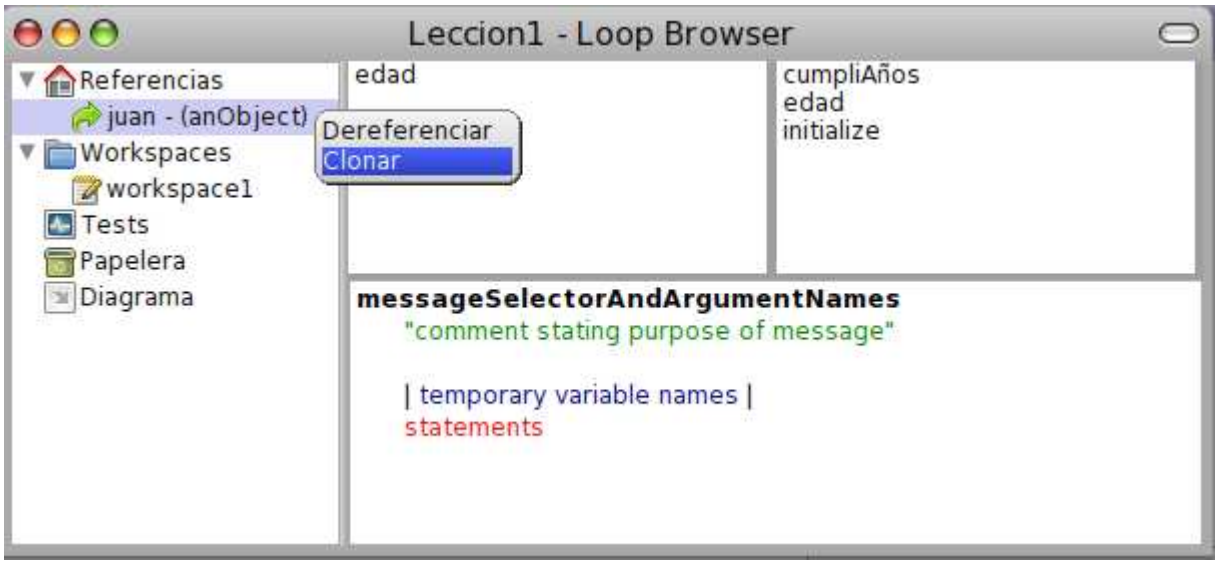
Para hacer esto podemos tomar al objeto referenciado por la variable juan como nuestro prototipo para crear nuevos objetos, cuando se le envie un mensaje a estos nuevos objetos van a buscar su comportamiento en ese prototipo pero que cada uno tiene sus propios atributos (cada uno tiene un estado interno diferente).

COMPORTAMIENTO COMÚN

> *Clonación (Prototype-Based)[10]: proceso por el cual creamos nuevos objetos a partir de un objeto prototipo, a estos nuevos objetos se los conoce como clones de ese objeto prototipo.*

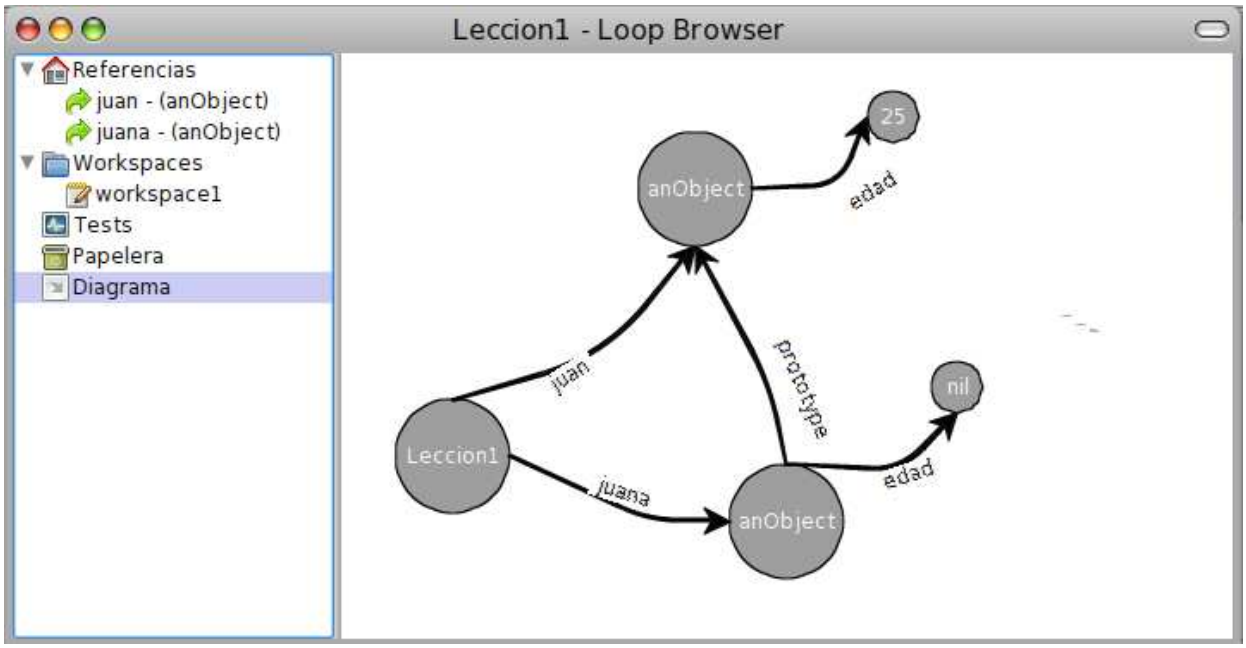
Para clonar al objeto referenciado por la variable juan tenemos que ir a **Referencias**, posicionarnos sobre juan y hacer botón derecho **clonar**.

[10] Lenguajes como JavaScript, Self, Perl, etc. usan este mecanismo



Escribimos el nombre de la nueva referencia a este nuevo objeto (en este caso juana) y le damos **Ok**.

Para ver como quedo nuestro ambiente podemos ver el **Diagrama**.



La lección tiene 2 referencias a 2 objetos distintos (juan y juana)

Cada uno de estos objetos tiene su propio atributo edad

- La edad del objeto referenciado por la variable juan es 25
- La edad del objeto referenciado por la variable juana es nil

> ***Method-Lookup (con Clonación):*** *cuando se le envía un mensaje al objeto A con el selector X se busca el método llamada X en ese objeto, de no encontrarlo se busca dicho método en el prototipo. Este proceso continúa hasta llegar a un objeto que no*

> *posea prototipo, dando el error "el objeto A no entiende el mensaje". Si se encuentra el método X se ejecuta sobre el objeto A.*

En la situación actual el objeto referenciado por la variable **juana** no tiene definido ningún método **PERO** entiende 3 mensajes.

COMPORTAMIENTO PARTICULAR

Vimos como no repetir el comportamiento que tienen varios objetos en común pero ¿qué hacemos cuando el comportamiento no es el mismo?.

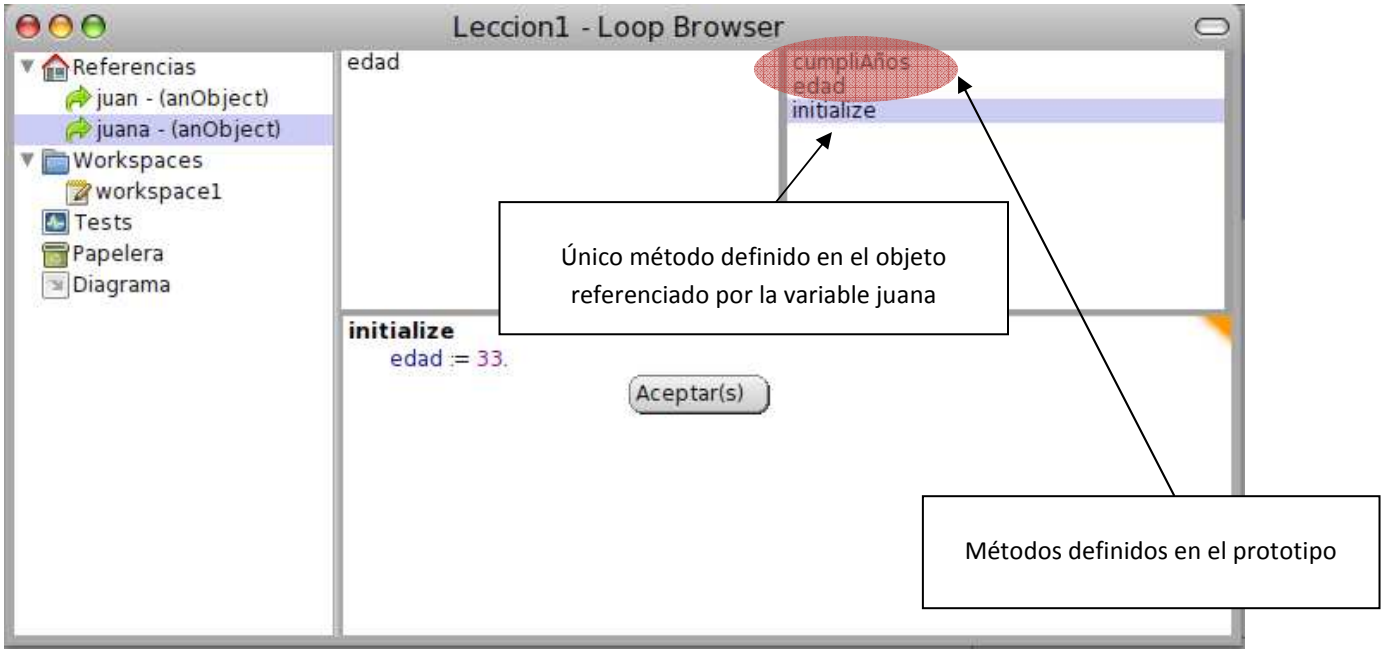
Ejemplo:

El objeto referenciado por **juana** cuando se inicializa hace que su variable edad apunte a 33

El método sería

```
initialize
   edad := 33.
```

Si este comportamiento solo es válido para el objeto referenciado por la variable **juana** debería ser un método que está solo definido en juana.

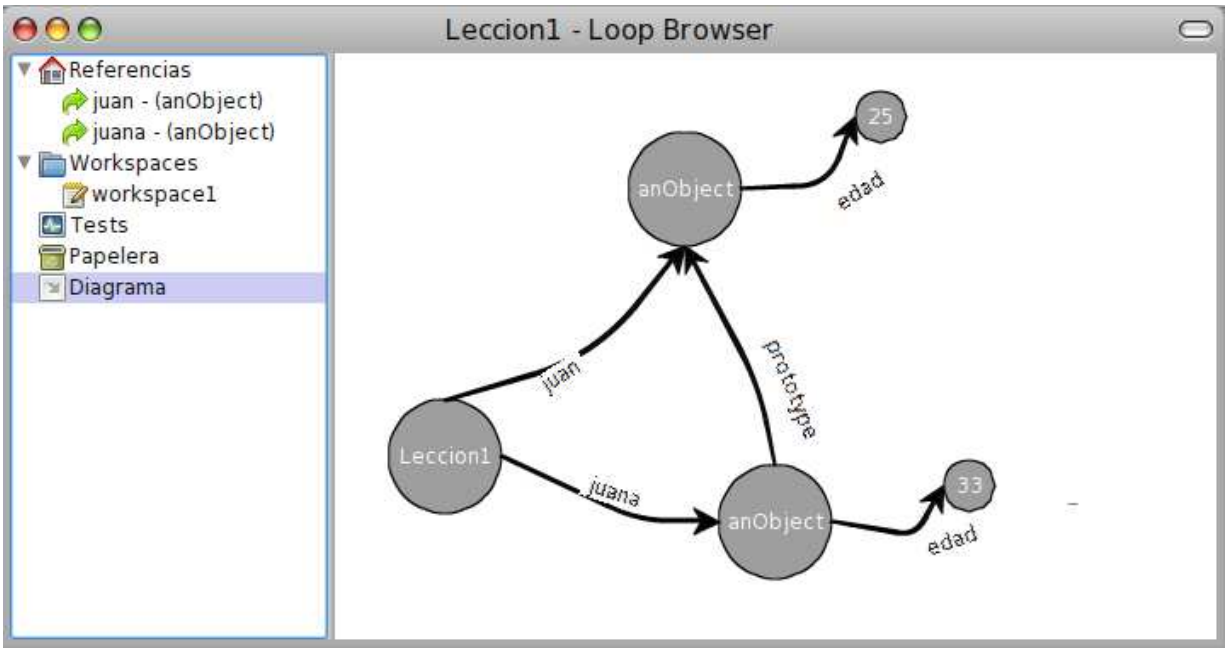


Si en un workspace ejecutamos:

```
juana initialize
```

El Method-Lookup se va a encargar de que se ejecute el **initialize** que le asigna a edad el objeto 33 (no le interesa si hay un initialize en el prototipo, una vez que encontró el método que buscaba el proceso de búsqueda finaliza).

Luego de ejecutar `juana initialize` en un workspace vemos en el **Diagrama**

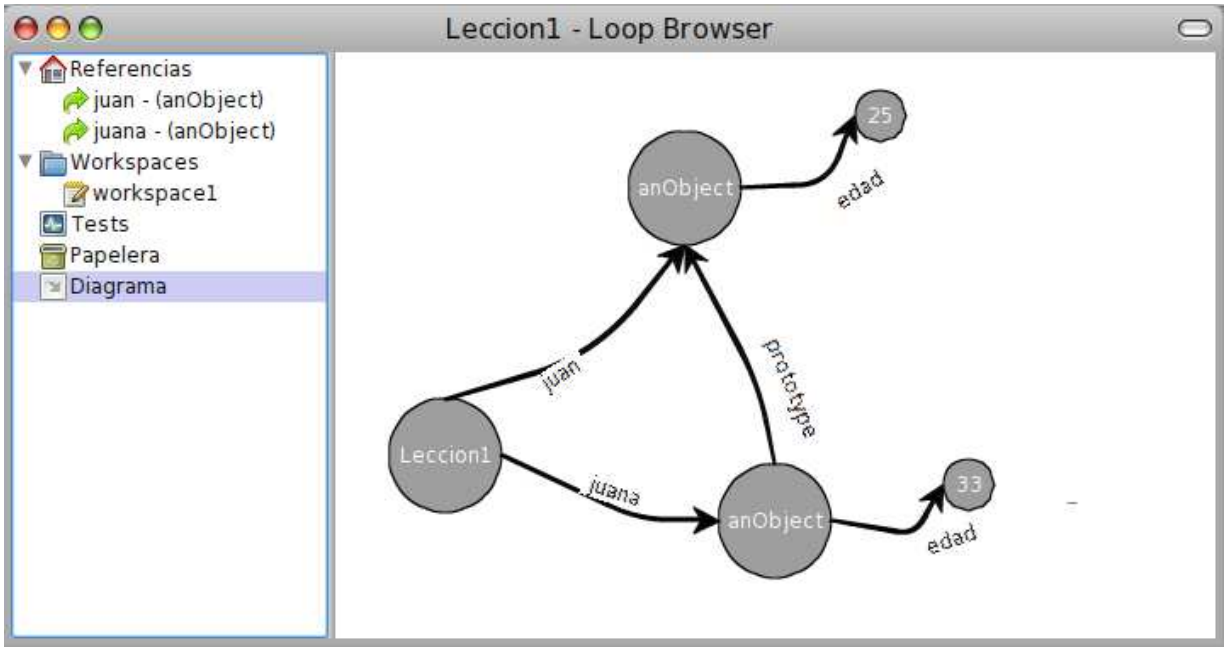

Leccion1 - Loop Browser
Referencias
juan - (anObject)
juana - (anObject)
Workspaces
workspace1
Tests
Papelera
Diagrama
25
anObject
edad
juan
prototype
Leccion1
juana
33
anObject
edad

GARBAGE COLLECTOR

Las referencias de la lección pueden a apuntar a distintos objetos.

¿Qué pasa cuando cambiamos el objeto al que apunta una de las referencias de la lección?

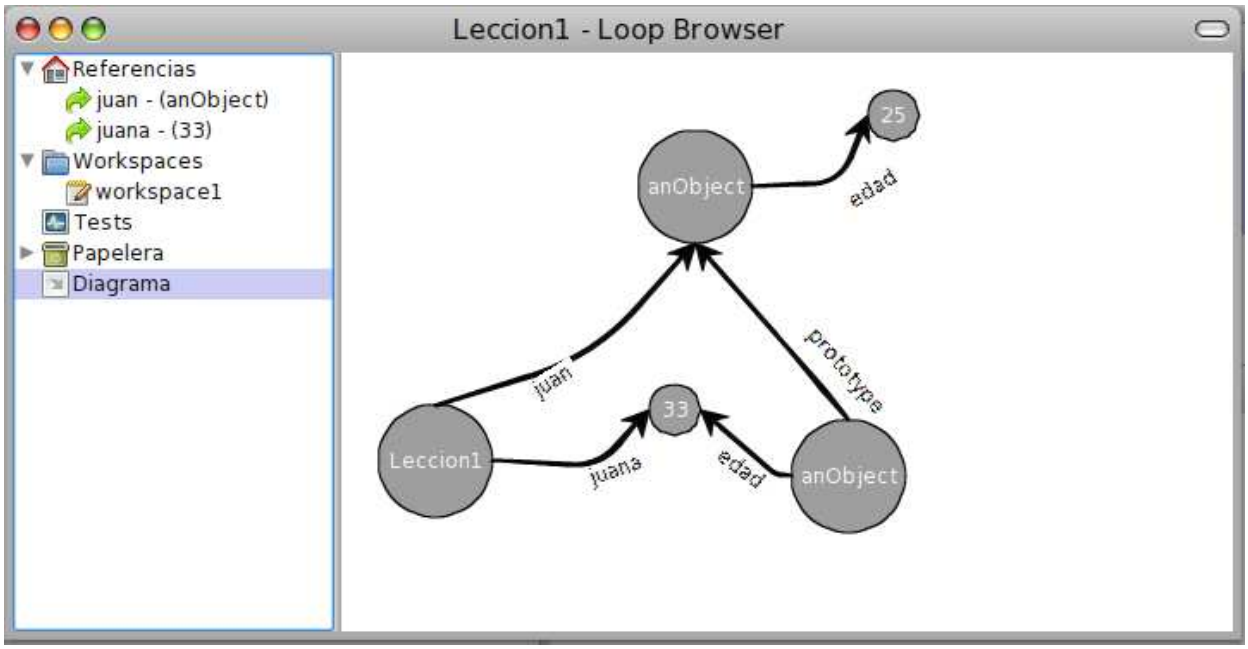
Si tenemos este ambiente



Y en workspace escribimos

```
juana := 33
```

Podemos observar que la variable juana deja de a apuntar a `anObject` (el que está más abajo) y pasa a apuntar al objeto 33 (ver **Diagrama**)



Si quisieramos hablar con el anObject que está más abajo, no tenemos ninguna referencia hacia él. Por lo tanto se dice que anObject se ha convertido en un objeto que no puede ser accedido.

> *Un objeto solo puede ser accedido si existe alguna referencia hacia él. Decimos que un objeto no puede ser accedido si en el diagrama no existe ninguna flecha que llegue a dicho objeto*

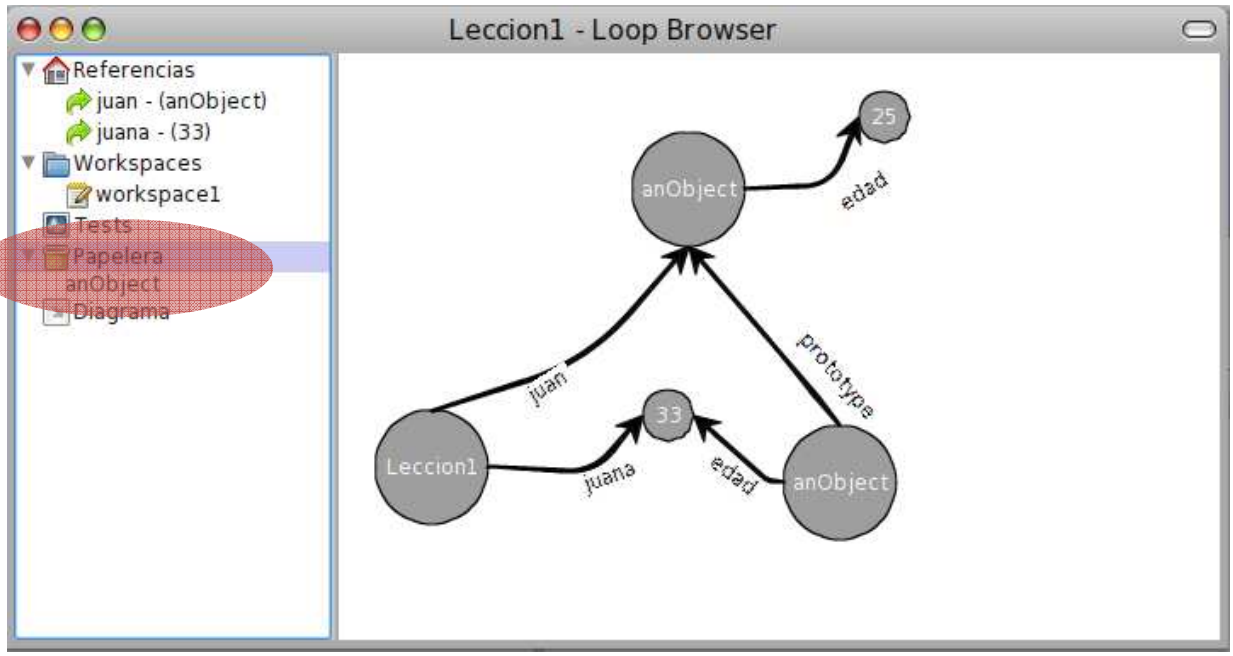
Si la referencia juana apuntara a otro objeto distinto del 33, la única referencia hacia el objeto 33 sería el atributo edad.

Ahora imagínense que se elimina al objeto anObject (que tiene su atributo edad apuntando al 33), cuando se elimina ese objeto se eliminan sus referencias.

Si juana no apuntara al 33 y edad no existiera, el 33 también sería un objeto que no puede ser accedido.

PAPELERA

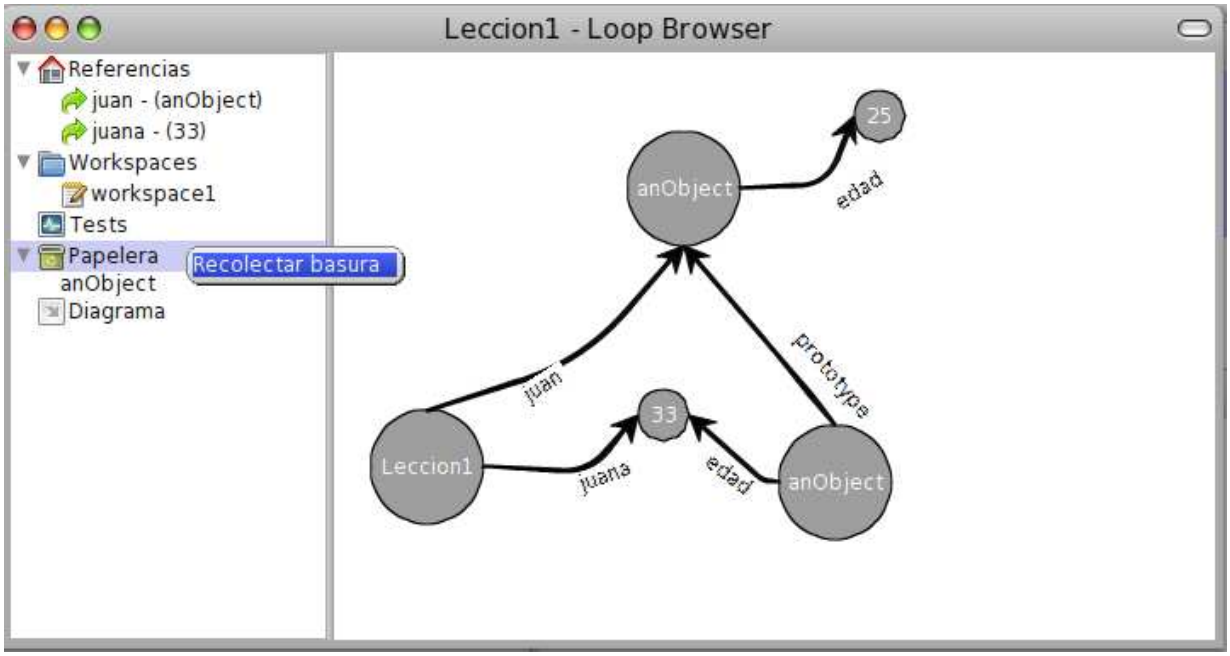
Podemos observar que ninguna flecha **llega** a ese objeto `anObject` y por lo tanto ese objeto paso a la **Papelera**.



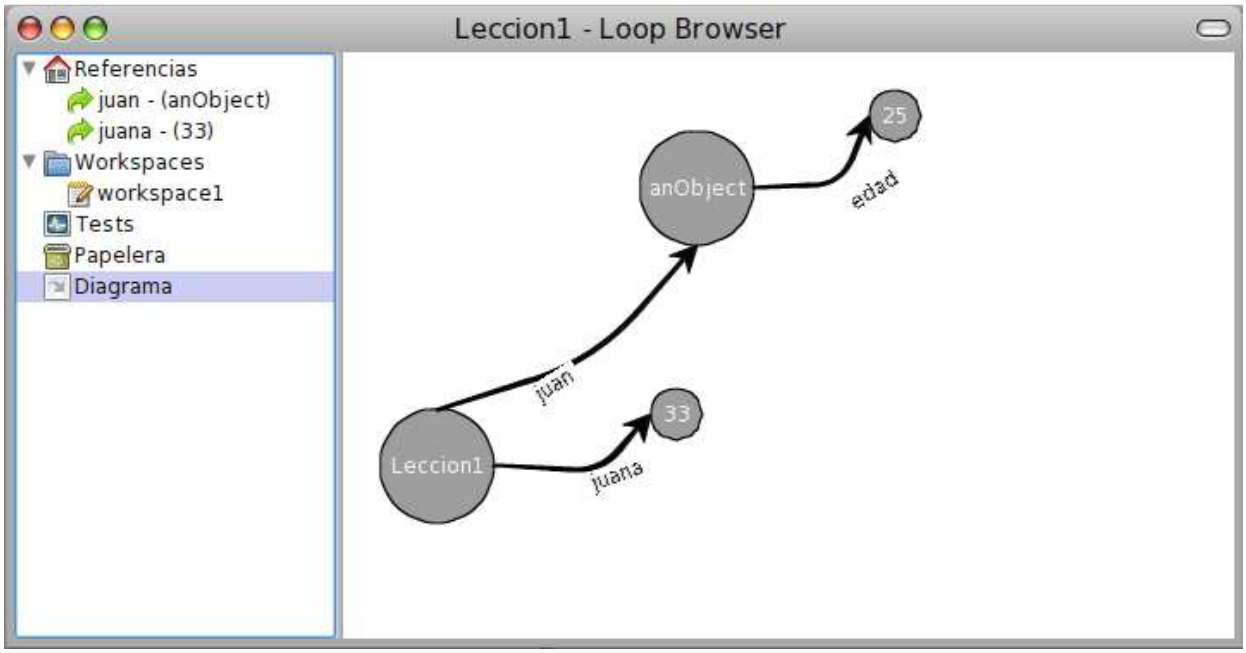
Los objetos que están en la papelera pueden ser eliminados ya que nadie puede mandarles mensajes. Para hacer esto debemos correr el Gargabe Collector

> *Garbage Collector (GC): mecanimos por el cual se liberan los recursos de los objetos que no pueden ser accedidos*

En LOOP podemos correr el GC de forma manual haciendo botón derecho sobre la Papelera y seleccionando **Recolectar basura**



Si actualizamos el Diagrama podemos ver el estado actual del ambiente



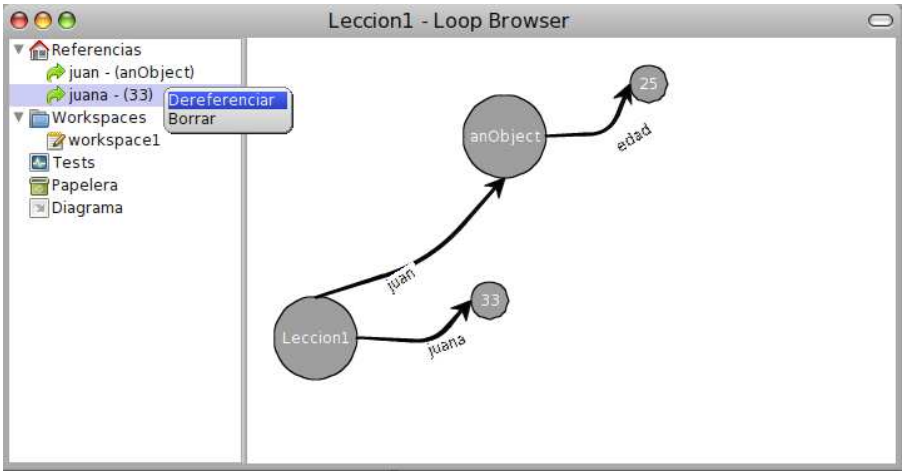
ELIMINANDO REFERENCIAS

Teniendo en cuenta el diagrama anterior, podemos ver que la referencia juana no nos es de mucha utilidad.

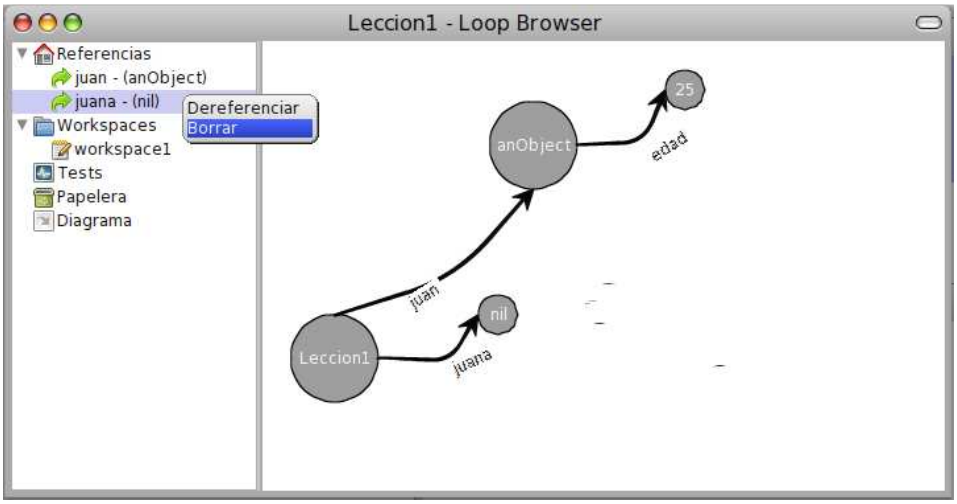
Para eliminar esa referencia debemos seguir 2 pasos

1. Desreferenciarla (hacer que apunte al objeto nil)
2. Eliminar la referencia

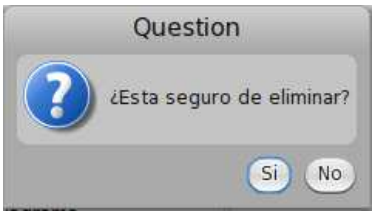
Paso 1



Paso 2



Se nos pide una confirmación



ÍNDICE