Testing en Ozono

Probar una aplicación permite verificar que el sistema (o una parte de él) funciona de acuerdo a lo especificado. Los tests permiten bajar la incertidumbre y aumentar la confianza que tenemos sobre el software que desarrollamos.

Hay diferentes formas de probar una aplicación, por ejemplo si tiene una interfaz de usuario el testeo podría realizarse manualmente siguiendo los pasos necesarios (eligiendo opciones de algún menú, llenando formularios, etc) para verificar que el sistema responde de la forma esperada. Programáticamente estamos acostumbrados a probar el sistema usando la funcionalidad desarrollada desde el mismo entorno de desarrollo, por ejemplo escribiendo en un workspace de Smalltalk. Supongamos que tenemos este ejemplo:

```
#pepita
puedeVolar
    ^ self getEnergia > 20
vola: unosKilometros
    self setEnergia: (self getEnergia - (unosKilometros * 2 - 8)).
```

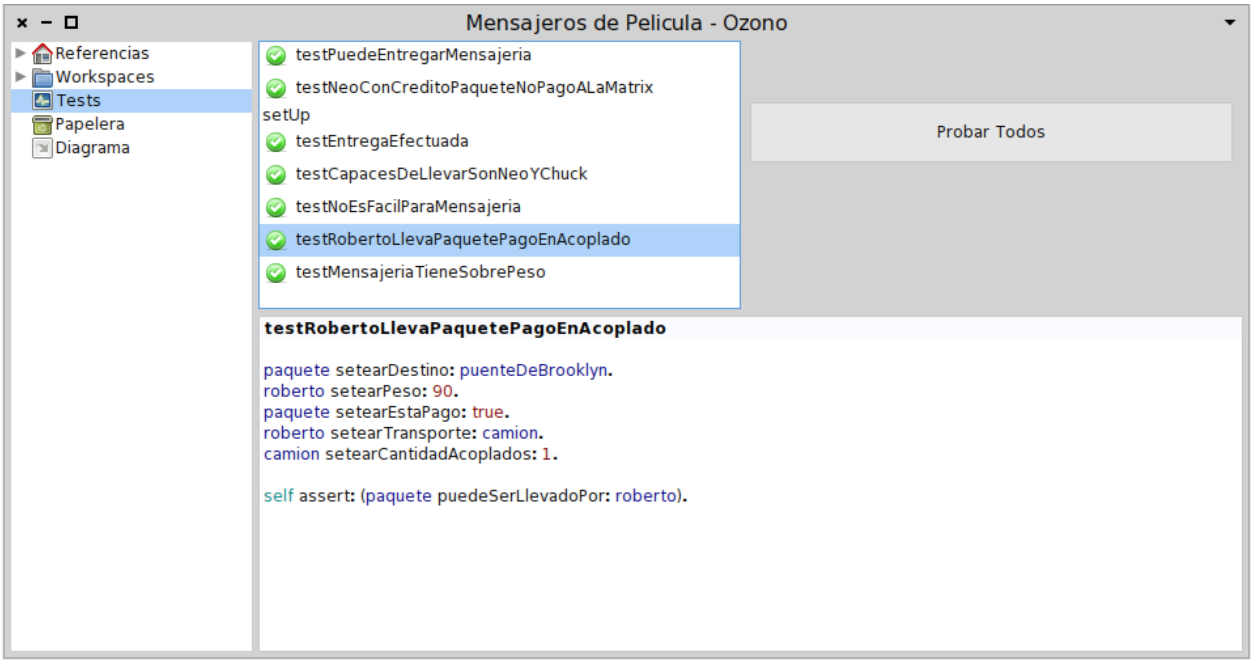
```
##WorkSpace

pepita energia: 100.
pepita vola: 25. "Le resta 25*2+8 de energía"
pepita getEnergia. "Debería retornar 42"
pepita puedeVolar. "Me debería decir que sí porque tiene más de 20 de energía"
pepita vola: 10. "Le resta 10*2+8 de energía"
pepita puedeVolar. "Me debería decir que no porque tiene menos de 20 de energía"
```

Y al ejecutar ese workspace se puede verificar si el valor retornado por el mensaje energia luego de volar 20 kilómetros retorna el valor que esperamos que retorne, si pepita puede volar cuando tiene más de 20 de energía y que no puede cuando tiene menos.
Sin embargo existe otra forma de realizar pruebas de forma sistemática que permite automatizar todo el proceso de testeo, ya que verificar que el valor de la energía de pepita sea el esperado es algo que hay que hacer manualmente. En Smalltalk vamos a usar un framework llamado SUnit (existe también para otros lenguajes con otro nombre, búsquenlo, aprovéchenlo!) que lo que permite es codificar no sólo los pasos que describen la prueba, sino también la validación de los resultados esperados.

Automatizar las pruebas requiere un esfuerzo extra, pero es una gran inversión porque esas pruebas quedan disponibles para ser ejecutadas N veces en el futuro. A medida que el sistema crece, la cantidad de casos a probar para verificar que el sistema completo está bien construido aumenta muchísimo, y poder asegurar que toda la funcionalidad que programé ahora y que fue programada antes sigue funcionando es muy valioso.

Testing en Ozono:

Para crear tests de nuestra lección en ozono iremos una opción del menú izquierdo de la lección llamada Tests.
Allí Ozono nos dejará escribir métodos en el rectángulo de abajo. Cada test entonces será un método escrito allí y cuyo nombre siempre empieza con la palabra “test”, seguido de un nombre que explique de forma clara que testea (sin importar demasiado el largo del nombre resultante):



Podemos decir que un test entonces, como vemos en la imagen, se divide de dos partes:

- Preparar los objetos a un estado conocido y necesario para realizar la prueba.
- Verificar que la acción que realizamos dé el resultado esperado.

Para la primera parte haremos lo mismo que haríamos en el Workspace, enviamos mensajes a nuestros objetos, para setear sus atributos y dejarlos en determinado estado.
Para lo segundo tenemos un par de opciones de lo que queremos hacer:

- *Verificar que se cumpla una condición*
  Esto lo haremos mediante el siguiente envío de mensajes:

```
self assert: (paquete puedeSerLlevadoPor: roberto).
```

Así si la condición da true, nuestro test pasará.

- *Verificar que se <u>NO</u> cumpla una condición*
  Si por el contrario esperamos que la condición no se cumpla entonces escribiremos:

```
self deny: (empresaMensajeria consideraPaqueteFacil: paquete).
```

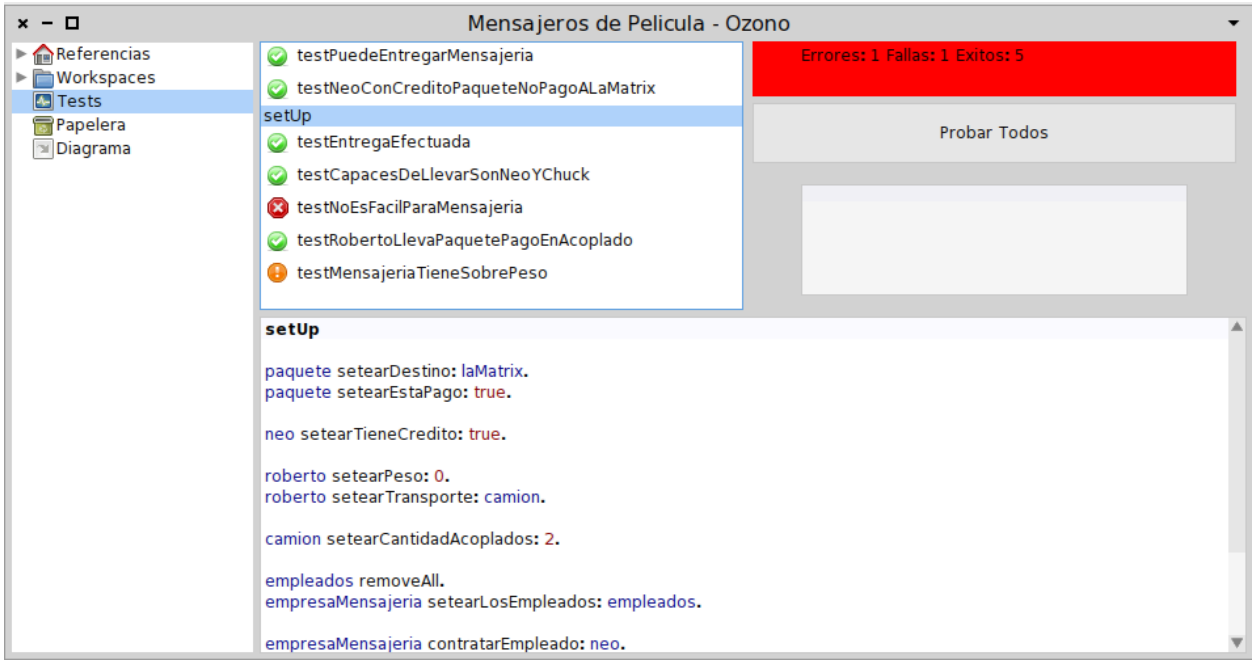
Así si la condición da false, nuestro test pasará.

- *Verificar que se obtenga cierto objeto como respuesta*
  Si queremos comprobar que el objeto resultante de un envío de mensajes es uno en particular

```
self assert: (neo pesoTotal) equals: 0
```

Así si el primer parámetro es igual que el segundo, el test pasará.
Tengan en cuenta que los tests que realizamos son generalmente unitarios, lo que significa que testean una cosa a la vez, por lo que en cada test debería tener uno o hasta no mucho más de 3 asserts o denys, porque sino cuando un test falla no podríamos identificar fácilmente cuál de todos fue el que falló.

<u>Correr los Tests:</u>
Una vez hechos nuestros tests llega el momento de la verdad… probar que den lo que deberían!.
Para eso apretamos en el botón a la izquierda de cada nombre si queremos correr uno en particular o en probar todos para… bueno probar todos :P.
Después de ejecutarse nos mostrará algo así:



Mmmm, no se ve demasiado bien, veamos qué sucedió.
Por un lado todos aquellos tests que aparecen con un cartel verde significa que pasaron :D !.
Los amarillos que fallaron, es decir, que uno o más resultados arrojados no fueron los esperados. Y por último los rojos arrojaron errores, como por ejemplo que un objeto no entiende el mensaje enviado.

Método setUp:

Una de las primeras cosas que dijimos para hacer tests en ozono es que el nombre del método de cada test debía empezar con “test”, entonces, ¿por qué hay un método que se llama “setUp”?
Porque no es un test!.
Este método setUp es un método que se ejecuta siempre antes de correr cada test, se utiliza para setear el estado común que tenemos a los casos que queremos probar.
Por ejemplo si nuestro amigo Leandro es un objeto que se va de compras por el supermercado, y quiere testear qué pasaría según qué compras realiza, podríamos decir que antes de cada caso de prueba siempre está con la billetera llena (tiene $1000) y con el changuito vacío, por lo tanto nuestro setUp se vería masomenos así.

```
setUp
lean setDinero: 1000.
lean vaciarChango.
```

Así ya no nos importa si dentro de un determinado test le pusimos al changuito articulos y lo dejamos sin plata al pobre Leandro, sabemos que para cuando comience otro test, Leandro volverá a con el chango vacío y felíz con su bolsillo lleno.