

# P8- Pruebas de aceptación con Selenium IDE

## Pruebas de aceptación de aplicaciones Web

El objetivo de esta práctica es automatizar pruebas de aceptación de pruebas emergentes funcionales sobre una aplicación Web, para lo que utilizaremos la herramienta Selenium IDE, junto con el navegador Firefox. Implementaremos los drivers para las pruebas con *scripts* l de comandos Selenese (que posteriormente guardaremos en formato html). Tal y como hemos visto en clase, Selenium nos permite "capturar" las acciones del usuario en el navegador para generar de forma automática las instrucciones de nuestros drivers, los cuales pueden ejecutarse de forma automática tantas veces como sea necesario. Esta forma de uso de la herramienta ("grabando" las acciones del usuario sobre el navegador) es lo que se conoce como el patrón *record-playback*.


## Bitbucket

El trabajo de esta sesión también debes subirlo a Bitbucket. Todo el trabajo de esta práctica, tanto los proyectos maven que vamos a crear, como cualquier otro documento con vuestras notas de trabajo, deberán estar en la carpeta **P8** de vuestro repositorio.

Para esta práctica proporcionamos código adicional en la carpeta PlantillasP8

## Firebug

Hemos visto en clase que esta herramienta nos permite averiguar el valor de los parámetro *target* y *value* de nuestros comandos selenese. En la máquina virtual no está instalado. Para instalarlo hacemos lo siguiente:

- ❖ desde el administrador de tareas (icono :  de la barra de tareas) seleccionamos "Complemento", y buscamos "firebug" en el cuadro de texto correspondiente
- ❖ una vez que nos aparezca en la lista, lo instalamos pulsando la opción "instalar"

## Aplicación web utilizada para los ejercicios de esta sesión

Para esta práctica proporcionamos código adicional en la carpeta Plantillas-P8

Utilizaremos una aplicación Web Java denominada JPetStore (<http://mybatis.github.io/spring/sample.html>) sobre la que haremos las pruebas. Se trata de una versión simplificada de la demo de Sun denominada Java PetStore.

La aplicación a probar está en el directorio Plantillas-P8/mybatis-jpetstore-6.0.1-sources. Abre el proyecto desde Netbeans. El directorio de fuentes está estructurado en 4 paquetes:

- ❖ **domain**: contiene las clases que representan los objetos del dominio del negocio, con sus getters y setters,
- ❖ **persistence**: contiene las interfaces para "mapear" los objetos java con los datos persistentes,
- ❖ **service**: contiene las clases con la lógica de negocio de la aplicación,
- ❖ **web.actions**: contiene las acciones, es decir, la lógica de la capa de presentación de la aplicación.

La aplicación utiliza el patrón MVC (Modelo-Vista-Controlador) con tres capas Ver (<http://www.mybatis.org/jpetstore-6/>) para una descripción más detallada:

- ❖ **vista** (capa de presentación): formada por ficheros jsp y ActionBeans,
- ❖ **controlador** (capa de negocio): formada por objetos del dominio y servicios, y
- ❖ **modelo** (capa de datos): formada por interfaces de mapeo con datos persistentes

## Pasos previos: construcción y despliegue de la aplicación web

Las aplicaciones web se empaquetan en ficheros **.war**, y dicho artefacto tiene que ser desplegado en un servidor (un servidor web o un servidor de aplicaciones). Una vez que nuestra aplicación web esté en ejecución en el servidor, podemos acceder a ella a través del navegador, utilizando la url en la que ha sido desplegada nuestra aplicación.

Para obtener el **.war** de nuestra aplicación web, ejecuta la fase **package** desde Netbeans. Verás que se ha creado el artefacto **jpetstore.war** en el directorio target.

Las  
aplicaciones  
web se  
empaquetan  
como ".war"

El tipo de empaquetado del proyecto Maven se especifica en las coordenadas del proyecto. Ya hemos visto que por defecto, si no se especifica el tipo de empaquetado, éste será .jar. Hasta ahora no hemos necesitado incluir esta información en nuestro pom. Pero ahora necesitamos que nuestra aplicación se empaquete como .war puesto que va a "residir" en un contenedor web (nuestra aplicación será gestionada por el contenedor web del servidor, ya sea un servidor web, p.e. Tomcat, o un servidor de aplicaciones, p.e. Glassfish). Para indicar el tipo de empaquetado utilizamos la coordenada `<packaging>war</packaging>`

Las aplicaciones web necesitan desplegarse en un contenedor web. Utilizaremos el servidor de aplicaciones Glassfish, al que podremos acceder desde Netbeans, concretamente desde la pestaña "Services", anidado en el elemento "Servers".

Lo primero que haremos será arrancar el servidor, desde el menú contextual del servidor Glassfish, con la opción "Start". Cuando el servidor está en marcha, deberemos ver un pequeño triángulo verde al lado del nombre del servidor.

Para obtener el **.war** de nuestra aplicación web, ejecuta la fase **package** desde Netbeans. Verás que se ha creado el artefacto **jpetstore.war** en el directorio target.

Plugin glassfish  
y goal deploy

Hemos incluido en el pom.xml un nuevo plugin (*maven-glassfish-plugin*, ver documentación en <https://maven-glassfish-plugin.java.net>) para poder desplegar la aplicación web en Glassfish directamente desde el proceso de construcción de nuestro proyecto. Concretamente, usaremos la *goal* "deploy" que, por defecto, está asociada a la fase *pre-integration-test*. Esta *goal* despliega un artefacto *war* (también denominado componente web) en el dominio del servidor de aplicaciones que hayamos configurado en el plugin. Si el dicho dominio no está activo, la *goal* lo activará (activar el dominio implica arrancar el servidor).

Para desplegar la aplicación web en el servidor ejecutaremos *mvn verify*. Si el despliegue se ha podido completar con éxito, verás el artefacto desplegado anidado en el elemento "Applications" del servidor Glassfish.

### Despliegues posteriores

La goal *deploy* del plugin glassfish fallará si el artefacto ya está desplegado en el servidor. Por lo tanto, una vez desplegada la aplicación, si volvemos a ejecutar “mvn verify”, la construcción del proyecto se detendrá indicándonos que no ha podido desplegar el war. Para volver a desplegar el war generado, tendremos que ejecutar previamente la goal “undeploy” (con mvn glassfish:undeploy), y a continuación volver a ejecutar “mvn verify”

En ocasiones puedes ser necesario “refrescar” la vista del servidor desde el menú contextual con la opción “Refresh”. Por ejemplo, si ejecutamos “mvn verify” sin haber arrancado previamente el servidor, y accedemos a la ventana “Services”, es probable que no veamos el icono verde que indica que el servidor está en marcha, por lo que tendremos que refrescar la vista para asegurarnos de que efectivamente el servidor está activo.

Una vez desplegada la aplicación, podemos acceder a ella a través de la url:  
http://localhost:8080/jpetstore

## Ejercicios

- Ejercicio 1.** Antes de comenzar a realizar las pruebas, ejecuta la aplicación para familiarizarte con ella. La pantalla principal nos muestra la bienvenida a la tienda de animales. Una vez que entres en la tienda puedes mostrar una página de ayuda pulsando sobre el enlace “?” en la parte superior central de la página. Pinchando sobre el logo de la parte superior izquierda vuelves a la página principal de la tienda, en donde encontrarás 5 catálogos de diferentes animales, que podrás añadir al carrito de la compra para luego poder hacer efectiva la compra.
  - Vamos a preparar algunos tests de pruebas de aceptación de propiedades emergentes funcionales con Selenium IDE sobre la aplicación web. Para ello tendremos que tener abierta la aplicación desde el navegador Firefox, y pondremos en marcha la herramienta de Selenium desde el menú Herramientas→Selenium IDE. Por defecto, Selenium automáticamente comienza a “grabar” las acciones que hagamos sobre el navegador. Vamos a desactivar este comportamiento desmarcando la casilla “Start recording immediately on open”, desde el menú “Options→Options” de Selenium IDE. De esta forma, activaremos el proceso de “grabación” cuando nos interese.
  - Vamos a realizar una primera grabación de una secuencia de acciones para familiarizarnos con la herramienta. Abre Selenium y pulsa en el botón rojo para iniciar la grabación. Mueve la ventana de Selenium a la derecha del navegador de forma que tengas tanto la aplicación JPetStore, como Selenium IDE totalmente a la vista. Ejecuta las siguientes acciones: introduce la url de la aplicación web y entra en la tienda. Pulsa sobre la imagen del pez, pulsa sobre el ítem del catálogo con identificador FI-FW-02, a continuación pulsa sobre “Return to Fish”, seguidamente sobre “Return to Main Menu”, y detén la grabación. A medida que has ido navegando por la aplicación habrás podido ver que Selenium IDE ha ido creando comandos Selenese asociados a las acciones que has realizado. Deberás ver la siguiente secuencia de comandos Selenese:

Command	Target	Value
open	/jpetstore/	
clickAndWait	link=Enter the Store	
clickAndWait	css=area[alt="Fish"]	
clickAndWait	link=FI-FW-02	
clickAndWait	link=Return to FISH	
clickAndWait	link=Return to Main Menu	

Observa el código html asociado (pestaña **Source**), y verás que cada comando constituye una fila de una tabla con tres columnas, tal y como se muestra en la pestaña **Table** que aparece visible por defecto. La columna **Target** hace referencia a lo que se conoce como “**locator**”, y representa un elemento UI (User Interface) dentro de una página html. Así, por ejemplo, el primer comando **clickAndWait** tiene como parámetro el *locator* **link**, que define un enlace html.

Puedes ver el código html asociado a cada uno de los elementos de la página que estás visualizando actualmente en el navegador utilizando el plugin para FireFox **FireBug**. Una forma alternativa a la que hemos comentado en clase para utilizar dicho plugin es activándolo estando situados sobre cualquier elemento de la página (por ejemplo, el enlace inicial de la página para entrar a la tienda), y con botón derecho seleccionar “Inspeccionar elemento con Firebug”. Se abrirá una ventana en la parte inferior del navegador mostrando el código fuente de la página actual en la que estáis navegando y aparecerá resaltado el código fuente sobre el elemento sobre el que habéis situado el ratón. Navega por la aplicación repitiendo los pasos del test y observa cuál es el código fuente para cada uno de los *Targets* asociados a los comandos de nuestro caso de prueba.

Si seleccionamos cualquiera de los comandos Selenese generados veremos que automáticamente en la pestaña Reference, de la parte inferior de la ventana de Selenium IDE nos muestra la información sobre la sintaxis de dicho comando.

- C) Ahora vamos a grabar nuestro test. Si nos fijamos en el panel Test Case, veremos que únicamente hay un caso de prueba con nombre “Untitled \*” (el asterisco aparece porque todavía no hemos “grabado” en el disco nuestro caso de prueba. Para grabar el test seleccionamos “**Archivo→Save Test Case**” del menú de Selenium IDE, y vamos a ponerle el nombre “**test-inicial**”.
- D) Finalmente vamos a ejecutar nuestro test-inicial. Vamos a cambiar la “velocidad de ejecución” a “**slow**” para apreciar mejor la ejecución. Ejecutaremos el caso de prueba actual. Observa que los comandos se somborean en verde a medida que se van ejecutando. Fíjate que lo primero que hacemos es abrir en el navegador la dirección que aparece como base URL en la ventana de Selenium IDE, concatenada con el Target del comando *open*. Además, verás que en la pestaña “**Log**” se nos muestra una “traza” de los comandos selenese que se están ejecutando

2. **Ejercicio 2.** Utilizando la aplicación JpetStore, vamos a crear dos casos de prueba nuevos. Uno para hacer una compra como un usuario no registrado previamente. Y otro para hacer una compra como un usuario que ya se ha registrado previamente en el sistema.

**Nota:** Recuerda que Selenium IDE nos permite en cualquier momento (tanto si estamos grabando como si no), editar los comandos Selenese, es decir, que podremos añadir, quitar, mover y/o modificar cualquier comando cuando lo deseemos, ya que el panel central de Selenium IDE es un **EDITOR** de comandos Selenese. Por lo tanto, si durante la “grabación” de comandos cometemos algún error, podremos repararlo fácilmente modificando directamente el editor, sin tener que detener la grabación y/o empezar de nuevo. Recuerda que por precaución deberías ir grabando el test a menudo en el disco duro para no perder el trabajo realizado hasta el momento si se produjese algún fallo en el sistema.

A) Creamos un nuevo Caso de prueba (Archivo→New Test Case), al que llamaremos “**test-compra-new-user**” (una vez creado, le pondremos el nombre desde el menú contextual del nuevo test, con la opción *Properties*, e introduciendo el nombre en el campo de texto *Title*). En este test actuaremos como un usuario no registrado que quiere hacer una compra de dos perros y un gato. La dirección Base URL será: [localhost:8080/jpetstore/actions/Catalog.action](http://localhost:8080/jpetstore/actions/Catalog.action). Iniciamos la grabación (asegúrate antes de que en el navegador has entrado ya a la tienda).

1. Pulsamos sobre la imagen del perro
2. Nos aseguraremos de que estamos en la página correcta seleccionando el texto “Dogs” y con botón derecho nos situamos sobre “Show All Available Commands” y seleccionaremos “VerifyText css=h2 Dogs”. Vamos a comprar un Bulldog. Verificaremos (igual que hemos hecho antes) que el texto “Bulldog” aparece en la página, y pulsamos sobre el enlace K9-BD-01.
3. Verificamos que estamos en la página correcta seleccionando el texto “Bulldog” que aparece sobre la tabla. Queremos comprar un macho adulto. Verificamos que en la página aparece el texto “Male Adult Bulldog”. A continuación pinchamos sobre “Add to cart”. Verificamos que estamos en la página correcta, utilizando el texto “Shopping Cart”. Queremos comprar dos unidades, por lo que editaremos el campo de texto y pondremos un 2. Después de pulsar el retorno de carro, verificaremos que el precio total acumulado es de “Sub Total:\$37,00”.
4. Ahora vamos a comprar un gato. Para lo cual pinchamos sobre el enlace “Cats” en la parte superior de la página. Verificamos que estamos en la página correcta y que debe aparecer el texto “Cats”. Antes de seguir, vamos a comprobar que efectivamente en el carrito tenemos los 2 bulldogs, para lo cual pinchamos sobre el icono del carrito de la compra de la parte superior de la página. Verificamos que efectivamente aparece el texto “Male Adult Bulldog” y el sub total es el mismo de antes. Volvemos de nuevo al enlace “Cats”, y procedemos a comprar un ejemplar de persa. Para ello verificamos el texto “Persian”, y pinchamos sobre FL-DLH-02. Verifica que el texto “Adult Female Persian” está presente en la página y añade al carrito una hembra adulta. Ahora nuestro carrito debe contener los ítems “EST-6” y “EST-16”, y el nuevo sub total debe ser de 130,50 dólares. Antes de proceder a hacer efectiva la compra, pensamos que igual no es buena idea la compra de dos perros y un gato. Decidimos no comprar el gato, por lo que eliminamos este ítem. Ahora vamos a comprobar que el identificador EST-16 NO aparece. Esto no lo podemos hacer de forma automática, sino que tendremos que introducir el comando manualmente. Se trata del comando `verifyNotText`. Para ello, nos situamos en el campo de texto *Command* de Selenium IDE y comenzamos a teclear el comando, nos aparecerá un desplegable con una lista, de forma que seleccionaremos el comando `verifyNotText`. En el campo de texto *Value* escribiremos EST-16. Para escribir el valor del campo *target*, pulsamos el botón “Select”, seleccionamos la tabla en el navegador, de forma que el valor de *target* aparezca en el campo de texto correspondiente. Seguidamente verificaremos que en la página el sub total vuelve a ser de 37 dólares. Hasta ahora nuestro script de pruebas debe tener el siguiente “aspecto”:

compra-new-user		
open	/jpetstore/actions/Catalog.action	
clickAndWait	css=area[alt="Dogs"]	
verifyText	css=h2	Dogs

verifyText	//div[@id='Catalog']/table/tbody/tr[2]/td[2]	Bulldog
clickAndWait	link=K9-BD-01	
verifyText	css=h2	Bulldog
verifyText	//div[@id='Catalog']/table/tbody/tr[2]/td[3]	Male Adult Bulldog
clickAndWait	link=Add to Cart	
verifyText	css=h2	Shopping Cart
type	name=EST-6	2
clickAndWait	name=updateCartQuantities	
verifyText	//div[@id='Cart']/form/table/tbody/tr[3]/td	Sub Total: \$37,00
clickAndWait	//div[@id='QuickLinks']/a[4]/img	
verifyText	css=h2	Cats
clickAndWait	name=img_cart	
verifyText	//div[@id='Cart']/form/table/tbody/tr[2]/td[3]	Male Adult Bulldog
verifyText	//div[@id='Cart']/form/table/tbody/tr[3]/td	Sub Total: \$37,00
clickAndWait	//div[@id='QuickLinks']/a[4]/img	
verifyText	//div[@id='Catalog']/table/tbody/tr[2]/td[2]	Persian
clickAndWait	link=FL-DLH-02	
verifyText	//div[@id='Catalog']/table/tbody/tr[2]/td[3]	Adult Female
clickAndWait	link=Add to Cart	
verifyText	link=EST-6	EST-6
verifyText	link=EST-16	EST-16
verifyText	//div[@id='Cart']/form/table/tbody/tr[4]/td	Sub Total: \$130,50
clickAndWait	xpath=(//a[contains(text(),'Remove')])[2]	
verifyNotText	css=table	EST-16
verifyText	//div[@id='Cart']/form/table/tbody/tr[3]/td	Sub Total: \$37,00

5. Ahora procederemos a realizar la compra, pinchando sobre “Proceed to Checkout”. Verificamos que en la nueva página nos aparece el mensaje: “You must sign on before attempting...”. Como somos un usuario nuevo, tendremos que registrarnos primero para poder hacer la compra, por lo que pinchamos sobre “Register now”. Nos aparecerá una página con el texto “User Information”. Rellenamos los campos. Por ejemplo podemos poner como User ID y password el carácter “z”. Rellenaremos todos los campos de “Account Information” con la letra “z” (por ejemplo. Si quieres, puedes poner cualquier cadena de caracteres). Recuerda rellenar TODOS los campos. Dejaremos el apartado “Profile Information” como aparece por defecto. Finalmente pincharemos sobre “Save Account Information”. Volvamos a nuestro carrito de la compra pulsando sobre el icono del carrito de la parte superior de la pantalla. Verificaremos que aparece el texto “Shopping Cart” y que el subtotal sigue siendo de \$37,00. Seguimos grabando hasta que completemos la compra, ahora ya podemos entrar en el sistema como un usuario registrado. Antes de confirmar el pedido, verificaremos que en la página aparece la cantidad total correcta (37 dólares). Hacemos un “Sign out” y terminamos la grabación. A continuación mostramos la secuencia de comandos que hemos añadido:

compra-new-user (continuación)
--------------------------------



clickAndWait	link=Proceed to Checkout	
verifyText	css=li	You must sign on before
clickAndWait	link=Register Now!	
type	name=username	z
type	name=password	z
type	name=repeatedPassword	z
type	name=account.firstName	z
type	name=account.lastName	z
type	name=account.email	z
type	name=account.phone	z
type	name=account.address1	z
type	name=account.address2	z
type	name=account.city	z
type	name=account.state	z
type	name=account.zip	z
type	name=account.country	z
clickAndWait	name=newAccount	
clickAndWait	name=img_cart	
verifyText	css=h2	Shopping Cart
verifyText	//div[@id='Cart']/form/table/tbody/	Sub Total: \$37,00
clickAndWait	link=Proceed to Checkout	
type	name=password	z
clickAndWait	name=signon	
clickAndWait	name=img_cart	
clickAndWait	link=Proceed to Checkout	
clickAndWait	name=newOrder	
clickAndWait	link=Confirm	
verifyText	//div[@id='Catalog']/table/tbody/	exact:Total: \$37,00
clickAndWait	link=Sign Out	

**Nota:** Al introducir el nombre de usuario, selenium habrá grabado como valor de target "stripes-946292712", que es el valor del atributo "id" del campo de texto en el que introducimos el valor de "Username" (puedes comprobarlo inspeccionando el elemento del campo de texto situado a la derecha de "Username"). Por defecto, Selenium siempre utiliza el atributo "id" del elemento html, o el atributo "name" si el atributo "id" no está presente. En este caso, nos interesa el valor del atributo "name", en lugar del atributo "id", por lo tanto tendremos que cambiar manualmente el valor del campo target por: "name=username".

6. Antes de ejecutar el test tendremos que "borrar" el usuario que hemos creado durante la creación del script de pruebas. Puesto que la aplicación utiliza una base de datos en memoria (HSQLDB). Una forma de "limpiar" la base de datos es reiniciar la aplicación. Para ello bastaría con parar y volver a arrancar el servidor de aplicaciones Glassfish desde la ventana *Services* de Netbeans:

Una vez que volvamos a arrancar el servidor, ya podemos ejecutar nuestro test desde Selenium IDE. Recuerda poner la velocidad más baja de ejecución para poder ir viendo claramente los pasos (también puedes pausar la ejecución en cualquier momento).

Si intentas repetir el test sin reiniciar el servidor puedes comprobar que el test fallará puesto que estaremos intentando darnos de alta dos veces en el sistema.

- B) Ahora vamos a crear otro caso de prueba con el nombre “**compra-old-user**”. En este caso se trata de un usuario previamente registrado, por ejemplo, el usuario z que acabamos de crear, que quiere comprar 500 ejemplares de iguana verde adulta (es decir, el identificador del producto es: RP-LI-02, y el identificador del ítem es: EST-13). Se trata de comprobar que, después de hacer la compra, deben quedar 500 ejemplares menos en stock. Para ello tendrás que incluir en el script de pruebas la navegación por la página en la que se muestra la información del stock inicial de un ejemplar, recordar esta cantidad para luego restarle 500 unidades, y volver a navegar por dicha página para comprobar que efectivamente el nuevo valor ha disminuido en 500 unidades. Para poder programar esto, vamos a tener que utilizar **variables** para almacenar los valores que necesitemos y operar con ellos, ya que vamos a trabajar con contenidos de la página html que son dinámicos (no sabemos a priori cuál será el stock, y dependiendo de la cantidad que queramos comprar, este stock tendrá un valor que tampoco conocemos a priori). A continuación indicaremos que comandos podéis utilizar para trabajar con la información dinámica que genera la página.
- C) Vamos a explicar cómo utilizar el comando Selenese **storeText**. El comando Selenese storeText, almacena un texto en una variable. Para referenciar el valor de una variable, pongamos por ejemplo var1 se utiliza el nombre de la variable entre llaves y precedida de un “\$”, es decir \${var1}

Los siguientes comandos almacenan en la variable textStock, el valor “6500 in stock”, y a continuación muestran dicho valor en la pantalla de logs de Selenium IDE.

storeText	//div[@id='Catalog']/table/tbody/tr[5]/td	textStock
echo	\${textStock}	

Para introducir este comando en el editor de Selenium IDE, tendremos que acceder a la página en donde se muestra el stock del ejemplar que nos interesa (puedes ver la imagen de dicha página en la siguiente figura), seleccionar el texto (“6500 in stock”), y con botón derecho seleccionamos “Show All Available Commands”, y buscamos el comando “storeText” en el sub-menú de comandos disponibles. Cuando nos pregunte el nombre de la variable, pondremos “textStock”. El valor del campo “Target” nos muestra una ruta XPath “relativa”. Esta ruta nos está diciendo que el elemento que queremos localizar está en la primera etiqueta <div> con el identificador ‘Catalog’, en la subetiqueta <table>, <tbody>, en la quinta fila, y finalmente en el elemento <td>.





De forma alternativa, podríamos introducir “manualmente” el comando. Como valor del campo “Target” tendríamos que copiar la ruta **XPath** del elemento. Para obtener dicha ruta, tenemos que seleccionar el texto que nos interese (por ejemplo “6500 in stock”, tal y como aparece en la imagen anterior), y con botón derecho inspeccionamos el elemento con Firebug. Una vez que nos aparece la ventana de Firebug en la parte inferior del navegador, tenemos que situarnos en el código html resaltado en azul y con botón derecho seleccionamos “Copiar XPath”. Ahora ya podemos “pegar” este valor en el campo “Target”. Para el ejemplo anterior, el valor de XPath es el siguiente: “/html/body/div[2]/div[2]/table/tbody/tr[5]/td”. Se trata de un XPath “absoluto”, fíjate que localiza el elemento de texto siguiendo la ruta de las etiquetas html de la página **DESDE** la etiqueta **RAÍZ** del documento html. Actualmente, el comando **storeText** requiere un XPath “relativo”, por lo que si utilizas esta opción debes añadir “/” al principio de la ruta, por lo que el campo Target quedaría así “//html/body/div[2]/div[2]/table/tbody/tr[5]/td”.

Ahora bien, necesitaríamos almacenar en una variable solamente la cantidad (6500), en lugar de todo el texto, para ello puedes utilizar el siguiente comando Selenese:

storeEval	storedVars['textStock'].match(/d+\.\?d*/g)	stock
echo	\${stock}	

El comando **storeEval** evalúa una sentencia JavaScript y la almacena en una variable. En este caso utilizamos la sentencia JavaScript **match** sobre la variable **textStock** (todas las variables creadas se almacenan en un array en el que los índices son los nombres de las variables). La función **match** recupera todos los caracteres que cumplan una determinado patrón (dado por una expresión regular), en este caso la expresión regular utilizada extraerá cualquier número de la cadena de caracteres **\$(textStock)**, y lo almacena en una variable a la que hemos denominado **stock**. El segundo parámetro simplemente muestra el valor de la variable en la que hemos almacenado el resultado.

Ahora ya tenemos en una variable el valor inicial del stock, necesitamos almacenar en otra variable el resultado de restar al stock inicial las 500 unidades que queremos comprar. Para ello podemos utilizar el comando **storeEval** que ya conocemos, de la siguiente forma:

store	500	amount
echo	\${amount}	
storeEval	storedVars['stock']-storedVars['amount']	expectedResult
echo	\${expectedResult}	

El comando **store** almacena un valor en una variable, en nuestro caso asignamos a la variable *amount* el valor de 500. El comando **storeEval**, calcula el resultado de restar los contenidos de las variables *stock* y *amount*, y almacena el resultado en la variable *expectedResult*. Una forma alternativa de realizar esta resta sería introducir en el campo Target la expresión “\${stock}-\${amount}”.

Finalmente, para comprobar que el resultado mostrado en la página una vez que hemos realizado la compra es el correcto, podemos utilizar el comando:

verifyTextPresent	\${expectedResult}	
-------------------	--------------------	--

Es importante que “grabas” la salida (acción **Sign Out**) de la cuenta de usuario cuando termines el test. Prueba a ejecutar varias veces la compra de 500 unidades. Deberás comprobar que el stock va decreciendo correctamente. ¿Qué pasa si no grabas en el test la salida de la cuenta del usuario?