Guía de desarrollo de test sobre navegador con Selenium

Referencia	Guia Selenium
Creación	1 de Noviembre de 2017
Autor(es)	Rafael Jiménez Ballester
Versión	1.2

ÍNDICE

1. Objetivo

Este documento tiene como objetivo ser una pequeña guía de instalación y configuración de Selenium. Selenium es una plataforma para automatizar navegadores web. Utilizaremos esta plataforma para realizar pruebas sobre nuestros portales y aplicativos web. Además en este documento se desarrolla un ejemplo de como realizar una prueba sobre un portal, y dejarla preparada para integrarla con un sistema de monitorización.

2. Instalación

2.1. Instalacion Selenium IDE (Plugin firefox)

Selenium IDE es un plugin que permite grabar macros sobre el navegador Firefox. Utilizaremos este plugin como base para generar nuestras pruebas. Para instalarlo hay que seguir los siguientes pasos:

- Entrar en la pagina:

  https://addons.mozilla.org/en-US/firefox/addon/selenium-ide/
- Pulsar en el boton:

  + Add to Firefox
- Darle a instalar plugin y reiniciar firefox.

Deberia salir un icono de selenium IDE en la barra de firefox y sino acceder por: Menu->Desarrollador->Selenium IDE

2.2. Instalacion Selenium Standalone Server

Selenium IDE solo sirve para poder grabar macros y reproducirlas sobre un navegador web en un equipo de usuario. En entornos corporativos interesa lanzar esas macros sobre servidores. Existe la librería Seleniun Web Driver, que mediante lenguaje de programación permite lanzar pruebas sobre un servidor Selenium. Este servidor es Selenium Standalone Server. El servidor Selenium Standalone tiene a su vez conectadas una o varias instancias de un navegador donde se lanzaran las pruebas. Las instancias del navegador pueden estar en el mismo servidor donde esta el Selenium Standalone Server o en otros. A continuación se detalla como instalar Selenium Standalone Server.

- Instalar jre de https://www.java.com/es/download/
- Crear el directorio c:\Archivos de programa\selenium\server

- Descargar el jar de la ultima version de http://www.seleniumhq.org/download/ y dejarlo en ese directorio.
- Crear el directorio c:\Archivos de programa\selenium\bin y crear dentro el fichero selenium-hub.bat y dentro poner:

```
SET SELENIUM_HOME="c:\Archivos de programa\selenium"
SET JAVA_HOME="c:\Archivos de programa\jre1.8.0_112"
%JAVA_HOME%\bin\java.exe -jar %SELENIUM_HOME%\server\selenium- server- standalone-3.0.1.jar  -role
hub
```

2.3. Cliente Internet Explorer

Para que las pruebas de Selenium se puedan realizar sobre Internet Explorer, hay que configurar una instancia cliente para usar dicho navegador, las instrucciones para configurarlo son las siguientes:

- Descargar el driver de Internet Explorer:
  - 32bits:

    http://selenium-release.storage.googleapis.com/3.0/IEDriverServer_Win32_3.0.0.zip
  - 64bits:

    http://selenium-release.storage.googleapis.com/3.0/IEDriverServer_x64_3.0.0.zip

- Crear el directorio c:\Archivos de programa\selenium\iexplorer y descomprimir dentro el fichero descargado
- Crear el fichero c:\Archivos de programa\selenium\bin\selenium-iexplorer.bat y dentro poner:

```
  SET SELENIUM_HOME="c:\Archivos de programa\selenium"
  SET JAVA_HOME="c:\Archivos de programa\jre1.8.0_112"
  %JAVA_HOME%\bin\java -Dwebdriver.ie.driver=%SELENIUM_HOME  %\iexplorer\IEDriverServer.exe -jar
%SELENIUM_HOME%\server\selenium-server-standalone-3.0.1.jar -port 5555 -role node -hub
http://localhost:4444/grid/register -browser "browserName=internet
explorer,version=11,platform=WINDOWS,maxInstances=10"
```

Además para que funcione con Internet Explorer 11 hay que hacer lo siguiente:

- Pulsar Alt para sacar el menu del IE11.
- Seleccionar "Herramientas-> Opciones de Internet" e ir a la pestaña de "Seguridad".
- Seleccionar cada zona (Internet, intranet Local , Sitios de confianza , Sitios Restringidos) y marcar "Habilitar Modo protegido".

2.4. Cliente chrome

Para instalar el cliente de Chrome hay que hacer lo siguiente:

- Descargar el driver chrome de https://sites.google.com/a/chromium.org/chromedriver/downloads
- Crear el directorio c:\Archivos de programa\selenium\chrome y descomprimir dentro chromedriver.exe
- Crear el fichero c:\Archivos de programa\selenium\bin\selenium-chrome.bat y dentro poner:

```
SET SELENIUM_HOME="c:\Archivos de programa\selenium"
SET JAVA_HOME="c:\Archivos de programa\jre1.8.0_112"
%JAVA_HOME%\bin\java -Dwebdriver.chrome.driver=%SELENIUM_HOME%\chrome\chromedriver.exe -jar
%SELENIUM_HOME%\server\selenium-server-standalone-3.0.1.jar -port 5556  -role node -hub
http://localhost:4444/grid/register -browser "browserName=chrome,platform=WINDOWS,maxInstances=10"
```

2.5. Cliente firefox

Las instrucciones para instalar el cliente de Firefox son las siguientes:

-El navegador tiene que estar actualizado a la ultima version sino no funcionarán los test.

-Descargar el driver firefox de https://github.com/mozilla/geckodriver/releases

-Crear el directorio c:\Archivos de programa\selenium\firefox y descomprimir dentro geckodriver.exe

-Crear el fichero c:\Archivos de programa\selenium\bin\selenium-firefox.bat y dentro poner:

```
SET SELENIUM_HOME="c:\Archivos de programa\selenium"
SET JAVA_HOME="c:\Archivos de programa\jre1.8.0_112"
%JAVA_HOME%\bin\java -Dwebdriver.gecko.driver=%SELENIUM_HOME%\firefox\geckodriver.exe -jar
%SELENIUM_HOME%\server\selenium-server-standalone-3.0.1.jar -port 5557  -role node -hub
http://localhost:4444/grid/register -browser "browserName=firefox,maxInstances=10"
```

Firefox nos permite tener varios perfiles, y desde Selenium se puede seleccionar uno u otro a la hora de realizar las pruebas. De esta forma podemos tener dos perfiles, uno normal, y en otro podemos instalar un certificado y usarlo en las pruebas que requieran de él. Para configurar los distintos perfiles de Firefox hay que lanzar el navegador de la siguiente forma:

```
firefox -P
```

Una vez creado el perfil, para configurar el certificado hay que hacer lo siguiente:

- Preferencias->Avanzado->Certificados
- Marcar->Seleccionar uno automáticamente

- Ver certificado->Añadir certificado

2.6. Estado instancias selenium

Una vez instalado Selenium Web Driver, y configuradas las distintas instancias de navegador para realizar los tests, se puede acceder a una pagina de gestión del Selenium en la URL http://localhost:4444/grid/console y ver el estado de las distintas instancias

3. Librerias Java Selenium

Para poder desarrollar pruebas de Selenium en java, hay que bajarse las librerías de selenium para java de:

http://www.seleniumhq.org/download/

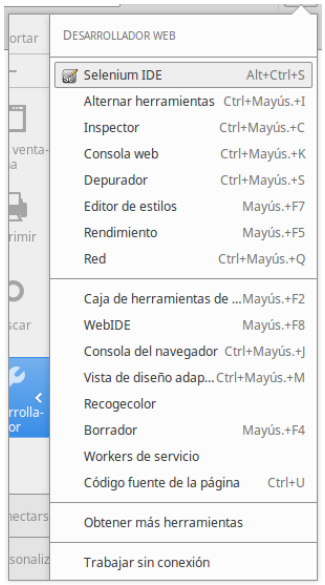
del apartado:

"Selenium Client & WebDriver Language Bindings".

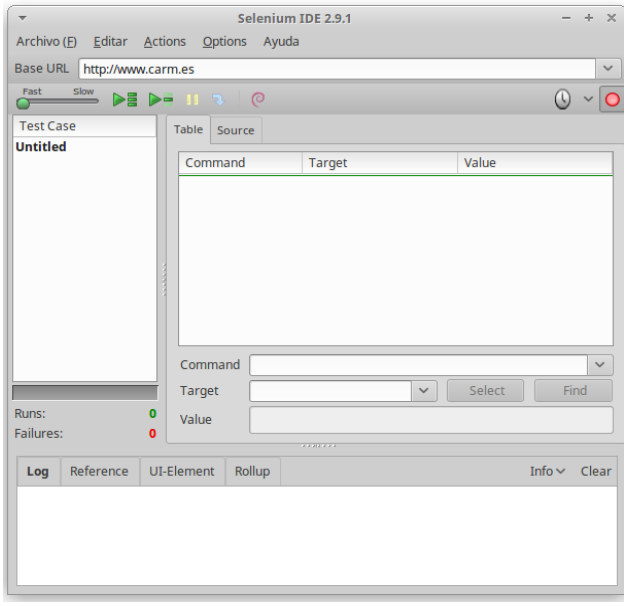
Una vez descargadas, habrá que añadirlas al proyecto Java de nuestra prueba Selenium.

4. Ejemplo Selenium IDE

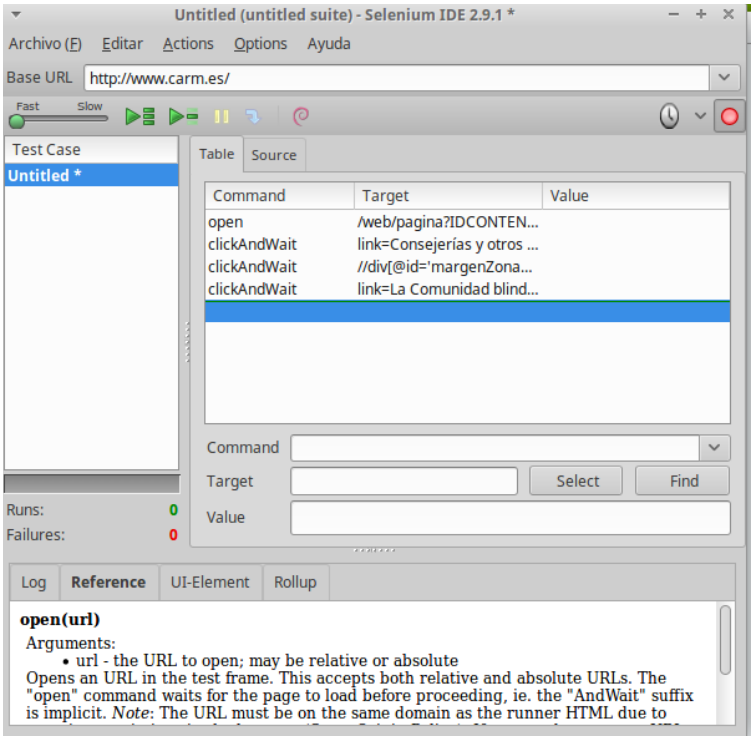
Una vez instalado Selenium IDE según el punto 2.1 de esta guía, podemos abrir el plugin accediendo al siguiente menú de Firefox (Menu→Desarrolador Web→Selenium IDE):



Se abre la siguiente interfaz:



BaseURL es la URL base de la que queremos grabar la Macro. El botón rojo de la derecha es el que indica si esta grabando la macro. Si el botón esta pulsado todas las acciones que hagamos en el navegador firefox desde el que abrimos el Selenium IDE se irán grabando.



Una vez grabada la macro hay que pulsar al botón rojo para que deje de grabar. Pulsando sobre el botón verde del play reproducirá la macro que hemos grabado. Si ponemos la velocidad de reproducción muy rápida es posible que la prueba falle, debido a que la prueba se ejecuta mas rápido de lo que el navegador tarda en cargar las paginas, es mejor poner una velocidad lenta. Con esta herramienta podemos grabar macros simples, o empezar a hacer el esqueleto de una prueba mas compleja. En los siguientes puntos de la documentación veremos como pasar de esta prueba en Selenium IDE a una prueba valida para monitorizar aplicaciones.

5. Estructura de un resultado de monitorización

Esta guía, a parte de hacer una pequeña introducción al uso de Selenium, también tiene por objetivo ayudar a integrar las pruebas generadas con Selenium en los sistemas de monitorización de la CARM. Para esto, las pruebas que se hagan con Selenium deberán proporcionar unas salida formateada de forma que el sistema de monitorización las entienda. La salida de las pruebas deberá seguir el siguiente esquema de salida:

```
(OK | WARNING | CRITICAL ) Texto descriptivo del resultado
Texto del Test
| variable=”tiempo”ms variable2=”tiempo2”ms …
```

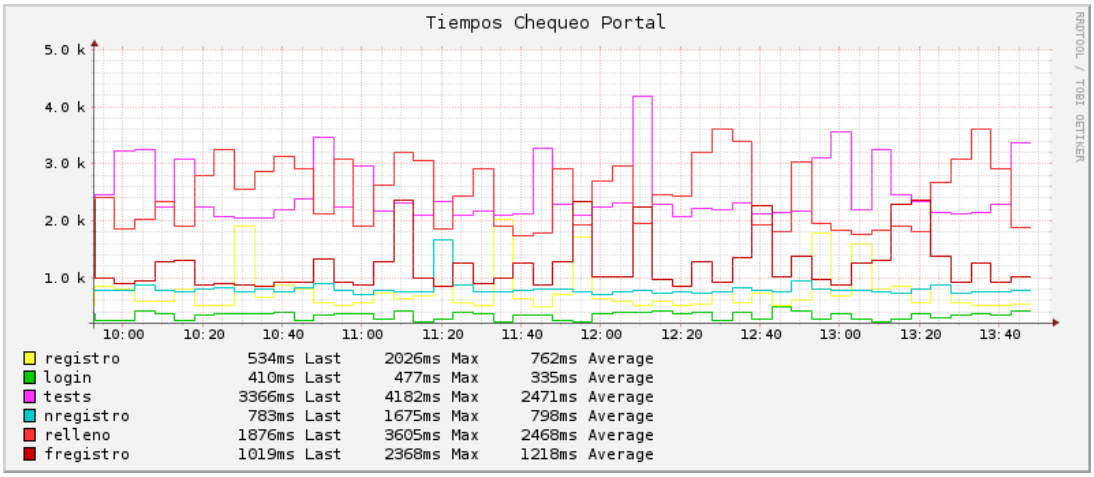
Primero deberá mostrar en texto el resultado del test, si ha sido correcto “OK” si hay algún problema en el test pero no es un problema critico deberá poner “WARNING” y en caso de fallo del test o de un problema critico en el mismo deberá poner “CRITICAL”. Además deberá de acompañarse de un texto descriptivo del resultado del test. Todo esto debe ser en una primera linea. Luego viene un texto libre de una o

varias lineas. Este texto se suele usar para describir las distintas partes que componen el test, y mostrar por ejemplo el tiempo que ha tardado cada una de estas partes. Por último existe la posibilidad de que el sistema de monitorización cree gráficas de la evolución en el tiempo de las variables que nosotros creamos convenientes de la prueba. Estas variables se ponen detrás de una tubería "|" con el formato "nombre_variable"="valor"+"unidad". La distintas variables se separan con espacios. Además la ejecución del test deberá devolver como estado 0 si el test es OK, 1 si es WARNING y 2 si es CRITICAL.

Un ejemplo de salida de un test podría ser el siguiente:

```
OK - Test Alsigem Correcto
1 Carga Pagina Registro -> 511ms
2 Login -> 345ms
3 Tests -> 2303ms
4 Nuevo Registro -> 753ms
5 Relleno Formulario -> 2835ms
6 Registro Nº 201797700063729 [Captura] -> 882ms
| registro=511ms login=345ms tests=2303ms nregistro=753ms relleno=2835ms fregistro=882ms
```

El sistema de monitorización crea la siguiente gráfica de la evolución de las variables que hemos indicado en la salida del test:



6. Libreria SeleniumCarm

Para facilitar el desarrollo de test en Java para la monitorización, se ha desarrollado una librería que facilita la conexión a los servidores Selenium, y la gestión de la salida de los test para hacerla compatible con la monitorización. La librería se llama SeleniumCarm.jar. La documentación Javadoc de esta clase, se adjunta al final de este documento como Anexo1. En los siguientes ejemplos veremos como utilizar la librería.

7. Pasos para realizar Test

Cuando decidimos realizar una prueba automática sobre navegador de un portal para su uso en los sistemas de monitorización podemos seguir los siguientes pasos:

- Realizar test o esqueleto del test con SeleniumIDE
- Exportar a Java dicho test
- Crear Proyecto Java y adjuntar librerias SeleniumCarm y Selenium para java.
- Probar el test en nuestro servidor local Selenium
- Distribuir el test en los servidores de producción

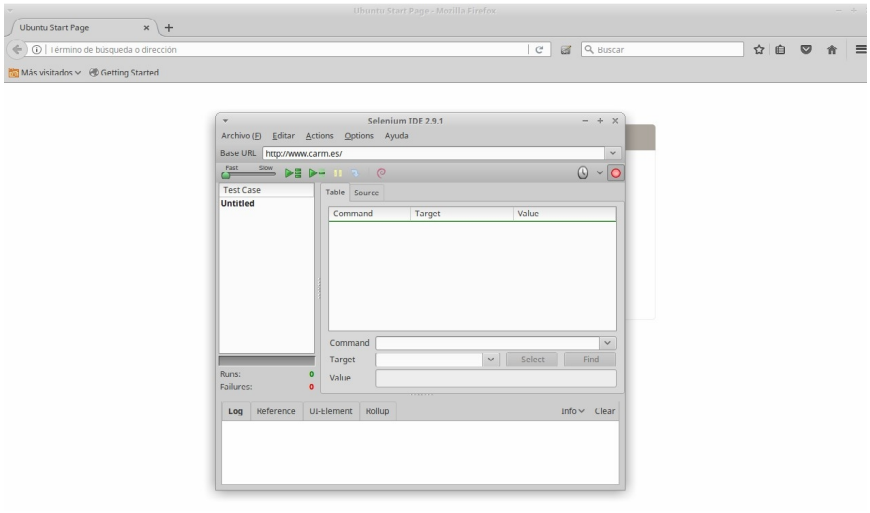
Para normalizar los test en java sobre Selenium, se propone que cada Proyecto Java de test tenga dos clases, una estática con un método main que lanzara el test, y otra clase que será el test en si mismo. La clase estática deberá coger cuatro parámetros obligatorios, la Url del servidor Selenium sobre el que realizar las pruebas, el directorio en dicho servidor donde se almacenarán las capturas de pantalla y las excepciones, una Url para el acceso por web a esos ficheros y el tipo de navegador sobre el que se realizara el test. Opcionalmente si se utiliza el firefox como navegador y se requiere usar un perfil distinto al por defecto, se debe pasar el perfil a usar.

```
-seleniumServer=url       (obligatorio) -> Url del servidor Selenium donde se lanzara la prueba
-filesUrl=url             (obligatorio) -> Directorio en el servidor Selenium donde se almacenan las
imagenes y las excepciones
-filesDir=directorio      (obligatorio) -> Url para acceder a las imagenes y excepciones generadas
-browser=Navegador        (obligatorio) -> Navegador para realizar la prueba
(firefox,chrome,iexplorer,phantomjs) por defecto firefox
-profile=perfil_Navegador (opcional)    -> En caso del que navegador soporte perfiles, cual queremos
usar\n"
```

Esta Clase estatica una vez ejecutada, deberá devolver el estado 0 si el test ha sido correcto, 1 si es un WARNING y 2 si es un CRITICAL. Además debe devolver texto acorde al punto 5 de este documento.

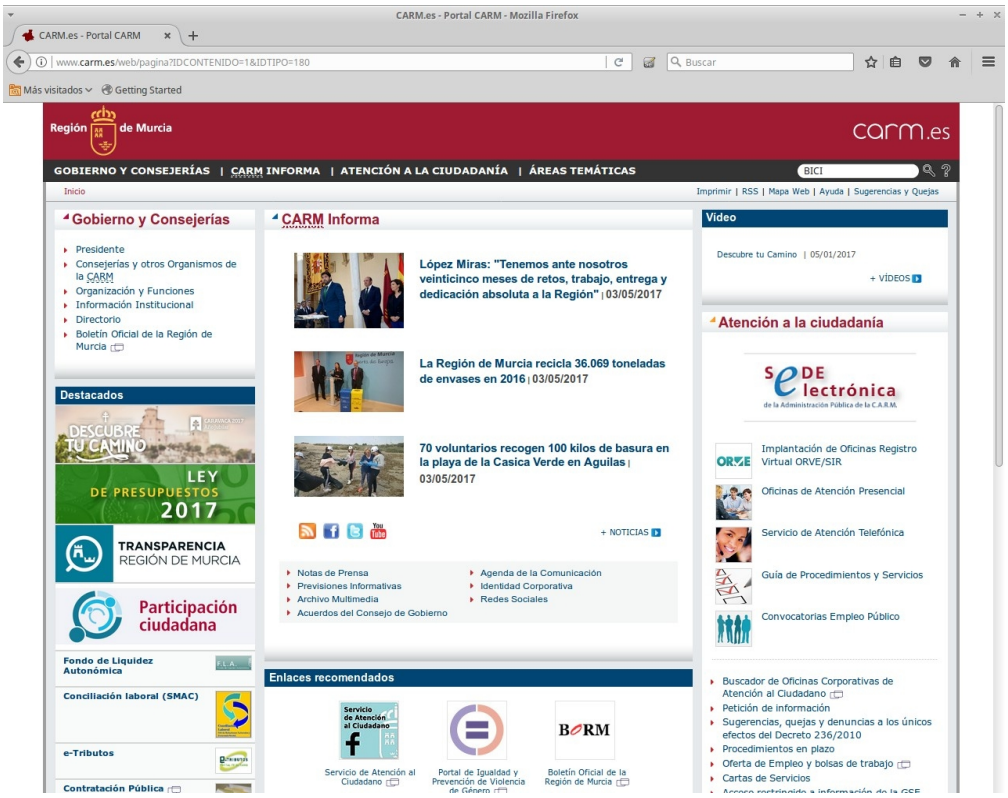
8. Ejemplo Basico Selenium IDE

Vamos a hacer un ejemplo completo de como monitorizar un portal con Selenium. Lo primero que tenemos que hacer es decidir en que consistirá la prueba. En este caso vamos a entrar en BICI a través del portal de la CARM y vamos a buscar el temario especifico de la oposición de Analista de Aplicaciones. Hay que abrir el Firefox y activar el plugin Selenium IDE.

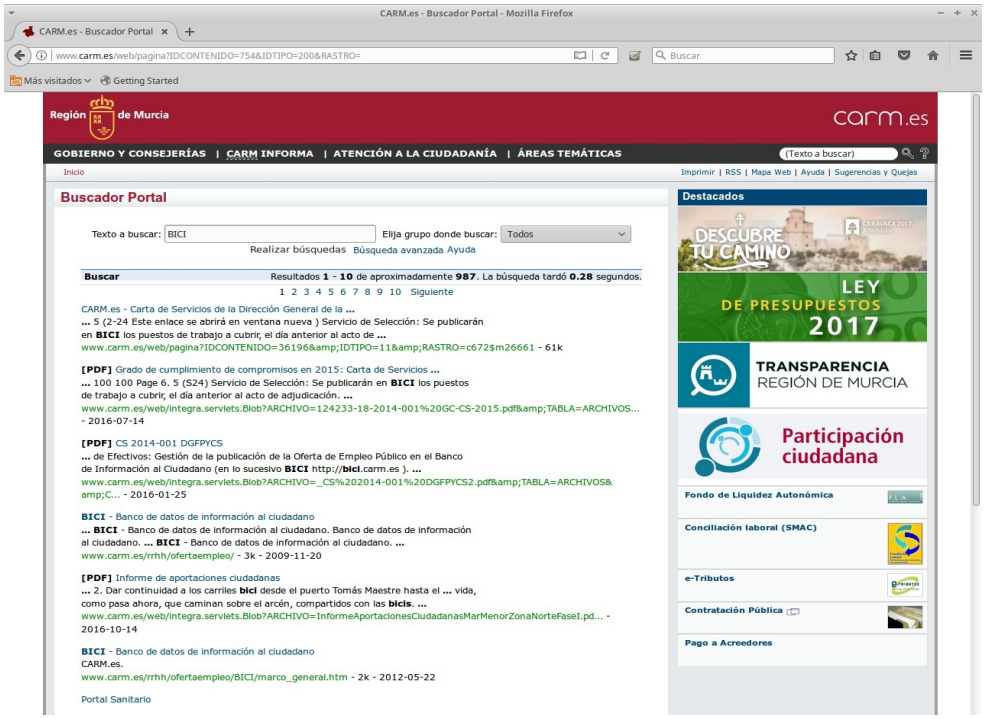


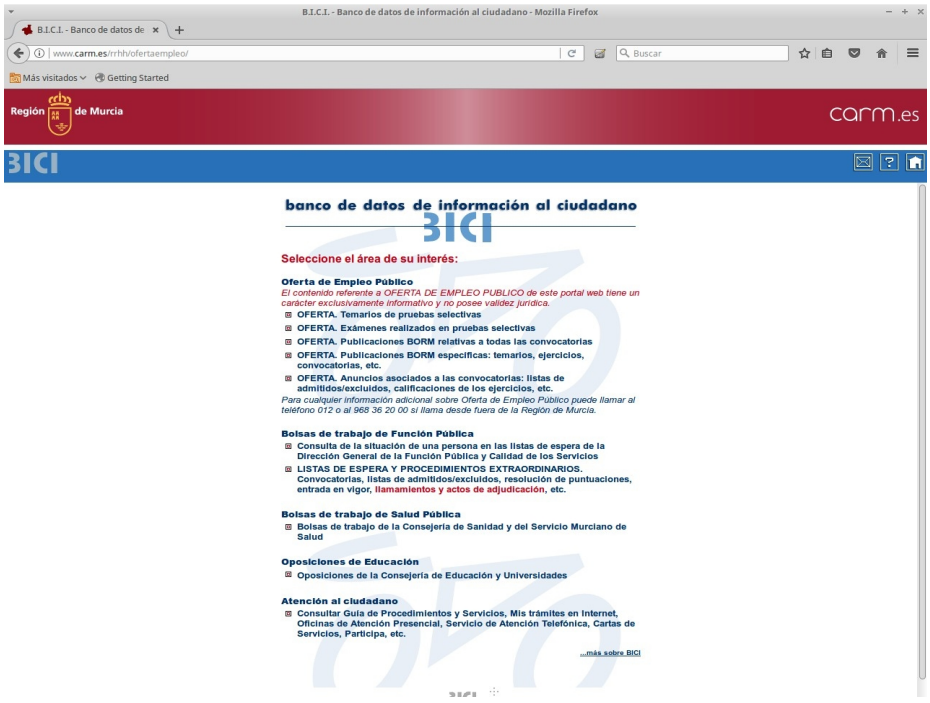
Introducimos el “Base URL” la Url por donde empezara nuestro test, en este caso “www.carm.es”, y damos al botón rojo de grabar.

Nos pasamos el navegador, introducimos la url y pulsamos intro. Se abre el portal de la Carm. Vamos a la parte derecha arriba, y en el buscador introducimos la palabra BICI y pinchamos sobre el botón de buscar con el ratón.



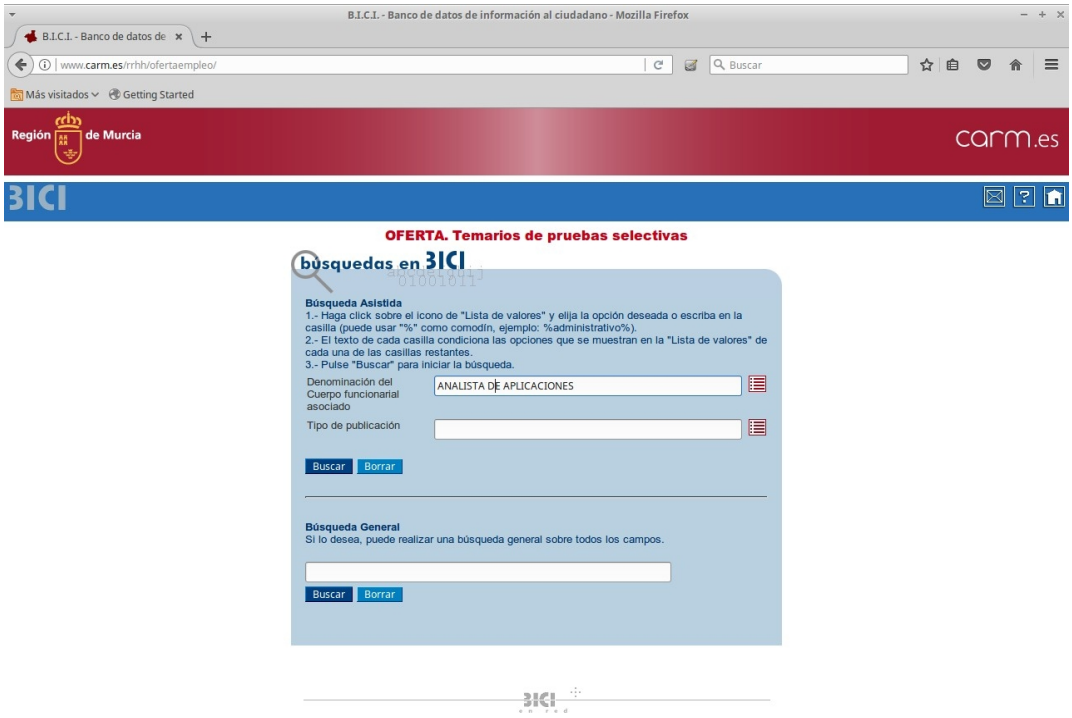
Obtenemos varios resultados en la búsqueda, pinchamos sobre el enlace “BICI- Banco de datos de información al ciudadano”



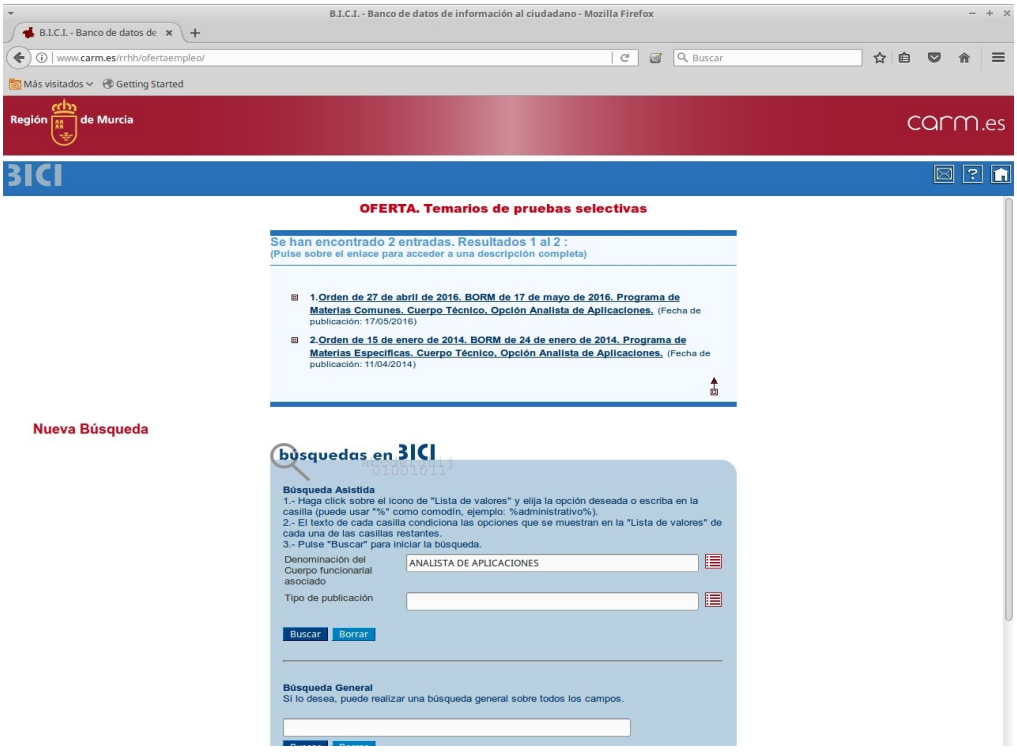


Luego pinchamos sobre “OFERTA. Temario de pruebas selectivas”, y en el campo “Denominación del Cuerpo funcionarial asociado” ponemos “ANALISTA DE APLICACIONES”

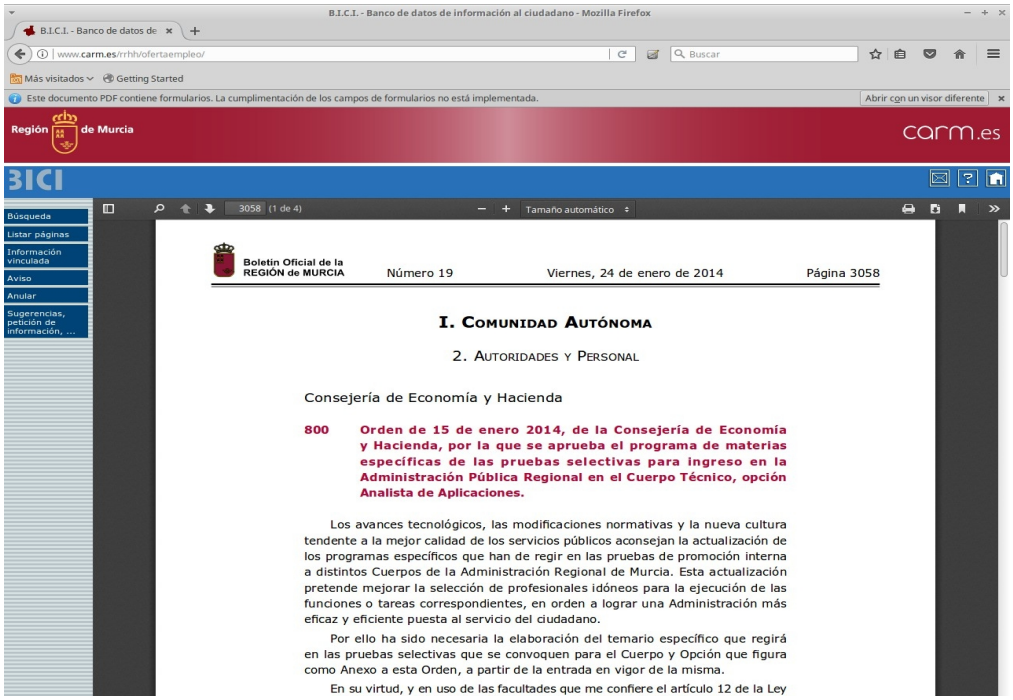
A



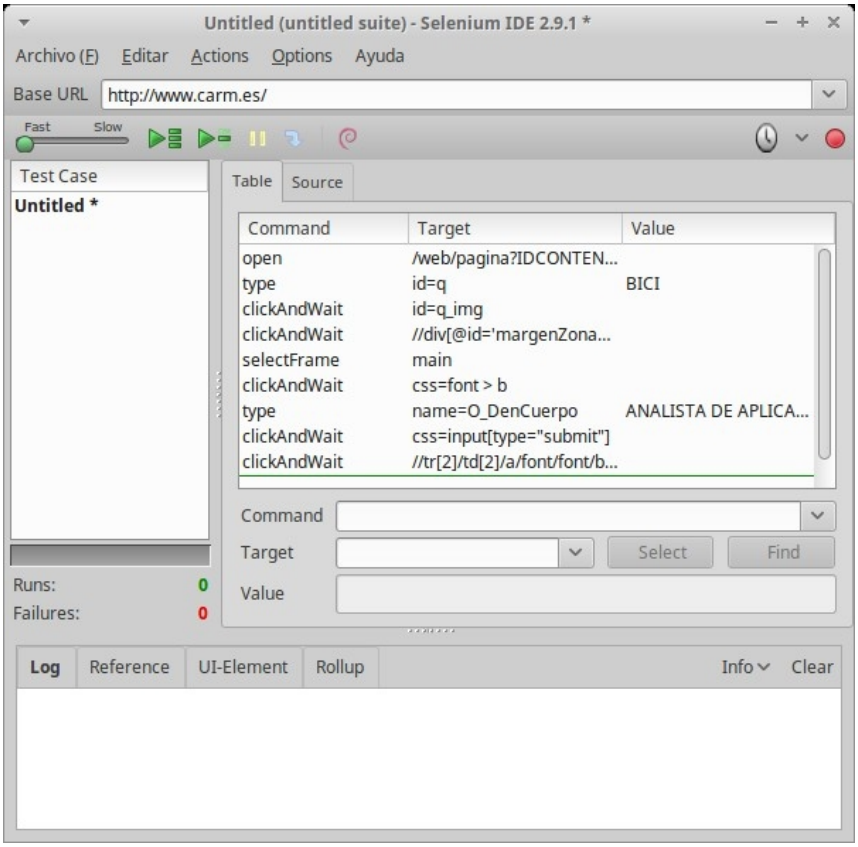
continuación pulsamos sobre el botón de “Buscar” con el ratón.



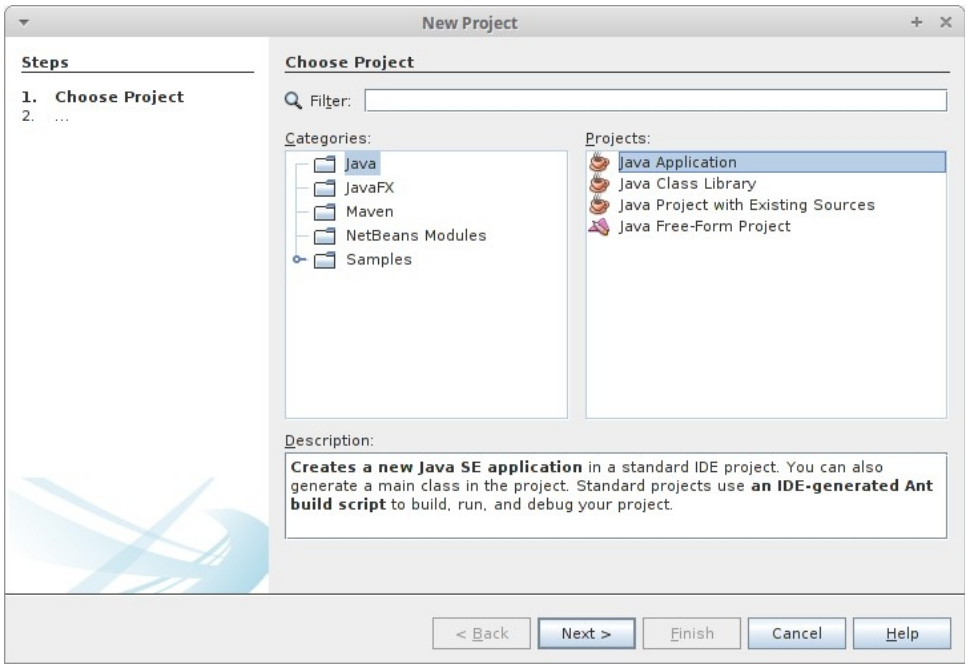
Por último pinchamos sobre el segundo enlace: “Programa de Materias Especificas”



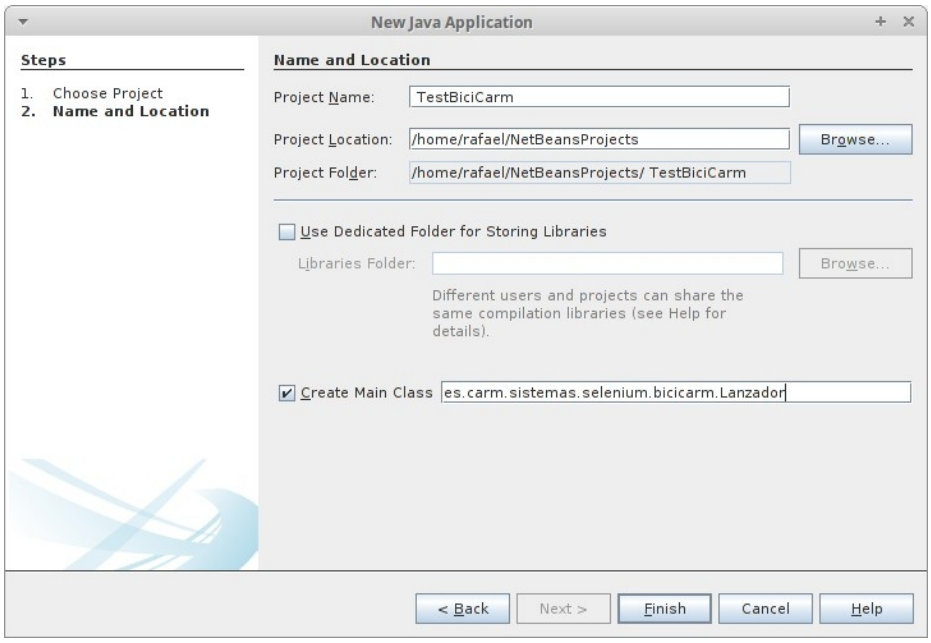
Hasta aquí ya tenemos grabado nuestro test en Selenium IDE. Debemos ir al interfaz de Selenium IDE y pulsar sobre el botón de grabar para que deje de hacerlo.



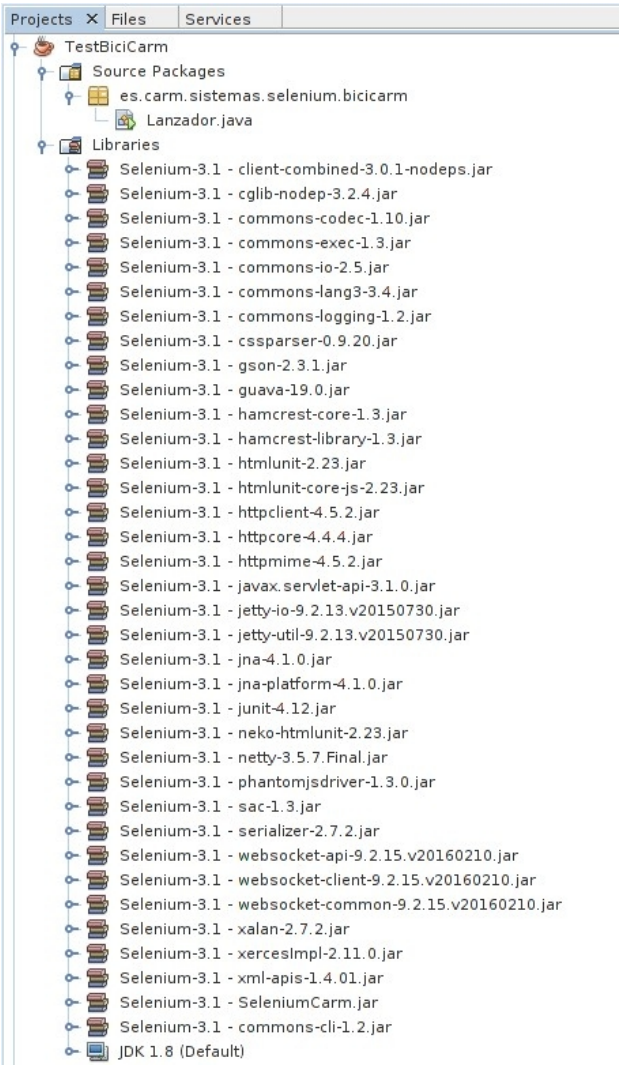
Podemos ver si se ha grabado la prueba como queremos si pulsamos al botón verde con tres rayas. Haciendo esto se reproducirá la prueba en nuestro navegador. Es importante bajar la velocidad de la prueba a Slow, porque sino es posible que la prueba vaya mas rápida que la respuesta del navegador y falle. Hasta aquí tenemos nuestra prueba en Selenium IDE, ahora tenemos que pasar esta prueba a un formato reproducible en un servidor. Para esto, vamos a crear una aplicación Java que se encargue de reproducir esta prueba sobre un servidor de pruebas Selenium. Abrimos nuestro entorno de desarrollo Java (para este ejemplo usamos NetBeans), y creamos un proyecto Java Application.



El nombre de proyecto será TestBiciCarm, y la clase principal será Lanzador dentro del es.carm.sistemas.selenium.bicicarm



Hay que añadir al proyecto las librerías de Selenium y la librería de SeleniumCarm. Para esto damos botón derecho sobre el nombre del proyecto y le damos a propiedades. Luego pinchamos sobre Librerías y le damos a Añadir JAR/FOLDER.



La clase “Lanzador” sera la ejecutable del proyecto y debe tener los siguientes parámetros:

```
-seleniumServer=url       (obligatorio) -> Url del servidor Selenium donde se lanzara la prueba
-filesUrl=url             (obligatorio) -> Directorio en el servidor Selenium donde se almacenan las
imagenes y las excepciones
-filesDir=directorio      (obligatorio) -> Url para acceder a las imagenes y excepciones generadas
-browser=Navegador        (obligatorio)    -> Navegador para realizar la prueba
(firefox,chrome,iexplorer,phantomjs) por defecto firefox
-profile=perfil_Navegador (opcional)    -> En caso del que navegador soporte perfiles, cual queremos
usar\n"
```

La clase SeleniumCarm ya se encarga del parseo de los parámetros de entrada de la prueba.

Además de la clase Lanzador, crearemos otra clase que será la que realmente encapsule el test. En este caso la he llamado Test. Esta clase en el constructor toma como parámetro un objeto de la clase TestSeleniumSSI. La clase Lanzador quedaría de la siguiente forma:

```
package es.carm.sistemas.selenium.bicicarm;
import es.carm.sistemas.TestSeleniumSSI;
```

```
public class Lanzador {
    public static void main(String[] args) {
        Test test = null;
        TestSeleniumSSI resultado = null;


        //Se crea un objeto de la Clase  TestSeleniumSSI, se le pasan los parametros de llamada del
test, y el nombre del mismo
        try {
           resultado = new TestSeleniumSSI(args, "BiciCarm");

        }  catch (Exception e) {
            System.out.println (e.getMessage());
            System.exit(0);
        }



//Se crea un objeto de la Clase que encapsula el TEST
        test = new Test(resultado);

//Se ejecuta el TEST
        resultado = test.test();

//Se saca por salida estandar el resultado del TEST
        System.out.println(resultado.getTResultado(false));

//Se retorna el TEST con el codigo de ejecución del mismo
        System.exit(resultado.getResultado());
}
```

La clase Test tendria el siguiente esqueleto:

```
public class Test {
    private String baseUrl;
    private WebDriver driver;
    TestSeleniumSSI resultado;
    public Test(TestSeleniumSSI resultado) {
        this.resultado = resultado;
    }
    public TestSeleniumSSI test() {
        try {
            driver = resultado.getDriver();;
            this.baseUrl = "http://www.carm.es";
            resultado.setOk("Test Correcto");

//AQUI IRIA EL CODIGO DEL TEST

        } catch (Exception e) {
            driver.switchTo().defaultContent();
            resultado.addMensajeExcepcion("Excepcion: " + e.getMessage().substring(0, 20), e);
            resultado.setCritical("Excepcion en el Test");
        } finally {
            driver.quit();
            return resultado;
        }
    }
}
```

A esta clase habría que añadir el código de lo que seria el Test en si. Para esto volvemos a Selenium IDE donde tenemos grabado el Test de ejemplo, hay que comprobar que en Options→Clipboard Format este marcado “Java/TestNG/WebDriver”. Despues seleccionamos todo el código que hemos generado y le damos a

Editar→Copiar. Luego vamos a nuestro código Java y pegamos el código generado por Selenium IDE. El código que genera es el siguiente:

```
            driver.get(baseUrl + "/web/pagina?IDCONTENIDO=1&IDTIPO=180");
            driver.findElement(By.id("q")).clear();
            driver.findElement(By.id("q")).sendKeys("BICI");
            driver.findElement(By.id("q_img")).click();

        driver.findElement(By.xpath("//div[@id='margenZonaPrincipal']/div[5]/p[4]/a/span")).click();
            // ERROR: Caught exception [ERROR: Unsupported command [selectFrame | main | ]]
            driver.findElement(By.cssSelector("font > b")).click();
            driver.findElement(By.name("O_DenCuerpo")).clear();
            driver.findElement(By.name("O_DenCuerpo")).sendKeys("ANALISTA DE APLICACIONES");
            driver.findElement(By.cssSelector("input[type=\"submit\"]")).click();
            driver.findElement(By.xpath("//tr[2]/td[2]/a/font/font/b/u")).click();
```

El código que genera Selenium IDE es un esqueleto para empezar a trabajar, ya que es posible que alguno de los comandos no los traduzca correctamente. En este caso vemos como no sabe traducir el comando para seleccionar el frame principal. En java se haría con el comando:

```
            driver.switchTo().frame("main");
```

Añadiendo este comando la clase del Test quedaría de la siguiente forma:

```
public class Test {
    private String baseUrl;
    private WebDriver driver;
    TestSeleniumSSI resultado;
    public Test(TestSeleniumSSI resultado) {
        this.resultado = resultado;
    }
    public TestSeleniumSSI test() {
        try {
                driver = resultado.initFirefoxDriver();
                this.baseUrl = "http://www.carm.es";
            driver.get(baseUrl + "/web/pagina?IDCONTENIDO=1&IDTIPO=180");
            driver.findElement(By.id("q")).clear();
            driver.findElement(By.id("q")).sendKeys("BICI Banco de datos de información al
ciudadano");
            driver.findElement(By.id("q_img")).click();

        driver.findElement(By.xpath("//div[@id='margenZonaPrincipal']/div[5]/p[4]/a/span")).click();
            driver.switchTo().frame("main");
            driver.findElement(By.cssSelector("font > b")).click();
            driver.findElement(By.name("O_DenCuerpo")).clear();
            driver.findElement(By.name("O_DenCuerpo")).sendKeys("ANALISTA DE APLICACIONES");
            driver.findElement(By.cssSelector("input[type=\"submit\"]")).click();
            driver.findElement(By.xpath("//tr[2]/td[2]/a/font/font/b/u")).click();
              resultado.setOk("Test Correcto");
        } catch (Exception e) {
            driver.switchTo().defaultContent();
            resultado.addMensajeExcepcion("Ex: " + e.getMessage().substring(0, 30), e);
            resultado.setCritical("Excepcion en el Test");
        } finally {
            driver.quit();
            return resultado;
        }
    }
}
```

El código que genera Selenium IDE para pinchar sobre los resultados de la búsqueda, no es del todo correcto ya que localiza los elementos por su situación en el resultado

de la búsqueda. Si cambia el orden del resultado de la búsqueda la prueba fallaría. Voy a cambiar esas localizaciones por buscar el texto del enlace. El resultado sería el siguiente:

```
Sustituir:
  driver.findElement(By.xpath("//div[@id='margenZonaPrincipal']/div[5]/p[4]/a/span")).click();
por:
  driver.findElement(By.linkText("BICI - Banco de datos de información al ciudadano")).click();

y

driver.findElement(By.xpath("//tr[2]/td[2]/a/font/font/b/u")).click();

por:

driver.findElement(By.linkText("1.Orden de 27 de abril de 2016. BORM de 17 de mayo de 2016. Programa
de Materias Comunes. Cuerpo Técnico, Opción Analista de Aplicaciones.")).click();
```

Podemos comprobar además, que el documento final que abrimos en la prueba, contenga cierto texto. Para esto añadimos las siguientes lineas al test:

```
            driver.switchTo().frame("main_set") ;


            //Se dan 10 segundos de espera para que pueda cargar el pdf
            try {
              wait = new WebDriverWait(driver,10);
              wait.until(ExpectedConditions.presenceOfElementLocated(By.xpath("//*[contains(text(),
'Orden de 27 de abril de 2016')]")));
            } catch (Exception e) {
              throw new Exception("El documento PDF no carga o no es correcto");
            }
```

Hasta aquí tendríamos preparado para ejecutarse el código que hemos generado con Selenium Ide. Podemos introducir algunas mejoras, por ejemplo obtener los datos que tardan en ejecutarse partes de este test. Podemos dividir el test en varias partes y sacar en el resultado del test lo que tarda cada una de estas partes, además el sistema de monitorización hará gráficas de estos valores en el tiempo. En este ejemplo voy a dividir la ejecución en 4 partes:

- Carga de la pagina
- Búsqueda de la palabra BICI
- Búsqueda temario de Analista de Aplicaciones en BICI
- Apertura del documento

Usaremos la función System.currentTimeMillis() para obtener los tiempos antes y después de la ejecución de las partes del test.

```
         //Toma de tiempo antes de la ejecución de la parte de la prueba
            inicio = System.currentTimeMillis();
            driver.get(baseUrl);
```

```
         //Toma de tiempo despúes de la ejecución de la parte de la prueba
            fin = System.currentTimeMillis();
         //Añadimos el resultado a la salida del test diciendole que no haga una captura de pantalla
            resultado.addMensaje("carga", "Carga pagina inicial", fin - inicio, false);
```

También podemos utilizar estas métricas para detectar fallos en el test, es decir, si el tiempo que tarda en ejecutarse una parte de la ejecución del test es demasiado alto, podemos marcar que el resultado del test es un warning o un error. En este caso tenemos dos opciones, o bien disparar una excepción y cortar la ejecución del test, o bien marcar el test como warning o error y dejar terminar el test, de forma que se tomen tiempos de las demás fases del mismo.

```
        if (Long.compare((fin-inicio), 8000)>0) resultado.setWarning("Tiempo de Carga demasiado
alto");
```

La clase del test completa sería la siguiente:

```
package es.carm.sistemas.selenium.bicicarm;
import es.carm.sistemas.TestSeleniumSSI;
import org.openqa.selenium.*;
import org.openqa.selenium.WebDriver;

public class Test {
    private String baseUrl;
    private WebDriver driver;
    TestSeleniumSSI resultado;

    public Test(TestSeleniumSSI resultado) {
        this.resultado = resultado;
    }
    public TestSeleniumSSI test() {
       long inicio;
        long fin;

        try {

            driver = resultado.getDriver();
            resultado.setOk("Test Correcto");

            this.baseUrl = "http://www.carm.es";

            inicio = System.currentTimeMillis();
            driver.get(baseUrl);
            fin = System.currentTimeMillis();
            resultado.addMensaje("carga", "Carga pagina inicial", fin - inicio, false);

            if (Long.compare((fin - inicio), 8000) > 0) {
                resultado.setWarning("Tiempo de Carga demasiado alto");
            }

            driver.findElement(By.id("q")).clear();

            driver.findElement(By.id("q")).sendKeys("BICI Banco de datos de información al
ciudadano");


            inicio = System.currentTimeMillis();
            driver.findElement(By.id("q_img")).click();
            fin = System.currentTimeMillis();
            resultado.addMensaje("busquedaCarm", "Busqueda BICI", fin - inicio, false);

            driver.findElement(By.linkText("BICI - Banco de datos de información al
ciudadano")).click();
```

```
            driver.switchTo().frame("main");
            driver.findElement(By.cssSelector("font > b")).click();
            driver.findElement(By.name("O_DenCuerpo")).clear();
            driver.findElement(By.name("O_DenCuerpo")).sendKeys("ANALISTA DE APLICACIONES");

            inicio = System.currentTimeMillis();
            driver.findElement(By.cssSelector("input[type=\"submit\"]")).click();
            fin = System.currentTimeMillis();
            resultado.addMensaje("busquedaBICI", "Busqueda Temario en BICI", fin - inicio, false);

            inicio = System.currentTimeMillis();
            driver.findElement(By.linkText("1.Orden de 27 de abril de 2016. BORM de 17 de mayo de
2016. Programa de Materias Comunes. Cuerpo Técnico, Opción Analista de Aplicaciones.")).click();


            driver.switchTo().frame("main_set") ;


            //Se dan 10 segundos de espera para que pueda cargar el pdf
            try {
              wait = new WebDriverWait(driver,10);
              wait.until(ExpectedConditions.presenceOfElementLocated(By.xpath("//*[contains(text(),
'Orden de 27 de abril de 2016')]")));
            } catch (Exception e) {
              throw new Exception("El documento PDF no carga o no es correcto");
            }

            fin = System.currentTimeMillis();
            resultado.addMensaje("abrirPdf", "Abrir PDF", fin - inicio, false);


        } catch (Exception e) {
            driver.switchTo().defaultContent();
            resultado.addMensajeExcepcion("Ex: " + e.getMessage().substring(0, 30), e);
            resultado.setCritical("Excepcion en el Test");

        } finally {
            driver.quit();
            return resultado;
        }
    }
}
```

Para lanzar la prueba desde la linea de comandos, teniendo configurado el servidor selenium en la misma maquina, y permisos para escribir sobre /tmp y un servidor web sirviendo el directorio /tmp en http://localhost/ficheros habria que ejecutar el siguiente comando:

```
java -jar TestBiciCarm.jar es.carm.sistemas.selenium.bicicarm.Lanzador
-seleniumServer="http://127.0.0.1:4444/wd/hub"  -filesUrl="http://localhost/ficheros/"
-filesDir="/tmp/" -browser="firefox"
```

La ejecución del test da como resultado la siguiente salida:

```
OK - Test Correcto
1 Carga pagina inicial  -> 2835ms
2 Busqueda BICI  -> 2133ms
3 Busqueda Temario en BICI  -> 75ms
4 Abrir PDF  -> 1396ms
 | carga=2835ms busquedaCarm=2133ms busquedaBICI=75ms abrirPdf=1396ms
```

9. Tips Selemiun

9.1. Inyección de código JavaScript

Selenium permite inyectar código javascript en cualquier momento de la ejecución de la prueba. Para esto hacemos lo siguiente:

```
 JavascriptExecutor js = (JavascriptExecutor) driver;
long loadEventEnd = (Long) js.executeScript("return window.performance.timing.loadEventEnd;");
long navigationStart = (Long) js.executeScript("return window.performance.timing.navigationStart;");

System.out.println("Tiempo de carga de la pagina es (" + loadEventEnd + "-" + navigationStart + ")"
+ (loadEventEnd -  navigationStart) + "ms.");
```

En este ejemplo calculamos el tiempo de carga de una página preguntándole al navegador por javascript.

9.2. Ir al frame principal

Cuando estamos manejando frames, a veces es necesario volver al frame principal, para esto ejecutamos:

```
driver.switchTo().defaultContent();
```

9.3. Seleccionar Frame

Para movernos a un frame en particular usamos el comando:

```
driver.switchTo().frame("Main");
```

9.4. Aceptar una ventana de alerta

Si se nos abre un popup con una ventana de alerta, podemos cerrarlo de la siguiente forma:

```
                    try {
                        if (driver.switchTo().alert() != null) {
                            Alert alert = driver.switchTo().alert();
                            //alert.dismiss();
                            alert.accept();
                        }
                    } catch (NoAlertPresentException e) {
                    }
```

9.5. Seleccionar en un desplegable

Para seleccionar una opción dentro de un desplegable:

```
new
Select(driver.findElement(By.id("Id_del_desplegable"))).selectByVisibleText("Texto_de_la_opcion_a_el
egir");
```

9.6. Esperar a que un elemento sea visible

Cuando una pagina contiene llamadas ajax, es posible que la pagina haya terminado de cargar, pero todavía haya elementos sin terminar de cargar, para esto podemos establecer esperas con timeout. En este caso esperamos como máximo 10 segundos a que un elemento sea visible:

```
wait = new WebDriverWait(driver, 10);
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("id_del_elemento")));
```

9.7. Cambiar a otra ventana

Cuando pulsamos sobre un enlace y este abre otra ventana, podemos cambiarnos de ventana de la siguiente forma:

```
          String MainWindow = driver.getWindowHandle();
          Set<String> s1 = driver.getWindowHandles();
           Iterator<String> i1 = s1.iterator();

           while (i1.hasNext()) {
               String ChildWindow = i1.next();
               if (!MainWindow.equalsIgnoreCase(ChildWindow)) {
                    driver.switchTo().window(ChildWindow);
                    driver.switchTo().defaultContent();
               //Estariamos en otra ventana
             }
        }
```

9.8. Stale elements

Puede ser que cuando estamos esperando un valor de un elemento html este cambie (Por una ejecución javascript, o por que es un elemento que se refresca por ajax y todavia no se ha cargado). Esto provoca una excepción del tipo StaleElementReferenceException. Podemos esperar a que el valor sea estable con el siguiente código:

```
   WebElement getStaleElemById(String id) {
       try {
           return driver.findElement(By.id(id));
       } catch (StaleElementReferenceException e) {
           return getStaleElemById(id);
       }
   }
```

9.9. Elegir perfil de firefox

Podemos tener varios perfiles configurados en firefox, en código para elegir un perfil u otro ejecutaremos:

```
driver = resultado.initFirefoxDriver("Nombre_del_perfil");
```

9.10. Seleccionar navegador

Si tenemos instalados clientes con distintos navegadores, podemos ejecutar la prueba en uno u otro de la siguiente forma:

```
//Para Firefox
driver = resultado.initFirefoxDriver();
//Para Internet Explorer
driver = resultado.initExplorerDriver();
//Para google Chrome
driver = resultado.initChromeDriver()
```