@Before Selenium, lo básico.

El siguiente apunte es una recopilación sencilla a considerar a la hora de empezar en el mundo de Selenium Webdriver. Asumo que programaste alguna vez, basta con haber programado en algún lenguaje tipo C, C++, C#, PHP, Ruby, etc. La sintaxis es similar y este apunte te va a venir bien para empezar a crear casos de pruebas lo antes posible. No veas este apunte como una manual, entiéndelo como la base con la cual nos vamos a comunicar con las pruebas o una referencia la cual echar mano.

La idea es tener las herramientas mínimas para empezar a realizar casos de prueba.

Contenido

1. DataType

Los tipos de datos son básicos en todo lenguaje de programación y en Java n oes la excepción, esta tabla resume algunos de los tipos más usados:

Tipo de Datos	Almacena en memoria	Observación	Declaración
Int	32 bits	Sólo números enteros.	int i = 1000;
long	64 bits	Es como el integer (int) pero con mayor capacidad.	long l = 100000;

double	64 bits	Número decimal, también permite almacenar números enteros.	double d1 = 10.1111;
			
			double d2 = 10000;

char		Utilizado para almacenar una letra.	char c = 'd';
boolean		Almacena valores booleano (true/false)	boolean b = true; boolean b = false;
string		No es un tipo de dato, pero es una clase útil para almacenar cadenas de texto.	string str = "Hola Mundo";

Ejemplo de código de cómo utilizarlos y declararlos:

```
public class tiposDatos {

 public static void main(String[] args) {
  int i         = 4523;   //Almacena 32 bit de valores enteros.
  long l        = 652345; //Almacena 64 bit de valores enteros.
  double d1     = 56.2354;//Almacena 64 bit de valores decimales.
  double d2     = 12456;  //También puede utilizar valores enteros.
  char c        = 'd';    //Almacena solo un caracter.
  boolean t     = true;   //Almacena solo valores TRUE o FALSE.
  String str    = "Hello World";  // Almacena una cadena de texto.

  System.out.println("La variable Integer i es = "+i);
  System.out.println("La variable Long l = "+l);
  System.out.println("La variable Double d1 es = "+d1);
  System.out.println("La variable Double d2 es = "+d2);
  System.out.println("LA variable char c es = "+c);
  System.out.println("La variable Boolean b es = "+t);
  System.out.println("La variable String str es = "+str);
```

```
 }
}
```

Ejecuta el ejemplo anterior y verifica la salida.

String Class[1]

En Java, STRING no es un tipo de dato, es una clase. La definición formal dice que:" son objetos preparados por una secuencia de caracteres", la clase String tiene muchas funciones incorporadas que podemos usar para realizar diferentes acciones en una cadena palabras.

Es importante manejar el concepto y la funcionalidad de String. Será de mucha utilidad a la hora de realizar casos de prueba, a continuación un pequeño ejemplo para entender los diferentes métodos de la clase String en Java:

```
package beforeSelenium;

public class Ejemplo_String {

public static void main(String[] args) {

            String st1 = "Hola Estimados QA";
            String st2 = " aprenderemos cómo funciona la clase String. ";

            //Comparo si dos String son iguales, retorna false
            System.out.println("st1 igual a st2 = "+st1.equals(st2));

            //Concatenar st2 con st1.
            System.out.println("Concatenar st1 y st2 = "+st1.concat(st2));

            //Recibo el noveno caracter indexado en el string st1.
            System.out.println("Noveno caracter = "+st1.charAt(9));

            //Muestro largo de st1.
            System.out.println("Largo de St1 = "+st1.length());

            //Pasar st1 a minúsculas.
            System.out.println("St1 en minúsculas = "+st1.toLowerCase());

            //convertir st1 a mayúsculas.
            System.out.println("St1 en mayúsculas = "+st1.toUpperCase());

            //Indica posición de primera letra 'a' de st1. Comienza en 0.
            System.out.println("Posición primera letra 'a' = "+st1.indexOf('a'));
```

[1] String(Java Platform SE 7), https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

```
        //Muestra posición de segunda letra 'a' en st1. Comienza en 0.
        System.out.println("Posición segundo caracter a = "+st1.indexOf('a', 4));

        //Recibe la posición de la palabra Estimados
        System.out.println("Posición de 'Estimados' = "+st1.indexOf("Estimados"));

        //Convertir int a String
        int j = 75;
        String val2 = String.valueOf(j);
        System.out.println("Valor de val2 = "+val2);

        //Convertir string a int.
        String val1="50";
        int i = Integer.parseInt(val1);
        System.out.println("Valor de i = "+i);

        //Imprimir el string desde la psoción 5 hasta la 12.
        System.out.println("Recibe sub string  de st1 = "+st1.substring(5, 13));

        //Split st1.
        String splt[] = st1.split("Estimados");
        System.out.println("Primera parte = "+splt[0]);
        System.out.println("segunda parte = "+splt[1]);

        //Trim. Elimina espacios en blanco en inicio y fin de la cadena.
        System.out.println("Trim a  st2 ="+st2.trim()+"FIN");
       }
      }
```

2. Declaración IF ELSE

If, if else y los if anidados son usados recurrentemente para para tomar decisiones o generar condicionantes en un flujo.

Simple IF

Será ejecutado sólo cuando la condición impuesta se cumpla:

```
if(i<j){
        System.out.println(“Valor de i(“+i+”) es menor que el valor de j(“+j+”).”);
}
```

IF-ELSE

Cuando retorne verdadero se ejecutará el primer bloque, en caso contrario se ejecutará el segundo bloque ELSE:

```
if (i>=j){
 System.out.println("Valor de i("+i+") es mayor o igual que j ("+j+")." );
}else{
 System.out.println("Valor de i("+i+") es menor o igual que j("+j+")." );
}
```

IF ELSE anidados

Se pueden anidar varios IF, de mucha utilidad

```
if (k<i){
         System.out.println("Valor de k("+k+") es menor que I i("+i+")" );
}else if (k>=i && k<=j){
         System.out.println("Valor de k("+k+") está entre i("+i+") y j("+j+")" );
}else{
         System.out.println("Valor de k("+k+") es mayor que j("+j+")" );
}
```

A continuación un ejemplo implementando todo lo anterior:

```
package beforeSelenium;

public class Aprendiendo_IF {

          public static void main(String[] args) {

           int i = 25;
           int j = 50;
           int k = 24;
           //Ejemplo if
           System.out.println("***Ejemplo de IF***");
           if (i<j) {
            System.out.println("Valor de i("+i+") es menor que j ("+j+")." );
           }

           //Ejemplo If Else
           System.out.println("\n***Ejemplo If Else***");
           if (i>=j) {
            System.out.println("Valor de i("+i+") es mayor o igual que j("+j+")." );
           }else{
            System.out.println("Valor de i("+i+") es menor que j("+j+")." );
           }

           //Ejemplo If Else anidado
           System.out.println("\n***Ejemplo If Else anidado***");
           if (k<i) {
```

```
            System.out.println("Valor de k("+k+") es menor que i("+i+")" );
           }else if (k>=i && k<=j){
            System.out.println("Valor de k("+k+") está entre i("+i+") y j("+j+")" );
           }else{
            System.out.println("Valor de k("+k+") es mayor que j("+j+")" );
           }
          }
         }
```

3. Ciclo For

Los ciclos (for, while, do while) cumplen un rol muy importante en selenium webdriver, como sabrás, algunas veces necesitas realizar una acción varias veces y los ciclos nos ayudan muchos.

El ciclo FOR se compone de tres partes: inicialización de variables(**I**), condición para terminar el ciclo(**II**) y la incrementación/decrementos de una variable(**III**):

```
for(int i=0; i<=3; i++){
 System.out.println("Value Of Variable i is " +i);
}
```

El siguiente código es un ejemplo:

```
package beforeSelenium;

public class Aprendiendo_For {

	public static void main(String[] args) {
		for(int i=0; i<=3; i++){ //Este ciclo se ejecutará 4 veces
			System.out.println("Valor de i = " +i);
		}

		int i=0;
		int k = 200;
		for(int j=3; j>=i; j--){ //Este ciclo se ejecutará 4 veces
			System.out.println("\nValor de j = " +j);
			k = k-10;
		}

		System.out.println("\nValor de k = " +k);
	}
}
```

4. While, Do while

Ciclo while, básicamente es un bloque de código que se ejecutará mientras se cumpla (sea verdadera) la condición:

```
int i = 0;
while(i<=3){
 System.out.println("Valor de I = "+i);
 i++;
}
```

Ciclo Do while, es igual al ciclo anterior, la diferencia es que primero ejecuta el bloque de código y después valida la condición.

```
int j=0;
do{
 System.out.println("Valor de j = "+j);
 j=j-1;
}while(j>0);
```

5. Arrays – Arreglos – Vectores

A modo general podemos mencionar arrays, arreglos o vectores y seguir hablar de lo mismo. Lo vamos a definir como una "variable[2]", que me permita almacenar 'n' otras variables dentro de ella, del mismo tipo (int, char, String), además cada dato almacenado tiene un índice(index) único. En Java existen dos tipos o formas de crear un arreglo: Una dimensión y bidimensional.

Arreglos unidimensionales

Índice	0	1	2	3	4	5
Valores	1	2	3	4	5	6

El ejemplo anterior muestra el índice y los valores del arreglo, lo podemos ver como una tabla de Excel. El índice nos dirá de forma única la ubicación de cada valor, podíamos tener el siguiente arreglo:

Índice	0	1	2	3	4	5
Valores	A	A	B	A	A	A

Cada valor 'A' es identificable porque tiene una posición única, esto nos permite realizar varias tareas sobre el arreglo, desde contar las letras 'A', hasta convertirlas todas a 'B' o lo que se nos ocurra, para ello debemos recorrer un arreglo y para eso nos ayudarán los índices:

```
package beforeSelenium;

public class Aprendiendo_Array {
```

[2] No me tiren piedras.

```
        public static void main(String[] args) {

         int a[] = new int[6]; //Declaro array y lo creo con largo 6.
         a[0] = 10; //Inicializo primer elemento
         a[1] = 12; //Inicializo el resto ...
         a[2] = 48;
         a[3] = 17;
         a[4] = 5;
         a[5] = 49;

         for(int i=0; i<a.length; i++){
                System.out.println(a[i]);
         }
        }

       }
```

Otra forma de crear el mismo arreglo es:

```
int a[] = {10,12,48,17,5,49};
```

Lo que hacemos con el ciclo **FOR** es obtener el tamaño del arreglo con 'a.length', en este caso es 6, e inicializamos 'i' en 0 (Cero), esto lo hacemos porque nuestro vector también empieza en '0' y la impresión de pantalla la realizaremos de la forma 'a[i]'.

Arreglos bidimensionales

			Índice de filas
Valor del Arreglo	Usuario1	Contraseña1	0
Valor del Arreglo	Usuario2	Contraseña2	1
Valor del Arreglo	Usuario3	Contraseña3	2
Índice de columnas	0	1	

Lo podemos resumir en varios arreglos unidimensionales pegados unos arriba sobre otro, para recorrerlo completo no sólo necesitaríamos un índice de posición de izquierda a derecha, si no que de arriba abajo (como una plantilla de Excel A1, Z45 donde la letra representa la columna y el número la fila). Para esto vamos a necesitar un índice de columnas y esto se traduce en un FOR anidado:

```
package beforeSelenium;

public class Aprendiendo_Array_dimensiones {

         public static void main(String[] args) {
          String str[][] = new String[3][2]; //3 filas y 2 columnas
          /*
           * Arreglo 3x2 => 3 filas y 2 columnas
```

```
            *
            *     |--Columna1-|---Columna2--|
            *     | Usuario1       |       Password1 | Fila1
            *     | Usuario2       |       Password2 | Fila2
            *     | Usuario3       |       Password3 | Fila3
            *
            */
           str[0][0]="Usuario1";
           str[1][0]="Usuario2";
           str[2][0]="Usuario3";
           str[0][1]="Password1";
           str[1][1]="Password2";
           str[2][1]="Password3";

           for(int i=0; i<str.length; i++){//Se ejecutará 3 veces.
           for(int j=0; j<str[i].length; j++){//Se ejecutará 2 veces por cada
iteración.
             System.out.println(str[i][j]);
            }
           }
          }
         }
```

También se puede declarar este tipo de arreglo de la siguiente forma:

```
String str[][] =
{{"Usuario1","Password1"},{"Usuario2","Password2"},{"Usuario3","Password3"}};
```

6. Métodos en Java

Cuando hacemos el paso a paso de los casos de pruebas tocará repetir, y mucho, por ejemplo iniciar sesión o cerrar sesión. Para no tener que escribirlos a cada rato utilizaremos lo que se conoce, en la programación, como métodos. Diremos que es un grupo de instrucciones o código que realizan algunas acciones u operaciones cuando es llamado desde el método principal.

```
package beforeSelenium;

public class Aprendiendo_Metodo {

          public static void main(String[] args) {
           Test1(); //Llamada dentro del método principal.

          }

          public static void Test1(){ //Muestra por pantalla.
           System.out.println("Test1 es llamado desde el método principal.");
```

```
        }

        public static void Test2(){ //Método que nunca es llamado.
         System.out.println("Test2 nunca es llamado.");

        }

       }
```

7. Modificadores de Acceso

Se trata de "configurar" o definir el nivel de acceso para una clase, un método, variables y constructores dentro del código. En Java son cuatro:

Público (public): Podemos acceder a métodos o variables públicas desde todas las clases desde el mismo paquete u otro.

Privado (private): Los métodos y variables sólo pueden ser accedidos desde la misma clase. No podemos acceder desde otras clases o sub clases del mismo paquete.

Protegido (protected): Los métodos protegidos pueden ser accedidos desde clases del mismo paquete o subclases.

Sin modificador de acceso (No Access Modifier): Si el método no tiene ningún modificador de acceso, entonces podemos acceder a él dentro de todas las clases del mismo paquete.

A continuación mostraré la forma de trabajar y diferenciar estos tipos de accesos:

```
package Accesos1;

public class Accessmodif {

 public static int i=10;
 private static String str="QA";
 protected static double d=30.235;
 static char c='g';

 public static void main(String[] args) {
  Acceso_publico();
  Acceso_privado();
  Acceso_protegido();
  Acceso_sma();
 }

 //Método de acceso público, puede ser accesible por cualquier clase.
 public static void Acceso_publico(){
  System.out.println("Acceso_publico() Ejecutado");
  System.out.println("Valor de i = "+i);
  System.out.println("Valor de str = "+str);
```

```
  System.out.println("Valor de d = "+d);
  System.out.println("Valor de c = "+c);
 }

 //Método con acceso privado. Puede ser accesible sólo desde la misma clase.
 private static void Acceso_privado(){
  System.out.println("Acceso_privado() Ejecutado");
 }

 //Método con acceso protegido. Puede ser accedido desde cualquier clase del
mismo paquete.
 protected static void Acceso_protegido(){
  System.out.println("Acceso_protegido() Ejecutado");
 }

 //Método sin modificador de acceso. Puede ser accedido por todas las clases
del mismo paquete.
 static void Acceso_sma(){
  System.out.println("Acceso_sma() Ejecutado");
 }
}
```

```
package Accesos1;

public class Access {

 public static void main(String[] args) {
         //Accede al método público.
         Accessmodif.Acceso_publico();

         //No puede acceder a un método privado fuera de la clase.
         //Accessmodif.Acceso_privado();

         //Puede acceder a un método protegido en el mismo paquete.
         Accessmodif.Acceso_protegido();

         //Accede a un método sin métodos de acceso dentro del mismo paquete.
         Accessmodif.Acceso_sma();

         //Accede a variables públicas desde fuera de la clase.
         System.out.println("\nValor de i = "+Accessmodif.i);

         //No puede acceder a variables privadas fuera de la clase.
         //System.out.println("Valor de str = "+Accessmodif.str);

         //Accede a variable protegidas dentro del mismo paquete.
```

```
        System.out.println("Valor de d = "+Accessmodif.d);

        //Accede a variables sin modificador de acceso dentro del mismo
paquete.
        System.out.println("Valor de c = "+Accessmodif.c);
 }
}
```

```
package Accesos2;

import Accesos1.Accessmodif;

public class Accessing extends Accessmodif{

 public static void main(String[] args) {
        //Accede a método público desde otro paquete.
        Acceso_publico();

        //No puede acceder a método privado fuera de la clase.
        //Acceso_privado();

        //Accede a método protegido dentro de una clase hijo.
        Acceso_protegido();

        // No puede acceder a un método sin modificador de acceso fuera del
mismo paquete.
        //Acceso_sma();

        //Accede a variable pública fuera del paquete.
        System.out.println("\nValor de i = "+i);

        //No puede acceder a variable privada.
        //System.out.println("Valor de str = "+str);

        //Accede a variable protegida como de una clase hija.
        System.out.println("Valor de d = "+d);

        //No puede acceder a la variable por estar fuera del paquete.
        //System.out.println("Valor de c = "+c);
 }
}
```

8. Retorno de métodos

¿Qué retornas los métodos al ser invocados? Pueden retornar algún resultado en algún 'tipo de dato' o simplemente retornar void(vacío):

```
package beforeSelenium;

public class Aprender_Retorno {

 static int c;
 static double d;
 public static void main(String[] args) {
         RetornaEntero(2,3);
         RetornaDouble(7,3);
                  System.out.println("Valor de c = "+c);
                  System.out.println("Valor de d = "+d);
         MensajeSinRetorno();
 }

 //Retorna valor entero
 public static int RetornaEntero(int a, int b){
  c=a*b;
  return c;
 }
 //Retorna double
 public static double RetornaDouble(double a, double b){
  d=a/b;
  return d;
 }
 //No retorna nada
 public static void MensajeSinRetorno(){
  System.out.println("Mensaje sin retorno 'hola Mundo!'");
 }
}
```

Si se fijan en el ejemplo anterior no declaramos en el main() las variables c y d.

9. Métodos estáticos y no estáticos

La palabra reservada 'Static' en un método, lo describe como un 'método estático', si no tiene dicha palabra entonces se entiende que el método no es estático, sencillo. Las diferencias son básicamente que podremos llamar directamente a un método estático desde un método estático, en un método no estático no podemos acceder dentro de un método estático. Trataré de explicarlo nuevamente:

- Método/variable no estático, no puede ser accesible desde un método/variable estático. Pero un método o variable no estática puede acceder a una variable/método estático, o sea que no tiene restricciones de acceso.

- Un método estático está asociado a una clase y un método no estático a un objeto.

```
package beforeSelenium;

public class Estatico_noEstatico {

 static int ruedas = 2;
 int precio = 25000;
 public static void main(String[] args) { //Método estático.
  // Podemos acceder a un método estático dentro de un método estático.
  bici1();

  //Puedes acceder a una variable estática dentro de un método estático.
  System.out.println("Desde étodo estático Main : ruedas = "+ruedas);

  //Una variable no estática no accede directamente dentro de un método
estático.
  //System.out.println("Desde método estático Main : ruedas = "+precio);

  //No puedes ingresar a un método no estática dentro de un método estático
  //bici2();

  //Creamos un objeto para acceder a una variable o método no estático,
dentro de un método estático.
  Estatico_noEstatico sn = new Estatico_noEstatico();

  //Ahora recién podremos acceder a un método no estático, dentro de un
método estático.
  sn.bici2();

  //Accedemos a una variable no estática dentro de un método estático usando
referencia de objetos.
  System.out.println("Desde método estático Main : price = "+sn.precio);

 }

 public static void bici1(){ //Método estático.
  //Accede a variable estática dentro de método estático.
  System.out.println("bicicleta 1 método estático : ruedas = "+ruedas);

  //No puede acceder a variable no estática desde un método estático.
  //System.out.println(price);

 }

 public void bici2(){ //Método no estático.
```

```
  //Puedes acceder a una variable estática dentro de un método no estático.
  System.out.println("Bicicleta 2 método no estático : ruedas = "+ruedas);

  //Puedes acceder a una variable no estática dentro un método no estático.
  System.out.println("Bicicleta 2 método no estático : precio = "+precio);

  //Accede a método estático desde método no estático.
  bici1();
 }
}
```

```
package beforeSelenium;

public class static_ousideclass {

 public static void main(String[] args) { //método estático.

  //Llama una función estática desde otra clase directamente usando el
nombre.
  Estatico_noEstatico.bici1();

  //Llama una variable estática desde otra clase directamente.
  System.out.println("Usando variable estática de otra
clase"+Estatico_noEstatico.ruedas);

  //Crea un objeto de la clase Estatico_noEstatico.
  Estatico_noEstatico oc = new Estatico_noEstatico();

  //Accede a variable no estática de otra clase dentro de un método estático
usando referencia a objeto.
  System.out.println("Accesando a una variable no estática fuera de la clase :
"+oc.precio);

  //Accedemos a método no estático desde otra clase dentro de un método
estático usando referencia a objeto.
  oc.bici2();
 }
}
```

10. Objetos en Java

El mantra dice: “un objeto es una instancia de una clase”, por lo que un objeto también es un conjunto de métodos y variables relacionados. Cada objeto tiene su propio estado y comportamiento. Los objetos son, generalmente, usados por constructores en Java. Para nuestros propósitos, Selenium Webdriver, vamos a crear y utilizar varios objetos en nuestros casos de prueba.

Para crear un objeto nos basaremos en que una bicicleta es una objeto de un a clase vehículo con sus propios estados y comportamientos, una motocicleta sería otro objeto de la misma clase vehículo. Un ejemplo de la creación de un objeto de la clase vehículo utilizando la palabra “new”:

```
public class vehiculo {
 public static void main(String[] args) {
  //Creación de objeto por clase vehículo.
  //Bicicleta es la variable de referencia de este objeto.
  vehiculo bicicleta = new vehiculo("Rojo");

 }
 //Constructor con parámetro color. Recibirá el valor del objeto vehículo.
 public vehiculo(String color){
  //Mostrará valor recibido.
  System.out.println("El color del vehículo es "+color);
 }
}
```

En el ejemplo anterior utilizamos un constructor para pasar el valor del objeto. El concepto de constructor lo veremos más adelante. Debemos tener algo en mente, la bicicleta no es un objeto si no que es la variable de referencia del objeto vehículo. Por lo tanto se puede dividir un objeto en tres partes:

Declaración: declaración de variable por objeto, en este ejemplo bicicleta l oes.

Instanciación: la creación de objetos utilizando la palabra “new”.

Inicialización: Llamar a un constructor es conocido como inicialización.

Puedes usar un objeto de una clase a acceder a una variable o método no estático de una misma clase o diferente. En el siguiente ejemplo muestro como crear múltiples clases de objetos pasando distintos tipos de valores en el constructor.

```
public class vehiculo {

 public static void main(String[] args) {
  //Crea 2 objetos, ambos con diferentes variables de referencia.
  vehicle bicicleta = new vehiculo("Negro", 2, 4500, 3.7);
  vehicle motorcicleta = new vehiculo("Azul", 2, 67000, 74.6);

 }

 public vehiculo(String color, int ruedas, int precio, double velocidad){
  System.out.println("Color = "+color+", Ruedas = "+ruedas+", Precio = "+precio+",
Velocidad = "+velocidad);
 }
}
```

11. Tipos de variabes

Como ya sabemos, una variable nos provee un espacio de memoria (tipo y tamaño declarado) para almacenar algún valor. Existen tres tipos de variables:

1. *Variables Locales*: son variables que son declaradas dentro del método o constructor. Están limitadas para ese método y constructor. Es necesario inicializar esta variable antes de usasrla.
2. *Variables de instancia (Non Static):* son usadas, normalmente, con objetos esto quiere decir que se crean y destruyen con el mismo. Estas variables son accesibles directamente por todos los métodos y constructores no estáticos de esa clase. Para acceder, dentro del método estático necesita crear un objeto de esa clase. Siempre se inicializan con sus valores predeterminados en función de sus tipos de datos. Pueden acceder directamente a la instancia de la variable dentro de la misma clase, si quieres acceder fuera de la clase debes referenciar el objeto con el nombre de la variable. Es importante porque la instancia de variable se usa mucho en Selenium Webdriver.
3. *Variables de clases (Static):* Igual como una variable de instancia, las variables de clase se declaran en el nivel clase (fuera del método o bloque del constructor), la única diferencia es que estas variables se declaran utilizando palabras clave estáticas.

```
package beforeSelenium;

public class Variable_Estatica {

        //Variable de clase – Nombre_Universidad será el mismo para ambos
departamentos declarados como variable estática.
        public static String Nombre_Universidad = "Aplapac";

        //Variables instanciadas.
        private String facultad = "Ingeniería en Computación";
        private String nombre;
        private double percentil;

        public static void main(String[] args) {//Método Estático
                //Puede acceder a una variable de clase directamente si lo
necesita ejemplo Nombre_Universidad.
                Variable_Estatica estudiante1 = new Variable_Estatica("Pepe");
                estudiante1.setPercentage(67.32);
                estudiante1.print_details();
                //Accede a una instancia de variable usando objeto por
referencia.
                //Ejemplo : student1.name = "Robert";

                Variable_Estatica estudiante2 = new Variable_Estatica("Lota");
                estudiante2.setPercentage(72.95);
                estudiante2.print_details();
```

```
        }

        public Variable_Estatica(String student_name){//Constructor
                //Puede acceder una instancia de variable directamente
dentro del constructor.
                nombre = student_name;
        }

        public void setPercentage(double perc){
                //Accede a una instancia de variable directamente dentro de
un método no estático.
                percentil = perc;
        }

        public void print_details(){
                int Year = 2018; //Variable local - no puede acceder fuera de
este método.
                System.out.println("Año = "+Year);
                System.out.println("Universidad = "+Nombre_Universidad);
                System.out.println("Facultad estudiante = "+facultad);
                System.out.println("Nombre estudiante = "+nombre);
                System.out.println("Percentil de estudiante = "+percentil+"%");
                System.out.println("**********************");
        }

}
```

```
package beforeSelenium;

public class Otra_Clase {

         private String Departmento = "Ingeniería Química";
         private String nombre;
         private double percentil;
         public static void main(String[] args) {
          Otra_Clase estudiante1 = new Otra_Clase("Juanito");
          estudiante1.setPorcentaje(57.35);
          estudiante1.imprime_detalles();
         }
         public Otra_Clase(String nombre_estudiante){
          nombre = nombre_estudiante;
         }

         public void setPorcentaje(double perc){
          percentil = perc;
```

```
         }

         public void imprime_detalles(){
          int Year = 2014;
          System.out.println("Año = "+Year);
          //Accede a otra variable de clase usando el nombre de la clase.
          System.out.println("Nombre Universidad =
"+Variable_Estatica.Nombre_Universidad);
          System.out.println("Facultad = "+Departmento);
          System.out.println("Nombre Estudiante = "+nombre);
          System.out.println("Percentil = "+percentil+"%");
          System.out.println("**********************");
         }
        }
```

12. Constructores

Formalmente un constructor es un bloque de código que es llamado y ejecutado en el momento de la creación del objeto, construyendo los valores(es decir los datos) para el objeto y es por esta tarea que se le conoce como constructor. Parece un método, pero debajo de las propiedades dadas del constructor se distinguen los métodos.

Reglas de creación para un constructor

1. El nombre de un constructor debe ser el mismo de su clase.
2. El constructor no devuelve nada.

Sencillo, ahora el código de ejemplo:

```
package beforeSelenium;

public class Estudiante {
    public static void main(String[] args) {
          //Dos objetos diferentes inicializados con valor.
         Estudiante stdn1 = new Estudiante("Brayatan");
         Estudiante stdn2 = new Estudiante("Deyanira");
   }

   //Constructor con parámetros pasan valor de objeto
   //Nombre del constructor es el mismo de la clase.
   public Estudiante(String nombre){
         String stdnname = nombre;
         System.out.println("Nombre estudiante =  "+stdnname);
   }

}
```

En el ejemplo anterior el constructor es llamado en la creación del objeto y realiza el paso de valores del objeto para imprimirlo.

Sobrecarga de constructores

Como ya lo mencionamos el nombre de un constructor debe ser el mismo de su clase, cuando creas más de un constructor con el mismo nombre, pero diferentes parámetros en la misma clase se le llama sobrecarga de constructor (constructor overloading). ¿En qué casos debería hacer una sobrecarga?, es útil cuando quieres construir un objeto de otra manera o podemos decir, usando diferentes parámetros. Obviamente tiene reglas de sobrecarga:

1. No se permiten dos constructores con los mismos argumentos.
2. Necesitas utilizar la palabra clave this() para llamar al constructor sobrecargado.
3. La primera instrucción para llamar a un constructor desde otro constructor sobrecargado debe ser la clave this().
4. Es una buena práctica llamar al constructor desde el constructor sobrecargado para que sea fácil de mantener a futuro.

```
package beforeSelenium;

public class Aprendo_Sobrecarga {
 String pnombre;
 String snombre;

 public static void main(String[] args) {
  Aprendo_Sobrecarga n1 = new Aprendo_Sobrecarga("Pedro");
  Aprendo_Sobrecarga n2 = new Aprendo_Sobrecarga("Juan", "Diego");
 }

 //Constructor con un argumento.
 public Aprendo_Sobrecarga(String nombre1){
  primerNombre = nombre1;//Variable local.
  System.out.println("1. Primer nombre = "+primerNombre);
 }

 //Sobrecarga de constructor con dos argumentos.
 public Aprendo_Sobrecarga(String nombre2, String nombre3){
  primerNombre = nombre2;
  segundoNombre = nombre3;
  System.out.println("2. Primer nombe = "+primerNombre);
  System.out.println("2. Segundo nombre = "+segundoNombre);
 }
}
```

Podemos hacer lo mismo usando la palabra reservada this():

```
package beforeSelenium;
```

```
public class Aprendo_Sobrecarga {
 String pnombre;
 String snombre;
 public static void main(String[] args) {
  Aprendo_Sobrecarga stdn2 = new Aprendo_Sobrecarga("Juan", "Diego");
 }

 //Constructor con un argumento.
 public Aprendo_Sobrecarga(String nombre1){
  pnombre = nombre1;
  System.out.println("Constructor. Primer nombre = "+pnombre);
 }

 //Sobrecarga constructor con dos argumentos.
 public Aprendo_Sobrecarga(String nombre2, String nombre3){
  //Llamada a primer constructor.
  this("Pedro");
  pnombre = nombre2;
  snombre = nombre3;
  System.out.println("Sobrecarga. Segundo nombre = "+pnombre);
  System.out.println("Sobrecarga. Tercer nombre = "+snombre);
 }
}
```

13. Herencia en Java

La herencia como concepto, padre-hijo, es muy utilizada en la orientación a objeto. Básicamente es la reutilización de código de clase 'padre'. Por ejemplo clase marraqueta[3] es clase hija de clase pan y se traduce en que la clase marraqueta podrá acceder y/o utilizar todas las propiedades no privadas de la clase pan (padre).

Clase Padre

```
package beforeSelenium;

//Clase padre
public class Auto {
        private String type="Vehiculo";
        public static int ruedas = 4;
        public String color = "azul";
        String gasolina = "Petróleo";

 public String getGasolina(){
  return gasolina;
```

[3] Marraqueta = pan francés = pan batido.

```
 }

 protected void Asientos(){
  int asientos = 4;
  System.out.println("Asientos = "+asientos);
 }
}
```

Clase hijo o subclase

```
package beforeSelenium;

//Clase hija
public class Mazda extends Auto{
          public int velocidad=150;

          public static void main(String[] args) {
           Mazda A = new Mazda();
           A.especificaciones();
           //Accede a una instanciad de variable de la clase padre usando
           //objeto por referencia de una clase hija dentro de un método
estático.
           System.out.println("Color = "+A.color);
           //Accede a método no estático de clase padre usando objetoo por
referencia
           // de clase hijo dentro de método estático.
           System.out.println("Gasolina = "+A.getGasolina());
          }

          public void especificaciones(){
           //Acede a variable de clase de clase padre directamente
           // dentro de clase hijo desde método no estático.
           System.out.println("Ruedas = "+ruedas);
           System.out.println("Velocidad = "+velocidad);
           //Accede a método no estático de la clase padre directamente,
           //dentro de método no estático de clase hijo.
           Asientos();
           //No puede acceder a variable privada de clase padre desde clase
hija.
           //System.out.println("¿Qué es un auto? = "+type);
          }
         }
```

Encapsilación(Overriding)

Comentado [IPS1]: Mejor explicado en https://www.arquitecturajava.com/java-override-y-encapsulacion/

En una subclase, cuando creas un método con la misma firma, devuelves los tipos y argumentos del método de la clase padre, entonces el método de esa subclase se conoce como un método reemplazado y se llama overriding en Java. Es útil para cambiar el comportamiento de un método de la clase padre. Un clásico ejemplo son los asientos de un auto, normalmente tienen cuatro asientos (método Asientos() = 4), pero supongamos que debemos tratar con un auto de seis asientos, en estos casos utilizaremos la encapsulación del método:

```
package beforeSelenium;

public class Subaru extends Auto{

          public static void main(String[] args) {
           Auto s = new Subaru();
           s.Asientos();
          }

          //Método asiento de clase padre es encapsulado.
          protected void Asientos(){
           int asientos = 6;
           System.out.println("Asientos = "+asientos);
          }
}
```

14. Interfaces

Después de entender herencia y encapsulación podemos comprender de mejor forma lo que es una interfaz. Por ejemplo Selenium WebDriver es una interfaz.

Usando una interfaz podemos definir las reglas de comportamiento detrás un de aplicativo. Se ve como una clase pero no lo es. Cuando implementas esa interfaz en cualquier clase, entonces todas esas reglas deben aplicarse a esa clase. Si implementas una interfaz en una clase entonces todos los métodos deben estar encapsulados. La interfaz creará una estructura de reglas a seguir para clases. Entonces, si miras el código de una interfaz te podrás hacer la idea acerca de la lógica de negocio. Cuando este diseñando una gran arquitectura, deberás usar una interfaz para definir la lógica de negocio en un nivel inicial.

Reglas para implementar una interfaz:

1. La interfaz no puede contener un constructor.
2. La interfaz no puede contener instancias de campos o variables.
3. La interfaz no puede contener métodos estáticos.
4. No se puede instanciar o crear un objeto en un interfaz.
5. Las variables dentro de una interfaz deben ser estáticas y es obligatorio inicializar la variable.
6. Cualquier clase puede implementar una interfaz, pero no pueden extender la interfaz.

7. Por defecto, todos los métodos y variables de la interfaz son públicos, por l oque no es necesario proporcionar modificadores de acceso.

Para que quede claro mostraré el siguiente ejemplo:

Crearemos una interfaz con el nombre College, crearemos tres clases con los nombres computer, mechanical y testcollege:

Universidad.java

```
package beforeSelenium;

//Les presento la interfaz
public interface Universidad {
	//Inicializo una variable estática.
	//Por defecto es estática.
	String Collegename = "Aplaplax";

	//Creo dos métodos estáticos sin cuerpo.
	void DetalleEstudiante();
	void ResultadoEstudiante();
}
```

Computacion.java

```
package beforeSelenium;

//Con la palabra 'implements' referencio la interfaz
public class Computacion implements Universidad {

	//@Override indica que el método está encapsulando el método de la
interfaz universidad.
	//Utilizo el mismo nombre de los métodos de la interfaz.
	//Retorna lo declarado en la interfaz.
	@Override
	public void DetalleEstudiante() {
		System.out.println("Estudiantes de Computación");
	}

	@Override
	public void ResultadoEstudiante() {
		System.out.println("Resultado estudiantes Computación");
	}
}
```

Mecanica.java

```
package beforeSelenium;

public class Mecanica implements Universidad{

@Override
public void DetalleEstudiante() {
         System.out.println("Estudiantes de mecánica.");
 }

@Override
public void ResultadoEstudiante() {
         System.out.println("Resultado estudiante Mecánica");
 }
}
```

ProbarInterfaz.java

```
package beforeSelenium;

public class ProbarInterfaz {

          public static void main(String[] args) {
                   //Podemos acceder a una variable de la interfaz
directactamente usando un nombre de interfaz
                   System.out.println(Universidad.Collegename+" detalles de los
estudiantes de la universidad.");

                   //Creamos el objeto computación con referencia a la
interfaz.
                   Universidad facultadComputacion = new Computacion();
                   //Métodos serán llamados des la clase Computación.
                   facultadComputacion.DetalleEstudiante();
                   facultadComputacion.ResultadoEstudiante();

                   //Creamos el objeto mecanica con referencia a la interfaz.
                   Universidad facultadMecanica = new Mecanica();
                   //Métodos llamados desde clase Mecánica.
                   facultadMecanica.StudentDetails();
                   facultadMecanica.StudentResult();
          }
}
```

15. ArrayList

Es una clase que implementa una lista de interfaces en Java y soporta arreglos dinámicos. Esto quiere decir que se puede trabajar con arreglos que pueden crecer según necesidad. Lo podemos utilizar para almacenar todos los links/enlaces de las páginas web, almacenar id de botones, etc.

Adivina, también es una interfaz: ArrayList Interface → List Interface → Collection interface

```
package beforeSelenium;
import java.util.ArrayList;

public class EjemploArrayList{
	public static void main(String[] args) {
		//Creamos el objeto.
		ArrayList<String> Ejemplo = new ArrayList<String>();

		Ejemplo.add("algo1"); //índice = 0.
		Ejemplo.add("algo2"); //índice = 1.
		Ejemplo.add("algo3"); //índice = 2.
		Ejemplo.add("algo4"); //índice = 3.

		for(int i=0; i<Ejemplo.size();i++){
			System.out.println(Ejemplo.get(i));
		}

		System.out.println("************************");

		//Obtener el índice de un 'item' desde el arraylist.
		int ItemIndex = Ejemplo.indexOf("button3");
		System.out.println("índice de algo3 = "+ItemIndex);
		System.out.println("************************");

		//Eliminar item desde arraylist.
		Ejemplo.remove(1);
		for(int i=0; i<Ejemplo.size();i++){
			System.out.println(Ejemplo.get(i));
		}
		System.out.println("************************");

		//Cambiar el valor de un item.
		Ejemplo.set(2, "algo8");

		for(int i=0; i<Ejemplo.size();i++){
			System.out.println(Ejemplo.get(i));
		}
	}
}
```

En el ejemplo se describe como crear un objeto arraylist, como añair valores, como indexarlo y recorrerlo, además de como eliminar valores y reiniciar un arraylist.

16. Hashtable

Es una clase en Java que provee una estructura para almacenar llaves (key) y sus valores en un formato de tabla. La llave no es un índice, se usa para mapear el valor por ejemplo:

```
package beforeSelenium;

import java.util.Hashtable;
public class Hash {
	public static void main(String[] args) {

		//Crea objeto para usar con diferentes propiedades.
		Hashtable<String, Integer> t1 = new Hashtable<String, Integer>();
		t1.put("Piernas", 4); //Almacena valor 4 con key = Piernas
		t1.put("Ojos",2);
		t1.put("Boca",1);

		//Accedemos a tabla hash usando keys.
		System.out.println("Piernas del animal = " +t1.get("Piernas"));
		System.out.println("Ojos del animal = " +t1.get("Ojos"));
		System.out.println("Boca del animal = " +t1.get("Boca"));
	}
}
```

En el ejemplo creamos un objeto Hashtable que almacena valores con lleve única. Usamos la clave para acceder e imprimir los valores.

17. Lectura-Escritura de Archivos de texto

Debemos utilizar la clase File para crear un archivo nuevo, FileWriter y BufferedWriter para escribir en un archivo, FileReader y BufferedReader para leer un archivo de texto:

```
package beforeSelenium;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class LecturaEscritura_Archivo {
```

```
public static void main(String[] args) throws IOException {

 //Crea archivo en disco duro D:
 String TestFile = "D:\\temp.txt";
 File FC = new File(TestFile);
 FC.createNewFile();

 //Escribe en archivo, tres líneas, la primera, espacio,
 // la segunda.
 FileWriter FW = new FileWriter(TestFile);
 BufferedWriter BW = new BufferedWriter(FW);
 BW.write("Primera Línea.");
 BW.newLine();
 BW.write("Segunda línea.");
 BW.close();

 //Leer archivo.
 FileReader FR = new FileReader(TestFile);
 BufferedReader BR = new BufferedReader(FR);
 String Content = "";

 //Ciclo para leer todas las líneas del archivo una
 //por una y las imprime por pantalla.
 while((Content = BR.readLine())!= null){
  System.out.println(Content);
 }
 }
}
```

18. Excepción de errores

Una excepción es un error generado durante la ejecución del código, en java existen dos tipos de errores:

1. Excepciones chequeadas: son todas esas excepciones que se chequean durante el tiempo de compilación y necesitan un bloque que los atrape (catch block) durante la compilación. Si el compilador no encontrara dicho bloque entonces entraría en un error de compilación.
2. Excepciones no chequeadas: son esas excepciones que no son controladas durante el tiempo de compilación.

Atrapar excepciones usando try-catch

Es un bloque que atrapará las excepciones y nos permite “humanizarlas”, en vez de que salga un mensaje raro, podemos indicar de que se trata, incluso escribir un log con dicho error.

```
public class Excepcion_ejemplo {
```

```
public static void main(String[] args) {
        int a[] = {3,1,6};
        // Si surge alguna excepción dentro de este bloque de prueba, el
control irá al bloque catch.
        try {
        System.out.println("Antes de Excepción.");
             //Excepción no verificada. Aquí existe una excepción por que
solo tenemos 3 valores en el arreglo.
             System.out.println(a[9]);
             System.out.println("Después de Excepción");
        }catch(Exception e){
                System.out.println("La excepción es "+e);
        }
       System.out.println("Fuera del try-catch.");
 }
}
```

Atrapar excepciones utilizando palabras clave

Otra forma de manejar la excepción es usar throws keyword con método como se muestra en el siguiente ejemplo. Supongamos que tiene un método throwexc que arroja alguna excepción y este método se llama desde algún otro método catchexc. Ahora quiere manejar la excepción del método de throwexc En método catchexc, entonces necesita usar la palabra clave throws con el método de throwexc.

```
public class Excepcion_ejemplo {
        public static void main(String[] args) {
                catchexc();
        }
        private static void catchexc() {
                try {
                        //Llamada al método throwexc().
                        throwexc();
                } catch (ArithmeticException e) {
                        System.out.println("Error división por 0.");
                }
        }
        //Excepción aritmética, división por 0.
        private static void throwexc() throws ArithmeticException {
                int i=15/0;
        }
}
```

Finalmente la palabra clave y su uso

Finalmente la palabra clave se usa con el bloque try catch al final del bloque try catch. El código escrito dentro del bloque finalmente se ejecutará siempre independientemente de la

excepción que haya ocurrido o no. La intención principal de usar finalmente con try catch block es: si quieres realizar alguna acción sin considerar la excepción ocurrida o no. finalmente el bloque se ejecutará en ambos casos:

```
public class Ejemplo_excepcion {

 public static void main(String[] args) {
  try{
    // Se lanzará una excepción.
    int i=5/0;
   //Esta declaración no se ejecutará
   System.out.println("Valor de i es = "+i);
  }catch (Exception e){
     //Muestra la excepción.
     System.out.println("Dentro del catch ="+e);
    //finalmente el bloque será ejecutado
  }finally {
   System.out.println("finalmente dentro.");
  }

  try{
   int j=5/2;
   System.out.println("El valor de J es = "+j);
  }catch (Exception e){
      System.out.println("Dentro del segundo catch."+e);
  }finally{
     System.out.println("Dentro del segundo try.");
  }
 }
}
```

En el ejemplo anterior, se usan 2 bloques try-catch-finally. Ejecute el ejemplo anterior y observe el resultado.

El primer bloque try arrojará un error y catch lo manejará y finalmente se ejecutará el bloqueo. El segundo bloque de prueba no arrojará ningún error, por lo que no se ejecutará la captura, pero finalmente se ejecutará el bloqueo. Entonces, en ambos casos, finalmente se ejecutará el bloque.

Por lo tanto, todo lo relacionado con la excepción y las formas de manejarlos lo ayudarán en el manejo de excepciones de prueba de su controlador de web.