Mario Hurtado López, Víctor Javier García Mascareñas.

Principios de construcción de software  7 de junio de 2017

Proyecto final: sistema de auto - acceso

Tabla de contenido

[2. Introducción. 2](#_Toc484587125)

[3. Estándar de codificación. 2](#_Toc484587126)

[4. Modelos de diseño. 5](#_Toc484587127)

[a. Modelos generales. 5](#_Toc484587128)

[i. Diagrama de casos de uso: 5](#_Toc484587129)

[ii. Diagrama de clases para el modelo de dominio: 6](#_Toc484587130)

[b. CU - Registrar actividad. 6](#_Toc484587131)

[i. Diagrama de actividades: 6](#_Toc484587132)

[ii. Diagrama de secuencia: 7](#_Toc484587133)

[iii. Diagrama de clases: 7](#_Toc484587134)

[c. CU – Administrar actividad del calendario. 8](#_Toc484587135)

[i. Diagrama de actividades: 8](#_Toc484587136)

[ii. Diagrama de secuencia: 8](#_Toc484587137)

[iii. Diagrama de clases: 10](#_Toc484587138)

[d. Prototipos. 10](#_Toc484587139)

[i. CU – Registrar actividad: 10](#_Toc484587140)

[ii. CU – Administrar actividad del calendario. 11](#_Toc484587141)

[5. Modelos de datos. 13](#_Toc484587142)

[e. Diagrama entidad relación: 13](#_Toc484587143)

[6. Reportes de pruebas unitarias. 15](#_Toc484587144)

[7. Referencias 19](#_Toc484587145)

1. Introducción.

La intención del presente documento es mostrar las bases, el diseño general del proyecto y particular para dos módulos del mismo. A sí mismo, se muestran las pruebas realizadas y las conclusiones de la realización.

Para el proyecto se escogieron los módulos de: Registrar actividad y Administrar actividad del calendario:

|  |  |
| --- | --- |
| **Módulo.** | **Descripción.** |
| Registrar actividad. | En éste módulo, un asesor puede registrar en el sistema que determinado usuario (alumno), realizo cierta actividad, con el motivo de tener un seguimiento de las actividades realizadas por el alumno. |
| Administrar actividad del calendario. | En éste módulo, el coordinador del centro de auto – acceso, administra (crea, edita y elimina) actividades para los usuarios (alumnos), mismas que serán registradas por un asesor. |

# Estándar de codificación.

A continuación se muestra la plantilla del estándar de codificación utilizado para la implementación del proyecto:

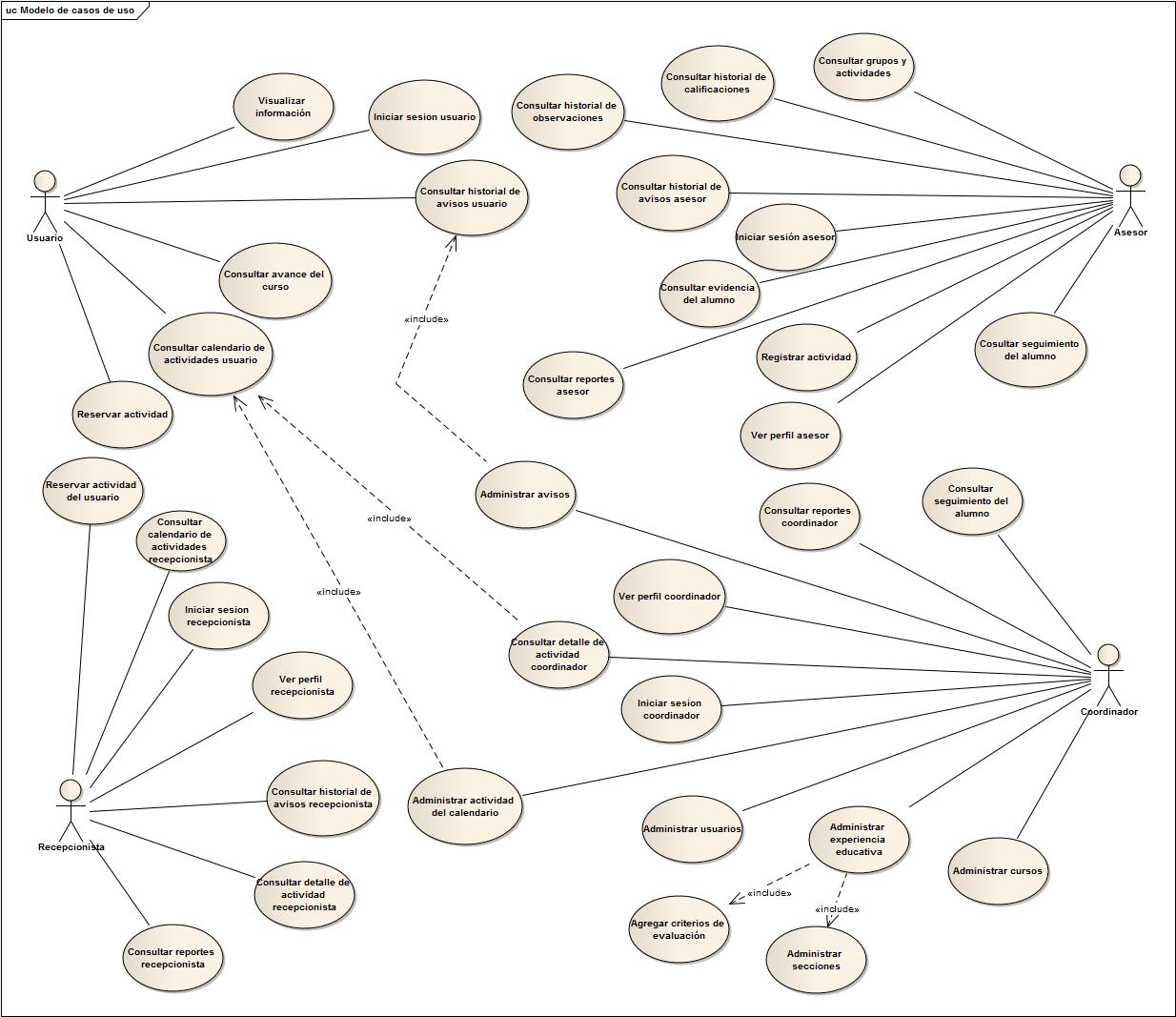
|  |  |
| --- | --- |
| **Propósito.** | Guiar el desarrollo de programas. |
| **Cabeceras de programa.** | Iniciar todos los programas con una cabecera descriptiva. |
| **Formato de cabecera.** | /\*\*  \*Nombre: “Nombre del programa.”  \*Desarrolladores: “Desarrollador 1. Desarrollador 2…”  \*Fecha: “Fecha de inicio de desarrollo.”  \*Descripción: “Descripción del programa. (Breve)”  \*/ |
| **Listar contenidos.** | Proporcionar un resumen del listado de contenidos. |
| **Ejemplo de contenidos.** | /\*\*  \*Resumen o lista de contenidos.  \*Instrucciones de uso:  \* 1…  \* 2…  \*Instrucciones de compilación:  \*Declaración de clases:  \*/ |
| **Instrucciones de reúso.** | * Describir como es usado el programa. Proporcionar el formato de declaración, valores, tipos y límites de parámetros. * Proporcionar precauciones de valores no válidos, condiciones overflow, u otras condiciones que potencialmente podrían resultar en operaciones impropias. |
| **Ejemplo de reúso.** | Límite de caracteres:  20 caracteres para variables y constantes.  50 caracteres para funciones. |
| **Identificadores.** | Usar nombres descriptivos para todas las variables, funciones, constantes y otros identificadores. Evitar abreviaciones o variables de una sola letra. |
| **Ejemplo de identificadores.** | int numeroDeEmpleados;  final int LARGO\_MAXIMO\_CADENA=20;  void separarCadena(string cadena); |
| **Comentarios.** | * Documentar el código para que entonces el lector pueda entender su operación. * Los comentarios deben representar el propósito y comportamiento del código. * Comentar las declaraciones de variables para indicar su propósito. |
| **Buen comentario.** | /\*\*  \* Regresa un objeto de CuentaValidada, con un código de  \* autenticación y un indice de cuenta, que en caso de ser válida es  \* mayor a 0.  \* @param usuario recibe el usuario o nickname de cuenta a  \* validar.  \* @param contrasena recibe la contraseña de cuenta a validar.  \* @return resultadoCuenta.  \*/  @Override  public CuentaValidada validarCuenta(String usuario, String contrasena) { |
| **Mal comentario.** | If (listaDePersonal[i].equals(nuevoEmpleado))//compara el registro. |
| **Secciones mayores.** | Preceder las secciones mayores del programa con un bloque de comentarios que describan el procedimiento realizado en la sección. |
| **Ejemplo.** | /\*\*  \*Esta sección del programa revisa nuevos nombres de emplea-  \*dos en el registro y actualiza posteriormente.  \*/ |
| **Espacios en blanco.** | * Escribir programas con suficiente espaciado para que no paresca atascado. * Separar cada parte del programa con al menos un espacio. |
| **Indentado.** | * Indentar cada nivel de su anterior. * Los niveles deben de estar alineados. |
| **Ejemplo de indentación.** | If (largoDeCadena>LARGO\_MAXIMO\_CADENA){  cout<<”El texto es muy largo.”;  }else{  codigoDeExito=”0X03”;  } |
| **Capitalización.** | * Capitalizar todos los defines y demás identificadores. * Los mensajes para el usuario deben dar una presentación limpia. |
| **Ejemplo de capitalización.** | final int NUMERO\_PERSONAL=8;//todas las constantes serán declaradas en este tipo de notación.  int numeroAux;  public void ayudaInt();//todo lo que no sea constante será en notación camello. |

# Modelos de diseño.

En esta sección se muestran los modelos de diseño generales del proyecto y los modelos particulares para los 2 módulos entregables.

## Modelos generales.

### Diagrama de casos de uso:



### Diagrama de clases para el modelo de dominio:



## CU - Registrar actividad.

### Diagrama de actividades:

### Diagrama de secuencia:



### Diagrama de clases:



## CU – Administrar actividad del calendario.

### Diagrama de actividades:



### Diagrama de secuencia:

* Agregar actividad:
* Editar actividad:



* Eliminar actividad.

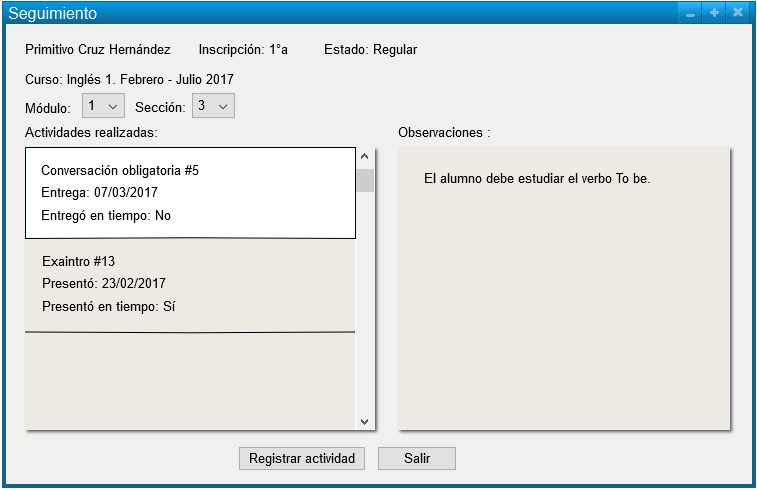
### Diagrama de clases:



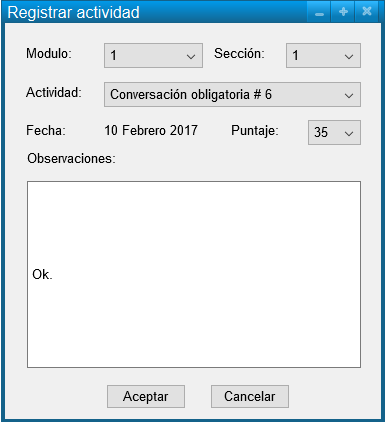
## Prototipos.

### CU – Registrar actividad:

* Ventana seguimiento:

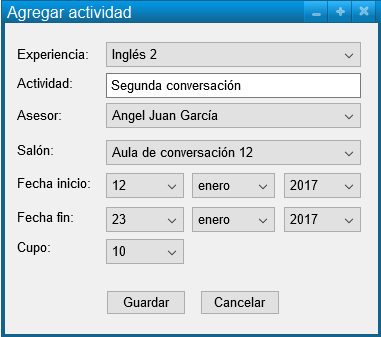


* Ventana registrar actividad:

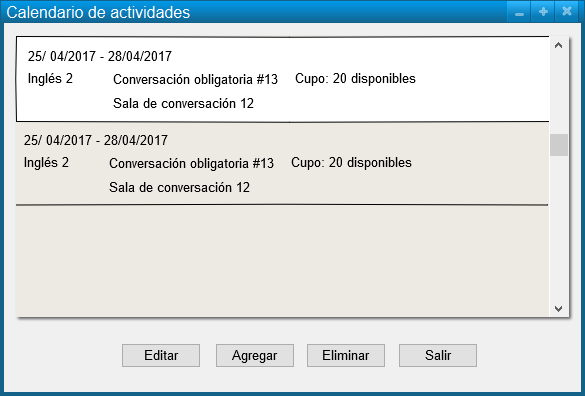


### CU – Administrar actividad del calendario.

* Ventana agregar/editar actividad:



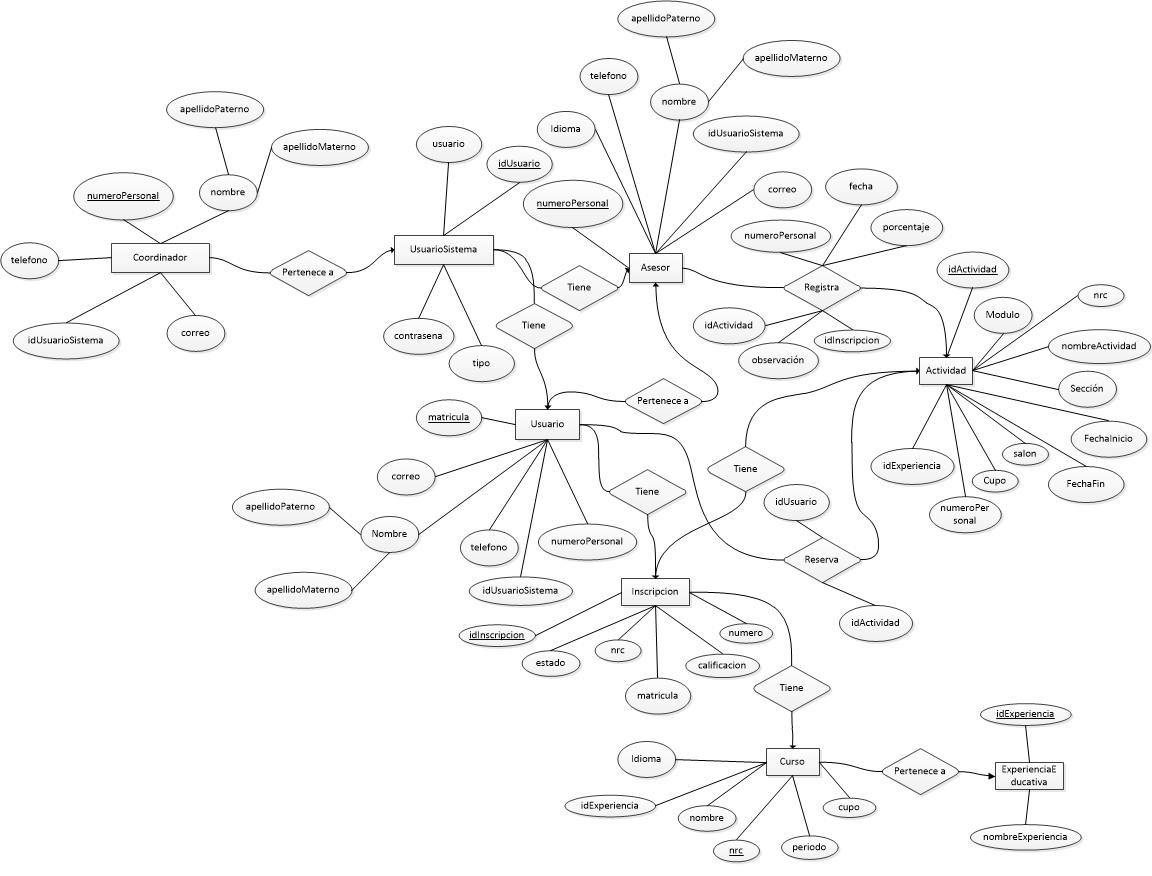
* Ventana mostrar actividad:



# Modelos de datos.

En éste apartado se muestran los diagramas: entidad – relación y modelo relacional, correspondientes al proyecto.

## Diagrama entidad relación:

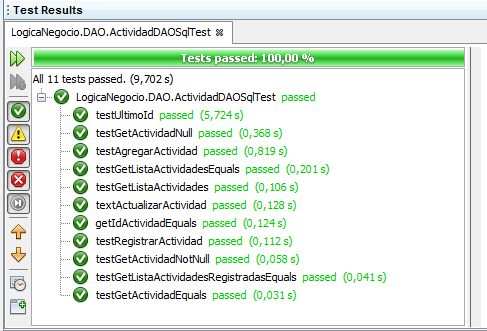


* 1. **Modelo relacional:**

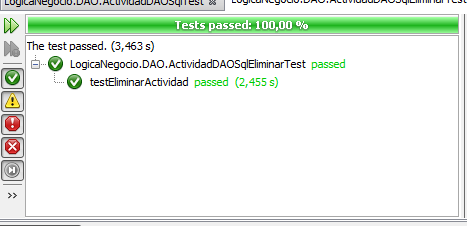
|  |
| --- |
| Asesor{numeroPersonal, idioma, idUsuarioSistema, nombre, apellidoMaterno, apellidoPaterno, correo, telefono} |
| Actividad{idActividad, nombreActividad, fechaInicio, fechaFin, cupo, seccion, modulo, numeroPersonal, idExperiencia, salon} |
| Usuario{matricula, nombre, apellidoPaterno, apellidoMaterno, numeroPersonal, idUsuarioSistema, correo, telefono} |
| registroActividad{porcentaje, fecha, observación, numeroPersonal, idActividad, idInscripcion} |
| Curso{nrc, nombre, periodo, cupo, idioma, idExperiencia} |
| Inscripcion{idInscripcion, estado, calificacion, numero, matricula, nrc} |
| ExperienciaEducativa{idExperiencia, nombreExperiencia} |
| Coordinador{numeroPersonal, nombre, apellidoMaterno, apellidoMaterno, correo, telefono, idUsuarioSistema} |
| UsuarioSistema{idUsuario, usuario, contrasena, tipo} |

## Reportes de pruebas unitarias.

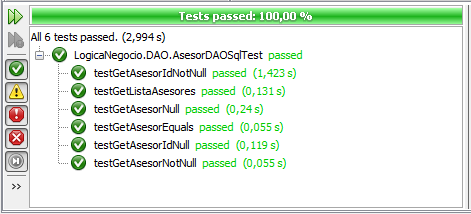
En este apartado se muestran todas las pruebas unitarias del proyecto referentes a las clases DAO que tienen una interacción con la parte de acceso a datos del sistema.



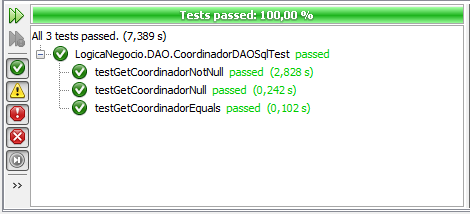
En esta prueba se muestran todos los resultados de los métodos aplicados a la clase ActividadDAOSql.



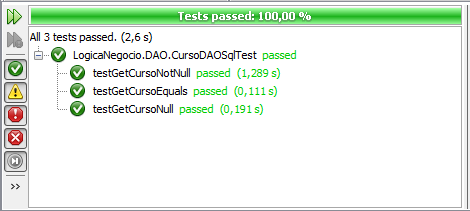
En esta prueba se muestra el método usado para verificar que una actividad puede ser eliminada. (método referente a la clase ActividadDAOSql), fue separado de la clase ActividadDAOSqlTest para un mejor flujo de las pruebas en ambas clases.



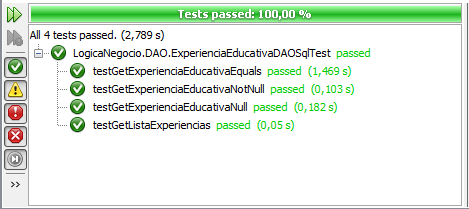
Prueba referente a la clase AsesorDAOSql , se muestran los resultado de aplicar las pruebas a la clase.



Pruebas referentes a la clase de DAO CoordinadorDAOSql, aquí se muestran los resultados de la aplicación de las pruebas a los métodos de la clase.



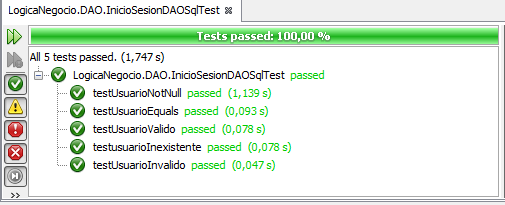
Pruebas referentes a la clase de DAO CursoDAOSql, aquí se muestran los resultados de la aplicación de las pruebas a los métodos de la clase.



Pruebas referentes a la clase de DAO ExperienciaEducativaDAOSql, en esta ilustración se muestran los métodos de prueba aplicados a la clase.



Esta ilustración muestra las pruebas aplicados a la clase ExperienciaDAOSql y a cada método de la clase con su respectiva prueba.



En esta ilustración se muestran las pruebas unitarias aplicadas a las a la clase de DAO inicio de sesionDAO aplicada a cada uno de los métodos de la clase 6. Conclusiones

Durante el desarrollo de este proyecto se utilizaron por primera vez diferentes tipos de técnicas para programación segura, un patrón de diseño y un estándar para su respectiva codificación. De igual forma, con el seguimiento del proyecto en su etapa de diseño por parte del profesor Octavio Ocharán Martínez, se obtuvo una mayor visión de la estrategia para la conclusión del mismo.

En este proyecto se establece una separación de clases con patrón de diseño Data Access-Object (DAO), gracias a este, durante la etapa de desarrollo se pudieron dividir por sus usos las entidades, la reutilización de clases en jerarquías y una menor dependencia entre estas.

El diseño del proyecto fue bastante substancial y complejo en su inicio de desarrollo, puesto que se presentaron dificultades con el documento de especificación de requerimientos y algunos datos no eran acordes a lo que el sistema necesitaba. Algunos casos de uso se añadieron a otros usuario que también los necesitaban se modelaron las interacciones de estos con el sistema.

# 7. Referencias

* Oracle(2015). Secure Coding Guidelines for Java SE. Oracle. Consultado el 4 de mayo de 2017. Recuperado de: http://www.oracle.com/technetwork/java/seccodeguide-139067.html#1
* Oracle(S/F). Java™ Platform, Standard Edition 7 API Specification.Oracle. Consultado el 23 de mayo de 2017.Recuperado de: https://docs.oracle.com/javase/7/docs/api/
* Oracle(1997). Java Code Conventions.Oracle. Consultado el 30 de mayo de 2017. recuperado de: http://www.oracle.com/technetwork/java/codeconventions-150003.pdf
* SparxSystems(S/F).Diagramas de actividades UML 2. SparxSystems.Consultado el 29 abril de 2017. Recuperado de: http://www.sparxsystems.com.ar/resources/tutorial/uml2\_activitydiagram.html
* Microsoft(S/F).Diagramas de secuencia UML.microsoft. Consultado el 29 de abril del 2017. Recuperado de https://msdn.microsoft.com/es-es/library/dd409377.aspx
* programacionOrientadaCom(S/F). interfaces, definición, implementacion de interfaces, herencia de interfaces.Consultado el 29 de abril del 2017. Recuperado de: https://sites.google.com/site/wwwprogramacionorientadacom/43-interfaces-definicin-implementacion-de-interfaces-herencia-de-interfaces
* SlideShare(2011).Manejo de excepciones.SlideShare. Consultado el 28 de mayo del 2017. Recuperado de: https://es.slideshare.net/exactlimon/5manejo-de-excepciones
* SlideShare(2011).JUnit con netbeans.SlideShare. Consultado el 23 de mayo del 2017. recuperado de: https://es.slideshare.net/bachispasaca/junit-con-netbeans
* utm(S/F).aserciones.utm.consultado el 10 de mayo del 2017. recuperado de: http://www.utm.mx/~mgarcia/CursoJava15-Aserciones.pdf
* Embgen, N.(2015).Defensive Programming.Informatik.Consultado el 21 de mayo del 2017. Recuperado de: http://www-ps.informatik.uni-tuebingen.de/teaching/ss15/sct/StudentMaterial/08%20-%20Nikolaus%20Embgen%20-%20Defensive%20Programming.pdf