Proyecto:

Sistema CADI

Versión:

1.1

Profesor:

Juan Carlos Pérez Arriaga

Integrantes:

Jesús Enrique Flores Nestozo

Fernando Manuel Guzmán Aja

Experiencia Educativa:

Principios de Construcción

Periodo:

Feb-Jul 2017

**Índice**

[INTRODUCCIÓN 2](#_Toc483228602)

[Estándar de Codificación 3](#_Toc483228603)

[MODELOS DE DISEÑO 8](#_Toc483228604)

[DIAGRAMA DE CASOS DE USO 8](#_Toc483228605)

[DIAGRAMA DE CLASES 9](#_Toc483228606)

[DIAGRAMAS DE ACTIVIDAD 10](#_Toc483228607)

[MODELO DE DATOS 12](#_Toc483228608)

[DIAGRAMA DE ENTIDAD RELACION 12](#_Toc483228609)

[PROTOTIPOS 13](#_Toc483228610)

[PROTOTIPO DE CU RESERVAR ACTIVIDAD 13](#_Toc483228611)

[PROTOTIPO DE CU EDITAR ESTUDIANTE 15](#_Toc483228612)

[PROPUESTAS DE CAPA 17](#_Toc483228613)

[CONCLUSIONES 18](#_Toc483228614)

[REFERENCIAS 19](#_Toc483228615)

# INTRODUCCIÓN

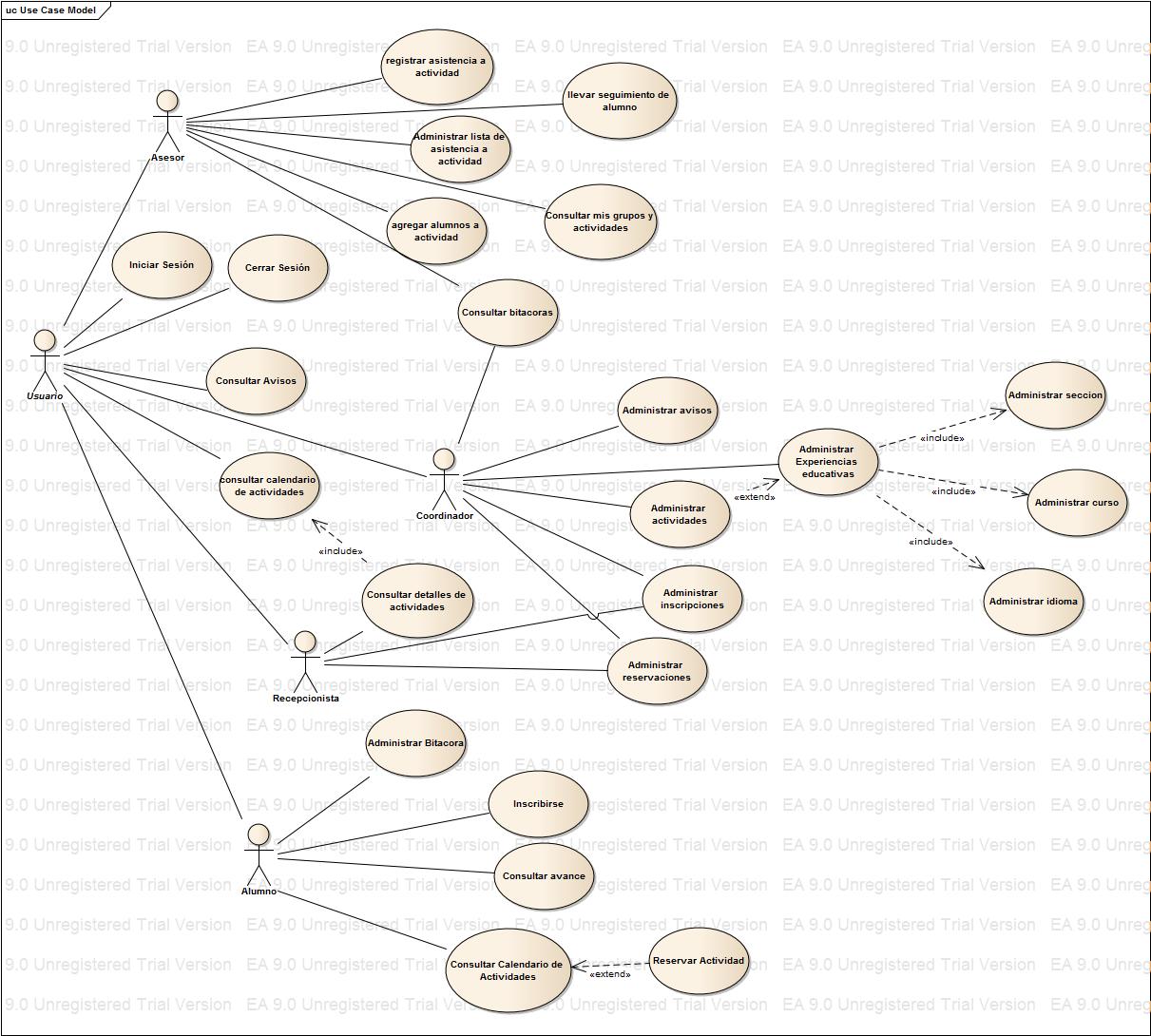
# Estándar de Codificación

|  |  |
| --- | --- |
| Propósito | Definir un estándar de las técnicas de programación para la experiencia educativa de Principios de construcción de software para el lenguaje JAVA |
| Cabecera del programa | Todas las páginas de código deben incluir un encabezado descriptivo del software. |
| Formato de Cabecera | /\*  \*  \*Nombre del proyecto:  \*  \*Nombre de la paquetería:  \*  \*Importaciones:  \*  \*Declaración de la clase:  \*Descripción de la clase:  \*  \*Nombre de desarrollador:  \*  \*Fecha de creación:  \*Fecha de última actualización:  \*  \*/ |
| Contenido del listado | Proporciona un resumen del contenido del listado. |
| Instrucciones de reutilización | * Describe como es usado el programa. Proporciona el formato de declaración, valores de parámetros y tipos, y límites de parámetros. * Proporciona advertencias de valores ilegales, desbordamiento de condiciones, u otras condiciones que podrían potencialmente resultar en operaciones impropias. |
| Identificadores | 1.\_Convenciones para Variables  Para la definición de variables se debe tener presente las siguientes consideraciones:  ■Los nombres de las variables deben ser cortos y significativos.  ■La elección de un nombre de variable debe ser mnemotécnica, esto es, diseñado para demostrar el propósito de su uso a cualquier observador.  ■Los nombres de variables de un solo carácter deben ser evitados, solo usarlo para variables temporales, como i, j o k para estructuras cíclicas  2.\_Convenciones para Constantes  Para la definición de constantes debe tener presente las siguientes consideraciones:  ■Los nombres de variables declaradas como constante deben ser todas en mayúsculas con palabras separadas por guion abajo (“\_”). |
|  | 3.\_Convenciones para Clases  Para la definición de clases debe tener en cuenta las siguientes reglas:  ■Tratar de que el nombre de la clase sea simple y descriptivo  ■Use palabras completas, evite abreviaturas o acrónimos, a menos que sean mucho más usadas que el nombre completo.  ■Los nombres de las clases deben ser sustantivos.  ■Los nombres de clase deben empezar con una letra mayúscula y el resto de letras deben estar escritas en minúscula. En el caso que pueda tener más de una palabra, las primeras letras de cada palabra interna deben ser con mayúscula.  ■Los nombres de clase no pueden contener espacios ni caracteres especiales, sólo son permitidas las letras de la “a” a la “z” y los números del 0 al 9  4.\_Convenciones para Atributos de Clases  Para definir los nombres de los atributos deberá tener en cuenta las siguientes reglas:  ■ Los nombres de atributos deben empezar con una letra minúscula y el resto de letras deben estar escritas en minúscula.  ■ Los nombres de atributo no pueden ser verbos.  ■ Los nombres de atributo no pueden contener espacios ni caracteres especiales, sólo son permitidas las letras de la “a” a la “z” y los números del 0 al 9.  ■ Si el nombre de atributo está compuesto por más de una palabra, cada palabra adicional debe empezar con mayúscula  5.\_Convenciones para Métodos  Para definir los nombres de los métodos deberá tener presente las siguientes reglas:  ■ Los nombres de los métodos deben empezar con una letra minúscula y el resto  de letras deben estar escritas en minúscula.  ■Los nombres de los métodos deben ser verbos o palabras que identifiquen de manera general el objetivo del método  ■Los nombres de los métodos no pueden contener espacios ni caracteres especiales, sólo son permitidas las letras de la “a” a la “z” y los números del 0 al 9.  ■Si el nombre de método requiere estar compuesto por más de una palabra, cada palabra adicional debe empezar con mayúscula. |
| Ejemplos de identificadores | 1.\_Convenciones para Variables  Int díasTranscurridos  Float pesoUsuario  2.\_Convenciones para Constantes  public static final DIAS\_SEMANA= 7;  public static final MESES\_AÑO=12;  3.\_Convenciones para Clases  Transporte  Persona  4.\_Convenciones para Atributos de Clases  strTipoTransporte  fltAlturaPersona  5.\_Convenciones para Métodos  runFast();  getBackground(); |
| Comentarios | * Todas las funciones tienen que estar documentadas explicando que realiza cada uno de ellas. Para ello se utilizará el estandar de documentación: * Nombre de función: * Parametro1: * Parametro2: * Descripción de la función: * Valor de retorno: |
| Buen comentario | \*/  Nombre de función: CalcularAreaTriangulo  Parametro1: fltBase, es el valor de la base del triángulo  Parametro2: fltAltura, es el valor de la altura del triángulo  Descripción de la función: calcula el area de un triángulo.  Valor de retorno:Resultado=(fltBAse\*fltAltura)/2  \*/ |
| Mal comentario | Public float CalcularAreaTriangulo(fltBase,fltAltura){  //Esta función calcula el area de un triangulo  //fltBase es el valor de la base del triángulo  //fltAltura es el valor de la altura del triángulo  Return (fltBAse\*fltAltura)/2;  } |
| Secciones principales | Antes de cada sección principal de código se comentará una breve descripción de la función de la sección. |
| Ejemplo | /\*  La siguiente sección de bloque aplicando los principios de recursividad recorrerá un árbol binario para encontrar el dato solicitado.  \*/ |
| Espacios en blanco | * Después de cada sección de código y entre funciones se dejará 2 líneas de espacio en blanco para permitir una mejor lectura. |
| Indentación | * Todo código desarrollado tendrá una Indentación de 4 espacios en blanco equivalente a un tabulador, aplicándose a cada nivel de profundidad. |
| Ejemplo de Indentación | For(i=0; i<100; i++){  if(strListaAlumno[i] == “perez”){  system.out.println(strListaAlumno[i];  }  } |
| Capitalización | Uso de mayúsculas:  Nombres de Constantes (todo el nombre)  Nombres de clases (Primera letra)  En caso de que se necesite ampliar un nombre para dejar más claro su función se usara una mayúscula para separar palabras  Uso de minúsculas:  Nombre de atributos  Prefijos de variables  Nombre de métodos  Uso de comentarios  Palabras reservadas |
| Ejemplo de capitalización | Uso de mayúsculas:  DIAS\_SEMANA;  Persona  calcularAreaTriangulo  Uso de minúsculas:  peso  int  correr()  system.println |

# MODELOS DE DISEÑO

## DIAGRAMA DE CASOS DE USO

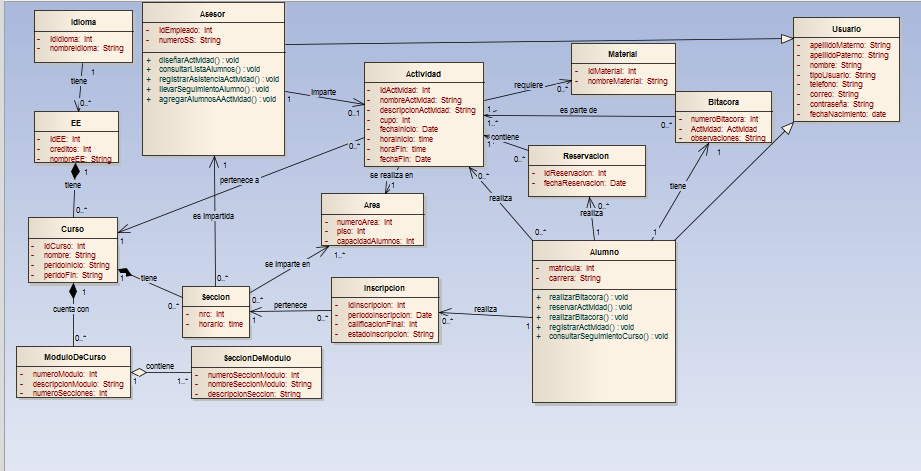
En este diagrama podemos apreciar los requerimientos funcionales del sistema, agrupado por los diferentes actores que interactúan en este sistema.



## DIAGRAMA DE CLASES

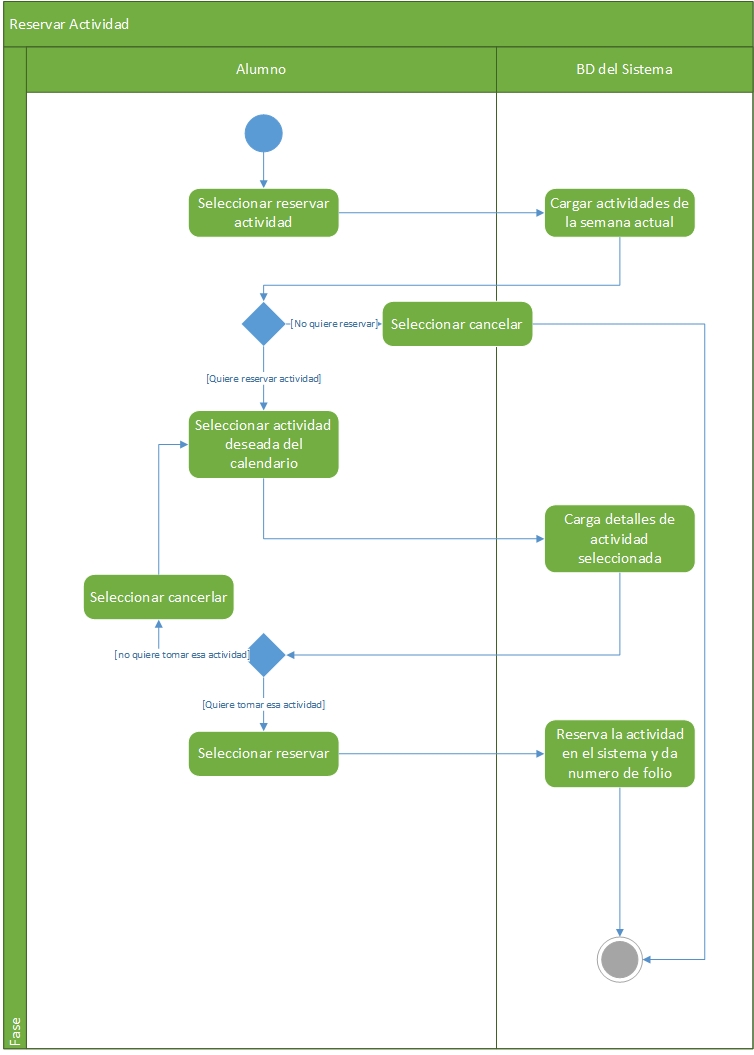
En este diagrama podemos apreciar las diferentes clases que intervienen dentro del sistema, así como sus diferentes atributos y métodos correspondientes.

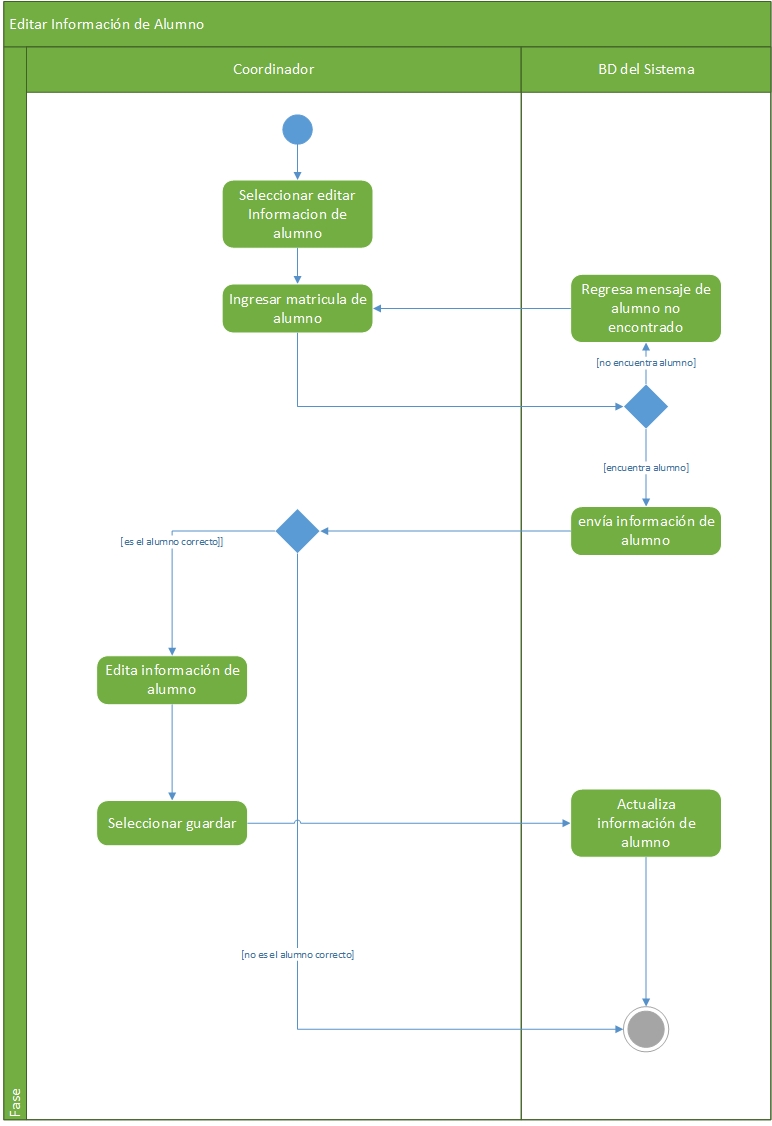
También se observa la relación entre estos, su cardinalidad y multiplicidad.



## DIAGRAMAS DE ACTIVIDAD

En estos diagramas se muestran las interacciones secuenciales del usuario con el sistema, separadas por procesos. Únicamente se muestran los diagramas que fueron seleccionados para implementar en el proyecto.

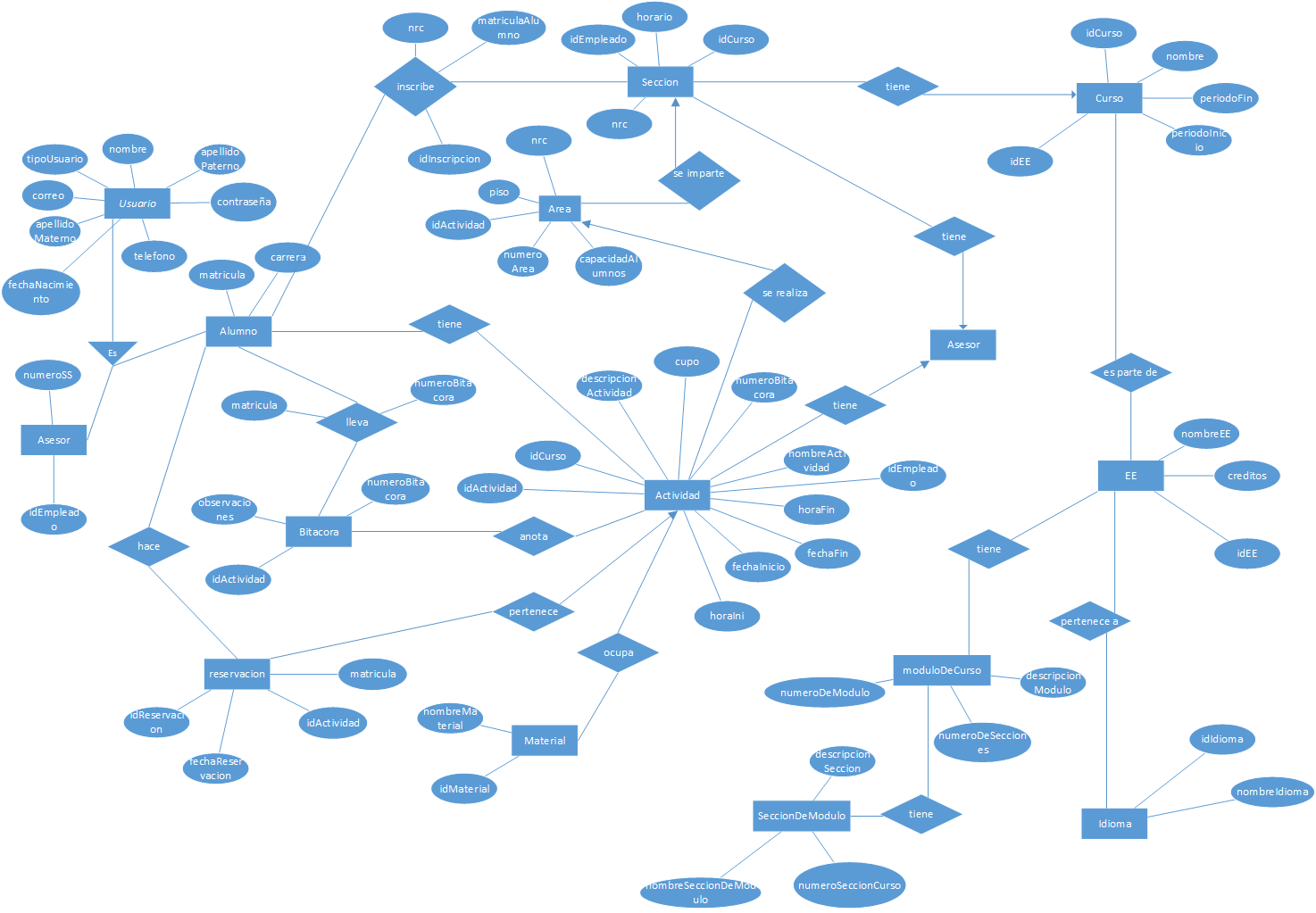




# MODELO DE DATOS

## DIAGRAMA DE ENTIDAD RELACION

En este diagrama se aprecia la relación que existe entre las diferentes tablas registradas en la base de datos, la cual nos ayudara a almacenar la información generada por el sistema (PDF anexado para mejor visualización).

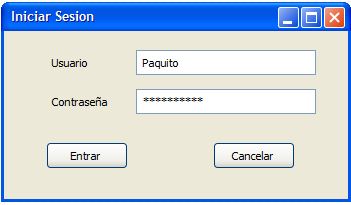


# PROTOTIPOS

## PROTOTIPO DE CU RESERVAR ACTIVIDAD

Las siguientes imágenes muestran el diseño de las interfaces del caso de uso: reservar actividad, de modo secuencial desde que inicia sesión el usuario.

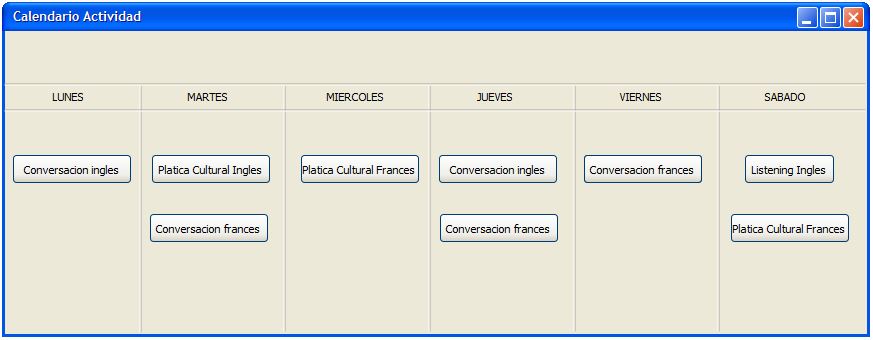
Inicio de sesión del usuario



Menú del usuario alumno:



Reservar actividad:



Seleccionar Actividad:



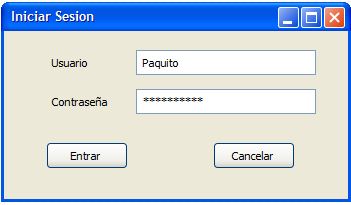
Generación folio de reservación:



## PROTOTIPO DE CU EDITAR ESTUDIANTE

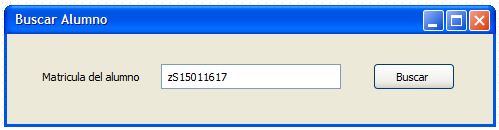
Las siguientes imágenes muestran el diseño de las interfaces del caso de uso: editar estudiante, de modo secuencial desde que inicia sesión el usuario.

Inicio de sesión del usuario:

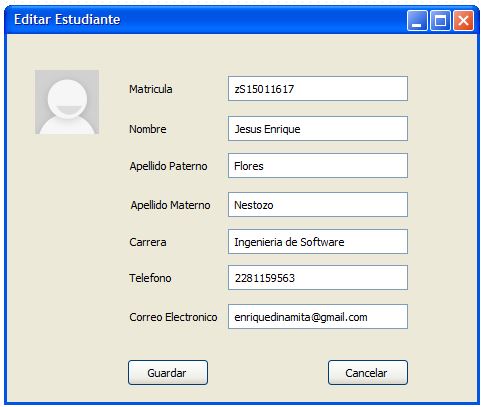


Menú del usuario coordinador:



Administrar usuario:

Editar Alumno:



# PROPUESTAS DE CAPA

Se manejara un patrón DAO de tres capas para la implementación del código. Las capas que se ocuparan son las siguientes:

Business Object

Business Object permite separar la lógica de negocio del resto de las capas (interfaz y de acceso a datos). Los objetos de negocio organizan el código de negocio en unidades lógicas que facilitan la mantenibilidad del sistema.

Data Access

Los componentes de acceso a datos encapsulan la tecnología empleada para acceder a la capa de datos, separando completamente la lógica de negocio de la lógica de acceso a datos. De esta forma se garantiza la portabilidad del sistema en los diferentes motores de base de datos (Oracle, SqlServer, etc).

Graphical User Interface

Esto representa una la parte grafica del sistema. Permite mapear los Business Object para darle al usuario la mayor experiencia del sistema.

# CONCLUSIONES

Para la realización del presente documento fue necesario entender el problema al que nos estamos enfrentando, por lo que se hicieron consulta de fuentes electrónicas, así como una visita a las instalaciones del centro de autoacceso, donde pudimos conversar con diferentes personas involucradas en los procesos que se lleven a cabo internamente.

Al hacer toda la investigación pertinente sobre el funcionamiento del centro de autoacceso nos dimos cuenta de la importancia de realizar un software correctamente; sobre todo bien diseñado, pues es aquí donde el diseño de software toma mucha importancia ya que al diseñarlo e irlo llevando por etapas nos permite corregir errores a tiempo con lo cual ahorramos tiempo a la hora de la implementación.

Leer sobre cómo se debe documentar el estándar de codificación también nos ha servido mucho ya que eso nos va a facilitar a la hora de realizar el sistema y que al haber muchas clases y atributos no nos enredemos y podamos llevar un estándar y un orden en el nombramiento de variables, clases, métodos, etc. Además que al usar el patrón de diseño DAO nuestro código tendrá una mejor organización.

# REFERENCIAS

Referencias consultadas para un mejor entendimiento del problema:

https://www.uv.mx/portalcadi/

https://www.uv.mx/portalcadi/formatos/

Referencias consultadas para la realización de los diagramas del sistema:

Ambler, S. (2004). The object primer : agile modeling-driven development with UML 2.0. New York: Cambridge University Press.

Stevens, P. & Pooley, R. (2007). Utilización de UML en ingeniería del software con objetos y componentes. Madrid: Pearson Addison Wesley.

Budgen, D. (2003). Software design. Harlow, England New York: Addison-Wesley.

Referencias consultadas para la realización del estándar de codificación:

https://google.github.io/styleguide/javaguide.html

http://www.oracle.com/technetwork/articles/java/index-137868.html

Referencias consultadas para el desarrollo de la base de datos:

http://cursos.aiu.edu/Base%20de%20Datos/pdf/Tema%202.pdf

http://www.uoc.edu/masters/oficiales/img/913.pdf