

Documento de entrega final

Entrega de documento final



8 de junio de 2016

Universidad VEracruzana

Ingeniería de Software

Índice

[Introducción 2](#_Toc453115831)

[Desarollo 3](#_Toc453115832)

[Análisis 4](#_Toc453115833)

[Requerimientos 5](#_Toc453115834)

[Diseño 6](#_Toc453115835)

[Base de datos 6](#_Toc453115836)

[Modelo de Clases 7](#_Toc453115837)

[Entidad Relación 8](#_Toc453115838)

[Prototipos 9](#_Toc453115839)

[Estándar 14](#_Toc453115840)

[Conclusiones 18](#_Toc453115841)

[Referencias 19](#_Toc453115842)

# Introducción

Este documento tiene como propósito complementar y demostrar la documentación del trabajo realizado, así como también justificar el trabajo por medio de los diagramas presentados en la materia de *Principios de diseño de Software* este documento tiene como fin dar a conocer al usuario el trabajo realizado este último semestre.

Se da al usuario este documento, para que sea tomado como referencia de lo trabajado en el proyecto, ya que en este se dan detalles de la implementación del software, se dará a conocer las soluciones al problema presentado, y se resolverán dudas comunes que podrán surgir, en complemento a este documento, también se tiene la documentación generada por el proyecto de java.

# Desarollo

Se darán a conocer el detalle de los procesos y funciones realizadas durante la realización del proyecto, así como la justificación de los diseños del por qué las decisiones tomadas en este último paso, se tomaron las decisiones de que el lenguaje de programación seria Java con el entorno de desarrollo de NetBeans

Por parte de los diagramas, estos fueron construidos con de la herramienta CASE: *Enterprise Architect* y con ayuda y colaboración del Dr. Jorge Octavio Ocharán Hernández estos diagramas fueron previamente revisados y autorizados por el mismo.

Se dan justificaciones en el documento del porqué de algunas cosas, por ejemplo, las interfaces de los módulos, incluso supuestos como, las funcionalidades del sistema que no están implementadas estas paran a versiones posteriores para poder desarrollar el software por completo.

## Análisis

Por medio de la elicitación de requerimientos determinamos que se necesitaba un sistema que pudiera hacer los siguientes procesos:



**Ilustración 1 Modelo de funcionalidad por paquete**

Ordenado por funcionalidad y tomando en cuenta que cada uno cubriera una necesidad del cliente

## Requerimientos

Requerimientos mínimos:

Una computadora de arquitectura de 32bits.

Java 8 cualquier versión.

PhPMyAdmin: 4.4.14.

MySQL Community Server 5.6.26.

Apache versión 2.4.16.

Memoria RAM de 1 GB.

Teclado.

Mouse.

Conexión a internet.

Requisitos Recomendados:

Una computadora de arquitectura de 32bits.

Java 8 actualización 91.

PhPMyAdmin: 4.6.2.

MySQL Community Server 5.6.26.

Apache versión 2.4.16.

Memoria RAM de 2 GB.

Teclado.

Mouse.

Lector de códigos de barras.

Conexión a internet.

Se necesita contar con un servidor dedicado en red o en el mismo pc, con base de datos previamente creada.

## Diseño

En esta parte se pondrán los diseños más representativos, creados previamente en la materia de *Principios de diseño de software*, estos fueron la base para la elaboración del proyecto, así mismo la implementación está reflejada en el sistema trabajado.

### Base de datos



**Ilustración 2 Modelo de base de datos**

### Modelo de Clases



**Ilustración 3 Modelo de clases**

### Entidad Relación

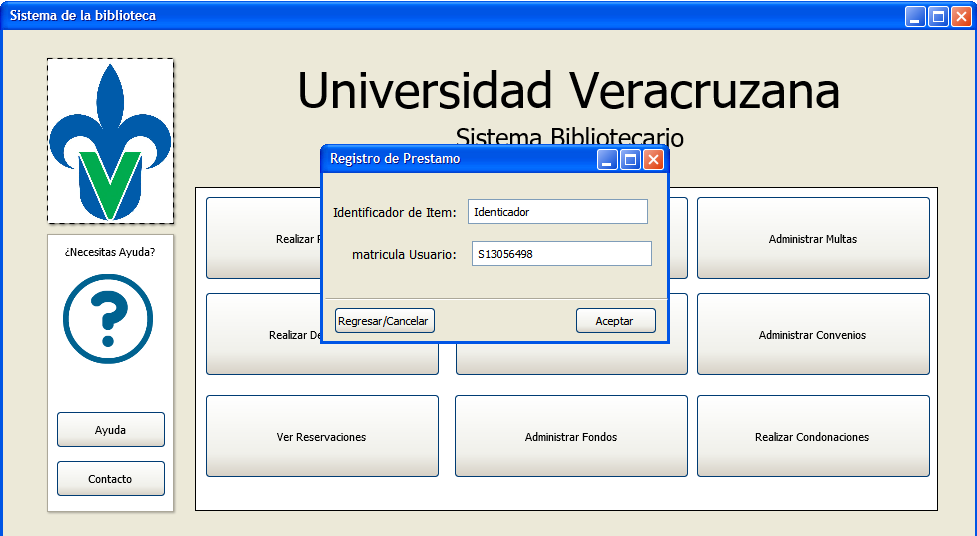


**Ilustración 4 Modelo entidad - relación**

### Prototipos

Prototipos generados con la herramienta pencil, fueron detallados a un mediano nivel según el grado de requerimientos

#### Modulo Préstamo



**Ilustración 5 Prototipo realizar préstamo**



**Ilustración 6 Prototipo préstamo correcto**



**Ilustración 7 Prototipo préstamo con error a base de datos**

#### Modulo Reservación



**Ilustración 8 Prototipo reservar ítem**



**Ilustración 9 Prototipo reservar ítem correcto**

## Estándar

|  |  |
| --- | --- |
| Cabecera de programa | La cabecera contendrá el nombre del proyecto, autores, fecha de la última edición y una descripción corta del objetivo de la clase |
| Formato de Cabecera | /\*\*  \* Clase tipo biblioteca, a través de ésta clase se pueden realizar  \* préstamos, reservaciones, búsqueda de ítems y sus correspondientes  \* validaciones.  \*  \* @author Luis Fernando Gomez Alejandre  \* @author Francisco Gerardo Mares Solano  \* @since 06/06/2016  \*/ |
| Tabla de contenido | /\*\*  \* **Descripcion del método**  \*  \* @param parametro1  \* @param parametro2  \* @param parametro3  \* @param parametroN…  \* @return Variable de retorno  \* @throws Si lanza alguna excepción  \*/ |
| Tabla de contenido  Ejemplo | /\*\*  \* **Busca cualquier ítem que esté registrado en la base de datos con el**  \* **identificador pasado.**  \*  \* @param identificador se realiza una validación de 15 caracteres y una  \* letra inicial 'I' mayúscula o minúscula.  \* @return Si el parámetro fue pasado de manera erronea o no existe en  \* la base de datos, regresa una lista vacia.  \* Nunca regresará una lista nula y siempre será un ArrayList  \* @throws SQLException Si no existe conexión con la base de datos o algo  \* sale mal durante la conexión con la misma,  \* se lanzará una SQLException con el correspondiente mensaje de error.  \*/ |
| Identificadores | Variables:  Se ocupara Lower Camel Case, dónde la primera letra del identificador será minúscula, a partir de ahí, las palabras se escribirán seguidas y la primera letra de cada palabra será mayúscula. En cada caso, serán nombre distintivos y se evitarán caracteres especiales excepto para bucles for o while, dónde podrán ser iterador i,j,k… en ese orden.  Constantes:  Se ocupará una escritura totalmente en mayúsculas, dónde la separación entre palabras será constituido por un guion bajo ( \_ ) y de igual manera, serán nombres descriptivos y palabras completas siempre declaradas como *public static final*.  Métodos:  Se ocupara Lower Camel Case, dónde la primera letra del identificador será minúscula, a partir de ahí, las palabras se escribirán seguidas y la primera letra de cada palabra será mayúscula, Si este supera los 7 argumentos se debera pasar por un objeto que haga referencia, a los atributos de ese objeto |
| Ejemplos de identificadores | private ArrayList<String> telefonos;  private ArrayList<String> redesSocial;  private ArrayList<String> correosElectronicos;  public static final int RESULTADO\_DE\_SALIDA = 0; |
| Comentarios | Los comentarios de una sola línea serán precedidos por dos diagonales (//) y no ocuparán más de un renglón. Su objetivo será ayudar a entender el motivo de un código de programación.  Los comentarios de mayor extensión serán escritos con diagonales y asteriscos para su delimitación (/\*\* … \*/) y en cada renglón intermedio habrá un asterisco (\*) como marcador de inicio y tendrá un espacio de identación incluyendo el último renglón.  Para ambos casos su identación será la misma que tenga el método, sentencia o bucle al que preceda. Siempre deben agregar información de utilidad y que extienden la información común. En el caso de *edit fold* son permitidos dos renglones con doble diagonal (//). |
| Buen comentario | //Busca contacto(s) con las funciones lambda contactosEncontrados=this.listaContactos  .stream()  .filter(contacto->contacto.getNombreContacto()  .equals(nombre))  .collect(Collectors.toList()); |
| Mal comentario | // Variable para Contador  // Es para contar números.  // Es privado para que solo la clase acceda  // Entero por que no necesito flotantes  Private int contadorNumeros; |
| Secciones principales | /\*\*  \* Descripción  \*  \* @author Nombre1  \* @author Nombre2  \* @since Fecha  \*/ |
| Ejemplo | /\*\*  \* **Clase tipo biblioteca, a través de ésta clase se pueden realizar préstamos,**  \* **reservaciones, búsqueda de ítems y sus correspondientes validaciones.**  \*  \* @author Luis Fernando Gomez Alejandre  \* @author Francisco Gerardo Mares Solano  \* @since 06/06/2016  \*/ |
| Espacios en blanco | * Dentro de las clases, después de la declaración de variables y constantes, se dejará un espacio en blanco para seguir con los métodos. * Entre cada método de la clase debe quedar un espacio en blanco. * Los paquetes importados son consecutivos y en el primer renglón, y después de ellos, debe haber un espacio en blanco. * Los *import* deben ser consecutivos y después de ellos debe ir un espacio en blanco, van después de la declaración del paquete. * Los comentarios nunca tendrán un espacio en blanco hacia abajo. * Cada operador lógico debera de ir separado ente espacios (“ = ”, “ + ”). * Todos los parámetros llevarán un espacio después de la coma para agregar el siguiente. |
| Identación | * 4 espacios para la identación, cada anidación cumple la misma norma. * Las llaves inician junto al nombre de la función y terminan en un renglón extra al nivel del nombre de la función o método. * Solo los comentarios con múltiples comentarios pueden tener un espacio como identación. * En los comentarios despues de la descripción debera tener un espacio en blanco y despues seran las declaraciones de parametros y valores de retorno |
| Ejemplo de Identación | public class Contacto{  private int identificador;  private ArrayList<String> correosElectronicos;  private String paginaWeb;  public String getNombreContacto() {  return nombreContacto;  }  } |
| Capitalización | * Variables y métodos: Ocuparán Lower Camel Case. * Constantes: Completamente en mayúsculas * Clases: Ocuparán Upper Camel Case. * Paquetes: Siempre en minúsculas. |
| Ejemplo de capitalización | package dataaccess;  public class ItemDAOImpl implements ItemDAO{  private Statement consulta;  private ResultSet resultados;  } |

# Conclusiones

En conclusión, podemos decir que el proyecto esta implementado al 25% pero es posible la escalabilidad del mismo, pero por razones de evaluación y tiempo no es posible la implementación de la demás funcionalidad, pero por medio de los diseños modelados en la materia de *Principios de diseño* sería posible terminar el sistema, y este estaría seria fiel a los diseños.

En conclusión, por el proyecto tener las pruebas de unidad, es más fácil hacer funcional el proyecto sin tener que implementar toda la funcionalidad,

# Referencias

Albin, S. T. (2003). *The Art of Software Architecture Design Methods Techniques.* Wiley.

Deitel & Associates, Inc. (2008). *Cómo programar en java.* Estado de México: Pearson Educación.

Long, F., Mohindra, D., Seacord, R. C., Sutherland, D. F., & Svoboda, D. (2014). *Java Coding Guidelines.* United States of America: Addison-Wesley.

McConnell, S. (2004). *Complete Code.* Redmond: Microsoft Press.

Oracle. (08 de 06 de 2016). *Oracle*. Obtenido de http://docs.oracle.com/javase/7

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2002). *Fundamentos de bases de datos.* Aravaca: McGRAW-HILL.

Wiegers, K., & Beatty, J. (2013). *Software Requeriments.* Redmond: Microsoft Press.