



# PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

## FACULTAD DE CIENCIAS E INGENIERÍA



### INTÉRPRETE Y ENTORNO DE DESARROLLO APLICADOS AL AUTO-APRENDIZAJE DE LOS CONCEPTOS DE PROGRAMACIÓN ORIENTADA A OBJETOS

Tesis para optar por el Título de Ingeniero Informático, que presenta el bachiller:

**Ever Ricardo Mitta Flores**

**Código: 20070257**

**ASESOR: Layla Hirsh**

Lima, Noviembre del 2012

## RESUMEN DEL PROYECTO DE TESIS

Con el surgimiento de los lenguajes de programación y el gran interés que estos atraen, cada vez hay más personas que deciden sumergirse a este mundo.

Otro punto a tomar en cuenta es que no solo hay un estilo de programación sino que la programación puede estar sujeta a diversos paradigmas, siendo estos el paradigma estructurado, el paradigma orientado a objetos, el paradigma orientado a eventos, entre otros.

Si bien los diversos paradigmas se relacionan entre ellos, es decir tienen características afines; en ocasiones, no es fácil para las personas dar un salto de un paradigma a otro. Otra dificultad existente suele ser que existen diversos lenguajes de programación orientados a un mismo paradigma lo que genera usualmente confusión en la definición de conceptos propios de dicho paradigma, a causa de las diferentes sintaxis y alcances que poseen estos lenguajes.

El propósito del proyecto es centrarse en la adaptación al paradigma orientado a objetos, definiendo así que problemas se presentan para su correcto aprendizaje; así como también buscar las soluciones existentes y proponer una solución de mejora que conlleve a su correcto aprendizaje.



FACULTAD DE  
**CIENCIAS E  
INGENIERÍA**  
ESPECIALIDAD DE  
INGENIERÍA INFORMÁTICA



PONTIFICIA  
**UNIVERSIDAD  
CATÓLICA**  
DEL PERÚ

## TEMA DE TESIS PARA OPTAR EL TÍTULO DE INGENIERO INFORMÁTICO

**TÍTULO:** INTÉRPRETE Y ENTORNO DE DESARROLLO APLICADOS AL AUTO-APRENDIZAJE DE LOS CONCEPTOS DE PROGRAMACIÓN ORIENTADA A OBJETOS

**ÁREA:** Ciencia de la Computación # 476

**PROPONENTE:** Layla Hirsh

**ASESOR:** Layla Hirsh

**ALUMNO:** Ever Ricardo MITTA FLORES

**CÓDIGO:** 20070257

**TEMA N°:** \_\_\_\_\_

**FECHA:** 11 de Junio de 2013



### DESCRIPCIÓN

Con el surgimiento de los lenguajes de programación y el gran interés que estos atraen, cada vez hay más personas que deciden sumergirse a este mundo.

Otro punto a tomar en cuenta es que no solo hay un estilo de programación sino que la programación puede estar sujeta a diversos paradigmas, siendo estos el paradigma estructurado, el paradigma orientado a objetos, el paradigma orientado a eventos, entre otros.

Si bien los diversos paradigmas se relacionan entre ellos, es decir tienen características afines; en ocasiones, no es fácil para las personas dar un salto de un paradigma a otro. Otra dificultad existente suele ser que existen diversos lenguajes de programación orientados a un mismo paradigma lo que genera usualmente confusión en la definición de conceptos propios de dicho paradigma, a causa de las diferentes sintaxis y alcances que poseen estos lenguajes.

El propósito del proyecto es centrarse en la adaptación al paradigma orientado a objetos, definiendo así que problemas se presentan para su correcto aprendizaje; así como también buscar las soluciones existentes y proponer una solución de mejora que conlleve a su correcto aprendizaje.

### OBJETIVO GENERAL

Diseñar e implementar un entorno de desarrollo integrado que cuente con las funcionalidades para programación de conceptos básicos del paradigma orientado a objetos y en español.

Av. Universitaria 1801  
San Miguel, Lima - Perú

Apartado Postal 1761  
Lima 100 - Perú

Teléfono:  
(511) 626 2000 Anexo 4801





## OBJETIVO ESPECÍFICOS

Diseñar una gramática en español de simple entendimiento que permita entender, manejar e implementar los conceptos de programación orientada a objetos.

Elaborar un intérprete que permita traducir la gramática diseñada.

Realizar una arquitectura de Información para la herramienta a diseñar, definiendo así la interfaz gráfica que la herramienta tendrá.

Implementar un entorno gráfico en el cual se pueda elaborar código utilizando la gramática diseñada y a su vez permita traducirla haciendo uso de su respectivo intérprete.

## ALCANCE

El alcance del proyecto es la construcción de un entorno de desarrollo integrado que permita el auto-aprendizaje del paradigma orientado a objetos.

El Entorno estará compuesto por un Editor de Textos que soporte la inserción de código, en el que los usuarios podrán programar libremente una infinidad de clases y contar con un asistente de ayuda que contribuya al auto-aprendizaje.

Se definirá un lenguaje de programación en Español y su correspondiente interprete de modo que cumpla con los conceptos básicos de la programación orientada a objetos.

*Máximo: 100 páginas*

## AGRADECIMIENTO

A mi familia por el apoyo incondicional que me dieron durante esta época de estudios.

A mi asesora de tesis Layla Hirsh por la orientación, apoyo y amistad que me brindó.

A los profesores que de verdad se preocuparon por el mejorar del alumnado, no solo en horas de clase sino también fuera de ellas.

A mis amigos que me apoyaron a lo largo de esta etapa.

A mis lugares de trabajo, que me brindaron una oportunidad para crecer profesionalmente y a mis compañeros de labores que hicieron agradables mis días de trabajo.

ÍNDICE

<b>1. GENERALIDADES.....</b>	<b>4</b>
1.1. DEFINICIÓN DE LA PROBLEMÁTICA .....	4
1.2. OBJETIVO GENERAL .....	5
1.3. OBJETIVOS ESPECÍFICOS .....	5
1.4. RESULTADOS ESPERADOS .....	6
1.5. ALCANCE Y LIMITACIONES.....	8
1.6. MÉTODOS Y PROCEDIMIENTOS .....	9
1.6.1 <i>Metodología del Proyecto</i> .....	9
1.6.2 <i>Metodología del Producto</i> .....	10
1.6.2.1. Backus-Naur Form(BNF) .....	10
1.6.2.2. Modelo de Análisis y Síntesis de la Compilación .....	10
1.6.2.3. Técnica de Interacción con el Contexto.....	11
1.6.2.4. Extreme Programing (XP) .....	11
1.7. JUSTIFICACIÓN Y VIABILIDAD .....	11
1.8. PLAN DE PROYECTO .....	13
<b>2. ESTADO DEL ARTE.....</b>	<b>17</b>
2.1. MARCO CONCEPTUAL .....	17
2.2. REVISIÓN DEL ESTADO DEL ARTE .....	22
2.2.1. <i>Métodos actuales usados para resolver el problema</i> .....	22
2.2.1.1 Aplicativos generadores de código basado en diagramas.....	22
2.2.1.2 Aplicativos con mecanismos Drag & Drop.....	22
2.2.1.3 Aplicativos con mecanismos de botones.....	23
2.2.1.3 Por apoyo de terceros.....	23
2.2.2. <i>Aplicaciones existentes</i> .....	24
2.2.2.1. jIDEE :: Java IDE for Education4 .....	24
2.2.2.2. BlueJ – The Interactive Java Enviroment .....	25
2.2.2.3. JotAzul Object Oriented .....	26
2.2.2.4. DrRacket.....	28
2.3. ANÁLISIS SOBRE LOS RESULTADOS DE LA REVISIÓN DEL ESTADO DEL ARTE.....	29
<b>3. DESCRIPCIÓN DE LA SOLUCIÓN.....</b>	<b>30</b>
3.1. DISEÑO DE LA GRAMÁTICA .....	30
3.2. CONSTRUCCIÓN DEL INTÉRPRETE .....	30
3.3. CONSTRUCCIÓN DEL ENTORNO DE DESARROLLO.....	30
3.4. PRUEBAS.....	31
<b>4. ANÁLISIS DEL INTÉRPRETE .....</b>	<b>32</b>
4.1. CARACTERÍSTICAS DEL LENGUAJE .....	32
4.2. CONCEPTOS CONSIDERADOS EN EL LENGUAJE .....	32
4.3. DESCRIPCIÓN DEL LENGUAJE .....	34
4.3.1 <i>Representación Base</i> .....	34
4.3.2 <i>Reglas de Producción</i> .....	35
4.4. GRAMÁTICA DEL LENGUAJE.....	39
4.4.1. <i>Clase @Principal</i> .....	39
4.4.2. <i>Comentarios</i> .....	40
4.4.3. <i>Definición de Atributos</i> .....	40
4.4.4. <i>Definición de Métodos</i> .....	40
4.4.5. <i>Declaración de Variables</i> .....	41
4.4.6. <i>Asignación de Valores</i> .....	42
4.4.7. <i>Definición de Clases</i> .....	42
4.4.8. <i>Llamar a Atributos</i> .....	43
4.4.9. <i>Llamar a Métodos</i> .....	44



4.4.10. Operaciones Matemáticas .....	46
4.4.11. Otras Funcionalidades.....	46
4.4.11.1. Incrementar y Decrementar .....	46
4.4.11.2. Métodos de Impresión .....	47
4.4.11.3. Condicionales.....	48
4.4.11.4. Iterativas .....	48
<b>5. DISEÑO DEL ENTORNO DE DESARROLLO.....</b>	<b>50</b>
5.1. ARQUITECTURA DE LA INFORMACIÓN .....	50
5.1.1. Menú de Opciones Generales [ZONA 1] .....	50
5.1.2. Barra de Opciones [ZONA 2].....	51
5.1.3. Menú de Botones [ZONA 3].....	52
5.1.4. Ayudante Virtual [ZONA 4].....	52
5.1.5. Panel de Edición de Texto [ZONA 5].....	52
5.1.6. Panel de Salida [ZONA 6].....	53
5.2. DISEÑO FINAL .....	53
<b>6. CONCLUSIONES Y RECOMENDACIONES.....</b>	<b>57</b>
6.1. CONCLUSIONES.....	57
6.2. RECOMENDACIONES .....	57





## ÍNDICE DE ILUSTRACIONES Y TABLAS

## ÍNDICE DE ILUSTRACIONES

<i>Ilustración 1.1. Modelo de Análisis y Síntesis de Compilación</i> .....	10
<i>Ilustración 1.2. Técnica de Interacción con el Contexto (Ronda 2007)</i> .....	11
<i>Ilustración 1.3 Fases del desarrollo del proyecto de tesis</i> .....	14
<i>Ilustración 1.4. Diagrama de Gantt de Tareas</i> .....	16
<i>Ilustración 2.1. Entorno JIDEE (Mithat 2010)</i> .....	25
<i>Ilustración 2.2. Entorno BlueJ (University of Kent 2010)</i> .....	26
<i>Ilustración 2.3. Entorno JotAzul (Morgade)</i> .....	27
<i>Ilustración 2.4. Entorno DrRacket (Racket)</i> .....	28
<i>Ilustración 4.1. Definición de Métodos</i> .....	39
<i>Ilustración 4.2. Uso de Comentarios en BOOP</i> .....	40
<i>Ilustración 4.3. Definición de Atributos</i> .....	40
<i>Ilustración 4.4. Definición de Métodos</i> .....	41
<i>Ilustración 4.5. Declaración de Variables en BOOP</i> .....	41
<i>Ilustración 4.6. Asignación de Valores en BOOP</i> .....	42
<i>Ilustración 4.7. Definición de Clase Simple en BOOP</i> .....	42
<i>Ilustración 4.8. Definición de Clases con Herencia</i> .....	43
<i>Ilustración 4.9. Definición de Clases con Interfaces</i> .....	43
<i>Ilustración 4.10. Llamar a Atributo de la Clase</i> .....	44
<i>Ilustración 4.11. Llamar a Atributo de un Objeto</i> .....	44
<i>Ilustración 4.12. Llamar a Método de la Clase</i> .....	45
<i>Ilustración 4.13. Llamar a Método de un Objeto</i> .....	45
<i>Ilustración 4.14. Llamar a Método con Retorno</i> .....	45
<i>Ilustración 4.15. Llamar a Método con Retorno</i> .....	46
<i>Ilustración 4.16. Incrementar y Decrementar</i> .....	47
<i>Ilustración 4.17. Métodos de Impresión</i> .....	47
<i>Ilustración 4.18. Condicional SI-SINO</i> .....	48
<i>Ilustración 4.19. Iterativa Mientras</i> .....	48
<i>Ilustración 4.20. Iterativa Para</i> .....	49
<i>Ilustración 5.1. Bosquejo de zonas</i> .....	50
<i>Ilustración 5.2. Pantalla inicial</i> .....	53
<i>Ilustración 5.3. Mecanismo de Botones</i> .....	54
<i>Ilustración 5.4. Manejo de Archivos</i> .....	54
<i>Ilustración 5.5. Color de Editor de Texto</i> .....	55
<i>Ilustración 5.6. Manejo de Errores</i> .....	55
<i>Ilustración 5.7. Impresión de Archivo</i> .....	56

## ÍNDICE DE TABLAS

<i>Tabla 1.1. Mapeo de Objetivos Específicos y Resultados Esperados</i> .....	7
<i>Tabla 1.2. Descripción de Tareas</i> .....	14
<i>Tabla 3.1. Planificación de Pruebas</i> .....	31
<i>Tabla 4.1. Listado de Tokens</i> .....	35

## 1. Generalidades

En el presente capítulo, se explica la necesidad de un medio que facilite el entendimiento de los conceptos de programación orientada a objetos. Motivo por el cual se plantea un proyecto que se basa en la construcción de un aplicativo que llene ese faltante.

Con el fin de estructurar de manera adecuada el proyecto, se definen los objetivos que el presente proyecto incorporó, mapeándolos de manera clara con los resultados que se planifican entregar; asimismo también se define los alcances y limitaciones que afectan al proyecto.

Asimismo, seguido de una base metodológica (descrita en el presente documento) y un plan de proyecto en el cual se vea descrito las actividades a seguir para la elaboración del proyecto, se define así las justificativa y viabilidad del proyecto que dará riendas al inicio de éste.

### 1.1. Definición de la Problemática

Existen casos en los que las personas que tratan de introducirse en el mundo de la programación orientada a objetos (POO), no cuentan con una base teórica de los conceptos asociados, lo que dificulta o en algunos casos hace imposible una correcta implementación de distintos programas elaborados bajo ese paradigma. Resumiendo a una sola idea “Se evidencia que existe dificultad al aprender a Programar” (CHUMPITAZ 2005).

Las personas que se están iniciando en la programación de aplicaciones basadas en el paradigma orientado a objetos suelen tener problemas debido a que por causa de una incorrecta metodología de aprendizaje, no entienden los conceptos básicos de este paradigma, tales como: encapsulamiento, ocultamiento, clases abstractas, interfaz, entre otros (SANJUR 2010).

Es necesario, para poder desarrollar aplicaciones/programas eficientes, entender de manera precisa estos conceptos, no solo cómo se deben tratar

sino también en qué casos se deben usar. El tener claro estos conceptos podría ayudar no solo a las personas que programan sino también a cualquier otra persona que utilice este paradigma desde otra perspectiva como la del análisis y/o la del diseño(SANJUR 2010).

Si bien haber tenido conocimientos previos en un paradigma estructurado puede ser un buen comienzo para desenvolverse en el paradigma orientado a objetos; el problema radica en tratar de usar estos conocimientos como única base para comprender el paradigma de la POO(HERNÁN 2009).

Lamentablemente, las herramientas/lenguajes utilizados mayormente para introducirse en este paradigma orientado a objetos no permiten entender estos conceptos en su totalidad, por el contrario en algunos casos el lenguaje utilizado suele limitar el uso correcto de todos los conceptos de este paradigma tal como sucede en el caso de no permitir la herencia múltiple (ejemplo: java, C#, delphi, entre otros) o, en su opuesto, permitir este tipo de herencia pero no permitir la implementación de interfaces (ejemplo: C++, Perl, entre otros).

La persona interesada en el aprendizaje de éste paradigma debe contar con una herramienta/lenguaje cuyo fin no sea desarrollar aplicaciones complejas (netbeans, eclipse, visual estudio, entre otros), sino más bien debe utilizar herramientas sencillas que le permitan comprender de manera adecuada el paradigma para luego poder migrar a herramientas/lenguajes mas especializadas en el desarrollo(LEÓN & TORRES 2007)

## 1.2. Objetivo General

El objetivo general del presente proyecto es diseñar e implementar un aplicativo que sirva como apoyo al auto-aprendizaje de los conceptos asociados al paradigma orientado a objetos.

## 1.3. Objetivos Específicos

Los objetivos específicos que abarca el presente proyecto son:



- **OE1 :**  
Diseñar una gramática en español que permita entender, manejar e implementar los conceptos de programación orientada a objetos.
- **OE2 :**  
Elaborar un intérprete que permita traducir la gramática diseñada.
- **OE3 :**  
Realizar una arquitectura de información para la herramienta a diseñar, definiendo así la interfaz gráfica que la herramienta tendrá y sobre la cual existirá una interacción con el usuario.
- **OE4 :**  
Implementar un entorno gráfico en el cual se pueda elaborar código utilizando la gramática diseñada y a su vez permita traducirla haciendo uso de su respectivo intérprete.

#### 1.4. Resultados Esperados

Los resultados esperados son:

- **RE1 :**  
Descripción de las reglas de la gramática planteada, la cual permitirá entender, manejar e implementar los conceptos de programación orientada a objetos. Enlazado con OE1.
- **RE2 :**  
Un intérprete que permita traducir la gramática planteada, lo que permitirá convertir en acciones toda aquella sentencia que reciba, siempre y cuando ésta cumpla con la gramática propuesta. Enlazado con OE2.
- **RE3 :**  
El documento de la arquitectura de Información, asociado a la herramienta a diseñar. Enlazado con OE3.

- **RE4 :**

Prototipo de un aplicativo que contenga una interfaz gráfica en el cual, por medio de su editor de texto, se pueda elaborar código utilizando la gramática diseñada y a su vez permita, por medio de sus opciones de compilación y ejecución, traducir éste código haciendo uso del intérprete diseñado. Enlazado con OE4.

En la tabla 1.1 se muestra el mapeo entre los objetivos específicos y los resultados esperados; asimismo, se plantea la verificación considerada.

Objetivo Específico	Resultado Esperado	Verificación
OE1	RE1	Verificar la lógica gramatical, para lo cual se harán pruebas de cumplimiento de sintaxis, siguiendo el esquema de su árbol de derivaciones respectivo. Verificar la implementación de las reglas de producción, en la cual las acciones a realizar deben ser afín a su representación conceptual en términos del paradigma orientado a objetos.
OE2	RE2	Verificar la correcta traducción de la gramática introducida. Teniendo un correcto funcionamiento en la interacción intérprete y lenguaje se podrá decir que el resultado esperado ha sido realizado.
OE3	RE3	Validar que los esquemas de interacción, navegación y usabilidad satisfagan el alcance propuesto, abarcando así todos los términos descritos en la gramática planteada.
OE4	RE4	Analizar el correcto cumplimiento de la arquitectura planteada; asimismo se debe validar la correcta funcionalidad en la interacción con el intérprete.

*Tabla 1.1. Mapeo de Objetivos Específicos y Resultados Esperados*

## 1.5. Alcance y Limitaciones

Para el presente proyecto, se debe tener en cuenta las siguientes consideraciones:

- El proyecto está enfocado a facilitar el entendimiento de los conceptos del paradigma orientado a objetos por parte del usuario que haga uso de este aplicativo, por lo que tiene una relación aplicativo-usuario, lo que da a entender que su fin no es el de usarlo como medio de enseñanza para la programación orientada a objetos; sino que por el contrario es un medio de apoyo al auto-aprendizaje de estos conceptos.
- En vista que el proyecto busca enfocarse en el paradigma orientado a objetos, se limita a brindar opciones relativas a estos conceptos (definidos en el marco conceptual), no considerando así el uso de opciones adicionales cuyo enfoque es el desarrollo de aplicativos complejos, opciones tales como el uso interfaces gráficas, pilas, colas, entre otros.
- Dado que se busca abarcar una programación de nivel básico, pero con gran énfasis en el auto-aprendizaje del paradigma orientado a objetos, el editor de texto del producto solo dispondrá de un espacio de inserción de código en el que el usuario podrá programar libremente una infinidad de clases, lo cual delimita así el uso de paquetes a un solo paquete genérico.
- Dado que la finalidad del proyecto es la construcción de una herramienta que facilite y apoye el auto-aprendizaje de los conceptos del POO, mas no inicie, el aprendizaje de éstos; no se medirá el impacto que esta tenga sobre el usuario; ya que para esto se debería hacer un estudio más prolongado.
- Dentro de las finalidades del producto, no se encuentra el de usarlo con fines comerciales, motivo por el cual no se desarrollará un estudio de negocio.



## 1.6. Métodos y Procedimientos

Aquí se define tanto las metodologías utilizadas para la elaboración del proyecto, como las metodologías propias de cada componente del producto elaborado. En este punto se explicará el propósito de la metodología implementada, mas no se colocará el desarrollo asociado.

### 1.6.1 Metodología del Proyecto

Para el desarrollo del proyecto se hizo uso de la metodología conocida como Project Management Body of Knowledge (PMBOK). Si bien el PMBOK cuenta con diferentes subprocesos, el presente trabajo en mención sólo dispondrá de ciertos puntos a considerar:

#### 1.6.1.1. Gestión de la Integración del Proyecto

Referente al permiso por parte de la especialidad de Ingeniería Informática para proseguir con el presente proyecto, la definición del alcance inicial que se estimó al iniciar el proyecto, la elaboración de la sección "Plan de Proyecto" del presente trabajo y la supervisión y control constante sobre el presente proyecto.

#### 1.6.1.2. Gestión del Alcance del Proyecto

Referente a la planificación que se tomó para la delimitación del presente proyecto, lo cual abarca el alcance presentando en éste y su validación; así como la elaboración de la estructura de desglose del trabajo presentada en la sección "plan del proyecto".

#### 1.6.1.3. Gestión del Tiempo del Proyecto

Referente a la especificación de las tareas necesarias para la ejecución del proyecto, la identificación de sus predecesores necesarios y la estimación de tiempos para cada actividad a

planificar. Por lo cual se observa en la sección "Plan de Proyecto" la descripción de tareas y el diagrama de gantt asociado a este proyecto. Asimismo, hace referencia al control constante en el que se constatará el avance y cumplimiento de tareas.

## 1.6.2 Metodología del Producto

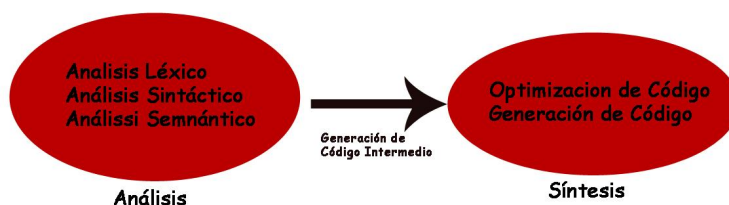
En este punto se abarca las metodologías a implementar para la correcta elaboración del producto final.

### 1.6.2.1. Backus-Naur Form(BNF)

BNF una manera matemática y formal de describir un lenguaje; consta en nombrar un símbolo seguido de las reglas que este representa. Asimismo, el lenguaje definido por ésta gramática representa el conjunto de cadenas de texto que son aceptadas con este lenguaje. (GRINNELL COLLEGE 2009)

### 1.6.2.2. Modelo de Análisis y Síntesis de la Compilación

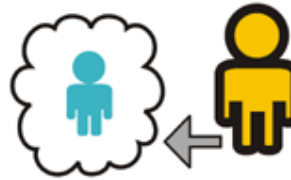
Este modelo consta dos fases llamadas "Análisis" y "Síntesis"; sin embargo, para el proyecto a trabajar no se utilizará la fase de síntesis pues ésta es básicamente enfocada en la optimización de código intermedio, lo cual no traerá consigo ningún beneficio para los fines del proyecto. (AHO 2007). En la Ilustración 1.1 se muestra en resumen las tareas de la etapa de análisis y síntesis.



*Ilustración 1.1. Modelo de Análisis y Síntesis de Compilación*

### 1.6.2.3. Técnica de Interacción con el Contexto

Dado que la arquitectura de la Información no busca un enfoque restrictivo, sino que por el contrario busca encontrar soluciones para la interacción con el usuario, la Técnica de Interacción con el Contexto es adecuada para este caso dado que ésta técnica hace referencia a la visualización del contexto (análisis de productos similares) con el fin de proponer propuestas mejores. (RONDA 2007). En la Ilustración 1.2 se muestra una gráfica descriptiva propuesta por el autor.



*Ilustración 1.2. Técnica de Interacción con el Contexto (Ronda 2007)*

### 1.6.2.4. Extreme Programming (XP)

Es una metodología Ágil para el desarrollo de software, que tiene un enfoque más práctico y dinámico puesto propone que el desarrollo del software puede ser sometido a posibles cambios que surgieran dentro del ciclo de vida del desarrollo del proyecto. (AGILE PROCESS 2009)

Dado que extreme programming es una metodología ideal para grupos de desarrollo pequeños, como en el caso de este proyecto, se adaptó correctamente a las necesidades del proyecto.

## 1.7. Justificación y Viabilidad

El proyecto está destinado a obtener un producto que ayude a entender los conceptos de programación orientada a objetos, este producto podrá ser



usado por cualquier persona en general que desee introducirse en este paradigma, aunque se recomienda tener conocimientos previos de lo que es la programación en general con el fin de darle un mejor aprovechamiento ya que la finalidad no es la de dar un énfasis general de programación sino es de dar un énfasis en conceptos propios al paradigma orientado a objetos.

Asimismo, si bien el proyecto está basado en el paradigma orientado a objetos, el producto final tendrá como característica la facilidad de modificación con el fin que se pueda guiar a facilitar el entendimiento de otros paradigmas (posibles proyectos futuros), tales como el paradigma estructurado, el paradigma orientado a eventos, entre otros.

Si bien la función del aplicativo final en este proyecto se limita a ser una herramienta que facilite el auto-aprendizaje, esto no delimita que con una investigación adecuada, en un futuro proyecto, se pueda implementar el producto como herramienta de enseñanza; lo que si se recomienda es usar en conjunto la base teórica obtenida de este proyecto tal como las metodologías actuales, con el fin de cumplir con la nueva perspectiva que se le dé al producto.

Con el fin de analizar la viabilidad económica del proyecto, se realiza una estimación en base al costo aproximado de "hora destinada". "La hora destinada" es un término que hace referencia al conjunto de costos que se dan en 60 minutos utilizados en realizar parte de una tarea asignada al proyecto. Este costo no sólo incluye el costo de hora hombre, sino que también abarca los costos indirectos a los cuales esta afecto como es el gasto de servicios básicos (ejemplo: luz), el costo de papel utilizado en esa hora, entre otros.

Teniendo como aproximación de costo de hora destinada un valor de s/10 y tomando en cuenta que el plan de proyecto se estima sobre un total de 250 horas, se observa que el costo final del proyecto es de 2500 soles, lo cual implica un bajo costo comparado a los fondos destinados que se otorgan para proyectos similares a éste.

## 1.8. Plan de Proyecto

Tomando en cuenta el inicio del proyecto con fecha 01-07-2012 y con fecha de cierre al 15-12-2013, se observa un total de 4 meses y medio que serán destinados al desarrollo del mismo. Asumiendo también, un tiempo destinado de 1 hora y 50 minutos al día para realizar los avances asociados a las actividades del proyecto, obtendremos que el tiempo para el desarrollo proyecto equivaldría a un total de 250 horas trabajadas sobre las cuales se debe realizar la división de tareas.

Asimismo, se divide el plan de proyecto en 4 fases de desarrollo, estas son:

- Definición del Lenguaje: orientado a describir el lenguaje, describiendo así las características que este posee y su gama de conceptos asociados a la programación orientada a objetos.
- Implementación del Lenguaje: implica el desarrollo del intérprete que permitirá usar el lenguaje implementado, para lo cual se desarrollará cada una de las fases según el modelo de análisis y síntesis de la compilación.
- Implementación del Entorno de Desarrollo: esta etapa se basa en el desarrollo del entorno gráfico que permitirá al usuario hacer uso del lenguaje implementado, apoyando así este uso con el panel de opciones que el entorno posea.
- Proceso de Finalización: esta fase se basa básicamente en la implementación de pruebas que certifiquen el correcto funcionamiento del producto final; así también, una vez corroborado el funcionamiento del aplicativo se debe proceder a diseñar el instalador respectivo del producto.

En la Ilustración 1.3 se observa la estructura de desglose del trabajo, describiendo así las fases planteadas para cada módulo del proyecto.

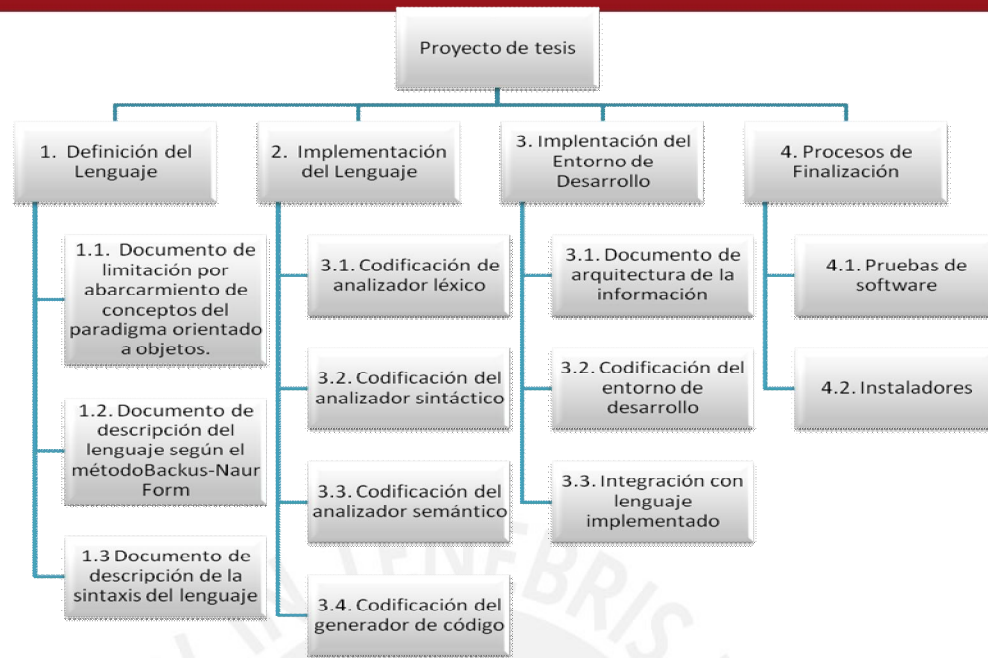


Ilustración 1.3 Fases del desarrollo del proyecto de tesis

Nº	Tarea	Duración	Predecesor
1	Definición de conceptos que abarcará el lenguaje	2 horas	
2	Descripción formal del lenguaje según el método Backus-Naur Form	24 horas	1
3	Descripción de la sintaxis que poseerá el lenguaje	3 horas	2
4	Codificación del analizador léxico	3 horas	3
5	Codificación del analizador sintáctico	12 horas	4
6	Codificación del analizador semántico	6 horas	5
7	Codificación del generador de código	50 horas	6
8	Diseño de arquitectura de la información	6 horas	
9	Codificación de la parte gráfica del entorno	30 horas	8
10	Codificación de la interacción del panel de opciones con el editor de texto	14 horas	9
11	Codificación del menú	10 horas	9
12	Codificación de interacción de estilos en el editor de texto	8 horas	9
13	Codificación de funcionalidades del ayudante	14 horas	9
14	Integración del entorno con el intérprete	20 horas	11 , 7
15	Fase de pruebas	38 horas	14
16	Creación del instalador	10 horas	15

Tabla 1.2. Descripción de Tareas

Definido ya los tiempos requeridos y los puntos que abarca el proceso de desarrollo del proyecto, se desarrolla una división por tareas mostrada en la Tabla 1.2.

Asimismo, la Ilustración 1.4. muestra el desarrollo del diagrama de Gantt de Tareas para el presente proyecto, tomando en cuenta las horas destinadas para el cumplimiento de las tareas descritas en la Tabla 1.2.

Para el diagrama de gantt elaborado no se toma en cuenta las fechas de inicio y culminación de cada actividad puesto que al ser un proyecto individual se estima que el avance puede darse sin seguir un estricto orden de fechas y no cumpliendo necesariamente con el estimado de 1 hora y 50 minutos de avance por día.

Asimismo, para la presente estimación, se toma en consideración que las tareas asociadas a la documentación de avances están inmersas en los tiempos estimados para el desarrollo de su avance asociado.

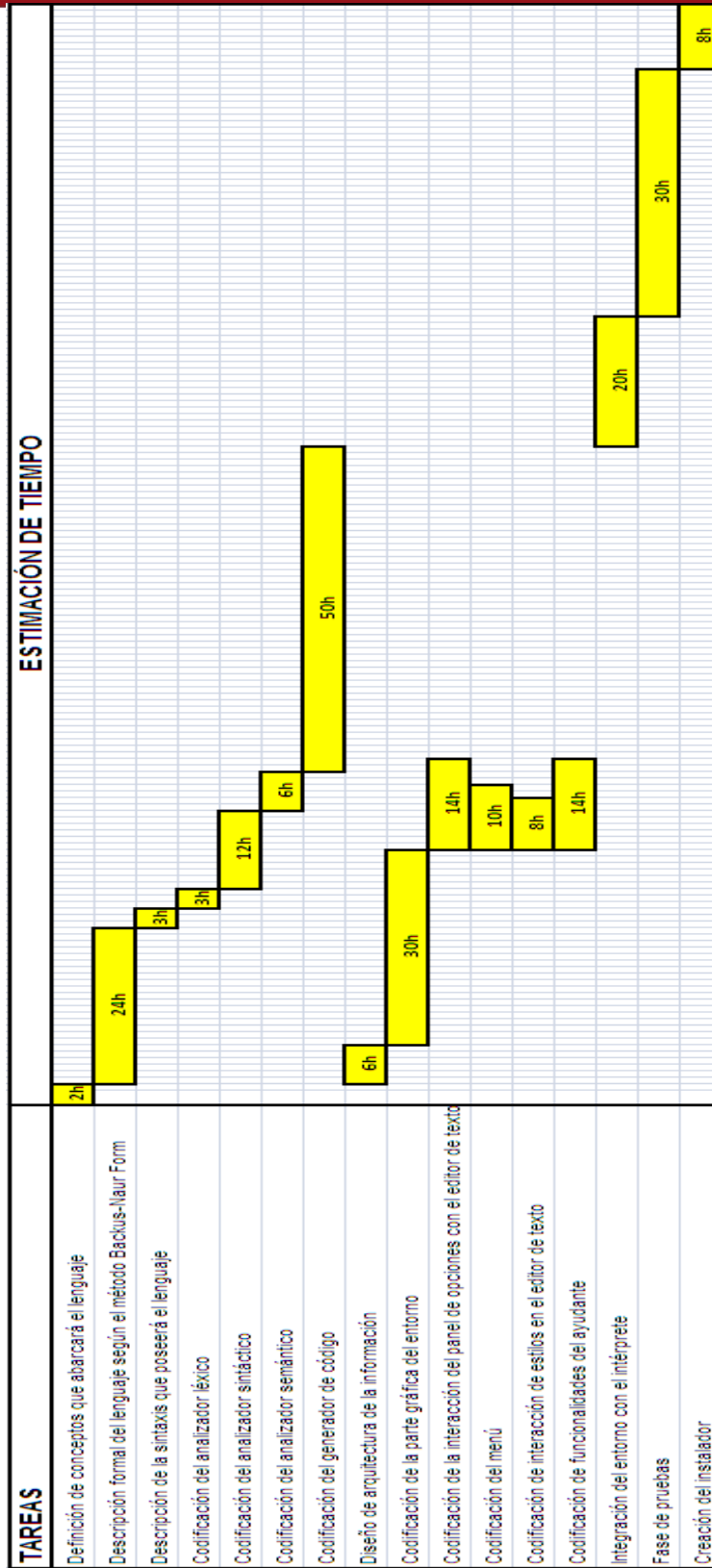


Ilustración 1.4. Diagrama de Gantt de Tareas



## 2. Estado del Arte

Para todo proyecto, es esencial tener como base ciertos conceptos con el fin de poder entender el contexto sobre el cual se está trabajando; motivo por el cual, el presente capítulo busca definir los conceptos necesarios para abarcarse en este proyecto.

Asimismo, también se da una descripción de los productos y metodologías actuales que satisfacen la solución al problema planteado en el capítulo anterior; buscando así, obtener las ventajas y desventajas que serán consideradas para el desarrollo del producto final.

### 2.1. Marco Conceptual

A continuación, tomando como referencia el libro “Java How to Program” de Deitel & Associates, Inc, se presenta algunos conceptos utilizados para la presente tesis:

#### 2.1.1. Lenguaje de Programación

Es la metodología que une estructura de datos y algoritmos con el fin de resolver un determinado problema.

#### 2.1.2. Programación Orientada a Objetos [POO]

Es un paradigma de la programación que se basa en diseñar formatos de datos con características y comportamientos.

#### 2.1.3. Clase

Es una estructura de datos que contiene características y comportamientos.

#### 2.1.4. Objeto

Según Deitel “Cualquier cosa que se vea en el mundo real son objetos” (DEITEL 2005). Por otra parte, se define a un objeto es una instancia de una clase; es decir un objeto es un ente que cumple con todas las características de su clase asociada.

#### 2.1.5. Atributo

Se denomina atributo a las características que contiene una clase, siendo éstos estructuras de datos de cualquier tipo.

#### 2.1.6. Método

También llamado función miembro, es el comportamiento que puede tener una clase; hace referencia a las funciones que una clase posee.

#### 2.1.7. Encapsulamiento

Es una característica propia del paradigma orientado a objeto, se basa en el agrupamiento de todos los elementos de una clase (atributos y métodos), lo cual permite ver a la clase como una caja negra puesto se puede usar los comportamientos (métodos) de la ésta sin saber la construcción de éstos.

#### 2.1.8. Ocultamiento

Es una propiedad fundamental en el paradigma orientado a objeto, puesto esto delimita el uso de métodos y atributos de una clase a solo aquellas instancias que tengan los permisos adecuados; es decir, consta en limitar la accesibilidad de los elementos de una clase.

### 2.1.9. Herencia

Es la propiedad que permite crear una clase padre a partir de otras ya existentes (clases hijas), pudiendo agrupar atributos y/o métodos en común de las clases hijas dentro de la clase padre.

### 2.1.10. Clase Base

Es la clase de la cual heredan sus propiedades las clases hijas.

### 2.1.11. Subclase

Dícese de un clase que ha heredado propiedades de otra clase (clase base).

### 2.1.12. Herencia Múltiple

Es la propiedad que permite que una clase pueda heredar propiedades de una o más clases bases.

### 2.1.13. Polimorfismo

“Es la propiedad de que un operador o una función actúen de modo diferente en función del objeto sobre el cual se aplican” (JOYANES 2006).

### 2.1.14. Sobreescritura de Métodos

Propiedad que se da cuando una clase define un método con las mismas características (nombre, número de argumentos y tipos de argumentos) que un método declarado en su clase base.

### 2.1.15. Sobrecarga de Métodos

Propiedad que se da cuando una clase define un método de más de una forma, es decir con diferentes características (número de

argumentos y tipos de argumentos).

#### **2.1.16. Clase Abstracta**

Dícese de una clase cuya finalidad es la herencia, para lo cual hace manejo de atributos y métodos abstractos, éstos de no ser implementados en la clase base deberán ser implementados en las subclases.

#### **2.1.17. Interfaz**

Hace referencia a un modelo cuya funcionalidad es solo definir los comportamientos (indicar el detalle del método, mas no su funcionamiento) que deberá implementar toda aquella clase que haga uso de la interfaz.

#### **2.1.18. Entorno de Desarrollo Integrado**

Dícese de un aplicativo cuya finalidad es ayudar al programador, dándole a éste facilidades para la interacción con él código fuente. En su mayoría consta de los siguientes elementos: compilador, Intérprete, editor de texto, depurador.

#### **2.1.19. Código Fuente**

Es todo aquel código escrito bajo la sintaxis de un lenguaje de programación con el fin de ser ejecutado.

#### **2.1.20. Sintaxis**

También llamada gramática, es un conjunto de reglas que describen la secuencia de elementos permitida en un determinado lenguaje de programación.

### 2.1.21. Compilador

Es un traductor que por medio de un análisis hacia el código fuente, permite generar un código equivalente escrito en otro lenguaje de programación.

### 2.1.22. Intérprete

Consta en el análisis hacia el código fuente con el fin de generar su ejecución. Las tareas a realizar en ésta ejecución dependerán de la definición de acciones descritas en la gramática.

### 2.1.23. Depurador

Es la herramienta que permite al programador observar paso a paso la ejecución del programa.

### 2.1.24. Editor de Texto

Es el espacio de trabajo en el cual un programador puede escribir su código fuente, por lo peculiar da facilidades como mostrar el número de línea en el cual se encuentra.

### 2.1.25. Aprendizaje

Es la acción de captar nuevos conceptos, para lo cual solo se necesita un actor el cual sería la persona que adquiere estos conocimientos, ya sea de manera intuitiva o por instrucción.

### 2.1.26. Enseñanza

Es el proceso mediante el cual se comunican o se transfieren conocimientos, para lo cual se necesitan dos actores como mínimo siendo uno de éstos el emisor y el otro optando por el rol de receptor



## 2.2. Revisión del Estado del Arte

En el presente punto, se presenta tanto las metodologías actuales para resolver el problema, así como los aplicativos ya existentes que tratan de resolver el problema.

### 2.2.1. Métodos actuales usados para resolver el problema

A continuación, se indica algunos de los métodos usados para facilitar el entendimiento de los conceptos (definidos en el marco conceptual) de la programación orientada a objetos [POO]:

#### 2.2.1.1 Aplicativos generadores de código basado en diagramas

Existencia de aplicativos que generan el código de clases a partir de un diagrama, acorde a esto existen varios aplicativos en los cuales el usuario tiene que diseñar un diagrama de clases con la finalidad de que el programa lo interprete y le muestre a éste el código correspondiente a las clases descritas, mostrando sus atributos, métodos e incluso relaciones.

- Basado en las entrevistas realizadas (ver anexos) y la experiencia del autor, se aprecia que este método presenta dos consideraciones: la primera que es necesario partir de un diagrama de clases adecuado y esto solamente se puede tener como resultado si es que se tienen claros algunos conceptos del paradigma orientado a objetos (ejemplo: clases abstractas, herencia, herencia múltiple); otro punto a considerar es que no permite tener una interacción inmediata con el código pues la generación de código se realiza una vez acabado el diagrama (en la mayoría de los casos).

#### 2.2.1.2 Aplicativos con mecanismos Drag & Drop

Existencia de entornos de desarrollo integrado que por medio de

mecanismos arrastrar y pegar (drag & drop) permiten a la persona elaborar interfaces gráficas de manera muy sencilla y con poca interacción con el código fuente.

- Basado en las entrevistas realizadas (ver anexos) y la experiencia del autor, se aprecia que lejos de ayudar al usuario, este tipo de herramientas permiten el desarrollo de aplicaciones sin necesariamente haber entendido los conceptos relacionados con la POO.

### 2.2.1.3 Aplicativos con mecanismos de botones

Existencia de entornos de desarrollo que contienen mecanismos de ayuda que generan el código de una acción seleccionada, en este caso se hace uso de botones o áreas diseñadas para la creación de elementos específicos como “crear clase”, “añadir método”, “añadir atributo”, entre otros.

- Basado en las entrevistas realizadas (ver anexos) y la experiencia del autor, se aprecia que éste es uno de los mejores métodos si se usa de manera adecuada, pues permite visualizar el código específico de la acción seleccionada, lo cual permite un mejor detalle de la situación. Es necesario en este punto tener en cuenta la finalidad del entorno de desarrollo, ya que lo que buscan es que el usuario entienda estos conceptos en lugar de que el usuario pueda desarrollar aplicaciones complejas.

### 2.2.1.3 Por apoyo de terceros

La explicación de los temas y el uso de documentación son puntos referidos a la educación en programación que el usuario pueda recibir (ejemplo: cursos en la educación superior o en carreras técnicas) y/o al uso de libros relacionados a este tema.

- Basado en las entrevistas realizadas (ver anexos) y la experiencia del autor, se aprecia que si bien este método es el más usado, se puede observar algunos percances que impediría cumplir con la misión planteada (ejemplo: distracción por parte del usuario durante ciertos puntos de la explicación/lectura, no permitir la práctica en conjunto en el caso de los libros)

### 2.2.2. Aplicaciones existentes

A continuación, se indicará algunos de los aplicativos actuales usados para facilitar aprendizaje de los conceptos de la programación orientada a objetos:

#### 2.2.2.1. jIDEE :: Java IDE for Education4

jIDEE es un entorno de programación gratuito diseñado para satisfacer los requerimientos pedagógicos para la introducción a Java.

La Ilustración 2.1 muestra el entorno de este aplicativo (Mithat 2010).

Debido a sus características podemos destacar:

#### **Ventajas:**

- Manejo de archivos fuentes, archivos con extensión “.class”.
- Ejecución con argumentos.
- Compilación y Ejecución.
- El manejo se realiza con un editor de texto, para lo cual contiene una interfaz para elegirlo.

- Contiene una interfaz para elegir el Path para el JDK.

### Desventajas:

- Usa un editor de texto externo
- No contiene mecanismos que faciliten la entendimiento de los conceptos de POO

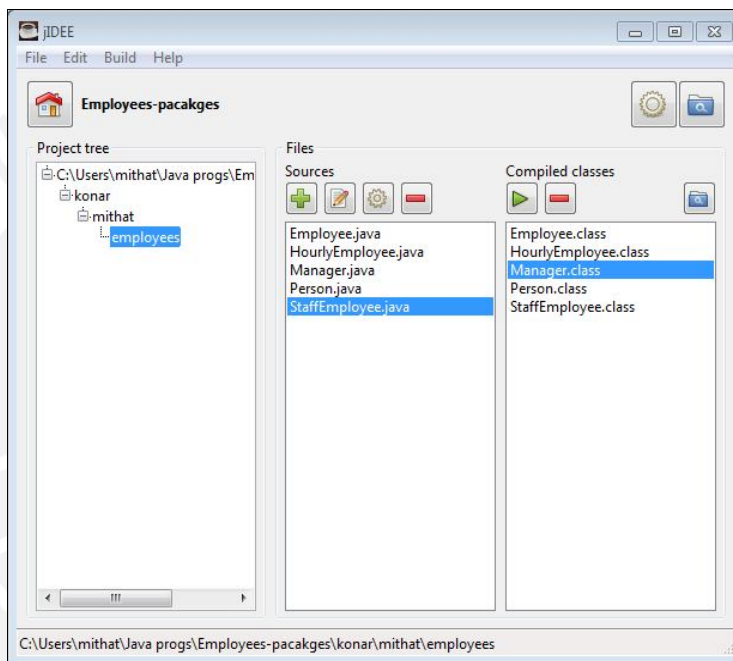


Ilustración 2.1. Entorno jIDEA (Mithat 2010)

### 2.2.2.2. BlueJ – The Interactive Java Enviroment

BlueJ es un entorno de desarrollo elaborado con el propósito de la enseñanza del paradigma de la orientación a objetos con Java.

La Ilustración 2.2 muestra el entorno de este aplicativo. (University of Kent 2010)

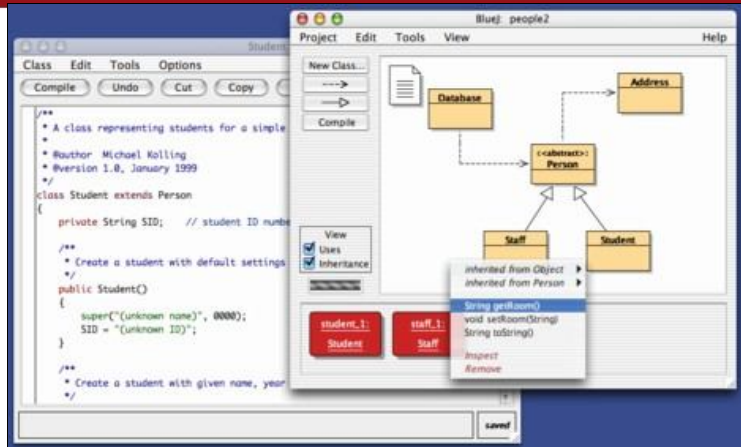


Ilustración 2.2. Entorno BlueJ (University of Kent 2010)

Debido a sus características podemos destacar:

#### Ventajas:

- Interpretación de paquetes y clases.
- Botones para el manejo de herencia y dependencia entre clases.
- Edición personalizada para clases.
- Interpretación de paquetes.

#### Desventajas:

- Los atributos y métodos son definidos de manera manual.
- No se muestra directamente el código generado al hacer uso de los mecanismos de herencia, se tiene que primero usar la herramienta y posteriormente ver la clase.

### 2.2.2.3. JotAzul Object Oriented

JotAzul es un entorno de desarrollo diseñado con el fin de ayudar a los principiantes en el mundo de la programación orientada a objetos en Java.



La Ilustración 2.3 muestra el entorno de este aplicativo.  
 (Morgade)

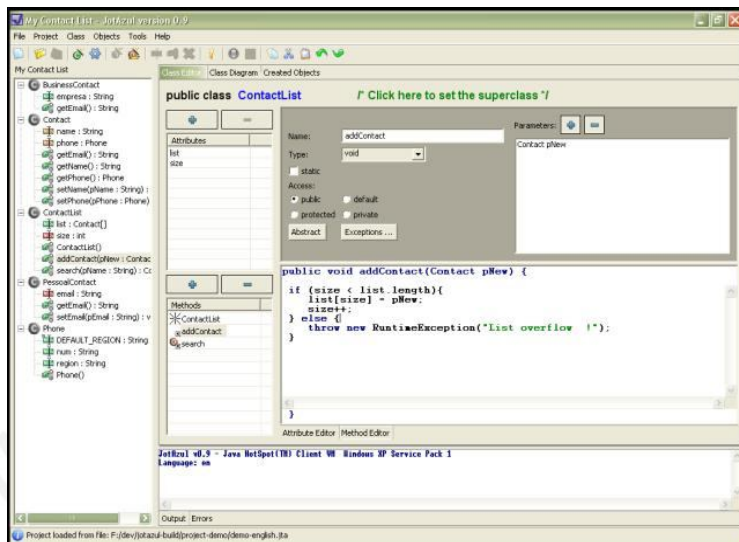


Ilustración 2.3. Entorno JotAzul (Morgade)

Debido a sus características podemos destacar:

#### Ventajas:

- Mecanismos para agregar atributos y clases de manera simple.
- Manejo de clases por medio de una estructura en la parte lateral izquierda.
- Uso de Diagrama de clases
- Manejo de abstracción

#### Desventajas:

- El lenguaje usado es Java, por lo cual no permite herencia múltiple.
- La interfaz es poco amigable, puesto todas las áreas están comprimidas.

- Dependencia de los mecanismos para agregar atributos y métodos, no se puede modificar el código como si fuese uno solo.
- Poco conocida por el público.

#### 2.2.2.4. DrRacket

DrRacket es un entorno de desarrollo para el aprendizaje de los conceptos de programación, anteriormente llamado PL-Scheme (conocido también como Dr Scheme). La Ilustración 2.4 muestra el entorno de este aplicativo. (Racket)

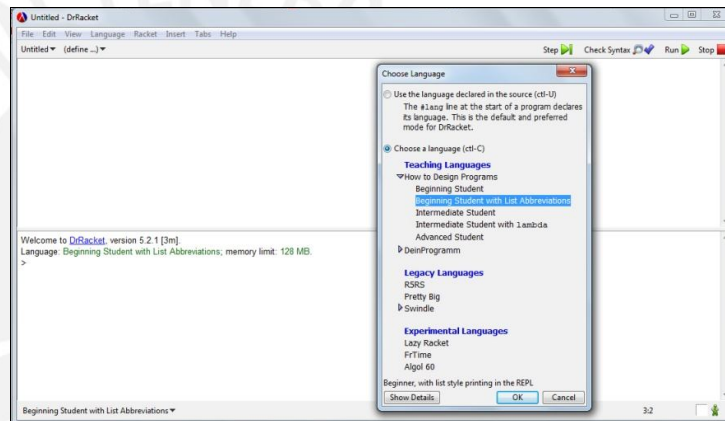


Ilustración 2.4. Entorno DrRacket (Racket)

Debido a sus características podemos destacar:

#### Ventajas:

- Es una de las herramientas más conocidas.
- Acepta diferentes paquetes de lenguaje.
- Contempla diferentes tipos de programación.

#### Desventajas:

- No contiene una interfaz gráfica con mecanismos de ayuda.

### 2.3. Análisis Sobre los Resultados de la Revisión del Estado del Arte

Según lo observado en el punto anterior, podemos deducir que no se cuenta con un producto que satisfaga en su totalidad el problema planteado, puesto se observan diversas desventajas a lo largo del análisis de cada uno de ellos.

Asimismo, se ve la dependencia a usar un lenguaje de programación orientado a objetos que ya existe y cuyo fin es el desarrollo de aplicaciones complejas; esto produce en ocasiones la confusión por parte del usuario al introducirse en otros lenguajes orientados a objetos que no cuenten con ciertas funcionalidades o por el contrario implementen nuevas funcionalidades (ambas ligadas a los conceptos de programación orientada a objetos).

Por ende se afirma el planteamiento de un nuevo lenguaje de programación que englobe los conceptos del paradigma orientado a objetos, con el fin de no establecer una dependencia a la sintaxis.

### 3. Descripción de la Solución

La solución tiene como resultado final un aplicativo denominado "BOOP - Basic Object Oriented Programming"; sin embargo su desarrollo engloba lo siguiente:

#### 3.1. Diseño de la Gramática

Hace referencia al objetivo específico 1, el cual implica diseñar una gramática en español que permita entender, manejar e implementar los conceptos de programación orientada a objetos.

Consiste en crear un lenguaje de programación representando sus reglas de producción según las normativas de la metodología Backus Naur Form.

#### 3.2. Construcción del Intérprete

Luego del diseño de la gramática y como primera tarea para la implementación del intérprete, se procede con implementar en lenguaje java un analizador léxico que pueda realizar la lectura de los caracteres ingresados por el usuario y convertirlos en tokens que serán utilizados a posteriori.

Asimismo, se procede a obtener el intérprete mediante el uso del generador de analizadores sintácticos YACC. Esta herramienta utiliza el analizador léxico elaborado, las notaciones BNF y acciones semánticas asignadas a estas para generar un código en java capaz de realizar el análisis del lenguaje.

#### 3.3. Construcción del Entorno de Desarrollo

Para la interacción con el usuario se realizará un entorno de desarrollo que contemple las características de arquitectura de información obtenidas a partir de la técnica de interacción con el contexto.

Para esto, se utiliza la herramienta Netbeans 7.2.1 que permite la elaboración de un proyecto realizado en lenguaje java que engloba tanto la parte visual como el procesamiento del texto ingresado por el usuario(integración con el intérprete diseñado un paso anterior).

### 3.4. Pruebas

La metodología implementada en el desarrollo presente proyecto requiere la realización tanto de pruebas unitarias constantes como de pruebas finales de integración; por lo cual, siguiendo con el diagrama de tareas planificado, se creó un esquema de pruebas que abarque tanto la validación del intérprete como la del entorno de desarrollo.

La Tabla 3.1 muestra la planificación de pruebas para el presente proyecto. Asimismo, el anexo E03 - "Resultado de Ejecución de Pruebas" muestra los resultados satisfactorios de las pruebas realizadas.

Nº Prueba	Descripción de la Prueba
P01	El entorno de desarrollo posee las secciones establecidas en el diseño.
P02	Se genera el código correcto al usar el botón "Crear Clase"
P03	Se genera el código correcto al usar el botón "Crear Objeto"
P04	Se genera el código correcto al usar el botón "Añadir Atributo"
P05	Se genera el código correcto al usar el botón "Añadir Método"
P06	Se genera el código correcto al usar el botón "Crear Clase Abstracta"
P07	Se genera el código correcto al usar el botón "Heredar"
P08	Se genera el código correcto al usar el botón "Implementar Interfaz"
P09	Se genera el código correcto al usar el botón "Sobreescribir Método"
P10	Se genera el código correcto al usar el botón "Sobrecargar Método"
P11	Se ejecuta correctamente un código que incluye la sintaxis de creación de clase
P12	Se ejecuta correctamente un código que incluye la sintaxis de creación de método
P13	Se ejecuta correctamente un código que incluye la sintaxis de creación de atributo
P14	Se ejecuta correctamente un código que incluye la sintaxis de creación de clase abstracta
P15	Se ejecuta correctamente un código que incluye la sintaxis de creación de interfaz
P16	Se ejecuta correctamente un código que incluye la sintaxis de uso de atributo
P17	Se ejecuta correctamente un código que incluye la sintaxis de uso de método
P18	Se ejecuta correctamente un código que incluye la sintaxis de herencia simple
P19	Se ejecuta correctamente un código que incluye la sintaxis de herencia múltiple
P20	Se ejecuta correctamente un código que incluye la sintaxis de uso de interfaces
P21	El ayudante virtual brinda la visualización de los conceptos asociados al paradigma orientado a objetos
P22	El ayudante virtual brinda el audio correspondiente a lectura del texto mostrado
P23	El aplicativo puede guardar correctamente el texto codificado en un archivo con extensión .boop
P24	El aplicativo puede abrir correctamente un archivo de la extensión .boop
P25	El aplicativo permite mandar a imprimir el texto codificado
P26	El aplicativo permite el cambio de colores

Tabla 3.1. Planificación de Pruebas



#### 4. Análisis del Intérprete

La solución necesita tener implementada un intérprete que contenga tanto los conceptos principales de la programación orientada a objetos, así como, una estructura simple de entender por parte del usuario; motivos por los cuales se realiza un análisis con el fin de definir los conceptos abarcados por éste y su estructura a implementar.

##### 4.1. Características del Lenguaje

Debido a que el aplicativo va dirigido a personas hispanohablantes, se decide optar por una gramática en español ya que no se busca una migración a otro lenguaje por medio de sintaxis similares, ya que esto puede conllevar a una dependencia de sintaxis, motivo por el cual no sería factible crear un nuevo lenguaje de programación. Asimismo, esto no dificulta la migración a otro lenguaje de programación ya que al tener un pleno conocimiento de todos los conceptos asociados al paradigma orientado a objetos, es fácil para el usuario introducirse en otro lenguaje de programación teniendo que identificar la sintaxis adecuada.

Asimismo, al ser un nuevo lenguaje de programación se tiene como requisito la utilización en paralelo de un entorno de desarrollo que haga uso del intérprete respectivo.

##### 4.2. Conceptos Considerados en el Lenguaje

Tomando como base los conceptos descritos en la sección “Marco Conceptual” del presente documento, se listará los conceptos seleccionados y considerados tanto para la elaboración del entorno de desarrollo como para la elaboración del Intérprete :

- Clases
- Objetos
- Tipos de Datos
  - Entero
  - Decimal

- Atributos
- Métodos
- Abstracción
- Herencia
  - Herencia Simple
  - Herencia Múltiple
- Interfaces
  - Implementación Simple
  - Implementación Múltiple
- Sobreescritura de Métodos
- Sobrecarga de Métodos
- Modificadores de acceso
  - Público
  - Privado
  - Protegido

Asimismo, se incluyen los siguientes conceptos generales en un lenguaje de programación:

- Operaciones matemáticas
  - Suma
  - Resta
  - Multiplicación
  - División
  - División Entera
  - Residuo
  - Potencia
- Otras Funcionalidades
  - Incrementar
  - Decrementar
  - Impresión
  - Impresión por Línea
  - Condicionales
  - Iterativas

### 4.3. Descripción del Lenguaje

En el presente punto, se presenta el manejo de sintaxis utilizado en la elaboración del intérprete.

#### 4.3.1 Representación Base

Las palabras son calificadas con el término "TOKEN", cada uno de estos serán usados en la interpretación del lenguaje(AHO 2007). En la tabla 4.1 se aprecia los tokens definidos en la elaboración del intérprete así como su valor equivalente (valor referente a texto del lenguaje).

TOKEN	VALOR
TK_CLASE	\$clase
TK_INTERFAZ	\$interfaz
TK_ATRIBUTO	#atributo
TK_METODO	#metodo
TK_PRINCIPAL	@principal
TK_EJECUTAR	#ejecutar
TK_ENTERO	\$entero
TK_DECIMAL	\$decimal
TK_PUBLICO	\$publico
TK_PRIVADO	\$privado
TK_PROTEGIDO	\$protegido
TK_ABSTRACCION	\$abstracta
TK_POTENCIA	\$potenciar
TK_DIVENTERA	\$divisionEntera
TK_RESIDUO	\$residuo
TK_HEREDA	\$hereda
TK_IMPLEMENTA	\$implementa
TK_COMO	\$de
TK_ENTORNO	\$entorno
TK_IMPRESION	\$imprime
TK_IMPRESION2	\$impr
TK_MET	\$llamar
TK_SI	\$si

TK_SINO	\$sino
TK_PARA	\$para
TK_MIENTRAS	\$mientras
TK_INC	\$inc
TK_DEC	\$dec

*Tabla 4.1. Listado de Tokens*

#### 4.3.2 Reglas de Producción

Las reglas de producción es la estructura que seguirá el intérprete con el fin de reconocer como válido el texto introducido. A continuación se lista las principales reglas de producción definidas, tomando como referencia la representación base elaborada, para el correcto funcionamiento del intérprete:

```

cuerpo : |
           |
           | clase cuerpo
           |
           | interfaz cuerpo

interfaz : TK_INTERFAZ ID_CLASE '{ seccionAtributos
           seccionDefMetodos }'

seccionDefMetodos : | defMetodo seccionDefMetodos

defMetodo : TK_METODO ':' modificadorAcceso tipoDato ID '('
           seccionParametros ')' ';'

clase : TK_CLASE abstraccion ID_CLASE seccionHerencia
           seccionInterfaces '{ seccionAtributos seccionMetodos }'

abstraccion : | siAbstraccion

siAbstraccion : TK_ABSTRACCION

seccionHerencia : | TK_HEREDA listaClasesHerencia ;

listaClasesHerencia : ID_CLASE
           | listaClasesHerencia ',' ID_CLASE

```

**seccionInterfaces** : | TK\_IMPLEMENTA listaClasesImplementa

**listaClasesImplementa** : ID\_CLASE  
| listaClasesImplementa ',' ID\_CLASE

**clasePrincipal** : TK\_PRINCIPAL

**seccionEjecutar** : TK\_EJECUTAR '{ listaInstrucciones }'

**seccionAtributos** : | atributo seccionAtributos

**atributo** : TK\_ATRIBUTO ':' modificadorAcceso tipoDato ID opcAtrib ':'

**opcAtrib** : | '=' NUM

**seccionMetodos** : | metodoAbs seccionMetodos  
| metodo seccionMetodos

**metodo** : TK\_METODO ':' modificadorAcceso tipoDato ID '('  
seccionParametros ')"' '{ listaInstrucciones }'

**metodoAbs** : TK\_METODO ':' siAbstraccion modificadorAcceso tipoDato  
ID '(' seccionParametros ')';'

**seccionParametros** : | seccionSiParametros

**seccionSiParametros** : tipoDato ID  
| seccionSiParametros ',' tipoDato ID

**listaInstrucciones** : | instruccion listaInstrucciones

**instruccion** : asignacion  
| declaracionVariable ';' ;'  
| imprimir ';' ;'  
| llamaMetodos ';' ;'  
| retornar  
| condicional  
| buclefor  
| buclewhile

| incrementos ';' ;

**condicional** : TK\_SI comparaciones '{' listaInstrucciones '}' listaELSE

**listaELSE**: | TK\_SINO '{' listaInstrucciones '}'

**buclefor** : TK\_PARA '(' asignacion compara ';' incrementos ')' '{'  
 listaInstrucciones '}'

**buclewhile** : TK\_MIENTRAS '(' compara ')' '{' '}'

**compara** : expresionGeneral '>' expresionGeneral  
 | expresionGeneral '<' expresionGeneral  
 | expresionGeneral '=' expresionGeneral  
 | expresionGeneral '>=' expresionGeneral  
 | expresionGeneral '<=' expresionGeneral

**comparaciones** : '(' compara ')'

**retornar** : TK\_RETORNA expresionGeneral ';' ;

**declaracionVariable** : tipoDato ID

**asignacion** : variables '=' expresionGeneral ';' ;

**imprimir** : TK\_IMPRESION '(' expresionGeneral ')' ;  
 | TK\_IMPRESION2 '(' expresionGeneral ')';

**expresionGeneral** :expresion;

**incrementos** : TK\_INC '(' variables ')' ;  
 | TK\_DEC '(' variables ')'

**expresion** : expresion '+' termino  
 | expresion '-' termino  
 | termino

**termino** : termino '\*' factor  
 | termino '/' factor



| termino TK\_DIVENTERA factor  
| termino TK\_RESIDUO factor  
| factor

**variables** : buscaAtributo  
| buscaAtributo '.' auxHerencia ID  
| buscaSimbolo  
| buscaSimbolo '.' auxHerencia ID  
| siHerencia ID

**llamaMetodos** : entradaMetodo TK\_ENTORNO '.' ID listaParams  
salidaMetodo  
| entradaMetodo siHerencia ID  
| entradaMetodo buscaAtributo '.' auxHerencia ID  
listaParams salidaMetodo  
| entradaMetodo buscaSimbolo '.' auxHerencia ID  
listaParams salidaMetodo

**entradaMetodo** : TK\_MET '['

**salidaMetodo** : ']

**buscaAtributo** : TK\_ENTORNO '.' ID

**buscaSimbolo** : ID  
| CADENA

**listaParams** : '('  
| '(' listaParamsSI ')'

**listaParamsSI** : expresionGeneral  
| listaParamsSI ',' expresionGeneral

**auxHerencia** : | siHerencia

**siHerencia** : TK\_COMO '[' ID\_CLASE ']' '.'

**factor** : '(' expresionGeneral ')'  
| '.' factor

```

| NUM
| operaciones
| llamaMetodos
| variables

```

**operaciones** : TK\_POTENCIA '(' expresion ',' expresion ')'

**tipoDato** : TK\_DECIMAL  
 | TK\_ENTERO  
 | ID\_CLASE  
 | TK\_NADA

**modificadorAcceso** : TK\_PUBLICO  
 | TK\_PRIVADO  
 | TK\_PROTEGIDO

#### 4.4. Gramática del Lenguaje

A continuación, se presenta algunos ejemplos de la gramática manejada por el lenguaje implementado.

##### 4.4.1. Clase @Principal

La clase principal es la clase inicial, es decir es la clase que se ejecutará e interpretará primero. Dentro de ésta clase se podrán definir atributos y métodos; sin embargo, cabe resaltar que su método de inicio es el método "**#ejecutar**". La Ilustración 4.1 muestra la estructura básica de la clase principal.

```

@principal {
    /** Sección Atributos **/

    /** Sección Métodos **/

    #ejecutar{
        /** Líneas de Código **/
    }
}

```

*Ilustración 4.1. Definición de Métodos*

#### 4.4.2. Comentarios

Un comentario es un conjunto de caracteres delimitados por la nomenclatura barra-asterisco "`/*`" y "`*/`", todo el código introducido dentro de estos delimitadores no serán considerados al momento de compilación y ejecución. La Ilustración 4.2 muestra un ejemplo del uso de comentarios.

```
/*Declaración de Variables*/
$entero edad; /* edad en años de la persona */
$entero dni; /* dni de la persona */
```

Ilustración 4.2. Uso de Comentarios en BOOP

#### 4.4.3. Definición de Atributos

Para definir un atributo dentro de una clase, se debe colocar el identificador `#atributo` seguido del símbolo `:`, seguido de las características del atributo. Las características del atributo son (identificador de acceso, tipo de dato y nombre).

La Ilustración 4.3 muestra como se debe realizar la definición de atributos.

```
$clase @Persona {
    #atributo:
    $publico $entero edad;

    #atributo:
    $privado $entero dni;
}
```

Ilustración 4.3. Definición de Atributos

#### 4.4.4. Definición de Métodos

Para definir un método dentro de una clase, se debe colocar el identificador "`#metodo`" seguido del signo dos puntos `:`, seguido de las características del método. Las características del método son (identificador de acceso, tipo de dato a devolver, nombre del método,

lista de parámetros limitados por paréntesis y su bloque de código correspondiente).

La Ilustración 4.4 muestra como se debe realizar la definición de métodos.

```

$clase @Persona {
    #metodo:
    $publico $nada imprimeEdad() {
        /*Lineas de Código*/
    }

    #metodo:
    $publico $entero duplicaEdad($entero edad) {
        /*Lineas de Código*/
    }
}

```

*Ilustración 4.4. Definición de Métodos*

#### 4.4.5. Declaración de Variables

Para la definición de variables se procede con colocar el tipo de dato seguido de la variable a declarar y finalizar la sentencia con un punto y coma.

Los tipos de datos con los que se cuentan \$entero que hace referencia a números enteros, \$decimal que hace referencia a números reales. Asimismo, una variable es un conjunto de letras alfanuméricas, teniendo como restricción que la letra inicial no puede ser un número.

La Ilustración 4.5 muestra como se debe realizar la declaración de variables.

```

$entero edad;
$entero dni;
$decimal estatura;
$decimal peso;

```

*Ilustración 4.5. Declaración de Variables en BOOP*

#### 4.4.6. Asignación de Valores

Para la asignación de valores, se hará uso del signo igual "=". La Ilustración 4.6 muestra como se debe realizar la asignación de valores a las variables definidas.

```
edad = 22;
dni = 46410647;
estatura = 1.69;
```

*Ilustración 4.6. Asignación de Valores en BOOP*

#### 4.4.7. Definición de Clases

Para definir una clase, se debe colocar el identificador \$clase seguido del nombre de la clase (los nombres de las clases deben empezar con el símbolo arroba "@", seguido del su bloque de código correspondiente ( bloque delimitado por dos llaves "{" y "}").

La Ilustración 4.7 muestra un ejemplo de la definición de clases.

```
$clase @Persona{
    /** SECCIÓN ATRIBUTOS **/
    #atributo:
    $publico $entero x=5;

    /** SECCIÓN MÉTODOS**/
    #metodo:
    $publico $noRetorno miPrimerMetodo(){
        /**CÓDIGO**/
        /**CÓDIGO**/
        /**CÓDIGO**/
    }
}
```

*Ilustración 4.7. Definición de Clase Simple en BOOP*

Asimismo si una clase quiere hacer uso de la herencia, se debe colocar la sección de herencia después del nombre de la clase; esta sección se denota colocando el identificador "\$hereda" seguido de las clases a heredar. La Ilustración 4.8 muestra dos ejemplos de la herencia de clases.

```

$clase @Perro $hereda @Animal{
    /** CÓDIGO **/
    /** CÓDIGO **/
}

$clase @Instructor $hereda @Alumno , @Profesor{
    /** CÓDIGO **/
    /** CÓDIGO **/
}

```

*Ilustración 4.8. Definición de Clases con Herencia*

En caso de querer hacer uso de la implementación de interfaces, se deberá colocar después del nombre de la clase, el identificador "\$implementa" seguido de la interfaz a implementar.

La Ilustración 4.9 muestra dos ejemplos de la implementación de clases:

```

$clase @BoletaDePago $implementa @Imprimible {
    /*Lineas de Código*/
}

$clase @Animal $implementa @Movimiento , @Sonido {
    /*Lineas de Código*/
}

```

*Ilustración 4.9. Definición de Clases con Interfaces*

#### 4.4.8. Llamar a Atributos

Para utilizar un atributo definido dentro de la clase en la que se está trabajando, se tiene que hacer uso del identificador "\$entorno", el cual se acompaña del signo punto "." y es seguido por el nombre del atributo.

Asimismo, en el caso de querer llamar a un atributo perteneciente a un objeto, se nombrará al objeto seguido del signo punto y se colocará el nombre del atributo

La Ilustración 4.10 muestra un ejemplo de la llamada a atributos.



```

@principal {
    #atributo:
    $publico $entero miNumero;

    #ejecutar{
        $entorno.miNumero = 20;
    }
}

```

*Ilustración 4.10. Llamar a Atributo de la Clase*

La Ilustración 4.11 muestra un ejemplo de la llamada a atributos pertenecientes a un objeto.

```

$clase @Persona {
    #atributo:
    $publico $entero edad;
}

@principal {
    #atributo:
    $publico @Persona personal;

    #ejecutar{
        $entorno.personal.edad = 50;
    }
}

```

*Ilustración 4.11. Llamar a Atributo de un Objeto*

#### 4.4.9. Llamar a Métodos

Para realizar una llamada a un método se tiene que hacer uso del identificador "\$llamar", seguido de dos corchetes "[" y "]" que contengan el nombre de la función a llamar.

Para colocar el nombre de la función, se debe utilizar la misma estructura utilizada para el llamado de atributos (uso de \$entorno o llamada desde objeto).

La Ilustración 4.12 muestra un ejemplo de llamada a métodos propios de una clase. Mientras que la Ilustración 4.13 muestra un ejemplo de llamada a métodos pertenecientes a un objeto.

```

@principal {
    #metodo:
    $publico $entero imprimeCabezera(){
        /** CÓDIGO **/
    }

    #ejecutar{
        $llamar[$entorno.imprimeCabezera()];
    }
}

```

*Ilustración 4.12. Llamar a Método de la Clase*

```

$clase @Persona{
    /** CÓDIGO **/
    #metodo:
    $publico $entero imprimeNombre(){
        /** CÓDIGO **/
    }
    /** CÓDIGO **/
}

@principal {
    #ejecutar{
        @Persona miPersona;
        /** CÓDIGO **/
        $llamar[miPersona.imprimeNombre()];
    }
}

```

*Ilustración 4.13. Llamar a Método de un Objeto*

En caso de que el método a llamar retorne un valor, se hará uso del signo igual "=" antes de realizar la llamada al método. La Ilustración 4.14 muestra un ejemplo de llamada a un método con retorno.

```

@principal {
    #metodo:
    $publico $entero calculaSuma($entero a, $entero b){
        $retorna (a+b);
    }

    #ejecutar{
        $entero sumaAB;
        sumaAB = $llamar[$entorno.calculaSuma(10,15)];
    }
}

```

*Ilustración 4.14. Llamar a Método con Retorno*

#### 4.4.10. Operaciones Matemáticas

El lenguaje también cuenta con funciones matemáticas propias. Entre ellas están las funciones básicas, tales como suma, resta, multiplicación, división, división entera, residuo, potencia.

La Ilustración 4.15 muestra un ejemplo del uso de todas las operaciones matemáticas posibles.

```

@principal {
  #ejecutar{
    $entero a;                                $entero b;
    $entero sumaAB;                          $entero restaAB;
    $entero multiplicacionAB; $decimal divisionAB;
    $entero divisionEnteraAB; $entero residuoAB;
    $entero potenciaAB;

    a=12;b=5;

    sumaAB = a+b;                            /** resultado=17 **/
    restaAB = a-b;                          /** resultado=12 **/

    multiplicacionAB = a*b;                 /** resultado=60 **/
    divisionAB= a/b;                       /** resultado=2.4 **/

    divisionEnteraAB = a $divisionEntera b; /** resultado=2 **/
    residuoAB = a $residuo b;              /** resultado=2 **/

    potenciaAB = $potenciar(a,b);          /** resultado=248832 **/
  }
}

```

Ilustración 4.15. Llamar a Método con Retorno

#### 4.4.11. Otras Funcionalidades

Al igual que otros lenguajes de programación, el lenguaje BOOP también cuenta con funcionalidades de apoyo al usuario. A continuación se dará un ejemplo de las principales funciones que posee este lenguaje:

##### 4.4.11.1. Incrementar y Decrementar

Se cuenta con las funciones "\$inc" y "\$dec" cuya

finalidades son aumentar en 1 y disminuir en 1 el valor de una variable dada como parámetro de entrada.

La Ilustración 4.16 muestra un ejemplo del uso de estas funciones.

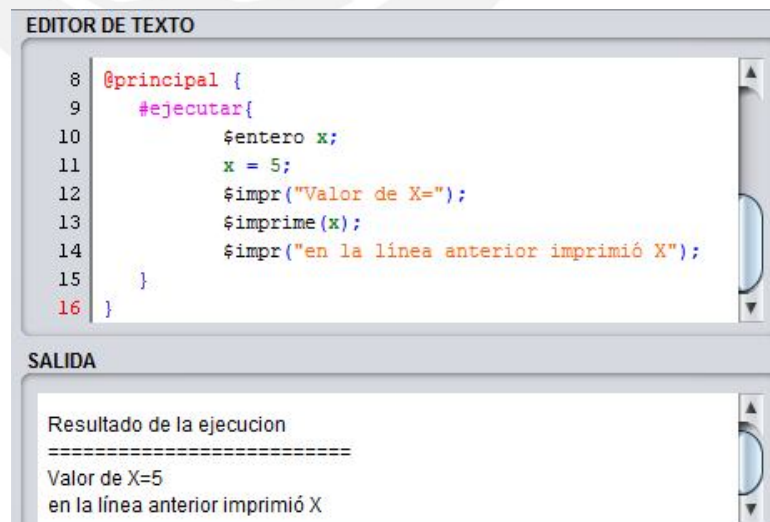
```
@principal {
    #ejecutar{
        $entero a;
        a=5;
        $inc(a); /** a = a + 1 , a = 6 **/
        $dec(a); /** a = a - 1 , a = 5 **/
    }
}
```

*Ilustración 4.16. Incrementar y Decrementar*

#### 4.4.11.2. Métodos de Impresión

Se cuenta con dos funciones, la función "\$impr" que permite imprimir un texto en consola y la función "\$imprime" que permite imprimir un texto en consola y realizar inmediatamente un salto de línea.

La Ilustración 4.17 muestra un ejemplo del uso de las funciones de impresión.



The screenshot shows a text editor window titled "EDITOR DE TEXTO" with the following code:

```
8 @principal {
9     #ejecutar{
10         $entero x;
11         x = 5;
12         $impr("Valor de X=");
13         $imprime(x);
14         $impr("en la línea anterior imprimió X");
15     }
16 }
```

Below the editor is a window titled "SALIDA" (Output) showing the result of the execution:

```
Resultado de la ejecucion
=====
Valor de X=5
en la línea anterior imprimió X
```

*Ilustración 4.17. Métodos de Impresión*

#### 4.4.11.3. Condicionales

Por medio de los identificadores "\$si" y "\$sino" permite condicionar acciones acorde al cumplimiento o no cumplimiento de una sentencia.

La ilustración 4.18 muestra un ejemplo del manejo de condicionales.

```
@principal {
  #ejecutar{
    $entero edad;
    edad = 15;

    $si(edad>=18){
      $imprime("MAYOR DE EDAD");
    }$sino{
      $imprime("MENOR DE EDAD");
    }
  }
}
```

*Ilustración 4.18. Condicional SI-SINO*

#### 4.4.11.4. Iterativas

Con la función "\$mientras" podremos generar bucles asociados a una condición cualquiera, mientras que la función "\$para" es mejor aprovechada al realizar bucles asociados al ascenso o descenso de un número.

La Ilustración 4.19 muestra un ejemplo del manejo de la función \$mientras.

```
@principal {
  #ejecutar{
    $entero puntaje;
    puntaje = 12;
    $mientras(puntaje<=18){
      $imprime("puntaje bajo");
      $inc(puntaje);
    }
  }
}
```

*Ilustración 4.19. Iterativa Mientras*

La Ilustración 4.20 muestra un ejemplo del manejo de la función \$para.

```
@principal {  
    #ejecutar{  
        $entero pasos;  
        $para (pasos=1; pasos<100;$inc(pasos)) {  
            /** CÓDIGO **/  
        }  
    }  
}
```

*Ilustración 4.20. Iterativa Para*



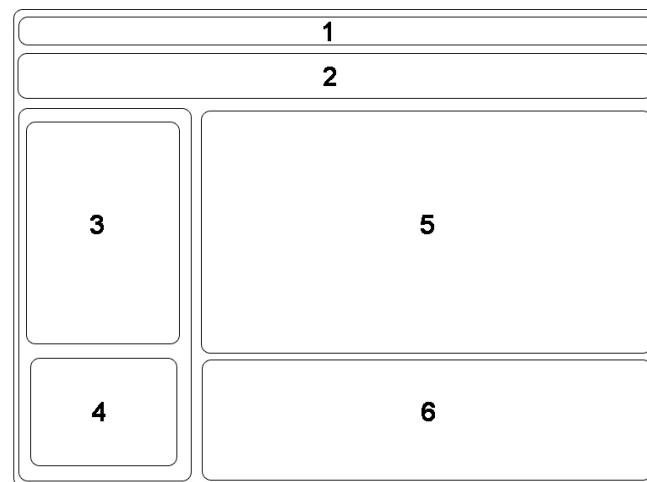


## 5. Diseño del Entorno de Desarrollo

El entorno de desarrollo será una herramienta cuya finalidad será al interacción con el usuario, motivo por el cual se busca crear un ambiente de trabajo “amigable” para el usuario; para conseguir la calificación de ambiente “amigable” se realiza un estudio de arquitectura de la información tomando en cuenta diversas técnicas y logrando así proveer el diseño estructural del producto a generar; cumpliendo así con los objetivos del creador, necesidades de usuarios y los aspectos funcionales y conceptuales asociados al proyecto.

### 5.1. Arquitectura de la Información

Contando como base el estudio de arquitectura de la información (documento adjunto), los resultados del bosquejo se pueden apreciar en la Ilustración 5.1.



*Ilustración 5.1. Bosquejo de zonas*

Describiendo así las siguientes zonas:

#### 5.1.1. Menú de Opciones Generales [ZONA 1]

Contiene opciones generales para un entorno de desarrollo, su estructura debe semejar a la siguiente:

- Archivo

- Nuevo Proyecto
- Abrir Proyecto
- Guardar Proyecto
- Guardar Como
- Imprimir
- Salir
- Editar
  - Cortar
  - Copiar
  - Pegar
  - Buscar
- Opciones
  - Compilar
  - Ejecutar
- Ayuda
  - Gramática
  - Acerca De...

### 5.1.2. Barra de Opciones [ZONA 2]

Contiene una barra de botones representados por imágenes, que permitirá al usuario realizar una acción asociada al presionar uno de éstos. Su estructura debe ser como la siguiente:

- Sección Manejo de Archivo
  - Nuevo Archivo
  - Abrir Archivo
  - Guardar Archivo
  - Guardar en Nuevo Archivo
- Sección Impresión
  - Imprimir
- Sección Opciones de Entorno de Desarrollo
  - Compilar
  - Ejecutar
- Sección Colores

- Paleta de Colores
- Sección Buscador
  - Buscador de palabras (cuadro de texto y botón de búsqueda)

### 5.1.3. Menú de Botones [ZONA 3]

Son las opciones asociadas a los conceptos de programación orientada a objetos, que se reflejarán por medio de botones y cuyo resultado serán la inserción de texto dentro del panel de edición de texto. Su estructura debe semejarse a la siguiente:

- Crear Clase
- Crear Objeto
- Añadir Atributo
- Añadir Método
- Crear Clase Abstracta
- Heredar
- Crear Interfaz
- Implementar Interfaz
- Sobrecribir Método
- Sobrecargar Método

### 5.1.4. Ayudante Virtual [ZONA 4]

Es el ayudante, que por medio de ventanas emergentes, permitirá al usuario conocer las definiciones asociadas a los conceptos de la programación orientada a objetos, así como también emitirá la lectura correspondiente al concepto mostrado.

### 5.1.5. Panel de Edición de Texto [ZONA 5]

Es el panel en el cual el usuario ingresará el texto a procesar por el intérprete.

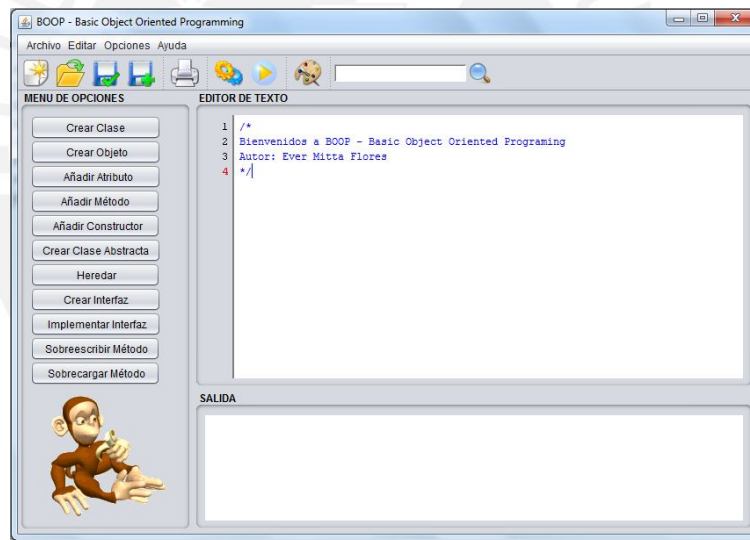
### 5.1.6. Panel de Salida [ZONA 6]

Es el panel en el cual se reflejará los mensajes otorgados por el aplicativo, mensajes tales como descripción de errores y/o finalización de procesos.

## 5.2. Diseño Final

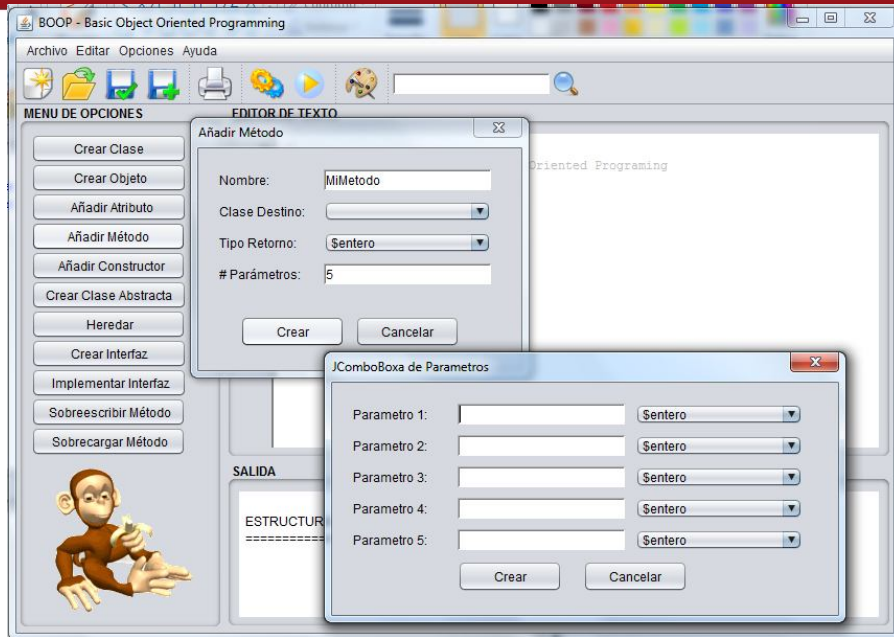
Siguiendo las métricas definidas en el presente documento y por el documento de arquitectura de la información (ver anexo E02) se implementó el entorno de desarrollo cumpliendo con lo establecido. A continuación se listará algunas pantallas del aplicativo:

- La Ilustración 5.2 muestra la pantalla inicial del aplicativo a ejecutar.



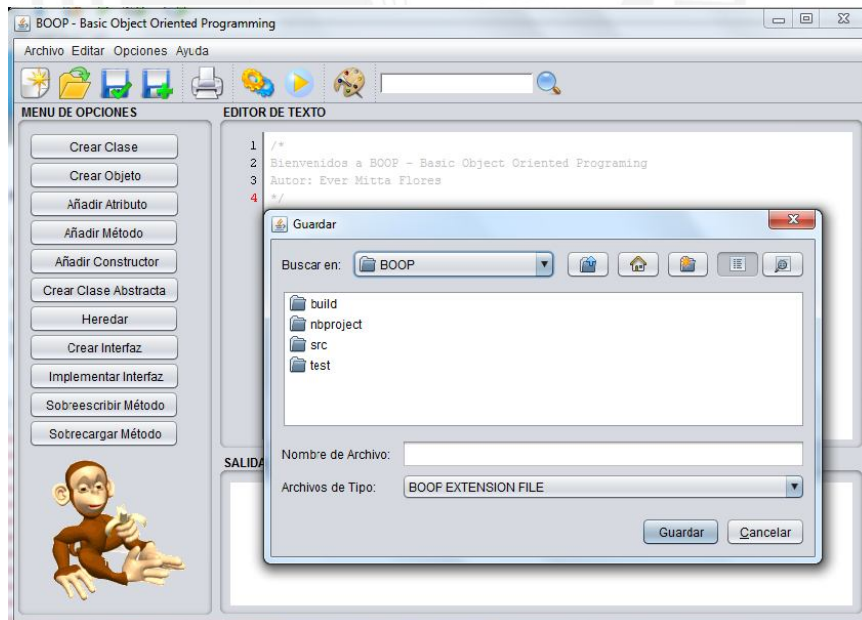
*Ilustración 5.2. Pantalla inicial*

- Se observa en la Ilustración 5.3 el manejo de los mecanismos por medio de botones para la inserción de código.



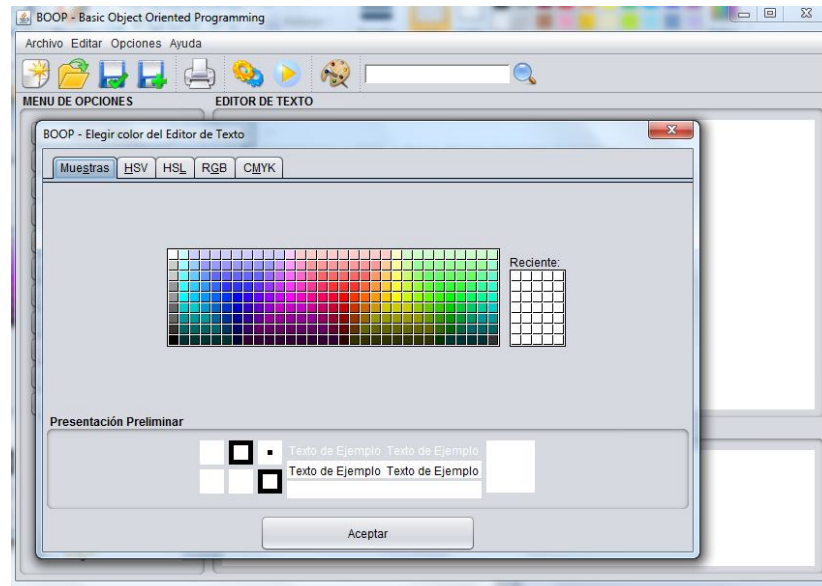
**Ilustración 5.3. Mecanismo de Botones**

- En la Ilustración 5.4 se observa el manejo de archivos, dando así una posibilidad a guardar o abrir archivos de la extensión BOOP.



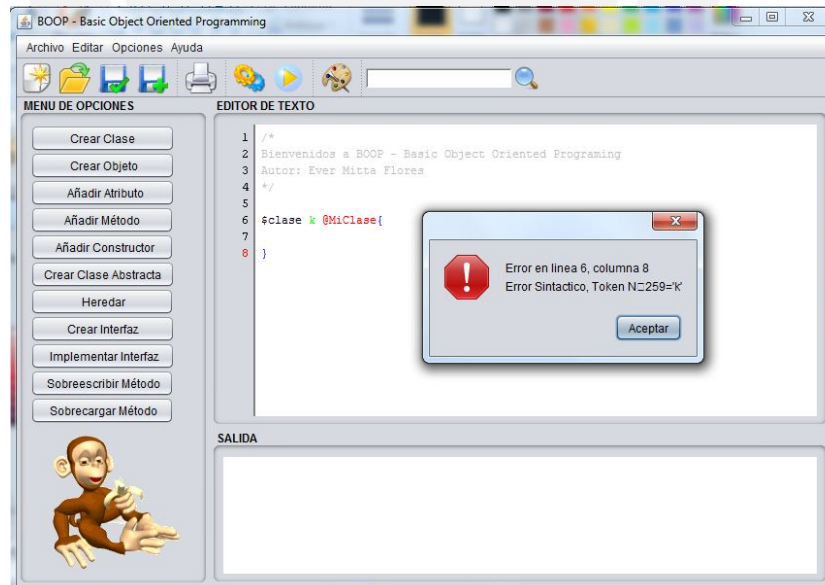
**Ilustración 5.4. Manejo de Archivos**

- Se presenta en la Ilustración 5.5 la posibilidad de colorear la zona de edición de texto, dando así un mayor confort al usuario.



**Ilustración 5.5. Color de Editor de Texto**

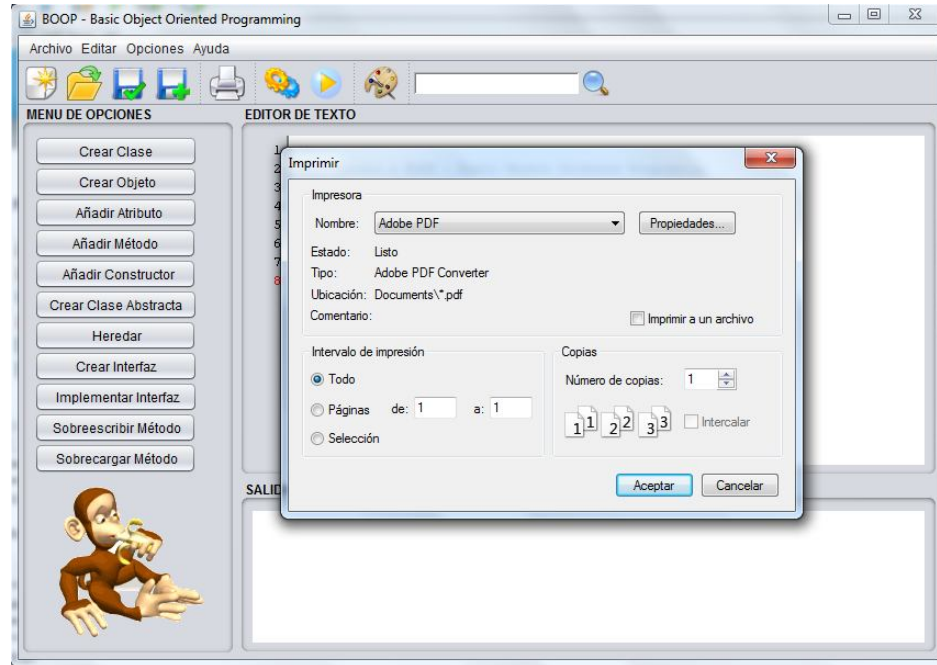
- La Ilustración 5.6 muestra el manejo de errores por parte de intérprete integrado al entorno de desarrollo, éste deberá mostrar al usuario cual es la causa del error así como datos significativos como el número de línea entre otros.



**Ilustración 5.6. Manejo de Errores**



- También se cuenta con la impresión del código trabajado, dando al la posibilidad de analizar el código fuera del entorno de desarrollo. La Ilustración 5.7 muestra un ejemplo de impresión de código.



*Ilustración 5.7. Impresión de Archivo*

## 6. Conclusiones y Recomendaciones

En el presente capítulo se presentan las conclusiones y recomendaciones obtenidas del desarrollo del presente proyecto.

### 6.1. Conclusiones

El presente proyecto tuvo como finalidad producir un lenguaje en español de alto nivel, con una gramática bajo los términos de la programación orientada a objetos. Por otro lado, no se abarcó una amplia gama de conceptos asociados a la programación puesto que se puso énfasis en los conceptos propios del paradigma orientado a objetos.

Asimismo se desarrolla un entorno de desarrollo que permita la interpretación de este lenguaje así como también apoye en el auto-aprendizaje por medio de mecanismos de ayuda. Por consiguiente, el entorno de desarrollo cuenta con mecanismos del tipo botones de inserción de código y un ayudante virtual que sirva como guía para repasar y/o aprender los conceptos relacionados al paradigma orientado a objetos.

La elaboración del proyecto ha sido sin fines de lucro, y con un enfoque de auto-aprendizaje.

### 6.2. Recomendaciones

Futuros trabajos sobre el presente proyecto pueden incluir la adaptación de conceptos asociados a la programación en sí, conceptos tales como arreglos, pilas, colas, entre otros.

Asimismo, en caso de querer usar el producto final y los conocimientos adquiridos de éste documento con un fin pedagógico, se recomienda hacer un estudio de aceptación como herramienta de apoyo al docente.

Finalmente, se resalta la característica de adaptabilidad del producto, lo que da posibilidad realizar cambios pertinentes para usar el entorno de desarrollo como ejecutor de un intérprete asociado a otro paradigma (como el paradigma orientado a eventos, paradigma estructurado, entre otros).



## REFERENCIAS

JOYANES AGUILAR, Luis

2006 *Programación en C++ Algoritmos, estructuras de datos y objetos*. 2da Edición. Madrid: McGraw-Hill/Interamericana de España, S.A.U.

DEITEL &amp; ASSOCIATES, INC.

2005 *Java How To Program*. 6ta Edición. New Jersey: Pearson Education, Inc.

CHUMPITAZ, Lucrecia

2005 *Informática Aplicada a los Procesos de Enseñanza-Aprendizaje*. Primera Edición. Lima: CISE.

MITTA FLORES, Ever

2012 *E01 - Dictado de POO*. Entrevista del 13 de Abril del 2012 a Johan Baldeón.

MITHAT KONAR

2010 *JiDEE :: Java IDE for Education*. Consulta: 11 de Abril del 2012.  
<<http://jidee.tuxfamily.org/>>

UNIVERSITY OF KENT

2010 *BlueJ – Teaching Java*. Consulta: 11 de Abril del 2012.  
<<http://www.bluej.org>>

MORGADE, Marcelo

*JotAzul – IDE Java para Aprendizaje de POO*. Consulta: 12 de Abril del 2012.  
<<http://jotazul.sourceforge.net/>>

RACKET

*Racket*. Consulta: 12 de Abril del 2012  
<<http://racket-lang.org/>>

## RONDA LEÓN, Rodrigo

- 2007 *Revisión de Técnicas de Arquitectura de Información. No Solo Usabilidad Journal. Consulta: 03 de Junio del 2012*  
<[http://www.nosolousabilidad.com/articulos/tecnicas\\_ai.htm](http://www.nosolousabilidad.com/articulos/tecnicas_ai.htm)>

## GRINNELL COLLEGE

- 2009 *Backus-Naur Form. CIS 302, "Programming Language Concepts". Consulta: 03 de Junio del 2012*  
<<http://www.math.grin.edu/~stone/courses/languages/Backus-Naur-form.pdf>>

## HERNÁN, ISIDORO

- 2009 *Dos Herramientas Educativas para el Aprendizaje de Programación: Generación de Comentarios y Creación de Objetos. Madrid, España. Consulta: 03 de Agosto del 2012*  
<<http://www.aipo.es/articulos/4/34.pdf>>

## SANJUR ARAÚZ, DIVA

- 2010 *Una Metodología para el aprendizaje de la Programación Orientada a Objetos basada en Programación Extrema (XP). Panamá. Consulta: 08 de Agosto del 2012.*  
<[http://www.eatis.org/eatis2010/portal/paper/memoria/html/files/sistemas/Diva\\_Sanjur\\_A.pdf](http://www.eatis.org/eatis2010/portal/paper/memoria/html/files/sistemas/Diva_Sanjur_A.pdf)>

## AGILE PROCESS

- 2009 *Extreme Programming : A Gentle Introduction. Consulta: 03 de Junio del 2012*  
<<http://www.extremeprogramming.org/>>

## AHO, A.V

- 2007 *Compilers: Principles, Techniques and Tools. Segunda Edición. Addison Wesley*

## PROJECT MANAGEMENT INSTITUTE (PMI)

- 2008 *Project Management Body Of Knowledge(PMBOK). Cuarta Edición. Pensilvania.*

LEÓN, ALEJANDRO & TORRES, CAROLINA

2007 *Construcción de Ayudas Didácticas para la Enseñanza de Programación Orientada a Objetos usando Java 5.0.*  
Colombia. Consulta: 08 de Agosto del 2012  
<[http://www.uelbosque.edu.co/sites/default/files/publicaciones/revistas/revista\\_tecnologia/volumen5\\_numero2/construccion\\_ayudas\\_didacticas5-2.pdf](http://www.uelbosque.edu.co/sites/default/files/publicaciones/revistas/revista_tecnologia/volumen5_numero2/construccion_ayudas_didacticas5-2.pdf)>





## ANEXOS

1. E01 – Dictado de POO  
Entrevistado: Johan Baldeón.
2. E02 - Documento de Arquitectura de la Información  
Ever Mitta Flores.
3. E03 - Resultado de Ejecución de Pruebas  
Ever Mitta Flores.

