**DISEÑO DE ALTO NIVEL**

**ITERACIÓN 2 TSP**

**STATUS QUO**

**ZAMIR ANDREI GARCÍA ROMERO**

**ÁLVARO DAVID LÓPEZ PINILLA**

**DIEGO ANDRÉS LOZANO ROLDÁN**

**WILMAN RINCON BAUTISTA**

**DEIVIS ENRIQUE VERGEL ARENAS**

**CONCEPTOS AVANZADOS DE INGENIERÍA DE SOFTWARE**

**UNIVERSIDAD DE LOS ANDES**

**2.015**

Tabla de contenido

[1 Diseño general 3](#_Toc416638960)

[2 Representación del diseño 3](#_Toc416638961)

[2.1 Formato operacional 3](#_Toc416638962)

[2.2 Formato Metáfora / Arquitectura 4](#_Toc416638963)

[2.3 Javadoc (Java) 5](#_Toc416638964)

[3 Estándares 6](#_Toc416638965)

[3.1 Estándar de codificación 6](#_Toc416638966)

[3.2 Estándar de conteo de LOC 7](#_Toc416638967)

[4 Inspecciones del diseño 8](#_Toc416638968)

# Diseño general

Como parte de la estandarización de los elementos que se deberán tener en cuenta durante el proceso de construcción, se nombran a continuación los ítems más importantes que se deben conservar en el desarrollo de la aplicación:

1. Elementos básicos en lenguajes OO
   1. Clase
   2. Método
   3. Objeto
   4. Variable
   5. Tipos
2. Conceptos fundamentales OO
   1. Abstracción
   2. Subtipado
   3. Herencia
   4. Dynamic lookup
3. Otros conceptos de lenguajes OO
   1. Polimorfismo
   2. Genéricos
4. Principios
   1. Alta cohesión bajo acoplamiento
   2. Ocultar información
   3. Acceso solo por medio de métodos
5. Patrón Model View Controller (MVC)

# Representación del diseño

Teniendo en cuenta las diferentes representaciones que se le pueden dar al diseño, se escogieron los siguientes formatos para llevar a cabo una representación más adecuada de los requerimientos:

## Formato operacional

|  |  |  |  |
| --- | --- | --- | --- |
| **Grupo** | Status Quo | **Fecha** | 01/04/2015 |
| **Programa** | Complejidad ciclomática | **Programa #** | 1 |
| **Instructor** | Luis Benavides | **Lenguaje** | Java |

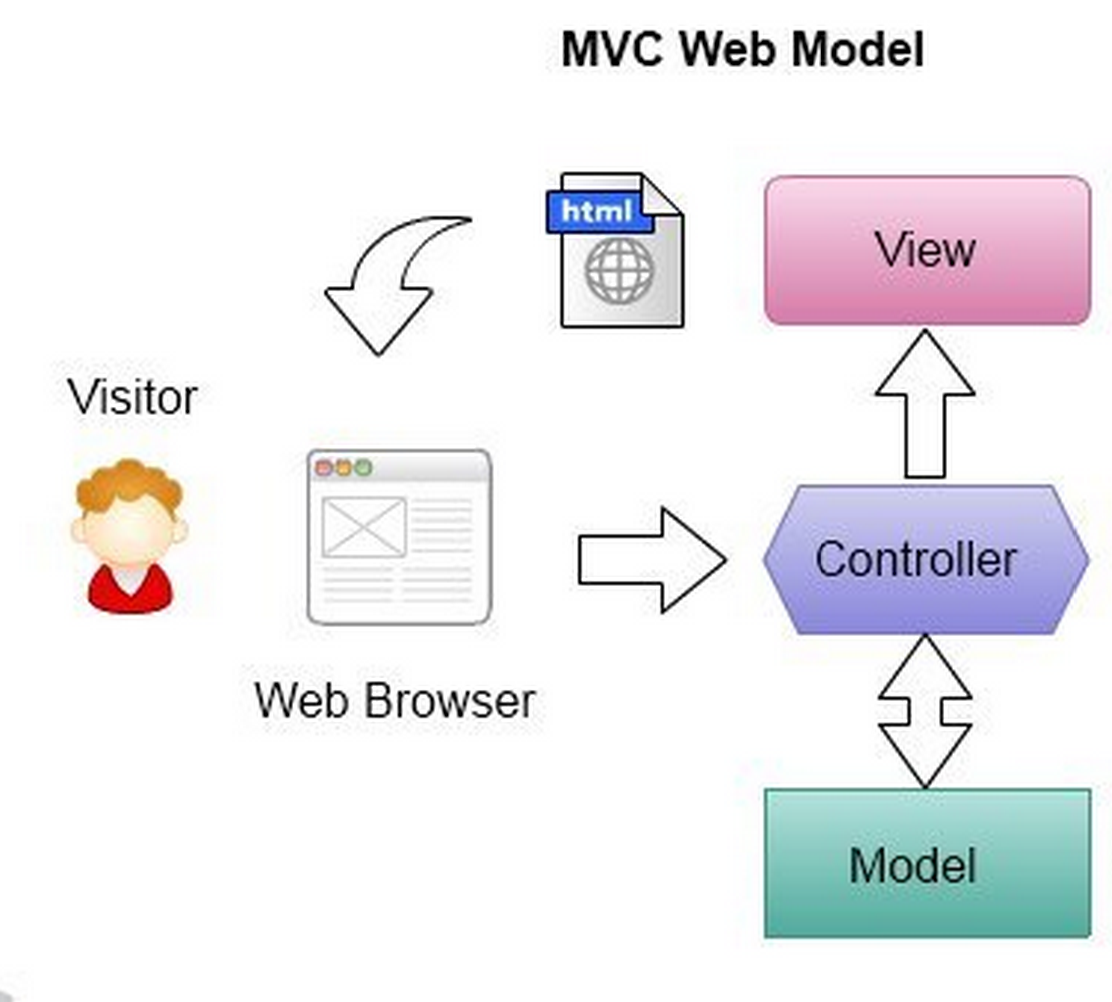
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Escenario numero** | **1** | **Usuario objetivo** | Usuario funcional | |
| **Objetivo del escenario** | | Realizar una prueba funcional correcta del sistema, calculando la cantidad de líneas LOC y los detalles de una aplicación suministrada. | | |
| **Recurso** | **Paso** | **Acción** | | **Comentarios** |
| Usuario | 1 | Ingresar a la pagina | |  |
| Sistema | 2 | Muestra la página inicial | |  |
| Usuario | 3 | Selecciona el proyecto que desea calcular | |  |
| Sistema | 4 | Muestra la información calculada del proceso | | Realiza los conteos de acuerdo al estándar suministrado |

## Formato Metáfora / Arquitectura

|  |  |  |  |
| --- | --- | --- | --- |
| Grupo | Status Quo | Date | 01/04/2015 |
| Programa | Complejidad ciclomática | Programa # | 1 |
| Instructor | Luis Daniel Benavides Navarro | Lenguaje | JAVA |

|  |  |
| --- | --- |
| **Referencias del** | Este programa se basa en el patrón MVC. Los requerimientos establecen un cálculo en |
| **Diseño** | la complejidad ciclomática usando el método de McGabe y cálculos de LOC de |
|  | programas suministrados. |

**Representación gráfica de la metáfora**

****

**Representación textual de la metáfora**

La aplicación debe tendrá una arquitectura basada en el patrón MVC, e implementa tres capas dispuestas según el proceso lógico que debe ejecutar cada componente para mantener principios de seguridad, encapsulamiento y rehusó entre otros, el usuario interactúa con dos de las capas, con una vista que provee todos los componentes gráficos necesarios para su interacción y con un controlador que atiende todas las peticiones del usuario realizadas desde la vista por medio de las acciones propuestas por el comportamiento de cada acción , desde este punto el controlador se encargara de gestionar la petición e invocar a la capa de modelo que contiene la lógica de comportamiento de la funcionalidad, y retornar los datos a través del controlador para que sean interpretados y graficados por la vista de nuevo.

## Javadoc (Java)

El javadoc se encuentra generado en el árbol del proyecto.

# Estándares

Los siguientes son los estándares que se deberán mantener durante todo el proceso de desarrollo:

## Estándar de codificación

|  |  |
| --- | --- |
| Purpose | * To guide implementation of Java programs |
| Program Headers | Begin all programs with a descriptive header. |
| Header Format | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* Program Assignment: the program number \*/  /\* Name: your name \*/  /\* Date: the date you started developing the program \*/  /\* Description: a short description of the program and what it does \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| Identifiers | Use descriptive names for all variable, function names, constants, and other identifiers. Avoid abbreviations or single-letter variables. |
| Identifier Example | Int number\_of\_students; /\* This is GOOD \*/  Float: x4, j, ftave; /\* This is BAD \*/ |
| Comments | * Document the code so the reader can understand its operation. * Comments should explain both the purpose and behavior of the code. * Comment variable declarations to indicate their purpose. |
| Good Comment | If(record\_count > limit) /\* have all records been processed? \*/ |
| Bad Comment | If(record\_count > limit) /\* check if record count exceeds limit \*/ |
| Major Sections | Precede major program sections by a block comment that describes the processing done in the next section. |
| Example | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* The program section examines the contents of the array ‘grades’ and calcu- \*/  /\* lates the average class grade. \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| Blank Spaces | * Write programs with sufficient spacing so they do not appear crowded. * Separate every program construct with at least one space. |
| Indenting | * Indent each brace level from the preceding level. * Open and close braces should be on lines by themselves and aligned. |
| Indenting Example | * while (miss\_distance > threshold) * { * success\_code = move\_robot (target \_location); * if (success\_code == MOVE\_FAILED) * { * printf(“The robot move has failed.\n”); * } * } |
| Capitalization | * Capitalize all defines. * Lowercase all other identifiers and reserved words. * To make them readable, user messages may use mixed case. |
| Capitalization Examples | #define DEFAULT-NUMBER-OF-STUDENTS 15  int class-size = DEFAULT-NUMBER-OF-STUDENTS; |

## Estándar de conteo de LOC

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre: | LOC java estándar | Lenguaje: | JAVA |
| Autor: | Status Quo | Fecha: | 04/04/2015 |

|  |  |  |
| --- | --- | --- |
| **Count Type** | **Type** | **Comments** |
| Physical/Logical | logical |  |
| **Statement Type** | **Included** | **Comments** |
| Executable | si |  |
| Nonexecutable: |  |  |
| Declarations | Si, Nota 1 | Private String var1; |
| Compiler Directives | No, Nota 3 | Import package |
| Comments | No,Nota 2 | // /\* \*/ |
| Blank lines | no |  |
| **Clarifications** |  | **Examples/Cases** |
| Espacios en blanco | no |  |
| Inicios de deciciones ,ciclos , o metodos | si | Se incluyen los inicios de ciclo CASE, DO, ELSE, ENUM, FOR, IF, SWITCH, WHILE etc, el inicio debe estar en la misma línea “{” en caso contrario se contaran los dos como inicio |
| fin de decisiones o ciclos | no | No se contaran los finales de ciclo que estén solos en una línea así “}” |
| Sentencias multilineas | si | Un métodos o sentencia multilinea será contada individualmente |
| Clases Embebidas | no | Se debe crear una clase por archivo .java |
| Note 1 |  | Se identifica como atributo de una clase una declaración por línea , si son del mismo tipo se deberá hacer en una línea independiente |
| Note 2 |  | Se excluyen los comentarios realizados en una línea completa con los símbolos “//” y los comentarios multilinea con los símbolos “/\*” y “\*/”, y las líneas comentarías intermedias siempre que comiencen por el símbolo “\*” |
| Note 3 |  | No se incluyen las líneas de imports y package |

# Inspecciones del diseño

El siguiente es el formato para la inspección del diseño:

|  |  |  |  |
| --- | --- | --- | --- |
| Group | Status Quo | Fecha | 04/04/2015 |
| Programa | ECOS | Programa # | 1 |
| Instructor | Luis Daniel Benavides Navarro | Lenguaje | JAVA |

|  |  |
| --- | --- |
| Purpose | To guide you in conducting an effective design review |
| General | * Review the entire program for each checklist category; do not attempt to review for more than one category at a time! * As you complete each review step, check off that item in the box at the right. * Complete the checklist for one program or program unit before reviewing the next. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Complete | Verify that the design covers all of the applicable requirements.   * All specified outputs are produced. * All needed inputs are furnished. * All required includes are stated. |  |  |  |  |
| External Limits | Where the design assumes or relies upon external limits, determine if behavior is correct at nominal values, at limits, and beyond limits. |  |  |  |  |
| Logic | * Verify that program sequencing is proper.   Stacks, lists, and so on are in the proper order.  Recursion unwinds properly.   * Verify that all loops are properly initiated, incremented, and terminated. * Examine each conditional statement and verify all cases. |  |  |  |  |
| Internal Limits | Where the design assumes or relies upon internal limits, determine if behavior is correct at nominal values, at limits, and beyond limits. |  |  |  |  |
| Special Cases | * Check all special cases. * Ensure proper operation with empty, full, minimum, maximum, negative, and Cero values for all variables. * Protect against out-of-limits, overflow, and underflow conditions. * Ensure “impossible” conditions are absolutely impossible. * Handle all possible incorrect or error conditions. |  |  |  |  |
| Functional Use | * Verify that all functions, procedures, or methods are fully understood and properly used. * Verify that all externally referenced abstractions are precisely defined. |  |  |  |  |
| System Considerations | * Verify that the program does not cause system limits to be exceeded. * Verify that all security-sensitive data are from trusted sources. * Verify that all safety conditions conform to the safety specifications. |  |  |  |  |
| Names | Verify that   * all special names are clear, defined, and authenticated * the scopes of all variables and parameters are self-evident or defined * all named items are used within their declared scopes |  |  |  |  |
| Standards | Ensure that the design conforms to all applicable design standards. |  |  |  |  |