Avance 1

Andres, Jimenez
andresjimqui@gmail.com

Ricardo, Castro
ricardocastrom06@gmail.com

Jean Carlo, Mata
jean.cms07@gmail.com

September 25, 2016

1 Requerimientos

2 Estandar de codificacion a utilizar a lo largo del proyecto semestral

2.1 Nomenclatura

En esta seccion definiremos la forma de dar sentido al nombre de clases, variables, constantes, entre otros. Todo estara escrito en ingles incluyendo comentarios.

2.1.1 Paquetes

Los paquetes se escribirán en minúsculas y sin utilizar caracteres especiales.

2.1.2 Interfaces

Los nombres de interfaces utilizarán el profijo Interface y estarán compuestos por palabras con la primera letra en mayúscula. Deben ser mezclas de mayúsculas y minúsculas, con la primera letra de cada palabra interna en mayúsculas. Se debe evitar el uso de abreviaciones que dificulten la comprensión del código y el objetivo de la interfaz.

Por ejemplo: InterfaceServerConnection

2.1.3 Clases

Los nombres de clases deben ser mezclas de mayúsculas y minúsculas, con la primera letra de cada palabra interna en mayúsculas. Debemos intentar mantener los nombres de clases simples y descriptivos.

Por ejemplo: FigureCircle

2.1.4 Métodos

Los métodos deberán ser verbos (en infinitivo), en mayúsculas y minúsculas con la primera letra del nombre en minúsculas, y con la primera letra de cada

palabra interna en mayúsculas. No se permiten caracteres especiales. El nombre ha de ser lo suficientemente descriptivo.

Por ejemplo: calcPerimeter()

2.1.5 Variables

Los nombres de las variables tanto de instancia como estáticas reciben el mismo tratamiento que para los métodos. Las variables deberán en mayúsculas y minúsculas con la primera letra del nombre en minúsculas, y con la primera letra de cada palabra interna en mayúsculas. No se permiten caracteres especiales. El nombre ha de ser lo suficientemente descriptivo.

Por ejemplo: redCircle

2.1.6 Constantes

Los nombres de constantes deben escribirse todas en mayúsculas con las palabras separadas por underscore. Todas serán declaradas como private static final.

Por ejemplo: private static final `RED_CIRCLE`

2.2 Codificacion

2.2.1 Comentarios

Los comentarios serán utilizados para dar información adicional al desarrollador/es sobre la implementación del diseño de la clase, metodo, interfaz, parametro, entre otros. Se tiene, por lo tanto, que evitar referencias al diseño funcional de la misma.

El uso abusivo de los comentarios es desaconsejable, principalmente por el trabajo extra necesario para su correcto mantenimiento.

Se tienen que evitar el uso de caracteres especiales dentro de los comentarios.

2.2.2 Declaraciones

Para la declaración de las variables se utiliza una declaración de cada vez y no se permiten dejar variables locales sin inicializar.

Por ejemplo: private Circle = new Circle();

2.3 Motivo de eleccion del estandar propuesto

El estandar propuesto ha sido elegido para mantener

2.3.1 Sentencias

- Una sentencia por línea de código.

- Todo bloque de sentencias entre llaves, aunque sea una sola sentencia después de un if.

Por ejemplo: for(int salary = 0; salary ¡ 3900000; salary++)

2.4 Notas. Buenas practicas de programacion

2.4.1 Constantes

Como norma general todas las constantes numéricas no deberían codificarse directamente, salvo la excepción de -1, 0 y 1.

2.4.2 Propiedades

El acceso/modificación de las propiedades de una clase (no constantes) siempre mediante métodos de acceso get/set.

La asignación de variables / propiedades no podrá ser consecutiva.

salary1 = salary2 = "1000000" [No válido]

No utilizar el operador asignación en sitios donde se pueda confundir con el operador igualdad. Ni dentro de expresiones complejas.

2.4.3 Métodos

No se debe acceder a un método estático desde una instancia de una clase, debemos utilizar la clase en sí misma.

2.5 Motivo por el que seleccionamos este estandar

La decision de implementar este estandar de codificacion, mas alla de contar como parte de la evaluacion del avance es una excelente forma de garantizar la futura mantenibilidad del codigo, ya sea por parte de nosotros mismos o diferentes grupos de trabajo. Esto porque existen mediciones/estudios que dicen que un 80% de la vida util de una pieza de software es necesario realizarle algun tipo de mantenimiento. Tambien el estandar le ofrece al codigo la posibilidad de tener una mejor legibilidad y mejor comprension. Dicho estandar propuesto es muy comun verlo en la mayoria de aplicaciones desarrolladas en Java, C y JavaScript, siendo esta una de las principales razones que nos llevo a elegirlo.

2.6 Herramienta para verificar el estandar de codificacion propuesto

Para la comprobacion del estandar propuesto se utilizara la herramienta llamada checkstyle.

Checkstyle automatiza el proceso de verificacion de codigo en Java evitando el realizar esta tarea manualmente, la cual podria es cansada y tediosa.

Es altamente configurable y esta hecha para soportar casi todo estandar de codificacion.

3 Diagrama de componentes y digramas UML para la primera iteracion de la arquitectura del sistema

3.1 Diagrama de componentes

3.2 Diagramas UML

4 Actividades de Aseguramiento de la calidad a realizar, una vez finalizado cada sprint

El standar de IEE 730-2002 basicamente nos recomienda elaborar un plan para el Aseguramiento de la Calidad del Software.

4.0.1 Tareas

4.0.2 Crear plan SQA

- 4.0.2.1 Desarrollar y documentar el plan de SQA
- 4.0.2.2 Aprobar plan SQA

4.0.3 Evaluar los requrimientos

- 4.0.3.1 Analizar requerimientos
- 4.0.3.2 Proponer cambios a los requerimientos, si se considera necesario
- 4.0.3.3 Proponer posibles soluciones/escenarios para resolver los requerimientos

4.0.4 Evaluar herramientas utilizadas

- 4.0.4.1 Evaluar heramientas. Verificar resultados generados.
- 4.0.4.2 Proponer nuevas herramientas o seguir utilizando las mismas

4.0.5 Unit Testing

- 4.0.5.1 Mejoras en test unitarios
- 4.0.5.1 Realizar nuevos test para las nuevas funcionalidades

4.0.6 Evaluar la Administracion de la configuracion

- 4.0.6.1 Configurar herramientas/ambientes involucradas en el sistema

4.0.7 Resolucion de errores

- 4.0.7.1 Identificar problema
- 4.0.7.2 Analizar soluciones
- 4.0.7.3 Verificar impacto de las soluciones propuestas

4.0.8 Matriz de de responsabilidades

Tarea	Jean Carlo	Andres	Ricardo
4.0.2.1		x	
4.0.2.2	x	x	x
4.0.3.1	x	x	
4.0.3.2	x	x	
4.0.3.3	x	x	x
4.0.4.1	x		
4.0.4.2			x
4.0.5.1	x		
4.0.5.1	x		
4.0.6.1	x		x
4.0.7.1		x	x
4.0.7.2	x		x
4.0.7.3	x	x	x