cujae_logoUniversidad Tecnológica de La Habana “José Antonio Echeverría”

**Seminario de Ingeniería de Software III.**

**“Desarrollo de un Sistema Informático para la automatización del Calendario Nacional de Béisbol.”**

“Rol Analista-Diseñador.”

**Autor: Diana Pérez Abreu**

**Facultad de Ingeniería Informática.**

**Fecha: 16 de marzo de 2020**

Índice

[1. Breve Descripción del negocio: 1](#_Toc35181023)

[2. Requerimientos del proyecto: 1](#_Toc35181024)

[1.1 Requisitos funcionales 1](#_Toc35181025)

[1.2 Requisitos no funcionales 2](#_Toc35181026)

[Descripción de los requisitos no funcionales. 3](#_Toc35181027)

[3. Diagrama de casos de uso del sistema 4](#_Toc35181028)

[4. Diagrama de clases 5](#_Toc35181029)

[5. Vista de la arquitectura de los casos de uso 5](#_Toc35181030)

[6. Casos de prueba 10](#_Toc35181031)

[7. Estándar de codificación 20](#_Toc35181032)

[8. Arquitectura candidata. 26](#_Toc35181033)

[8.1 NetBeans 26](#_Toc35181034)

[8.2 IntelliJ IDEA 27](#_Toc35181035)

[8.3 Eclipse 27](#_Toc35181036)

# Breve Descripción del negocio:

Se desea crear una aplicación para generar de manera automatizada los calendarios de la Serie Nacional de Béisbol en Cuba que permita ahorrar tiempo y combustible en comparación a como se realiza actualmente. Para dar solución a lo mencionado anteriormente se debe generar un calendario inicial que sea factible y luego se puedan realizar cambios o mutaciones a este calendario para poco a poco mejorar el resultado final. Un calendario factible es aquel donde se cumplen las siguientes restricciones: no puede existir un equipo que se esté enfrentando con dos equipos en una misma fecha, y, un enfrentamiento entre dos provincias solo puede efectuarse una vez. El nivel de calidad entre dos calendarios se medirá por la cantidad de kilómetros recorridos en total por todos los equipos durante los desplazamientos hacia las diferentes ciudades de Cuba.

# Requerimientos del proyecto:

La ingeniería de requerimientos trata de establecer lo que el sistema debe hacer, sus propiedades emergentes deseadas y esenciales, y las restricciones en el funcionamiento del sistema y los procesos de desarrollo de software. Por lo tanto, se debe considerar a la ingeniería de requerimientos como el proceso de comunicación entre los clientes, usuarios del software y los desarrolladores del mismo. La captura de requisitos es la actividad mediante la que el equipo de desarrollo de un sistema de software extrae, de cualquier fuente de información disponible, las necesidades que debe cubrir dicho sistema.

El objetivo fundamental del proyecto a realizar es la implementación de una aplicación que permita generar el calendario de la Serie Nacional de Béisbol de Cuba utilizando metaheurísticas. A continuación, se mencionan los requerimientos funcionales y no funcionales de la aplicación.

## Requisitos funcionales

Tabla 1 Requisitos funcionales

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  | **Interesados que lo solicitaron** |
| ID | **Requisito** | **Prioridad** | Cliente (Dr. Alejandro Rosete) |
| R1 | Generar calendario inicial. | 3 | X |
| R2 | Seleccionar el tipo de calendario (a dos vueltas o no). | 2 | X |
| R3 | Seleccionar el tipo del primer enfrentamiento (entre el campeón y subcampeón de la serie pasada o no). | 2 | X |
| R4 | Implementar mutaciones que permitan la variación del calendario inicial. | 3 | X |
| R5 | Extraer datos de los equipos desde un fichero externo. | 3 | X |
| R6 | Crear matriz de distancias entre provincias. | 3 | X |
| R7 | Calcular la distancia total de un calendario. | 3 | X |
| R8 | Comparar calendario según su distancia total. | 2 | X |
| R9 | Validar que un equipo no puede aparecer en dos enfrentamientos en la misma fecha. | 3 | X |
| R10 | Diferenciar entre equipos locales y visitantes. | 2 | X |

## Requisitos no funcionales

Los requisitos no funcionales son cualidades o propiedades que el producto a desarrollar debe tener. Los requerimientos no funcionales también añaden funcionalidad al producto, pues hacen que un producto sea fácil de usar, seguro, o interactivo.

A continuación, se listan los requisitos no funcionales identificados, así como la trazabilidad entre estos y algunos problemas frecuentes en los sistemas de mediana y alta complejidad:

Tabla 2 Requisitos no funcionales y problemas frecuentes

|  |  |
| --- | --- |
| Requisitos no funcionales | Problemas frecuentes |
| Apariencia o interfaz externa | Control de la navegación del usuario. |
| Control de los eventos generados por el usuario. |
| Validación |
| Interfaz interna | Estructuración y subdivisión del sistema en partes más pequeñas. |
| Usabilidad | Control de la navegación del usuario. |
| Control de los eventos generados por el usuario. |
| Tratamiento de excepciones |
| Confiabilidad | Tolerancia a fallos |

### Descripción de los requisitos no funcionales.

* Interfaz interna:

Organizar y subdividir el código de la aplicación en paquetes dependiendo de la funcionalidad, respetando las buenas prácticas, para que la probabilidad de que ocurra un error que afecte el desarrollo y ejecución de la aplicación sea mínima, así como permitir una mayor reutilización del código. Además, el software debe implementar diversos aspectos, los cuales se desarrollan de manera modular, es decir, cada uno separado por su bloque de código correspondiente, lo que permite una mayor organización y usabilidad del código.

* Apariencia o interfaz externa:

El usuario debe ser capaz de conocer en todo momento en qué parte del sistema se encuentra. Además, la aplicación debe ser interactiva y simple de usar, sin mucho nivel de complejidad. El usuario debe ser capaz de visualizar el resultado de todas las acciones que se ha realizado sobre el calendario inicial o el calendario que ha sufrido modificaciones.

* Usabilidad:

El sistema puede ser utilizado por personas con o sin conocimientos básicos de inteligencia artificial e informática por lo que la interfaz de usuario debe carecer de complejidad; brindando facilidad de uso para todo tipo de personas que hagan uso de esta. Por otra parte, el sistema debe manejar las excepciones, con el objetivo de que un error inesperado no sorprenda al usuario.

* Confiabilidad:

El sistema debe notificar de forma breve y concisa si el usuario ha realizado una acción errónea, con el fin de que este sea capaz de conocer el error cometido y si puede o no corregirlo.

# Diagrama de casos de uso del sistema

El modelo de casos de uso establece un acuerdo entre clientes y desarrolladores sobre las condiciones y posibilidades (requisitos) que debe cumplir el sistema, es decir, sobre las condiciones y posibilidades que debe cumplir el sistema y proporciona la entrada fundamental para el análisis, el diseño y las pruebas.

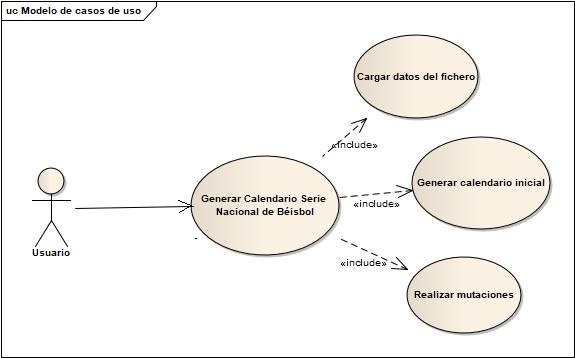


Figura 1 Diagrama de casos de uso del sistema

# Diagrama de clases

El objetivo principal de este modelo es la representación de los aspectos estáticos del sistema, utilizando diversos mecanismos de abstracción (clasificación, generalización y agregación). El diagrama de clases recoge las clases de objetos y sus asociaciones. En este diagrama se representa la estructura y el comportamiento de cada uno de los objetos del sistema y sus relaciones con los demás objetos.

A continuación, se muestra en la Figura 1, el diagrama de diseño de clases que representa las entidades del sistema y las relaciones entre ellas.

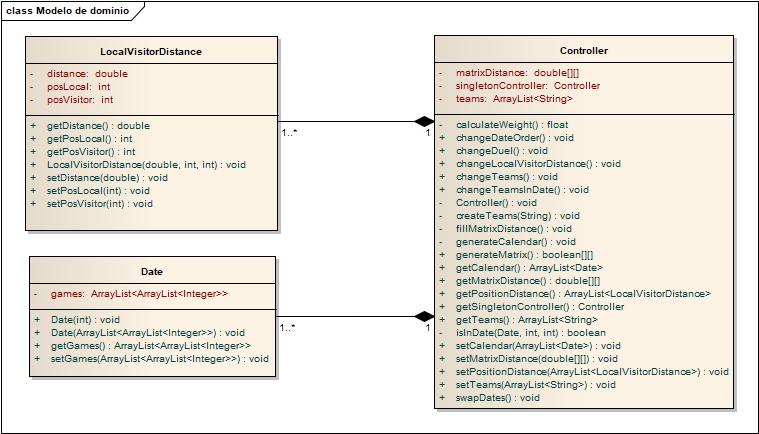


Figura 2 Diagrama de Clases

# Vista de la arquitectura de los casos de uso

En la realización de este proyecto se consideran como casos de uso de mayor importancia para el usuario y significativo para la arquitectura del sistema todos los casos de uso mostrados en el diagrama de casos de uso del sistema: **Generar Calendario Serie Nacional de Béisbol, Cargar datos del fichero, Generar calendario inicial y Realizar mutaciones**. Este se clasifica de acuerdo a su impacto en la arquitectura como crítico, debido a que cubre las principales tareas y funciones que el sistema ha de realizar; define la arquitectura básica del sistema.

**Especificación de alto nivel de los casos de usos**

Tabla 3 Especificación de alto nivel : Generar Clendario Serie Nacional de Béisbol

|  |  |
| --- | --- |
| Generar Calendario Serie Nacional de Béisbol |  |
| Actores | Usuario |
| Propósito | Generar el calendario de la Serie Nacional de Béisbol con la menor cantidad de kilómetros posibles. |
| Descripción | El caso de uso comienza cuando el usuario lee los datos del fichero. A partir de los datos leídos  se crea una matriz que almacena las distancias entre cada una de las provincias y con esta se genera un calendario inicial. Luego de esto, se selecciona un tipo de mutación y se le aplica a dicho calendario. Posteriormente se calculan las distancias del calendario nuevo y la del calendario inicial. Una vez obtenidas ambas distancias, estas son comparadas. Si la distancia del calendario nuevo es menor que la distancia del calendario inicial se selecciona el calendario nuevo como mejor opción hasta el momento. Luego de todo este proceso, se aplican nuevamente mutaciones al calendario que esté etiquetado como la mejor opción hasta el momento; y así sucesivamente hasta llegar a al número máximo de iteraciones establecidas. |
| Requisitos funcionales | * Generar calendario inicial que sea factible. * Seleccionar el tipo de calendario (a dos vueltas o no). * Seleccionar el tipo del primer enfrentamiento (entre el campeón y subcampeón de la serie pasada o no). * Implementar mutaciones que permitan la variación del calendario inicial. * Extraer datos de los equipos desde un fichero externo. * Crear matriz de distancias. * Calcular la distancia total de un calendario. * Comparar calendario según su distancia total. * Validar que un equipo no puede aparecer en dos enfrentamientos en la misma fecha. * Diferenciar entre equipos locales y visitantes. |
| Casos de uso asociados | * Cargar datos del fichero. * Generar calendario inicial. * Realizar mutaciones |
| Precondiciones | - |
| Requisitos no funcionales | Usabilidad, Apariencia o interfaz externa, Rendimiento, Portabilidad |
| Postcondiciones | Se obtiene el Calendario de la Serie Nacional de Béisbol que se puede desarrollar recorriendo la menor cantidad de kilómetros. |

Tabla 4 Especificación de alto nivel: Cargar datos del fichero

|  |  |
| --- | --- |
| Cargar datos del fichero |  |
| Actores | Usuario |
| Propósito | Cargar las distancias entre provincias. |
| Descripción | El caso de uso comienza cuando el usuario extrae los datos de un fichero (nombre provincia1, nombre provincia2, distancia(km) entre dichas provincias) para crear los equipos y asignar las distancias correspondientes en una matriz cuadrada donde todos los equipos se posicionarán en filas y columnas, y su intersección será el valor de la distancia entre ellos. |
| Requisitos funcionales | * Extraer datos de los equipos desde un fichero externo. * Crear matriz de distancias. |
| Casos de uso asociados | - |
| Precondiciones | - |
| Requisitos no funcionales | Usabilidad, Apariencia o interfaz externa |
| Postcondiciones | - |

Tabla 5 Especificación de alto nivel: Generar calendario inicial

|  |  |
| --- | --- |
| Generar calendario inicial |  |
| Actores | Usuario |
| Propósito | Generar un calendario inicial factible para la Serie Nacional de Béisbol que se pueda ir modificando posteriormente a través de mutaciones. |
| Descripción | A partir de una matriz cuadrada triangular superior, se recorren las filas y se van almacenando dos enfrentamientos que no incumplan con las restricciones (no puede existir un equipo que se esté enfrentando con dos equipos en una misma fecha, y, un enfrentamiento entre dos provincias solo puede efectuarse una vez). |
| Requisitos funcionales | * Generar calendario inicial que sea factible. * Seleccionar el tipo de calendario (a dos vueltas o no). * Seleccionar el tipo del primer enfrentamiento (entre el campeón y subcampeón de la serie pasada o no). * Validar que un equipo no puede aparecer en dos enfrentamientos en la misma fecha. * Diferenciar entre equipos locales y visitantes. |
| Casos de uso asociados | - |
| Precondiciones | Se deben haber extraído todos los datos correspondientes a los equipos y sus distancias del fichero. |
| Requisitos no funcionales | - |
| Postcondiciones | .Se obtiene un calendario inicial factible para la Serie Nacional de Béisbol que puede que no sea la mejor opción. |

Tabla 6 Especificación de alto nivel: Realizar mutaciones

|  |  |
| --- | --- |
| Realizar mutaciones |  |
| Actores | Usuario |
| Propósito | Variar el calendario inicial de la Serie Nacional de Béisbol para buscar la mejor opción. |
| Descripción | El caso de uso comienza cuando el usuario selecciona un calendario y le aplica a este una mutación que es seleccionada de manera random del total de mutaciones diseñadas. |
| Requisitos funcionales | * Implementar mutaciones que permitan la variación del calendario inicial. |
| Casos de uso asociados | - |
| Precondiciones | Se deben haber extraído todos los datos correspondientes a los equipos y sus distancias del fichero; y haber generado el calendario inicial. |
| Requisitos no funcionales | - |
| Postcondiciones | Se obtiene el calendario que se tiene almacenado como mejor opción hasta el momento modificado. |

# Estilos y patrones de arquitectura

Los estilos arquitectónicos no son más que un conjunto de principios. Cada estilo define un conjunto de normas que especifica los tipos de componentes que puede utilizar para montar un sistema, las limitaciones sobre la forma en que se ensamblan y las hipótesis sobre el significado de cómo ponerlos juntos.

Por otra parte, los patrones arquitectónicos expresan el esquema de organización estructural fundamental para sistemas de software. Son plantillas para arquitecturas de software concretas, que especifican las propiedades estructurales de una aplicación. Por lo tanto, la selección de los patrones arquitectónicos son una decisión fundamental en el desarrollo de una aplicación.

En el proyecto se identificaron los siguientes estilos y patrones de arquitectura:

|  |  |
| --- | --- |
| Estilo | Patrón |
| Flujo de datos de entrada | Filtros y tuberías |
| Llamada y retorno | N-Capa |

### 2.4.1 Justificación de los estilos y patrones de arquitectura propuestos para el proyecto.

A continuación, se definen y justifican los diferentes estilos y patrones arquitectónicos propuestos para la realización del proyecto, mencionados anteriormente:

#### Flujo de datos de entrada: Filtros y tuberías.

En estilo de flujos de entrada, el dato ingresa en el sistema y fluye entre los componentes, de uno en uno, hasta que se le asigne un destino final. Este estilo arquitectónico provee la estructura y los mecanismos para los sistemas que deben procesar flujos de datos. Se basa en un patrón de tuberías y filtros que consta de un conjunto de componentes denominados “filtros” conectados entre sí por “tuberías” que transmiten los datos desde un componente a otro. Este estilo implica una transformación incremental de los datos por sucesivos incrementos.

Figura 3 Filtros y tuberías

#### Llamada y retorno: Patrón N-Capa

El patrón de arquitectura por capas es una de las técnicas que se utilizan para dividir sistemas de softwares complicados. En este esquema la capa más alta utiliza varios servicios definidos por la inferior, pero la última es inconsciente de la superior. Además, normalmente cada capa oculta las capas inferiores de las superiores a esta y se prioriza que se usen solo los servicios de la capa inmediata inferior, sin que se salten capas. A su vez, se puede entender una capa como un todo, sin considerar las otras.

El objetivo primordial es la separación de la lógica de negocios de la lógica de diseño. En este caso, el desarrollo se llevará a cabo en varios niveles y, en caso de que sobrevenga algún cambio, sólo se ataca al nivel requerido sin tener que revisar entre código mezclado; así como que cada grupo de trabajo estará totalmente abstraído del resto de los niveles. En la siguiente figura se puede observar el uso de este patrón arquitectónico en el diseño de la aplicación mencionada.

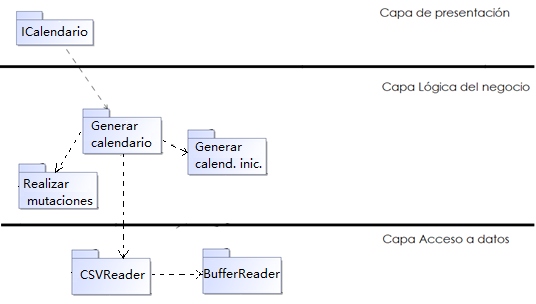


Figure 4 Patrón N-capas: enfoque basado en responsabilidades

# Casos de prueba

Tabla 7 Caso de Prueba: Obtener un calendario inicial

|  |  |
| --- | --- |
| Caso de Prueba | Obtener un calendario inicial |
| Caso de Uso | Generar calendario inicial. |
| Desarrollador | Diana Pérez Abreu. |
| Curso a probar | Generar un calendario inicial que sea factible. |
| Probador | Dariel Luis Rubio Oviedo. |
| Fecha | 11/03/2020. |
| Versión | 1.0. |
| Objetivo de la prueba | Comprobar que se obtiene un calendario inicial que sea factible para la Serie Nacional de Béisbol. |
| Descripción de la prueba | Luego de haber cargado los datos de un fichero .csv se mostrará una matriz con los posibles enfrentamientos entre los equipos que constituirá el calendario inicial para Serie Nacional de Béisbol. |
| Condiciones | Para realizar la prueba es necesario haber cargado datos de un fichero con anterioridad. |
| Condiciones de Entrada | **Resultados Esperados** |
| 1. Para generar un calendario inicial que sea factible se debe haber cargado el fichero .csv que contiene las distancias entre las provincias que participan en la Serie Nacional. 2. Generar una matriz cuadrada triangular superior con las posibles combinaciones de los enfrentamientos de los equipos:   [A, B] [C, D]  Nota: [A, B] es un enfrentamiento, donde A es local y B visitante | 1. Se muestra en el marco de trabajo la matriz que representa el calendario inicial para el conjunto de datos de entrada, donde una fecha está dada por un conjunto de enfrentamientos:   Fecha 1: [A, B] [C, D]  Fecha 2: [A, C] [B, D]  Fecha 3: [A, D] [B, C] |
| Resultados reales | |
|  | |
| Observaciones | |
|  | |

Tabla 8 Caso de Prueba: Realizar mutación: cambio de fecha

|  |  |
| --- | --- |
| Caso de Prueba | Realizar mutación: cambio de fecha |
| Caso de Uso | Realizar mutaciones. |
| Desarrollador | Diana Pérez Abreu. |
| Curso a probar | Realizar un cambio de fechas al calendario actual. |
| Probador | Dariel Luis Rubio Oviedo. |
| Fecha | 11/03/2020. |
| Versión | 1.0. |
| Objetivo de la prueba | Comprobar que se obtiene un nuevo calendario a partir de uno actual en el que las fechas estén intercambiadas. |
| Descripción de la prueba | Luego de haber generado un calendario factible para la Serie Nacional de Béisbol, se cambia la fecha de un enfrentamiento por otra y los demás enfrentamientos quedan desplazados. |
| Condiciones | Se deben haber cargado los datos desde el fichero .csv y haber generado un calendario inicial factible para la Serie Nacional de Béisbol. |
| Condiciones de Entrada | **Resultados Esperados** |
| 1. Se debe haber cargado el fichero .csv que contiene las distancias entre las provincias que participan en la Serie Nacional. 2. Se debe haber generado un calendario inicial factible para el conjunto de enfrentamientos:   [A, B] [C, D]; de la siguiente manera:  Fecha 1: [A, B] [C, D]  Fecha 2: [A, C] [B, D]  Fecha 3: [A, D] [B, C];  A: Local B: Visitante   1. Cambiar la fecha 2 por la fecha 1 | 1. Se obtiene un nuevo calendario factible para la Serie Nacional de Béisbol, en el que la fecha 2 del calendario anterior ha sido intercambiada por la fecha 1:   Fecha 1: [A, C] [B, D]  Fecha 2: [A, B] [C, D]  Fecha 3: [A, D] [B, C] |
| Resultados Reales | |
|  | |
| Observaciones | |
|  | |

Tabla 9 Caso de Prueba: Realizar mutación: cambio de equipo

|  |  |
| --- | --- |
| Caso de Prueba | Realizar mutación: cambio de equipo |
| Caso de Uso | Realizar mutaciones. |
| Desarrollador | Diana Pérez Abreu. |
| Curso a probar | Realizar un cambio de equipo por otro en el calendario actual. |
| Probador | Dariel Luis Rubio Oviedo. |
| Fecha | 11/03/2020. |
| Versión | 1.0. |
| Objetivo de la prueba | Comprobar que se obtiene un nuevo calendario a partir de uno actual en el que un equipo esté intercambiado por otro. |
| Descripción de la prueba | Luego de haber generado un calendario factible para la Serie Nacional de Béisbol, se cambia un equipo dado por otro. |
| Condiciones | Se deben haber cargado los datos desde el fichero .csv y haber generado un calendario inicial factible para la Serie Nacional de Béisbol. |
| Condiciones de Entrada | **Resultados Esperados** |
| 1. Se debe haber cargado el fichero .csv que contiene las distancias entre las provincias que participan en la Serie Nacional. 2. Se debe haber generado un calendario inicial factible para el conjunto de enfrentamientos:   [A, B] [C, D]; de la siguiente manera:  Fecha 1: [A, B] [C, D]  Fecha 2: [A, C] [B, D]  Fecha 3: [A, D] [B, C];  A: Local B: Visitante   1. Intercambiar el equipo A por el equipo B. | 1. Se obtiene un nuevo calendario factible para la Serie Nacional de Béisbol, en el que el equipo A ha sido intercambiado por el equipo B   Fecha 1: [B, A] [C, D]  Fecha 2: [B, C] [A, D]  Fecha 3: [B, D] [A, C] |
| Resultados Reales | |
|  | |
| Observaciones | |
|  | |

Tabla 10 Caso de Prueba: Realizar mutación: cambio de visitadores por sedes

|  |  |
| --- | --- |
| Caso de Prueba | Realizar mutación: cambio de visitadores por sedes |
| Caso de Uso | Realizar mutaciones. |
| Desarrollador | Diana Pérez Abreu. |
| Curso a probar | Realizar un cambio de equipo por otro en el calendario actual. |
| Probador | Dariel Luis Rubio Oviedo. |
| Fecha | 11/03/2020. |
| Versión | 1.0. |
| Objetivo de la prueba | Comprobar que se obtiene un nuevo calendario a partir de uno actual en el que los visitadores por sedes han sido intercambiados. |
| Descripción de la prueba | Luego de haber generado un calendario factible para la Serie Nacional de Béisbol, se cambian los equipos visitantes por sedes. |
| Condiciones | Se deben haber cargado los datos desde el fichero .csv y haber generado un calendario inicial factible para la Serie Nacional de Béisbol. |
| Condiciones de Entrada | **Resultados Esperados** |
| 1. Se debe haber cargado el fichero .csv que contiene las distancias entre las provincias que participan en la Serie Nacional. 2. Se debe haber generado un calendario inicial factible para el conjunto de enfrentamientos:   [A, B] [C, D]; de la siguiente manera:  Fecha 1: [A, B] [C, D]  Fecha 2: [A, C] [B, D]  Fecha 3: [A, D] [B, C];  A: Local B: Visitante   1. Cambiar en la fecha 3 los visitadores por sedes. | 1. Se obtiene un nuevo calendario factible para la Serie Nacional de Béisbol, en el que el equipo A ha sido intercambiado por el equipo B   Fecha 1: [A, B] [C, D]  Fecha 2: [A, C] [B, D]  Fecha 3: [D, A] [C, B] |
| Resultados Reales | |
|  | |
| Observaciones | |
|  | |

Tabla 11 Caso de Prueba: Realizar mutación: cambio de visitador por sede

|  |  |
| --- | --- |
| Caso de Prueba | Realizar mutación: cambio de visitador y sede |
| Caso de Uso | Realizar mutaciones. |
| Desarrollador | Diana Pérez Abreu. |
| Curso a probar | Realizar un cambio de visitador y sede en un enfrentamiento de una fecha dada en el calendario actual. |
| Probador | Dariel Luis Rubio Oviedo. |
| Fecha | 11/03/2020. |
| Versión | 1.0. |
| Objetivo de la prueba | Comprobar que se obtiene un nuevo calendario a partir de uno actual en el que el visitador y la sede de un enfrenamiento de una fecha dada han sido modificados. |
| Descripción de la prueba | Luego de haber generado un calendario factible para la Serie Nacional de Béisbol, se cambia de una fecha seleccionada, el visitador y la sede de un enfrentamiento. |
| Condiciones | Se deben haber cargado los datos desde el fichero .csv y haber generado un calendario inicial factible para la Serie Nacional de Béisbol. |
| Condiciones de Entrada | **Resultados Esperados** |
| 1. Se debe haber cargado el fichero .csv que contiene las distancias entre las provincias que participan en la Serie Nacional. 2. Se debe haber generado un calendario inicial factible para el conjunto de enfrentamientos:   [A, B] [C, D]; de la siguiente manera:  Fecha 1: [A, B] [C, D]  Fecha 2: [A, C] [B, D]  Fecha 3: [A, D] [B, C];  A: Local B: Visitante   1. En una fecha, intercambiar en un enfrentamiento el visitador y la sede (fecha 1, 2do enfrentamiento). | 1. Se obtiene un nuevo calendario factible para la Serie Nacional de Béisbol, en el que para el segundo enfrentamiento de la Fecha 1 el visitador y la sede han sido modificados   Fecha 1: [A, B] [D, C]  Fecha 2: [A, C] [B, D]  Fecha 3: [A, D] [B, C] |
| Resultados Reales | |
|  | |
| Observaciones | |
|  | |

Tabla 12 Caso de Prueba: Realizar mutación: invertir visitador por sede de un equipo

|  |  |
| --- | --- |
| Caso de Prueba | Realizar mutación: invertir visitador por sede de un equipo |
| Caso de Uso | Realizar mutaciones. |
| Desarrollador | Diana Pérez Abreu. |
| Curso a probar | Realizar un cambio de visitador por sede de un equipo dado en el calendario actual. |
| Probador | Dariel Luis Rubio Oviedo. |
| Fecha | 11/03/2020. |
| Versión | 1.0. |
| Objetivo de la prueba | Comprobar que se obtiene un nuevo calendario a partir de uno actual en el que el visitador de un equipo dado ha sido intercambiado por sede y el de sede por visitador. |
| Descripción de la prueba | Luego de haber generado un calendario factible para la Serie Nacional de Béisbol, se invertirá el visitador por sede de un equipo dado. |
| Condiciones | Se deben haber cargado los datos desde el fichero .csv y haber generado un calendario inicial factible para la Serie Nacional de Béisbol. |
| Condiciones de Entrada | **Resultados Esperados** |
| 1. Se debe haber cargado el fichero .csv que contiene las distancias entre las provincias que participan en la Serie Nacional. 2. Se debe haber generado un calendario inicial factible para el conjunto de enfrentamientos:   [A, B] [C, D]; de la siguiente manera:  Fecha 1: [A, B] [C, D]  Fecha 2: [A, C] [B, D]  Fecha 3: [A, D] [B, C];  A: Local B: Visitante   1. Invertir visitador por sede de un equipo (equipo B), a partir del calendario actual. | 1. Se obtiene un nuevo calendario factible para la Serie Nacional de Béisbol, luego de invertir visitador por sede del equipo B y viceversa.   Fecha 1: [B, A] [D, C]  Fecha 2: [A, C] [D, B]  Fecha 3: [A, D] [C, B] |
| Resultados Reales | |
|  | |
| Observaciones | |
|  | |

Tabla 13 Caso de Prueba: Realizar mutación: invertir orden de fechas

|  |  |
| --- | --- |
| Caso de Prueba | Realizar mutación: invertir orden de fechas |
| Caso de Uso | Realizar mutaciones. |
| Desarrollador | Diana Pérez Abreu. |
| Curso a probar | Realizar un cambio en el orden de las fechas en el calendario actual. |
| Probador | Dariel Luis Rubio Oviedo. |
| Fecha | 11/03/2020. |
| Versión | 1.0. |
| Objetivo de la prueba | Comprobar que se obtiene un nuevo calendario a partir de la modificación del orden de las fechas de los enfrentamientos de un calendario ya existente. |
| Descripción de la prueba | Luego de haber generado un calendario factible para la Serie Nacional de Béisbol, se modificará el orden de una sección de fechas consecutivas. |
| Condiciones | Se deben haber cargado los datos desde el fichero .csv y haber generado un calendario inicial factible para la Serie Nacional de Béisbol. |
| Condiciones de Entrada | **Resultados Esperados** |
| 1. Se debe haber cargado el fichero .csv que contiene las distancias entre las provincias que participan en la Serie Nacional. 2. Se debe haber generado un calendario inicial factible para el conjunto de enfrentamientos:   [A, B] [C, D]; de la siguiente manera:  Fecha 1: [A, B] [C, D]  Fecha 2: [A, C] [B, D]  Fecha 3: [A, D] [B, C];  A: Local B: Visitante   1. Invertir el orden de una sección de fechas consecutivas (F1, F2, F3 pasa a ser F3, F2, F1) | 1. Se obtiene un nuevo calendario factible para la Serie Nacional de Béisbol, con un nuevo orden de fechas:   Fecha 3: [A, D] [C, B]  Fecha 2: [A, C] [D, B]  Fecha 1: [B, A] [D, C] |
| Resultados Reales | |
|  | |
| Observaciones | |
|  | |

# Estándar de codificación

Los estándares de código, son parte de las llamadas buenas prácticas o mejores prácticas, estas son un conjunto no formal de reglas, que han ido surgiendo en las distintas comunidades de desarrolladores con el paso del tiempo y las cuales, bien aplicadas pueden incrementar la calidad del código notablemente.

Se entiende como estándar de código a un conjunto de convenciones establecidas de ante mano (denominaciones, formatos, etc.) para la escritura de código. Estos estándares varían dependiendo del lenguaje de programación seleccionado, en este caso Java.

1. **Indentación**

* Se deben emplear cuatro espacios como unidad de indentación según las convenciones de código de Java.
* Evitar líneas de más de 80 caracteres debido a que según la documentación de Java estas no son bien manejadas por muchas herramientas y terminales.
* Cuando una expresión no entre en una línea, debe romperse teniendo en cuenta los siguientes aspectos:
* Romper luego de una coma.
* Romper antes de un operador.
* Alinear la nueva línea con el comienzo de la expresión al mismo nivel que la línea anterior.

1. **Convenciones de nombres**

Tabla 14 Convención de nombres

|  |  |  |
| --- | --- | --- |
| **Tipos de identificadores** | **Reglas** | **Ejemplos** |
| Clases | Los nombres de las clases implementadas deben ser sustantivos y cuando sean nombres compuestos tendrán la primera letra de cada palabra que lo conforma en mayúsculas. El nombre asignado debe ser simple y descriptivo. Usar palabras completas, evitando las abreviaturas. | class SerieNacional; |
| Interfaces | Los nombres de las interfaces deben seguir las mismas reglas de las clases. | interface Deporte; |
| Métodos | Los nombres de los métodos o funciones deben ser verbos que describan de forma breve su objetivo. Cuando el nombre sea compuesto, debe tener la primera letra en minúscula y la primera letra de cada una de las siguientes palabras que lo conforma en mayúscula. | RealizarCalendarioInicial(); |
| Variables | Los nombres de los métodos o funciones deben ser verbos que describan de forma breve su objetivo. Cuando el nombre sea compuesto, debe tener la primera letra en minúscula y la primera letra de cada una de las siguientes palabras que lo conforma en mayúscula. Los nombres de las variables no deben empezar con los caracteres “\_” o signo “$”. Deben ser nombres cortos pero que describan su objetivo.  Los nombres de las variables de un solo caracter se deben evitar, excepto para variables ídices temporales (i,j,k,m y n para enteros; c,d y e para caracteres). | int i;  float maxDistancia; |
| Constantes | Los nombres para las variables que sean declaradas como constantes deben ir en mayúscula separando las palabras con un subguión (“\_”). | float PI=3.14; |

1. **Declaraciones**

* **Cantidad de declaraciones por línea:**
* Se recomienda una declaración por línea, ya que provee la escritura de comentarios, como, por ejemplo:

**int equipos; // cantidad de quipos**

**int tamRuta; //tamaño de una ruta**

y **no: int equipos, tamRuta;**

* No poner diferentes tipos de variables en la misma línea.
* **Inicialización**

Inicializar las variables locales donde se declaran. La única excepción para no inicializar una variable donde se declara es si el valor inicial de esta depende de cálculos que deben realizarse.

* **Colocación**

Las declaraciones sólo deben ubicarse al inicio de cada bloque. No se debe esperar al primer uso de las variables para que sean declaradas ya que limitan la portabilidad del código dentro de su ámbito de visibilidad. Excepción de esto son los índices de bucles for, que en Java se pueden declarar en la sentencia for. De igual manera, evitar realizar declaraciones locales que ocultan declaraciones que se encuentran en un nivel superior.

1. **Sentencias**

* **Sentencias simples:** Cada línea debe contener como mucho una sentencia.
* **Sentencias compuestas:**
* **Sentencias de retorno:** Una sentencia que devuelve un valor no debería usar paréntesis al no ser que hagan el valor de retorno más claro (return (**ruta? ruta: rutaPorDefecto**) ;).
* **Sentencias de if, if-else, if else-if else:** Deben seguir la siguiente estructura:
* if (*condicion*) {

*sentencias;*

}

* if (*condicion*) {

*sentencias;*

} else {

*sentencias;*

}

* if (*condicion*) {

*sentencias;*

} else if (*condicion*) {

*sentencias;*

} else {

*sentencias;*

}

* **Sentencias for:** Debe tener la siguiente estructura:
* for *(inicialización; condición; actualización*) {

*sentencias;*

}

* for *(inicialización; condición; actualización*); //si es una sentencia for vacía.

**Nota:** Al usar el operador “,” en la cláusula de inicialización o actualización de una sentencia de este tipo, evitar la complejidad de usar más de tres variables.

* **Sentencias while:** Debe tener la siguiente estructura:
* while (*condition*) {

*sentencias;*

}

* while (*condicion*); //si es una sentencia while vacía
* **Sentencias do-while:** Debe tener la siguiente estructura:
* do {

*sentencias;*

} while (*condicion*);

* **Sentencias switch:** Debe tener la siguiente estructura:
* switch (*condición*) {

case A:

*sentencias*;

break;

case B:

*sentencias;*

break;

default:

*sentencias;*

break;

}

* **Sentencias try-catch:** Debe tener la siguiente estructura:
* try {

*sentencias;*

} catch (ExceptionClass e) {

*sentencias;*

}

**Nota:** Puede ir seguida de un finally, cuya ejecución se efectuará independientemente de que el bloque try se haya realizado con éxito o no.

1. **Espacios en blanco**

* **Líneas en blanco:**
* Se deben usar siempre dos líneas en blanco entre las secciones de un fichero fuente y entre las definiciones de clases e interfaces.
* Se debe usar siempre una línea en blanco ente métodos, entre las variables locales de un método y su primera sentencia, entre las distintas secciones lógicas de un método para facilitar la lectura y antes de un comentario de bloque o de un comentario de línea.
* **Espacios en blanco:**
* Una palabra clave del lenguaje seguida por un paréntesis debe separarse por un espacio.
* Debe aparecer un espacio en blanco después de cada coma en la lista de argumentos.
* Los operadores binarios se deben separar de sus operandos con un espacio en blanco.
* Las expresiones en una sentencia for se deben separar con espacios en blanco.

1. **Comentarios**

Se deben usar los comentarios para dar descripciones de código y facilitar información adicional que no es legible en el código mismo. Los comentarios deben contener sólo información que es relevante para la lectura y entendimiento del programa. En Java existen dos tipos de comentarios: de implementación y de documentación.

* **Comentarios de implementación:** Se utilizan como bien dice su nombre para comentar el código o una implementación particular. Se utilizará el delimitador de comentario “//” sólo para convertir en comentarios una línea completa o una parte de una línea, por lo que no se usará para hacer comentarios de varias líneas consecutivas; en este caso, se utilizará el delimitador de comentario “/\* \*/”.
* **Comentarios de documentación:** Describen clases, interfaces, constructores, métodos y atributos. Este tipo de comentario debe aparecer justo antes de una declaración:

/\*\*

\* La clase Ruta es la encargada de …

\*/

public class Ruta {

…

}

# 8. Arquitectura candidata.

Para dar solución a la problemática ya antes mencionada se seleccionó el lenguaje de programación Java por ser un lenguaje simple, cuya curva de aprendizaje es realmente corta. Es un lenguaje orientado a objetos, los cuales encapsulan información, clases y funciones que pueden ser manipuladas o utilizadas más adelante. Además, con este lenguaje, se tiene la posibilidad de desarrollar aplicaciones distribuidas. Una de las principales ventajas de este lenguaje es su compilación, que se llega a asimilar al lenguaje ensamblador, es decir, desde la base puede ser interpretado. Esto ayuda muchísimo a la ejecución de aplicaciones compiladas en Java, pues se puede ejecutar básicamente en cualquier lugar sin mayor problema.

Para realizar y facilitar las tareas de editar, compilar, ejecutar y depurar el código implementado se necesita un IDE (*Integrated Development Environment*) específico para Java. Existen diferentes entornos de desarrollo que nos posibilitan realizar estas tareas. Los mejores entornos de desarrollo para Java son: Eclipse, NetBeans, IntelliJ IDEA. Existen, además de los tres anteriores, muchos otros productos en esta categoría, los cuales no llegan a alcanzar la madurez y funcionalidad de los ya mencionados.

## 8.1 NetBeans

Inicialmente desarrollado por Sun y ahora en manos de Oracle, NetBeans es uno de los IDE para desarrollo Java más completo. NetBeans tiene una estructura modular fácilmente ampliable mediante complementos, existiendo configuraciones predefinidas para desarrollo java SE, java EE y también dirigidas a otros lenguajes de programación como PHP o C++. NetBeans es un proyecto de código abierto, está desarrollado en Java y, en consecuencia, puede instalarse en cualquier sistema para el que exista un JRE de Java SE, incluidos Windows, GNU/Linux y OS X/MacOS.

## 8.2 IntelliJ IDEA

Intellij IDEA es un IDE para Java desarrollado por la empresa Jet Brains, estando disponible para Windows, OS X/MacOS y GNU/Linux. Este IDE puede incorporar soporte para otros lenguajes de programación, al igual que NetBeans y Eclipse. A diferencia de ellos, sin embargo, no se trata de un proyecto totalmente basado en software libre, sino de un producto comercial. Existe, no obstante, una edición reducida denominada Community que puede obtenerse gratuitamente.

## 8.3 Eclipse

El entorno de desarrollo integrado Eclipse emplea módulos para proporcionar toda su funcionalidad al frente de la Plataforma de Cliente, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. Adicionalmente, permite a Eclipse extenderse usando otros lenguajes de programación como son C/C++ y Python, permite a Eclipse trabajar con lenguajes para procesado de texto como LaTeX, aplicaciones de red como Telnet y Sistemas de Gestión de bases de datos, y, escribir cualquier extensión deseada en el ambiente, como sería Gestión de la configuración. Se provee soporte para Java y CVS en el SDK de Eclipse. Y no tiene por qué ser usado únicamente para soportar otros lenguajes de programación. Por todo lo mencionado, es considerado por muchos el IDE por excelencia, al incorporar un gran abanico de complementos que facilitan prácticamente todas las tareas relativas al desarrollo del software.

Por todo lo antes mencionado, y analizando cada una de las características que brindan estos entornos de desarrollo integrado, se decide utilizar como herramienta para dar solución a la problemática expuesta anteriormente el entorno de desarrollo Eclipse.