Escuela Politécnica Nacional

Facultad de Ingeniería de Sistemas

Programación II

Estándares de Desarrollo de Software para Programación Orientada a objetos con Base de datos (SQLITE)

Integrantes:

- Calles Kevin
- Cobos Antony
- Pagual Jonathan
- Pisco Christian

Fecha de Entrega: 29/02/2024

Docente: Ing. Patricio Paccha

2023-B

Índice

1. Propósito

En este documento tiene como objetivo proporcionar una explicación para las reglas y normas básicas que se deben tener en cuenta al momento de llevar a cabo una codificación para un proyecto de Java con implementación de base de datos de modo que sea comprensible, facilite la legibilidad del código para un mantenimiento adecuado, y favorecer futuras mejoras.

2. Destinatario

El principal destinatario de este documento son los integrantes universitarios y Docente que forman parte del desarrollo del proyecto de clase, al igual que las personas que lleguen a tener interés en como emplear una buena práctica al momento de llevar a cabo una programación en Java.

3. Plataforma Java

3.1 Buenas Practicas

Convención de Nomenclatura

• Se debe adoptar la convención de nomenclatura "UpperCase" para el lenguaje de programación empleado en este proyecto. Según esta convención, los nombres de las clases deben comenzar con la primera letra en mayúscula, mientras que las letras iniciales de las palabras intermedias deben estar en minúsculas. Además, no se deben utilizar espacios ni símbolos de puntuación entre las palabras. En el caso de que un nombre conste de más de una palabra, se empleará la notación UpperCase, donde la primera letra de cada palabra adicional será mayúscula. A modo de ejemplo, "MiClase" cumple con esta convención.

Integración de bibliotecas

• Se fomenta la utilización de bibliotecas existentes con el propósito de agilizar el desarrollo del proyecto. Se dará preferencia a bibliotecas reconocidas por su calidad, seleccionando únicamente aquellas funciones e implementaciones necesarias para evitar una carga innecesaria en el programa.

Al incorporar bibliotecas, se exigirá la adopción de la palabra clave "import" seguida del nombre del paquete que alberga la biblioteca correspondiente. Este enfoque asegurará una integración ordenada y eficiente de las funcionalidades proporcionadas por las bibliotecas externas.

Nombres de variables, métodos y constantes (CamelCase)

• En los nombres de variables y métodos se deben utilizar nombre que describan el método o el valor de la variable, deben comenzar con minúsculas, evitar utilizar abreviaturas o acrónimos y no deben ser demasiados largos o difíciles de recordar. Por ejemplo, “leerQr” o “miVariable”. Las constantes deben ir en mayúsculas y en caso de dos o más palabras se debe usar como separador un carácter guion bajo (_). Por ejemplo, “TARIFA_DIA”.

Encapsulamiento y métodos de acceso

• Se requiere el empleo de las palabras clave "private", "public" o "protected" para la declaración de campos, es decir, variables de instancia dentro de una clase.

• Adicionalmente, se establece la obligatoriedad de implementar métodos de acceso, comúnmente conocidos como "getters" y "setters", para interactuar con los campos o atributos de la clase. Estos métodos garantizan un acceso controlado y seguro a las variables internas, fortaleciendo el principio de encapsulación en el diseño del código.

Principios de diseño: Herencia, interface y clase abstracta.

• **Preferencia por la Composición sobre la Herencia:** Se fomenta la adopción del principio de composición sobre la herencia siempre que sea factible. Esta práctica contribuye a un diseño más flexible y menos propenso a complejidades inherentes a la herencia, promoviendo la reutilización de componentes a través de la composición de objetos.

• **Uso de Interfaces para Definir Contratos:** Se insta al empleo de interfaces para establecer contratos entre componentes del sistema. La utilización de interfaces facilita la implementación de múltiples interfaces en una clase, proporcionando un mecanismo efectivo para la gestión de contratos y promoviendo la coherencia en el diseño del software.

• **Nombrado Significativo para Clases Abstractas:** En el caso de clases abstractas, se requiere la asignación de un nombre significativo que refleje claramente el propósito y la funcionalidad de la clase abstracta. A modo de ejemplo, se sugiere utilizar nombres como "Abstract" seguido de una descripción que exprese de manera precisa la naturaleza de la clase abstracta. Este enfoque contribuye a una comprensión más clara y mantenible del código.

Sobreescritura de métodos y evasión de sobrecarga

• **Uso de la Notación "@Override" para la Sobreescritura:** Al llevar a cabo la sobreescritura de métodos, se requiere la incorporación de la notación "@Override". Esta práctica es esencial para indicar explícitamente que se está reemplazando un método de la clase principal. Facilita la identificación y el mantenimiento del código, fortaleciendo la claridad y la correcta implementación de la lógica de la aplicación.

• **Evitar la Sobrecarga de Métodos con Nombres Similares:** Se aconseja evitar la sobrecarga de métodos que posean nombres demasiado similares, ya que esto podría generar confusiones en el entendimiento y uso del código. La claridad en la nomenclatura de los métodos contribuye a un desarrollo más eficiente y comprensible. En su lugar, se recomienda adoptar nombres descriptivos y distintivos para cada método, destacando su propósito y funcionalidad específicos.

Manejo de excepciones y tiempo de ejecución

• **Utilización de "try-catch" para el Manejo de Excepciones:** Se prescribe la aplicación de bloques "try-catch" para gestionar excepciones y prevenir posibles errores que puedan afectar el funcionamiento del programa. Este enfoque permite un control más efectivo de situaciones excepcionales, mejorando la estabilidad y robustez del software.

• **Evitar el Uso Excesivo de Tiempo de Ejecución:** Se insta a evitar el uso excesivo de excepciones durante el tiempo de ejecución del programa. Es fundamental emplear excepciones específicas únicamente cuando sea estrictamente necesario, evitando la generación de excepciones genéricas que puedan dificultar el diagnóstico y la resolución de problemas. La selección cuidadosa de excepciones contribuye a un manejo más eficiente y comprensible de las situaciones excepcionales.

Comentarios y documentación

1. **• Comentarios Significativos para la Implementación:**

    - Es imperativo suministrar comentarios que detallen la función de las variables locales, las fases lógicas de ejecución y la captura de excepciones en la implementación del código. Estos comentarios proporcionan una guía esencial para comprender el propósito y el funcionamiento interno de las porciones críticas del código.

2. **• Uso de Documentación para Especificaciones:**

    - La documentación se emplea para describir la especificación del código de manera independiente a la implementación. Está diseñada para ser consultada por desarrolladores que no tengan acceso al código fuente. La documentación detallada de la funcionalidad, interfaces y comportamientos esperados permite una comprensión integral del software, facilitando el mantenimiento y la colaboración en el desarrollo del proyecto.

Uso de pausas

• Se utilizan las funciones thread.sleep() para introducir pausas en la ejecución del programa, tanto para validar información o como un contador de tiempo.

4. Plataforma SQLITE

4.1 Convención JDBC

La convención JDBC (Java Database Connectivity) es una API estándar de Java para interactuar con bases de datos relacionales. Cuando se utiliza JDBC para trabajar con bases de datos, se deben seguir prácticas específicas relacionadas con la conexión, la ejecución de consultas y la gestión de resultados.

4.2 Establecimiento de conexiones

- Utiliza la interfaz java.sql.Connection para establecer y gestionar conexiones con la base de datos.
- Emplea la palabra clave try-with-resources al crear objetos Connection para garantizar su cierre adecuado.

4.3 Creación y ejecución de consultas

- Se utiliza objetos de tipo PreparedStatement para ejecutar consultas parametrizadas y prevenir la inyección SQL.
- Se usa Statement para consultas sin parámetros.

4.4 Comentarios y Documentación:

Se deberá proporcionar comentarios significativos en el código para explicar la lógica detrás de las operaciones con la base de datos. También, documenta la estructura y el propósito de la base de datos en un nivel más alto.

4.5 Gestión de Excepciones:

Implementa manejo de excepciones específico para las operaciones JDBC. Captura y gestiona las excepciones relacionadas con la base de datos de manera apropiada.

4.6 Usar DataSource (opcional):

- Considera el uso de **javax.sql.DataSource** para obtener conexiones, lo que puede mejorar la administración de conexiones en aplicaciones empresariales.