UNIVERSIDAD DE LAS FUERZAS ARMADAS-ESPE

SEDE SANTO DOMINGO DE LOS TSÁCHILAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN - DCCO-SS

CARRERA DE INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

PERIODO	:	202450 Mayo– Septiembre 2024
ASIGNATURA	:	Programación Orientada a Objetos
TEMA	:	Proyecto Grupal-Gestión de inmuebles
ESTUDIANTE	:	**Grupo I**
NIVEL-PARALELO - NRC:		Segundo A/B - Nrc 15279
DOCENTE	:	Ing. Verónica Martínez C., Mgs.
FECHA DE ENTREGA	:	07 de junio de 2024.

SANTO DOMINGO – ECUADOR

Contenido

1. Introducción

Se ha planteado como razón principal de este proyecto, el poder crear una aplicación o programa en el cual podamos aplicar todos los conocimientos relacionados a la programación orientada a objetos, por lo que en este ejercicio se podrá ver aplicado cada uno de los temas cubiertos en las tareas y puntos principales vistos en el silabo de la materia.

2. Objetivos

2.1 Objetivo General:

Crear un programa en java donde se vean aplicados los conocimientos correspondientes a la programación orientada a objetos

2.2 Objetivos Específicos:

Revisar cada punto del silabo para crear un programa de calidad.
Tener un programa controlado en los errores que este pueda presentar.
Documentar y presentar paso a paso de la creación de este programa.

3. Marco Teórico/ Práctica

3.1 Sistemas de control de Versionamiento.

Un sistema de control de versiones permite rastrear el historial de una colección de archivos y permite la colaboración en esos archivos entre muchas personas. Ayuda mucho al momento en que un programador necesita ayuda con algún proyecto y generar un proyecto de calidad con la posibilidad de regresar si existiera errores Linus Torvalds es el creador de Git el más conocido sistema de control de Versionamiento.

3.1.1 Software VCS: Git, GitHub.

GitHub es una plataforma de desarrollo colaborativo. Puedes alojar y revisar código, administrar proyectos y construir software junto a millones de otros desarrolladores

- **Colaboración:** GitHub proporciona un lugar para que los desarrolladores trabajen juntos en proyectos, permitiendo la colaboración efectiva en el código.

- **Integración con herramientas:** GitHub se integra con muchas herramientas de desarrollo de software y proporciona su propia API para que los desarrolladores puedan crear sus propias herramientas.

- **Seguimiento de problemas:** GitHub proporciona seguimiento de problemas que permite a los desarrolladores mantener un registro de los problemas y las características solicitadas.

3.2 Paradigmas de Programación.

Los paradigmas de programación representan diferentes maneras de pensar sobre cómo se debe estructurar y organizar el código. Algunos de los paradigmas de

programación más comunes incluyen:

Programación imperativa: Este paradigma se centra en describir "cómo" se debe realizar una tarea. Los lenguajes de programación imperativos incluyen C, C++, Java y Python.

Programación declarativa: Este paradigma se centra en describir "qué" resultado se desea, sin especificar necesariamente cómo obtenerlo. Los lenguajes de programación declarativos incluyen SQL y HTML.

Programación orientada a objetos (OOP): Este paradigma se centra en los objetos y sus interacciones. Los lenguajes de programación OOP incluyen Java, C++, Python y JavaScript.

Programación funcional: Este paradigma se centra en las funciones y evita cambiar el estado y los datos mutables. Los lenguajes de programación funcional incluyen Haskell, Erlang y Clojure.

3.2.1 Transición de Paradigma

La transición de un paradigma de programación a otro puede ser un desafío, pero también puede abrir nuevas formas de pensar y resolver problemas. Aquí hay algunos puntos a considerar al hacer la transición:
Entender las diferencias: Es importante entender las diferencias fundamentales entre los paradigmas de programación. Esto puede ayudar a evitar confusiones y facilitar la transición.

Practicar: La práctica es esencial cuando se aprende un nuevo paradigma de programación. Trabajar en proyectos pequeños puede ser una excelente manera de familiarizarse con el nuevo paradigma.

Paciencia: La transición a un nuevo paradigma de programación puede llevar tiempo. Es importante ser paciente y dar un paso atrás si es necesario para asegurarse de que se comprenden completamente los conceptos.

3.3 Entorno de Desarrollo

3.3.1 Características e Instalación

Instalación de programas incluida dentro de anteriores trabajos

3.3.2 Administración y Configuración del Área de Trabajo

Instalación y configuración incluida dentro de anteriores trabajos

3.3.3 Líneas de Comando

Incluidas dentro de anteriores trabajos

3.4 Revisión de conceptos generales de la POO

3.4.1 Principios Generales de la Programación Orientada a Objetos

La programación orientada a objetos (POO) es un paradigma de programación que se basa en la idea de que todo en un programa es un objeto que interactúa con otros objetos. Los principios generales de la POO son los siguientes:

Abstracción: La abstracción es el proceso de identificar las características esenciales de un objeto y eliminar los detalles irrelevantes. En POO, la abstracción se logra mediante la creación de clases que representan objetos del mundo real y la definición de sus propiedades y métodos.

Encapsulación: La encapsulación es el proceso de ocultar los detalles de implementación de un objeto y exponer solo una interfaz pública para interactuar con

él. En POO, la encapsulación se logra mediante la definición de atributos y métodos privados y públicos en una clase.

Herencia: La herencia es el proceso de crear una nueva clase a partir de una clase existente. La nueva clase hereda todas las propiedades y métodos de la clase existente y puede agregar nuevos atributos y métodos. La herencia se utiliza para crear jerarquías de clases y para reutilizar el código existente.

Polimorfismo: El polimorfismo es la capacidad de un objeto para tomar muchas formas diferentes. En POO, el polimorfismo se logra mediante la definición de métodos con el mismo nombre en diferentes clases. Cada clase puede implementar el método de manera diferente, lo que permite que los objetos de diferentes clases se comporten de manera diferente cuando se llama al mismo método.

3.4.2 Definición de Clases, Objetos, Atributos y Métodos.

Clase: En la programación orientada a objetos (POO), una clase es un diseño que se puede utilizar para crear varios objetos individuales. Las clases son modelos o abstracciones de la realidad que representan un elemento de un conjunto de objetos. Una clase define un grupo o conjunto de datos, en objetos se llaman atributos, que definen los objetos, así como un conjunto de comportamientos, las funciones o métodos del objeto, que lo manipulan y relacionan los objetos unos de otros.

Objetos: Los objetos en programación representan cosas del mundo real, así como conceptos abstractos con sus características y comportamientos específicos. Un objeto es una copia creada a partir de una clase. Cada objeto creado a partir de la clase se denomina instancia de la clase.

Atributos: En la programación orientada a objetos, los atributos son las propiedades que pueden asumir los objetos dentro de una clase. Estas son descripciones de los datos. Por ejemplo, el atributo «color» puede tener el valor «azul» o «magenta». Todos los objetos de una clase tienen los mismos atributos. Los atributos son la información que guarda en un momento determinado cada objeto que se instancia de una clase.

Métodos: En programación orientada a objetos, un método es una función o un procedimiento asociado a un objeto. Los métodos describen el comportamiento de un objeto y proporcionan una forma de manipular el estado de un objeto o de interactuar con otros objetos. Un método es un bloque de código que tiene un nombre y puede recibir uno o varios parámetros.

```java
package GrupoI_GestionInmuebles;

import java.util.*;

public class GrupoI_Inmueble {
    Scanner inputInm = new Scanner(System.in);
    private String address; //Ubicacion del inmueble
    private String locationArea;  //Variable del tipo de zona (rural/urbano)
    private double price; //Precio del inmuelbe
    private int squareMeter; //Metros cuadrados
    private String tipePay; //Tipo de pago credito o al contado
    private int timePayments; //Por meses
    private String propertyLand; //Esta variable referencia el tipo de terreno (Urbano, urbanizable, rústico).
    public GrupoI_Inmueble( String locationArea, String address, double price, int squareMeter, String tipePay, int timePayments,
            String propertyLand) {
        this.address = address;
        this.locationArea = locationArea;
        this.price = price;
        this.squareMeter = squareMeter;
        this.tipePay = tipePay;
        this.timePayments = timePayments;
        this.propertyLand = propertyLand;
    }

    public String toString() { //Retorna todos los valores ingresados o seleccionados.
        return "direccion "+address+"\n"+
                "Zona (Rural/Urbana)"+locationArea+
                "Precio: "+price+"\n"+
                "Metros cuadrados: "+squareMeter+"\n"+
                "Tipo de pago (Contado/Crédito)"+tipePay+"\n"+
                "Tiempo de pago: "+timePayments+
                "Terreno de la propiedad: "+propertyLand+"\n";
    }

    public double CalculatorPriceInm() { //Si el pago corresponde a un tipo de crédito, se añadirá el método.
        double priceFinalMens;
        double impuesto = 0.12;
        priceFinalMens = (price/timePayments)+(price*impuesto);
        return priceFinalMens;
```

Clase Madre o Padre con Atributos, Constructor y sus Métodos

3.5 Modelamiento de Clases y Objetos

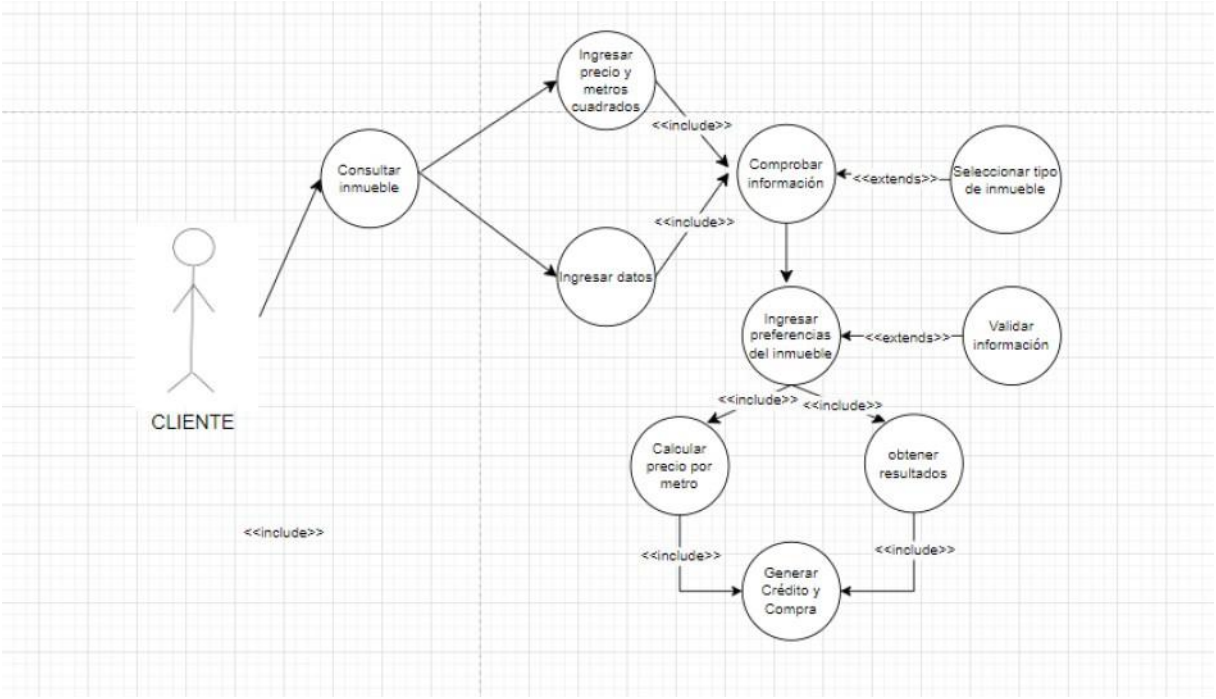
3.5.1 UML: Diagramas de Caso de Uso.

Los Diagramas de Casos de Uso son una parte integral del Lenguaje Unificado de Modelado (UML). Estos diagramas se utilizan para representar las interacciones entre los actores (usuarios o sistemas externos) y el sistema de información[12].
Los elementos principales de un Diagrama de Casos de Uso son:

Actores: Son los usuarios o sistemas externos que interactúan con el sistema. Se representan con una imagen de un “muñeco de palo” con el nombre del actor debajo.

Casos de Uso: Son las funcionalidades o requisitos funcionales del sistema con los que interactúan los actores.

Relaciones: Representan las interacciones entre los actores y los casos de uso.
Estos diagramas son útiles para representar los requisitos funcionales, guiar el desarrollo del sistema, y facilitar la comunicación precisa entre el cliente y el desarrollador. Aunque son bastante estáticos en comparación con otros diagramas de comportamiento en UML, los Diagramas de Casos de Uso proporcionan una visión valiosa del comportamiento esperado de un sistema o software en un caso de uso concreto.
Para el presente proyecto se ha desarrollado el siguiente diagrama de casos de Uso:

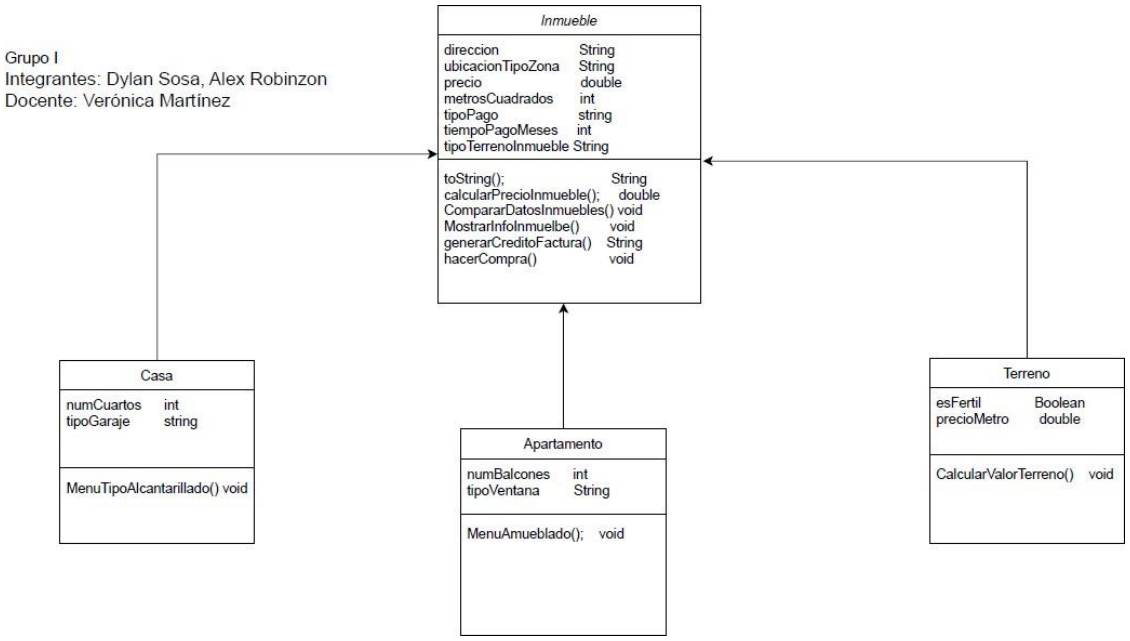


3.5.2 UML: Diagrama de Clases

Los Diagramas de Clases son una parte esencial del Lenguaje Unificado de Modelado (UML). Estos diagramas se utilizan para visualizar la estructura de un sistema específico al modelar sus clases, atributos, operaciones y las relaciones entre los objetos.

Los elementos principales de un Diagrama de Clases son:
Clases: Son los componentes básicos de los objetos y los diagramas de clases. Pueden representar las clases que se programarán en realidad, los objetos principales o la interacción entre clases y objetos.

Atributos: Son las características o propiedades de una clase.
Operaciones: Son las funciones o métodos que una clase puede realizar.
Relaciones: Representan las conexiones entre las clases.
Estos diagramas son útiles para representar la estructura de un sistema, guiar el desarrollo del sistema, y facilitar la comunicación precisa entre el cliente y el desarrollador. Aunque son bastante estáticos en comparación con otros diagramas de comportamiento en UML, los Diagramas de Clases proporcionan una visión valiosa de la estructura esperada de un sistema o software. En el proyecto se realizó el siguiente diagrama de clases:



3.5.3 Identificación de Clases de un Sistema, Uso Correcto de Identificadores

La **identificación de clases** en un sistema es un proceso crucial en la programación orientada a objetos. Los objetos se determinan subrayando cada nombre o cláusula nominal e introduciéndola en una tabla simple. Los objetos pueden manifestarse de varias formas, como entidades externas, cosas, ocurrencias o sucesos, papeles o roles, unidades organizacionales, lugares, y estructuras.
Los **identificadores** son fundamentales en la programación, ya que se utilizan para nombrar variables, funciones, clases y otras entidades de programación. Para nombrar los identificadores, se deben seguir ciertas reglas. Los identificadores deben ser descriptivos y reflejar claramente lo que representa la entidad de programación.

3.5.4 Modificadores de Acceso

Los **modificadores de acceso** son palabras clave utilizadas para controlar el acceso a los atributos y métodos de una clase. Los modificadores de acceso más comunes son “public”, “private” y “protected”, pero dependiendo del lenguaje de programación, existen modificadores adicionales como “internal” o “package-private”.

3.5.5 Implementación de Clases

La **implementación de clases** sigue los mismos pasos que la implementación de un módulo de datos, es decir, hay que obtener una representación de la misma a partir de las clases ya existentes y, a continuación, codificar sus operaciones en el lenguaje de programación correspondiente.

3.6 Código Limpio

3.6.1 Estándares de Implementación, Buenas Prácticas de Programación

Las buenas prácticas de programación y los estándares de implementación son esenciales para garantizar la eficiencia, la mantenibilidad y la reducción de errores del software. Aquí hay algunos puntos que puede incluir en su reseña.

Programación Orientada a Objetos (POO): Este paradigma de programación se basa en el concepto de "objetos" que pueden contener datos y código. Los datos son propiedades y el código son métodos.
Principio DRY (No te repitas): este principio se basa en reducir la duplicación de código. Se debe intentar la reutilización del código siempre que sea posible.
Pruebas unitarias y de integración: las pruebas son esenciales para garantizar que el código funcione según lo previsto. Las pruebas unitarias se utilizan para probar la funcionalidad de una sección específica de código, mientras que las pruebas de integración se utilizan para probar si los diferentes componentes de un sistema funcionan bien juntos.
Refactorización: La refactorización es el proceso de cambiar la estructura del código de un programa sin cambiar su comportamiento. Es una tecnología utilizada para mejorar la legibilidad del código, reducir la complejidad, eliminar el código duplicado y mejorar el rendimiento del código.
Documentación: La documentación es una parte importante de cualquier proyecto de software.
Debe ser claro, conciso y estar sincronizado con los cambios de código.
Control de versiones: el control de versiones es una práctica importante de desarrollo de software. Esto permite a los desarrolladores desarrollar diferentes funciones en paralelo y administrar versiones de software más fácilmente.
Integración continua/Implementación continua (CI/CD): CI/CD es una práctica de DevOps que permite a los desarrolladores integrar su trabajo con frecuencia e implementar código en producción de forma rápida y segura. Seguridad: La seguridad debe considerarse en todas las etapas del desarrollo de software. Esto incluye administrar dependencias, administrar secretos, proteger contra ataques comunes como inyección SQL y XSS, y realizar auditorías de seguridad.

3.6.2 Atributos de Calidad de Código

Los atributos de calidad del software son características o patrones consensuados que se utilizan para realizar una medición de los sistemas. Aquí te dejo algunos atributos de calidad que podrías incluir en tu informe:
1. **Funcionalidad**: El software debe cumplir con las funciones implícitas y explícitas para las cuales fue diseñado.
2. **Confiabilidad**: Un sistema de información debe ser confiable y sobre todo responder en momentos críticos.
3. **Usabilidad**: Este atributo se refiere a la facilidad de uso del software. Un software con buena usabilidad es intuitivo y fácil de aprender.
4. **Rendimiento**: Se refiere a la eficiencia del software. Un software con buen rendimiento es rápido y eficiente en el uso de recursos.
5. **Mantenibilidad**: Este atributo se refiere a la facilidad con la que se pueden hacer

cambios en el software. Un software mantenible es fácil de modificar y extender.
6. **Correctitud**: Ausencia de errores.
7. **Consistencia**: Coherencia entre las operaciones que realiza el usuario.
8. **Completitud**: Capacidad del sistema para realizar todas las operaciones que el usuario podría requerir

1. Estructura general de un programa

3.7.1 Tipos de datos, Primitivos y Referenciados

Los datos primitivos son los aquello que ya tenemos conocidos desde la anterior programación, estos serán:

- **Int**: para números enteros
- **Float**: para números decimales y enteros o flotantes
- **Double**: para números reales como float pero con doble precisión
- **Char**: para caracteres
- **Boolean**: para valores lógicos como Verdadero o Falso Los datos Referenciados son aquellos que necesitan de una estructura de referencia a un objeto, esta tendrá la dirección de la memorita del objeto, entre los datos referenciados están los siguientes:
- **String**: la típica cadena de caracteres el cual usamos para el presente proyecto.
- **Array**: Arreglo de datos
- **Object**: Objetos o instancias de clases.

Lectura y escritura de datos por consola

Entrada de Datos

Para la entrada de datos en consola o todo lo que se aparecerá al momento de la compilación es necesario el uso del Scanner () el cual creara un objeto de esta clase, este además de un next”Tipo de dato”() será el que permita que un usuario o una persona pueda ingresar, a continuación un ejemplo de ingreso de datos por consola en el proyecto realizado.

```
Scanner sc = new Scanner(System.in);
int opc =0;
```

Intanciación del objeto scanner de la clase Scannerpara poder ingresar datos

Iniciación de la variable opc por medio de la funcipon Scanner

Salida de Datos

Para la salida de datos por consola es mucho más fácil que el ingreso, ya que ahora no será necesario el crear un objeto scanner y solo necesitaremos utilizar una de las siguientes funciones:
System.ou.println(): Esta función mostrara la información a imprimir la cual estará dentro del paréntesis, pero imprimirá todo de manera seguida.

System.out.println(): Esta función también permitirá la salida de información, pero ahora ya usará un salto de línea al imprimir. A continuación, mostramos ejemplos de la aplicación dentro del proyecto.

```
System.out.println("BIENVENID@ A LA GESTION DE COMPRAS DE INMUELBES");
System.out.println("Ingrese las caracteristicas de su inmueble deseado: ");
System.out.print("Ubicacion del inmuelbe: ");
```

Usamos la función println para poder tener un salto de linea dentro de la consola, ademas dentro del parentesis tenemos texto pero tambien puede tener una variable que contenga algun valor

Excepciones

Definición

Las excepciones son controles aplicados en el código para que cuando algo llegue a fallar este le encuentre una solución y de no ser así, se lanza un mensaje al programador para que este pueda arreglar el código o el problema en donde se haya lanzado una alerta.

Excepciones y Errores

Los errores son problemas graves que le suceden al programa y son imprevistos mientras que las excepciones pueden ser manejadas y controladas por código dentro de un programa

Clases de Excepciones

Existen dos tipos de Excepciones, estas serán, las excepciones verificadas, que pueden darse no necesariamente a nuestro programa sino también a errores al leer archivos o al querer conectarse a algún tipo de red y las excepciones no verificadas las cuales pueden ser vistas o manejadas a elección del programador en el código

Tipos de Excepciones

Existen varios tipos de excepciones como las de tiempo de ejecución, excepciones de red y excepciones personalizadas, entre otras

Excepciones Personalizadas

Las excepciones personalizadas serán los controles que se le darán al programa para situaciones específicas dentro del código que no son cubiertas por algún otro tipo de excepción y son creadas por el programador y vienen de la clase “Exception”. A continuación, tendremos un ejemplo de excepción utilizada dentro del código del proyecto

Encapsulamiento

Definición

Se refiere a la agrupación de datos con los métodos que operan en esos datos, así también como la restricción del acceso directo a algunos componentes de un objeto. Este concepto es el que permite ocultar el estado de un objeto de datos estructurados dentro de una clase, así, evita el acceso directo a ellos por parte de los clientes de una manera que podría exponer detalles de implementación ocultos.

Clases

Una clase es una plantilla que define un conjunto de variables y métodos apropiados para operar con datos estructurados según un modelo predefinido. Cada objeto creado a partir de una clase se denomina instancia de la clase. Las clases son utilizadas para representar entidades o conceptos, como los sustantivos en el lenguaje, y que permite abstraer los datos y operaciones asociadas al modo de una caja negra.

Paquetes

Es una forma de organizar y agrupar un conjunto de clases y otros elementos relacionados, como lo son las interfaces, subpaquetes y recursos. Estos paquetes se utilizan para evitar conflictos de nombres y para poder proporcionar una estructura jerárquica para el código fuente. En un lenguaje de programación como lo es Java, los paquetes pueden ser utilizados para la organización de las clases de un sistema de archivos de directorios y subdirectorios.

Librerías/Bibliotecas, Métodos Static

Las Librerías/Bibliotecas, son un conjunto de código predefinido que se puede utilizar para la realización de tareas específicas en un programa. Las bibliotecas pueden contener clases, funciones, métodos y otros elementos que pueden ser utilizados para simplificar el desarrollo de software y para reducir el tiempo de programación. Los métodos estáticos, son los que pertenecen a la clase en lugar de instancias específicas de la clase. Se puede invocar un método estático directamente mediante el nombre de la clase, sin tener la necesidad de crear una instancia de esa clase. Estos métodos son comunes en utilidades y funciones que no dependen del estado específico de un objeto.

Constructores

Es un método especial el cual se ejecuta al inicializar un objeto de una clase. Su funcionalidad es inicializar el objeto, es decir, asignar valores iniciales a los atributos del objeto. Estos también pueden recibir parámetros, lo que permite crear diferentes constructores con diferentes conjuntos de parámetros. En algunos lenguajes, como lo es Java, es posible tener más de un constructor en una misma clase, lo que se conoce

como sobrecarga de constructores.

Tipo de constructores

Existen diferentes tipos de constructores y cada uno tiene su propósito específico: Constructor por defecto: Es un constructor que no toma ningún argumento. Es utilizado para inicializar los valores predeterminados de los atributos de la clase cuando se crea un objeto sin proporcionar argumentos. Constructor parametrizado: Este, toma uno o varios argumentos. Se utiliza para permitir una inicialización personalizada de los atributos de la clase al crear un objeto, la cual proporciona valores específicos como argumentos.

Practica o Desarrollo

Se planteó como proyecto hacer una gestión de inmuebles, teniendo una clase madre, tres clases hijas y además una clase principal, el cual tendrá 4 opciones, donde se usará todo lo aprendido en este parcial, usando Arrays, ArraysList, archivos tanto JSON como CSV, los cuales se pueda guardar información y también poder leerla, eliminarla o editarla. Capturas del código correspondiente al proyecto.

4. Conclusiones

El estudio del encapsulamiento revela su importancia como principio fundamental de la programación orientada a objetos. Su aplicación permite agrupar datos y métodos que manipulan esos datos dentro de una unidad, que es la clase, contribuyendo a la seguridad y la facilidad de manejo del código.

La investigación sobre los métodos constructores destaca su papel esencial en las clases, ya que se utilizan para inicializar objetos. Comprender los diferentes tipos de constructores y cómo se utilizan es crucial para la creación eficiente de objetos.

Los métodos getters y setters emergen como una parte integral de la programación orientada a objetos en el estudio. Proporcionan un mecanismo para controlar el acceso a los atributos de un objeto, permitiendo un mayor grado de control y seguridad en el manejo de datos.

5. Recomendaciones

Se recomienda utilizar el encapsulamiento siempre que sea posible para mejorar la seguridad y la facilidad de manejo del código. Es importante asegurarse de que solo las funciones necesarias sean accesibles externamente.

Es crucial entender y utilizar correctamente los constructores al crear clases.

Se aconseja utilizar métodos getters y setters en lugar de acceder directamente a los atributos de un objeto. Esto proporciona un mayor control sobre cómo se accede a los datos y puede ayudar a prevenir errores.

6. Bibliografía/ Referencias

NetBeans. (s/f). Ucm.es. Recuperado el 12 de mayo de 2024, de https://www.fdi.ucm.es/profesor/luis/fp/devtools/NetBeans.html

Jesús. (2024, febrero 21). *Ventajas y Desventajas de Eclipse Java*. Tutoriales Dongee. https://www.dongee.com/tutoriales/ventajas-y-desventajas-de-eclipse-java/

de Zúñiga, F. G. (2024, enero 3). *¿Qué es Visual Studio Code y cuáles son sus ventajas?* Blog de arsys.es; Arsys. https://www.arsys.es/blog/que-es-visual-studio-code-y-cuales-son-sus-ventajas

7. Anexos:

NO APLICA

8. Legalización de documento

   Nombres y Apellidos: Dylan Miguel Sosa Guzmán
   CI: 2300206956
   Firma: Imagen de su firma