UNIVERSIDAD NACIONAL DE COLOMBIA

Manual de Backend - Roomade

Juan José Hernández Medina

Juan Pablo Tejeiro

Santiago Villareal

Sergio Nicolas Vanegas

Universidad Nacional de Colombia Facultad de ingeniería

Bogotá, Colombia

Año 2024

Tabla de contenido:

8.5.7.2. Descripción

8.5.8. Clase CrearItem

8.5.8.1. Código

8.5.8.2. Descripción

Resumen

Este documento detalla el desarrollo de un sistema de visualización 3D para el modelado de habitaciones con el fin de ordenar los elementos indicados en el programa, este programa esta diseñado para brindarle una opción de organización en las habitaciones a los usuarios en función de sus propias necesidades en un determinado cuarto. Este programa utiliza una estructura de software basada en el patrón de arquitectura Modelo-Vista-Controlador (MVC), implementado en Java, junto con una base de datos MySQL. Se describen las tecnologías usadas los retos enfrentados durante el desarrollo, y se presentan posibles mejoras y futuras aplicaciones del sistema.

Introducción

La organización y disposición eficiente de objetos en un entorno determinado es esencial para mejorar la funcionalidad y el confort de un espacio. Con ese mismo argumento, se ha desarrollado un sistema de visualización 3D para el modelado de habitaciones, cuyo objetivo es ofrecer a los usuarios una herramienta que les permita organizar sus habitaciones de acuerdo con sus necesidades específicas. Este programa, denominado Roomade, utiliza una arquitectura de software basada en el patrón MVC, implementado en Java, y se conecta a una base de datos MySQL.

El documento que se presenta describe el proceso de desarrollo de este sistema, destacando las tecnologías utilizadas, como Java y MySQL, así como los desafíos superados durante su implementación. También se discuten las futuras aplicaciones y posibles mejoras del sistema para ofrecer una experiencia de usuario aún más intuitiva y funcional.

Estado del arte

1. **Visualización 3D en diseño de interiores:**

   - **Herramientas de modelado 3D avanzadas:** Existen programas como AutoCAD, SketchUp, y Blender que permiten la creación de modelos 3D de alta precisión, tanto para arquitectos como para diseñadores de interiores. Estos programas son altamente personalizables y ofrecen simulaciones realistas de iluminación, texturas y materiales.

- **Simuladores para usuarios no técnicos:** Aplicaciones como **Planner 5D**, **Roomstyler** y **HomeByMe** están diseñadas para usuarios sin experiencia técnica, permitiéndoles diseñar y organizar habitaciones mediante interfaces amigables e intuitivas. Estas plataformas permiten a los usuarios visualizar y reorganizar muebles y objetos en tiempo real, lo que las hace comparables a Roomade, aunque estas suelen enfocarse más en el diseño estético que en la optimización del uso del espacio.

- **Simuladores para diseñar interiores Online:** Cabe recalcar que existen paginas las cuales están diseñadas para modelar habitaciones en línea mediante interfaces, tales son como **Planificador 3D**.

2. **Arquitectura MVC en Aplicaciones de Diseño:**
    - La adopción de **Modelo-Vista-Controlador (MVC)** facilita el mantenimiento y escalabilidad del sistema, lo que es crucial en aplicaciones de modelado y simulación. Esta arquitectura permite una organización modular que es común en aplicaciones 3D modernas como **Unity** y **Unreal Engine**, que también aplican MVC para manejar la complejidad de las interacciones y renderizado 3D.
3. **Bases de Datos para la Gestión de Objetos**
    - El uso de **MySQL** en "Roomade" para almacenar diseños, objetos y configuraciones sigue una tendencia estándar en la industria, donde la persistencia de datos es clave para la personalización del entorno. Bases de datos relacionales y en la nube como **Firebase** permiten manejar grandes volúmenes de datos de manera eficiente en aplicaciones similares.
4. **Optimización del Uso del Espacio**
    - Sistemas avanzados están comenzando a integrar **inteligencia artificial** para optimizar la disposición de objetos en tiempo real según criterios como la iluminación, tráfico o espacio disponible. Aunque Roomade aún no cuenta con IA avanzada, ya facilita la organización a partir de las necesidades específicas del usuario, lo que lo hace comparable con soluciones más automatizadas.
5. **Simulación de Iluminación y Ambiente**
    - Roomade podría evolucionar implementando tecnologías de simulación avanzada de iluminación, como las usadas por **V-Ray** o **Lumion**, que permiten un renderizado más realista. Esto mejoraría la visualización del espacio bajo diferentes condiciones de luz.
6. **Realidad Virtual y Aumentada (VR/AR)**
    - Una tendencia emergente en la industria es la integración de **VR/AR** para la simulación de espacios interiores. Aplicaciones como **IKEA Place** permiten visualizar muebles en entornos reales. En el futuro, Roomade podría incorporar estas tecnologías para ofrecer experiencias inmersivas

que permitirían a los usuarios visualizar su espacio de forma realista antes de realizar cambios.

Resumen del estado del arte

Roomade se encuentra en una etapa inicial en comparación con las herramientas líderes del mercado, pero tiene un enfoque claro en la personalización y organización espacial en función de las necesidades del usuario. Las tendencias actuales sugieren que futuras versiones podrían incorporar simulaciones más avanzadas, IA para optimización automática, y tecnologías de realidad virtual, lo que lo posicionaría como una herramienta aún más poderosa en el campo del diseño de interiores 3D.

Tecnologías Empleadas

El desarrollo de **Roomade** se basa en una combinación de tecnologías consolidadas que garantizan la funcionalidad y eficiencia del sistema. A continuación, se hace detalle de las principales tecnologías utilizadas:

- **Java**: Es el lenguaje de programación empleado para desarrollar la lógica del negocio, implementar el patrón arquitectónico **Modelo-Vista-Controlador (MVC)** y manejar la interacción con la base de datos. Java es conocido por su robustez y portabilidad, lo que lo convierte en una opción ideal para **Roomade**.
- **JSwing**: Esta biblioteca de Java es la base para la construcción de las interfaces gráficas de usuario (GUI) en **Roomade**. **JSwing** permite la creación de interfaces visuales intuitivas y eficientes para que los usuarios interactúen con el sistema, proporcionando componentes gráficos como botones, tablas y menús, que facilitan la gestión y organización de habitaciones y objetos.
- **JavaFX**: Aunque **JSwing** es responsable de la mayoría de las interfaces gráficas, **JavaFX** se utiliza específicamente para el modelado 3D de objetos en el sistema. JavaFX permite la representación gráfica de los objetos dentro de las habitaciones, mejorando la experiencia visual del usuario. Esta combinación aprovecha las fortalezas de ambas tecnologías para una experiencia de usuario sólida.

- **Jackson**: Utilizado para almacenar y gestionar datos en formato de JSON dentro de la base de datos MySQL. Los objetos JSON permiten guardar estructuras de datos complejas, como las configuraciones de habitaciones y la disposición de elementos, facilitando una rápida lectura y escritura de datos en la base de datos. Este formato flexible y ligero garantiza que la información pueda ser fácilmente procesada y manipulada dentro de la aplicación. En el sistema se utilizan 3 tipos de Jackson:
    - **Jackson Core:** Proporciona las herramientas esenciales para procesar JSON en su forma básica. Se encarga de la lectura y escritura de JSON de manera eficiente, manejando los datos en formato de flujo (streaming).
    - **Jackson Databind:** Facilita la serialización (convertir objetos Java a JSON) y deserialización (convertir JSON a objetos Java). Es el módulo que permite convertir los objetos del sistema, como los Roomades y sus propiedades, en JSON y viceversa.
    - **Jackson Annotations:** Se utiliza para personalizar el proceso de serialización y deserialización mediante el uso de anotaciones en los objetos Java. Esto permite especificar qué atributos deben ser incluidos o excluidos del JSON, así como definir cómo deben ser representados ciertos campos dentro de la estructura JSON.
- **MySQL**: El sistema de gestión de bases de datos relacional utilizado en **Roomade** es **MySQL**, que almacena toda la información relacionada con los usuarios, las configuraciones, los diseños de habitaciones y los objetos. La integración con Java a través de **JDBC** permite una comunicación eficiente entre la aplicación y la base de datos, garantizando la persistencia de los datos y su rápida recuperación.

Gestión de la información

Desde el inicio Roomade ha sido programado con la finalidad de facilitar la manipulación y gestión de datos mediante la base de datos MySQL. A lo largo

del diseño, se han seguido utilizando prácticas estándar para garantizar la integridad y disponibilidad de los datos, implementando métodos eficientes para gestionar la creación, lectura, actualización y eliminación de información (CRUD).

El proyecto ya incluye todas las bibliotecas y dependencias necesarias, lo que elimina la necesidad de configuraciones adicionales. Esto facilita una transición sin problemas desde el entorno de desarrollo hasta la implementación final. La solidez de la arquitectura asegura un manejo seguro de los datos y permite que el sistema sea fácilmente adaptable para futuras actualizaciones o modificaciones, sin requerir grandes cambios en su estructura base.

Base de datos: MySQL

La base de datos denominada **"roomade"** está diseñada para almacenar, gestionar y manipular toda la información relacionada con los usuarios, así como con sus habitaciones personalizadas y elementos (Items). Además, la base de datos gestiona los "Roomades", que son combinaciones de elementos y habitaciones. Esta estructura permite almacenar tanto los nombres como los hashes de estos datos, facilitando su organización y acceso eficiente.

Estructura de base de datos

La base de datos se compone por 4 tablas en donde se guardan los datos:

- **Tabla “users”:**
    - **Descripción:** El fin de esta tabla es almacenar la información básica de los usuarios registrados en la aplicación. Cada registro en esta tabla hace referencia a un único usuario.
    - **Campos:**
        - **usuario:** Nombres únicos de los usuarios, se utiliza como identificador primario. Tipo **VARCHAR (50)**.

        - **contraseña:** Contraseñas de los usuarios. Tipo **VARCHAR (50)**.
        - **correo:** Correos de los usuarios. Tipo **VARCHAR (100)**.
        - **nombre:** Nombres de los usuarios. Tipo **VARCHAR (100)**.
- **Tabla "item":**
    - **Descripción:** La tabla se utiliza para almacenar el nombre y las medidas de los ítems creados por los usuarios, esta misma ayuda a la función de preservar la información.
    - **Campos:**
        - **keyItem:** Es un id para cada objeto de los usuarios, se utiliza como identificador primario para permitir la creación de información de más de un objeto por un mismo usuario. Tipo **INT (11)**, **auto incrementable**.
        - **usuario:** Almacena a los usuarios creadores de los objetos, se utiliza como índice de búsqueda. Tipo **VARCHAR (50)**.
        - **nombre:** Nombres de los objetos. Tipo **VARCHAR (100)**.
        - **base:** Medidas de la base de los objetos. Tipo **FLOAT**.
        - **altura:** Medidas de la altura de los objetos. Tipo **FLOAT**.
        - **profundidad:** Medidas de la profundidad de los objetos. Tipo **FLOAT**.
- **Tabla "room":**
    - **Descripción:** La tabla se utiliza para almacenar el nombre y las medidas de los rooms creados por los usuarios, esta misma ayuda a la función de preservar la información.
    - **Campos:**
        - **keyItem:** Es un id para cada cuarto de los usuarios, se utiliza como identificador primario para permitir la creación de información de más de un objeto por un mismo usuario. Tipo **INT (11)**, **auto incrementable**.
        - **usuario:** Almacena a los usuarios creadores de los objetos, se utiliza como índice de búsqueda. Tipo **VARCHAR (50)**.
        - **nombre:** Nombres de los cuartos. Tipo **VARCHAR (100)**.
        - **base:** Medidas de la base de los cuartos. Tipo **FLOAT**.
        - **altura:** Medidas de la altura de los cuartos. Tipo **FLOAT**.

        - **profundidad:** Medidas de la profundidad de los cuartos. Tipo **FLOAT**.
- **Tabla "rooms":**
    - **Descripción:** La tabla se utiliza para almacenar el nombre y las medidas de los roomades hechos por los usuarios, esta misma resguarda a través de un *hash* todas las medidas, objetos y características de un roomade.
    - **Campos:**
        - **usuario:** Almacena los usuarios creadores de los roomades. Tipo **VARCHAR (50)**.
        - **nombre:** Nombres de los roomades. Tipo **VARCHAR (50)**.
        - **hash:** Almacena una cadena de caracteres de tipo **json**. Tipo **VARCHAR (10000)**.

Integración de librerías

Toda la aplicación se programó con en el IDE **Netbeans**.

- **Integración de JavaFX:**
    - **Descripción general:** En este proyecto se utiliza JavaFX como motor gráfico principal para el funcionamiento del modelado 3D, específicamente toda la integración de físicas, modelado, renderizado e interacción con los objetos.
    - **Configuración para el uso de JavaFX:** Para instalar JavaFX se ingresa a su pagina oficial en el cual se encuentra un paso a paso de como instalar en la IDE de preferencia. El primer paso es descargar todos los jmods los cuales posteriormente se usarán como librerías en el entorno de programación deseado.
      Nota: Se recomienda verificar todas las rutas hacia los jmods de JavaFX.
- **Integración de Jackson:**
    - **Descripción general:** En el proyecto se utilizan 3 tipos de Jackson, **Jackson Databind, Jackson Core** y **Jackson Annotations** para guardar en los hashes los JSON que contienen a su vez toda la información de un roomade de un usuario.

    - **Configuración para el uso de Jackson:** Para instalar los **Jackson** únicamente se agregan como librerías a la IDE y luego se corrigen las rutas de los archivos en el programa.
- **Integración de MySQL:**
    - **Descripción general:** En el proyecto se usa **MySQL** para toda la gestión de información de los usuarios.
    - **Configuración para el uso de MySQL:** En el proyecto para usar la base de datos se integra como librería el archivo de MySQL el cual ayudara en el funcionamiento de la conexión con la base de datos.

Ejecución de la aplicación

Una vez importadas todas las librerías necesarias y configurados los parámetros requeridos para el proyecto, simplemente procede a ejecutar la aplicación Roomade. Al iniciar, tendrás acceso a las diversas funcionalidades, como la creación de diseños personalizados, la gestión de habitaciones (Rooms) y la organización de elementos (Items) dentro de los espacios modelados.

La interfaz te permitirá crear nuevos diseños, fijar objetos dentro de las habitaciones, agregar opciones de iluminación y guardar los resultados en la base de datos. Adicionalmente, podrás cargar diseños previamente guardados, revisar tus Roomades en "Mis Diseños" y gestionar tus habitaciones y objetos de forma eficiente.

La aplicación ha sido diseñada para ofrecer una experiencia fluida e intuitiva, permitiendo a los usuarios organizar sus espacios en base a sus necesidades,

facilitando la personalización de sus habitaciones de manera visual y funcional.

Clases dentro de *controller*

Clase MenuNotView:

- **Código:**

```java
package controller;

import java.io.FileNotFoundException;
import view.Login;
/**
 *
 * @author Juan Pablo Tejeiro, Santiago Villareal, Juan José Hernandez, Sergio Nicolas Vanegas;
 * Grupo Roomade
 *
 */
public class MenuNotView {
    public static void main(String[] args) throws FileNotFoundException {

        Login Logeo = new Login();
        Logeo.setVisible(true);
        Logeo.setLocationRelativeTo(null);

    }
}
```

- **Descripción:** Este código inicia la aplicación mostrando una ventana de inicio de sesión. Se crea una instancia de la clase **Login**, se hace visible y se posiciona en el centro de la pantalla. El método **main** es el punto de entrada para ejecutar la aplicación. Esta clase sirve principalmente para dar inicio al programa.

Clases dentro de *controller.implement*

Clase Abstracta Molde

- **Código:**
  - **Atributos:**

```java
package controller.implement;

/**
 *
 * @author Juan Pablo Tejeiro, Santiago Villareal, Juan José Hernandez, Sergio Nicolas Vanegas;
 * Grupo Roomade
 *
 */

public abstract class Molde {
    private String nombreObjeto;
    private double base;
    private double altura;
    private double profundidad;
```

- **Constructor:**

```
public Molde(String nombreObjeto, double base, double altura, double profundidad) {
    this.nombreObjeto = nombreObjeto;
    this.base = base;
    this.altura = altura;
    this.profundidad = profundidad;
}
```

- **Getters and Setters:**

```
public String getNombreObjeto() {
    return nombreObjeto;
}

public void setNombreObjeto(String NombreObjeto) {
    this.nombreObjeto = NombreObjeto;
}

public double getBase() {
    return base;
}

public double getAltura() {
    return altura;
}

public double getProfundidad() {
    return profundidad;
}

public void setBase(double base) {
    this.base = base;
}

public void setAltura(double altura) {
    this.altura = altura;
}

public void setProfundidad(double profundidad) {
    this.profundidad = profundidad;
}
```

- **@override:**

```
@Override
public String toString() {
    return "RoomItems{" + "base=" + base + ", altura=" + altura + ", profundidad=" + profundidad + '}';
}
```

- **Descripción:** Este código define una clase abstracta llamada Molde, que actúa como una plantilla para objetos con propiedades geométricas. La clase tiene cuatro atributos: nombreObjeto, base, altura, y profundidad, que representan las dimensiones y el nombre del

objeto. También incluye un constructor para inicializar estos atributos y métodos getter y setter para acceder y modificar sus valores.

La clase tiene un método abstracto llamado getArea, que debe ser implementado por las clases derivadas, y un método toString para generar una representación en cadena del objeto.

Clase Roomade

- **Código:**
  - **Importaciones:**

```java
package controller.implement;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;
import model.services.ServicesUsuario;
import java.util.ArrayList;
import java.util.Base64;
import java.util.List;
import java.util.Map;
```

  - **Atributos:**

```java
/**
 *
 * @author Juan Pablo Tejeiro, Santiago Villareal, Juan José Hernandez, Sergio Nicolas Vanegas;
 * Grupo Roomade
 *
 */

public class RooMades{
    // Crea listas para almacenar los valores de cada Room
    private List<Double> bases = new ArrayList<>();
    private List<Double> alturas = new ArrayList<>();
    private List<Double> profundidades = new ArrayList<>();
    private List<Integer> numerosDeObjetos = new ArrayList<>();
    private List<String> nombres = new ArrayList<>();
```

- **Roomades:**

```
public RooMades(List<RoomJFX> roomJFX) {
    for (int i = 0; i < roomJFX.size(); i++) {  // Corregido: Empezar desde 0
        Room room = MedidasRoom(roomJFX.get(i).getHash());

        if (room != null) {
            nombres.add(roomJFX.get(i).getNombre());
            bases.add(room.getBase());
            alturas.add(room.getAltura());
            profundidades.add(room.getProfundidad());
            numerosDeObjetos.add(NumeroDeObjetosRoom(roomJFX.get(i).getHash()));
        } else {
            System.err.println("Room no encontrado para el hash: " + roomJFX.get(i).getHash());
            // Añade valores predeterminados si el room no es encontrado
            nombres.add("Desconocido");
            bases.add(0.0);
            alturas.add(0.0);
            profundidades.add(0.0);
            numerosDeObjetos.add(0);
        }
    }
}
```

- **Getters and Setters:**

```
public List<Double> getBases() {
    return bases;
}

public void setBases(List<Double> bases) {
    this.bases = bases;
}

public List<Double> getAlturas() {
    return alturas;
}

public void setAlturas(List<Double> alturas) {
    this.alturas = alturas;
}

public List<Double> getProfundidades() {
    return profundidades;
}

public void setProfundidades(List<Double> profundidades) {
    this.profundidades = profundidades;
}

public List<Integer> getNumerosDeObjetos() {
    return numerosDeObjetos;
}

public void setNumerosDeObjetos(List<Integer> numerosDeObjetos) {
    this.numerosDeObjetos = numerosDeObjetos;
}

public List<String> getNombres() {
    return nombres;
}

public void setNombres(List<String> nombres) {
    this.nombres = nombres;
}
```

- **decodeObjectMedidasRoom:**

```
private Room decodeObjectMedidasRoom(Map<String, Object> objectData) {
    String type = (String) objectData.get("type");
    Room room = null;
    if (type.equals("Box")){
        room = new Room(
            (String) ServicesUsuario.getUsuario(),
            (double) objectData.get("width"),
            (double) objectData.get("height"),
            (double) objectData.get("depth")
            );
        return room;
    }
    return room;
}
```

- **decodeObjectNumeroDeItems:**

```
private int decodeObjectNumeroDeItems(Map<String, Object> objectData, int i) {
    String type = (String) objectData.get("type");
    Room room = null;
    switch (type) {

        case "ItemNuevo" -> {
            i++;
            return i;
        }

        case "Sphere" ->{
            i++;
            return i;
        }
        case "Cylinder" ->{
            i++;
            return i;
        }
        case "Group" -> {
            i++;
            return i;
        }
        case "MesaDeNoche" -> {
            i++;
            return i;
        }
        case "Mueble" -> {
            i++;
            return i;
        }
        case "Armario" -> {
            i++;
            return i;
        }
```

```
            case "Mesa" -> {
                i++;
                return i;
            }
            case "Silla" -> {
                i++;
                return i;
            }
            case "CamaSimple" -> {
                i++;
                return i;
            }
            case "CamaDoble" -> {
                i++;
                return i;
            }
            case "TV" -> {
                i++;
                return i;
            }
            case "TVGrande" -> {
                i++;
                return i;
            }
        }
        return i;
    }
```

- **MedidasRoom:**

```
private Room MedidasRoom(String encodedJson) {
    ObjectMapper objectMapper = new ObjectMapper();
    Room room = null;
    Room backupRoom = null;  // Para almacenar un room que usaremos como backup para reemplazar
    try {
        byte[] decodedBytes = Base64.getDecoder().decode(encodedJson);
        String json = new String(decodedBytes);
        Map<String, Object> sceneData = objectMapper.readValue(json, new TypeReference<Map<String, Object>>() {});

        if (sceneData == null || !sceneData.containsKey("objects")) {
            System.err.println("El JSON no contiene la clave 'objects'");
            return null;
        }

        List<Map<String, Object>> objectsData = (List<Map<String, Object>>) sceneData.get("objects");
        if (objectsData == null || objectsData.isEmpty()) {
            System.err.println("La lista 'objects' está vacía o es nula.");
            return null;
        }

        for (Map<String, Object> objectData : objectsData) {
            room = (Room) decodeObjectMedidasRoom(objectData);
            if (room != null) {
                // Si encontramos un objeto con 10.0 en alguna medida, verificamos el siguiente objeto para reemplazar esa medida
                if (room.getBase() == 10.0 || room.getAltura() == 10.0 || room.getProfundidad() == 10.0) {
                    backupRoom = findNextValidRoom(objectsData, room);  // Buscar otro Room para tomar la medida válida
                    if (backupRoom != null) {
                        // Reemplazar solo la medida que es 10.0 con la del siguiente objeto válido
                        if (room.getBase() == 10.0 && backupRoom.getBase() != 10.0) {
                            room.setBase(backupRoom.getBase());
                        }
                        if (room.getAltura() == 10.0 && backupRoom.getAltura()!= 10.0) {
                            room.setAltura(backupRoom.getAltura());
                        }
                        if (room.getProfundidad() == 10.0 && backupRoom.getProfundidad()!= 10.0) {
                            room.setProfundidad(backupRoom.getProfundidad());
                        }
                    }
                }
                // Cuando encontramos un room válido con las medidas corregidas, salimos del bucle
                break;
            }
        }

    } catch (JsonProcessingException e) {
        System.out.println(e);
    }
    return room;
}
```

- **findNextValidRoom:**

```
private Room findNextValidRoom(List<Map<String, Object>> objectsData, Room currentRoom) {
    for (Map<String, Object> objectData : objectsData) {
        Room room = (Room) decodeObjectMedidasRoom(objectData);
        // Asegurarse de que el room siguiente tenga medidas válidas
        if (room != null && room.getAltura() != currentRoom.getAltura()) {
            return room;  // Retornar el siguiente room válido
        }
    }
    return null;  // Si no se encuentra otro room válido
}
```

- **NumeroDeObjetosRoom:**

```
private int NumeroDeObjetosRoom(String encodedJson) {
    // Aquí deberías obtener el 'encodedJson' desde un archivo o entrada del usuario
    ObjectMapper objectMapper = new ObjectMapper();
    int i = 0;
    try {
        byte[] decodedBytes = Base64.getDecoder().decode(encodedJson);
        String json = new String(decodedBytes);
        Map<String, Object> sceneData = objectMapper.readValue(json, new TypeReference<Map<String, Object>>() {});

        List<Map<String, Object>> objectsData = (List<Map<String, Object>>) sceneData.get("objects");
        for (Map<String, Object> objectData : objectsData) {
            i = (int) decodeObjectNumeroDeItems(objectData, i);


        }
        return i;
    } catch (JsonProcessingException e) {
        System.out.println(e);
    }
    return i;
}
```

- **Descripción:** Estos códigos pertenecen a la clase RooMades, que está diseñada para manejar la carga y el procesamiento de objetos de tipo RoomJFX. La clase realiza varias funciones relacionadas con la extracción de información de habitaciones (rooms) en un sistema de modelado 3D para el proyecto 'RooMade', esta clase se usa para ayudar a la gestión de visualización de datos en “Mis Diseños”. A continuación, se describen los principales aspectos del código:
    - **Atributos:** La clase RooMades mantiene listas para almacenar las dimensiones (bases, alturas, profundidades) y nombres de los objetos, así como el número de objetos (numerosDeObjetos) para cada habitación.
    - **Constructor:** El constructor recibe una lista de RoomJFX y, para cada elemento, obtiene las dimensiones del objeto de la habitación (usando el método MedidasRoom) y el número de objetos en la habitación (usando el método NumeroDeObjetosRoom). Si no se encuentra un Room, se agregan valores predeterminados.
    - **decodeObjectMedidasRoom:** Este método convierte un objeto de datos en un objeto Room (representando una habitación), verificando el tipo de objeto (ejemplo: "Box").
    - **decodeObjectNumeroDeItems:** Este método procesa un mapa de datos de un objeto y aumenta un contador de acuerdo con el tipo de objeto que encuentra. Este contador se usa para llevar el registro del número de objetos en una habitación.

- **findNextValidRoom:** Este método se usa para buscar la siguiente habitación con medidas válidas en una lista de objetos de habitación.
- **MedidasRoom:** Decodifica un JSON en Base64 que contiene los datos de una habitación, extrae las dimensiones y, si alguna de las medidas es un valor inválido (10.0), trata de corregirlo usando otro objeto de la lista (findNextValidRoom).
- **NumeroDeObjetosRoom:** Decodifica el JSON de la escena para contar el número de objetos presentes en la habitación, utilizando el método **decodeObjectNumeroDeItems,** que aumenta el contador según el tipo de objeto (ejemplos: "ItemNuevo", "Sphere", "Mesa").

Este código facilita la manipulación de datos de habitaciones en un sistema de visualización 3D, decodificando información desde un formato JSON y ajustando medidas cuando se encuentran valores incorrectos.

RooMakingJFX3D:

OrdenarObjetos()

Descripción: Organiza aleatoriamente objetos movibles dentro del cuarto, evitando solapamientos y considerando la iluminación disponible.

Detalles:

- Define rangos de ubicación posibles en base a las dimensiones de las paredes.
- Actualiza el estado de iluminación del cuarto.
- Coloca objetos como ventanas de forma fija y otros objetos en posiciones aleatorias no solapadas.

- Verifica que no haya superposición entre objetos utilizando un margen de seguridad.
- Utiliza hasta 100 intentos para encontrar una posición adecuada para cada objeto.

actualizarEstadoIluminacion()

Descripción: Determina qué luces o ventanas están presentes en las paredes y techo del cuarto.

Detalles:

- Verifica los nodos hijos de root3D para identificar si hay luces en las paredes y techo, y ventanas en las diferentes posiciones.
- Actualiza variables booleanas (contieneLuzPared1, contieneVentana1, etc.) según lo que se encuentre.

determinarPosicionSegunIluminacion(Movible movible, double minX, double maxX, double minZ, double maxZ)

Descripción: Asigna posiciones a los objetos en función de la iluminación disponible en las paredes.

Detalles:

- Si el objeto es un Armario, se asigna a una pared sin luz.
- Si es un Mueble, se posiciona opuesto a una pared con luz o ventana.
- Las Camas se colocan preferiblemente en paredes sin ventana.

- Otros objetos se colocan aleatoriamente si no hay restricciones.

determinarRotacion(double x, double z, double minX, double maxX, double minZ, double maxZ)

Descripción: Calcula la rotación que debe tener un objeto según su posición en el cuarto.

Detalles:

- Retorna la rotación basada en la posición del objeto respecto a las paredes (minX, maxX, minZ, maxZ).
- Si está cerca de minX, rota a 0 grados; si está cerca de maxX, rota a 180 grados, y así sucesivamente.

isOverlapping(Movible obj1, Movible obj2)

Descripción: Verifica si dos objetos Movible están solapados en el espacio 3D.

Detalles:

- Utiliza los límites (Bounds) de ambos objetos para comprobar si se intersectan en sus posiciones.
- Retorna true si hay superposición, de lo contrario, false.

showDimensionDialog(String objectType)

Descripción: Muestra un diálogo para ingresar las dimensiones del objeto a crear.

Detalles:

- Crea una ventana que solicita las dimensiones de un objeto según su tipo (Cilindro, Esfera, Cubo).
- Dependiendo del tipo de objeto, ciertos campos de dimensión se habilitan o deshabilitan.
- Al hacer clic en "Crear", se validan las dimensiones y se añade el objeto con las especificaciones dadas.
- Si las dimensiones son inválidas, se muestra un error con un Alert.

showRoomDialog()

Descripción: Muestra un diálogo para seleccionar una habitación (Room) en la cual trabajar.

Detalles:

- Muestra una tabla con una lista de habitaciones disponibles (Room).
- Permite seleccionar una habitación sobre la cual el usuario desea trabajar.
- Se configura la tabla con columnas y celdas para mostrar las propiedades de las habitaciones.

showItemDialog()

Descripción: Muestra un diálogo que contiene una tabla con los ítems creados previamente. Permite seleccionar uno para crear un nuevo objeto en la escena.

Detalles:

- Crea un diálogo con una tabla que lista los objetos de tipo Item, incluyendo sus dimensiones (base, altura, profundidad).
- El usuario selecciona un ítem de la tabla, y al hacer clic en "Crear", se genera un nuevo objeto en la escena.
- Verifica si el objeto seleccionado ya existe en la escena, y si es así, lo elimina antes de agregar el nuevo.
- Configura las posiciones iniciales del objeto y habilita eventos de mouse (OnMousePressed, OnMouseDragged).
- El objeto también incluye un menú contextual para acciones adicionales.

createFloor(double width, double height, double depth)

Descripción: Crea el objeto de piso de la escena con las dimensiones dadas.

Detalles:

- Genera un Box para representar el piso, con el material de color gris.
- El piso se coloca en la posición Y definida por la variable floorY.

createWallBack(double width, double height, double depth)

Descripción: Crea la pared trasera de la habitación con las dimensiones especificadas.

Detalles:

- Genera un Box para representar la pared trasera, con un grosor fijo de 10 unidades.
- La pared se posiciona en el eje Z de la habitación en el centro de profundidad (depth/2).

createWallLeft(double width, double height, double depth)

Descripción: Crea la pared izquierda de la habitación con las dimensiones dadas.

Detalles:

- Genera un Box para la pared izquierda, con un grosor fijo de 10 unidades.
- Posiciona la pared a la izquierda del cuarto, moviéndola al eje X negativo a la mitad del ancho (-width/2).
-

createItemNuevo(double width, double height, double depth)

Descripción: Crea un nuevo objeto ItemNuevo con las dimensiones especificadas.

Detalles:

- Crea una instancia de la clase ItemNuevo con las dimensiones de base, altura, y profundidad.
- Este método facilita la creación de nuevos objetos de tipo ItemNuevo para la escena.

addObject(String objectType, double width, double height, double depth)

Descripción: Añade un nuevo objeto a la escena en base al tipo de objeto seleccionado (Prisma, Esfera, Cilindro).

Detalles:

- Dependiendo del tipo de objeto seleccionado (Prisma, Esfera, Cilindro), crea el objeto adecuado (Sphere3D, Cylinder3D).
- Configura las posiciones iniciales del objeto y los eventos de mouse (OnMousePressed, OnMouseDragged, OnMouseReleased).
- Asigna un menú contextual al objeto para acciones adicionales.
- Añade el nuevo objeto al nodo root3D de la escena.

addGroup(String groupType)

Descripción: Añade un grupo de objetos predefinidos a la escena (e.g., Cama Simple, Armario, Televisor).

Detalles:

- Dependiendo del tipo de grupo seleccionado (Cama Simple, Cama Doble, Mesa de Noche, etc.), crea la instancia correspondiente.
- Configura las posiciones iniciales del objeto y los eventos de mouse (OnMousePressed, OnMouseDragged).
- Añade el objeto al nodo root3D de la escena.
- También incluye un menú contextual para interacciones adicionales con el grupo.

cloneGroup(Group original)

Descripción: Clona un grupo de objetos, creando una copia exacta de los nodos contenidos en el grupo original.

Detalles:

- Recorre los nodos hijos del grupo original. Si el nodo es un Box, lo clona creando un nuevo Box con las mismas dimensiones y material.
- Posiciona el clon en la misma ubicación que el objeto original, replicando las coordenadas de translateX, translateY, y translateZ.
- Devuelve el nuevo grupo clonado.

setupContextMenu(Shape3D object)

Descripción:

Configura un menú contextual para un objeto de tipo Shape3D. El menú permite eliminar o rotar el objeto.

Detalles:

- Eliminar: Elimina el objeto del grupo root3D cuando se selecciona la opción "Eliminar".
- Rotar: Rota el objeto 90 grados alrededor del eje Y cuando se selecciona la opción "Rotar".

setupContextMenu(Group group)

Descripción:

Configura un menú contextual para un grupo de objetos Group. El menú permite eliminar o rotar el grupo.

Detalles:

- Eliminar: Elimina el grupo del grupo root3D cuando se selecciona la opción "Eliminar".
- Rotar: Rota el grupo 90 grados alrededor del eje Y cuando se selecciona la opción "Rotar".

rotateObjectY(Shape3D object)

Descripción:

Rota un objeto de tipo Shape3D 90 grados alrededor del eje Y.

Detalles:

- Aplica una rotación de 90 grados alrededor del eje Y al objeto especificado.

rotateObjectY(Group group)

Descripción:

Rota un grupo de objetos Group 90 grados alrededor del eje Y.

Detalles:

- Aplica una rotación de 90 grados alrededor del eje Y al grupo especificado.

handleObjectPressed(MouseEvent event)

Descripción:

Maneja el evento de presionar un objeto de tipo Movible. Calcula el desplazamiento del ratón para el movimiento posterior del objeto.

Detalles:

- Solo se maneja el evento si se presiona el botón izquierdo del ratón.
- Calcula el desplazamiento en X, Y, y Z para ajustar la posición del objeto.

handleObjectDragged(MouseEvent event)

Descripción:

Maneja el evento de arrastrar un objeto de tipo Movible. Actualiza la posición del objeto basándose en el movimiento del ratón.

Detalles:

- Movimiento: Actualiza las coordenadas X y Y del objeto, y opcionalmente la Z si se presiona la tecla Shift.

handleGroupPressed(MouseEvent event)

Descripción:

Maneja el evento de presionar un grupo de objetos Movible. Calcula el desplazamiento del ratón para el movimiento posterior del grupo.

Detalles:

- Solo se maneja el evento si se presiona el botón izquierdo del ratón.
- Desplazamiento: Calcula el desplazamiento en X, Y, y Z para ajustar la posición del grupo.

handleGroupDragged(MouseEvent event)

Descripción:

Maneja el evento de arrastrar un grupo de objetos Movible. Actualiza la posición del grupo basándose en el movimiento del ratón.

Detalles:

- Actualiza las coordenadas X y Y del grupo, y opcionalmente la Z si se presiona la tecla Shift.
- Solo realiza el ajuste si el objeto está marcado como fijo (fijo).

togglePositionsLock()

Descripción:

Alterna el estado de "fijo" para todos los objetos Movible en root3D.

Detalles:

- Cambia el estado de fijo para permitir o impedir el movimiento de los objetos.

handleMousePressed(MouseEvent event)

Descripción:

Maneja el evento de presionar el ratón. Guarda las coordenadas del ratón para la manipulación posterior.

Detalles:

- Guarda las coordenadas X e Y del ratón en las variables mouseX y mouseY.

handleMouseDragged(MouseEvent event)

Descripción:

Maneja el evento de arrastrar el ratón. Rota el grupo root3D basado en el movimiento del ratón.

Detalles:

- Calcula la rotación en función del desplazamiento del ratón y actualiza los ángulos de rotación rotateX y rotateY.

rotateGroup(double deltaX, double deltaY)

Descripción:

Rota el grupo root3D en función del desplazamiento del ratón.

Detalles:

- Ajusta los ángulos de rotación rotateX y rotateY según los valores deltaX y deltaY.

unproject(double sceneX, double sceneY, double z)

Descripción:

Convierte las coordenadas de la escena a coordenadas del mundo 3D.

Detalles:

- Utiliza la cámara y las transformaciones para calcular la posición en el espacio 3D a partir de las coordenadas de la escena.

handleScroll(ScrollEvent event)

Descripción:

Maneja el evento de desplazamiento del ratón. Ajusta la posición Z de la cámara para hacer zoom.

Detalles:

- Ajusta la distancia de la cámara (cameraExterna) en función del desplazamiento del ratón.

startPhysics()

Descripción:

Inicia una animación continua para aplicar física a los objetos en root3D.

Detalles:

- Calcula la gravedad y las colisiones en cada frame utilizando un Timeline.

applyGravity(javafx.scene.Node node)

Descripción:

Aplica la gravedad a un objeto (node). Ajusta la posición Y del objeto si está por debajo del nivel del suelo (floorY).

Detalles:

- Ajusta la posición Y del objeto para simular la gravedad, asegurándose de que no caiga por debajo del nivel del suelo.

getObjectHeight(javafx.scene.Node node)

Descripción:

Calcula la altura de un objeto en función de su tipo (Box, Sphere, Cylinder, o Group).

Detalles:

- Devuelve la altura del objeto o del grupo de objetos, considerando sus transformaciones.

detectCollisions(javafx.scene.Node node)

Descripción: Detecta y resuelve las colisiones entre un nodo dado y otros nodos en la escena.

Detalles:

- Itera sobre todos los nodos hijos en root3D.
- Si el nodo actual no es el mismo que el nodo dado y el nodo actual es una instancia de Shape3D o Group, verifica si hay una colisión entre los dos nodos utilizando el método isColliding.
- Si se detecta una colisión, llama al método resolveCollision para resolverla.

isColliding(javafx.scene.Node node1, javafx.scene.Node node2)

Descripción: Verifica si dos nodos colisionan basándose en sus cajas delimitadoras (Bounding Boxes).

Detalles:

- Obtiene las cajas delimitadoras de ambos nodos utilizando getBoundsInParent.
- Comprueba si las cajas delimitadoras se intersectan.
- Retorna true si hay intersección; de lo contrario, false.

resolveCollision(javafx.scene.Node node1, javafx.scene.Node node2)

Descripción: Resuelve la colisión entre dos nodos moviéndolos para evitar la superposición.

Detalles:

- Obtiene las cajas delimitadoras de ambos nodos.
- Calcula las diferencias y superposiciones en los ejes X, Y y Z.
- Si uno de los nodos es el floor, wallBack o wallLeft, mueve los nodos verticalmente para separarlos.
- Si no, ajusta las posiciones de los nodos según el eje con la menor superposición para resolver la colisión.

saveScene()

Descripción: Guarda el estado actual de la escena en formato JSON codificado en Base64.

Detalles:

- Crea un mapa que contiene la información de todos los nodos en root3D.
- Codifica esta información en un formato JSON y luego en Base64.
- Imprime el JSON codificado en la consola y lo retorna.

loadScene(String encodedJson)

Descripción: Carga una escena a partir de una cadena JSON codificada en Base64.

Detalles:

- Decodifica el JSON de Base64 a una cadena.
- Convierte la cadena JSON en un mapa de datos de la escena.

- Limpia root3D y agrega los nodos decodificados a la escena.

encodeObject(javafx.scene.Node node)

Descripción: Codifica la información de un nodo en un mapa que puede ser convertido a JSON.

Detalles:

- Obtiene el tipo de nodo y sus propiedades de transformación (traslación y rotación).
- Codifica propiedades específicas según el tipo de nodo (Shape3D, Group, PointLight).

encodeShape3D(Shape3D shape, Map<String, Object> objectData)

Descripción: Codifica la información específica de un objeto Shape3D en un mapa.

Detalles:

- Codifica las propiedades del material y las dimensiones del objeto según su tipo (Box, Sphere, Cylinder).

encodeGroup(Group group, Map<String, Object> objectData)

Descripción: Codifica la información de un grupo de nodos en un mapa.

Detalles:

- Itera sobre los nodos hijos del grupo y codifica cada uno.
- Agrega la lista de nodos hijos codificados al mapa del grupo.

encodePointLight(PointLight light, Map<String, Object> objectData)

Descripción: Codifica la información de un objeto PointLight en un mapa.

Detalles:

- Codifica el color de la luz en formato hexadecimal.

encodeTransforms(javafx.scene.Node node, Map<String, Object> objectData)

Descripción: Codifica las transformaciones de un nodo en un mapa.

Detalles:

- Itera sobre las transformaciones del nodo.
- Codifica transformaciones de tipo Rotate en el mapa.

decodeObject(Map<String, Object> objectData)

Descripción: Decodifica un mapa en un nodo de JavaFX.

Detalles:

- Crea un nuevo nodo basado en el tipo especificado en el mapa.
- Configura las propiedades del nodo y sus hijos según la información del mapa.

createShape3D(String type, Map<String, Object> objectData)

Descripción: Crea un objeto Shape3D basado en el tipo especificado y la información del mapa.

Detalles:

- Crea el objeto Shape3D correspondiente (Box, Sphere, Cylinder).
- Configura el color y otras propiedades.

setColorForShape3D(Shape3D shape, Map<String, Object> objectData)

Descripción: Configura el color para un objeto Shape3D basado en la información del mapa.

Detalles:

- Establece el color del material del objeto Shape3D si el color está presente y es válido.

createGroupNode(String type, Map<String, Object> objectData)

Descripción: Crea un objeto Group basado en el tipo especificado y la información del mapa.

Detalles:

- Crea el objeto Group correspondiente y agrega los nodos hijos.
- Configura propiedades adicionales para grupos específicos como ItemNuevo.

createPointLight(Map<String, Object> objectData)

Descripción: Crea un objeto PointLight basado en la información del mapa.

Detalles:

- Configura el color de la luz basado en la información del mapa.

setCommonProperties(javafx.scene.Node node, Map<String, Object> objectData)

Descripción: Configura las propiedades comunes de un nodo 3D en JavaFX a partir de un mapa de datos.

Detalles:

- Establece las propiedades de traducción (TranslateX, TranslateY, TranslateZ) del nodo basándose en los valores proporcionados en el mapa.
- Aplica transformaciones adicionales como rotaciones si están especificadas en los datos. Para cada rotación, se configuran ángulo, pivote y eje de rotación.

getDoubleOrDefault(Map<String, Object> data, String key, double defaultValue)

Descripción: Obtiene un valor numérico de tipo double de un mapa o devuelve un valor predeterminado.

Detalles:

- Busca el valor asociado con la clave proporcionada en el mapa.
- Si el valor es un número, lo convierte a double. Si no, devuelve un valor predeterminado.

showCargarDialog()

Descripción: Muestra un diálogo para cargar configuraciones guardadas desde la base de datos.

Detalles:

- Crea una ventana secundaria con un ComboBox para seleccionar opciones de configuración.
- Muestra un botón para cargar la configuración seleccionada.
- Cuando se selecciona una opción, el botón se hace visible y permite cargar la configuración correspondiente.

cargarOpcionesDesdeBD()

Descripción: Carga las opciones disponibles de configuraciones guardadas desde la base de datos.

Detalles:

- Recupera las habitaciones guardadas y sus hashes desde la base de datos.
- Si hay configuraciones guardadas, agrega sus nombres y hashes a listas y las devuelve en un mapa.

opcionesIluminacion(Box wallLeft, Box wallBack)

Descripción: Muestra un diálogo para configurar la iluminación en diferentes paredes y el techo.

Detalles:

- Crea una ventana secundaria con ComboBox para seleccionar opciones de iluminación para las paredes y el techo.
- Muestra un botón para aplicar la configuración seleccionada.

- Cuando se seleccionan todas las opciones, el botón se hace visible y permite aplicar la configuración.

verificarSeleccion(ComboBox<String> pared1ComboBox, ComboBox<String> pared2ComboBox, ComboBox<String> pared3ComboBox, ComboBox<String> techoComboBox, Button aplicarButton)

Descripción: Verifica si todas las opciones de iluminación están seleccionadas y activa el botón de aplicar si es así.

Detalles:

- Si todas las ComboBox tienen un valor seleccionado, hace visible el botón de aplicar; de lo contrario, lo oculta.

configuracionIluminacion(String pared1, String pared2, String pared3, String techo, Box wallLeft, Box wallBack)

Descripción: Configura la iluminación en el escenario 3D basado en las selecciones de las paredes y el techo.

Detalles:

- Elimina las luces existentes.
- Configura nuevas luces o ventanas en función de las selecciones para cada pared y el techo.
- Ajusta la posición y añade las luces o ventanas a la escena según la selección.

addItemToScene(Group itemGroup)

Descripción: Agrega un grupo de ítems a la escena 3D y configura sus eventos.

Detalles:

- Inicializa la posición del grupo de ítems en el origen (0, 0, 0).
- Configura los eventos para manejar la interacción con el grupo (presionar y arrastrar).