



Manual de Backend-Roomade

Juan José Hernández Medina

Juan Pablo Tejeiro

Santiago Villarreal

Sergio Nicolas Vanegas

Universidad Nacional de Colombia
Facultad de ingeniería
Bogotá, Colombia Año 2024

Tabla de contenido:

- 1. Resumen.**
- 2. Introducción**
- 3. Estado del arte**
 - 3.1. Resumen del estado del arte**
- 4. Tecnologías empleadas**
- 5. Gestión de información**
 - 5.1. Base de datos: MySQL**
 - 5.2. Estructura de base de datos**
- 6. Integración de librerías**
- 7. Ejecución de la aplicación**
- 8. Clases del programa por package**
 - 8.1. Clases de controller**
 - 8.1.1. Clase MenuNotView**
 - 8.1.1.1. Código**
 - 8.1.1.2. Descripción**
 - 8.2. Clases de controller.implement**
 - 8.2.1. Clase Item**
 - 8.2.1.1. Código**
 - 8.2.1.2. Descripción**
 - 8.2.2. Clase Abstracta Molde**
 - 8.2.2.1. Código**
 - 8.2.2.2. Descripción**
 - 8.2.3. Clase RooMades**
 - 8.2.3.1. Código**
 - 8.2.3.2. Descripción**
 - 8.2.4. Clase Room**
 - 8.2.4.1. Código**
 - 8.2.4.2. Descripción**
 - 8.2.5. Clase RoomJFX**
 - 8.2.5.1. Código**
 - 8.2.5.2. Descripción**
 - 8.2.6. Clase Usuario**
 - 8.2.6.1. Código**
 - 8.2.6.2. Descripción**
 - 8.3. Clases de model**
 - 8.3.1. Clase Mysql_BD**
 - 8.3.1.1. Código**

8.3.1.2. Descripción

8.4. Clases de model.services

8.4.1. Clase abstract Services

8.4.1.1. Código

8.4.1.2. Descripción

8.4.2. Clase ServicesItem

8.4.2.1. Código

8.4.2.2. Descripción

8.4.3. Clase ServicesRoom

8.4.3.1. Código

8.4.3.2. Descripción

8.4.4. Clase ServicesRoomJFX

8.4.4.1. Código

8.4.4.2. Descripción

8.4.5. Clase ServicesUsuario

8.4.5.1. Código

8.4.5.2. Descripción

8.5. Clases de View

8.5.1. Clase ConfiguracionView

8.5.1.1. Código

8.5.1.2. Descripción

8.5.2. Clase CrearDiseño

8.5.2.1. Código

8.5.2.2. Descripción

8.5.3. Clase Login

8.5.3.1. Código

8.5.3.2. Descripción

8.5.4. Clase Menu

8.5.4.1. Código

8.5.4.2. Descripción

8.5.5. Clase MisDiseños

8.5.5.1. Código

8.5.5.2. Descripción

8.5.6. Clase RooMaking

8.5.6.1. Código

8.5.6.2. Descripción

8.5.7. Clase RooMakingJFX3D

8.5.7.1. Código

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 está 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.

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 y de objetos en el sistema, además permite gestionar la escena 3D. 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":

- o **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.
- o **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”:

- o **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.
- o **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:

- o **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.
- o **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:

- o **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.

- o **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:

- o **Descripción general:** En el proyecto se usa **MySQL** para toda la gestión de información de los usuarios.
- o **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 ayudará 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:

```
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);

    }
}
```

Código:

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:

o Atributos:

```
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;
```

o Constructor:

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

o 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;  
}
```

o @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:

o Importaciones:

```
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;
```

o Atributos:

```
/**
 *
 * @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<>();
```

o 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);  
        }  
    }  
}
```

o 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;  
}
```

o **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;  
}
```

o 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;
}
```

o 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;
}
```

o 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
}
```

o 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:

- o **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.
- o **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.
- o **decodeObjectMedidasRoom:** Este método convierte un objeto de datos en un objeto Room (representando una habitación), verificando el tipo de objeto (ejemplo: "Box").
- o **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.

- o **findNextValidRoom:** Este método se usa para buscar la siguiente habitación con medidas válidas en una lista de objetos de habitación.
- o **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).
- o **NumeroDeObjetosRoom:** Decodifica el JSON de la escena para contar el número de objetos presentes en la habitación, utilizando el método **decodeObjectNumeroDeltems**, 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.

Clases dentro de *model*

Clase Mysql_BD:

Codigo:

```

1  package model;
2
3  import java.sql.Connection;
4  import java.sql.DriverManager;
5  import java.sql.SQLException;
6
7  public class Mysql_BD {
8
9      private static Connection conn;
10     private static final String driver = "com.mysql.cj.jdbc.Driver";
11     private final String url = "jdbc:mysql://localhost:3306/roomade";
12     private final String user = "root";
13     private final String password = "";
14
15     public Connection conectar() {
16         try {
17             Class.forName(driver);
18             conn = DriverManager.getConnection(url, user, password);
19             if(conn!=null){
20                 System.out.println("Conexion establecida :D en: " + this.toString());
21             }
22             return conn;
23         } catch (ClassNotFoundException | SQLException e) {
24             System.out.println("Error en la conexión: " + e.getMessage());
25             throw new RuntimeException(e);
26             //return null;
27         }
28     }
29 }
30
31 
```

Descripción: Esta clase maneja la conexión con una base de datos MySQL. Contiene los

detalles necesarios para establecer una conexión y permite interactuar con la base de datos alojada en un servidor local. Esta clase encapsula los métodos y atributos que se requieren para conectarse a la base de datos y gestionar cualquier error durante el proceso de conexión.

Atributos:

- `private static Connection conn:`

Variable estática que almacena la conexión actual con la base de datos.

- `private static final String driver:`

Contiene el nombre del driver que se utiliza para conectarse a la base de datos MySQL (`com.mysql.cj.jdbc.Driver`).

- `private final String url:`

Almacena la URL de la base de datos a la cual se intenta conectar. En este caso, la base de datos está alojada en un servidor local con el nombre `roomade`.

- `private final String user:`

Almacena el nombre del usuario que se utiliza para autenticarse en la base de datos. El valor por defecto es `"root"`.

- `private final String password:`

Almacena la contraseña correspondiente al usuario de la base de datos. En este caso, está vacío (`""`).

Métodos:

- **Conectar:**

Establece una conexión con la base de datos MySQL usando los detalles almacenados en los atributos de la clase. Si la conexión es exitosa, imprime un mensaje en la consola indicando que la conexión ha sido establecida. Si ocurre un error durante el proceso, maneja las excepciones y lanza un `RuntimeException`.

Detalles:

- Intenta cargar el driver de MySQL utilizando la clase `Class.forName(driver)`.
- Establece una conexión usando `DriverManager.getConnection(url, user,`

password).

- En caso de error, maneja dos posibles excepciones:
 - `ClassNotFoundException`: Si el driver no se encuentra.
 - `SQLException`: Si ocurre un error relacionado con la conexión a la base de datos.

Clases dentro de *model.services*

Clase Services (Abstracta):

Código:

```
1 package model.services;
2
3 import controller.implement.Molde;
4
5 public abstract class Services {
6     public abstract void ActualizarDatos(String Usuario, Molde molde);
7 }
8
```

Descripción:

Esta clase abstracta es una clase en desarrollo cuya implementación vendrá en versiones posteriores y pretende especificar los métodos generales para todas las clases “Services”.

Métodos:

- **ActualizarDatos:**

Código:

```
8 public abstract void ActualizarDatos(String Usuario, Molde molde);
}
```

Descripción:

Se espera implementar para la actualización de datos de las clases “Services”, es un método abstracto.

Clase Services Item:

Código:

```
20 public class ServicesItem extends Services {
21     private final String tabla_Item = "item";
22     private static List<Item> items = new ArrayList<>();
23     private static List<Item> itemsRoom = new ArrayList<>();
24
25     Mysql_BD bd = new Mysql_BD();
26
27     public static List<Item> getItems() {...3 lines }
30
31     public static void setItems(Item item) {...3 lines }
34
35     public static List<Item> getItemsRoom() {...3 lines }
38
39     Connection con = bd.conectar();
40     public List<Item> getItemsByUsuario(String usuario) {...25 lines }
65
66     public void guardar_item(String Username, Item item) {...21 lines }
87
88     public void eliminar_item(String Username, Item item) {...23 lines }
111
112     @Override
113     public void ActualizarDatos(String Usuario, Molde objeto) {...20 lines }
134 }
```

Descripción:

La clase ServicesItem maneja el guardado, actualización y obtención de los Items creados por el usuario para su implementación en sus Roomades, cabe destacar que estos son los items creados específicamente por el usuario, no los que vienen por defecto. Esta clase tiene comunicación con la tabla "item" de nuestra base de datos en MySql y hereda Services como su SuperClase abstracta.

Atributos:

- **private Final String tabla_Item = "item";**
Este atributo reconoce la tabla "item" creada en la base de datos y la implementa dentro de la clase
- **private static List<Item> items = new ArrayList<>();**
Esta ArrayList debe contener a todos los ítems que el usuario haya creado y aquí se cargan una vez que el mismo inicia sesión, debe ser estática para que funcione como una sola en todo el programa:

- **private static List<Item> itemsRoom = new ArrayList<>();**

Esta ArrayList debe contener a todos los ítems que el usuario haya guardado en alguna de sus Rooms, los identifica en la construcción de las mismas, debe ser estática para que funcione como una sola en todo el programa:

- **Mysql_BD bd = new Mysql_BD();**

Instancia de la clase Mysql_BD que maneja la conexión a la base de datos.

- **Connection con = bd.conectar();**

Variable estática que almacena la conexión actual con la base de datos.

Métodos:

- **getItems:**

Código:

```
27  public static List<Item> getItems() {  
28      return items;  
29  }
```

Descripción:

Este método se utiliza para la obtención de los ítems del usuario en otras clases del programa, es estático para identificar en todos sitios un único array de ítems.

- **setItems:**

Código:

```
31 public static void setItems(Item item) {  
32     items.add(item);  
33 }  
34
```

Descripción:

Este método se utiliza para cargar de la base de datos los ítems de cada usuario y contenerlos en el ArrayList que los maneja.

- **getItemsRoom:**

Código:

```
35 public static List<Item> getItemsRoom() {  
36     return itemsRoom;  
37 }  
38
```

Descripción:

Este método se utiliza para la obtención de los ítems del usuario guardados por él en determinadas Rooms en otras clases del programa, es estático para identificar en todos sitios un único array de ítems.

- **getItemsByUsuario:**

Código:

```
39 public List<Item> getItemsByUsuario(String usuario) {
40     List<Item> itemsDb = new ArrayList<>();
41     String query = "SELECT nombre, base, altura, profundidad FROM item WHERE usuario = ?";
42
43     try (Connection conn = con;
44         PreparedStatement pstmt = conn.prepareStatement(query)) {
45
46         pstmt.setString(1, usuario);
47         ResultSet rs = pstmt.executeQuery();
48
49         while (rs.next()) {
50             String nombre = rs.getString("nombre");
51             float base = rs.getFloat("base");
52             float altura = rs.getFloat("altura");
53             float profundidad = rs.getFloat("profundidad");
54
55             Item itemsito = new Item(nombre, base, altura, profundidad);
56             itemsDb.add(itemsito);
57         }
58     } catch (SQLException e) {
59         e.printStackTrace();
60     }
61
62     return itemsDb;
63 }
```

Descripción:

Este método se comunica con la base de datos y obtiene cada uno de los ítems asociados a un usuario a través de un bucle while para cargarlos en un array que se retorna por el mismo método y permite obtener los ítems en cuestión.

- guardarItem:

Código:

```
65 public void guardar_item(String Username, Item item ){
66     try{
67         PreparedStatement consulta;
68         double Base = item.getBase(), Altura = item.getAltura(), Profundidad = item.getProfundidad();
69         String nombreObjeto= item.getNombreObjeto();
70
71         consulta = con.prepareStatement("INSERT INTO " + this.tabla_Item + "(usuario, nombre, base, altura, profundidad)"
72                                     + " VALUES(?, ?, ?, ?, ?)");
73         consulta.setString(1, Username);
74         consulta.setString(2, nombreObjeto);
75         consulta.setDouble(3, Base);
76         consulta.setDouble(4, Altura);
77         consulta.setDouble(5, Profundidad);
78
79         consulta.executeUpdate();
80
81         System.out.println("Se guardaron los datos correctamente. ");
82     }
83     catch (SQLException ex){
84         System.out.println("Hubo un error al momento de guardar los datos. " + ex.getMessage());
85     }
86 }
```

Descripción:

Este método se comunica con la base de datos para crear un nuevo registro a partir de los datos asignados a un objeto de tipo Item e identificandolo a través del nombre de usuario del usuario que esté solicitando el guardado, una vez concertada dicha labor se imprime un mensaje por consola notificando del éxito o una excepción si aconteció algún fallo.

- **eliminarItem:**

Código:

```
88 public void eliminar_item(String Username, Item item) {
89     try {Connection con = bd.conectar(); // Crear la conexión aquí
90         PreparedStatement consulta;
91         String nombreObjeto = item.getNombreObjeto();
92
93         // Preparar la consulta para eliminar el registro que coincide con el usuario y el nombre del objeto
94         consulta = con.prepareStatement("DELETE FROM " + this.tabla_Item + " WHERE usuario = ? AND nombre = ?");
95         consulta.setString(1, Username);
96         consulta.setString(2, nombreObjeto);
97
98         int filasAfectadas = consulta.executeUpdate();
99
100         // Verificar si se eliminó algún registro
101         if (filasAfectadas > 0) {
102             System.out.println("El diseño fue eliminado correctamente.");
103         } else {
104             System.out.println("No se encontró ningún diseño para eliminar con esos parámetros.");
105         }
106     } catch (SQLException ex) {
107         System.out.println("Hubo un error al eliminar el diseño. " + ex.getMessage());
108     }
109 }
110 }
```

Descripción:

Este método se comunica con la base de datos para sustraer la información que se le mande como referencia de la tabla “tabla_Item”, tras intentar dicha ejecución puede notificar por consola si la eliminación fue exitosa, si no se encontraron archivos relacionados para efectuarla o si ocurre algún otro tipo de error.

- **actualizarDatos:**

Código:

```
112 @Override
113 public void ActualizarDatos(String Usuario, Molde objeto) {
114     try{
115         PreparedStatement consulta;
116
117         double Base = objeto.getBase(), Altura = objeto.getAltura(), Profundidad = objeto.getProfundidad();
118         String nombreObjeto= objeto.getNombreObjeto();
119
120         consulta = con.prepareStatement("UPDATE " + this.tabla_Item + " SET base = ? WHERE usuario = ?");
121         consulta.setString(2, nombreObjeto);
122         consulta.setDouble(3, Base);
123         consulta.setDouble(4, Altura);
124         consulta.setDouble(5, Profundidad);
125         consulta.executeUpdate();
126
127         System.out.println("Se guardaron los datos correctamente. ");
128     }
129     catch (SQLException ex){
130         System.out.println("Hubo un error al momento de guardar los datos. "+ ex.getMessage());
131     }
132 }
```

Descripción:

Este método se comunica con la base de datos para modificar un registro existente asociado a un usuario y de atributos reconocidos por el parámetro especificado, en caso de ser exitoso notifica esto mismo por consola y de no serlo arroja una excepción legible por consola.

Clase ServicesRoom:

Código:

```
19 public class ServicesRoom{
20     private final String tabla_Diseño = "room";
21     Mysql_BD bd = new Mysql_BD();
22     private static List<Room> rooms= new ArrayList<>();
23
24     public static List<Room> getRooms() {...3 lines }
25
26
27     public static void setRooms(Room room) {...3 lines }
28
29
30
31
32     Connection con = bd.conectar();
33
34     public void guardar_room(String username, Room room){...20 lines }
35
36
37
38
39     public void eliminar_room(String username, Room room) {...23 lines }
40
41
42
43
44     public List<Room> getRoomsByUsuario(String usuario) {...25 lines }
45
46
47
48
49 }
```

Descripción:

La clase ServicesRoom maneja el guardado, actualización y obtención de los Rooms creados por el usuario para su implementación en sus RoomMades. Esta clase tiene comunicación con la tabla “room” de nuestra base de datos en MySQL.

Atributos:

- **private final String tabla_Diseño = "room";**
Este atributo reconoce la tabla “item” creada en la base de datos y la implementa dentro de la clase
- **Mysql_BD bd = new Mysql_BD();**

Instancia de la clase Mysql_BD que maneja la conexión a la base de datos.

- **private static List<Room> rooms= new ArrayList<>();**

Esta ArrayList debe contener a todos los Rooms que el usuario haya creado y aquí se cargan una vez que el mismo inicia sesión, debe ser estática para que funcione como una sola en todo el programa:

- **Connection con = bd.conectar();**

Variable estática que almacena la conexión actual con la base de datos.

Métodos:

- **setRooms:**

Código:

```
28 public static void setRooms(Room room) {  
29     rooms.add(room);  
30 }
```

Descripción:

Este método añade Rooms obtenidas en sus parametros y las añade al ArrayList “rooms”.

- **getRooms:**

Código:

```
24 public static List<Room> getRooms() {  
25     return rooms;  
26 }
```

Descripción:

Este método es utilizado para la obtención del ArrayList “rooms” en otras clases del programa.

- **guardarRoom:**

Código:

```
34 public void guardar_room(String username, Room room){
35     try{
36         PreparedStatement consulta;
37         double Base = room.getBase(), Altura = room.getAltura(), Profundidad = room.getProfundidad();
38         String nombreObjeto= room.getNombreObjeto();
39
40         consulta = con.prepareStatement("INSERT INTO " + this.tabla_Diseño + "(usuario, nombre, base, altura, "
41             + "profundidad) VALUES(?, ?, ?, ?, ?)");
42         consulta.setString(1, username);
43         consulta.setString(2, nombreObjeto);
44         consulta.setDouble(3, Base);
45         consulta.setDouble(4, Altura);
46         consulta.setDouble(5, Profundidad);
47         consulta.executeUpdate();
48
49         System.out.println("Se guardaron los datos correctamente. ");
50     }
51     catch (SQLException ex){
52         System.out.println("Hubo un error al momento de guardar los datos. " + ex.getMessage());
53     }
54 }
```

Descripción:

Este método se comunica con la base de datos para crear un nuevo registro a partir de los datos asignados a un objeto de tipo Room e identificandolo a través del nombre de usuario del usuario que esté solicitando el guardado, una vez concertada dicha labor se imprime un mensaje por consola notificando del éxito o una excepción si aconteció algún fallo.

- **eliminarRoom:**

Código:

```
56 public void eliminar_room(String username, Room room) {
57     try (Connection con = bd.conectar(); // Crear la conexión aquí
58         PreparedStatement consulta = con.prepareStatement("DELETE FROM " + this.tabla_Diseño + " WHERE"
59             + " usuario = ? AND nombre = ?")) {
60
61         String nombreObjeto = room.getNombreObjeto();
62
63         // Asignar los valores a la consulta
64         consulta.setString(1, username);
65         consulta.setString(2, nombreObjeto);
66
67         int filasAfectadas = consulta.executeUpdate();
68
69         // Verificar si se eliminó algún registro
70         if (filasAfectadas > 0) {
71             System.out.println("El Room fue eliminado correctamente.");
72         } else {
73             System.out.println("No se encontró ningún Room para eliminar con esos parámetros.");
74         }
75     }
76     catch (SQLException ex) {
77         System.out.println("Hubo un error al eliminar el Room. " + ex.getMessage());
78     }
79 }
```

Descripción:

Este método se comunica con la base de datos para sustraer la información que se le mande como referencia de la tabla “tabla_Diseño”, tras intentar dicha ejecución puede notificar por consola si la eliminación fue exitosa, si no se encontraron archivos relacionados para efectuarla o si ocurre algún otro tipo de error.

- **getRoomsByUsuario:**

Código:

```
81 public List<Room> getRoomsByUsuario(String usuario) {
82     List<Room> roomsDb = new ArrayList<>();
83     String query = "SELECT nombre, base, altura, profundidad FROM room WHERE usuario = ?";
84
85     try (Connection conn = con;
86         PreparedStatement pstmt = conn.prepareStatement(query)) {
87
88         pstmt.setString(1, usuario);
89         ResultSet rs = pstmt.executeQuery();
90
91         while (rs.next()) {
92             String nombre = rs.getString("nombre");
93             float base = rs.getFloat("base");
94             float altura = rs.getFloat("altura");
95             float profundidad = rs.getFloat("profundidad");
96
97             Room roomsito = new Room(nombre, base, altura, profundidad);
98             roomsDb.add(roomsito);
99         }
100     } catch (SQLException e) {
101         e.printStackTrace();
102     }
103
104     return roomsDb;
105 }
```

Descripción:

Este método se comunica con la base de datos y obtiene cada una de las rooms asociadas a un usuario a través de un bucle while para cargarlos en un array que se retorna por el mismo método y permite obtener los ítems en cuestión.

Clase ServicesRoomJFX:

Código:

```
21 public class ServicesRoomJFX {
22     private final String tabla_Diseño = "rooms";
23     Mysql_BD bd = new Mysql_BD();
24     private static List<RoomJFX> roomsJFX = new ArrayList<>();
25 }
```

Descripción:

La clase ServicesRoomJFX proporciona métodos para interactuar con una base de datos MySQL que almacena información relacionada con diseños de habitaciones (RoomJFX).

Además, incluye funcionalidades para gestionar el movimiento de una cámara en una escena 3D en JavaFX, permitiendo al usuario mover la cámara a través de controles de teclado, con límites definidos según las dimensiones de una habitación seleccionada.

Atributos:

- `private final String tabla_Diseño:`

Nombre de la tabla en la base de datos MySQL donde se almacenan los diseños de habitaciones. El valor es "rooms".

- `Mysql_BD bd:`

Instancia de la clase `Mysql_BD` que maneja la conexión a la base de datos.

- `private static List<RoomJFX> roomsJFX:`

Lista estática que contiene los objetos `RoomJFX` que representan las habitaciones cargadas o creadas en la aplicación.

Métodos:

- **getRooms:**

Código:

```
26 | public static List<RoomJFX> getRooms() {  
27 |     return roomsJFX;  
28 | }
```

Descripción:

Retorna la lista de habitaciones (`RoomJFX`) que están actualmente almacenadas en el programa.

- **setRooms:**

Código:

```
30 | public static void setRooms(RoomJFX room) {  
31 |     roomsJFX.add(room);  
32 | }
```

Descripción:

Añade una habitación (RoomJFX) a la lista estática roomsJFX.

Argumentos:

RoomJFX room: Habitación que se añadirá a la lista.

Detalles:

Este método es estático y agrega un objeto RoomJFX a la lista roomsJFX.

- **guardar_diseño:**

Código:

```
35 public void guardar_diseño(String username, RoomJFX room) {
36     try {Connection con = bd.conectar(); // Crear la conexión aquí
37         PreparedStatement consulta = con.prepareStatement("INSERT INTO " + this.tabla_Diseño +
38             "(usuario, nombre, hash) VALUES(?, ?, ?)"); {
39
40         String nombreObjeto = room.getNombre();
41         String hash = room.getHash();
42
43         consulta.setString(1, username);
44         consulta.setString(2, nombreObjeto);
45         consulta.setString(3, hash);
46         consulta.executeUpdate();
47
48         System.out.println("Se guardaron los datos correctamente.");
49     } catch (SQLException ex) {
50         System.out.println("Hubo un error al momento de guardar los datos. " + ex.getMessage());
51     }
52 }
```

Descripción:

Guarda un diseño de habitación en la base de datos MySQL. Inserta el nombre de usuario, el nombre de la habitación, y el hash asociado a esa habitación en la tabla rooms.

Argumentos:

- String username: Nombre del usuario que está guardando el diseño.
- RoomJFX room: Objeto de tipo RoomJFX que contiene la información de la habitación a guardar.

Detalles:

- Utiliza una conexión a la base de datos y una consulta PreparedStatement para insertar los valores en la tabla.
- En caso de error, se captura una excepción SQLException y se imprime un mensaje de error.

- **eliminar_diseño:**

Código:

```
54 public void eliminar_diseño(String username, RoomJFX room) {
55     try (Connection con = bd.conectar()); // Crear la conexión aquí
56     PreparedStatement consulta = con.prepareStatement("DELETE FROM " + this.tabla_Diseño +
57     " WHERE usuario = ? AND nombre = ?") {
58
59         String nombreObjeto = room.getNombre();
60
61         // Asignar los valores a la consulta
62         consulta.setString(1, username);
63         consulta.setString(2, nombreObjeto);
64
65         int filasAfectadas = consulta.executeUpdate();
66
67         // Verificar si se eliminó algún registro
68         if (filasAfectadas > 0) {
69             System.out.println("El Room fue eliminado correctamente.");
70         } else {
71             System.out.println("No se encontró ningún Room para eliminar con esos parámetros.");
72         }
73
74     } catch (SQLException ex) {
75         System.out.println("Hubo un error al eliminar el Room. " + ex.getMessage());
76     }
77 }
```

Activar

Descripción:

Elimina un diseño de habitación de la base de datos MySQL basándose en el nombre del usuario y el nombre de la habitación.

Argumentos:

- String username: Nombre del usuario asociado al diseño que se desea eliminar.
- RoomJFX room: Objeto de tipo RoomJFX que contiene el nombre de la habitación a eliminar.

Detalles:

- Utiliza una conexión a la base de datos y una consulta PreparedStatement para eliminar el registro correspondiente.
- Verifica si se ha eliminado un registro y notifica el resultado.
- En caso de error, captura la excepción SQLException.

- **getRoomsJFXByUsuario:**

Código:

```

79 public List<RoomJFX> getRoomsJFXByUsuario(String usuario) {
80     List<RoomJFX> roomsDb = new ArrayList<>();
81     String query = "SELECT nombre, hash FROM rooms WHERE usuario = ?";
82
83     try (Connection conn = bd.conectar(); // Crear la conexión aquí
84         PreparedStatement pstmt = conn.prepareStatement(query)) {
85
86         pstmt.setString(1, usuario);
87         ResultSet rs = pstmt.executeQuery();
88
89         while (rs.next()) {
90             String nombre = rs.getString("nombre");
91             String hash = rs.getString("hash");
92
93             RoomJFX roomsito = new RoomJFX(nombre, hash);
94             roomsDb.add(roomsito);
95         }
96     } catch (SQLException e) {
97         e.printStackTrace();
98     }
99
100     return roomsDb;
101 }

```

Descripción:

Obtiene todos los diseños de habitación almacenados en la base de datos para un usuario específico.

Argumentos:

- String usuario: Nombre del usuario cuyos diseños se desean recuperar.

Detalles:

- Ejecuta una consulta SQL para seleccionar los diseños de habitaciones basados en el nombre del usuario.
- Devuelve una lista de objetos RoomJFX que contiene los nombres y los hashes de las habitaciones recuperadas.

- **moverCamaraHaciaAdelante:**

Código:

```

103 private void moverCamaraHaciaAdelante(PerspectiveCamera camera, double distance) {
104     // Obtener la rotación actual en el eje Y (girar izquierda/derecha) y en el eje X (girar arriba/abajo)
105     double rotationY = Math.toRadians(camera.getRotate());
106     double rotationX = Math.toRadians(camera.getRotationAxis() == Rotate.X_AXIS ? camera.getRotate() : 0);
107
108     // Calcular las componentes del movimiento en los ejes X, Y, Z
109     double deltaX = -Math.sin(rotationY) * distance * Math.cos(rotationX);
110     double deltaY = Math.sin(rotationX) * distance;
111     double deltaZ = -Math.cos(rotationY) * distance * Math.cos(rotationX);
112
113     // Actualizar la posición de la cámara
114     camera.setTranslateX(camera.getTranslateX() + deltaX);
115     camera.setTranslateY(camera.getTranslateY() - deltaY); // Restar porque el eje Y suele ir hacia abajo
116     camera.setTranslateZ(camera.getTranslateZ() + deltaZ);
117 }

```

Descripción:

Mueve la cámara hacia adelante, calculando el movimiento en los ejes X, Y y Z según la rotación actual de la cámara.

Argumentos:

- PerspectiveCamera camera: Cámara a la que se le aplicará el movimiento.
- double distance: Distancia que se moverá la cámara.

Detalles:

- Calcula las componentes del movimiento de la cámara utilizando funciones trigonométricas y actualiza las coordenadas de la cámara.
- Este método es privado.

● moverCamaraHaciaAtras:

Código:

```

119 private void moverCamaraHaciaAtras(PerspectiveCamera camera, double distance) {
120     // Obtener la rotación actual en el eje Y (girar izquierda/derecha) y en el eje X (girar arriba/abajo)
121     double rotationY = Math.toRadians(camera.getRotate());
122     double rotationX = Math.toRadians(camera.getRotationAxis() == Rotate.X_AXIS ? camera.getRotate() : 0);
123
124     // Calcular las componentes del movimiento en los ejes X, Y, Z
125     double deltaX = Math.sin(rotationY) * distance * Math.cos(rotationX);
126     double deltaY = -Math.sin(rotationX) * distance;
127     double deltaZ = Math.cos(rotationY) * distance * Math.cos(rotationX);
128
129     // Actualizar la posición de la cámara
130     camera.setTranslateX(camera.getTranslateX() + deltaX);
131     camera.setTranslateY(camera.getTranslateY() - deltaY); // Restar porque el eje Y suele ir hacia abajo
132     camera.setTranslateZ(camera.getTranslateZ() + deltaZ);
133 }

```

Descripción:

Mueve la cámara hacia atrás, calculando el movimiento en los ejes X, Y y Z según la rotación actual de la cámara.

Argumentos:

- PerspectiveCamera camera: Cámara a la que se le aplicará el movimiento.
- double distance: Distancia que se moverá la cámara.

Detalles:

- Calcula las componentes del movimiento de la cámara utilizando funciones trigonométricas y actualiza las coordenadas de la cámara.
- Este método es privado.

● MoverCamara:

Código:

```
138 |  
139 |  
140 |  
141 |  
142 |  
143 |  
144 |  
145 |  
146 |  
147 |  
148 |  
149 |  
150 |  
151 |  
152 |  
153 |  
  
public void MoverCamara(SubScene scene, PerspectiveCamera cameraInterna) {  
    Platform.runLater(() -> scene.requestFocus());  
    scene.setOnKeyPressed(event -> {  
        double movementSpeed = 10.0;  
        if (scene.getCamera() == cameraInterna) {  
            System.out.println("Me están leyendo");  
            switch (event.getCode()) {  
                case W -> moverCamaraHaciaAdelante(cameraInterna, movementSpeed);  
                case S -> moverCamaraHaciaAtras(cameraInterna, movementSpeed);  
                case A -> cameraInterna.setTranslateX(cameraInterna.getTranslateX() - movementSpeed);  
                case D -> cameraInterna.setTranslateX(cameraInterna.getTranslateX() + movementSpeed);  
            }  
            limitarMovimientoCamara(cameraInterna); // Llamada para asegurarse de que la cámara permanece dentro de los límites  
        }  
    });  
}
```

Descripción:

Asigna eventos de teclado para mover la cámara en las direcciones hacia adelante, atrás, izquierda, y derecha en una escena 3D.

Argumentos:

- SubScene scene: Subescena que contendrá los eventos de movimiento.
- PerspectiveCamera cameraInterna: Cámara interna que será movida.

Detalles:

- Mueve la cámara según la tecla presionada (W, A, S, D) y limita su movimiento

utilizando el método limitarMovimientoCamara.

- Utiliza Platform.runLater para asegurarse de que la escena recibe el foco.

- **limitarMovimientoCamara:**

Código:

```
155 private void limitarMovimientoCamara(PerspectiveCamera camera) {
156     double minX = (-roomSeleccionada.getBase())/2; // Limites de la habitación
157     double maxX = (roomSeleccionada.getBase())/2;
158     double minY = (-roomSeleccionada.getAltura())/2;
159     double maxY = (roomSeleccionada.getAltura())/2;
160     double minZ = (-roomSeleccionada.getProfundidad())/2;
161     double maxZ = (roomSeleccionada.getProfundidad())/2;
162
163     if (camera.getTranslateX() < minX) camera.setTranslateX(minX);
164     if (camera.getTranslateX() > maxX) camera.setTranslateX(maxX);
165     if (camera.getTranslateY() < minY) camera.setTranslateY(minY);
166     if (camera.getTranslateY() > maxY) camera.setTranslateY(maxY);
167     if (camera.getTranslateZ() < minZ) camera.setTranslateZ(minZ);
168     if (camera.getTranslateZ() > maxZ) camera.setTranslateZ(maxZ);
169 }
170
171 }
```

Descripción:

Limita el movimiento de la cámara dentro de los límites de la habitación seleccionada.

Argumentos:

- PerspectiveCamera camera: Cámara que será limitada en su movimiento.

Detalles:

- Verifica las coordenadas de la cámara en los ejes X, Y y Z y las ajusta si se salen de los límites definidos por las dimensiones de la habitación (roomSeleccionada).
- Este método es privado.

Clase ServicesUsuario:

Código:

```
19 public class ServicesUsuario extends Services {
20     private final String tabla_Usuarios = "uwu";
21     Mysql_BD bd = new Mysql_BD();
22     private static String usuario;
23
24     Connection con = bd.conectar();
25
26     public String setId(){...4 lines }
30
31     public static void setUsuario(String usuario) {...3 lines }
34
35     public static String getUsuario() {...3 lines }
38
39     public void guardar_usuario(Usuario usuario){...18 lines }
57
58     public boolean Info_UsuarioLogin(String Usuario, String Contraseña) {...15 lines }
73
74     public void actualizar_usuario(String newusuario, String usuario){...15 lines }
89
90     public void actualizar_email(String newemail, String email){...15 lines }
105
106     public void actualizar_password(String newpassword, String password){...15 lines }
121
122     public void actualizar_nombres(String newnames, String names){...15 lines }
137
138     public String getEmail(String Usuario) {...15 lines }
153
154     public String getNombres(String Usuario) {...15 lines }
169
170     @Override
171     public void ActualizarDatos(String Usuario, Molde molde) {
172         throw new UnsupportedOperationException("Not supported yet.");
173     }
174 }
```

Descripción:

La clase ServicesUsuario comprende las necesidades en el tratamiento de los objetos de tipo usuario, desde su creación hasta interacción con las diferentes clases del programa.

Atributos:

- **private final String tabla_Usuarios = "uwu";**

Este atributo reconoce la tabla “uwu” creada en la base de datos y la implementa dentro de la clase.

- **Mysql_BD bd = new Mysql_BD();**

Instancia de la clase Mysql_BD que maneja la conexión a la base de datos.

- **private static String usuario;**

Variable estática que guarda el nombre del usuario cuando se loguea.

- **Connection con = bd.conectar();**

Variable estática que almacena la conexión actual con la base de datos.

Métodos:

- **setID:**

Código:

```
25 | public String setID(){  
26 |     UUID uid= UUID.randomUUID();  
27 |     return uid.toString();  
28 | }
```

Descripción:

Este método genera un ID aleatorio y lo devuelve retornandolo en el mismo método.

- **setUsuario:**

Código:

```
30 | public static void setUsuario(String usuario) {  
31 |     ServicesUsuario.usuario = usuario;  
32 | }
```

Descripción:

Este método asigna a la variable estática usuario de esta misma clase el nombre del usuario que se trae por parametro.

- **getUsuario:**

Código:

```
34 | public static String getUsuario() {  
35 |     return usuario;  
36 | }
```

Descripción:

Retorna el valor de la variable usuario de la clase en la que estamos.

- **guardarUsuario:**

Código:

```
38 public void guardar_usuario(Usuario usuario){
39     try{
40         PreparedStatement consulta;
41
42         consulta = con.prepareStatement("INSERT INTO " + this.tabla_Usuarios + "(usuario, contraseña,"
43             + " correo, nombre) VALUES(?, ?, ?, ?)");
44         consulta.setString(1, usuario.getUsername());
45         consulta.setString(2, usuario.getPassword());
46         consulta.setString(3, usuario.getEmail());
47         consulta.setString(4, usuario.getName());
48
49         consulta.executeUpdate();
50
51         System.out.println("Se guardaron los datos correctamente. ");
52     }
53     catch (SQLException ex){
54         System.out.println("Hubo un error al momento de guardar los datos. ");
55     }
56 }
```

Descripción:

Este método se comunica con la base de datos para crear un nuevo registro a partir de los datos asignados a un objeto de tipo Usuario, una vez concertada dicha labor se imprime un mensaje por consola notificando del éxito o una excepción si aconteció algún fallo.

- **Info_UsuarioLogin:**

Código:

```
58 public boolean Info_UsuarioLogin(String Usuario, String Contraseña) {
59     try {
60         PreparedStatement consulta = con.prepareStatement("SELECT contraseña FROM " + this.tabla_Usuarios +
61             " WHERE usuario = ?");
62         consulta.setString(1, Usuario);
63         ResultSet resultado = consulta.executeQuery();
64         if (resultado.next()) {
65             String contraseñaObtenida = resultado.getString("contraseña");
66             return Contraseña.equals(contraseñaObtenida);
67         }
68     } catch (SQLException ex) {
69         // Manejo de excepciones opcionalmente
70         System.out.println("Error en la verificación del login: " + ex.getMessage());
71     }
72     return false;
73 }
```

Descripción:

Este método recibe un nombre de usuario y una contraseña, revisa en la base de datos si el usuario que se está consultando tiene asociada la contraseña que se está mandando y de tenerla retornar un dato "true" que se utilizará para corroborar el inicio de sesión, en caso de no tenerla se mostrará un mensaje de error por consola y se retornara un "false".

- **actualizarUsuario:**

Código:

```
75 public void actualizar_usuario(String newusuario, String usuario){
76     try{
77         PreparedStatement consulta;
78
79         consulta = con.prepareStatement("UPDATE " + this.tabla_Usuarios + " SET usuario = ? WHERE usuario = ?");
80         consulta.setString(1, newusuario);
81         consulta.setString(2, usuario);
82         consulta.executeUpdate();
83
84         System.out.println("Se guardaron los datos correctamente. ");
85     }
86     catch (SQLException ex){
87         System.out.println("Hubo un error al momento de guardar los datos. " + ex.getMessage());
88     }
89 }
```

Descripción:

Este método se comunica con la base de datos para modificar un registro existente de tipo usuario, funciona identificando el objeto existente por su identidad en el inicio de sesión, y cambia la información de interés (Nombre de usuario) a la de una String que contiene el nuevo nombre deseado. En caso de ser exitoso notifica esto mismo por consola y de no serlo arroja una excepción legible por consola.

- **ActualizarEmail:**

Código:

```
91 public void actualizar_email(String newemail, String email){
92     try{
93         PreparedStatement consulta;
94
95         consulta = con.prepareStatement("UPDATE " + this.tabla_Usuarios + " SET correo = ? WHERE correo = ?");
96         consulta.setString(1, newemail);
97         consulta.setString(2, email);
98         consulta.executeUpdate();
99
100         System.out.println("Se guardaron los datos correctamente. ");
101     }
102     catch (SQLException ex){
103         System.out.println("Hubo un error al momento de guardar los datos. " + ex.getMessage());
104     }
105 }
```

Descripción:

Este método se comunica con la base de datos para modificar un registro existente de tipo usuario, funciona identificando el objeto existente por su identidad en el inicio de sesión, y cambia la información de interés (Email) a la de una String que contiene el nuevo email deseado. En caso de ser exitoso notifica esto mismo por consola y de no serlo arroja una excepción legible por consola.

- **ActualizarPassword:**

Código:

```
107 public void actualizar_password(String newpassword, String password){
108     try{
109         PreparedStatement consulta;
110
111         consulta = con.prepareStatement("UPDATE " + this.tabla_Usuarios + " SET contraseña = ? WHERE contraseña = "
112         consulta.setString(1, newpassword);
113         consulta.setString(2, password);
114         consulta.executeUpdate();
115
116         System.out.println("Se guardaron los datos correctamente. ");
117     }
118     catch (SQLException ex){
119         System.out.println("Hubo un error al momento de guardar los datos. "+ ex.getMessage());
120     }
121 }
```

Descripción:

Este método se comunica con la base de datos para modificar un registro existente de tipo usuario, funciona identificando el objeto existente por su identidad en el inicio de sesión, y cambia la información de interés (Contraseña) a la de una String que contiene la nueva contraseña deseada. En caso de ser exitoso notifica esto mismo por consola y de no serlo arroja una excepción legible por consola.

- **ActualizarNombres:**

Código:

```
123 public void actualizar_nombres(String newnames, String names){
124     try{
125         PreparedStatement consulta;
126
127         consulta = con.prepareStatement("UPDATE " + this.tabla_Usuarios + " SET nombre = ? WHERE nombre = ?");
128         consulta.setString(1, newnames);
129         consulta.setString(2, names);
130         consulta.executeUpdate();
131
132         System.out.println("Se guardaron los datos correctamente. ");
133     }
134     catch (SQLException ex){
135         System.out.println("Hubo un error al momento de guardar los datos. "+ ex.getMessage());
136     }
137 }
```

Descripción:

Este método se comunica con la base de datos para modificar un registro existente de tipo usuario, funciona identificando el objeto existente por su identidad en el inicio de sesión, y cambia la información de interés (Nombres) a la de una String que contiene los nuevos nombres deseados. En caso de ser exitoso notifica esto mismo por consola y de no serlo arroja una excepción legible por consola.

- **getEmail:**

Código:

```
139 public String getEmail(String Usuario) {
140     try {
141         PreparedStatement consulta = con.prepareStatement("SELECT correo FROM " + this.tabla_Usuarios +
142             " WHERE usuario = ?");
143         consulta.setString(1, Usuario);
144         ResultSet resultado = consulta.executeQuery();
145         if (resultado.next()) {
146             String emailObtenido = resultado.getString("correo");
147             return emailObtenido;
148         }
149     } catch (SQLException ex) {
150         // Manejo de excepciones opcionalmente
151         System.out.println("Error en la verificación del email: " + ex.getMessage());
152     }
153     return null;
154 }
```

Descripción:

Este método retorna el Email del usuario identificandolo por el nombre de usuario del mismo. Con este parámetro se busca el dato asociado en la base de datos y se retorna en caso se encontrarse, de no ser así sale un mensaje de error por pantalla y se retorna "null".

- **getNombres:**

Código:

```
156 public String getNombres(String Usuario) {
157     try {
158         PreparedStatement consulta = con.prepareStatement("SELECT nombre FROM " + this.tabla_Usuarios +
159             " WHERE usuario = ?");
160         consulta.setString(1, Usuario);
161         ResultSet resultado = consulta.executeQuery();
162         if (resultado.next()) {
163             String nombresObtenido = resultado.getString("nombre");
164             return nombresObtenido;
165         }
166     } catch (SQLException ex) {
167         // Manejo de excepciones opcionalmente
168         System.out.println("Error en la verificación del nombre: " + ex.getMessage());
169     }
170     return null;
171 }
```

Descripción:

Este método retorna lo Nombres del usuario identificandolo por el nombre de usuario del mismo. Con este parámetro se busca el dato asociado en la base de datos y se retorna en caso se encontrarse, de no ser así sale un mensaje de error por pantalla y se retorna "null".

- **ActualizarDatos:**

Código:

```
173 @Override
174 public void ActualizarDatos(String Usuario, Molde molde) {
175     throw new UnsupportedOperationException("Not supported yet.");
176 }
```

Descripción:

Método abstracto a la espera de implementar su funcionalidad en futuras actualizaciones.

Clases dentro de *view*

Clase CrearItem:

Código

```
43 public class CrearItem extends Application {
44
45     private PerspectiveCamera camera;
46     private double mouseX, mouseY;
47     private double rotateX = 0, rotateY = 0;
48     private double rotateSpeed = 0.2;
49     private double zoomSpeed = 1.1;
50     private double floorY = 1000;
51
52     private List<Shape3D> shapes = new ArrayList<>();
53     private boolean distancesFixed = false;
54     private Shape3D selectedShape = null;
55     private List<double[]> distances = new ArrayList<>();
56     private Shape3D unanchoredShape = null;
57
58     private Group root3D;
59     private Stage primaryStage;
60     private SubScene subScene;
61     private Shape3D selectedObject;
62     private double objectMouseOffsetX, objectMouseOffsetY;
63
64     private RooMakingJFX3D parentWindow;
65
66     private Button fijarposicionesBtn;
67 }
```

Descripción:

La clase CrearItem es una aplicación de JavaFX que permite a los usuarios diseñar items en 3D para sus habitaciones utilizando diferentes formas geométricas como esferas, cilindros, cubos, y otros poliedros. Proporciona funcionalidades para agregar y manipular objetos 3D dentro de una escena, mover la cámara, y fijar distancias entre los objetos. También permite al usuario ajustar los parámetros de los objetos para que configure el diseño general de la escena.

Atributos:

- private PerspectiveCamera camera: Cámara utilizada para ver y navegar la escena 3D.
- private double mouseX, mouseY: Coordenadas del ratón utilizadas para el control de rotación de la cámara.
- private double rotateX = 0, rotateY = 0: Variables que almacenan el ángulo de rotación de la cámara en los ejes X e Y.
- private double rotateSpeed = 0.2: Velocidad de rotación de la cámara cuando el ratón es arrastrado.
- private double zoomSpeed = 1.1: Factor de velocidad de zoom para la cámara.
- private double floorY = 1000: Altura del "piso" en la escena 3D, utilizada como referencia.
- private List<Shape3D> shapes = new ArrayList<>(): Lista de objetos 3D que han sido añadidos a la escena.
- private boolean distancesFixed = false: Bandera que indica si las distancias entre los objetos 3D están fijas.
- private Shape3D selectedShape = null: Objeto 3D seleccionado actualmente por el usuario.
- private List<double[]> distances = new ArrayList<>(): Lista de distancias entre los objetos cuando las posiciones están fijadas.
- private Shape3D unanchoredShape = null: Objeto 3D que ha sido desanclado para moverse libremente en la escena.
- private Group root3D: Nodo raíz que contiene todos los objetos 3D dentro de la subescena.
- private Stage primaryStage: Ventana principal de la aplicación.
- private SubScene subScene: Subescena donde se renderiza la vista 3D.
- private Shape3D selectedObject: Objeto 3D seleccionado en la escena que puede ser movido.
- private double objectMouseOffsetX, objectMouseOffsetY: Desplazamiento del ratón respecto a la posición del objeto seleccionado.
- private RooMakingJFX3D parentWindow: Ventana padre que contiene la escena 3D.
- private Button fijarposicionesBtn: Botón para fijar las posiciones de los objetos en la escena.

Métodos:

- **start:**

Código:

```
68  @Override
69  public void start(Stage primaryStage) {
70      this.primaryStage = primaryStage;
71      root3D = new Group();
72      subScene = new SubScene(root3D, 600, 400, true, javafx.scene.SceneAntialiasing.BALANCED);
73      subScene.setFill(Color.BEIGE);
74
75      camera = new PerspectiveCamera(true);
76      camera.setTranslateZ(-2000);
77      camera.setTranslateY(500);
78      camera.setNearClip(1);
79      camera.setFarClip(10000);
80      camera.setFieldOfView(50);
81      subScene.setCamera(camera);
82
83
84      subScene.setOnMousePressed(this::handleMousePressed);
85      subScene.setOnMouseDragged(this::handleMouseDragged);
86      subScene.setOnScroll(this::handleScroll);
87
88      VBox controlPanel = createControlPanel();
89
90      HBox mainLayout = new HBox(10);
91      mainLayout.setPadding(new Insets(10));
92      mainLayout.getChildren().addAll(subScene, controlPanel);
93
94      mainLayout.setStyle(
95          "-fx-background-color: linear-gradient(to bottom right, rgb(245,245,220), rgb(162,217,206));" + // Fondo degradado de beige a verde claro
96          "-fx-border-color: rgb(210,180,140);" + // Color del borde tierra
97          "-fx-border-width: 2px;" + // Grosor del borde
98          "-fx-border-radius: 10px;" + // Bordes redondeados
99          "-fx-background-radius: 10px;" + // Bordes redondeados para el fondo
100         "-fx-effect: dropshadow(gaussian, rgba(0, 0, 0, 0.2), 10, 0.5, 0, 0);" // Efecto de sombra
101     );
102
103     Scene mainScene = new Scene(mainLayout);
104
105     primaryStage.setTitle("RoomDesigner3D");
106     primaryStage.setScene(mainScene);
107     primaryStage.setWidth(800);
108     primaryStage.setHeight(460);
109     primaryStage.show();
110
111     root3D.requestFocus();
112 }
```

Descripción:

Inicializa la aplicación, configura la cámara, subescena y controles de usuario para interactuar con los objetos 3D.

Argumentos:

- Stage primaryStage: Ventana principal de la aplicación.

- **Detalles:**

- Establece la cámara, subescena, y el panel de control. Muestra la ventana principal.

- **CrearItem:**

Código:

```
114 public CrearItem(RooMakingJFX3D parentWindow) {
115     this.parentWindow = parentWindow;
116 }
```

Descripción:

Constructor de la clase que recibe una ventana padre.

Argumentos:

- RooMakingJFX3D parentWindow: Ventana padre que contiene la escena principal.

- **createControlPanel:**

Código:

```
118 private VBox createControlPanel() {
119     VBox controlPanel = new VBox(10);
120     controlPanel.setPadding(new Insets(10));
121
122     Button fixDistancesBtn = new Button("Fijar Distancias");
123     fixDistancesBtn.setStyle(
124         "-fx-background-color: linear-gradient(to bottom, rgb(126,188,137), rgb(162,217,206));" + // Degradado de verde
125         "-fx-text-fill: white;" + // Texto en blanco
126         "-fx-font-weight: bold;" + // Texto en negrita
127         "-fx-border-color: rgb(126,188,137);" + // Borde del botón
128         "-fx-border-radius: 5;" + // Bordes redondeados
129         "-fx-background-radius: 5;" // Bordes redondeados para el fondo
130     );
131     fixDistancesBtn.setOnAction(event -> fixDistances());
132
133     Label titleLabel = new Label("Seleccionar objeto:");
134     ComboBox<String> objectSelector = new ComboBox<>();
135     objectSelector.getItems().addAll("Esfera", "Cilindro", "Cubo", "Poliedro", "Tetraedro", "Piramide");
136     objectSelector.setStyle(
137         "-fx-background-color: linear-gradient(to bottom, rgb(162,217,206), rgb(126,188,137));" + // Degradado de verde claro a verd
138         "-fx-border-color: rgb(210,180,140);" + // Borde color tierra
139         "-fx-border-radius: 5;" // Bordes redondeados
140     );
141     objectSelector.setOnAction(e -> addObject(objectSelector.getValue()));
142
143     Button roomMakingBtn = new Button("RooMaking");
```

```

144     roomMakingBtn.setStyle(
145         "-fx-background-color: linear-gradient(to bottom, rgb(126,188,137), rgb(162,217,206));" + // Degradado de verde
146         "-fx-text-fill: white;" + // Texto en blanco
147         "-fx-font-weight: bold;" + // Texto en negrita
148         "-fx-border-color: rgb(126,188,137);" + // Borde del botón
149         "-fx-border-radius: 5;" + // Bordes redondeados
150         "-fx-background-radius: 5;" // Bordes redondeados para el fondo
151     );
152
153     roomMakingBtn.setOnAction(e -> handleRoomMaking());
154
155     controlPanel.setAlignment(Pos.CENTER);
156     controlPanel.getChildren().addAll(titleLabel, objectSelector, fixDistancesBtn, roomMakingBtn);
157     controlPanel.setStyle(
158         "-fx-background-color: rgb(245,245,220);" + // Fondo beige claro
159         "-fx-border-color: rgb(210,180,140);" + // Borde color tierra
160         "-fx-border-width: 2px;" + // Grosor del borde
161         "-fx-border-radius: 10px;" // Bordes redondeados
162     );
163     return controlPanel;
164 }

```

Descripción:

Crea el panel de control de la interfaz, permitiendo al usuario seleccionar objetos y fijar distancias.

Detalles:

- El panel incluye un ComboBox para seleccionar el tipo de objeto y botones para fijar posiciones y crear una nueva habitación.

● handleRoomMaking:

Código:

```

166 private void handleRoomMaking() {
167     // Crear un nuevo Group con todos los objetos de la escena actual
168     Group itemGroup = new Group(root3D.getChildren());
169
170     // Enviar el Group a la ventana padre
171     parentWindow.addItemToScene(itemGroup);
172
173     // Cerrar la ventana actual
174     primaryStage.close();
175 }

```

Descripción:

Maneja la creación de un nuevo grupo de objetos 3D, en este caso el ítem creado por el usuario y lo envía a la ventana padre.

- addObject:

Código:

```
177 private void addObject(String objectType) {
178     Stage dialog = new Stage();
179     dialog.setTitle("Ingresar dimensiones para " + objectType);
180
181     final Shape3D[] newObject = new Shape3D[1]; // Utiliza un array de un solo elemento
182     VBox dialogVBox = new VBox(10); // para que sea efectivamente final
183     dialogVBox.setPadding(new Insets(10));
184
185     Label widthLabel = new Label("Ancho:");
186     TextField widthField = new TextField();
187     Label heightLabel = new Label("Altura:");
188     TextField heightField = new TextField();
189     Label depthLabel = new Label("Profundidad:");
190     TextField depthField = new TextField();
191     Label colorLabel = new Label("Color (en formato HEX, ej: #FF5733):");
192     TextField colorField = new TextField();
193
194     // Configurar la visibilidad de los campos según el tipo de objeto
195     switch (objectType) {
196         case "Cilindro":
197             depthLabel.setDisable(true);
198             depthField.setDisable(true);
199             break;
200         case "Esfera":
201             depthLabel.setDisable(true);
202             depthField.setDisable(true);
203             heightLabel.setDisable(true);
204             heightField.setDisable(true);
205             break;
206         case "Cubo":
207         case "Poliedro":
208         case "Tetraedro":
209         case "Piramide":
210             // Todos los campos son necesarios
211             break;
212     }
```

```

213 Button createButton = new Button("Crear");
214 createButton.setOnAction(e -> {
215     try {
216         float width = Float.parseFloat(widthField.getText());
217         float height = objectType.equals("Esfera") ? width :
218             Float.parseFloat(heightField.getText());
219         float depth = (objectType.equals("Esfera") || objectType.equals("Cilindro")) ? height :
220             Float.parseFloat(depthField.getText());
221         Color color = Color.web(colorField.getText());
222
223         // Crear el objeto según el tipo y las propiedades
224         switch (objectType) {
225             case "Cubo":
226                 newObject[0] = new Box(width, height, depth);
227                 newObject[0].setMaterial(new PhongMaterial(color));
228                 break;
229             case "Esfera":
230                 newObject[0] = new Esfera3D(width, color).getEsfera();
231                 break;
232             case "Cilindro":
233                 newObject[0] = new Cilindro3D(width, height, color).getCilindro();
234                 break;
235             case "Poliedro":
236                 newObject[0] = new PoliedroPersonalizado(width, height, depth, color);
237                 break;
238             case "Tetraedro":
239                 newObject[0] = new Tetraedro(width, height, depth, color);
240                 break;
241             case "Piramide":
242                 newObject[0] = new Piramide(width, height, depth, color);
243                 break;
244             default:
245                 break;
246         }
247
248         // Agregar el objeto a la escena
249         if (newObject[0] != null) {
250             addObjectToScene(newObject[0]);
251         }
252     } catch (IllegalArgumentException ex) {
253         Alert alert = new Alert(Alert.AlertType.ERROR);
254         alert.setTitle("Error");
255         alert.setHeaderText("Entrada inválida");
256         alert.setContentText("Por favor, ingrese dimensiones y un color válidos.");
257         alert.showAndWait();
258     }
259 }
260
261 dialogVBox.getChildren().addAll(widthLabel, widthField, heightLabel, heightField,
262     depthLabel, depthField, colorLabel, colorField, createButton);
263 Scene dialogScene = new Scene(dialogVBox, 300, 300);
264 dialog.setScene(dialogScene);
265 dialog.show();
266 }

```

Descripción:

Añade un nuevo objeto 3D a la escena basado en el tipo seleccionado.

Argumentos:

- String objectType: Tipo de objeto a crear (esfera, cilindro, cubo, etc.).

- **Detalles:**

- Solicita las dimensiones y el color del objeto antes de añadirlo a la escena.

- **addObjectToScene:**

Código:

```
268 private void addObjectToScene(Shape3D newObject) {
269     newObject.setTranslateX(0);
270     newObject.setTranslateY(0);
271     newObject.setTranslateZ(0);
272     newObject.setOnMousePressed(this::handleObjectPressed);
273     newObject.setOnMouseDragged(this::handleObjectDragged);
274     newObject.setOnMouseReleased(event -> selectedObject = null);
275     newObject.setOnMouseClicked(this::handleObjectClicked);
276
277     setupContextMenu(newObject);
278     root3D.getChildren().add(newObject);
279     shapes.add(newObject);
280 }
```

Descripción:

Añade un objeto 3D a la subescena y configura los eventos de interacción.

Argumentos:

- Shape3D newObject: Objeto 3D que será añadido a la escena.

- **setupContextMenu**

Código:

```
282 private void setupContextMenu(Shape3D object) {
283     ContextMenu contextMenu = new ContextMenu();
284     MenuItem removeItem = new MenuItem("Eliminar");
285     removeItem.setOnAction(e -> {
286         root3D.getChildren().remove(object);
287         shapes.remove(object);
288     });
289     contextMenu.getItems().add(removeItem);
290
291     object.setOnContextMenuRequested(e -> contextMenu.show(object, e.getScreenX(), e.getScreenY()));
292 }
```

Descripción:

Configura un menú contextual para eliminar objetos de la escena.

Argumentos:

- Shape3D object: Objeto al que se le agregará el menú contextual.

- **handleObjectPressed:**

Código:

```
294 | private void handleObjectPressed(MouseEvent event) {  
295 |     if (event.getButton() == MouseButton.PRIMARY) {  
296 |         selectedObject = (Shape3D) event.getSource();  
297 |         System.out.println("Se presiono una figura");  
298 |         objectMouseOffsetX = selectedObject.getTranslateX() - event.getSceneX();  
299 |         objectMouseOffsetY = selectedObject.getTranslateY() - event.getSceneY();  
300 |     }  
301 |     event.consume();  
302 | }
```

Descripción:

Maneja el evento de presionar un objeto 3D, lo selecciona y ajusta el desplazamiento del ratón.

Argumentos:

- MouseEvent event: Evento del ratón que dispara la selección.

- **handleObjectDragged:**

Código:

```
304 | private void handleObjectDragged(MouseEvent event) {  
305 |     if (selectedObject != null) {  
306 |         selectedObject.setTranslateX(event.getSceneX() + objectMouseOffsetX);  
307 |         selectedObject.setTranslateY(event.getSceneY() + objectMouseOffsetY);  
308 |  
309 |         if (distancesFixed && selectedObject != unanchoredShape) {  
310 |             adjustPositions(selectedObject);  
311 |         }  
312 |     }  
313 |     event.consume();  
314 | }
```

Descripción:

Maneja el arrastre de un objeto seleccionado en la escena.

Argumentos:

- MouseEvent event: Evento del ratón que dispara el arrastre.

- **handleObjectClicked:**

Código:

```
316 | private void handleObjectClicked(MouseEvent event) {
317 |     Shape3D clickedShape = (Shape3D) event.getSource();
318 |
319 |     if (distancesFixed) {
320 |         unanchoredShape = clickedShape;
321 |     } else {
322 |         unanchoredShape = null;
323 |     }
324 |
325 |     selectedShape = clickedShape;
326 | }
```

Descripción:

Maneja el evento de clic en un objeto para seleccionarlo o desanclarlo.

Argumentos:

- MouseEvent event: Evento de clic que selecciona el objeto.

- **fixDistances:**

Código:

```
328 | private void fixDistances() {
329 |     distancesFixed = true;
330 |     distances.clear();
331 |
332 |     for (int i = 0; i < shapes.size(); i++) {
333 |         for (int j = i + 1; j < shapes.size(); j++) {
334 |             Shape3D shapeA = shapes.get(i);
335 |             Shape3D shapeB = shapes.get(j);
336 |
337 |             double distanceX = shapeB.getTranslateX() - shapeA.getTranslateX();
338 |             double distanceY = shapeB.getTranslateY() - shapeA.getTranslateY();
339 |             double distanceZ = shapeB.getTranslateZ() - shapeA.getTranslateZ();
340 |
341 |             distances.add(new double[]{i, j, distanceX, distanceY, distanceZ});
342 |         }
343 |     }
344 | }
```

Descripción:

Fija las distancias actuales entre los objetos 3D en la escena.

Detalles:

- Calcula y almacena las distancias entre los objetos 3D.

- **adjustPositions:**

Código:

```
346 private void adjustPositions(Shape3D movedShape) {
347     int movedIndex = shapes.indexOf(movedShape);
348
349     for (double[] distance : distances) {
350         int indexA = (int) distance[0];
351         int indexB = (int) distance[1];
352         double distanceX = distance[2];
353         double distanceY = distance[3];
354         double distanceZ = distance[4];
355
356         if (indexA == movedIndex || indexB == movedIndex) {
357             Shape3D shapeA = shapes.get(indexA);
358             Shape3D shapeB = shapes.get(indexB);
359
360             if (indexA == movedIndex) {
361                 updateShapePosition(shapeB, shapeA, distanceX, distanceY, distanceZ);
362             } else {
363                 updateShapePosition(shapeA, shapeB, -distanceX, -distanceY, -distanceZ);
364             }
365         }
366     }
367 }
```

Descripción:

Ajusta las posiciones de los objetos 3D para mantener las distancias fijas cuando uno de ellos se mueve.

Argumentos:

- Shape3D movedShape: Objeto que se ha movido en la escena.

- **updateShapePosition:**

Código:

```
369 private void updateShapePosition(Shape3D shapeToMove, Shape3D referenceShape,
370     double distanceX, double distanceY, double distanceZ) {
371     shapeToMove.setTranslateX(referenceShape.getTranslateX() + distanceX);
372     shapeToMove.setTranslateY(referenceShape.getTranslateY() + distanceY);
373     shapeToMove.setTranslateZ(referenceShape.getTranslateZ() + distanceZ);
374 }
```

Descripción:

Actualiza la posición de un objeto basado en la posición de referencia de otro.

Argumentos:

- Shape3D shapeToMove: Objeto cuya posición será ajustada.
- Shape3D referenceShape: Objeto de referencia.
- double distanceX, distanceY, distanceZ: Distancias a mantener entre los objetos.

- **handleMousePressed:**

Código:

```
376 private void handleMousePressed(MouseEvent event) {
377     mouseX = event.getSceneX();
378     mouseY = event.getSceneY();
379 }
```

Descripción:

Captura la posición del ratón cuando se presiona.

Argumentos:

- MouseEvent event: Evento de presionar el ratón.

- **handleMouseDragged:**

Código:

```
381 private void handleMouseDragged(MouseEvent event) {  
382     rotateX += (event.getSceneY() - mouseY) * rotateSpeed;  
383     rotateY += (event.getSceneX() - mouseX) * rotateSpeed;  
384  
385     camera.setRotationAxis(new Point3D(1, 0, 0));  
386     camera.setRotate(rotateX);  
387  
388     camera.setRotationAxis(new Point3D(0, 1, 0));  
389     camera.setRotate(rotateY);  
390  
391     mouseX = event.getSceneX();  
392     mouseY = event.getSceneY();  
393 }
```

Descripción:

Mueve la cámara al arrastrar el ratón.

Argumentos:

- MouseEvent event: Evento de arrastrar el ratón.

- **handleScroll:**

Código:

```
395 private void handleScroll(ScrollEvent event) {  
396     double zoomFactor = (event.getDeltaY() > 0) ? zoomSpeed : 1 / zoomSpeed;  
397     camera.setTranslateZ(camera.getTranslateZ() * zoomFactor);  
398 }
```

Descripción:

Ajusta el zoom de la cámara según el desplazamiento de la rueda del ratón.

Argumentos:

- ScrollEvent event: Evento de desplazamiento del ratón.

- **createShapeFromKey:**

Código:

```
102 private Shape3D createShapeFromKey(String key, float width, float height, float depth, Color color) {
103     Shape3D newObject = null;
104
105     if (key.contains("Box")) {
106         newObject = new Box(width, height, depth);
107         newObject.setMaterial(new PhongMaterial(color));
108     } else if (key.contains("Sphere")) {
109         newObject = new Esfera3D(width / 2, color).getEsfera(); // Aquí se asume que width es el diámetro
110     } else if (key.contains("Cylinder")) {
111         newObject = new Cilindro3D(width / 2, height, color).getCilindro(); // Aquí se asume que width es el diámetro
112     } else if (key.contains("Tetrahedron")) {
113         newObject = new Tetraedro(width, height, depth, color);
114     } else if (key.contains("Pyramid")) {
115         newObject = new Piramide(width, height, depth, color);
116     } else {
117         newObject = new Box(width, height, depth); // Default shape
118         newObject.setMaterial(new PhongMaterial(color));
119     }
120
121     return newObject;
122 }
```

Descripción:

Crea un objeto 3D basado en un tipo y dimensiones proporcionadas.

Argumentos:

- String key: Tipo de objeto.
- float width, height, depth: Dimensiones del objeto.
- Color color: Color del objeto.

● createTetrahedronMesh:

Código:

```
423 private TriangleMesh createTetrahedronMesh(float width, float height, float depth) {
424     TriangleMesh mesh = new TriangleMesh();
425
426     mesh.getPoints().addAll(
427         0, 0, 0, // Vertex 0
428         0, height, depth, // Vertex 1
429         width, height, depth, // Vertex 2
430         width / 2, 0, depth / 2 // Vertex 3
431     );
432
433     mesh.getFaces().addAll(
434         0, 0, 1, 0, 3, 0,
435         0, 0, 2, 0, 1, 0,
436         1, 0, 2, 0, 3, 0,
437         2, 0, 0, 0, 3, 0
438     );
439
440     return mesh;
441 }
```

Descripción:

Crea una malla para un tetraedro con las dimensiones dadas.

Argumentos:

- float width, height, depth: Dimensiones del tetraedro.

- **createPyramidMesh:**

Código:

```
443 private TriangleMesh createPyramidMesh(float width, float height, float depth) {  
444     TriangleMesh mesh = new TriangleMesh();  
445  
446     mesh.getPoints().addAll(  
447         0, 0, 0, // Base vertex 0  
448         0, depth, 0, // Base vertex 1  
449         width, 0, 0, // Base vertex 2  
450         width, depth, 0, // Base vertex 3  
451         width / 2, depth / 2, height // Apex vertex  
452     );  
453  
454     mesh.getFaces().addAll(  
455         0, 0, 1, 0, 4, 0,  
456         1, 0, 2, 0, 4, 0,  
457         2, 0, 3, 0, 4, 0,  
458         3, 0, 0, 0, 4, 0  
459     );  
460     return mesh;  
461 }
```

Descripción:

Crea una malla para una pirámide con las dimensiones dadas.

Argumentos:

- float width, height, depth: Dimensiones de la pirámide.

Clase Item:

- **Código:**

```

package controller.implement;

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

public class Item extends Molde{
    //private String diseñoId;}
    private String room;

    public Item(String nombreObjeto, double base, double altura, double profundidad) {
        super(nombreObjeto, base, altura, profundidad);
    }

    public String getUroom() {
        return room;
    }

    public void setUroom(String usuario) {
        this.room = usuario;
    }

    @Override
    public void getArea(double base, double altura) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public String toString() {
        return "Ambiente{" + super.toString() + "nombre=" + '}';
    }
}

```

- **Descripción:**

Sirve para crear un objeto de tipo “Item” los cuales tendran 3 parámetros muy importantes a tener en cuenta, la base, la altura y la profundidad.

Clase Room:

- **Código:**

```

package controller.implement;

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

public class Room extends Molde{

    public Room(String NombreObjeto, double base, double altura, double profundidad) {
        super(NombreObjeto, base, altura, profundidad);
    }
}

```

```

        @Override
        public void getArea(double x, double y) {
            throw new UnsupportedOperationException("Not supported yet.");
        }
    }
}

```

- **Descripción:** Sirve para crear un objeto de tipo "Item" los cuales tendran 3 parámetros muy importantes a tener en cuenta, la base, la altura y la profundidad.

Clase RoomJFX:

- **Código:**

```

package controller.implement;

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

public class RoomJFX {

    private String nombre;
    private String hash;

    public RoomJFX(String nombre, String hash){
        this.nombre = nombre;
        this.hash = hash;
    }

    public String getNombre(){
        return nombre;
    }

    public String getHash(){
        return hash;
    }
}

```

- **Descripción:**
RoomJFX guarda en formato nombre y hash, lo cual nos facilita guardarlo en la base de datos y se utiliza también para la clase RooMades donde se guardan las medidas del cuarto y los numeros de objetos.

Clase Usuario:

- **Código:**

Importaciones y atributos:

```

package controller.implement;
import model.services.ServicesUsuario;

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

public class Usuario {

    private String username, password, email, name, userId;
    ServicesUsuario serUsu = new ServicesUsuario();

```

Constructores:

```

public Usuario (String username, String password, String email, String name){
    //this.userId = UUID.randomUUID().toString();
    this.username= username;
    this.password= password;
    this.email= email;
    this.name= name;
    this.userId= serUsu.setId();
}

```

Getters and Setters:

```

public String getUsername() {
    return username;
}

public void setUsername(String username) {
    this.username = username;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

```

@Override

```

@Override
public String toString() {
    return username+", "+password+", "+email+", "+name;
}

```

- **Descripción:**
Clase para crear el usuario con todo sus respectivos datos de tipo **String**

Classes View:

Clase RooMakingJFX3D:

- **OrdenarObjetos()**


```

private void OrdenarObjetos(Group root3D) {
    Random random = new Random();
    double rangoMinX = -wallBack.getWidth() / 2;
    double rangoMaxX = wallBack.getWidth() / 2;
    double rangoMinZ = -wallLeft.getDepth() / 2;
    double rangoMaxZ = wallLeft.getDepth() / 2;
    double margenSeguridad = 50;
    List<Movable> colocados = new ArrayList<>();
    List<Pair<Double, Double>> espaciosOcupadosX = new ArrayList<>();
    List<Pair<Double, Double>> espaciosOcupadosZ = new ArrayList<>();

    boolean considerarIluminacion = servIllum.actualizarEstadoIluminacion(root3D);

    for (javafx.scene.Node node : root3D.getChildren()) {
        if (node instanceof Movable) {
            Movable movable = (Movable) node;
            if (movable instanceof Ventana) {
                movable.setFijo(positionsLocked);
            }
            if (!movable.isFijo()) {
                continue;
            }

            boolean solapado;
            int intentos = 0;
            int maxIntentos = 100;

            do {
                solapado = false;
                intentos++;

                if (intentos > maxIntentos) {
                    System.out.println("No se encontró una posición adecuada para el objeto tras " + maxIntentos + " intentos.");
                    break;
                }

                Bounds dimensiones = movable.getBoundsInParent();
                double anchura = dimensiones.getWidth();
                double profundidad = dimensiones.getDepth();
                double altura = dimensiones.getHeight();

                double efectivoMinX = rangoMinX + anchura / 2 + margenSeguridad;
                double efectivoMaxX = rangoMaxX - anchura / 2 - margenSeguridad;
                double efectivoMinZ = rangoMinZ + profundidad / 2 + margenSeguridad;
                double efectivoMaxZ = rangoMaxZ - profundidad / 2 - margenSeguridad;

                double nuevaX, nuevaZ;
                int rotacion;

```

```

double efectivoMinX = rangoMinX + anchura / 2 + margenSeguridad;
double efectivoMaxX = rangoMaxX - anchura / 2 - margenSeguridad;
double efectivoMinZ = rangoMinZ + profundidad / 2 + margenSeguridad;
double efectivoMaxZ = rangoMaxZ - profundidad / 2 - margenSeguridad;

double nuevaX, nuevaZ;
int rotacion;

if (considerarIluminacion) {
    Pair<Double, Double> posicion = servIllum.determinarPosicionSegunIluminacion(movable, efectivoMinX, efectivoMaxX,
        efectivoMinZ, efectivoMaxZ);
    nuevaX = posicion.getKey();
    nuevaZ = posicion.getValue();
    rotacion = determinarRotacion(nuevaX, nuevaZ, efectivoMinX, efectivoMaxX, efectivoMinZ, efectivoMaxZ);
} else {
    if (random.nextBoolean()) {
        nuevaX = random.nextBoolean() ? efectivoMinX : efectivoMaxX;
        nuevaZ = efectivoMinZ + (efectivoMaxZ - efectivoMinZ) * random.nextDouble();
        rotacion = (nuevaX == efectivoMinX) ? 0 : 180;
    } else {
        nuevaZ = random.nextBoolean() ? efectivoMinZ : efectivoMaxZ;
        nuevaX = efectivoMinX + (efectivoMaxX - efectivoMinX) * random.nextDouble();
        rotacion = (nuevaZ == efectivoMinZ) ? 270 : 90;
    }
}

movible.setRotationAxis(Rotate.Y_AXIS);
movible.setRotate(rotacion);

for (Movable colocado : colocados) {
    if (isOverlapping(movable, colocado)) {
        solapado = true;
        break;
    }
}

if (!solapado) {
    movable.setTranslateX(nuevaX);
    movable.setTranslateZ(nuevaZ);
    movable.setTranslateY(wallBack.getHeight() / 2 - altura / 2);
    colocados.add(movable);

    espaciosOcupadosX.add(new Pair<>(nuevaX - anchura / 2, nuevaX + anchura / 2));
    espaciosOcupadosZ.add(new Pair<>(nuevaZ - profundidad / 2, nuevaZ + profundidad / 2));
} while (solapado);
}
}
}

```

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.

actualizarEstadolluminacion()

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(Movable movable, 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(Movable obj1, Movable obj2)

```
public boolean isOverlapping(Node objeto1, Node objeto2) {
    Bounds bounds1 = objeto1.getBoundsInParent();
    Bounds bounds2 = objeto2.getBoundsInParent();

    return bounds1.intersects(bounds2);
}
```

Descripción: Verifica si dos objetos Movable 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)

```

private void showDimensionDialog(String objectType, Group root3D) {
    Stage dialog = new Stage();
    dialog.setTitle("Ingresar dimensiones para " + objectType);

    VBox dialogVBox = new VBox(10);
    dialogVBox.setPadding(new Insets(10));

    Label widthLabel = new Label("Ancho:");
    TextField widthField = new TextField();
    Label heightLabel = new Label("Altura:");
    TextField heightField = new TextField();
    Label depthLabel = new Label("Profundidad:");
    TextField depthField = new TextField();

    if (objectType.equals("Cilindro")) {
        depthLabel.setDisable(true);
        depthField.setDisable(true);
    } else if (objectType.equals("Esfera")) {
        depthLabel.setDisable(true);
        depthField.setDisable(true);
        heightLabel.setDisable(true);
        heightField.setDisable(true);
    } else if (objectType.equals("Cubo")) {
    }

    Button createButton = new Button("Crear");
    createButton.setOnAction(e -> {
        try {
            double width = Double.parseDouble(widthField.getText());
            double height = objectType.equals("Esfera") ? width : Double.parseDouble(heightField.getText());
            double depth = objectType.equals("Esfera") || objectType.equals("Cilindro") ? height : Double.parseDouble(depthField.getText());
            addObject(objectType, width, height, depth, root3D);
            dialog.close();
        } catch (NumberFormatException ex) {
            Alert alert = new Alert(AlertType.ERROR);
            alert.setTitle("Error");
            alert.setHeaderText("Dimensiones inválidas");
            alert.setContentText("Por favor ingrese dimensiones válidas.");
            alert.showAndWait();
        }
    });

    dialogVBox.getChildren().addAll(widthLabel, widthField, heightLabel, heightField, depthLabel, depthField, createButton);

    Scene dialogScene = new Scene(dialogVBox, 300, 250);
    dialog.setScene(dialogScene);
    dialog.show();
}

```

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()

```

private void showRoomDialog(Group root3D) {
    Stage dialog = new Stage();
    dialog.setTitle("Elija la room sobre la cual desea trabajar");

    TableView<Room> tableView = new TableView<>();

    TableColumn<Room, String> column1 = new TableColumn<>("Room");
    column1.setCellValueFactory(new PropertyValueFactory<>("nombreObjeto"));

    TableColumn<Room, Double> column2 = new TableColumn<>("Base");
    column2.setCellValueFactory(new PropertyValueFactory<>("base"));

    TableColumn<Room, Double> column3 = new TableColumn<>("Altura");
    column3.setCellValueFactory(new PropertyValueFactory<>("altura"));

    TableColumn<Room, Double> column4 = new TableColumn<>("Profundidad");
    column4.setCellValueFactory(new PropertyValueFactory<>("profundidad"));

    tableView.getColumns().add(column1);
    tableView.getColumns().add(column2);
    tableView.getColumns().add(column3);
    tableView.getColumns().add(column4);

    tableView.setPrefHeight(200);
    tableView.setPrefWidth(500);

    for (Room room : ServicesRoom.getRooms()) {
        tableView.getItems().add(room);
    }

    tableView.getSelectionModel().selectedItemProperty().addListener((obs, oldSelection, newSelection) -> {
        if (newSelection != null) {
            setRoomSeleccionada(newSelection);
        }
    });

    Button createButton = new Button("Crear");
    createButton.setOnAction(e -> {
        try {
            double width = roomSeleccionada.getBase();
            double height = roomSeleccionada.getAltura();
            double depth = roomSeleccionada.getProfundidad();

            if (floor != null) {
                root3D.getChildren().remove(floor);
            }
            if (wallBack != null) {
                root3D.getChildren().remove(wallBack);
            }
        }
    });
}

```

```

    }
    if (wallLeft != null) {
        root3D.getChildren().remove(wallLeft);
    }

    floor = createFloor(width, height, depth);
    wallBack = createWallBack(width, height, depth);
    wallLeft = createWallLeft(width, height, depth);

    root3D.getChildren().addAll(floor, wallBack, wallLeft);

    dialog.close();
}
catch (NumberFormatException ex) {
    Alert alert = new Alert(AlertType.ERROR);
    alert.setTitle("Error");
    alert.setHeaderText("Dimensiones inválidas");
    alert.setContentText("Por favor ingrese dimensiones válidas.");
    alert.showAndWait();
}
});

VBox vbox = new VBox(tableView, createButton);
Scene dialogScene = new Scene(vbox, 300, 250);
dialog.setScene(dialogScene);
dialog.show();
}

// ... and showDialog(Scene root3D) ...

```

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()


```

private void showItemDialog(Group root3D) {
    Stage dialog = new Stage();
    dialog.setTitle("Item's Creados");

    // Crear una tabla con el tipo Item
    TableView<Item> tableView = new TableView<>();

    // Crear las columnas y configurar cómo se llenan
    TableColumn<Item, String> column1 = new TableColumn<>("Item");
    column1.setCellValueFactory(new PropertyValueFactory<>("nombreObjeto"));

    TableColumn<Item, Double> column2 = new TableColumn<>("Base");
    column2.setCellValueFactory(new PropertyValueFactory<>("base"));

    TableColumn<Item, Double> column3 = new TableColumn<>("Altura");
    column3.setCellValueFactory(new PropertyValueFactory<>("altura"));

    TableColumn<Item, Double> column4 = new TableColumn<>("Profundidad");
    column4.setCellValueFactory(new PropertyValueFactory<>("profundidad"));

    tableView.getColumns().addAll(column1, column2, column3, column4);

    // Configurar un tamaño mínimo para la tabla si es necesario
    tableView.setPrefHeight(200);
    tableView.setPrefWidth(500);

    // Añadir los items a la tabla
    for (Item item : ServicesItem.getItems()) {
        tableView.getItems().add(item);
    }

    // Añadir un listener para detectar la selección de la fila
    tableView.getSelectionModel().selectedItemProperty().addListener((obs, oldSelection, newSelection) -> {
        if (newSelection != null) {
            setItemSeleccionado(newSelection);
        }
    });

    Button createButton = new Button("Crear");
    createButton.setOnAction(e -> {
        try {
            double width = itemSeleccionado.getBase();
            double height = itemSeleccionado.getAltura();
            double depth = itemSeleccionado.getProfundidad();

            // Verificar si ya existen los objetos y eliminarlos si es necesario
            if (newObjectItem != null) {
                root3D.getChildren().remove(newObjectItem);
            }
        }
    });
}

```

```

        if (newObjectItem != null) {
            root3D.getChildren().remove(newObjectItem);
        }

        // Crear el nuevo objeto
        newObjectItem = new ItemNuevo(width, height, depth);

        root3D.getChildren().add(newObjectItem);

        // Configurar la posición y eventos del objeto
        newObjectItem.setTranslateX(0);
        newObjectItem.setTranslateY(-500);
        newObjectItem.setTranslateZ(0);
        newObjectItem.setOnMousePressed(this::handleGroupPressed);
        newObjectItem.setOnMouseDragged(this::handleGroupDragged);
        setupContextMenu(newObjectItem, root3D); // Configurar menú contextual

        dialog.close();
    } catch (NumberFormatException ex) {
        Alert alert = new Alert(AlertType.ERROR);
        alert.setTitle("Error");
        alert.setHeaderText("Dimensiones inválidas");
        alert.setContentText("Por favor ingrese dimensiones válidas.");
        alert.showAndWait();
    }
});

VBox vbox = new VBox(tableView, createButton);
Scene dialogScene = new Scene(vbox, 300, 250);
dialog.setScene(dialogScene);
dialog.show();
}

```

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)

```
private Box createFloor(double width, double height, double depth) {  
    Box floor = new Box(width, 10, depth);  
    floor.setMaterial(new PhongMaterial(Color.GRAY));  
    floor.setTranslateY(floorY);  
    return floor;  
}
```

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)

```
private Box createWallBack(double width, double height, double depth) {  
    Box wallBack = new Box(width, height, 10);  
    wallBack.setMaterial(new PhongMaterial(Color.GRAY));  
    wallBack.setTranslateZ(depth/2);  
    return wallBack;  
}
```

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)

```
private Box createWallLeft(double width, double height, double depth) {  
    Box wallLeft = new Box(10, height, depth);  
    wallLeft.setMaterial(new PhongMaterial(Color.GRAY));  
    wallLeft.setTranslateX(-width/2);  
    return wallLeft;  
}
```

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)

```
private ItemNuevo createItemNuevo(double width, double height, double depth){  
    ItemNuevo itemNuevo = new ItemNuevo(width, height, depth);  
    return itemNuevo;  
}
```

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)

```
private void addObject(String objectType, double width, double height, double depth, Group root3D) {
    Movable newObject = null;
    switch (objectType) {
        case "Prisma":
            //newObject = new Box(width, height, depth);
            break;
        case "Esfera":
            newObject = new Sphere3D(width, Color.RED);
            break;
        case "Cilindro":
            newObject = new Cylinder3D(width/2, height, Color.GREEN);
            break;
    }

    if (newObject != null) {
        //newObject.setMaterial(new PhongMaterial(Color.color(Math.random(), Math.random(), Math.random())));
        newObject.setTranslateX(0);
        newObject.setTranslateY(-500);
        newObject.setTranslateZ(0);
        newObject.setOnMousePressed(this::handleObjectPressed);
        newObject.setOnMouseDragged(this::handleObjectDragged);
        newObject.setOnMouseReleased(event -> selectedObject = null);
        setupContextMenu(newObject, root3D); // Configurar menú contextual
        root3D.getChildren().add(newObject);
    }
}
```

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)

```
private void addGroup(String groupType, Group root3D) {
    Movable newObject = null;
    switch (groupType) {
        case "Cama Simple":
            newObject = new CamaSimple();
            break;
        case "Cama Doble":
            newObject = new CamaDoble();
            break;
        case "Mesa de Noche":
            newObject = new MesaDeNoche();
            break;
        case "Armario":
            newObject = new Armario();
            break;
        case "Mueble":
            newObject = new Mueble();
            break;
        case "Mesa":
            newObject = new Mesa();
            break;
        case "Silla":
            newObject = new Silla();
            break;
        case "Televisor":
            newObject = new TV();
            break;
        case "Televisor Grande":
            newObject = new TVGrande();
            break;
    }

    if (newObject != null) {
        newObject.setTranslateX(0);
        newObject.setTranslateY(-500);
        newObject.setTranslateZ(0);
        newObject.setOnMousePressed(this::handleGroupPressed);
        newObject.setOnMouseDragged(this::handleGroupDragged);
        setupContextMenu(newObject, root3D); // Configurar menú contextual
        root3D.getChildren().add(newObject);
    }
}
```

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)

```
private Group cloneGroup(Group original) {
    Group clone = new Group();
    for (javafx.scene.Node node : original.getChildren()) {
        if (node instanceof Box) {
            Box originalBox = (Box) node;
            Box boxClone = new Box(originalBox.getWidth(), originalBox.getHeight(), originalBox.getDepth());
            boxClone.setMaterial(originalBox.getMaterial());
            boxClone.setTranslateX(originalBox.getTranslateX());
            boxClone.setTranslateY(originalBox.getTranslateY());
            boxClone.setTranslateZ(originalBox.getTranslateZ());
            clone.getChildren().add(boxClone);
        }
    }
    return clone;
}
```

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)

```
private void setupContextMenu(Shape3D object, Group root3D) {  
    ContextMenu contextMenu = new ContextMenu();  
    MenuItem removeItem = new MenuItem("Eliminar");  
    removeItem.setOnAction(e -> root3D.getChildren().remove(object));  
  
    MenuItem rotateItem = new MenuItem("Rotar");  
    rotateItem.setOnAction(e -> rotateObjectY(object));  
  
    contextMenu.getItems().addAll(removeItem, rotateItem);  
  
    object.setOnContextMenuRequested(e -> contextMenu.show(object, e.getScreenX(), e.getScreenY()));  
}
```

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)

```
private void setupContextMenu(Group group, Group root3D) {
    ContextMenu contextMenu = new ContextMenu();
    MenuItem deleteItem = new MenuItem("Eliminar");
    deleteItem.setOnAction(event -> root3D.getChildren().remove(group));

    MenuItem rotateItem = new MenuItem("Rotar");
    rotateItem.setOnAction(e -> rotateObjectY(group));

    contextMenu.getItems().addAll(deleteItem, rotateItem);

    group.setOnContextMenuRequested(event ->
        contextMenu.show(group, event.getScreenX(), event.getScreenY())
    );
}
```

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)

```
public static void rotateObjectY(Shape3D object) {  
    Rotate rotate = new Rotate(90, Rotate.Y_AXIS);  
    object.getTransforms().add(rotate);  
}
```

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)

```
public static void rotateObjectY(Group group) {
    Rotate rotate = new Rotate(90, Rotate.Y_AXIS);
    group.getTransforms().add(rotate);
}
```

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)

```
private void handleObjectPressed(MouseEvent event) {
    if (event.getButton() == MouseButton.PRIMARY) {
        selectedShape = (Movable) event.getSource();
        objectMouseOffsetX = selectedObject.getTranslateX() - event.getSceneX();
        objectMouseOffsetY = selectedObject.getTranslateY() - event.getSceneY();

        if (event.isShiftDown()) {
            objectMouseOffsetZ = selectedObject.getTranslateZ() - event.getSceneY();
        }
    }
    event.consume();
}
```

Descripción:

Maneja el evento de presionar un objeto de tipo Movable. 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)

```
private void handleObjectDragged(MouseEvent event) {  
    if (selectedShape != null && !positionsLocked) {  
        Point3D newPosition = unproject(event.getSceneX(), event.getSceneY(), selectedShape.getTranslateZ());  
        selectedShape.setTranslateX(newPosition.getX());  
        selectedShape.setTranslateY(newPosition.getY());  
  
        if (event.isShiftDown()) {  
            double newZ = selectedShape.getTranslateZ() + (event.getSceneY() - (selectedShape.getTranslateY() - objectMouseOffsetY));  
            selectedShape.setTranslateZ(newZ);  
        }  
    }  
    event.consume();  
}
```

Descripción:

Maneja el evento de arrastrar un objeto de tipo Movable. 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 Movable. 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 Movable. 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 Movable 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)

```

private Point3D unproject(double sceneX, double sceneY, double z) {
    double subSceneX = sceneX - RooMakingJFX3DStart.getSubScene().getLayoutX();
    double subSceneY = sceneY - RooMakingJFX3DStart.getSubScene().getLayoutY();

    double normalizedX = (subSceneX / RooMakingJFX3DStart.getSubScene().getWidth()) * 2 - 1;
    double normalizedY = -((subSceneY / RooMakingJFX3DStart.getSubScene().getHeight()) * 2 - 1);

    Point3D clipPoint = new Point3D(normalizedX, normalizedY, -1);

    Point3D viewPoint = RooMakingJFX3DStart.getCameraExterna().localToScene(clipPoint);
    viewPoint = RooMakingJFX3DStart.getCameraExterna().sceneToLocal(viewPoint);

    Point3D worldPoint = new Point3D(viewPoint.getX() * -RooMakingJFX3DStart.getCameraExterna().getTranslateZ(),
                                     viewPoint.getY() * RooMakingJFX3DStart.getCameraExterna().getTranslateZ(),
                                     z);

    Rotate rxInverse = new Rotate(-rotateY, Rotate.X_AXIS);
    Rotate ryInverse = new Rotate(-rotateX, Rotate.Y_AXIS);
    worldPoint = rxInverse.transform(worldPoint);
    worldPoint = ryInverse.transform(worldPoint);

    return worldPoint;
}

```

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()

```
void startPhysics(Group root3D) {
    Timeline timeline = new Timeline(new KeyFrame(Duration.millis(16), e -> {
        for (javafx.scene.Node node : root3D.getChildren()) {
            if ((node instanceof Shape3D || node instanceof Group) && node != selectedObject) {
                if (node instanceof Ventana) {
                    detectCollisions(node, root3D);
                } else {
                    applyGravity(node);
                    detectCollisions(node, root3D);
                }
            }
        }
    }));
    timeline.setCycleCount(Timeline.INDEFINITE);
    timeline.play();
}
```

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)

```
private void applyGravity(javafx.scene.Node node) {  
    if (node instanceof Group) {  
        for (javafx.scene.Node child : ((Group) node).getChildren()) {  
            double bottomY = child.getTranslateY() + getObjectHeight(child) / 2 + node.getTranslateY();  
            if (bottomY < floorY) {  
                node.setTranslateY(node.getTranslateY() + gravity);  
            } else {  
                node.setTranslateY(floorY - getObjectHeight(child) / 2);  
            }  
        }  
    } else {  
        double bottomY = node.getTranslateY() + getObjectHeight(node) / 2;  
        if (bottomY < floorY) {  
            node.setTranslateY(node.getTranslateY() + gravity);  
        } else {  
            node.setTranslateY(floorY - getObjectHeight(node) / 2);  
        }  
    }  
}
```

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)

```
private double getObjectHeight(javafx.scene.Node node) {  
    if (node instanceof Box) {  
        return ((Box) node).getHeight();  
    } else if (node instanceof Sphere) {  
        return ((Sphere) node).getRadius() * 2;  
    } else if (node instanceof Cylinder) {  
        return ((Cylinder) node).getHeight();  
    } else if (node instanceof Group) {  
        double maxHeight = 0;  
        double grHeight = 0;  
        for (javafx.scene.Node child : ((Group) node).getChildren()) {  
            if (child instanceof Box) {  
                grHeight = ((Box) child).getHeight() + child.getTranslateY();  
                return grHeight;  
            } else if (child instanceof Sphere) {  
                grHeight = ((Sphere) child).getRadius() * 2 + child.getTranslateY();  
                return grHeight;  
            } else if (child instanceof Cylinder) {  
                grHeight = ((Cylinder) child).getHeight() + child.getTranslateY();  
                return grHeight;  
            }  
            maxHeight = Math.max(maxHeight, grHeight);  
        }  
        return maxHeight;  
    }  
    return 0;  
}
```

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(javax.swing.scene.Node node)

```
private void detectCollisions(javax.swing.scene.Node node, Group root3D) {  
    for (javax.swing.scene.Node otherNode : root3D.getChildren()) {  
        if (node != otherNode && (otherNode instanceof Shape3D || otherNode instanceof Group)) {  
            if (isColliding(node, otherNode)) {  
                resolveCollision(node, otherNode);  
            }  
        }  
    }  
}
```

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)

```
private boolean isColliding(javafx.scene.Node node1, javafx.scene.Node node2) {  
    // Obtener las Bounding Boxes de los nodos  
    Bounds bounds1 = node1.getBoundsInParent();  
    Bounds bounds2 = node2.getBoundsInParent();  
  
    // Comprobar si las Bounding Boxes se intersectan  
    return bounds1.intersects(bounds2);  
}
```

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)

```

private void resolveCollision(javaafx.scene.Node model, javafx.scene.Node node2) {
    Bounds bounds1 = model.getBoundsInParent();
    Bounds bounds2 = node2.getBoundsInParent();

    // Calcular las diferencias en las posiciones de los nodos
    double diffX = bounds1.getCenterX() - bounds2.getCenterX();
    double diffY = bounds1.getCenterY() - bounds2.getCenterY();
    double diffZ = bounds1.getCenterZ() - bounds2.getCenterZ();

    // Calcular la superposición en cada eje
    double overlapX = (bounds1.getWidth() + bounds2.getWidth()) / 2 - Math.abs(diffX);
    double overlapY = (bounds1.getHeight() + bounds2.getHeight()) / 2 - Math.abs(diffY);
    double overlapZ = (bounds1.getDepth() + bounds2.getDepth()) / 2 - Math.abs(diffZ);

    if ((model == floor || node2 == floor) || (model == wallBack || node2 == wallBack) || (model == wallLeft || node2 == wallLeft)
        && ((floor != null) && (wallBack != null) && (wallLeft != null))) {
        model.setTranslateY(model.getTranslateY() - 1);
        node2.setTranslateY(node2.getTranslateY() + 1);
    } else {
        // Asegurarse de que hay una colisión en algún eje
        if (overlapX > 0 && overlapY > 0 && overlapZ > 0) {
            // Encuentra el eje con la menor superposición para mover el nodo en esa dirección
            if (overlapX < overlapY && overlapX < overlapZ) {
                double adjustment = overlapX / 2;
                if (diffX > 0) {
                    model.setTranslateX(model.getTranslateX() + adjustment);
                    node2.setTranslateX(node2.getTranslateX() - adjustment);
                } else {
                    model.setTranslateX(model.getTranslateX() - adjustment);
                    node2.setTranslateX(node2.getTranslateX() + adjustment);
                }
            } else if (overlapY < overlapX && overlapY < overlapZ) {
                double adjustment = overlapY / 2;
                if (diffY > 0) {
                    model.setTranslateY(model.getTranslateY() + adjustment);
                    node2.setTranslateY(node2.getTranslateY() - adjustment);
                } else if (overlapY < overlapX && overlapY < overlapZ) {
                    double adjustment = overlapY / 2;
                    if (diffY > 0) {
                        model.setTranslateY(model.getTranslateY() + adjustment);
                        node2.setTranslateY(node2.getTranslateY() - adjustment);
                    } else {
                        model.setTranslateY(model.getTranslateY() - adjustment);
                        node2.setTranslateY(node2.getTranslateY() + adjustment);
                    }
                }
            } else {
                double adjustment = overlapZ / 2;
                if (diffZ > 0) {
                    model.setTranslateZ(model.getTranslateZ() + adjustment);
                    node2.setTranslateZ(node2.getTranslateZ() - adjustment);
                } else {
                    model.setTranslateZ(model.getTranslateZ() - adjustment);
                    node2.setTranslateZ(node2.getTranslateZ() + adjustment);
                }
            }
        }
    }
}

```

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()

```
private String saveScene(Group root3D) {
    Map<String, Object> sceneData = new HashMap<>();
    List<Map<String, Object>> objectsData = new ArrayList<>();

    for (javafx.scene.Node node : root3D.getChildren()) {
        if (node instanceof Shape3D || node instanceof Group || node instanceof Movable || node instanceof PointLight) {
            objectsData.add(encodeObject(node));
        }
    }

    sceneData.put("objects", objectsData);

    try {
        String json = objectMapper.writeValueAsString(sceneData);
        String encodedJson = Base64.getEncoder().encodeToString(json.getBytes());

        // Aquí puedes guardar 'encodedJson' en un archivo o mostrarlo al usuario
        System.out.println("Escena guardada: " + encodedJson);
        return encodedJson;
    } catch (JsonProcessingException e) {
        return null;
    }
}
```

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)

```

private void loadScene(String encodedJson, Group root3D) {
    // Aquí deberías obtener el 'encodedJson' desde un archivo o entrada del usuario
    try {
        byte[] decodedBytes = Base64.getDecoder().decode(encodedJson);
        String json = new String(decodedBytes);
        Map<String, Object> sceneData = objectMapper.readValue(json, new TypeReference<Map<String, Object>>() {});

        root3D.getChildren().clear();
        List<Map<String, Object>> objectsData = (List<Map<String, Object>>) sceneData.get("objects");
        for (Map<String, Object> objectData : objectsData) {
            javafx.scene.Node node = decodeObject(objectData, root3D);
            if (node != null) {
                root3D.getChildren().add(node);
            }
        }
    } catch (JsonProcessingException e) {
        System.out.println(e);
    }
}

```

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)

```
private Map<String, Object> encodeObject(javafx.scene.Node node) {  
    Map<String, Object> objectData = new HashMap<>();  
    objectData.put("type", node.getClass().getSimpleName());  
    objectData.put("translateX", node.getTranslateX());  
    objectData.put("translateY", node.getTranslateY());  
    objectData.put("translateZ", node.getTranslateZ());  
  
    if (node instanceof Shape3D shape) {  
        encodeShape3D(shape, objectData);  
    } else if (node instanceof Group group) {  
        encodeGroup(group, objectData);  
    } else if (node instanceof PointLight light) {  
        encodePointLight(light, objectData);  
    }  
  
    encodeTransforms(node, objectData);  
  
    return objectData;  
}
```

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)

```

private void encodeShape3D(Shape3D shape, Map<String, Object> objectData) {
    PhongMaterial material = (PhongMaterial) shape.getMaterial();
    if (material != null) {
        Color color = material.getDiffuseColor();
        objectData.put("color", String.format("#%02X%02X%02X",
            (int) (color.getRed() * 255),
            (int) (color.getGreen() * 255),
            (int) (color.getBlue() * 255)));
    }

    if (shape instanceof Box box) {
        objectData.put("width", box.getWidth());
        objectData.put("height", box.getHeight());
        objectData.put("depth", box.getDepth());
    } else if (shape instanceof Sphere sphere) {
        objectData.put("radius", sphere.getRadius());
    } else if (shape instanceof Cylinder cylinder) {
        objectData.put("radius", cylinder.getRadius());
        objectData.put("height", cylinder.getHeight());
    }
}

```

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)

```

private void encodeGroup(Group group, Map<String, Object> objectData) {
    List<Map<String, Object>> childrenData = new ArrayList<>();
    for (javafx.scene.Node child : group.getChildren()) {
        childrenData.add(encodeObject(child));
    }
    objectData.put("children", childrenData);
}

```

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)

```
private void encodePointLight(PointLight light, Map<String, Object> objectData) {  
    Color color = light.getColor();  
    objectData.put("color", String.format("#%02X%02X%02X",  
        (int) (color.getRed() * 255),  
        (int) (color.getGreen() * 255),  
        (int) (color.getBlue() * 255)));  
}
```

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)

```

private void encodeTransforms(javafx.scene.Node node, Map<String, Object> objectData) {
    List<Map<String, Object>> transformsData = new ArrayList<>();
    for (Transform transform : node.getTransforms()) {
        if (transform instanceof Rotate rotate) {
            Map<String, Object> rotateData = new HashMap<>();
            rotateData.put("type", "Rotate");
            rotateData.put("angle", rotate.getAngle());
            rotateData.put("pivotX", rotate.getPivotX());
            rotateData.put("pivotY", rotate.getPivotY());
            rotateData.put("pivotZ", rotate.getPivotZ());
            Point3D axis = rotate.getAxis();
            rotateData.put("axisX", axis.getX());
            rotateData.put("axisY", axis.getY());
            rotateData.put("axisZ", axis.getZ());
            transformsData.add(rotateData);
        }
    }
    if (!transformsData.isEmpty()) {
        objectData.put("transforms", transformsData);
    }
}

```

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)

```

private javafx.scene.Node decodeObject(Map<String, Object> objectData, Group root3D) {
    String type = (String) objectData.get("type");
    javafx.scene.Node node = null;

    switch (type) {
        case "Box", "Sphere", "Cylinder" -> node = createShape3D(type, objectData, root3D);
        case "Group", "ItemNuevo", "MesaDeNoche", "Mueble", "Armario", "Mesa", "Silla", "CamaSimple", "CamaDoble", "TV", "TVGrande", "Ventana" -> node = createGroupNode(type, objectData, root3D);
        case "PointLight" -> node = createPointLight(objectData);
    }

    if (node != null) {
        setCommonProperties(node, objectData);
    }

    return node;
}

```

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)

```
private Shape3D createShape3D(String type, Map<String, Object> objectData) {
    Shape3D shape = switch (type) {
        case "Box" -> new Box(
            (double) objectData.get("width"),
            (double) objectData.get("height"),
            (double) objectData.get("depth")
        );
        case "Sphere" -> new Sphere((double) objectData.get("radius"));
        case "Cylinder" -> new Cylinder(
            (double) objectData.get("radius"),
            (double) objectData.get("height")
        );
        default -> throw new IllegalArgumentException("Unknown shape type: " + type);
    };

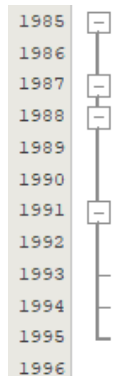
    setColorForShape3D(shape, objectData);
    setupContextMenu(shape);
    return shape;
}
```

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)



```
private void setColorForShape3D(Shape3D shape, Map<String, Object> objectData) {
    String colorStr = (String) objectData.get("color");
    if (colorStr != null && !colorStr.isEmpty()) {
        try {
            Color color = Color.web(colorStr);
            shape.setMaterial(new PhongMaterial(color));
        } catch (IllegalArgumentException e) {
            System.err.println("Invalid color string for Shape3D: " + colorStr);
        }
    }
}
```

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)

```
1997 private Group createGroupNode(String type, Map<String, Object> objectData) {
1998     Group group = switch (type) {
1999         case "Group" -> new Group();
2000         case "ItemNuevo" -> new ItemNuevo(
2001             getDoubleOrDefault(objectData, "width", 100.0),
2002             getDoubleOrDefault(objectData, "height", 100.0),
2003             getDoubleOrDefault(objectData, "depth", 100.0)
2004         );
2005         case "MesaDeNoche" -> new MesaDeNoche();
2006         case "Mueble" -> new Mueble();
2007         case "Armario" -> new Armario();
2008         case "Mesa" -> new Mesa();
2009         case "Silla" -> new Silla();
2010         case "CamaSimple" -> new CamaSimple();
2011         case "CamaDoble" -> new CamaDoble();
2012         case "TV" -> new TV();
2013         case "TVGrande" -> new TVGrande();
2014         case "Ventana" -> new Ventana();
2015         default -> throw new IllegalArgumentException("Unknown group type: " + type);
2016     };
2017
2018     List<Map<String, Object>> childrenData = (List<Map<String, Object>>) objectData.get("children");
2019     if (childrenData != null) {
2020         for (Map<String, Object> childData : childrenData) {
2021             javafx.scene.Node child = decodeObject(childData);
2022             if (child != null) {
2023                 group.getChildren().add(child);
2024             }
2025         }
2026     }
2027
2028     if (group instanceof ItemNuevo itemNuevo) {
2029         itemNuevo.setTranslateX(0);
2030         itemNuevo.setTranslateY(-500);
2031         itemNuevo.setTranslateZ(0);
2032         itemNuevo.setOnMousePressed(this::handleGroupPressed);
2033         itemNuevo.setOnMouseDragged(this::handleGroupDragged);
2034     }
2035
2036     setupContextMenu(group);
2037     return group;
2038 }
```

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)

```
2040 private PointLight createPointLight (Map<String, Object> objectData) {
2041     PointLight light = new PointLight();
2042     String colorStr = (String) objectData.get("color");
2043     if (colorStr != null && !colorStr.isEmpty()) {
2044         try {
2045             light.setColor(Color.web(colorStr));
2046         } catch (IllegalArgumentException e) {
2047             System.err.println("Invalid color string for PointLight: " + colorStr);
2048         }
2049     }
2050     return light;
2051 }
```

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)

```
2053 private void setCommonProperties(javafx.scene.Node node, Map<String, Object> objectData) {
2054     node.setTranslateX((double) objectData.getDefault("translateX", 0.0));
2055     node.setTranslateY((double) objectData.getDefault("translateY", 0.0));
2056     node.setTranslateZ((double) objectData.getDefault("translateZ", 0.0));
2057
2058     List<Map<String, Object>> transformsData = (List<Map<String, Object>>) objectData.get("transforms");
2059     if (transformsData != null) {
2060         for (Map<String, Object> transformData : transformsData) {
2061             String transformType = (String) transformData.get("type");
2062             if ("Rotate".equals(transformType)) {
2063                 double angle = (double) transformData.get("angle");
2064                 double pivotX = (double) transformData.get("pivotX");
2065                 double pivotY = (double) transformData.get("pivotY");
2066                 double pivotZ = (double) transformData.get("pivotZ");
2067                 double axisX = (double) transformData.get("axisX");
2068                 double axisY = (double) transformData.get("axisY");
2069                 double axisZ = (double) transformData.get("axisZ");
2070                 Point3D axis = new Point3D(axisX, axisY, axisZ);
2071                 node.getTransforms().add(new Rotate(angle, pivotX, pivotY, pivotZ, axis));
2072             }
2073         }
2074     }
2075 }
```

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)

```
private double getDoubleOrDefault(Map<String, Object> data, String key, double defaultValue) {  
    Object value = data.get(key);  
    return value instanceof Number ? ((Number) value).doubleValue() : 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()

```

2082 private void showCargarDialog() {
2083     Stage secondaryStage = new Stage();
2084     Map<String, List<String>> resultado = null;
2085
2086     ComboBox<String> roomComboBox = new ComboBox<>();
2087
2088     if (resultado == null) {
2089         resultado = cargarOpcionesDesdeBD();
2090     }
2091
2092     roomComboBox.getItems().clear();
2093
2094     List<String> opciones = resultado.get("opciones");
2095     List<String> hashes = resultado.get("hashes");
2096
2097     roomComboBox.getItems().addAll(opciones);
2098     roomComboBox.setStyle(
2099         "-fx-background-color: linear-gradient(to bottom, rgb(245,245,245), rgb(210,210,210));" + // Degradado de gris claro
2100         "-fx-border-color: rgb(200,200,200);" + // Borde gris claro
2101         "-fx-border-radius: 5;" + // Bordes redondeados
2102         "-fx-background-radius: 5;" // Bordes redondeados para el fondo
2103     );
2104     roomComboBox.setPromptText("Selecciona una opción");
2105
2106     Button cargarRoomButton = new Button("Cargar Room");
2107     cargarRoomButton.setStyle(
2108         "-fx-background-color: linear-gradient(to bottom, rgb(126,188,137), rgb(162,217,206));" + // Degradado verde
2109         "-fx-text-fill: white;" + // Texto blanco
2110         "-fx-font-weight: bold;" + // Texto en negrita
2111         "-fx-border-color: rgb(126,188,137);" + // Borde verde
2112         "-fx-border-radius: 5;" + // Bordes redondeados
2113         "-fx-background-radius: 5;" // Bordes redondeados para el fondo
2114     );
2115     cargarRoomButton.setVisible(false);
2116
2117     roomComboBox.setOnAction(e -> {
2118         if (roomComboBox.getValue() != null) {
2119             cargarRoomButton.setVisible(true);
2120         }
2121     });
2122
2123     roomComboBox.getSelectionModel().selectedIndexProperty().addListener((observable, oldValue, newValue) -> {
2124         if (newValue.intValue() != -1) {
2125             int indiceSeleccionado = newValue.intValue();
2126             String elemento = hashes.get(indiceSeleccionado);
2127             cargarRoomButton.setOnAction(e -> {
2128                 loadScene(elemento);
2129                 secondaryStage.close();
2130             });
2131         }
2132     });
2133
2134     VBox secondaryLayout = new VBox(10);
2135     secondaryLayout.setPadding(new Insets(15));
2136     secondaryLayout.setStyle(
2137         "-fx-background-color: rgb(245,245,220);" +
2138         "-fx-padding: 15;" + // Espaciado interno
2139         "-fx-border-color: rgb(200,200,200);" + // Borde gris claro
2140         "-fx-border-width: 2px;" + // Grosor del borde
2141         "-fx-border-radius: 10px;" // Bordes redondeados
2142     );
2143     secondaryLayout.getChildren().addAll(
2144         new Label("Rooms Guardadas:") {
2145             setStyle(
2146                 "-fx-font-size: 16px;" + // Tamaño de fuente
2147                 "-fx-font-weight: bold;" + // Negrita
2148                 "-fx-text-fill: rgb(0,0,0);" // Color del texto
2149             );
2150         },
2151         roomComboBox,
2152         cargarRoomButton
2153     );
2154
2155     Scene secondaryScene = new Scene(secondaryLayout, 300, 200);
2156     secondaryStage.setScene(secondaryScene);
2157     secondaryStage.setTitle("Cargar Room");
2158     secondaryStage.initModality(Modality.APPLICATION_MODAL); // Bloquea interacción con otras ventanas mientras está abierto
2159     secondaryStage.show();
2160 }
2161

```

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()

```
private Map<String, List<String>> cargarOpcionesDesdeBD() {
    List<RoomJFX> rooms = servRoomJFX.getRoomsJFXByUsuario(ServicesUsuario.getUsuario());

    if (rooms.isEmpty()) {
        System.out.println("No se encontraron habitaciones para el usuario.");
    } else {
        for (RoomJFX room : rooms) {
            ServicesRoomJFX.setRooms(room);
            System.out.println("Habitación: " + room.getNombre());
        }
    }

    List<String> opciones = new ArrayList<>();
    List<String> hashes = new ArrayList<>();

    List<RoomJFX> savedRooms = ServicesRoomJFX.getRooms();
    if (savedRooms.isEmpty()) {
        System.out.println("No se encontraron habitaciones guardadas.");
    } else {
        for (int i = 0; i < rooms.size(); i++) {
            opciones.add(rooms.get(i).getNombre());
            hashes.add(rooms.get(i).getHash());
            System.out.println(opciones);
        }
    }

    // Crear el resultado y añadir las listas
    Map<String, List<String>> resultado = new HashMap<>();

    resultado.put("opciones", opciones);
    resultado.put("hashes", hashes);

    return resultado;
}
```

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)

```
private void opcionesIluminacion(Box wallLeft, Box wallBack) {
    Stage secondaryStage = new Stage();

    Label pared1Label = new Label("Pared Trasera");
    ComboBox<String> pared1ComboBox = new ComboBox<>();
    pared1ComboBox.getItems().addAll("Bombillo", "Ventana", "sin luz");
    pared1ComboBox.setStyle(
        "-fx-background-color: linear-gradient(to bottom, rgb(162,217,206), rgb(126,188,137));" + // Degradado de verde claro a verde suave
        "-fx-border-color: rgb(210,180,140);" + // Borde color tierra
        "-fx-border-radius: 5;" // Bordes redondeados
    );

    Label pared2Label = new Label("Pared Izquierda");
    ComboBox<String> pared2ComboBox = new ComboBox<>();
    pared2ComboBox.getItems().addAll("Bombillo", "Ventana", "sin luz");
    pared2ComboBox.setStyle(
        "-fx-background-color: linear-gradient(to bottom, rgb(162,217,206), rgb(126,188,137));" + // Degradado de verde claro a verde suave
        "-fx-border-color: rgb(210,180,140);" + // Borde color tierra
        "-fx-border-radius: 5;" // Bordes redondeados
    );

    Label pared3Label = new Label("Pared Derecha");
    ComboBox<String> pared3ComboBox = new ComboBox<>();
    pared3ComboBox.getItems().addAll("Bombillo", "Ventana", "sin luz");
    pared3ComboBox.setStyle(
        "-fx-background-color: linear-gradient(to bottom, rgb(162,217,206), rgb(126,188,137));" + // Degradado de verde claro a verde suave
        "-fx-border-color: rgb(210,180,140);" + // Borde color tierra
        "-fx-border-radius: 5;" // Bordes redondeados
    );

    Label techoLabel = new Label("Techo");
    ComboBox<String> techoComboBox = new ComboBox<>();
    techoComboBox.getItems().addAll("Bombillo", "sin luz");
    techoComboBox.setStyle(
        "-fx-background-color: linear-gradient(to bottom, rgb(162,217,206), rgb(126,188,137));" + // Degradado de verde claro a verde suave
        "-fx-border-color: rgb(210,180,140);" + // Borde color tierra
        "-fx-border-radius: 5;" // Bordes redondeados
    );
}
```

```

Button aplicarButton = new Button("Aplicar");
aplicarButton.setStyle(
    "-fx-background-color: linear-gradient(to bottom, rgb(126,188,137), rgb(162,217,206));" + // Degradado de verde
    "-fx-text-fill: white;" + // Texto en blanco
    "-fx-font-weight: bold;" + // Texto en negrita
    "-fx-border-color: rgb(126,188,137);" + // Borde del botón
    "-fx-border-radius: 5;" + // Bordes redondeados
    "-fx-background-radius: 5;" // Bordes redondeados para el fondo
);
aplicarButton.setVisible(false); // Inicialmente oculto

pared1ComboBox.setOnAction(e -> verificarSeleccion(pared1ComboBox, pared2ComboBox, pared3ComboBox, techoComboBox, aplicarButton));
pared1ComboBox.setOnAction(e -> verificarSeleccion(pared1ComboBox, pared2ComboBox, pared3ComboBox, techoComboBox, aplicarButton));
pared3ComboBox.setOnAction(e -> verificarSeleccion(pared1ComboBox, pared2ComboBox, pared3ComboBox, techoComboBox, aplicarButton));
techoComboBox.setOnAction(e -> verificarSeleccion(pared1ComboBox, pared2ComboBox, pared3ComboBox, techoComboBox, aplicarButton));

aplicarButton.setOnAction(e -> {
    configuracionIluminacion(pared1ComboBox.getValue(), pared2ComboBox.getValue(), pared3ComboBox.getValue(), techoComboBox.getValue(), wallLeft, wallBack);
    secondaryStage.close();
});

VBox secondaryLayout = new VBox();
secondaryLayout.setAlignment(Pos.CENTER);
secondaryLayout.getChildren().addAll(pared1Label, pared1ComboBox, pared2Label, pared2ComboBox, pared3Label, pared3ComboBox, techoLabel, techoComboBox, aplicarButton);
secondaryLayout.setStyle(
    "-fx-background-color: linear-gradient(to bottom right, rgb(245,245,220), rgb(162,217,206));" + // Fondo degradado de beige a verde claro
    "-fx-border-color: rgb(210,180,140);" + // Color del borde tierra
    "-fx-border-width: 2px;" + // Grosor del borde
    "-fx-border-radius: 10px;" + // Bordes redondeados
    "-fx-background-radius: 10px;" + // Bordes redondeados para el fondo
    "-fx-effect: dropshadow(gaussian, rgba(0, 0, 0, 0.2), 10, 0.5, 0, 0);" // Efecto de sombra
);

Scene iluminacionDialog = new Scene(secondaryLayout, 300, 300);
secondaryStage.setScene(iluminacionDialog);
secondaryStage.sizeToScene();
secondaryStage.setTitle("Opciones de Iluminación");
secondaryStage.show();
}

```

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)

```
private void verificarSelecion(ComboBox<String> pared1ComboBox, ComboBox<String> pared2ComboBox, ComboBox<String> pared3ComboBox, ComboBox<String> techoComboBox, Button aplicarButton) {
    if (pared1ComboBox.getValue() != null &&
        pared2ComboBox.getValue() != null &&
        pared3ComboBox.getValue() != null &&
        techoComboBox.getValue() != null) {
        aplicarButton.setVisible(true);
    } else {
        aplicarButton.setVisible(false);
    }
}
```

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)

```

public void configuracionIluminacion(String pared1, String pared2, String pared3, String techo, Box wallLeft, Box wallBack) {
    // Limpia las configuraciones de luz existentes si es necesario
    root3D.getChildren().removeIf(node -> node instanceof PointLight || node instanceof Ventana);

    // Aplicar configuración para Pared 1
    if ("Bombillo".equals(pared1)) {
        PointLight luzPared1 = new PointLight(Color.WHITE);
        luzPared1.setTranslateX(wallBack.getTranslateX()); // Colocar la luz en la posición adecuada
        luzPared1.setTranslateY(wallBack.getTranslateY() * 0.8);
        luzPared1.setTranslateZ(wallBack.getTranslateZ());
        root3D.getChildren().add(luzPared1);
    } else if ("Ventana".equals(pared1)) {
        Ventana newVentana1 = new Ventana();
        newVentana1.setTranslateX(wallBack.getTranslateX());
        newVentana1.setTranslateY(wallBack.getTranslateY());
        newVentana1.setTranslateZ(wallBack.getTranslateZ() - 5);
        rotateObjectY(newVentana1);
        togglePositionsLock();
        root3D.getChildren().add(newVentana1);
        PointLight luzPared1 = new PointLight(Color.WHITE);
        luzPared1.setTranslateX(wallBack.getTranslateX());
        luzPared1.setTranslateY(wallBack.getTranslateY());
        luzPared1.setTranslateZ(wallBack.getTranslateZ() - 5);
        root3D.getChildren().add(luzPared1);
    }

    if ("Bombillo".equals(pared2)) {
        PointLight luzPared2 = new PointLight(Color.WHITE);
        luzPared2.setTranslateX(wallLeft.getTranslateX());
        luzPared2.setTranslateY(-wallLeft.getTranslateY() * 0.8);
        luzPared2.setTranslateZ(wallLeft.getTranslateX());
        root3D.getChildren().add(luzPared2);
    } else if ("Ventana".equals(pared2)) {
        Ventana newVentana2 = new Ventana();
        newVentana2.setTranslateX(wallLeft.getTranslateX() + 5);
        newVentana2.setTranslateY(wallLeft.getTranslateY());
        newVentana2.setTranslateZ(wallLeft.getTranslateZ());
        togglePositionsLock();
        root3D.getChildren().add(newVentana2);
        PointLight luzPared2 = new PointLight(Color.WHITE);
        luzPared2.setTranslateX(wallLeft.getTranslateX() + 5);
        luzPared2.setTranslateY(wallLeft.getTranslateY());
        luzPared2.setTranslateZ(wallLeft.getTranslateZ());
        root3D.getChildren().add(luzPared2);
    }

    if ("Bombillo".equals(pared3)) {
        PointLight luzPared3 = new PointLight(Color.WHITE);
        luzPared3.setTranslateX(-wallLeft.getTranslateX());
        luzPared3.setTranslateY(-wallLeft.getTranslateY() * 0.8);
        luzPared3.setTranslateZ(wallLeft.getTranslateZ());
        root3D.getChildren().add(luzPared3);
    } else if ("Ventana".equals(pared3)) {
        Ventana newVentana3 = new Ventana();
        newVentana3.setTranslateX(-wallLeft.getTranslateX() + 5);
        newVentana3.setTranslateY(wallLeft.getTranslateY());
        newVentana3.setTranslateZ(wallLeft.getTranslateZ());
        rotateObjectY(newVentana3);
        rotateObjectY(newVentana3);
        togglePositionsLock();
        root3D.getChildren().add(newVentana3);
        PointLight luzPared3 = new PointLight(Color.WHITE);
        luzPared3.setTranslateX(-wallLeft.getTranslateX() + 5);
        luzPared3.setTranslateY(wallLeft.getTranslateY());
        luzPared3.setTranslateZ(wallLeft.getTranslateZ());
        root3D.getChildren().add(luzPared3);
    }

    if ("Bombillo".equals(techo)) {
        PointLight luzTecho = new PointLight(Color.WHITE);
        luzTecho.setTranslateX(0);
        luzTecho.setTranslateY(-wallLeft.getTranslateY());
        luzTecho.setTranslateZ(0);
        root3D.getChildren().add(luzTecho);
    }
}

```

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)

```
public void addItemToScene(Group itemGroup) {  
    itemGroup.setTranslateX(0);  
    itemGroup.setTranslateY(0);  
    itemGroup.setTranslateZ(0);  
    itemGroup.setOnMousePressed(this::handleGroupPressed);  
    itemGroup.setOnMouseDragged(this::handleGroupPressed);  
    setupContextMenu(itemGroup);  
    root3D.getChildren().add(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).