```java
//RecipeService.java

package recipes;

import java.math.BigDecimal;
import java.time.LocalTime;
import java.util.List;
import java.util.Objects;
import java.util.Scanner;
import recipes.entity.Category;
import recipes.entity.Ingredient;
import recipes.entity.Recipe;
import recipes.entity.Step;
import recipes.entity.Unit;
import recipes.exception.DbException;
import recipes.service.RecipeService;

public class Recipes {

    private Scanner scanner = new Scanner(System.in);
    private RecipeService recipeService = new RecipeService();
    private Recipe currentRecipe;

    private List<String> operations = List.of(
      "1) Create and populate all tables",
      "2) Add a recipe",
      "3) List recipes",
      "4) Select current recipe",
      "5) Add ingredient to current recipe",
      "6) Add step to current recipe",
      "7) Add category to current recipe",
      "8) Modify step in current recipe",
      "9) Delete recipe" );

    public static void main(String[] args) {
        new Recipes().displayMenu();
    }

    private void displayMenu() {
        boolean done = false;

        while (!done) {
            try {
                int operation = getOperation();
                switch (operation) {
                case -1:
                    done = exitMenu();
                    break;

                case 1:
                    createTables();
                    break;

                case 2:
                    addRecipe();
                    break;

                case 3:
                    listRecipes();
                    break;

                case 4:
                    setCurrentRecipe();
                    break;

                case 5:
                    addIngredientToCurrentRecipe();
                    break;

                case 6:
                    addStepToCurrentRecipe();
                    break;

                case 7:
                    addCategoryToCurrentRecipe();
                    break;
                case 8:
                    modifyStepInCurrentRecipe();
                    break;

                case 9:
                    deleteRecipe();
                    break;

                default:
                    System.out.println("\n" + operation + " is not valid. Please try again.");
                    break;

                }
            } catch (Exception e) {
                System.out.println("\nError: " + e.toString() + " Try again.");
            }
        }
    }

    private void deleteRecipe() {
        listRecipes();
        Integer recipeId =getIntInput("Enter the ID of the recipe to delete");

        if(Objects.nonNull(recipeId)) {
            recipeService.deleteRecipe(recipeId);

            System.out.println("You have deleted recipe with id of : " + recipeId);

            if(Objects.nonNull(currentRecipe) && currentRecipe.getRecipeId().equals(recipeId)) {
                currentRecipe = null;
            }
        }
    }

    private void modifyStepInCurrentRecipe() {
        if (Objects.isNull(currentRecipe)) {
            System.out.println("\nPlease select a recipe first.");
            return;
        }

        List<Step> steps = recipeService.fetchSteps(currentRecipe.getRecipeId());

        System.out.println("\nSteps for current recipe");
        steps.forEach(step -> System.out.println("   " + step));

        Integer stepId = getIntInput("Enter step ID of step to modify");

        if(Objects.nonNull(stepId)) {
            String stepText = getStringInput("Enter new step text");

            if(Objects.nonNull(stepText)) {
                Step step = new Step();

                step.setStepId(stepId);
                step.setStepText(stepText);

                recipeService.modifyStep(step);
                currentRecipe = recipeService.fetchRecipeById(currentRecipe.getRecipeId());
            }
        }

    }

    private void addCategoryToCurrentRecipe() {
        if (Objects.isNull(currentRecipe)) {
            System.out.println("\nPlease select a recipe first.");
            return;
        }

        List<Category> categories = recipeService.fetchCategories();

        categories.forEach(category -> System.out.println("   " + category.getCategoryName()));

        String category =  getStringInput("Enter the category to add");

        if(Objects.nonNull(category)) {
            recipeService.addCategoryToRecipe(currentRecipe.getRecipeId(), category);
            currentRecipe = recipeService.fetchRecipeById(currentRecipe.getRecipeId());
        }
    }

    private void addStepToCurrentRecipe() {
        if (Objects.isNull(currentRecipe)) {
            System.out.println("\nPlease select a recipe first.");
            return;
        }

        String stepText = getStringInput("Enter the step text");

        if (Objects.nonNull(stepText)) {
            Step step = new Step();

            step.setRecipeId(currentRecipe.getRecipeId());
            step.setStepText(stepText);

            recipeService.addStep(step);
            currentRecipe = recipeService.fetchRecipeById(step.getRecipeId());
        }
    }

    private void addIngredientToCurrentRecipe() {
        if (Objects.isNull(currentRecipe)) {
            System.out.println("\nPlease select a recipe first.");
            return;
        }

        String name = getStringInput("Enter the ingredient name");
        String instruction = getStringInput("Enter an intruction if any (like \"finely chopped\")");
        Double inputAmount = getDoubleInput("Enter the ingredient amount (like .25)");
        List<Unit> units = recipeService.fetchUnits();

        BigDecimal amount = Objects.isNull(inputAmount) ? null : new BigDecimal(inputAmount).setScale(2);

        System.out.println("Units:");

        units.forEach(unit -> System.out.println("   "+ unit.getUnitId()+ ": "+ unit.getUnitNameSingular()+ "("+ ur

        Integer unitId = getIntInput("Enter a unit ID (press Enter for none)");

        Unit unit = new Unit();
        unit.setUnitId(unitId);

        Ingredient ingredient = new Ingredient();

        ingredient.setRecipeId(currentRecipe.getRecipeId());
        ingredient.setUnit(unit);
        ingredient.setIngredientName(name);
        ingredient.setInstruction(instruction);
        ingredient.setAmount(amount);

        recipeService.addIngrient(ingredient);
        currentRecipe = recipeService.fetchRecipeById(ingredient.getRecipeId());
        }

    private void setCurrentRecipe() {
        List<Recipe> recipes = listRecipes();

        Integer recipeId = getIntInput("Select a recipe ID");
        currentRecipe = null;

        for (Recipe recipe : recipes) {
            if (recipe.getRecipeId().equals(recipeId)) {
                currentRecipe = recipeService.fetchRecipeById(recipeId);
                break;
            }
        }
        if (Objects.isNull(currentRecipe)) {
            System.out.println("\nInvalid recipe Selected.");
        }
    }

    private List<Recipe> listRecipes() {
        List<Recipe> recipes = recipeService.fetchRecipes();
```

```java
        System.out.println("\nRecipes:");

        recipes.forEach(recipe -> System.out.println("      " + recipe.getRecipeId() + ": " + recipe.getRecipeName(

        return recipes;
    }

    private void addRecipe() {
        String name = getStringInput("Enter the recipe name ");
        String notes = getStringInput("Enter the recipe notes ");
        Integer NumServings = getIntInput("Enter the number of servings ");
        Integer prepMinutes = getIntInput("Enter prep time in minutes ");
        Integer cookMinutes = getIntInput(" Enter cook time in minutes ");

        LocalTime preptime = minutesToLocalTime(prepMinutes);
        LocalTime cooktime = minutesToLocalTime(cookMinutes);

        Recipe recipe = new Recipe();

        recipe.setRecipeName(name);
        recipe.setNotes(notes);
        recipe.setNumServings(NumServings);
        recipe.setPrepTime(preptime);
        recipe.setCookTime(cooktime);

        Recipe dbRecipe = recipeService.addRecipe(recipe);
        System.out.println("You added this recipe: \n" + dbRecipe);

        // 29:00 Doctor Rob does not add the class name in front of currentRecipe, when
        // i do
        // this i get an error
        currentRecipe = recipeService.fetchRecipeById(dbRecipe.getRecipeId());

    }

    private LocalTime minutesToLocalTime(Integer numMinutes) {
        int min = Objects.isNull(numMinutes) ? 0 : numMinutes;
        int hours = min / 60;
        int minutes = min % 60;

        return LocalTime.of(hours, minutes);

    }

    private void createTables() {
        recipeService.createAndPopulateTables();
        System.out.println("\nTables created and populated!");

    }

    private boolean exitMenu() {
        System.out.println("\nYou have now EXITED the menu.");
        return true;
    }

    private int getOperation() {
        printOperations();
        Integer op = getIntInput("\nEnter a operation number (Press ENTER to quit)");

        return Objects.isNull(op) ? -1 : op;
    }

    private void printOperations() {
        System.out.println();
        System.out.println("Here's what you can do:");

        operations.forEach(op -> System.out.println("   " + op));

        if (Objects.isNull(currentRecipe)) {
            System.out.println("\nYou are currently not working with a recipe!");
        } else {
            System.out.println("\nYou are working with recipe " + currentRecipe);
        }
    }

    private Integer getIntInput(String prompt) {
        String input = getStringInput(prompt);
        if (Objects.isNull(input)) {
            return null;
        }
        try {
            return Integer.parseInt(input);
        } catch (NumberFormatException e) {
            throw new DbException(input + " is not a valid number.");
        }
    }

    @SuppressWarnings("unused")
    private Double getDoubleInput(String prompt) {
        String input = getStringInput(prompt);

        if (Objects.isNull(input)) {
            return null;
        }
        try {
            return Double.parseDouble(input);
        } catch (NumberFormatException e) {
            throw new DbException(input + " is not a valid number.");
        }
    }

    private String getStringInput(String prompt) {
        System.out.print(prompt + ": ");
        String line = scanner.nextLine();

        return line.isBlank() ? null : line.trim();
    }

}




//RecipeDao.java


package recipes.dao;

import java.math.BigDecimal;
import java.sql.*;
import java.util.*;
import provided.util.DaoBase;
import recipes.entity.*;
import recipes.exception.DbException;
import java.time.*;

public class RecipeDao extends DaoBase {

    private static final String CATEGORY_TABLE = "category";
    private static final String INGREDIENT_TABLE = "ingredient";
    private static final String RECIPE_TABLE = "recipe";
    private static final String RECIPE_CATEGORY_TABLE = "recipe_category";
    private static final String STEP_TABLE = "step";
    private static final String UNIT_TABLE = "unit";

    public List<Recipe> fetchAllRecipes() {
        String sql = "SELECT * FROM " + RECIPE_TABLE + " ORDER BY recipe_id";

        try (Connection connection = DbConnection.getConnection()) {
            startTransaction(connection);

            try (PreparedStatement statement = connection.prepareStatement(sql)) {
                try (ResultSet resultset = statement.executeQuery()) {
                    List<Recipe> recipes = new LinkedList<>();

                    while (resultset.next()) {
                        recipes.add(extract(resultset, Recipe.class));
                    }
                    return recipes;
                }
            } catch (Exception e) {
                rollbackTransaction(connection);
                throw new DbException(e);
            }
        } catch (SQLException e) {
            throw new DbException(e);
        }
    }


    public Optional<Recipe> fetchRecipeById(Integer recipeId){
        String sql = "SELECT * FROM " + RECIPE_TABLE + " WHERE recipe_id = ?";

        try(Connection connection = DbConnection.getConnection()) {
            startTransaction(connection);

            try {
                Recipe recipe = null;

                try(PreparedStatement statement = connection.prepareStatement(sql)){
                    setParameter(statement, 1, recipeId, Integer.class);

                    try(ResultSet resultset = statement.executeQuery()){
                        if(resultset.next()) {
                            recipe = extract(resultset, Recipe.class);
                        }
                    }
                }

                if(Objects.nonNull(recipe)) {
                    recipe.getIngredients().addAll(fetchRecipeIngredients(connection, recipeId));

                    recipe.getSteps().addAll(fetchRecipeSteps(connection, recipeId));
                    recipe.getCategories().addAll(fetchRecipeCategories(connection, recipeId));
                }
                //first missing line of code
                commitTransaction(connection);
                return Optional.ofNullable(recipe);
            }
            catch(Exception e) {
                rollbackTransaction(connection);
                throw new DbException(e);
            }

        } catch (SQLException e) {
            throw new DbException(e);
        }
    }

    private List<Category> fetchRecipeCategories(Connection connection, Integer recipeId) throws  SQLException{
        String sql = ""
                + "SELECT c.* "
                + "FROM " + RECIPE_CATEGORY_TABLE + " rc "
                + "JOIN " + CATEGORY_TABLE + " c USING (category_id) "
                + "WHERE rc.recipe_id = ? "
                + "ORDER BY c.category_name";

        try(PreparedStatement statement = connection.prepareStatement(sql)){
            setParameter(statement, 1, recipeId, Integer.class);

            try(ResultSet resultSet = statement.executeQuery()){
                List<Category> categories = new LinkedList<Category>();

                while (resultSet.next()) {
                    categories.add(extract(resultSet, Category.class));
                }

                return categories;
            }
        }
    }

    private List<Step> fetchRecipeSteps(Connection connection, Integer recipeId) throws  SQLException{
        String sql = "SELECT * FROM " + STEP_TABLE + " s WHERE s.recipe_id = ?";

        try(PreparedStatement statement = connection.prepareStatement(sql)){
            setParameter(statement, 1, recipeId, Integer.class);

            try(ResultSet resultSet = statement.executeQuery()){
                List<Step> steps = new LinkedList <Step>();
                while(resultSet.next()) {
                    steps.add(extract(resultSet, Step.class));
                }

                return steps;
            }
        }
    }

    private List<Ingredient> fetchRecipeIngredients(Connection connection, Integer recipeId) throws SQLException {
        String sql = ""
                + "SELECT i.*, u.unit_name_singular, u.unit_name_plural "
                + "FROM " + INGREDIENT_TABLE + " i "
                + "LEFT JOIN " + UNIT_TABLE + " u USING (unit_id) "
                + "WHERE recipe_id = ? "
                + "ORDER BY i.ingredient_order";

        try(PreparedStatement statement = connection.prepareStatement(sql)){
            setParameter(statement, 1, recipeId, Integer.class);

            try(ResultSet resultset = statement.executeQuery()){
                List<Ingredient> ingredients = new LinkedList<Ingredient>();

                while(resultset.next()){
                    Ingredient ingredient = extract(resultset, Ingredient.class);
                    Unit unit = extract(resultset, Unit.class);

                    ingredient.setUnit(unit);
                    ingredients.add(ingredient);
                }

                return ingredients;
            }
        }
    }

    public Recipe insertRecipe(Recipe recipe) {
        String sql = ""
                + "INSERT INTO " + RECIPE_TABLE + " "
                + "(recipe_name , notes , num_servings, prep_time, cook_time) "
                + "VALUES "
                + "(?, ?, ?, ?, ?)";

        try (Connection connection = DbConnection.getConnection()) {
            startTransaction(connection);

            try (PreparedStatement statement = connection.prepareStatement(sql)) {
                setParameter(statement, 1, recipe.getRecipeName(), String.class);
                setParameter(statement, 2, recipe.getNotes(), String.class);
                setParameter(statement, 3, recipe.getNumServings(), Integer.class);
                setParameter(statement, 4, recipe.getPrepTime(), LocalTime.class);
                setParameter(statement, 5, recipe.getCookTime(), LocalTime.class);

                statement.executeUpdate();
                Integer recipeId = getLastInsertId(connection, RECIPE_TABLE);

                commitTransaction(connection);

                recipe.setRecipeId(recipeId);
                return recipe;

            } catch (Exception e) {
                rollbackTransaction(connection);
                throw new DbException(e);
            }
        } catch (SQLException e) {
            throw new DbException(e);
        }
    }

    public void executeBatch(List<String> sqlBatch) {
        try (Connection connection = DbConnection.getConnection()) {
            startTransaction(connection);

            try (Statement statement = connection.createStatement()) {
                for (String sql : sqlBatch) {
                    statement.addBatch(sql);
                }

                statement.executeBatch();
                commitTransaction(connection);

            } catch (Exception e) {
                rollbackTransaction(connection);
                throw new DbException(e);
            }
        } catch (SQLException e) {
            throw new DbException(e);
        }
    }


    public List<Unit> fetchAllUnits(){
        String sql = "SELECT * FROM " + UNIT_TABLE + " ORDER BY unit_name_singular";

        try(Connection connection = DbConnection.getConnection()) {
            startTransaction(connection);

            try(PreparedStatement statement = connection.prepareStatement(sql)){
                try(ResultSet resultSet = statement.executeQuery()){
                    List<Unit> units = new LinkedList<>();

                    while(resultSet.next()) {
                        units.add(extract(resultSet, Unit.class));
                    }

                    return units;
                }
            }
            catch(Exception e) {
                rollbackTransaction(connection);
                throw new DbException(e);
            }
        }catch(SQLException e) {
        throw new DbException(e);
        }
    }


    public void addIngredientToRecipe(Ingredient ingredient) {
        String sql = "INSERT INTO " + INGREDIENT_TABLE
                + " (recipe_id, unit_id, ingredient_name, instruction, ingredient_order, amount) "
                + " VALUES (?, ?, ?, ?, ?, ?)";

        try(Connection connection = DbConnection.getConnection()){
            startTransaction(connection);

            try {
                Integer order = getNextSequenceNumber(connection, ingredient.getRecipeId(), INGREDIENT_TABLE, "reci
                try(PreparedStatement statement = connection.prepareStatement(sql)){
                    setParameter(statement, 1, ingredient.getRecipeId(), Integer.class);
                    setParameter(statement, 2, ingredient.getUnit().getUnitId(), Integer.class);
                    setParameter(statement, 3, ingredient.getIngredientName(), String.class);
                    setParameter(statement, 4, ingredient.getInstruction(), String.class);
                    setParameter(statement, 5, order, Integer.class);
                    setParameter(statement, 6, ingredient.getAmount(), BigDecimal.class);

                    statement.executeUpdate();
                    commitTransaction(connection);

                }
            }catch(Exception e) {
                rollbackTransaction(connection);
                throw new DbException(e);
            }
            }catch (SQLException e) {
            throw new DbException(e);
        }

    }


    public void addStepToRecipe(Step step) {
```

```java
        String sql = "INSERT INTO " + STEP_TABLE + " (recipe_id, step_order, step_text)" + " VALUES (?, ?, ?)";

        try (Connection connection = DbConnection.getConnection()) {
            startTransaction(connection);

            Integer order = getNextSequenceNumber(connection, step.getRecipeId(), STEP_TABLE, "recipe_id");

            try (PreparedStatement statement = connection.prepareStatement(sql)) {
                setParameter(statement, 1, step.getRecipeId(), Integer.class);
                setParameter(statement, 2, order, Integer.class);
                setParameter(statement, 3, step.getStepText(), String.class);

                statement.executeUpdate();
                commitTransaction(connection);
            } catch (Exception e) {
                rollbackTransaction(connection);
                throw new DbException(e);
            }
        } catch (SQLException e) {
            throw new DbException(e);
        }
    }


    public List<Category> fetchAllCategories() {
        String sql = "SELECT * FROM " + CATEGORY_TABLE + " ORDER BY category_name";
        try(Connection connection = DbConnection.getConnection()){
            startTransaction(connection);

            try(PreparedStatement statement = connection.prepareStatement(sql)){
                try(ResultSet resultSet = statement.executeQuery()){
                    List<Category> categories = new LinkedList<>();

                    while(resultSet.next()) {
                        categories.add(extract(resultSet, Category.class));
                    }
                    return categories;
                }
            }
            catch (Exception e) {
                rollbackTransaction(connection);
                throw new DbException(e);
            }
        }catch (SQLException e) {
            throw new DbException(e);
        }
    }


    public void addCategoryToRecipe(Integer recipeId, String category) {
        String subQuery = "(SELECT category_id FROM " + CATEGORY_TABLE + " WHERE category_name = ?)" ;

        String sql = "INSERT INTO " + RECIPE_CATEGORY_TABLE + " (recipe_id, category_id) VALUES (?, " + subQuery +

        try(Connection connection = DbConnection.getConnection()){
            startTransaction(connection);

            try (PreparedStatement statement = connection.prepareStatement(sql)){
                setParameter(statement, 1, recipeId, Integer.class);
                setParameter(statement, 2, category, String.class);

                statement.executeUpdate();
                commitTransaction(connection);

            } catch (Exception e) {
                rollbackTransaction(connection);
                throw new DbException(e);
            }

        }catch (SQLException e) {
            throw new DbException(e);
        }
    }


    public List<Step> fetchRecipeSteps(Integer recipeId) {
        try(Connection connection = DbConnection.getConnection()){
            startTransaction(connection);
            try {
                List<Step> steps = fetchRecipeSteps(connection, recipeId);
                commitTransaction(connection);

                return steps;

            } catch (Exception e) {
                rollbackTransaction(connection);
                throw new DbException();
            }
        }catch (SQLException e) {
            throw new DbException(e);
        }
    }


    public boolean modifyRecipeStep(Step step) {
        String sql = "UPDATE " + STEP_TABLE + " SET step_text = ? WHERE step_id = ? ";

        try (Connection connection = DbConnection.getConnection()){
            startTransaction(connection);

            try(PreparedStatement statement = connection.prepareStatement(sql)){
                setParameter(statement, 1, step.getStepText(), String.class);
                setParameter(statement, 2, step.getStepId(), Integer.class);

                boolean updated = statement.executeUpdate() == 1;
                commitTransaction(connection);

                return updated;

            }catch (Exception e) {
                rollbackTransaction(connection);
                throw new DbException(e);
            }

        } catch (Exception e) {
            throw new DbException(e);
        }
    }


    public boolean deleteRecipe(Integer recipeId) {
        String sql = "DELETE FROM " + RECIPE_TABLE + " WHERE recipe_id = ?";

        try(Connection connection = DbConnection.getConnection()){
            startTransaction(connection);
            try(PreparedStatement statement = connection.prepareStatement(sql)){
                setParameter(statement, 1, recipeId, Integer.class);

                boolean deleted = statement.executeUpdate() == 1;

                commitTransaction(connection);
                return deleted;

            } catch (Exception e) {
                rollbackTransaction(connection);
                throw new DbException(e);
            }
        }catch (SQLException e) {
            throw new DbException(e);
        }
    }
}




//Category.java

package recipes.entity;

public class Category {
    private Integer categoryId;
    private String categoryName;
    public String getCategoryName() {
        return categoryName;
    }
    public void setCategoryName(String categoryName) {
        this.categoryName = categoryName;
    }
    public Integer getCategoryId() {
        return categoryId;
    }
    public void setCategoryId(Integer categoryId) {
        this.categoryId = categoryId;
    }
    @Override
    public String toString() {
        return "ID = " + categoryId + ", categoryName = " + categoryName;
    }

}
```

```java
//Ingredient.java

package recipes.entity;

import java.math.BigDecimal;
import java.util.Objects;

import provided.entity.EntityBase;

public class Ingredient extends EntityBase{
    private Integer ingredientId;
    private Integer recipeId;
    private Unit unit;
    private String ingredientName;
    private String instruction;
    private Integer ingredientOrder;
    private BigDecimal amount;

    @Override
    public String toString() {
        StringBuilder b = new StringBuilder();

        b.append("ID = ").append(ingredientId).append(": ");
        b.append(toFraction(amount));

        if(Objects.nonNull(unit) && Objects.nonNull(unit.getUnitId())) {
            String singular = unit.getUnitNameSingular();
            String plural = unit.getUnitNamePlural();
            String word = amount.compareTo(BigDecimal.ONE) > 0 ? plural : singular;

            b.append(word).append(" ");
        }
        b.append(ingredientName);
        if(Objects.nonNull(instruction)) {
            b.append(", ").append(instruction);
        }
        return b.toString();
    }
    public Integer getIngregredient_id() {
        return ingredientId;
    }
    public void setIngregredient_id(Integer ingregredient_id) {
        this.ingredientId = ingregredient_id;
    }
    public Integer getRecipeId() {
        return recipeId;
    }
    public void setRecipeId(Integer recipeId) {
        this.recipeId = recipeId;
    }
    public Unit getUnit() {
        return unit;
    }
    public void setUnit(Unit unit) {
        this.unit = unit;
    }
    public String getIngredientName() {
        return ingredientName;
    }
    public void setIngredientName(String ingredientName) {
        this.ingredientName = ingredientName;
    }
    public String getInstruction() {
        return instruction;
    }
    public void setInstruction(String instruction) {
        this.instruction = instruction;
    }
    public Integer getIngredientOrder() {
        return ingredientOrder;
    }
    public void setIngredientOrder(Integer ingredientOrder) {
        this.ingredientOrder = ingredientOrder;
    }
    public BigDecimal getAmount() {
        return amount;
    }
    public void setAmount(BigDecimal amount) {
        this.amount = amount;
    }

}
```

```java
//Recipe.java

package recipes.entity;

import java.time.*;
import java.time.format.DateTimeFormatter;
import java.util.*;

public class Recipe {
    private Integer recipeId;
    private String recipeName;
    private String notes;
    private Integer numServings;
    private LocalTime prepTime;
    private LocalTime cookTime;
    private LocalDateTime createdAt;

    private List<Ingredient> ingredients= new LinkedList<>();
    private List<Step> steps = new LinkedList<>();
    private List<Category> categories = new LinkedList<>();

    @Override
    public String toString() {
        DateTimeFormatter fmt = DateTimeFormatter.ofPattern("dd-MMM-yyyy HH:mm");
        String createTime = Objects.nonNull(createdAt) ? fmt.format(createdAt) : "(null)";
        String recipe = " ";

        recipe += "\n ID = " + recipeId;
        recipe += "\n Recipe Name = " + recipeName;
        recipe += "\n Notes = " + notes;
        recipe += "\n Number of Servings = " + numServings;
        recipe += "\n Prep Time = " + prepTime;
        recipe += "\n Cook Time = " + cookTime;
        recipe += "\n Created At = " + createTime;

        recipe += "\n Ingredients: ";

        for (Ingredient ingredient : ingredients) {
            recipe += "\n       " + ingredient;
        }

        recipe += "\n   Steps";

        for (Step step : steps) {
            recipe += "\n       " + step;
        }
        recipe += "\n   Categories";

        for(Category category : categories) {
            recipe+= "\n        "+ category;
        }

        return recipe;
    }

    public Integer recipeId() {
        return recipeId;
    }
    public Integer getRecipeId() {
        return recipeId;
    }

    public void setRecipeId(Integer recipeId) {
        this.recipeId = recipeId;
    }

    public void recipeId(Integer recipe_id) {
        this.recipeId = recipe_id;
    }
    public String getRecipeName() {
        return recipeName;
    }
    public void setRecipeName(String recipeName) {
        this.recipeName = recipeName;
    }
    public String getNotes() {
        return notes;
    }
    public void setNotes(String notes) {
        this.notes = notes;
    }
    public Integer getNumServings() {
        return numServings;
    }
    public void setNumServings(Integer numServings) {
        this.numServings = numServings;
    }
    public LocalTime getPrepTime() {
        return prepTime;
    }
    public void setPrepTime(LocalTime prepTime) {
        this.prepTime = prepTime;
    }
    public LocalTime getCookTime() {
        return cookTime;
    }
    public void setCookTime(LocalTime cookTime) {
        this.cookTime = cookTime;
    }
    public LocalDateTime getCreatedAt() {
        return createdAt;
    }
    public void setCreatedAt(LocalDateTime createdAt) {
        this.createdAt = createdAt;
    }
    public List<Ingredient> getIngredients() {
        return ingredients;
    }

    public List<Step> getSteps() {
        return steps;
    }
    public List<Category> getCategories() {
        return categories;
    }


}
```

```java
//Step.java

package recipes.entity;

public class Step {
    private Integer stepId;
    private Integer recipeId;
    private Integer stepOrder;
    private String stepText;

    @Override
    public String toString() {
        return "ID = " + stepId + ", stepText = " + stepText;
    }

    public Integer getStepId() {
        return stepId;
    }

    public void setStepId(Integer stepId) {
        this.stepId = stepId;
    }

    public Integer getRecipeId() {
        return recipeId;
    }

    public void setRecipeId(Integer recipeId) {
        this.recipeId = recipeId;
    }

    public Integer getStepOrder() {
        return stepOrder;
    }

    public void setStepOrder(Integer stepOrder) {
        this.stepOrder = stepOrder;
    }

    public String getStepText() {
        return stepText;
    }

    public void setStepText(String stepText) {
        this.stepText = stepText;
    }
}



//Unit.java

package recipes.entity;

public class Unit {
    public Integer getUnitId() {
        return unitId;
    }
    public void setUnitId(Integer unitId) {
        this.unitId = unitId;
    }
    public String getUnitNameSingular() {
        return unitNameSingular;
    }
    public void setUnitNameSingular(String unitNameSingular) {
        this.unitNameSingular = unitNameSingular;
    }
    public String getUnitNamePlural() {
        return unitNamePlural;
    }
    public void setUnitNamePlural(String unitNamePlural) {
        this.unitNamePlural = unitNamePlural;
    }
    private Integer unitId;
    private String unitNameSingular;
    private String unitNamePlural;
    @Override
    public String toString() {
        return "unit [unitId=" + unitId + ", unitNameSingular=" + unitNameSingular + ", unitNamePlural="
                + unitNamePlural + "]";
    }

}



//DbException
package recipes.exception;

@SuppressWarnings("serial")
public class DbException extends RuntimeException {

    public DbException() {
    }

    public DbException(String message) {
        super(message);
    }

    public DbException(Throwable cause) {
        super(cause);
    }

    public DbException(String message, Throwable cause) {
        super(message, cause);
    }

}




//RecipeService.java

package recipes.service;

import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.*;
import java.util.stream.Collectors;
import recipes.exception.DbException;
import recipes.dao.*;
import recipes.entity.Category;
import recipes.entity.Ingredient;
import recipes.entity.Recipe;
import recipes.entity.Step;
import recipes.entity.Unit;

public class RecipeService {
    private static final String SCHEMA_FILE = "recipe_schema.sql";
    private static final String DATA_FILE = "recipe_data.sql";

    private RecipeDao recipeDao = new RecipeDao();

    public Recipe fetchRecipeById(Integer recipeId) {
        return recipeDao.fetchRecipeById(recipeId)
                .orElseThrow(() -> new NoSuchElementException("Recipe with ID= " + recipeId + "does not exist"));

    }

    public void createAndPopulateTables() {
        loadFromFile(SCHEMA_FILE);
        loadFromFile(DATA_FILE);
    }

    private void loadFromFile(String fileName) {
        String content = readFileContent(fileName);
        List<String> sqlStatements = convertContentToSqlStatements(content);

        recipeDao.executeBatch(sqlStatements);
    }

    private List<String> convertContentToSqlStatements(String content) {
        content = removeComments(content);
        content = replaceWhiteSpaceSequencesWithSingleSpace(content);

        return extractLinesFromContent(content);
    }

    private List<String> extractLinesFromContent(String content) {
        List<String> lines = new LinkedList<>();

        while (!content.isEmpty()) {
            int semicolon = content.indexOf(";");

            if (semicolon == -1) {
                if (!content.isBlank()) {
                    lines.add(content);
                }
                content = "";
            } else {
                lines.add(content.substring(0, semicolon).trim());
                content = content.substring(semicolon + 1);
            }
        }

        return lines;
    }

    private String replaceWhiteSpaceSequencesWithSingleSpace(String content) {
        return content.replaceAll("\\s+", " ");
    }

    private String removeComments(String content) {
        StringBuilder builder = new StringBuilder(content);
        int commentPos = 0;

        while ((commentPos = builder.indexOf("-- ", commentPos)) != -1) {
            int eolPos = builder.indexOf("\n", commentPos + 1);

            if (eolPos == -1) {
                builder.replace(commentPos, builder.length(), "");
            } else {
                builder.replace(commentPos, eolPos + 1, "");
            }
        }
        return builder.toString();
    }

    private String readFileContent(String fileName) {
        try {
            Path path = Paths.get(getClass().getClassLoader().getResource(fileName).toURI());
            return Files.readString(path);
        } catch (Exception e) {
            throw new DbException(e);
        }
    }

    public Recipe addRecipe(Recipe recipe) {
        return recipeDao.insertRecipe(recipe);

    }

    public List<Recipe> fetchRecipes() {
        return recipeDao.fetchAllRecipes().stream().sorted((r1, r2) -> r1.getRecipeId() - r2.getRecipeId())
                .collect(Collectors.toList());
    }

    public List<Unit> fetchUnits() {
        return recipeDao.fetchAllUnits();
    }

    public void addIngrient(Ingredient ingredient) {
        recipeDao.addIngredientToRecipe(ingredient);

    }

    public void addStep(Step step) {
        recipeDao.addStepToRecipe(step);
    }

    public List<Category> fetchCategories() {
        return recipeDao.fetchAllCategories();
    }

    public void addCategoryToRecipe(Integer recipeId, String category) {
        recipeDao.addCategoryToRecipe(recipeId, category);

    }

    public List<Step> fetchSteps(Integer recipeId) {
        return recipeDao.fetchRecipeSteps(recipeId);
    }

    public void modifyStep(Step step) {
        if(!recipeDao.modifyRecipeStep(step)) {
            throw new DbException("Step with ID= "+ step.getStepId() + " does not exist.");
        }
    }

    public void deleteRecipe(Integer recipeId) {
        if(!recipeDao.deleteRecipe(recipeId)) {
            throw new DbException("Recipe with ID = " + recipeId + " does not exist.");
        }
    }


}




//REFERENCES
//YOUTUBE
//>>>>https://youtu.be/rcuK02JZFmU
//GITHUB
//>>>>https://github.com/fmd5045/Week07-11SQLRecipe
```