Домашнее задание: MiniBank + Hibernate

Цель

Адаптировать консольное приложение MiniBank, написанное на Spring Core, для работы с PostgreSQL через Hibernate.

Результат выполнения:

- данные пользователей и счетов сохраняются в БД;
- после перезапуска приложения данные остаются доступными;
- основные банковские операции работают через Hibernate `SessionFactory` в транзакциях;
- код остается в рамках Spring Core без Spring Boot и автоконфигураций.

Ожидаемый результат

После выполнения задания студент должен уметь:

- разметить POJO-классы как Hibernate-сущности;
- настроить `SessionFactory` вручную в `@Configuration` классе;
- выполнять CRUD/бизнес-операции через `Session` и транзакции;
- реализовать поведение nested-транзакций (join в уже открытую);
- отлаживать SQL Hibernate через `show_sql` / `format_sql`.

Предварительные знания

- Spring Core: `@Component`, `@Configuration`, `@PropertySource`, DI через конструкторы.
- Java Core: коллекции, исключения, работа с консолью (`Scanner`).
- SQL basics: `SELECT`, `INSERT`, `UPDATE`, `JOIN`, `WHERE`.
- Hibernate basics: `@Entity`, `@Table`, `@Id`, `@OneToMany`, `@ManyToOne`, `SessionFactory`.

Ограничения

- Только Spring Core + Hibernate, без Spring Boot.
- Без автоконфигураций и `spring-boot-starter-*`.
- Подключение и Hibernate-настройки задаются вручную.
- Консольный интерфейс сохраняется.

Разрешённые зависимости

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>6.1.5</version>
    </dependency>
    <dependency>
        <groupId>jakarta.annotation</groupId>
        <artifactId>jakarta.annotation-api</artifactId>
        <version>2.1.1</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate.orm</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>6.4.4.Final</version>
    </dependency>
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <version>42.7.3</version>
    </dependency>
</dependencies>
```

Что нужно реализовать

Краткий план выполнения

1. Разметить `User` и `Account` как Hibernate-сущности и настроить связи.
2. Поднять `SessionFactory` вручную и подключить сущности.
3. Переписать сервисы на работу через `Session` и транзакции.

4. Реализовать корректную транзакционную обвязку с поддержкой nested-вызовов.
5. Прогнать сценарии проверок и убедиться, что данные сохраняются в БД.

1) Перевести модели на Hibernate Entities

Перевести `User` и `Account` с in-memory модели на ORM-сущности.

Требования к маппингу:

- `User`: таблица `users`, первичный ключ с авто-генерацией.
- `Account`: таблица `accounts`, первичный ключ с авто-генерацией.
- Связь: `User (1) -> (N) Account` через `@OneToMany` и `@ManyToOne`.
- У аккаунта должен быть owner (`user_id`).

2) Настроить Hibernate вручную

Создать `HibernateConfiguration` с `@Bean SessionFactory`.

Минимально необходимые свойства:

- `hibernate.connection.driver_class`
- `hibernate.connection.url`
- `hibernate.connection.username`
- `hibernate.connection.password`
- `hibernate.dialect`
- `hibernate.hbm2ddl.auto`
- `hibernate.current_session_context_class=thread`
- `hibernate.show_sql=true`
- `hibernate.format_sql=true`

Подключить сущности через `addAnnotatedClass(...)`.

3) Обновить сервисы под БД

Переписать `UserService` и `AccountService` так, чтобы операции выполнялись через Hibernate:

- создание пользователя;

- создание аккаунта;
- пополнение;
- снятие;
- перевод;
- закрытие счета;
- выборка пользователя(ей).

4) Транзакционное поведение

Все операции с изменением данных должны выполняться в транзакции.

Обязательное поведение nested-вызовов:

- если метод вызван внутри уже активной транзакции, он должен присоединиться к ней;
- такой метод не должен самостоятельно делать commit/rollback/close;
- commit/rollback/close делает только тот уровень, который открыл транзакцию.

5) Консольный контракт

Команды MiniBank должны остаться рабочими:

- `USER_CREATE`
- `SHOW_ALL_USERS`
- `ACCOUNT_CREATE`
- `ACCOUNT_DEPOSIT`
- `ACCOUNT_WITHDRAW`
- `ACCOUNT_TRANSFER`
- `ACCOUNT_CLOSE`
- `EXIT`

Бизнес-правила:

- при создании пользователя создаётся дефолтный счёт;
- комиссия перевода применяется только между разными пользователями;
- перевод между своими счетами без комиссии;
- единственный счёт закрыть нельзя;

- суммы операций и ID должны быть > 0;
- сообщения об ошибках должны быть понятными (`Error: ...`).

Настройки приложения

Пример `application.properties`:

```
account.default-amount=500
account.transfer-commission=0.02

db.driver=org.postgresql.Driver
db.url=jdbc:postgresql://localhost:5432/bank
db.username=postgres
db.password=root
db.dialect=org.hibernate.dialect.PostgreSQLDialect

hibernate.hbm2ddl.auto=update
hibernate.show_sql=true
hibernate.format_sql=true
```

Запуск PostgreSQL через Docker

Вариант 1: одной командой

```
docker run --name minibank-postgres \
  -e POSTGRES_DB=bank \
  -e POSTGRES_USER=postgres \
  -e POSTGRES_PASSWORD=root \
  -p 5432:5432 \
  -d postgres:16
```

Вариант 2: через `docker-compose.yml`

```
services:
  postgres:
    image: postgres:16
    container_name: minibank-postgres
    environment:
```

```
      POSTGRES_DB: bank
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: root
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

Проверка подключения:

- URL: `jdbc:postgresql://localhost:5432/bank`
- User: `postgres`
- Password: `root`

Что сдавать

- Исходный код MiniBank с Hibernate-интеграцией.
- Рабочую конфигурацию подключения к PostgreSQL.
- Реализованную транзакционную обвязку с поддержкой nested-вызовов.
- Краткий отчёт/README с инструкцией запуска и проверки.

Критерии приёмки

- Приложение стартует и выполняет все команды без падения.
- Пользователь и счета сохраняются в БД.
- После перезапуска данные не теряются.
- Переводы считают комиссию корректно.
- При исключении внутри транзакции изменения откатываются.
- SQL Hibernate виден в логах.
- Нет Spring Boot/автоконфигураций.

Чеклист самопроверки

1. Создать 2 пользователей и убедиться, что у каждого дефолтный счет.
2. Создать доп. счет первому пользователю.

3. Сделать перевод между своими счетами и проверить отсутствие комиссии.
4. Сделать перевод между разными пользователями и проверить комиссию.
5. Проверить ошибки для отрицательных/нулевых сумм и несуществующих ID.
6. Проверить запрет закрытия единственного счета.
7. Перезапустить приложение и убедиться, что данные сохранились.
8. В `ACCOUNT_TRANSFER` временно бросить исключение в конце метода и убедиться, что перевод откатился.

Частые ошибки и анти-паттерны

- Выполнять `save/update/delete` без открытой транзакции — изменения не фиксируются в БД.
- Делать несколько связанных операций в одном методе, но не оборачивать их в одну транзакцию — получаются частичные изменения.
- Внутри активной транзакции пытаться открыть новую вместо присоединения к текущей — ломается целостность и может происходить двойной `commit/rollback`.
- Не включать привязку сессии к потоку (`hibernate.current_session_context_class=thread`) и не проверять статус транзакции — nested-вызовы не работают корректно.