Подсказки к домашнему заданию MiniBank + Hibernate

1) Как правильно связать `User` и `Account`

Минимальная схема:

- `users` и `accounts` в отдельных таблицах;
- `accounts.user_id` хранит владельца счета;
- `User` содержит список счетов;
- `Account` содержит ссылку на `User`.

Пример направления маппинга:

```
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @OneToMany(mappedBy = "user", fetch = FetchType.EAGER)
    private List<Account> accountList;
}

@Entity
@Table(name = "accounts")
public class Account {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id")
    private User user;
}
```

2) Как настроить `SessionFactory` вручную

Используйте `@Configuration` + `@Bean` и задайте свойства через `setProperty`.

Полный пример конфигурации:

```
@Configuration
public class HibernateConfiguration {

    @Bean
    public SessionFactory sessionFactory() {
        org.hibernate.cfg.Configuration configuration = new
org.hibernate.cfg.Configuration();

        configuration
                .addAnnotatedClass(User.class)
                .addAnnotatedClass(Account.class)
                .setProperty("hibernate.dialect",
"org.hibernate.dialect.PostgreSQLDialect")
                .setProperty("hibernate.connection.driver_class",
"org.postgresql.Driver")
                .setProperty("hibernate.connection.url",
"jdbc:postgresql://localhost:5432/bank")
                .setProperty("hibernate.connection.username",
"postgres")
                .setProperty("hibernate.connection.password", "root")
                .setProperty("hibernate.show_sql", "true")
                .setProperty("hibernate.format_sql", "true")

.setProperty("hibernate.current_session_context_class", "thread")
                .setProperty("hibernate.hbm2ddl.auto", "update");

        ServiceRegistry serviceRegistry = new
StandardServiceRegistryBuilder()
                .applySettings(configuration.getProperties())
                .build();

        return configuration.buildSessionFactory(serviceRegistry);
    }
}
```

Важно:

- включите `hibernate.current_session_context_class=thread`;

- включите `show_sql` и `format_sql` для отладки;
- добавьте все entity через `addAnnotatedClass(...)`.

3) Как сделать транзакционную обвязку с поддержкой nested-вызовов

Зачем это нужно:

- один метод может вызывать другой, и оба работают с БД;
- если каждый метод всегда открывает и закрывает транзакцию, возникают двойные `commit/rollback`;
- при ошибке изменения могут частично сохраниться.

Решение:

- если транзакция уже активна, просто присоединиться к ней;
- commit/rollback/close выполняет только тот метод, который открыл транзакцию.

Рекомендуемый шаблон:

- получить `Session` через `getCurrentSession()`;
- проверить статус текущей транзакции;
- если транзакция не активна, открыть новую и запомнить себя как owner;
- выполнить действие;
- commit/rollback/close делать только если вы owner.

Каркас:

```
private <T> T executeInTransactionOrJoin(Supplier<T> action) {
    Session session = sessionFactory.getCurrentSession();
    Transaction tx = session.getTransaction();
    boolean owner = tx.getStatus() == TransactionStatus.NOT_ACTIVE;

    if (owner) {
        tx = session.beginTransaction();
    }

    try {
        T result = action.get();
```

```
            if (owner) {
                tx.commit();
            }
            return result;
        } catch (RuntimeException e) {
            if (owner) {
                tx.rollback();
            }
            throw e;
        } finally {
            if (owner) {
                session.close();
            }
        }
    }
```

4) Как избежать проблемы detached-сущностей

Проблема часто появляется так:

- в одном сервисе загрузили `User`;
- транзакцию закрыли;
- этот объект передали в другой сервис для `persist` связанной сущности.

Безопасный подход:

- принимать в командном слое ID (`userId`, `accountId`), а не entity;
- загружать owner внутри той же транзакции, где выполняется операция записи.

5) Валидации, которые лучше сделать заранее

Проверяйте сразу в командах и/или сервисах:

- `login` не пустой;
- `id > 0`;
- `amount > 0`;
- source и target счета в переводе не совпадают;
- при снятии/переводе достаточно средств.

Пример сообщения:

```
Error: insufficient funds on account id=1, moneyAmount=0, attempted
withdraw=100
```

6) Комиссия в переводах

Логика:

- если счета одного пользователя, комиссия не применяется;
- если счета разных пользователей, получатель получает `amount * (1 - commission)`.

Подсказка: сравнивайте владельцев по `userId`, а не через `==` ссылок на объекты.

7) Проверка rollback на переводе

Сценарий проверки:

1. В конце метода `transfer` временно бросьте исключение после изменения балансов.
2. Выполните перевод.
3. Убедитесь в БД, что изменения не сохранились.
4. Удалите искусственное исключение после проверки.

Это проверяет, что транзакция откатывается полностью.

8) Что логировать во время отладки

Минимум полезного для диагностики:

- команда и входные параметры;
- понятные `Error: ...` сообщения для пользователя;
- SQL Hibernate (`show_sql`, `format_sql`) для проверки, что ORM работает ожидаемо.

9) Если что-то «работает не так»

Быстрый чек:

1. База поднята и доступна по URL из properties.

2. Все сущности добавлены в Hibernate configuration.
3. Все операции записи внутри транзакции.
4. Нет двойного commit/rollback в nested-вызовах.
5. Ошибки не скрываются stack trace в консоли, а выводятся понятно.