Assignment-9-PBLJ

Easy Level: Spring DI with Java Config

1. Configuration Class

```
// AppConfig.java
@Configuration
public class AppConfig {
    @Bean
    public Course course() {
        return new Course("Java Programming", "8 weeks");
    }

    @Bean
    public Student student(Course course) {
        Student student = new Student();
        student.setName("Alice");
        student.setCourse(course);
        return student;
    }
}
```

2. Main Application

```
// Main.java
public static void main(String[] args) {
    ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);
    Student student = context.getBean(Student.class);
    System.out.println(student); // Outputs student details with course info
}
```

Implements Spring DI using Java config as shown in[1]

Medium Level: Hibernate CRUD Operations

1. Hibernate Configuration

```
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/student_db</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">root</property>
```

```
<mapping class="com.example.Student"/>
```

2. CRUD Operations

```
// StudentDao.java
public class StudentDao {
    public void saveStudent(Student student) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        Transaction tx = null;
        try {
            tx = session.beginTransaction();
            session.save(student);
            tx.commit();
        } catch (Exception e) {
            if (tx != null) tx.rollback();
            e.printStackTrace();
        } finally {
            session.close();
        }
    }
    // Implement other CRUD methods similarly
}
```

Hard Level: Transaction Management

1. Service Layer with Transactions

```
// BankServiceImpl.java
@Service
@Transactional
public class BankServiceImpl implements BankService {
    @Autowired
    private AccountDao accountDao;

    public void transferFunds(Long fromId, Long toId, BigDecimal amount) {
        Account fromAccount = accountDao.findById(fromId);
        Account toAccount = accountDao.findById(toId);

        if(fromAccount.getBalance().compareTo(amount) < 0) {
            throw new InsufficientFundsException("Not enough balance");
        }

        fromAccount.setBalance(fromAccount.getBalance().subtract(amount));
        toAccount.setBalance(toAccount.getBalance().add(amount));

        accountDao.update(fromAccount);
        accountDao.update(toAccount);
    }
}
```

2. Transaction Rollback Configuration

```
@Configuration
@EnableTransactionManagement
public class PersistenceConfig {
    @Bean
    public PlatformTransactionManager transactionManager() {
        HibernateTransactionManager txManager = new HibernateTransactionManager();
        txManager.setSessionFactory(sessionFactory().getObject());
        return txManager;
    }

    @Bean
    public LocalSessionFactoryBean sessionFactory() {
        LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();
        sessionFactory.setDataSource(dataSource());
        sessionFactory.setPackagesToScan("com.example.model");
        sessionFactory.setHibernateProperties(hibernateProperties());
        return sessionFactory;
    }
}
```

Key Implementation Details

Transaction Rollback Handling

- **Unchecked exceptions** automatically trigger rollback[4]
- For checked exceptions, specify explicitly:

  ```
  @Transactional(rollbackFor = InsufficientFundsException.class)
  ```

- Manual rollback using `TransactionAspectSupport.currentTransactionStatus().setRollbackOnly()`

Best Practices

1. Always use `try-with-resources` with Hibernate Sessions[2]
2. Separate business logic (Service) from data access (DAO) layers
3. Use Hibernate's `@Version` for optimistic locking
4. Configure connection pooling (HikariCP recommended)

Testing Transactions

```
@SpringBootTest
public class BankServiceTest {
    @Autowired
    private BankService bankService;

    @Test
    void testTransferRollback() {
        assertThrows(InsufficientFundsException.class, () -> {
            bankService.transferFunds(1L, 2L, new BigDecimal("1000"));
        });

        // Verify balances remain unchanged
    }
}
```