

ПРОГРАММА

ПРОФЕССИОНАЛЬНОЙ ПЕРЕПОДГОТОВКИ

**“Python-разработка для БПЛА: паттерны проектирования, API-интерфейсы и библиотеки для оптимизации решений”**

**Задание 7.**

**Паттерны проектирования для работы с базами данных**

Преподаватель курса

**Максимов Егор Васильевич**

Выполнил

**Зубков Евгений Юрьевич**

**№ потока БПЛА\_256-1, группа 1**

**Москва 2024 г.**

**Задание 7. Паттерны проектирования для работы с базами данных**

**Инструкция:**

Шаг 1. Реализовать паттерн Абстрактная фабрика для работы с БД. Создать интерфейс базового класса DBFactory и классы MySQLFactory, PostgreSQLFactory и др.

Шаг 2. Реализовать паттерн Строитель для создания запросов. Создать класс QueryBuilder с методами select(), where(), order\_by() и др.

Шаг 3. Реализовать Отображение объектно-реляционное. Создать класс User, класс UserMapper для преобразования User в строку SQL и обратно.

Шаг 4. Реализовать паттерн Хранитель для управления соединениями с БД. Класс DBConnectionManager будет отвечать за установку, разрыв соединений.

Шаг 4. Протестировать реализованные паттерны. Создать подключения через фабрику. Построить запрос со строителем. Получить данные, преобразовать их в объекты с отображением ОР.

**Решение:**

*"""  
Домашнее задание 7  
Шаг 1. Реализовать паттерн Абстрактная фабрика для работы с БД. Создать интерфейс базового класса DBFactory и классы MySQLFactory, PostgreSQLFactory и др.  
Шаг 2. Реализовать паттерн Строитель для создания запросов. Создать класс QueryBuilder с методами select(), where(), order\_by() и др.  
Шаг 3. Реализовать Отображение объектно-реляционное. Создать класс User, класс UserMapper для преобразования User в строку SQL и обратно.  
Шаг 4. Реализовать паттерн Хранитель для управления соединениями с БД. Класс DBConnectionManager будет отвечать за установку, разрыв соединений.  
Шаг 4. Протестировать реализованные паттерны. Создать подключения через фабрику. Построить запрос со строителем. Получить данные, преобразовать их в объекты с отображением ОР.  
"""*# Шаг 1: Реализация паттерна Абстрактная фабрика  
from abc import ABC, abstractmethod  
  
  
class DBFactory(ABC):  
 @abstractmethod  
 def create\_connection(self):  
 pass  
  
  
  
class MySQLFactory(DBFactory):  
 def create\_connection(self):  
 return MySQLConnection()  
  
  
  
class MongoDBFactory(DBFactory):  
 def create\_connection(self):  
 return MongoDBConnection()  
  
  
  
# Классы соединений и строителей запросов  
class MySQLConnection:  
 def connect(self):  
 print("Connecting to MySQL database...")  
  
  
class MongoDBConnection:  
 def connect(self):  
 print("Connecting to MongoDB database...")  
  
  
# Шаг 2: Реализация паттерна Строитель для создания запросов  
class QueryBuilder:  
 def \_\_init\_\_(self):  
 self.query = ""  
 self.params = []  
  
 def select(self, fields):  
 self.query = f"SELECT {fields} "  
 return self  
  
 def where(self, condition):  
 self.query += f"WHERE {condition} "  
 return self  
  
 def order\_by(self, field, order='ASC'):  
 self.query += f"ORDER BY {field} {order} "  
 return self  
  
 def insert\_into(self, table, fields):  
 self.query = f"INSERT INTO {table} ({', '.join(fields)}) VALUES ({', '.join('?' for \_ in fields)}) "  
 return self  
  
 def values(self, \*values):  
 self.params.extend(values)  
 return self  
  
 def add\_params(self, \*params):  
 self.params.extend(params)  
 return self  
  
 def get\_params(self):  
 return self.params  
  
 def get\_query(self):  
 return self.query.strip()  
  
 def clear(self):  
 self.query = ""  
 self.params = []  
  
  
# Шаг 3: Реализация Отображение объектно-реляционное  
class User:  
 def \_\_init\_\_(self, user\_id, name, email):  
 self.user\_id = user\_id  
 self.name = name  
 self.email = email  
  
class UserMapper:  
 def to\_sql(self, user):  
 return f"INSERT INTO users (id, name, email) VALUES ({user.user\_id}, '{user.name}', '{user.email}')"  
  
 def from\_sql(self, record):  
 user\_id, name, email = record  
 return User(user\_id, name, email)  
  
  
# Шаг 4: Реализация паттерна Хранитель для управления соединениями с БД  
class DBConnectionManager:  
 def \_\_init\_\_(self):  
 self.connections = []  
  
 def add\_connection(self, connection):  
 self.connections.append(connection)  
 connection.connect()  
  
 def close\_all(self):  
 for connection in self.connections:  
 print("Closing connection...")  
 self.connections = []  
  
  
# Тестирование реализованных паттернов  
def main():  
 # Шаг 1: Подключения через фабрику  
 mysql\_factory = MySQLFactory()  
 mongo\_factory = MongoDBFactory()  
  
 mysql\_connection = mysql\_factory.create\_connection()  
 mongo\_connection = mongo\_factory.create\_connection()  
  
 connection\_manager = DBConnectionManager()  
 connection\_manager.add\_connection(mysql\_connection)  
 connection\_manager.add\_connection(mongo\_connection)  
  
 # Шаг 2: Строим запросы  
 query\_builder = QueryBuilder()  
 query = query\_builder.select("\*").where("id=1").order\_by("name").get\_query()  
 print("Built query:", query)  
 query = query\_builder.insert\_into("user", ['id', 'name', 'email']).values('1', 'Петр', 'petr@mail.ru').get\_query()  
 print("Built query:", query)  
 print(query\_builder.get\_params())  
 query\_builder.clear()  
  
  
 # Шаг 3: Работа с объектно-реляционным отображением  
 user = User(1, "Петр", "petr@mail.ru")  
 user\_mapper = UserMapper()  
 sql\_query = user\_mapper.to\_sql(user)  
 print("SQL query from user:", sql\_query)  
 user\_from\_sql = user\_mapper.from\_sql(("1", "Петр", "petr@mail.ru"))  
 print(f"id from SQL: {user\_from\_sql.user\_id}, User: {user\_from\_sql.name}, email: {user\_from\_sql.email}")  
  
 # Шаг 4: Завершение работы с соединениями  
 connection\_manager.close\_all()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()