Министерство Образования Республики Молдова

Технический Университет Молдовы

Департамент программной инженерии и автоматики

**Отчет**

Лабораторная работа №4

Тема: «Factory method»

Выполнил: TI-197-Герца Николай

Проверил: Булдумаг Олег

Кишинев-2021

**Суть паттерна Factory method:**

• Определяет интерфейс создания объекта, но позволяет субклассам выбрать класс создаваемого экземпляра. Таким образом, Фабричный метод делегирует операцию создания экземпляра субклассам.

• Идея в том, что мы создаем фабрику, которая может создавать нам объекты. И сразу же возникает вопрос, а почему мы не можем просто использовать оператор new, чтобы создавать объекты? Есть ситуации, когда мы хотим скрыть снаружи реализацию создания объекта и в этом случае нам поможет паттерн Factory.

• Основная цель и задача данного шаблона – это создание класса, который будет помогать нам создавать определенные объекты на основании входных данных. Для создания фабрики необходимо минимум 2 класса. Класс конструктор, который генерирует новый объект и класса, который вызывает это создание с определенными параметрами

**Что будет происходить:**

• В данном коде объект класса Employee создает 4 разных вида оплаты у сотрудников, зависящая от тарифа. У каждого тарифа есть своя почасовая оплата. Метод create() по сути и есть Factory method. Так же создаем 4 класса-конструктора, которые будут представлять собой возможные тарифы.

**Диаграмма UML:**

•

**Реализация:**

•

//Основная цель и задача данного шаблона – это создание класса, который будет помогать нам создавать определенные объекты на основании входных данных.

//Для создания фабрики необходимо минимум 2 класса

//Класс конструктор, который генерирует новый объект и класса, который вызывает это создание с определенными параметрами

class Economy {

  constructor() {

    this.rate = "200 лей";

  }

}

class Comfort {

  constructor() {

    this.rate = "300 лей";

  }

}

class Premium {

  constructor() {

    this.rate = "400 лей";

  }

}

class Business {

  constructor() {

    this.rate = "500 лей";

  }

}

class Employee {

  create(type) {

    let employee;

    if (type === "economy") {

      employee = new Economy();

    } else if (type === "comfort") {

      employee = new Comfort();

    } else if (type === "premium") {

      employee = new Premium();

    } else if (type === "business") {

      employee = new Business();

    }

    employee.type = type;

    employee.say = function () {

      console.log(

        `Работа в такси класса ${this.type} оплачивается ${this.rate}/час`

      );

    };

    return employee;

  }

}

const factory = new Employee();

economy = factory.create("economy");

comfort = factory.create("comfort");

premium = factory.create("premium");

business = factory.create("business");

economy.say(); //Работа в такси класса economy оплачивается 200 лей/час

comfort.say(); //Работа в такси класса comfort оплачивается 300 лей/час

premium.say(); //Работа в такси класса premium оплачивается 400 лей/час

business.say(); //Работа в такси класса business оплачивается 500 лей/час

**Применение:**

• Когда нужно создавать множество однотипных объектов (объекты с одинаковой структурой, но с разными данными)

• Когда заранее неизвестны типы и зависимости объектов, с которыми должен работать ваш код.

**Преимущества:**

• Фабрику стоит использовать, если в конструкторе у вас очень высокая сложность создания объекта, чаще всего, когда он может создаваться из нескольких источников;

• Подходит, когда вам нужно создавать много объектов с одинаковыми полями;

• Позволяет сделать код создания объектов более универсальным, не привязываясь к конкретным классам, а оперируя лишь общим интерфейсом;

• Позволяет установить связь между параллельными иерархиями классов;

• Реализует принцип открытости/закрытости.

**Недостатки:**

• Может создать дополнительную сложность в приложении там, где ее можно было бы избежать.

• Может привести к созданию больших параллельных иерархий классов, так как для каждого класса продукта надо создать свой подкласс создателя.