Министерство Образования Республики Молдова

Технический Университет Молдовы

Департамент программной инженерии и автоматики

**Отчет**

Лабораторная работа №2

Тема: «Prototype + builder»

Выполнил: TI-197-Герца Николай

Проверил: Булдумаг Олег

Кишинев-2021

**Суть паттерна Prototype:**

• Prototype - передает свойства для объектов, которые создаются, например, через new

• Свойство Prototype может расширять функционал класса

• Поле prototype является указателем на объект, у которого есть конструктор

**Что будет происходить:**

• Я создаю класс Employee, который будет принимать 5 аргументов: имя, фамилию, машину, опыт и зарплату. В этом классе будет метод getInfoAbout(), который при вызове будет показывать основную информацию про работника. Затем, через прототипирование, я расширяю данный класс новым методом getSalaryInfo(), который будет выводить зарплату определенного сотрудника. Затем мы создаем объект класса и вызываем два метода. Но когда мы вызываем сам объект, в нем нет данного метода.

**Диаграмма UML:**

•



**Реализация:**

class Employee {

  constructor(firstName, secondName, vehicle, experience, salary) {

    this.firstName = firstName;

    this.secondName = secondName;

    this.vehicle = vehicle;

    this.experience = experience;

    this.salary = salary;

  }

  getInfoAbout() {

    return `Сотрудник: ${this.firstName} ${this.secondName} имеет автомобиль ${this.vehicle} и стаж вождения ${this.experience} года.`;

  }

}

Employee.prototype.getSalaryInfo = function () {

  return `Сотрудник ${this.firstName} ${this.secondName} имеет зарпалту ${this.salary}`;

};

const employee = new Employee("Maxim", "Neescas", "BMW", "3", "6500");

console.log(employee.getInfoAbout()); //Сотрудник: Maxim Neescas имеет автомобиль BMW и стаж вождения 3 года.

console.log(employee.getSalaryInfo()); //Сотрудник Maxim Neescas имеет зарпалту 6500

console.log(employee); //Выводит без метода getSalaryInfo()

**Суть паттерна Builder:**

- Builder - это шаблон проектирования, обеспечивающий гибкое решение для создания объектов.

- Builder отделяет построение сложного объекта от его представления.

- Builder создает сложный объект, используя простые объекты, обеспечивая пошаговый подход. Он принадлежит к творческим образцам.

**Что будет происходить:**

• Мы создаем класс Client, у которого есть имя и фамилия. Затем создаем ClientBuilder, который будет генерировать объекты Client. Здесь мы задаем и возвращаем имя, фамилию и создание нового объекта класса. Затем уже создаем новый объект client и используем ClientBuilder для создание этого объекта.

**Диаграмма UML:**

•

**Реализация:**

class Client {

  constructor(firstName, secondName) {

    this.firstName = firstName;

    this.secondName = secondName;

  }

}

let ClientBuilder = function () {

  let firstName;

  let secondName;

  return {

    setFirstName: function (firstName) {

      this.firstName = firstName;

      return this;

    },

    setSecondName: function (secondName) {

      this.secondName = secondName;

      return this;

    },

    build: function () {

      return new Client(firstName, secondName);

    },

  };

};

let client = new ClientBuilder().setFirstName("Vlad").setSecondName("Vladov");

console.log(client);

**Применение:**

• Если множественное наследование, можно определить 1 метод для объекта и не держать его в классе;

• Когда клонирование объекта является более предпочтительным вариантом нежели его создание и инициализация с помощью конструктора;

• Когда необходимо обеспечить получение различных вариаций объекта в процессе его создания;

**Преимущества:**

• Ускоряет создание объектов;

• Альтернатива созданию подклассов для конструирования сложных объектов;

**Недостатки:**

• Сложно клонировать составные объекты, имеющие ссылки на другие объекты