Министерство Образования Республики Молдова

Технический Университет Молдовы

Департамент программной инженерии и автоматики

**Отчет**

Лабораторная работа №3

Тема: «Composite»

Выполнил: TI-197-Герца Николай

Проверил: Булдумаг Олег

Кишинев-2021

**Суть паттерна:**

- Компоновщик — это структурный паттерн проектирования, который позволяет сгруппировать множество объектов в древовидную структуру, а затем работать с ней так, как будто это единичный объект.

**Что будет происходить:**

- Ввожу класс Employees, который содержит основную информацию про работников. В нем 5 методов. Это - класс конструктор, который поможет создавать объекты с единым интерфейсом взаимодействия. Затем для каждого работника создам отдельный класс, то есть эти работники будут являются составляющими частями компаний такси. После этого создаю класс компоновщик, который будет считать количество работников и общую ЗП в компании. В конце создаю класс Yandex, который является какой-то компанией. Затем создаю экземпляр этой компании.

**Диаграмма UML:**

-



**Реализация:**

class Employees {

  setName(name) {

    this.name = name;

  }

  setCar(car) {

    this.car = car;

  }

  setSalary(salary) {

    this.salary = salary;

  }

  getSalary() {

    return this.salary;

  }

  getName() {

    return this.name;

  }

}

class Empl1 extends Employees {

  constructor() {

    super();

    this.setName("Maxim");

    this.setCar("BMW");

    this.setSalary(5000);

  }

}

class Empl2 extends Employees {

  constructor() {

    super();

    this.setName("Egor");

    this.setCar("AUDI");

    this.setSalary(4500);

  }

}

class Empl3 extends Employees {

  constructor() {

    super();

    this.setName("Ruslan");

    this.setCar("KIA");

    this.setSalary(3000);

  }

}

class Composite extends Employees {

  constructor() {

    super();

    this.employee = [];

  }

  add(employee) {

    this.employee.push(employee);

  }

  getSalary() {

    return this.employee.map((em) => em.getSalary()).reduce((a, b) => a + b);

  }

  getNumberOfEmployees() {

    return this.employee.length;

  }

}

class Companies extends Composite {

  constructor() {

    super();

    this.setName("Yandex");

  }

}

const yandex = new Companies();

yandex.add(new Empl1());

yandex.add(new Empl2());

yandex.add(new Empl3());

console.log(`

${yandex.getName()} has

${yandex.getNumberOfEmployees()} employees and total salary

${yandex.getSalary()}`);

**Применение:**

- когда вам нужно представить древовидную структуру объектов;

- когда клиенты должны единообразно трактовать простые и составные объекты

**Преимущества:**

- упрощает архитектуру клиента при работе со сложным деревом компонентов;

- облегчает добавление новых видов компонентов.

**Недостатки:**

- создаёт слишком общий дизайн классов.