Министерство Образования Республики Молдова

Технический Университет Молдовы

Департамент программной инженерии и автоматики

**Отчет**

Лабораторная работа №3

Тема: «Singleton»

Выполнил: TI-197-Герца Николай

Проверил: Булдумаг Олег

Кишинев-2021

**Суть паттерна Singleton:**

• Singleton - это паттерн проектирования который ограничивает создание некого класса, позволяя создавать только один экземпляр. Обычно он используется для управления неким глобальным состояние приложения. Singleton должен быть неизменяемым с точки зрения его использования, поэтому не должно быть никаких рисков при попытке создать несколько экземпляров подобного объекта.

**Что будет происходить:**

• Cоздадим глобальную переменную и будем обращаться к ней. Объявляем сущность instance и класс Employee. Создаем конструктор, который проверяет, существует ли instance. Если его нет, то он создает его и начинает на него ссылаться, если он есть, то конструктор получается ссылку на него. Теперь любой объект, созданный с помощью конструктора Employee, будет получать ссылку на instance, а не каждый раз создавать новый. В конструктор помещаем свойство count и два метода.

**Диаграмма UML:**

•

**Реализация:**

•

let instance;

class Employee {

  constructor() {

    if (!instance) {

      instance = this;

      instance.count = 10;

    }

    return instance;

  }

  getNumberOfDrivers() {

    return instance.count;

  }

  increaseNumberOfDrivers() {

    return instance.count++;

  }

}

const yandexDrivers = new Employee();

const uberDrivers = new Employee();

yandexDrivers.increaseNumberOfDrivers();

yandexDrivers.increaseNumberOfDrivers();

uberDrivers.increaseNumberOfDrivers();

uberDrivers.increaseNumberOfDrivers();

//Несмотря на то, что тут два разных объекта созданных при помощи функции конструктора, они все равно ссылаются на один объект singleton и изменяют его 4 раза

console.log(yandexDrivers.getNumberOfDrivers()); //14

console.log(uberDrivers.getNumberOfDrivers()); //14

**Применение:**

• Нужен тогда, когда в системе должен быть объект в едином экземпляре и к которому может быть доступ из разных частей программы. Если это интернет магазин, например, то это – корзина или карта клиента. Или, например, экземпляр плеера.

• Используется существующая объектно-ориентированная библиотека и ей нужен объект, унаследованный от определённого класса/интерфейса

**Преимущества:**

• При попытке создания данного объекта он создаётся только в том случае, если ещё не существует, в противном случае возвращается ссылка на уже существующий экземпляр и нового выделения памяти не происходит.

• Можно пользоваться именно экземпляром класса, так как при этом во многих случаях становится доступной более широкая функциональность. Например, к описанным компонентам класса можно обращаться через интерфейс

• Контролируемый доступ к единственному экземпляру

**Недостатки:**

• Один из самых главных минусов – это то, что он сильно затрудняет юнит-тестирование. Он привносит в программу глобальное состояние, поэтому вы не можете просто взять и изолировать классы, которые полагаются на Singleton.

• В объектно-ориентированном программировании существует правило хорошего тона — «Принцип едиственной обязанности» (Single Responsibility Principle, первая буква в аббревиатуре SOLID). Согласно этому правилу, каждый класс должен отвечать лишь за один какой-то аспект. Совершенно очевидно, что любой Singleton-класс отвечает сразу за две вещи: за то, что класс имеет лишь один объект, и за реализацию того, для чего этот класс вообще был создан. Принцип единственной обязанности был создан не просто так — если класс отвечает за несколько действий, то, внося изменения в один аспект поведения класса, можно затронуть и другой, что может сильно усложнить разработку.

• Переиспользование (reusability) класса практически невозможно