# Тестовое задание 2

## Метод решения

Основная задача — обработать начальные данные о сервисах и произвести расчет значений конфигурационных параметров в зависимости от указанных факторов. Конфигуратор поддерживает расширение на новые сервисы и параметры. Конфигуратор строится на основе класса `Calculator`, который принимает начальную конфигурацию сервисов и набор параметров. Каждый сервис имеет свои уникальные параметры, вычисляемые по правилам, описанным в ТЗ. Эти правила реализуются с помощью системы классов параметров, каждый из которых содержит бизнес-логику расчета конкретного параметра сервиса. Каждое значение параметра рассчитывается динамически, на основе входных значений.

## Фабрика параметров

Для каждого параметра создан отдельный класс, инкапсулирующий логику его расчета. `ParameterFactory` отвечает за создание нужного параметра для каждого сервиса, используя имя сервиса и имя параметра. Этот подход позволяет легко добавлять новые параметры и изменять существующие правила, не изменяя основной код. Фабрика параметров позволяет гибко обрабатывать конфигурации и поддерживает добавление новых параметров без изменения основной структуры программы.

## Сервисы

Сервис в конфигураторе представлен набором параметров, таких как `replicas`, `memory`, `cpu`, `storage`, и имеет свой класс в проекте (Фактически есть только 1 класс, экземпляры разных сервисов создаются динамически). Класс сервиса обращается к фабрике параметров для инициализации всех параметров и их последующего расчета. Таким образом, логика расчета параметров для каждого сервиса изолирована в отдельных классах.

## Взаимодействие с пользователем

Конфигуратор работает с входными данными из файлов config.json и params.json, что позволяет пользователю удобно передавать начальные конфигурации и параметры без необходимости взаимодействовать с программой напрямую.

## Тесты

Проект включает набор тестов, которые проверяют корректность работы конфигуратора с различными входными данными. Тесты охватывают как корректные, так и некорректные входные данные.

## Расширяемость

Проект изначально построен с учетом возможности расширения. Добавление нового сервиса или параметра требует только создания нового класса параметра и указания правил расчета. Благодаря фабричному методу, новые классы параметров автоматически интегрируются в систему без изменения основного кода. Это делает конфигуратор гибким и легким в поддержке, позволяя добавлять новые зависимости и параметры при необходимости.

## Запуск программы

Если файлы config.json и params.json находятся в корневой директории проекта, вы можете запустить программу следующим образом:

```
python3 main.py --config=config.json --params=params.json --output=stdout
```

Эта команда загрузит конфигурацию и параметры из указанных файлов, выполнит расчеты и выведет результат в консоль. Если вы хотите сохранить результат в файл, например output.json, используйте:

```
python3 main.py --config=config.json --params=params.json --output=output.json
```

Для запуска тестов используйте:

```
python3 -m pip install -r requirements.txt
python3 -m pytest tests/
```