# Exercises: SOLID

This document defines the exercises for the ["Java Advanced" course @ Software University](https://softuni.bg/trainings/4375/java-oop-february-2024).

In the resources, you will find a Java project. The source directory contains the **Main** and **CalorieCalculator** classes as well as the **products** class **package** with several products in it. **CalorieCalculator** finds the total amount of calories and their average value from a food collection.

### Notes

* **Drinks** are initialized by taking **milliliters** as an argument, while **foods** take on **weight**.
* The **weight** of the drink is found by **multiplying** **the milliliters by the density**. (milliliters \* density)
* The **value of** **calories** is found by **multiplying the amount of calories per 100 grams of product by the amount of product in grams**. (CALORIES\_PER\_100\_GRAMS / 100) \* grams

Carefully review the resources provided, then, relying on **SOLID** and **OOP** principles, you have to perform a task that consists of the following steps:

### Single Responsibility Principe

Step 1: The **CalorieCalculator** class implements methods that violate the **Single Responsibility Principle**. Refactor the code so that the relevant principle is relied upon.

Hint: Create a **Printer** class that defined the correspondent methods.

### Open-Closed Principe

Step 2: Create a new **Chips** product. Then refactor the logic so that we follow the **Open-Closed Principle**.

- **Chips** contain **529** calories per **100** grams.

**Hint**: Create a **Product** **interface** with a method that finds the **amount of calories** that products can implement.

### Interface Segregation Principle

Each product needs to be able to provide information on its quantity as follows:

* The amount of food in kilograms.
* The amount of drinks in liters.

**Hint**: Create **Food** and **Drink** **interfaces** with the desired methods. Food will implement **Food**, and drinks - **Drink**.

**Expand** the application by implementing functionality that finds from a collection of food products their **total amount** in kilograms and their **average value**.

* The **kilograms of drinks** are found **by multiplying the liters by the density**. (litters \* density)

**Hint**: Create a quantity calculator class similar to **CalorieCalculator** that executes the desired logic.

### Dependency Inversion Principle

Relying on the **Dependency Inversion principle**, refactor the **Printer** class so that it can work with both types of calculators.

### Liskov

Create a **Cloud** class to implement the Food interface.

**Hint**: You violated the principle here! Why?