## **1. Product Classes Mix Product Data with Tax Logic — Violates Single Responsibility Principle**

In the current design, each product class (Food, Cloth, Electronics) is responsible for two separate concerns:

* Representing a product’s name, price, and type — which is the product’s core business data.
* Calculating taxes for different regions — which is domain-specific policy logic.

Mixing these two responsibilities makes the code harder to maintain. For example, if a country changes its tax rate or if a product becomes exempt in a certain region, you have to go inside each product class and modify the getTax() method.

This violates the **Single Responsibility Principle**, which says that a class should only have one reason to change. Here, changes in tax policy force us to touch the product model itself — even though product data and tax policy are conceptually separate.

## **2. Tax Logic is Hardcoded and Spread Across Classes — Violates Open/Closed Principle**

Another major issue is that the tax logic is **hardcoded** in each product class, with explicit if-else branches for each region. This leads to two big problems:

* **Adding a new region** (like "Africa" or "South America") requires modifying every single product class to handle that new case.
* **Adding a new tax type** (like an import tax or luxury surcharge) also requires editing all product classes again.

This means the system is **not open for extension** (you can't easily add new behavior) and **not closed for modification** (you have to change existing code frequently). This directly violates the **Open/Closed Principle**, which states that software should be extendable without modifying its existing parts.

The result is a fragile, repetitive system where the same kind of logic (regional tax checks) is duplicated across all classes — increasing the risk of inconsistencies and bugs.

## **Summary**

* **Product classes are doing too much**, combining both product modeling and tax calculation, which leads to tight coupling and code that's hard to maintain.
* **Tax rules are scattered and duplicated** across all product types, making it hard to scale the system as you add new tax policies or regions.
* **The design violates key object-oriented principles** like SRP and OCP, which aim to keep code clean, maintainable, and extensibl