1. **Intent**

**Visitor** is a behavioral design pattern that lets you separate algorithms from the objects on which they operate

1. **Motivation: Tax Calculation in an E-commerce Platform**

**Context**

You’re developing an e-commerce platform that sells various types of products:

* FoodItem (bread, milk, …)
* Electronics (phones, TVs, …)
* Clothing (shirts, jackets, …)

Each product type has different tax rules:

* Food items may be tax-free
* Electronics are taxed at 15%
* Clothing is taxed at 10%, but children’s clothing may be exempt

These tax rules can vary by region, change frequently due to new regulations, or differ for imported goods.

**Task**

Now you want to calculate the tax for each product type during the checkout process.

But your system also needs to:

* Handle different tax policies (like USA, Canada, EU)
* Be easy to update when tax rules change
* Possibly support multiple tax visitors at once (local tax, environmental tax, …)

**What would you do?**

Add a calculateTax() method inside every product class?

If you take that approach, you'll quickly run into problems:

* You’ll need to modify every product class whenever tax rules change
* Adding new tax policies (for Canada, EU, or other regions) means duplicating or altering existing logic
* Your product classes will become cluttered with tax-related code, violating the Single Responsibility Principle
* Most importantly, this design breaks the Open/Closed Principle — classes should be open for extension, but closed for modification

A better solution?

🡪 Use the Visitor pattern to move tax-related logic into separate visitor classes. This keeps your product classes clean and stable, makes it easier to manage tax changes, and allows you to support multiple tax systems without modifying existing code.