1. **Intent**

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

1. **Motivation: Form Validation and Export in a Web Admin Panel**

**Context**  
You’re developing a web-based admin dashboard — for example, a CMS or HR management system — that allows users to build custom forms. These forms are composed of various field types:

* TextInput (for names or descriptions)
* DatePicker (for birthdates or deadlines)
* Checkbox (for terms and conditions)
* FileUpload (for resumes or attachments)
* Dropdown (for selecting roles or departments)

Each field type has different properties, but the system also needs to support several behaviors:

* **Validating** user input based on rules for each field type
* **Exporting** form definitions to formats like JSON or PDF
* **Localizing** labels and placeholders for multiple languages
* **Previewing** forms in a test mode before deployment

**Task**Now you want to implement these behaviors in a way that’s flexible, maintainable, and easy to extend as more requirements emerge.

**What would you do?**

Add validate(), export(), translate(), and preview() methods inside every field class?

That sounds convenient at first — but it causes several problems:

* Each field class becomes bloated with logic unrelated to its core purpose
* Adding or changing one behavior (how PDF export works) requires modifying every single field class
* You repeat similar logic across classes or introduce conditional checks
* You violate the **Single Responsibility Principle** and **Open/Closed Principle**

**A better solution?**  
→ Use the **Visitor pattern** to move each behavior into its own dedicated class.