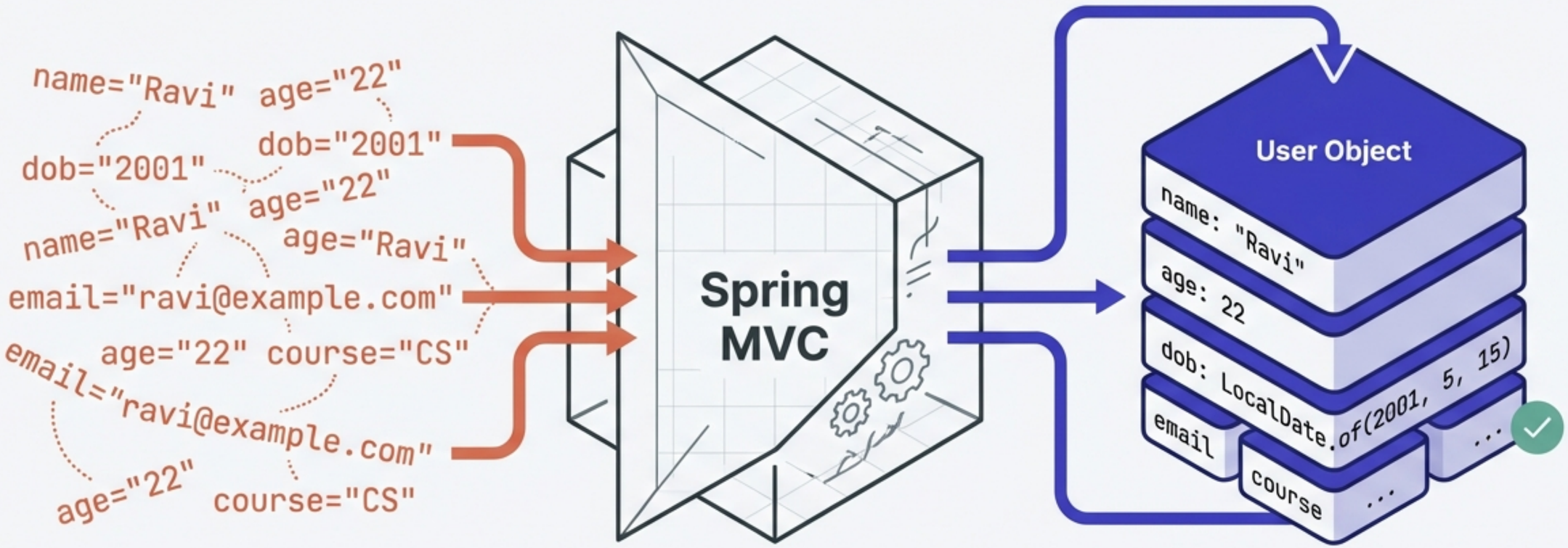
Posting Forms & Data Binding in Spring MVC

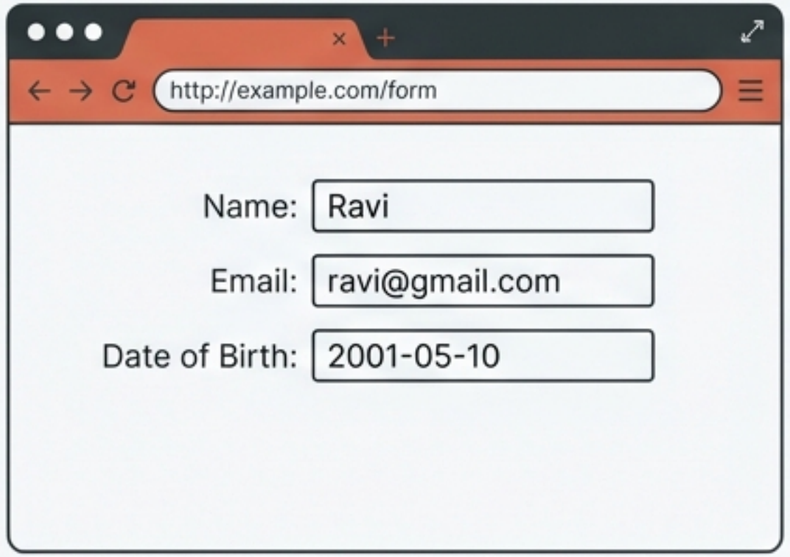
Translating HTTP Requests into Java Objects



A browser does not understand Java. A Java server does not understand HTML. Spring MVC acts as the translator between them.

Browsers Speak Text. Java Speaks Objects.

The Browser World

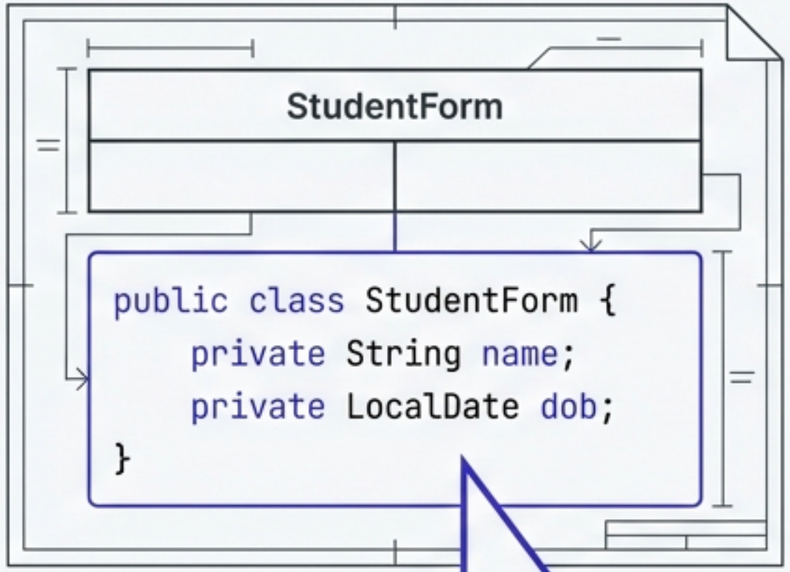



```
name = 'Ravi'
email = 'ravi@gmail.com'
dob = '2001-05-10'
```

HTTP only sends **Strings**. Never **objects**.

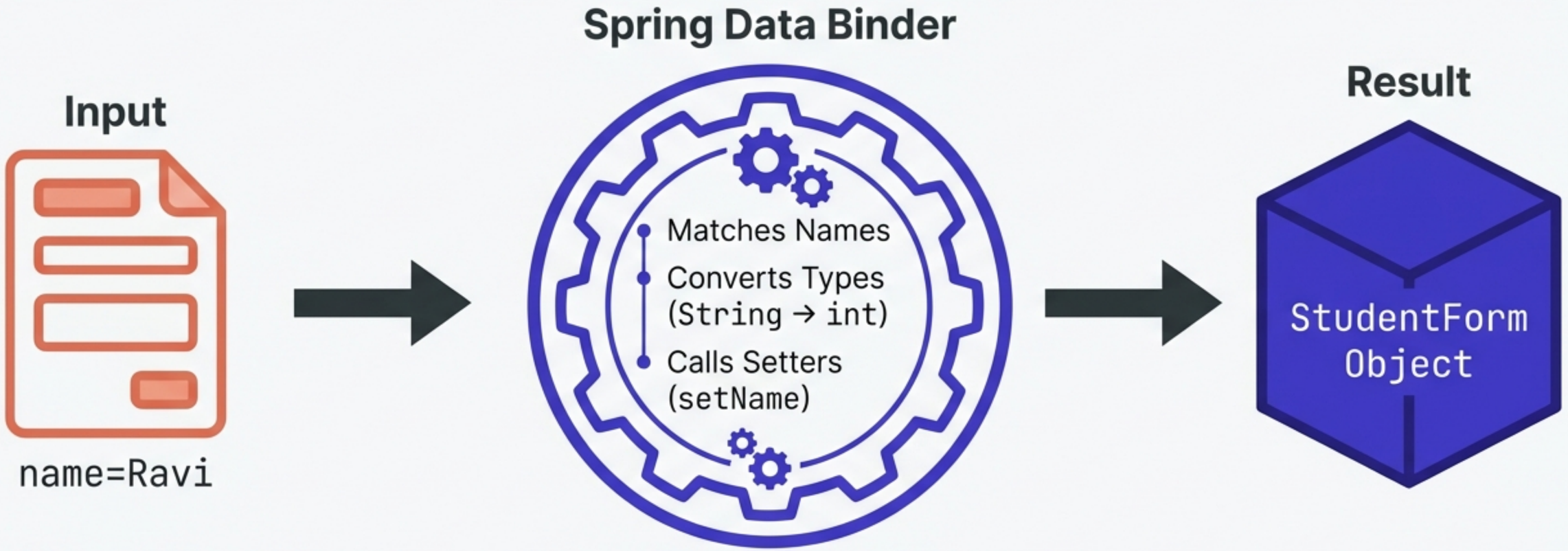
The Translation Gap

The Java World



Java needs **strongly** typed **Objects**.

Data Binding: The Automated Translator



Key Concept: Spring behaves like a disciplined intern. It automatically finds the right setter for every input field—provided the names match perfectly.

@ModelAttribute: The Form Handler

Picks the whole tree, not just one leaf.

```
@PostMapping("/students")
public String saveStudent(@ModelAttribute StudentForm form) {
    // Spring has already created and filled 'form'
    return "success";
}
```

@RequestParam	@ModelAttribute
• Picks one value	• Builds a complex object
• Good for simple queries	• Essential for Forms
• Manual extraction	• Automatic binding

Pro Tip This annotation works for both POST (saving data) and GET (pre-filling search forms).

Reading the HTTP "Parcel"

Data can be placed in **different parts of the request parcel.**

You must choose the specific annotation that tells Spring exactly **where to look.**

The "Manual" Way: HttpServletRequest

```
public void handle(HttpServletRequest request) {
    String name = request.getParameter("name");
    // Manual extraction required
}
```

Why to Avoid It

- Like opening the parcel and reading every piece of paper yourself.
- Boilerplate code.
- Hard to test.
- Easy to forget validation.

Advice: Only use when you need low-level control (e.g., raw stream manipulation). Otherwise, trust Spring's annotations.

Extracting from the URL

@RequestParam

Context: Filtering & Sorting

```
search(@RequestParam String query)
```

/search? query=spring

Optional Instructions

@PathVariable

Context: Identification

```
getStudent(@PathVariable int id)
```

/students/ 10

Mandatory Address

@RequestBody: Handling Modern APIs

Modern apps (React, Mobile) send JSON, not form fields.


```
@PostMapping("/api/students")
public StudentForm save(@RequestBody StudentForm form) { ... }
```

Key Difference: Uses the **Jackson** library to parse JSON. Do not confuse with @ModelAttribute!

Reading the Metadata

@RequestHeader

Reads information *about* the request (e.g., User-Agent, Authorization).

```
header(@RequestHeader("User-Agent") String agent)
```

@CookieValue

Reads ‘memories’ stored by the browser (e.g., theme, session_id).

```
theme(@CookieValue("theme") String theme)
```

These annotations handle the **context** of the request, not the user input.

The Date Problem: Teaching Spring Human Formats

User types `'15-01-2026'` (String) → Java wants `` `LocalDate` `` (Object). Default conversion ✕ often fails.

Solution 1: The Quick Fix

```
@DateTimeFormat(pattern="dd-MM-yyyy")
```

Placed directly on the field. Best for simple, one-off cases.

Solution 2: The Custom Teacher (@InitBinder)

```
@InitBinder
public void initBinder(WebDataBinder binder) {
  // Teach Spring a new date format
  binder.registerCustomEditor(...);
}
```

Runs *before* binding starts. Teaches the Data Binder how to interpret specific formats for the entire controller.

File Uploads: The Multipart Request

HTTP Parcel

Checklist for Success

- **HTML:** Must use `` `enctype="multipart/form-data\"` ``
- **Controller:** Use `@RequestParam MultipartFile file`
- **Security:** Always check file size and verify content type. **Never trust the filename!**

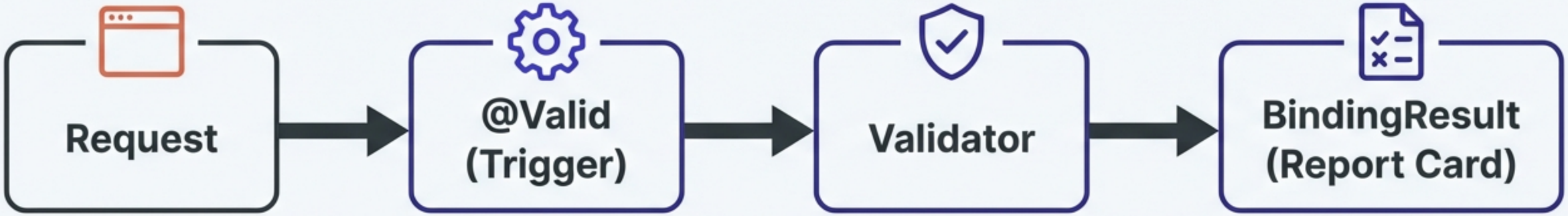
```java
public String upload(@RequestParam MultipartFile file) {
  ...
}
```

“Client-side validation is for User Experience. Server-side validation is for Security.”

Bean Validation Rules

Rules live on the Java Bean. Spring checks them automatically.

The Validation Workflow



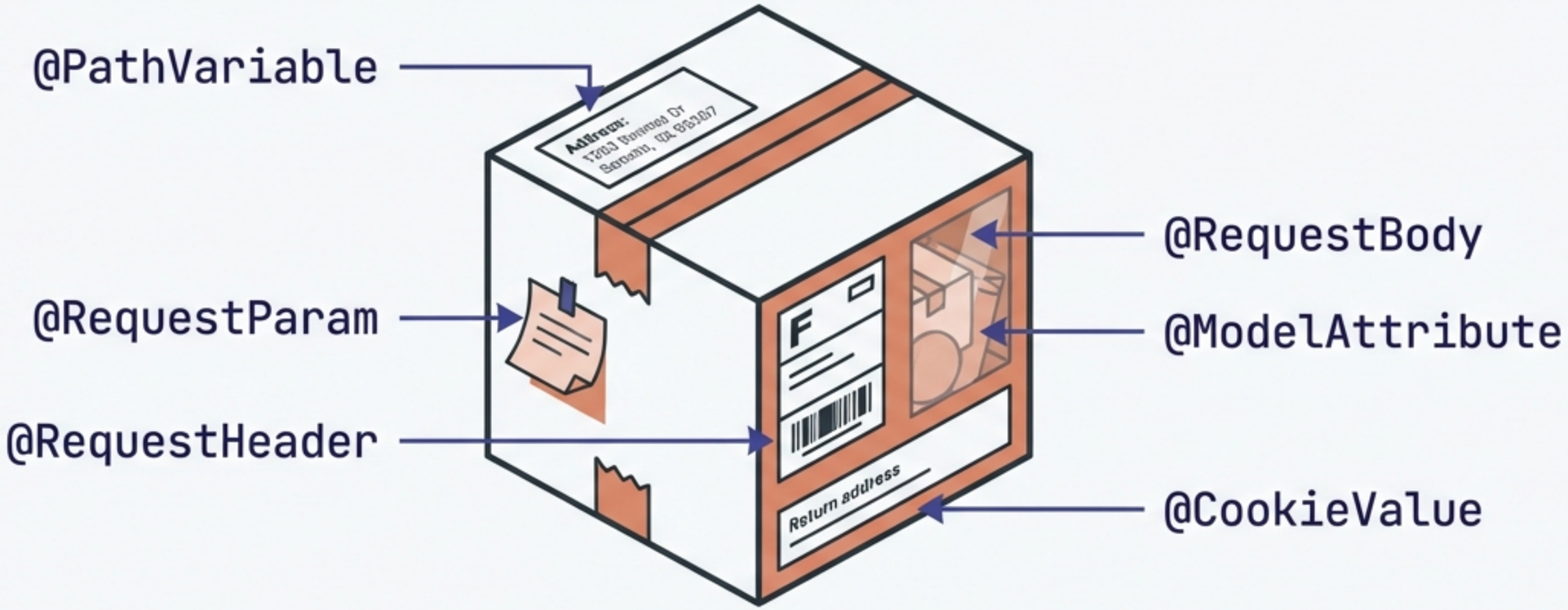

```
public String save(@Valid @ModelAttribute StudentForm form,
                   BindingResult result) {
    if (result.hasErrors()) return "form-view";
    return "success";
}
```

CRITICAL: The `BindingResult` argument must appear **immediately after** the object being validated. If you change the order, the application will crash.

The Master Memory Table

Annotation	Reads From	Purpose
`@RequestParam`	Query / Form Data	Optional values (Filtering)
`@PathVariable`	URL Path	Resource ID (Address)
`@RequestBody`	JSON Body	Complex Objects (APIs)
`@ModelAttribute`	Form Data	Binds entire objects (Forms)
`@RequestHeader`	HTTP Headers	Metadata (User-Agent)
`@CookieValue`	Cookies	Client State (Theme)
`@InitBinder`	Setup	Custom conversion (Dates)
`@Valid`	Validation	✅ Triggers security check

Spring MVC is Your Translator



Spring MVC is not magic. It simply knows **where to look** and **how to convert** data. If you know where the data lives in the HTTP request, you know exactly which annotation to use.