**SME to provide explanation about the following aspects:**

**Explain how spring framework takes care of converting the request payload into country bean:**

When you send a **POST** request with a JSON body like this:

{

"code": "IN",

"name": "India"

}

and your controller method is:

public Country addCountry(@RequestBody Country country)

**Spring Boot uses an internal mechanism to:**Read the JSON body of the HTTP request.  
Map (deserialize) the JSON into a Java object of type Country.  
Pass the created Country object as the parameter to your controller method.

**Spring parses the JSON request payload data using Jackson parser:**

The **Jackson library** (included with Spring Boot starter) does the actual JSON → Java and Java → JSON conversion.

* Jackson has a class called ObjectMapper.
* It reads the JSON from the request body.
* It creates an instance of the specified class (Country).
* Then it populates the fields of the object with the JSON data.

**For each attribute in JSON, respective method name is constructed by applying initcaps and get prefix. For example, the name attribute is changed with initcaps as Name, then get is prefixed to it which results in getName, based on this the respective method is invoked using Reflection API:**

**What does this mean?**

**JSON Key → Java Field**

When Jackson parses JSON:

{

"name": "India"

}

It looks for a **Java Bean property** named name.

**Java Bean Convention**

According to JavaBeans conventions:

* Field: private String name;
* Getter: public String getName()
* Setter: public void setName(String name)

The **property name** is name because of the method pair getName() / setName().

**Method Construction**

For each property:

* Take the JSON key → e.g., name
* Capitalize first letter → Name
* Prefix with get → getName() or set → setName(String)

**Reflection**

Jackson then uses **Java Reflection API** to:

* Search for a method called setName(String)
* If found → invoke it, passing "India" as the value

**Example Table:**

| **N JSON Key** | **Init** **→ Initcap** | **→ Setter Name** | **Get Getter** |
| --- | --- | --- | --- |
| N name | N name | setName(String) | Get getName() |
| code | Co code | Set setcode(String) | Get getCode() |

**Why Reflection?**

Because Jackson does not know at compile time what class it’s deserializing into.  
At runtime:

* It reads the JSON keys as strings.
* Converts each key → method name → looks it up via Reflection.
* Calls the method if it exists.

**Example Walkthrough**

**JSON:**

{

"name": "India",

"code": "IN"

}

**Bean:**

private String name;

private String code;

public String getName() { return name; }

public void setName(String name) { this.name = name; }

public String getCode() { return code; }

public void setCode(String code) { this.code = code; }

**Jackson Flow:**

| **N JSON Key** | **Init** **→ Initcap** | **→ Setter Name** | **Get → Called?** |
| --- | --- | --- | --- |
| N name | N name | setName(String) | Get ✅ |
| code | Co code | Set setcode(String) | Get ✅ |

**Code Example:**

Field field = Country.class.getDeclaredField("name");

Method setter = Country.class.getMethod("setName", String.class);

setter.invoke(countryObj, "India");

This is essentially what Jackson does internally!

**Why initcaps and prefix?**

Because of **JavaBean conventions** — tools like Jackson, Spring, Introspector all follow this standard:

* Property x → getX(), setX().

**Spring creates country object and invokes the respective setter method based on JSON data:**

**What happens inside Spring & Jackson?**

When you send a request like:

{

"code": "IN",

"name": "India"

}

and your controller looks like this:

@PostMapping

public Country addCountry(@RequestBody Country country) {

}

**Step-by-step:**

**Spring detects @RequestBody**

* Since your method parameter has @RequestBody, Spring knows:

This parameter must be populated from the **HTTP request body**.

**Jackson creates the object**

* Spring delegates to the **Jackson library (ObjectMapper)** to read the JSON.
* Jackson does:
  + Calls new Country() → creates an empty Country object.
  + This happens because the Country class has a **default constructor**.

**Jackson maps JSON keys to Java properties**

* For each JSON key ("code", "name"), Jackson:
  + Converts it to a property name.
  + Looks for the setter method:
    - setCode(String code)
    - setName(String name)

**Jackson invokes setters**

* Jackson calls the setters using **Reflection API**:

country.setCode("IN");

country.setName("India");

**Fully populated object**

* After setting all the properties, the Country object now looks like:

Country{code='IN', name='India'}

* This populated object is passed as the argument to your controller method.

**The it invokes the controller method passing the country object created:**

Client sends JSON →  
✅ Spring uses Jackson to parse JSON & create a Country object →  
✅ Jackson sets the fields by calling setters →  
✅ Spring finds your controller method →  
✅ Passes the Country object to the method →  
✅ Method executes with the populated object.

**Provide explanation regarding bean naming conventions:**

**JavaBean Naming Conventions**

These are standard rules for defining a Java Bean, which Spring & Jackson rely on:

**Fields (Properties)**

* Private fields (encapsulation).

private String name;

**Getters**

* Public method to get field value.
* Name starts with get + capitalized field name.

public String getName() {

return name;}

**Setters**

* Public method to set field value.
* Name starts with set + capitalized field name.

public void setName(String name) {

this.name = name;

}

Example:

| **Field** | **Getter** | **Setter** |
| --- | --- | --- |
| name | getName() | setName(String name) |
| code | getCode() | setCode(String code) |

**Why follow these conventions?**

✅ Jackson & Spring use **Reflection** to find getters/setters.  
✅ JSON key → field name → finds matching setFieldName() method.  
✅ If conventions are not followed → JSON mapping may fail.