**Exercise 4: Online Bookstore - Processing Request Body and Form Data**

Business Scenario:

Create endpoints to accept and process JSON request bodies and form data for customer registrations.

1. **JSON Request Body**: A POST endpoint that creates a new customer from a JSON payload.
2. **Form Data**: An endpoint that processes customer registration using form parameters.

**Key Points of the Code**

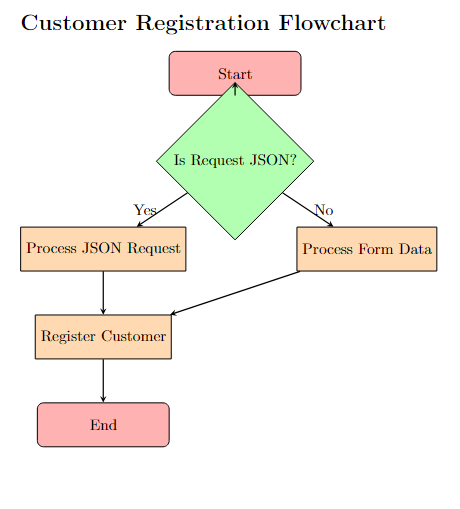
**1. JSON Request Body**

* **Endpoint**: POST /customers
* **Purpose**: To create a new customer by accepting a JSON request body.
  + **@RequestBody**: This annotation tells Spring Boot to map the incoming JSON request body to a Customer object.
  + **Customer**: A model class that represents a customer with attributes such as id, name, and email.
  + **Return**: The endpoint returns the Customer object that was created, allowing the client to confirm the registration details.

**2. Form Data**

* **Endpoint**: POST /customers/register
* **Purpose**: To process form data for customer registrations.
* **@RequestParam**: This annotation is used to extract form parameters from the request.
* **Form Data**: The name and email parameters are extracted from the form data and used to register the customer.
* **Return**: The endpoint returns a confirmation string with the provided name and email, indicating successful registration.

**FLOWCHART:**



**Explanation:**

1. **Start**: The flowchart begins here.
2. **Is Request JSON?**: Decision node to check if the request is JSON.
   * If **Yes**, the process moves to **Process JSON Request**.
   * If **No**, the process moves to **Process Form Data**.
3. **Process JSON Request**: Handles customer data sent as a JSON request body.
4. **Process Form Data**: Handles customer data sent as form parameters.
5. **Register Customer**: Both types of data are then used to register the customer.
6. **End**: The process concludes.

**CLASS DIAGRAM :**



**Explanation:**

* **Book Class**: Represents a book with attributes like id, title, author, price, and isbn.
* **Customer Class**: Represents a customer with attributes like id, name, and email.
* **BookController Class**: Contains methods for managing books (getAllBooks, addBook, updateBook, deleteBook, getBookById, searchBooks).
* **CustomerController Class**: Contains methods for managing customers (createCustomer, registerCustomer).

**Relationships:**

* **BookController** manages instances of the Book class.
* **CustomerController** manages instances of the Customer class.