|  |  |  |
| --- | --- | --- |
| 1. **Expected Outcome** | 1. **Activity/Analogy/Real-life usecase** | 1. **Demo idea or Code snippet  (with new learnings mentioned)** |
| 1. User understands the need for serialisation and deserialisation in an application | **Use-case #1:** Consider a messaging application that needs to send complex message objects between clients. | // **1.1 Defining a serializable class**  class MyClass implements Serializable {  private static final long serialVersionUID = 1L;  private String data;  public MyClass(String data) {  this.data = data;  }  }  // **1.2 Serialization**  try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("serializedObject.ser"))) {  MyClass myObject = new MyClass("Hello, Serialization!");  oos.writeObject(myObject);  System.out.println("Object has been serialized.");  } catch (IOException e) {  e.printStackTrace();  } |
| **Use-case #2** Consider a messaging application that receive complex message objects between clients**.** | **// 1.3 Deserialization**  try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("serializedObject.ser"))) {  MyClass deserializedObject = (MyClass) ois.readObject();  System.out.println("Deserialized Object Data: " + deserializedObject.getData());  }  catch (IOException) {  e.printStackTrace();  } |
| 2. User understands the need for a library like Jackson to deserialise JSON | **Activity #1:** To write a program where an application needs to process data received from a RESTful API. | // **2.1 Extract JSON values without the use of a library**  private static JsonObject parseJson(String jsonString) {  **// Implementation of manual JSON parsing**  **// This is a simplified example; real-world scenarios can be much more complex and error-prone when done manually.**  **// It is also limited in handling nested structures, arrays, etc.**  **// The purpose is to highlight the challenges without a dedicated library.**    JsonObject jsonObject = new JsonObject();  **// 2.2 Manual parsing logic …..**  return jsonObject;  } |
| 3. User understands the need for @JsonProperty annotation | **Use-case #3:** Consider a JSON object with keys like "full\_name," "user\_age," and "user\_city," and a Java class with properties like "fullName," "age," and "city." | // **3.1 Failing to perform deserialization**  Jackson would be able to perform the deserialization correctly if the property names (fullName, age, city) match the corresponding JSON keys (fullName, age, city). However, if the JSON keys followed a different naming convention or had a different structure, automatic mapping might fail. |
| 4. User should be able to use the @JsonProperty annotation | **Use-case #4** Illustrate how @JsonProperty can bridge the gap in **Use-case #3** | **// 4.1 Use of @JsonProperty annotation**  @JsonProperty("full\_name")  private String fullName;  @JsonProperty("user\_age")  private int age;  @JsonProperty("user\_city")  private String city;  ObjectMapper objectMapper = new ObjectMapper();  try {  User user = objectMapper.readValue(jsonString, User.class);  System.out.println(user.toString());  }  catch (JsonProcessingException e) {  e.printStackTrace();  }  // **We can perform deserialization even if the property name doesnot match the corresponding Json keys because of the use of the @JsonProperty annotation** |
| 5. User should be able to deserialise JSON with Jackson | **Activity #2:** To write a program where an application needs to process data received from a RESTful API using Jackson library. | //**5.1 Importing Jackson library**  import com.fasterxml.jackson.databind.ObjectMapper;  **// 5.2 Jackson JSON deserialization**  ObjectMapper objectMapper = new ObjectMapper();  try {  **// 5.3 Read JSON string and map it to a JsonNode**  JsonNode jsonNode = objectMapper.readTree(jsonString);  **// 5.4 Accessing values**  String name = jsonNode.get("name").asText();  int age = jsonNode.get("age").asInt();  String city = jsonNode.get("city").asText();  **// 5.5 Using extracted values**  System.out.println("Name: " + name);  System.out.println("Age: " + age);  System.out.println("City: " + city);  }  catch (Exception e) {  e.printStackTrace();  } |
| 6. User should be able to serialise Java object to a JSON file with Jackson | **Activity #3 :** To serialize a Java object to a JSON file using Jackson. | **// 6.1 Import Important libraries**  import com.fasterxml.jackson.databind.ObjectMapper;  import java.io.File;  import java.io.IOException;  User user = new User("John Doe", 25, "New York");  **// 6.2 Create an ObjectMapper instance**    ObjectMapper objectMapper = new ObjectMapper();  **// 6.3Specify the file path for the JSON file**  String jsonFilePath = "user.json";  try {  **// 6.4 Serialize the User object to a JSON file**  objectMapper.writeValue(new File(jsonFilePath), user);  System.out.println("Serialization successful. " + jsonFilePath);  }  catch (IOException e) {  e.printStackTrace();  } |