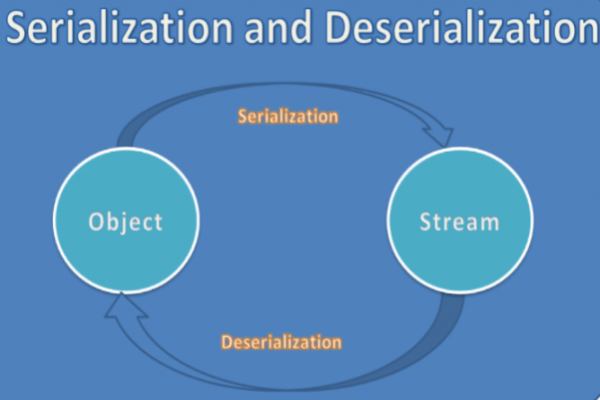
**Serializers/De-serializers in 5 Mins**

[[](https://www.java67.com/2020/05/15-java-serialization-interview-questions-answers.html)](https://www.java67.com/2020/05/15-java-serialization-interview-questions-answers.html)

When it comes to developing a modern-day application, especially web applications there are many topics to can easily go under the table without any notice and developers often ignore the topics while implementing their cutting-edge solutions.

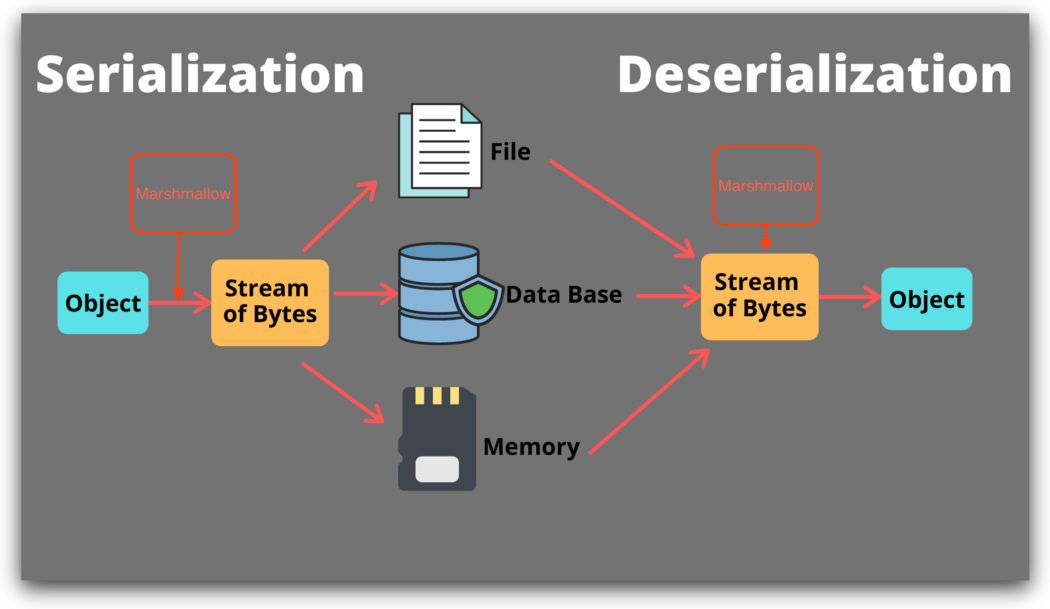
But this does not in any way decreases the importance of these topics. More often or not having knowledge of these topics helps in understanding, finding, and implementing different solutions for existing problems.

[Serializers/Deserializers](https://javarevisited.blogspot.com/2016/09/how-to-serialize-object-in-java-serialization-example.html) are one such topic that might be ignored till the very last point of getting a blocker in your progress. Ignoring these may not be the worst solution all the time since many of the modern libraries and frameworks have automated a lot of stuff which otherwise we need to take care of manually. But not knowing these at all can result in a lot of head scratching and frustration when one gets into a problem which is related to the serialization of data.

In this article, we will take a look into [what are serializations and deserialization](https://www.java67.com/2016/01/what-is-default-serialization-in-java.html) and also how we can use the existing serializers to have a custom implementation.

**What is Serialization/Deserialization?**

**Serialization** is a mechanism of converting the state of an object into a byte stream. The byte stream created is platform-independent. So, the object serialized on one platform can be deserialized on a different platform. **Deserialization**as the name suggest is the opposite of the serialization process. Whenever we want to fetch/use the stored data, to make the data human usable we need to deserialize the byte stream to objects. Deserialization is the process where the byte stream is used to recreate the actual object in memory. This mechanism is used to persist the objects.

[[](https://www.java67.com/2021/04/java-serialization-tutorial-example-.html)](https://www.java67.com/2021/04/java-serialization-tutorial-example-.html)

**Why do we need Serialization/Deserialization?**

Almost every modern-day languages use objects to store and work with any form of data. These are just the human understandable form of data. Whenever we want to send/store these objects we need to convert the objects to a byte stream.

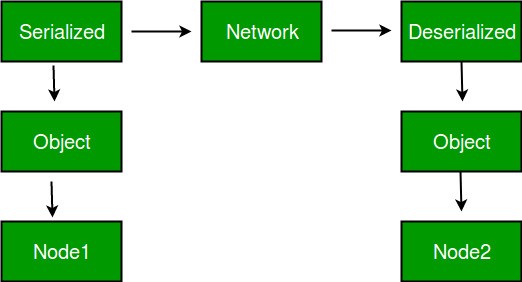
This is where serializations come to play which helps convert the objects to a byte stream. Now since we are converting the objects to byte stream we will need to do the reverse when we are receiving the objects from the network. This is where de-serializers come to play which convert the byte stream to a language object.

**Why custom Serializer/De-serializer?**

There are many popular serializers and de-serializers libraries out there like [Jackson](https://www.java67.com/2019/09/3-ways-to-ignore-null-fields-in-json-java-jackson.html), [GSON](https://javarevisited.blogspot.com/2022/03/3-examples-to-parse-json-in-java-using.html), etc. Most of the time implementation of the serializers is sufficient enough to meet the requirement.

But when you are dealing with **complex data objects(eg: FHIR Resources)**then the default implementation may not be sufficient enough for handling these on its own. Even if you don't want to handle complex objects, custom serializers/de-serializers would go long way in saving the computation time that you might have to do on your data.

Below we will discuss how we can implement the [custom Jackson Serializers](https://www.java67.com/2019/05/how-to-read-csv-file-in-java-using-jackson-library.html)and also we will see how we can transform the data passed over the rest call. In this example, we will look at how we can change the Peron object to the Employee object on the fly using the custom Jackson Serializers/De-Serializers.

[[](https://javarevisited.blogspot.com/2014/12/how-to-read-write-json-string-to-file.html)](https://javarevisited.blogspot.com/2014/12/how-to-read-write-json-string-to-file.html)

**Step 1: Create Model Class for the Data**

Following is the Person mode class which will act as the placeholder for the person’s data inside our application. In the following code, we have also mentioned the Serializers/De-Serializers to be used for the Person model classes.

@Builder  
@Getter  
@Setter  
@JsonSerialize(using = CustomSerializer.class)  
@JsonDeserialize(using = CustomDeserializer.class)  
public class Person {  
 int id;  
 String firstName;  
 String lastName;  
 int age;  
 String sex;  
}

**Step 2: Custom Serializer/De-serializer**

**Serializers:**Following is the implementation of the CustomSerializer. This serializer will convert the person object to the Employee

public class CustomSerializer extends StdSerializer<Person> {  
  
 protected CustomSerializer(Class<Person> t) {  
 super(t);  
 }  
 public CustomSerializer(){  
 this(null);  
 }  
  
 @Override  
 public void serialize(Person person, JsonGenerator gen, SerializerProvider provider) throws IOException {  
 gen.writeStartObject();  
 gen.writeNumberField("Emp Id",person.getId());  
 gen.writeStringField("Emp Name",person.getFirstName()+" "+person.getLastName());  
 gen.writeNumberField("Emp Age",person.getAge());  
 gen.writeStringField("Emp Gender", person.getSex());  
 }  
}

**2. De-serializer:**Following is the implementation of the CustomDeserializer. This will translate the incoming Employee object to the Person object.

public class CustomDeserializer extends StdDeserializer<Person> {  
 protected CustomDeserializer(Class<Person> vc) {  
 super(vc);  
 }  
 public CustomDeserializer(){  
 this(null);  
 }  
  
 @Override  
 public Person deserialize(JsonParser p, DeserializationContext ctxt) throws IOException, JacksonException {  
 JsonNode node=p.getCodec().readTree(p);  
 Person person=Person.*builder*()  
 .id(node.get("Emp Id").asInt())  
 .firstName(node.get("Emp Name").asText().split(" ")[0])  
 .lastName(node.get("Emp Name").asText().split(" ")[1])  
 .sex(node.get("Emp Gender").asText())  
 .age(node.get("Emp Age").asInt()).build();  
 return person;  
 }  
}

**Step 3: Registering the Custom Serializers/De-serializers**

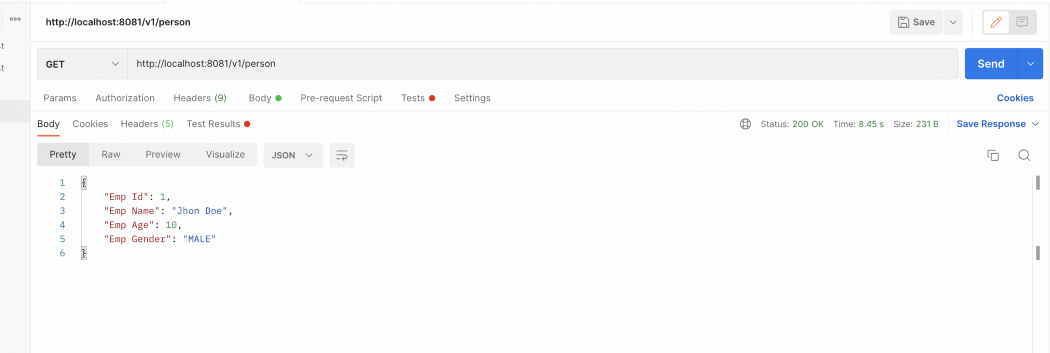
[Spring Boot](https://medium.com/javarevisited/10-free-spring-boot-tutorials-and-courses-for-java-developers-53dfe084587e) uses **Jackson’s ObjectMapper library** to serialize/de-serialize the incoming/outgoing objects. For the object mapper to use our custom implementation we need to register our custom implementation with [ObjectMapper](https://javarevisited.blogspot.com/2018/02/how-to-parse-json-with-date-field-in-java-jackson-example.html). To do this task ObjectMapper provides the SimpleModule class to register all your Serializers/De-serializers.

@Configuration  
public class SerializationConfig {  
  
 @Bean  
 @SuppressWarnings({"rawtypes", "unchecked"})  
 public ObjectMapper getObjectMapper() {  
 ObjectMapper mapper = new ObjectMapper();  
 SimpleModule simpleModule = new SimpleModule();  
 simpleModule.addSerializer(Person.class, new CustomSerializer());  
 simpleModule.addDeserializer(Person.class, new CustomDeserializer());  
 mapper.registerModule(simpleModule);  
 return mapper;  
 }  
}

**Step 4: Examples:**

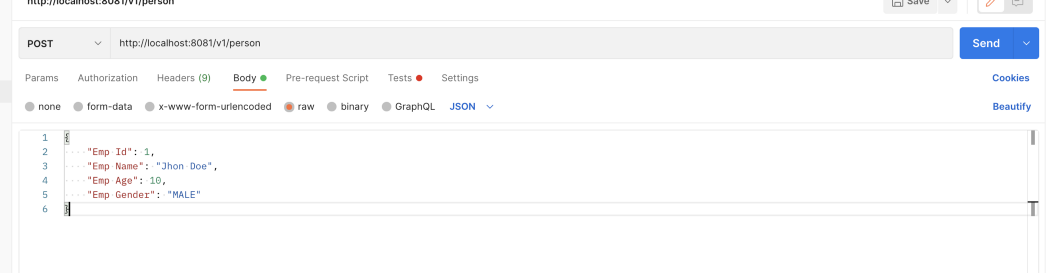
1. **Serialization:**While we are sending the object to the network, [Spring Boot](https://medium.com/javarevisited/top-10-courses-to-learn-spring-security-and-oauth2-with-spring-boot-for-java-developers-8f0222d6066d) will internally call the Jackson Serializer to serialize the object to the stream. If the object’s type matches the registered serializer, it will automatically use our custom implementation of the serializers. Hence on sending the Person object will be converted to the Employee object.

@RestController  
@RequestMapping("/v1/person")  
@AllArgsConstructor  
public class PersonController {  
  
  
 @GetMapping  
 public Person getPerson(){  
 Person person=Person.*builder*()  
 .id(1)  
 .firstName("Jhon")  
 .lastName("Doe")  
 .sex("MALE")  
 .age(10).build();  
  
 return person;  
 }  
  
 @PostMapping  
 public void savePerson(@RequestBody Person person){  
 System.*out*.println("\*");  
 System.*out*.println("\*");  
 System.*out*.println("\*");  
 System.*out*.println(person);  
 System.*out*.println("\*");  
 System.*out*.println("\*");  
 System.*out*.println("\*");  
 }  
  
}

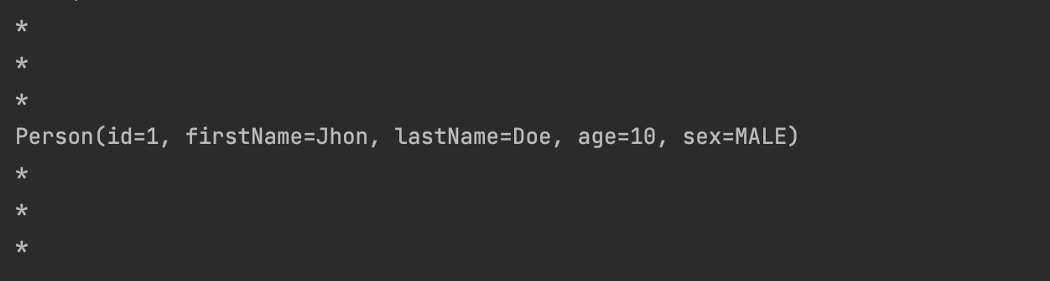


Output: getPerson

2**. De-serialization:**While we are receiving the Object over the network, it will be in stream format and to make this object human readable[Spring Boot](https://medium.com/javarevisited/top-5-books-to-learn-spring-boot-and-microservices-for-experienced-java-developers-499a9d921d18?source=---------25------------------) will use the JacksonDeserializer to de-serialize the incoming stream to the object, if the incoming stream matches the registered the deserializer Jackson will use the same to de-serialize the stream to object.

[[](https://www.java67.com/2021/02/spring-boot-actuator-interview-questions-answers-java.html)](https://www.java67.com/2021/02/spring-boot-actuator-interview-questions-answers-java.html)

Calling: savePerson

[[](https://www.java67.com/2012/08/spring-interview-questions-answers.html)](https://www.java67.com/2012/08/spring-interview-questions-answers.html)

Output: savePerson

**Summary**

In the above article, we looked into what is serialization/de-serialization, what is the need for serialization/de-serialization and how the serializer/de-serializers are implemented. We also looked into how we can implement a custom serializer/de-serializer that will translate the Person object to the Employee Object.