3. Crie uma classe Java de nome Serializa para instanciar objetos da classe definida na Questão 1 e adicionar esses objetos em uma Lista. Depois percorrer a lista e Serializar os objetos em disco/ssd. Serialize para os formatos a seguir: 1. Serialização de objetos da própria API Java (Introduction to Java Serialization | Baeldung), 2. JSON usando a biblioteca Jackson (Intro to the Jackson ObjectMapper | Baeldung), 3. XML usando a biblioteca Jackson (XML Serialization and Deserialization with Jackson | Baeldung).

4. Crie uma classe java de nome Desserializa para ler / desserializar os objetos Serializados na Questão 2 e exibi-los. Não precisa implementar nos 3 formatos usados na Questão 3. Basta escolher um deles (Objeto Java, JSON ou XML).

Como fiz?

- Como ambas as questões se complementam resolvi fazer ambas no mesmo projeto.
- O projeto possui o seguinte modelo, e é com ele que vamos brincar com xml, json e a serialização padrão do java:

```
package com.serializing.models;
import java.io.Serializable;

public class Person implements Serializable {
  private String name;
  private String email;
  private String phone;
  private Integer age;

  public Person(){}
  public Person(String name, String email, String phone, Integer age) {
```

```
    this.name = name;
    this.email = email;
    this.phone = phone;
    this.age = age;
  }

  public String getName() { return name; }
  public String getEmail() { return email; }
  public String getPhone() { return phone; }
  public Integer getAge() { return age; }

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

  public void setEmail(String email) {
    this.email = email;
  }

  public void setPhone(String phone) {
    this.phone = phone;
  }

  public void setAge(Integer age) {
    this.age = age;
  }

  @Override
  public String toString() {
    return "{ name:" + this.name + ", email: " + this.email + ", phone: " + this
  }
}
```

- Resolvi criar classes genéricas para mapear os objetos, seja em JSON, XML ou objeto java. Para que a serialização possa ser realizada as classes mapeadoras devem implementar a inteface MapperContract. Basicamente essa interface necessita que os métodos `serialization` e `desserialization` sejam implementados, veja abaixo a interface:

```
package com.serializing.contracts;

import java.io.InputStream;
import java.io.OutputStream;

public interface MapperContract<T> {
  public void serialization(OutputStream source, Object value) throws Exception;

  public T deserialization(InputStream source, Class<T> valueType) throws Except
}
```

- Veja acima que por ser genérico podemos criar abstrações que retornam diretamente o tipo da classe do objeto passado.
- Como as questões pedem para serializarmos e desserilizarmos os objetos em json, xml e objeto java então criei as seguintes classes mapeadoras.

```
// -> JSON Mapper
package com.serializing.helpers.mappers;
import java.io.InputStream;
import java.io.OutputStream;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.serializing.contracts.MapperContract;

public class MapperJSON<T> implements MapperContract<T> {

  private ObjectMapper objectMapper;

  public MapperJSON() {
    this.objectMapper = new ObjectMapper();
  }

  @Override
  public void serialization(OutputStream source, Object value) throws Exception
    this.objectMapper.writeValue(source, value);
  }

  @Override
  public T deserialization(InputStream source, Class<T> valueType) throws Except
    T valueObject = this.objectMapper.readValue(source, valueType);
    return valueObject;
  }

}

// -> XML mapper
package com.serializing.helpers.mappers;
import java.io.InputStream;
import java.io.OutputStream;
import com.fasterxml.jackson.dataformat.xml.XmlMapper;
import com.serializing.contracts.MapperContract;

public class MapperXML<T> implements MapperContract<T> {

  private XmlMapper mapper;

  public MapperXML() {
    this.mapper = new XmlMapper();
  }

  @Override
```

```
  public void serialization(OutputStream source, Object value) throws Exception
    this.mapper.writeValue(source, value);
  }

  @Override
  public T deserialization(InputStream source, Class<T> valueType) throws Except
    T valueObject = this.mapper.readValue(source, valueType);
    return valueObject;
  }

}


// -> Object mapper
package com.serializing.helpers.mappers;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.OutputStream;
import com.serializing.contracts.MapperContract;

public class MapperObject<T> implements MapperContract<T> {

  public MapperObject() {}

  @Override
  public void serialization(OutputStream source, Object value) throws Exception
    ObjectOutputStream objectOutputStream = new ObjectOutputStream(source);
    objectOutputStream.writeObject(value);
    objectOutputStream.close();
  }

  @Override
  public T deserialization(InputStream source, Class<T> valueType) throws Except
    ObjectInputStream objectInputStream = new ObjectInputStream(source);
    @SuppressWarnings("unchecked")
    T valueObject = (T) objectInputStream.readObject();
    objectInputStream.close();

    return valueObject;
  }

}
```

- Por fim temos uma classe de serviço que se chama `MapperServicePerson` ela é onde ocorre toda a lógica de ler os dados dos arquivos, desserailizar e também serializar.

Aqui seria a questão 3 :D

  - No método `serialization` criei de maneira estática três objetos de `Person` e adiciono

em uma lista, e a partir dessa lista itero sobre cada objeto e crio um diretório dentro de `resources` onde o nome do diretório é próprio atributo `name` do objeto e dentro dele é criado os arquivos com os dados serializados em json, xml e objeto java, abaixo é mostrado o log de quando serializamos esses dados:

```
------ Serializing objects ------
generate files to Henricker:
- resources/Henricker/object.json
- resources/Henricker/object.xml
- resources/Henricker/object.txt
generate files to Ylana:
- resources/Ylana/object.json
- resources/Ylana/object.xml
- resources/Ylana/object.txt
generate files to Clidenor:
- resources/Clidenor/object.json
- resources/Clidenor/object.xml
- resources/Clidenor/object.txt
```

- A estrutura de pastas dos arquivos serializados é algo dessa forma

```
resources
|
-- Clidenor
|  |
|  -- object.json
|  -- object.txt
|  -- object.xml
|
-- Henricker
|  |
|  -- object.json
|  -- object.txt
|  -- object.xml
-- Ylana
|  |
|  -- object.json
|  -- object.txt
|  -- object.xml
```

Aqui seria a questão 4 :D

- No método `desserialization` ele recebe como parâmetro o mimetype do arquivo a ser desserializado (xml, json ou txt) a partir disso será lido todos os objetos a partir do mimetype enviado, siga o abaixo onde eu carrego os objetos apenas pelos arquivos json.

```
MapperServicePerson msp = new MapperServicePerson();
msp.desserialization("json"); // txt, json or xml on params
```

- Como output será printado no terminal os objetos (toString) carregados pelo tipo de arquivo enviado como parâmetros:

```
------ Deserializalizing objects by json------
Actual directory: resources/Ylana
- Reading: resources/Ylana/object.json
- Object: { name:Ylana, email: ylana@email.com, phone: 008888888888, age: 21
Actual directory: resources/Henricker
- Reading: resources/Henricker/object.json
- Object: { name:Henricker, email: henricker@email.com, phone: 8800000000, a
Actual directory: resources/Clidenor
- Reading: resources/Clidenor/object.json
- Object: { name:Clidenor, email: clidenor@email.com, phone: 11122233333, ag
```

Como rodar o projeto ?

- Basta ir na classe principal App.java e rodar, dentro do método main faço a chamada tanto do método que vai criar os arquivos quanto do método que desserializa com base no tipo de arquivo (json, xml e txt).
- Sou aberto a melhorias!, podem ver a fundo o projeto, criticar, e avaliar!.