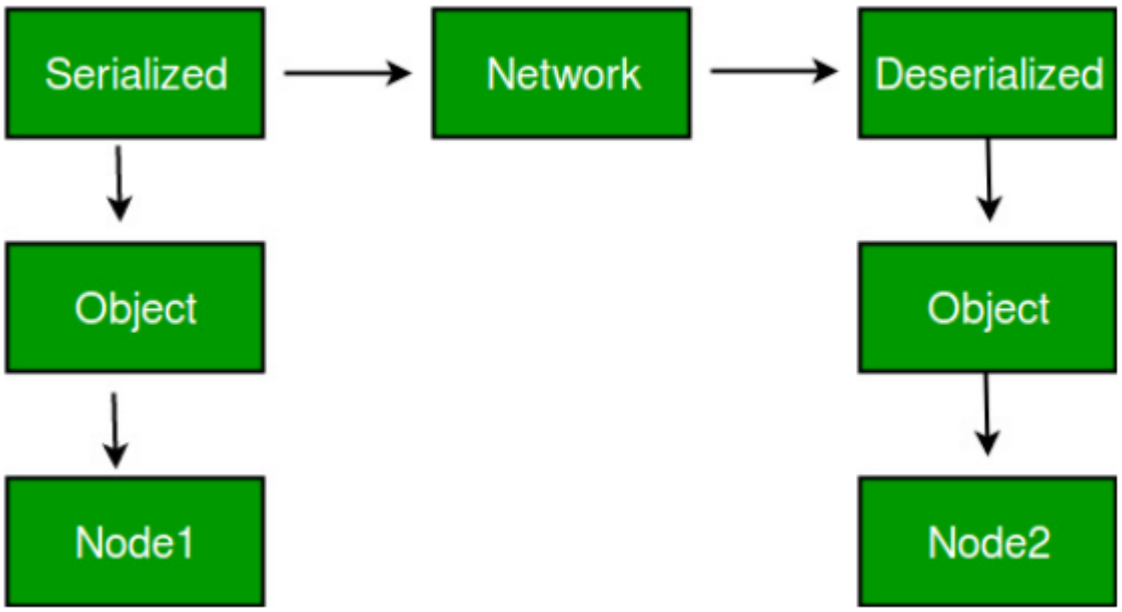
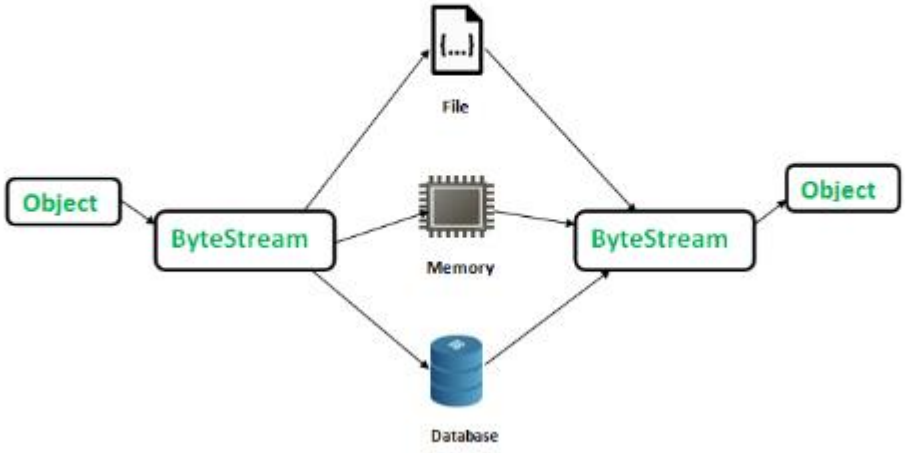
Serialization



Serialization **De-Serialization**



Reference Docs:
https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html
https://docs.oracle.com/javase/8/docs/technotes/guides/serialization/index.html
https://www.baeldung.com/java-serialization

Important points

1. Class must have implemented Serializable interface for serialization process, else NotSerializableException will be thrown.
2. All family members (Object Graph) should have implanted Serializable interface.
3. Serializable is a marker interface.
4. transient keyword is used to avoid serialization, can be used with variables.
5. Callback method can be used by serialization process.

```
    private void writeObject(java.io.ObjectOutputStream out) throws IOExcep-
tion;

    private void readObject(java.io.ObjectInputStream in)
        throws IOException, ClassNotFoundException;

    private void readObjectNoData()
        throws ObjectStreamException;
```

6. The *serialVersionUID* attribute is an identifier that is used to serialize/deserialize an object of a Serializable class.

```
import java.io.Serializable;


//if Student does not implements Serializable then NotSerializableException will be thrown at runtime
public class Student implements Serializable {
    int rollNumber;
    String name;

    public Student(int rollNumber, String name) {
        this.rollNumber = rollNumber;
        this.name = name;
    }

    public int getRollNumber() {
        return rollNumber;
    }

    public String getName() {
        return name;
    }

    @Override
    public String toString() {
        return "Student{" +
                "rollNumber=" + rollNumber +
                ", name='" + name + '\'' +
                '}';
    }
}


import java.io.*;

public class Serialization {

    public static void main(String[] args) throws IOException, ClassNotFoundException {
```

```
        Student student = new Student (1, "John");
        serializeStudent(student);


        Student student1 = deSerializeStudent();
        System.out.println(student1);

    }

    public static void serializeStudent(Student student) throws IOException {
        FileOutputStream fos = new FileOutputStream("john.serialized");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(student);
        oos.flush();
        oos.close();//Closes the stream. This method must be called to release any resources associated with
the stream.
    }

    public static Student deSerializeStudent() throws IOException, ClassNotFoundException {
        FileInputStream fos = new FileInputStream("john.serialized");
        ObjectInputStream oos = new ObjectInputStream(fos);
        Student john = (Student1) oos.readObject();
        oos.close();//Closes the stream. This method must be called to release any resources associated with
the stream.
        return john;
    }
}
```

```
//all family members should have implemented Serializable for serialization process.

import java.io.Serializable;

class City implements Serializable{
    String name;

    public City(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return "City{" +
                "name='" + name + '\'' +
                '}';
    }
}

//Address must be Serializable
class Address implements Serializable {

    String lane1Address;
    String lane2Address;

    City city;


    public Address(String lane1Address, String lane2Address, City city) {
        this.lane1Address = lane1Address;
        this.lane2Address = lane2Address;
        this.city = city;
    }

    @Override
    public String toString() {
        return "Address{" +
                "lane1Address='" + lane1Address + '\'' +
                ", lane2Address='" + lane2Address + '\'' +
                ", city=" + city +
                '}';
    }
```

```
}

public class Student1 implements Serializable {

    int rollNumber;
    String name;

    Address address;

    public Student1(int rollNumber, String name, Address address) {
        this.rollNumber = rollNumber;
        this.name = name;
        this.address = address;
    }

    public int getRollNumber() {
        return rollNumber;
    }

    public String getName() {
        return name;
    }

    @Override
    public String toString() {
        return "Student1{" +
                "rollNumber=" + rollNumber +
                ", name='" + name + '\'' +
                ", address=" + address +
                '}';
    }
}



import java.io.*;

public class Serialization {

    public static void main(String[] args) throws IOException, ClassNotFoundException {

        Student1 student = new Student1(1, "John", new Address("line-1", "line-2", new City("Pune")));
        serializeStudent(student);


        Student1 student1 = deSerializeStudent();
        System.out.println(student1);

    }

    public static void serializeStudent(Student1 student) throws IOException {
        FileOutputStream fos = new FileOutputStream("john.serialized");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(student);
        oos.flush();
        oos.close();//Closes the stream. This method must be called to release any resources associated with
the stream.
    }

    public static Student1 deSerializeStudent() throws IOException, ClassNotFoundException {
        FileInputStream fos = new FileInputStream("john.serialized");
        ObjectInputStream oos = new ObjectInputStream(fos);
        Student1 john = (Student1) oos.readObject();
        oos.close();//Closes the stream. This method must be called to release any resources associated with
the stream.
        return john;
    }
}
```

```
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;


//from a jar, cannot chnage
class Coller {
    int collerId;

    public Coller(int collerId) {
        this.collerId = collerId;
    }

    public int getCollerId() {
        return collerId;
    }

    @Override
    public String toString() {
        return "Coller{" +
                "collerId='" + collerId + '\'' +
                '}';
    }
}
public class Dog implements Serializable {

    String name;
    transient Coller coller;

    static int dogId = 10;

    public Dog(String name, Coller coller) {
        this.name = name;
        this.coller = coller;
    }

    public String getName() {
        return name;
    }

    public Coller getColler() {
        return coller;
    }


    //callback method,  hook, this will be called by Seriazlier before writing
    private void writeObject(ObjectOutputStream oos) throws IOException {
        oos.defaultWriteObject();
        oos.writeInt(this.coller.collerId);
    }

    //callback method,  hook, this will be called by Seriazlier before reading
    private void readObject(ObjectInputStream ois) throws IOException, ClassNotFoundException {
        ois.defaultReadObject();
        int collerId = ois.readInt();
        this.coller = new Coller(collerId);
    }

    @Override
    public String toString() {
        return "Dog{" +
                "name='" + name + '\'' +
                ", coller=" + coller +
                '}';
    }
}


import java.io.*;
```

```
public class DogSerializableTest {

    public static void main(String[] args) throws IOException, ClassNotFoundException {

        Dog hachiko = new Dog("hachiko", new Coller(100));
        serializeDog(hachiko);

        Dog hachiko1 = deSerializeDog();
        System.out.println(hachiko1);
        //System.out.println(hachiko1.dogId);

    }

    public static void serializeDog(Dog dog) throws IOException {
        FileOutputStream fos = new FileOutputStream("dog.serialized");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(dog);
        oos.flush();
        oos.close();//Closes the stream. This method must be called to release any resources associated with
the stream.
    }

    public static Dog deSerializeDog() throws IOException, ClassNotFoundException {
        FileInputStream fos = new FileInputStream("dog.serialized");
        ObjectInputStream oos = new ObjectInputStream(fos);
        Dog dog = (Dog) oos.readObject();
        oos.close();//Closes the stream. This method must be called to release any resources associated with
the stream.
        return dog;
    }
}
```

Transient variables – The values of the transient variables are never considered (they are excluded from the serialization process). i.e. When we declare a variable transient, after de-serialization its value will always be null, false, or, zero (default value).

Static variables – The values of static variables will not be preserved during the de-serialization process. In-fact static variables are also not serialized but since these belongs to the class. After de-serialization they get their current values from the class.

```
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;


//from a jar, cannot chnage
class Coller {
   int collerId;

   public Coller(int collerId) {
      this.collerId = collerId;
   }

   public int getCollerId() {
      return collerId;
   }

   @Override
   public String toString() {
      return "Coller{" +
```

```
            "collerId='" + collerId + '\'' +
            '}';
    }
}
public class Dog implements Serializable {

    String name;
    transient Coller coller;

    static int dogId;

    public static void setDogId(int id){
        dogId = id;
    }

    public Dog(String name, Coller coller) {
        this.name = name;
        this.coller = coller;
    }

    public String getName() {
        return name;
    }

    public Coller getColler() {
        return coller;
    }


    //callback method,  hook, this will be called by Seriazlier before writing
    private void writeObject(ObjectOutputStream oos) throws IOException {
        oos.defaultWriteObject();
        oos.writeInt(this.coller.collerId);
    }

    //callback method,  hook, this will be called by Seriazlier before reading
    private void readObject(ObjectInputStream ois) throws IOException, ClassNotFoundException {
        ois.defaultReadObject();
        int collerId = ois.readInt();
        this.coller = new Coller(collerId);
    }

    @Override
    public String toString() {
        return "Dog{" +
                "name='" + name + '\'' +
                ", coller=" + coller +
                '}';
    }
}
package com.hdfc.serialization;

import java.io.*;

public class DogSerializableTest {

    public static void main(String[] args) throws IOException, ClassNotFoundException {

        Dog hachiko = new Dog("hachiko", new Coller(100));
        Dog.setDogId(10);
        serializeDog(hachiko);

        Dog.setDogId(100);

        Dog hachiko1 = deSerializeDog();
        System.out.println(hachiko1);
        System.out.println(hachiko1.dogId);

    }
```

```
  public static void serializeDog(Dog dog) throws IOException {
    FileOutputStream fos = new FileOutputStream("dog.serialized");
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.writeObject(dog);
    oos.flush();
    oos.close();//Closes the stream. This method must be called to release any resources associated with the stream.
  }

  public static Dog deSerializeDog() throws IOException, ClassNotFoundException {
    FileInputStream fos = new FileInputStream("dog.serialized");
    ObjectInputStream oos = new ObjectInputStream(fos);
    Dog dog = (Dog) oos.readObject();
    oos.close();//Closes the stream. This method must be called to release any resources associated with the stream.
    return dog;
  }
}
```

Java Serialization is a mechanism provided by the Java programming language to convert Java objects into a byte stream, which can be saved to a file or transmitted over a network, and then reconstructed back into an object when needed. It is mainly used for object persistence, where objects need to be stored and retrieved at a later time.

To make a Java object serializable, it must implement the **java.io.Serializable** interface. This interface acts as a marker, indicating that the object can be serialized. It doesn't have any methods that need to be implemented.

The serialization process is straightforward. To serialize an object, you need to follow these steps:

Create an instance of the java.io.FileOutputStream class to write the byte stream to a file or any other destination.
Create an instance of the java.io.ObjectOutputStream class and pass the FileOutputStream instance to its constructor.
Call the writeObject() method of the ObjectOutputStream class, passing the object you want to serialize as an argument.
Close the streams to release the resources.
Here's an example that demonstrates the serialization process:

```
java
import java.io.*;

public class SerializationExample {
    public static void main(String[] args) {
        // Object to be serialized
        Person person = new Person("John Doe", 25);

        // Serialization
        try {
            FileOutputStream fileOut = new FileOutputStream("person.ser");
            ObjectOutputStream out = new ObjectOutputStream(fileOut);
            out.writeObject(person);
            out.close();
            fileOut.close();
            System.out.println("Object serialized and saved to person.ser");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

class Person implements Serializable {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
```

```
    // Getters and setters (omitted for brevity)
}
```

In the above example, the Person class implements Serializable, indicating that objects of this class can be serialized. The Person object is serialized and saved to a file called "person.ser".

Deserialization is the process of reconstructing an object from the serialized byte stream. Here's an example:

```
java
import java.io.*;

public class DeserializationExample {
    public static void main(String[] args) {
        // Deserialization
        try {
            FileInputStream fileIn = new FileInputStream("person.ser");
            ObjectInputStream in = new ObjectInputStream(fileIn);
            Person person = (Person) in.readObject();
            in.close();
            fileIn.close();
            System.out.println("Object deserialized: " + person.getName() + ", " + person.getAge());
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

In the above example, the Person object is deserialized from the file "person.ser", and its state is printed out.

It's worth noting that Java Serialization has certain security concerns and may not be suitable for all use cases. It's important to consider the implications and alternatives, such as using JSON or other serialization libraries, depending on your specific requirements.