Serialization and Deserialization

Serialization in Java is a mechanism of writing the state of an object into a byte-stream.
The reverse operation of serialization is called *deserialization* where a byte-stream is converted into an object.

Now let us understand seialization with an example:

1. Let us create a class with name, crn number and balance as an attribute whose object needs to be byte-stream.

   Customer.java

```
class Customer{
     private String name;
     private long crn;
     private float balance;
     public Customer(String name, long crn, float
balance) {
          super();
          this.name = name;
          this.crn = crn;
          this.balance = balance;
     }
     public String getName() {
          return name;
     }
     public void setName(String name) {
          this.name = name;
     }
     public long getCrn() {
          return crn;
     }
     public void setCrn(long crn) {
          this.crn = crn;
```

```
    }
    public float getBalance() {
        return balance;
    }
    public void setBalance(float balance) {
        this.balance = balance;
    }

}
```

Above is the class with name, crn number, balance as an instance variable, we have created constructor, setters and getters.

- Now let's create a main method and perform following operations
- Create an object of Customer class
- Store the path of file inside String variable
- To write the object into the file need to Create an object of FileOutputStream and ObjectOutputStream
- Call writeObject() to write the object into the file

```
public class Alpha {
    public static void main(String[] args) {
        Customer customer = new Customer("Alex",
12345678, 20000);
        String path =
"C:\\Users\\Megha\\Desktop\\MyFiles\\object.txt";

        FileOutputStream fos = new
FileOutputStream(path);
        ObjectOutputStream oos = new
ObjectOutputStream(fos);
        oos.writeObject(customer);
        oos.flush();
    }
```

```
}
```

Output:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
	Unhandled exception type FileNotFoundException
	Unhandled exception type IOException
	Unhandled exception type IOException
	Unhandled exception type IOException

	at Alpha.main(Alpha.java:41)
```

As you can see from the above output you are getting **FileNotFoundException, IOException**. Now let's handle these exception using try-catch

```
public class Alpha {
	public static void main(String[] args) {
		Customer customer = new Customer("Alex",
12345678, 20000);
		String path =
"C:\\Users\\Megha\\Desktop\\MyFiles\\object.txt";

		try {
			FileOutputStream fos = new
FileOutputStream(path);
			ObjectOutputStream oos = new
ObjectOutputStream(fos);
			oos.writeObject(customer);
			oos.flush();
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		}
		catch (IOException e) {
			e.printStackTrace();
		}
	}
```

Output:

```
java.io.NotSerializableException: Customer
        at java.base/java.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java:1197)
        at java.base/java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java:354)
        at Alpha.main(Alpha.java:46)
```

After handling **FileNotFoundException, IOException**, you are still getting the exception which is NotSerializableException. It is because whenever you want to write object to the byte-stream that time the **class whose object need to created should implement the Serializable interface**

Now let's **implement serializable interface**

```
class Customer implements Serializable{
     private String name;
     private long crn;
     private float balance;
     public Customer(String name, long crn, float
balance) {
          super();
          this.name = name;
          this.crn = crn;
          this.balance = balance;
     }
     public String getName() {
          return name;
     }
     public void setName(String name) {
          this.name = name;
     }
     public long getCrn() {
          return crn;
     }
     public void setCrn(long crn) {
          this.crn = crn;
     }
     public float getBalance() {
```

```
        return balance;
    }
    public void setBalance(float balance) {
        this.balance = balance;
    }

}
```

Now we have successfully written the object into byte-stream using serialization.

Deserialization

To read the object from the byte stream we need to do deserialization

Steps to do deserialization

- Create an object of FileInputStream
- Create an object ObjectInputStream
- Call readObject() to read the object from byte stream, this will return the object you need to type cast to the type of class and store it in Customer object
- Using the object reference print name, crn number and balance

```
public class Alpha {
    public static void main(String[] args) {
        String path =
"C:\\Users\\Megha\\Desktop\\MyFiles\\object.txt";
        FileInputStream fis = new
FileInputStream(path);
        ObjectInputStream ois = new
ObjectInputStream(fis);
        Customer customer = ois.readObject();
        System.out.println(customer.getName());
        System.out.println(customer.getCrn());
        System.out.println(customer.getBalance());
```

```
    }
}
```

Output:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
	Type mismatch: cannot convert from Object to Customer

	at Alpha.main(Alpha.java:43)
```

Here you have got an exception because you are trying to store the object into the variable of type Customer. Now we need to typecast object to Customer using explicit type casting as shown below

```
Customer customer = (Customer)ois.readObject();
```

Output:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
	Unhandled exception type FileNotFoundException
	Unhandled exception type IOException
	Unhandled exception type IOException
	Unhandled exception type ClassNotFoundException

	at Alpha.main(Alpha.java:41)
```

As you can see from the above output you have got FileNotFoundException, IOException, ClassNotFound Exception now to resolve we need to handle these exceptions using try-catch as shown below

```
public class Alpha {
    public static void main(String[] args) {
        String path =
"C:\\Users\\Megha\\Desktop\\MyFiles\\object.txt";
        try {
            FileInputStream fis = new
FileInputStream(path);
```

```
            ObjectInputStream ois = new
ObjectInputStream(fis);
            Customer customer =
(Customer)ois.readObject();
            System.out.println(customer.getName());
            System.out.println(customer.getCrn());
            System.out.println(customer.getBalance());
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```
Alex
12345678
20000.0
```

As you can see from the above output we have successfully read the data from the file

Serialversionuid:

SerialVersionUID is used to ensure that during deserialization the same class (that was used during serialisation process) is loaded.

At the time of serialization, with every object sender side JVM will save a **Unique Identifier**. JVM is responsible to generate that unique ID based on the corresponding .class file which is present in the sender system.
Deserialization at the time of deserialization, receiver side JVM will compare the unique ID associated with the Object with local class Unique ID i.e. JVM will also create a Unique ID based on the corresponding .class file which is present in the receiver system. If both unique ID matched then only deserialization will be performed. Otherwise, we will get a Runtime Exception saying InvalidClassException. This unique Identifier is nothing but **SerialVersionUID**.

A serializable class can declare its own serialVersionUID explicitly by declaring a field named "serialVersionUID" that must be static, final, and of type long:

Syntax:
private static final long SerialVersionUID=10l;

Like this you can avoid InvalidClassException