In this example, we will use **ObjectOutputStream** class to write employee object to file.

* To write or send the object to the external world the object must implement java.io.Serializable interface type.
* It means this object's class must be a subclass of a *java.io.Serializable* interface, else *write()*method throws*java.io.NotSerializableException*.
* The method *writeObject* is used to write an object to the stream. Any object, including Strings and arrays, is written with writeObject. Multiple objects or primitives can be written to the stream. The objects must be read back from the corresponding ObjectInputstream with the same types and in the same order as they were written.

We can also write String, Arrays, Integer, and date to file using ObjectOutputStream class because these classes internally implement a *java.io.Serializable* interface.

**2. Write an Object to File Example**

Let's first create *Employee* class and which implements*a java.io.Serializable* interface.

class Employee implements Serializable {

private static final long serialVersionUID = 1L;

private int id;

private String name;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

Let's create *an employees.txt* file under some directory. Write code to write employee object to file *employees.txt*.

/\*\*

\* This Java program demonstrates how to write object in file.

\* @author javaguides.net

\*/

public class ObjectOutputStreamExample {

public static void main(String[] args) {

final Employee employee = new Employee();

employee.setId(100);

employee.setName("ramesh");

try (final FileOutputStream fout = new FileOutputStream("employees.txt");

final ObjectOutputStream out = new ObjectOutputStream(fout)) {

out.writeObject(employee);

out.flush();

System.out.println("success");

} catch (IOException e) {

e.printStackTrace();

}

}

}

In this example, we will use **ObjectInputStream** class to read employee object to file.

The deserialization process is quite similar with the serialization, you need to use **ObjectInputStream** to read the content of the file and convert it back to a Java object.

We can also read *String, Arrays, Integer, and Date* from a file using **ObjectInputStream** class because these classes internally implement *the java.io.Serializable* interface.

**2. Read Object from File Example**

1. Let's first create Employee class and which implements the *java.io.Serializable*interface.

class Employee implements Serializable {

private static final long serialVersionUID = 1L;

private int id;

private String name;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

Let'*employees.txt* file under some directory write employee object into a file using *ObjectOutputStream* class.

/\*\*

\* This Java program demonstrates how to read object from file.

\* @author javaguides.net

\*/

public class ObjectInputStreamExample {

public static void main(String[] args) {

try (ObjectInputStream in = new ObjectInputStream(new FileInputStream("employees.txt"))) {

final Employee employee = (Employee) in.readObject();

System.out.println(" printing employee object details");

System.out.println(employee.getId() + " " + employee.getName());

System.out.println(" printing address object details");

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

}

}

}

Output:

printing employee object details

100 ramesh

printing address