Serialization in Java

Yulia Newton, Ph.D.

CS151, Object Oriented Programming

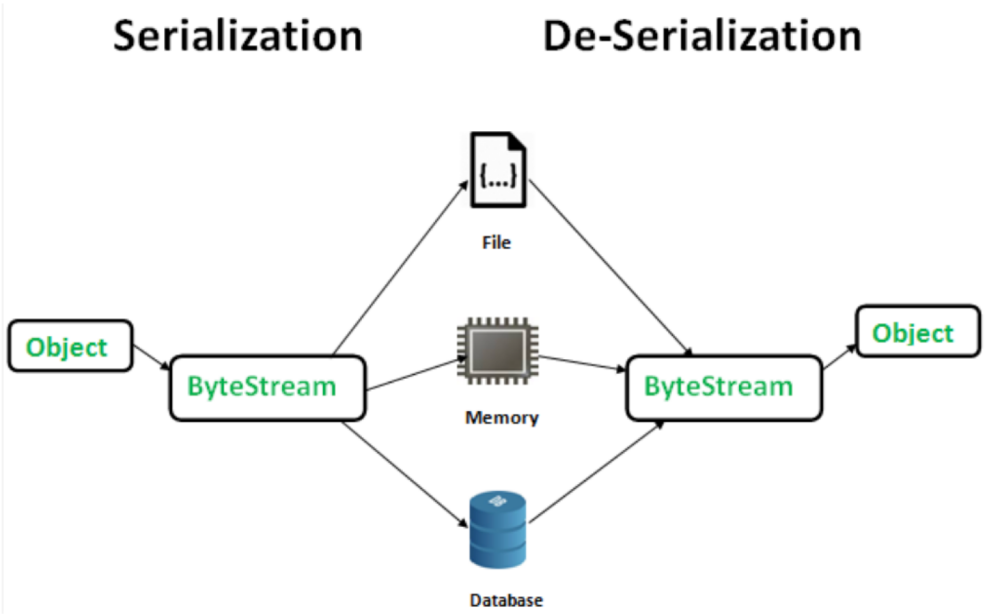
San Jose State University

Spring 2020

Introduction to serialization

- Serialization is a process of converting the state of an object to a byte stream
  - Sequence of bytes
- Deserialization is a reverse process
- The byte stream can be saved to a number of destinations (e.g. file, memory, database)
  - Byte stream is platform independent
  - Objects serialized on one platform can be deserialized on another platform

Serialization at a glance



https://www.geeksforgeeks.org/

Motivation for serialization

- When an object of a given type is created its lifetime is at most for the duration of the program execution
  - Once the program is terminated the object and its state are lost
- What if we want to re-use this object in exact state it was in without re-creating it?
- What if we want to preserve the object and its state for other purposes?
  - Debugging
  - Logging
  - Reuse in other programs
  - Reuse in other executions

Serialization with Java

- Two conditions for an object to be serializable:
  - Types eligible for serialization must implement *Serializable* interface
    - *java.io.Serializable*
    - Marker interface
  - All the fields in the class must be serializable
    - Non-serializable fields are marked as *transient*
    - Check API and other documentation to see if a built-in or third party types are serializable

- *ObjectOutputStream* and *ObjectInputStream* classes contain methods for serializing and deserializing objects
  - High level representations of streams
  - Call *writeObject()* in *ObjectOutputStream* class
    - *public final void writeObject(Object x) throws IOException*
  - Call *readObject()* in *ObjectInputStream* class
    - *public final Object readObject() throws IOException, ClassNotFoundException*

Example: serializing an employee object to file

```java
import java.io.*;

class Employee implements Serializable {
    private Integer id;
    private String name;
    private int age;
    private transient String ssn;

    Employee(String name, int age, Integer id) {
        this.name = name;
        this.age = age;
        this.id = id;
    }

    public void setSSN(String ssn){this.ssn = ssn;}
    public String getSSN(){return this.ssn;}
    public String getName(){return this.name;}
    public Integer getAge(){return this.age;}
    public Integer getID(){return this.id;}
    public String toString() {
        return this.name+" is "+this.age+"yo (id = "+this.id+")";
    }
}
```

we do not want to be able to save SSN field

```java
public class Test {
    public static void main(String[]args) {
        Employee empl = new Employee("John Smith", 35, 101);
        empl.setSSN("000-11-4444");

        try {
            FileOutputStream streamOut = new FileOutputStream("./employee.ser");
            ObjectOutputStream objectOutput = new ObjectOutputStream(streamOut);
            objectOutput.writeObject(empl);
            objectOutput.close();
            streamOut.close();
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

file to which we wish to serialize

An instance of ObjectOutputStream is used to serialize an Employee object by calling writeObject() method; a FileOutputStream object is used to write the stream to a file

Deserialization of an object

- Can be done from the same program or from another program or from another run of the same program
- Constructor of the class is not called when the object is deserialized
- All fields of the class must be serializable
- Fields declared as transient are null after deserialization

Example: deserializing an employee object from file

```java
import java.io.*;

class Employee implements Serializable {
    private Integer id;
    private String name;
    private int age;
    private transient String ssn;

    Employee(String name, int age, Integer id) {
        this.name = name;
        this.age = age;
        this.id = id;
    }

    public void setSSN(String ssn){this.ssn = ssn;}
    public String getSSN(){return this.ssn;}
    public String getName(){return this.name;}
    public Integer getAge(){return this.age;}
    public Integer getID(){return this.id;}
    public String toString() {
        return this.name+" is "+this.age+"yo (id = "+this.id+")";
    }
}
```

Output:

```
Deserialized Employee object:
Name: John Smith
Age: 35
SSN: null
ID: 101
```

SSN field was not serialized

```java
public class Test {
    public static void main(String[]args) {
        Employee empl = new Employee("John Smith", 35, 101);
        empl.setSSN("000-11-4444");

        try {
            FileOutputStream streamOut = new FileOutputStream("./employee.ser");
            ObjectOutputStream objectOutput = new ObjectOutputStream(streamOut);
            objectOutput.writeObject(empl);
            objectOutput.close();
            streamOut.close();
        } catch (IOException e) {
            System.out.println(e);
        }

        Employee empl2 = null;
        try {
            FileInputStream streamIn = new FileInputStream("./employee.ser");
            ObjectInputStream objectInput = new ObjectInputStream(streamIn);
            empl2 = (Employee) objectInput.readObject();
            objectInput.close();
            streamIn.close();
        } catch (IOException e) {
            System.out.println(e);
            return;
        } catch (ClassNotFoundException c) {
        System.out.println("Employee class not found");
        c.printStackTrace();
        return;
        }

      System.out.println("Deserialized Employee object:");
      System.out.println("Name: " + empl2.getName());
      System.out.println("Age: " + empl2.getAge());
      System.out.println("SSN: " + empl2.getSSN());
      System.out.println("ID: " + empl2.getID());
    }
}
```

cast the return object from *readObject()* method as Employee type

thrown if JVM cannot find the bytecode for the object

A few notes about serializing to a file

- No requirements for a file name extension
  - E.g. “ser” or “txt” or “obj” or “dat” file types
- Remember the lecture on IO programming?
  - Can use any of the IO approaches to save the object to the file system

Serialization with aggregation (“has-a” relationship)

- In order for an object to be successfully serialized, all fields in the class have to be serializable
  - *NotSerializableException* runtime exception is thrown
- Ways to resolve this issue
  - Make the field serializable
  - Declare the field as *transient* to prevent its serialization

Example: serializing an object when not all fields are serializable

```java
import java.io.*;

class Address {
    private int streetNum;
    private String streetName;
    private String city;
    private String state;
    private int zip;

    Address(int streetNum, String streetName, String city, String state, int zip){
        this.streetNum = streetNum;
        this.streetName = streetName;
        this.city = city;
        this.state = state;
        this.zip = zip;
    }

    public String toString(){
        return this.streetNum+" "+this.streetName+", "+this.city+" "+this.state+", "+this.zip;
    }
}

class Employee implements Serializable {
    private Integer id;
    private String name;
    private int age;
    private transient String ssn;
    private Address address;

    Employee(String name, int age, Integer id, Address address) {
        this.name = name;
        this.age = age;
        this.id = id;
        this.address = address;
    }

    public void setSSN(String ssn){this.ssn = ssn;}
    public String getSSN(){return this.ssn;}
    public String getName(){return this.name;}
    public Integer getAge(){return this.age;}
    public Integer getID(){return this.id;}
    public Address getAddress(){return this.address;}
    public String toString() {
        return this.name+" is "+this.age+"yo (id = "+this.id+")\n"+this.address;
    }
}
```

```java
public class Test {
    public static void main(String[]args) {
        Employee empl = new Employee("John Smith", 35, 101, new Address(10,"Main St","MyTown","CA",91111));
        empl.setSSN("000-11-4444");

        try {
            FileOutputStream streamOut = new FileOutputStream("./employee.ser");
            ObjectOutputStream objectOutput = new ObjectOutputStream(streamOut);
            objectOutput.writeObject(empl);
            objectOutput.close();
            streamOut.close();
        } catch (IOException e) {
            System.out.println(e);
        }

        Employee empl2 = null;
        try {
            FileInputStream streamIn = new FileInputStream("./employee.ser");
            ObjectInputStream objectInput = new ObjectInputStream(streamIn);
            empl2 = (Employee) objectInput.readObject();
            objectInput.close();
            streamIn.close();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println(e);
            return;
        }

      System.out.println("Deserialized Employee object:");
      System.out.println("Name: " + empl2.getName());
      System.out.println("Age: " + empl2.getAge());
      System.out.println("SSN: " + empl2.getSSN());
      System.out.println("ID: " + empl2.getID());
      System.out.println("Address: " + empl2.getAddress());
    }
}
```

Output: `java.io.WriteAbortedException: writing aborted; java.io.NotSerializableException: Address`

Example: declaring the Address field as *transient* prevents exception from being thrown

```java
import java.io.*;

class Address {
    private int streetNum;
    private String streetName;
    private String city;
    private String state;
    private int zip;

    Address(int streetNum, String streetName, String city, String state, int zip){
        this.streetNum = streetNum;
        this.streetName = streetName;
        this.city = city;
        this.state = state;
        this.zip = zip;
    }

    public String toString(){
        return this.streetNum+" "+this.streetName+", "+this.city+" "+this.state+", "+this.zip;
    }
}

class Employee implements Serializable {
    private Integer id;
    private String name;
    private int age;
    private transient String ssn;
    private transient Address address;

    Employee(String name, int age, Integer id, Address address) {
        this.name = name;
        this.age = age;
        this.id = id;
        this.address = address;
    }

    public void setSSN(String ssn){this.ssn = ssn;}
    public String getSSN(){return this.ssn;}
    public String getName(){return this.name;}
    public Integer getAge(){return this.age;}
    public Integer getID(){return this.id;}
    public Address getAddress(){return this.address;}
    public String toString() {
        return this.name+" is "+this.age+"yo (id = "+this.id+")\n"+this.address;
    }
}
```

Address field is *transient*

```java
public class Test {
    public static void main(String[]args) {
        Employee empl = new Employee("John Smith", 35, 101, new Address(10,"Main St","MyTown","CA",91111));
        empl.setSSN("000-11-4444");

        try {
            FileOutputStream streamOut = new FileOutputStream("./employee.ser");
            ObjectOutputStream objectOutput = new ObjectOutputStream(streamOut);
            objectOutput.writeObject(empl);
            objectOutput.close();
            streamOut.close();
        } catch (IOException e) {
            System.out.println(e);
        }

        Employee empl2 = null;
        try {
            FileInputStream streamIn = new FileInputStream("./employee.ser");
            ObjectInputStream objectInput = new ObjectInputStream(streamIn);
            empl2 = (Employee) objectInput.readObject();
            objectInput.close();
            streamIn.close();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println(e);
            return;
        }

      System.out.println("Deserialized Employee object:");
      System.out.println("Name: " + empl2.getName());
      System.out.println("Age: " + empl2.getAge());
      System.out.println("SSN: " + empl2.getSSN());
      System.out.println("ID: " + empl2.getID());
      System.out.println("Address: " + empl2.getAddress());
    }
}
```

Output:

```
Deserialized Employee object:
Name: John Smith
Age: 35
SSN: null
ID: 101
Address: null
```

Example: serializing Address field

```
import java.io.*;

class Address implements Serializable {
    private int streetNum;
    private String streetName;
    private String city;
    private String state;
    private int zip;

    Address(int streetNum, String streetName, String city, String state, int zip){
        this.streetNum = streetNum;
        this.streetName = streetName;
        this.city = city;
        this.state = state;
        this.zip = zip;
    }

    public String toString(){
        return this.streetNum+" "+this.streetName+", "+this.city+" "+this.state+", "+this.zip;
    }
}

class Employee implements Serializable {
    private Integer id;
    private String name;
    private int age;
    private transient String ssn;
    private Address address;

    Employee(String name, int age, Integer id, Address address) {
        this.name = name;
        this.age = age;
        this.id = id;
        this.address = address;
    }

    public void setSSN(String ssn){this.ssn = ssn;}
    public String getSSN(){return this.ssn;}
    public String getName(){return this.name;}
    public Integer getAge(){return this.age;}
    public Integer getID(){return this.id;}
    public Address getAddress(){return this.address;}
    public String toString() {
        return this.name+" is "+this.age+"yo (id = "+this.id+")\n"+this.address;
    }
}
```

making Address class serializable allows us to serialize its instance with the Employee object

```
public class Test {
    public static void main(String[]args) {
        Employee empl = new Employee("John Smith", 35, 101, new Address(10,"Main St","MyTown","CA",91111));
        empl.setSSN("000-11-4444");

        try {
            FileOutputStream streamOut = new FileOutputStream("./employee.ser");
            ObjectOutputStream objectOutput = new ObjectOutputStream(streamOut);
            objectOutput.writeObject(empl);
            objectOutput.close();
            streamOut.close();
        } catch (IOException e) {
            System.out.println(e);
        }

        Employee empl2 = null;
        try {
            FileInputStream streamIn = new FileInputStream("./employee.ser");
            ObjectInputStream objectInput = new ObjectInputStream(streamIn);
            empl2 = (Employee) objectInput.readObject();
            objectInput.close();
            streamIn.close();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println(e);
            return;
        }

      System.out.println("Deserialized Employee object:");
      System.out.println("Name: " + empl2.getName());
      System.out.println("Age: " + empl2.getAge());
      System.out.println("SSN: " + empl2.getSSN());
      System.out.println("ID: " + empl2.getID());
      System.out.println("Address: " + empl2.getAddress());
    }
}
```

Output:

```
Deserialized Employee object:
Name: John Smith
Age: 35
SSN: null
ID: 101
Address: 10 Main St, MyTown CA, 91111
```

empl2 object has deserialized empl object's address

Serialization with inheritance (“is-a” relationship)

- If a class is serializable then all its descendants are as well
  - Children
  - Children of children

Example: Employee inherits serialization from Person

```java
import java.io.*;

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

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

    public void setSSN(String ssn){this.ssn = ssn;}
    public String getSSN(){return this.ssn;}
    public String getName(){return this.name;}
    public Integer getAge(){return this.age;}
    public String toString(){
        return this.name+" is "+this.age+"yo";
    }
}

class Employee extends Person {
    private Integer id;

    Employee(String name, int age, Integer id, String ssn) {
        super(name, age, ssn);
        this.id = id;
    }

    public void setSSN(String ssn){super.setSSN(ssn);}
    public String getSSN(){return super.getSSN();}
    public String getName(){return super.getName();}
    public Integer getAge(){return super.getAge();}
    public Integer getID(){return this.id;}
    public String toString() {
        return super.toString()+" (id = "+this.id+")";
    }
}
```

```java
public class Test {
    public static void main(String[]args) {
        Employee empl = new Employee("John Smith", 35, 101,"111-11-1111","CS");
        empl.setSSN("000-11-4444");

        try {
            FileOutputStream streamOut = new FileOutputStream("./employee.ser");
            ObjectOutputStream objectOutput = new ObjectOutputStream(streamOut);
            objectOutput.writeObject(empl);
            objectOutput.close();
            streamOut.close();
        } catch (IOException e) {
            System.out.println(e);
        }

        Employee empl2 = null;
        try {
            FileInputStream streamIn = new FileInputStream("./employee.ser");
            ObjectInputStream objectInput = new ObjectInputStream(streamIn);
            empl2 = (Employee) objectInput.readObject();
            objectInput.close();
            streamIn.close();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println(e);
            return;
        }

      System.out.println("Deserialized Employee object:");
      System.out.println("Name: " + empl2.getName());
      System.out.println("Age: " + empl2.getAge());
      System.out.println("SSN: " + empl2.getSSN());
      System.out.println("ID: " + empl2.getID());
      System.out.println("Department: " + empl2.getDepartment());
    }
}
```

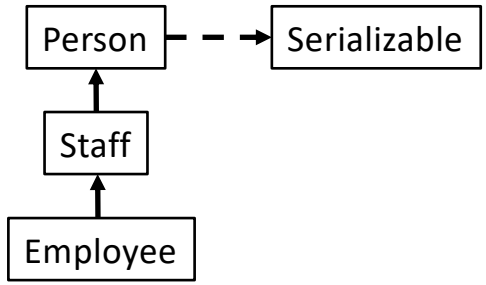
Output:

```
Deserialized Employee object:
Name: John Smith
Age: 35
SSN: null
ID: 101
```

Example: more than one level of inheritance of serializable class

```java
import java.io.*;

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

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

    public void setSSN(String ssn){this.ssn = ssn;}
    public String getSSN(){return this.ssn;}
    public String getName(){return this.name;}
    public Integer getAge(){return this.age;}
    public String toString(){
        return this.name+" is "+this.age+"yo";
    }
}

class Staff extends Person {
    private Integer id;

    Staff(String name, int age, Integer id, String ssn) {
        super(name, age, ssn);
        this.id = id;
    }

    public void setSSN(String ssn){super.setSSN(ssn);}
    public String getSSN(){return super.getSSN();}
    public String getName(){return super.getName();}
    public Integer getAge(){return super.getAge();}
    public Integer getID(){return this.id;}
    public String toString() {
        return super.toString()+" (id = "+this.id+")";
    }
}
```

```java
class Employee extends Staff {
    private String department;

    Employee(String name, int age, Integer id, String ssn, String department) {
        super(name, age, id, ssn);
        this.department = department;
    }

    public void setSSN(String ssn){super.setSSN(ssn);}
    public String getSSN(){return super.getSSN();}
    public String getName(){return super.getName();}
    public Integer getAge(){return super.getAge();}
    public Integer getID(){return super.getID();}
    public String getDepartment(){return this.department;}
    public String toString() {
        return super.toString()+", works in "+this.department+" department";
    }
}
```



Example: more than one level of inheritance of serializable class (cont'd)

```
public class Test {
    public static void main(String[]args) {
        Employee empl = new Employee("John Smith", 35, 101,"111-11-1111","CS");
        empl.setSSN("000-11-4444");

        try {
            FileOutputStream streamOut = new FileOutputStream("./employee.ser");
            ObjectOutputStream objectOutput = new ObjectOutputStream(streamOut);
            objectOutput.writeObject(empl);
            objectOutput.close();
            streamOut.close();
        } catch (IOException e) {
            System.out.println(e);
        }

        Employee empl2 = null;
        try {
            FileInputStream streamIn = new FileInputStream("./employee.ser");
            ObjectInputStream objectInput = new ObjectInputStream(streamIn);
            empl2 = (Employee) objectInput.readObject();
            objectInput.close();
            streamIn.close();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println(e);
            return;
        }

      System.out.println("Deserialized Employee object:");
      System.out.println("Name: " + empl2.getName());
      System.out.println("Age: " + empl2.getAge());
      System.out.println("SSN: " + empl2.getSSN());
      System.out.println("ID: " + empl2.getID());
      System.out.println("Department: " + empl2.getDepartment());
    }
}
```

Output:

```
Deserialized Employee object:
Name: John Smith
Age: 35
SSN: null
ID: 101
Department: CS
```

Serialization of static fields

- Static members of serializable classes are not serializable
  - Static members belong to the class, not an instance

Example: static fields are not serialized

```
import java.io.*;

class Person implements Serializable {
    private String name;
    private int age;
    private transient String ssn;
    public static String species;

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

    public void setSSN(String ssn){this.ssn = ssn;}
    public String getSSN(){return this.ssn;}
    public String getName(){return this.name;}
    public Integer getAge(){return this.age;}
    public String toString(){
        return this.name+" is "+this.age+"yo";
    }
}

class Staff extends Person {
    private Integer id;

    Staff(String name, int age, Integer id, String ssn) {
        super(name, age, ssn);
        this.id = id;
    }

    public void setSSN(String ssn){super.setSSN(ssn);}
    public String getSSN(){return super.getSSN();}
    public String getName(){return super.getName();}
    public Integer getAge(){return super.getAge();}
    public Integer getID(){return this.id;}
    public String toString() {
        return super.toString()+" (id = "+this.id+")";
    }
}
```

static field

```
class Employee extends Staff {
    private String department;

    Employee(String name, int age, Integer id, String ssn, String department) {
        super(name, age, id, ssn);
        this.department = department;
    }

    public void setSSN(String ssn){super.setSSN(ssn);}
    public String getSSN(){return super.getSSN();}
    public String getName(){return super.getName();}
    public Integer getAge(){return super.getAge();}
    public Integer getID(){return super.getID();}
    public String getDepartment(){return this.department;}
    public String toString() {
        return super.toString()+", works in "+this.department+" department";
    }
}
```

Example: static fields are not serialized (cont'd)

```java
public class Test {
    public static void main(String[]args) {
        Employee empl = new Employee("John Smith", 35, 101,"111-11-1111","CS");
        empl.setSSN("000-11-4444");
        empl.species = "Homo sapiens";

        try {
            FileOutputStream streamOut = new FileOutputStream("./employee.ser");
            ObjectOutputStream objectOutput = new ObjectOutputStream(streamOut);
            objectOutput.writeObject(empl);
            objectOutput.close();
            streamOut.close();
        } catch (IOException e) {
            System.out.println(e);
        }

        Employee empl2 = null;
        try {
            FileInputStream streamIn = new FileInputStream("./employee.ser");
            ObjectInputStream objectInput = new ObjectInputStream(streamIn);
            empl2 = (Employee) objectInput.readObject();
            objectInput.close();
            streamIn.close();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println(e);
            return;
        }

      System.out.println("Deserialized Employee object:");
      System.out.println("Name: " + empl2.getName());
      System.out.println("Age: " + empl2.getAge());
      System.out.println("SSN: " + empl2.getSSN());
      System.out.println("ID: " + empl2.getID());
      System.out.println("Department: " + empl2.getDepartment());
      System.out.println("Species: " + empl2.species);

      System.out.println("\nAfter changing species");
      empl.species = "Beter humans";
      System.out.println("Species: " + empl2.species);
    }
}
```

Output:

```
Deserialized Employee object:
Name: John Smith
Age: 35
SSN: null
ID: 101
Department: CS
Species: Homo sapiens

After changing species
Species: Beter humans
```

species field has not been serialized; it has a value because it has been initialized for all instances of Employee class; if this field had not been initialized before it would be null here

to demonstrate that we change this field in empl instance and see it reflected in empl2 instance

Serialization of collections or arrays

- All objects of the collection or array must be serializable
- Serialization will fail if at least one of the elements is not serializable

Example: serializing an ArrayList of animals

```
import java.io.*;

class Animal implements Serializable {
    private int weight;
    private int numLegs;
    private String name;

    Animal(String name, int numLegs, int weight){
        this.weight = weight;
        this.numLegs = numLegs;
        this.name = name;
    }

    public int getWeight(){return this.weight;}
    public int getLegs(){return this.numLegs;}
    public String getName(){return this.name;}
}

class Dog extends Animal {
    private String breed;

    Dog(String breed, String name, int weight, int numLegs){
        super(name, numLegs, weight);
        this.breed = breed;
    }

    public int getWeight(){return super.getWeight();}
    public int getLegs(){return super.getLegs();}
    public String getName(){return super.getName();}
    public String getBreed(){return this.breed;}
    public String toString(){
        return "Dog: "+super.getName()+", "+this.breed+", "+
                super.getLegs()+" legs, "+super.getWeight()+" lbs";
    }
}
```

```
class Cat extends Animal {
    private String type;

    Cat(String type, String name, int weight, int numLegs){
        super(name, numLegs, weight);
        this.type = type;
    }

    public int getWeight(){return super.getWeight();}
    public int getLegs(){return super.getLegs();}
    public String getName(){return super.getName();}
    public String getType(){return this.type;}
    public String toString(){
        return "Cat: "+super.getName()+", "+this.type+", "+
                super.getLegs()+" legs, "+super.getWeight()+" lbs";
    }
}

class Racoon extends Animal{
    private boolean albino;

    Racoon(boolean albino, String name, int weight, int numLegs){
        super(name, numLegs, weight);
        this.albino = albino;
    }

    public int getWeight(){return super.getWeight();}
    public int getLegs(){return super.getLegs();}
    public String getName(){return super.getName();}
    public boolean getAlbino(){return this.albino;}
    public String toString(){
        if(this.albino){
            return "Racoon: "+super.getName()+", albino, "+
                super.getLegs()+" legs, "+super.getWeight()+" lbs";
        }else{
            return "Racoon: "+super.getName()+", "+
                super.getLegs()+" legs, "+super.getWeight()+" lbs";
        }
    }
}
```

Example: serializing an ArrayList of animals (cont'd)

```
public class Test {
    public static void main(String[]args) {
        List <Animal> myAnimals = new ArrayList <Animal>();
        myAnimals.add(new Dog("Mastiff","Joey",150,4));
        myAnimals.add(new Cat("domestic","Princess",20,4));
        myAnimals.add(new Racoon(false,"Jamie",30,4));
        myAnimals.add(new Dog("Chihuahua","Sparkey",10,3));
        myAnimals.add(new Cat("wild","Tracy",15,4));
        myAnimals.add(new Dog("German Shepherd","Tom",50,4));
        myAnimals.add(new Racoon(true,"Johny",25,4));

        try {
            FileOutputStream streamOut = new FileOutputStream("./animals.ser");
            ObjectOutputStream objectOutput = new ObjectOutputStream(streamOut);
            objectOutput.writeObject(myAnimals);
            objectOutput.close();
            streamOut.close();
        } catch (IOException e) {
            System.out.println(e);
        }

        List <Animal> myAnimals2 = new ArrayList <Animal>();
        try {
            FileInputStream streamIn = new FileInputStream("./animals.ser");
            ObjectInputStream objectInput = new ObjectInputStream(streamIn);
            myAnimals2 = (ArrayList) objectInput.readObject();
            objectInput.close();
            streamIn.close();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println(e);
            return;
        }

        for(Animal a: myAnimals2){
            System.out.println(a);
        }
    }
}
```

instantiate ArrayList and add elements; all elements inherit from Animal class, which is serializable

cast as ArrayList

Output:

```
Dog: Joey, Mastiff, 4 legs, 150 lbs
Cat: Princess, domestic, 4 legs, 20 lbs
Racoon: Jamie, 4 legs, 30 lbs
Dog: Sparkey, Chihuahua, 3 legs, 10 lbs
Cat: Tracy, wild, 4 legs, 15 lbs
Dog: Tom, German Shepherd, 4 legs, 50 lbs
Racoon: Johny, albino, 4 legs, 25 lbs
```

Example: not all elements of ArrayList are serializable

```
import java.io.*;

class Animal {
    private int weight;
    private int numLegs;
    private String name;

    Animal(String name, int numLegs, int weight){
        this.weight = weight;
        this.numLegs = numLegs;
        this.name = name;
    }

    public int getWeight(){return this.weight;}
    public int getLegs(){return this.numLegs;}
    public String getName(){return this.name;}
}

class Dog extends Animal implements Serializable {
    private String breed;

    Dog(String breed, String name, int weight, int numLegs){
        super(name, numLegs, weight);
        this.breed = breed;
    }

    public int getWeight(){return super.getWeight();}
    public int getLegs(){return super.getLegs();}
    public String getName(){return super.getName();}
    public String getBreed(){return this.breed;}
    public String toString(){
        return "Dog: "+super.getName()+", "+this.breed+", "+
                super.getLegs()+" legs, "+super.getWeight()+" lbs";
    }
}
```

Output: `java.io.NotSerializableException: Racoon`

```
class Cat extends Animal implements Serializable {
    private String type;

    Cat(String type, String name, int weight, int numLegs){
        super(name, numLegs, weight);
        this.type = type;
    }

    public int getWeight(){return super.getWeight();}
    public int getLegs(){return super.getLegs();}
    public String getName(){return super.getName();}
    public String getType(){return this.type;}
    public String toString(){
        return "Cat: "+super.getName()+", "+this.type+", "+
                super.getLegs()+" legs, "+super.getWeight()+" lbs";
    }
}

class Racoon extends Animal{
    private boolean albino;

    Racoon(boolean albino, String name, int weight, int numLegs){
        super(name, numLegs, weight);
        this.albino = albino;
    }

    public int getWeight(){return super.getWeight();}
    public int getLegs(){return super.getLegs();}
    public String getName(){return super.getName();}
    public boolean getAlbino(){return this.albino;}
    public String toString(){
        if(this.albino){
            return "Racoon: "+super.getName()+", albino, "+
                super.getLegs()+" legs, "+super.getWeight()+" lbs";
        }else{
            return "Racoon: "+super.getName()+", "+
                super.getLegs()+" legs, "+super.getWeight()+" lbs";
        }
    }
}
```

SerialVersionUID

- SerialVersionUID is an ID associated with each serializable class
  - Association is created at runtime
  - Automatically generated for each serializable class by JVM
  - Ensures compatibility of the objects
- Used to verify that both the sender and receiver have the same ID associated with the class
  - InvalidClassException is thrown if not a match
  - Sender = the party that serializes the object
  - Receiver = the party that deserializes the object
- Can declare our own SerialVersionUID
  - Must be a long type
  - Must be static and final
  - Name the field SerialVersionUID
    - *private static final long serialVersionUID=1L;*

Example: using *serialVersionUID* field

```java
import java.io.*;

class Employee implements Serializable {
    private Integer id;
    private String name;
    private int age;
    private transient String ssn;
    private static final long serialVersionUID = 42L;

    Employee(String name, int age, Integer id) {
        this.name = name;
        this.age = age;
        this.id = id;
    }

    public void setSSN(String ssn){this.ssn = ssn;}
    public String getSSN(){return this.ssn;}
    public String getName(){return this.name;}
    public Integer getAge(){return this.age;}
    public Integer getID(){return this.id;}
    public String toString() {
        return this.name+" is "+this.age+"yo (id = "+this.id+")";
    }
}
```

serialVersionUID field

```java
public class Test {
    public static void main(String[]args) {
        Employee empl = new Employee("John Smith", 35, 101);
        empl.setSSN("000-11-4444");

        try {
            FileOutputStream streamOut = new FileOutputStream("./employee.ser");
            ObjectOutputStream objectOutput = new ObjectOutputStream(streamOut);
            objectOutput.writeObject(empl);
            objectOutput.close();
            streamOut.close();
        } catch (IOException e) {
            System.out.println(e);
        }

        Employee empl2 = null;
        try {
            FileInputStream streamIn = new FileInputStream("./employee.ser");
            ObjectInputStream objectInput = new ObjectInputStream(streamIn);
            empl2 = (Employee) objectInput.readObject();
            objectInput.close();
            streamIn.close();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println(e);
            return;
        }

      System.out.println("Deserialized Employee object:");
      System.out.println("Name: " + empl2.getName());
      System.out.println("Age: " + empl2.getAge());
      System.out.println("SSN: " + empl2.getSSN());
      System.out.println("ID: " + empl2.getID());
    }
}
```

Example: *serialVersionUID* field mismatch

Code that serializes an Employee object:

```
import java.io.*;

class Employee implements Serializable {
    private Integer id;
    private String name;
    private int age;
    private transient String ssn;
    private static final long serialVersionUID = 42L;

    Employee(String name, int age, Integer id) {
        this.name = name;
        this.age = age;
        this.id = id;
    }

    public void setSSN(String ssn){this.ssn = ssn;}
    public String getSSN(){return this.ssn;}
    public String getName(){return this.name;}
    public Integer getAge(){return this.age;}
    public Integer getID(){return this.id;}
    public String toString() {
        return this.name+" is "+this.age+"yo (id = "+this.id+")";
    }
}

public class Test {
    public static void main(String[]args) {
        Employee empl = new Employee("John Smith", 35, 101);
        empl.setSSN("000-11-4444");

        try {
            FileOutputStream streamOut = new FileOutputStream("./employee.ser");
            ObjectOutputStream objectOutput = new ObjectOutputStream(streamOut);
            objectOutput.writeObject(empl);
            objectOutput.close();
            streamOut.close();
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

Code that deserializes an Employee object but has a different serialVersionUID value:

```
import java.io.*;

class Employee implements Serializable {
    private Integer id;
    private String name;
    private int age;
    private transient String ssn;
    private static final long serialVersionUID = 43L;

    Employee(String name, int age, Integer id) {
        this.name = name;
        this.age = age;
        this.id = id;
    }

    public void setSSN(String ssn){this.ssn = ssn;}
    public String getSSN(){return this.ssn;}
    public String getName(){return this.name;}
    public Integer getAge(){return this.age;}
    public Integer getID(){return this.id;}
    public String toString() {
        return this.name+" is "+this.age+"yo (id = "+this.id+")";
    }
}

public class Test {
    public static void main(String[]args) {
        Employee empl2 = null;
        try {
            FileInputStream streamIn = new FileInputStream("./employee.ser");
            ObjectInputStream objectInput = new ObjectInputStream(streamIn);
            empl2 = (Employee) objectInput.readObject();
            objectInput.close();
            streamIn.close();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println(e);
            return;
        }

      System.out.println("Deserialized Employee object:");
      System.out.println("Name: " + empl2.getName());
      System.out.println("Age: " + empl2.getAge());
      System.out.println("SSN: " + empl2.getSSN());
      System.out.println("ID: " + empl2.getID());
    }
}
```

Output:

```
java.io.InvalidClassException: Employee; local class incompatible: stream classdesc serialVersionUID = 42, local class serialVersionUID = 43
```

Custom serialization

- Normally we rely on built-in functionality to perform serialization and deserialization in Java
- Custom serialization can be useful when we want to serialize an object with some unserializable attributes
- Custom serialization can be achieved by
  - Custom implementation of *writeObject()* and *readObject()* methods
    - *implement Serializable*
  - Implementing Externalizable interface and implement *writeExternal()* and *readExternal()* methods
    - *implement Externalizable*

Custom serialization with custom *writeObject()* and *readObject()* methods

- Custom serialization can be achieved by implementing the following two methods:
  - *private void writeObject(ObjectOutputStream out) throws IOException {…}*
  - *private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException {…}*
- *writeObject()* method is used to serialize individual fields
- *readObject()* method is used to deserialize individual fields

Example: implementing custom serialization methods

```java
import java.io.*;

class Address implements Serializable {
    private int streetNum;
    private String streetName;
    private String city;
    private String state;
    private int zip;

    Address(int streetNum, String streetName, String city, String state, int zip){
        this.streetNum = streetNum;
        this.streetName = streetName;
        this.city = city;
        this.state = state;
        this.zip = zip;
    }

    public String getState(){return this.state;}
    public void setState(String state){this.state=state;}
    public String toString(){
        return this.streetNum+" "+this.streetName+", "+this.city+" "+this.state+", "+this.zip;
    }
}
```

Note: declared as private, no other class but the implementor is intended to run these methods

Custom implementation converts state to upper case regardless of the case the field is

```java
class Employee implements Serializable {
    private Integer id;
    private String name;
    private int age;
    private transient String ssn;
    private Address address;

    Employee(String name, int age, Integer id, Address address) {
        this.name = name;
        this.age = age;
        this.id = id;
        this.address = address;
    }

    public void setSSN(String ssn){this.ssn = ssn;}
    public String getSSN(){return this.ssn;}
    public String getName(){return this.name;}
    public Integer getAge(){return this.age;}
    public Integer getID(){return this.id;}
    public Address getAddress(){return this.address;}
    public String toString() {
        return this.name+" is "+this.age+"yo (id = "+this.id+")\n"+this.address;
    }

    private void writeObject(ObjectOutputStream out) throws IOException{
        out.defaultWriteObject();
        out.writeObject(this.address.getState().toUpperCase());
    }

    private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException{
        in.defaultReadObject();
        String state = (String) in.readObject();
        this.address.setState(state.toUpperCase());
    }
}
```

Example: implementing custom serialization methods (cont'd)

```java
public class Test {
    public static void main(String[]args) {
        Employee empl = new Employee("John Smith", 35, 101, new Address(10,"Main St","MyTown","ca",91111));
        empl.setSSN("000-11-4444");

        try {
            FileOutputStream streamOut = new FileOutputStream("./employee.ser");
            ObjectOutputStream objectOutput = new ObjectOutputStream(streamOut);
            objectOutput.writeObject(empl);
            objectOutput.close();
            streamOut.close();
        } catch (IOException e) {
            System.out.println(e);
        }

        Employee empl2 = null;
        try {
            FileInputStream streamIn = new FileInputStream("./employee.ser");
            ObjectInputStream objectInput = new ObjectInputStream(streamIn);
            empl2 = (Employee) objectInput.readObject();
            objectInput.close();
            streamIn.close();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println(e);
            return;
        }

      System.out.println("Deserialized Employee object:");
      System.out.println("Name: " + empl2.getName());
      System.out.println("Age: " + empl2.getAge());
      System.out.println("SSN: " + empl2.getSSN());
      System.out.println("ID: " + empl2.getID());
      System.out.println("Address: " + empl2.getAddress());
    }
}
```

Output:

```
Deserialized Employee object:
Name: John Smith
Age: 35
SSN: null
ID: 101
Address: 10 Main St, MyTown CA, 91111
```

The state is upper case in the output (after deserialization) while it was entered in lower case in empl instance

Externalizable interface

- Marker interface for custom serialization
- No built-in serialization provided, must implement serialization in the class that implements this interface
  - Explicit serialization of individual fields
- *writeExternal()* method is used to serialize individual fields
- *readExternal()* method is used to deserialize individual fields
- Class that implements Externalizable must implement a default constructor

Example: Employee implements Externalizable

```java
import java.io.*;

class Employee implements Externalizable {
    private Integer id;
    private String name;
    private int age;
    private transient String ssn;
    private String address;

    public Employee(String name, int age, Integer id, String address) {
        this.name = name;
        this.age = age;
        this.id = id;
        this.address = address;
        this.ssn = "- -";
    }

    public Employee() {
        this.name = "";
        this.age = 0;
        this.id = 0;
        this.ssn = "- -";
        this.address = "";
    }

    public void setSSN(String ssn){this.ssn = ssn;}
    public String getSSN(){return this.ssn;}
    public String getName(){return this.name;}
    public Integer getAge(){return this.age;}
    public Integer getID(){return this.id;}
    public String getAddress(){return this.address;}
    public String toString() {
        return this.name+" is "+this.age+"yo (id = "+this.id+")\n"+this.address;
    }

    public void writeExternal(ObjectOutput out) throws IOException{
        out.writeObject(this.id);
        out.writeObject(this.name);
        out.writeInt(this.age);
        out.writeObject(this.address);
    }

    public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException{
        this.id = (Integer) in.readObject();
        this.name = (String) in.readObject();
        this.age = (int) in.readInt();
        this.ssn = "- -";
        this.address = (String) in.readObject();
    }
}
```

```java
public class Test {
    public static void main(String[]args) {
        Employee empl = new Employee("John Smith", 35, 101, "10 Main St, MyTown CA 91111");
        empl.setSSN("000-11-4444");

        try {
            FileOutputStream streamOut = new FileOutputStream("./employee.ser");
            ObjectOutputStream objectOutput = new ObjectOutputStream(streamOut);
            objectOutput.writeObject(empl);
            objectOutput.close();
            streamOut.close();
        } catch (IOException e) {
            System.out.println(e);
        }

        Employee empl2 = null;
        try {
            FileInputStream streamIn = new FileInputStream("./employee.ser");
            ObjectInputStream objectInput = new ObjectInputStream(streamIn);
            empl2 = (Employee) objectInput.readObject();
            //empl2 = new Employee();
            objectInput.close();
            streamIn.close();
        } catch (IOException | ClassNotFoundException e) {  // | ClassNotFoundException
            System.out.println(e);
            return;
        }

      System.out.println("Deserialized Employee object:");
      System.out.println("Name: " + empl2.getName());
      System.out.println("Age: " + empl2.getAge());
      System.out.println("SSN: " + empl2.getSSN());
      System.out.println("ID: " + empl2.getID());
      System.out.println("Address: " + empl2.getAddress());
    }
}
```

Output:

```
Deserialized Employee object:
Name: John Smith
Age: 35
SSN: - -
ID: 101
Address: 10 Main St, MyTown CA 91111
```

Several notes about deserializing when implement Externalizable interface

- The order of the fields as they are being read in while deserializing must be the same as in the class definition
  - java.io.EOFException

- Public default constructor has to be declared in the class as well as every member class

Example: custom serialization of Employee objects and their members

```java
import java.io.*;

class Address implements Externalizable {
    private int streetNum;
    private String streetName;
    private String city;
    private String state;
    private int zip;

    public Address(int streetNum, String streetName, String city, String state, int zip){
        this.streetNum = streetNum;
        this.streetName = streetName;
        this.city = city;
        this.state = state;
        this.zip = zip;
    }

    public Address(){
        this.streetNum = 0;
        this.streetName = "";
        this.city = "";
        this.state = "XX";
        this.zip = 0;
    }

    public String getState(){return this.state;}
    public void setState(String state){this.state=state;}
    public String toString(){
        return this.streetNum+" "+this.streetName+", "+this.city+" "+this.state+", "+this.zip;
    }

    public void writeExternal(ObjectOutput out) throws IOException{
        out.writeInt(this.streetNum);
        out.writeObject(this.streetName);
        out.writeObject(this.city);
        out.writeObject(this.state);
        out.writeInt(this.zip);
    }

    public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException{
        this.streetNum = (int) in.readInt();
        this.streetName = (String) in.readObject();
        this.city = (String) in.readObject();
        this.state = (String) in.readObject();
        this.zip = (int) in.readInt();

        this.streetName = this.streetName.toUpperCase();
        this.city = this.city.toUpperCase();
        this.state = this.state.toUpperCase();
    }
}
```

```java
class Employee implements Externalizable {
    private Integer id;
    private String name;
    private int age;
    private transient String ssn;
    private Address address;

    public Employee(String name, int age, Integer id, Address address) {
        this.name = name;
        this.age = age;
        this.id = id;
        this.address = address;
        this.ssn = "- -";
    }

    public Employee() {
        this.name = "";
        this.age = 0;
        this.id = 0;
        this.ssn = "- -";
        this.address = new Address();
    }

    public void setSSN(String ssn){this.ssn = ssn;}
    public String getSSN(){return this.ssn;}
    public String getName(){return this.name;}
    public Integer getAge(){return this.age;}
    public Integer getID(){return this.id;}
    public Address getAddress(){return this.address;}
    public String toString() {
        return this.name+" is "+this.age+"yo (id = "+this.id+")\n"+this.address;
    }

    public void writeExternal(ObjectOutput out) throws IOException{
        out.writeObject(this.id);
        out.writeObject(this.name);
        out.writeInt(this.age);
        out.writeObject(this.address);
    }

    public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException{
        this.id = (Integer) in.readObject();
        this.name = (String) in.readObject();
        this.age = (int) in.readInt();
        this.ssn = "- -";
        this.address = (Address) in.readObject();
    }
}
```

Example: custom serialization of Employee objects and their members (cont'd)

```
public class Test {
    public static void main(String[]args) {
        Employee empl = new Employee("John Smith", 35, 101, new Address(10,"Main St","MyTown","ca",91111));
        empl.setSSN("000-11-4444");

        try {
            FileOutputStream streamOut = new FileOutputStream("./employee.ser");
            ObjectOutputStream objectOutput = new ObjectOutputStream(streamOut);
            objectOutput.writeObject(empl);
            objectOutput.close();
            streamOut.close();
        } catch (IOException e) {
            System.out.println(e);
        }

        Employee empl2 = null;
        try {
            FileInputStream streamIn = new FileInputStream("./employee.ser");
            ObjectInputStream objectInput = new ObjectInputStream(streamIn);
            empl2 = (Employee) objectInput.readObject();
            //empl2 = new Employee();
            objectInput.close();
            streamIn.close();
        } catch (IOException | ClassNotFoundException e) {  // | ClassNotFoundException
            System.out.println(e);
            return;
        }

      System.out.println("Deserialized Employee object:");
      System.out.println("Name: " + empl2.getName());
      System.out.println("Age: " + empl2.getAge());
      System.out.println("SSN: " + empl2.getSSN());
      System.out.println("ID: " + empl2.getID());
      System.out.println("Address: " + empl2.getAddress());
    }
}
```

Output:

```
Deserialized Employee object:
Name: John Smith
Age: 35
SSN: - -
ID: 101
Address: 10 MAIN ST, MYTOWN CA, 91111
```

The address is converted to upper case after deserialization

Concluding remarks

- Advantages of serialization
  - Ability to persist objects and their state
  - Allows to send objects across networks

- To be serializable the following two criteria have to be satisfied
  - Class implements serializable interface
  - All members of the class are either serializable or declared *transient*
  - Static class members are not serialized
- When serializing collections or arrays, ensure each element is serializable
- Serialization is inheritable by descendants of the class
- *serialVersionUID* provides a way to ensure object compatibility
- Constructor of the class is not called when the object is deserialized
- Custom serialization can be achieved with
  - Custom implementation of private *writeObject()* and *readObject()* methods while implementing Serializable
  - Implementing Externalizable and providing implementations of public *writeObject()* and *readObject()* methods