LearnQuest

Presents

Java Serialization

www.LearnQuest.com

Serializing Objects

- Java objects are inherently ephemeral
  - ✓ They are time bounded – they exist only while the Java program is running
  - ✓ They are space bounded – they exist only in the JVM where they were created (specifically on that JVM's memory heap)
- Serialization writes a Java object out to persistent storage
  - ✓ This allows the object to be reconstituted later in another Java program
  - ✓ It also allows an object to be recreated in another JVM
  - ✓ For example, the persistent file can be sent over a network

The Serializable Interface

- Classes that implement the Serializable interface can be save to disk and recovered
  - ✓ Serialization writes the object
  - ✓ Deserialization recovers the object
- The underlying mechanism of how the process is executed is handled by Java
  - ✓ We don’t have to write code to save or recover the object
  - ✓ This is all handled by Java

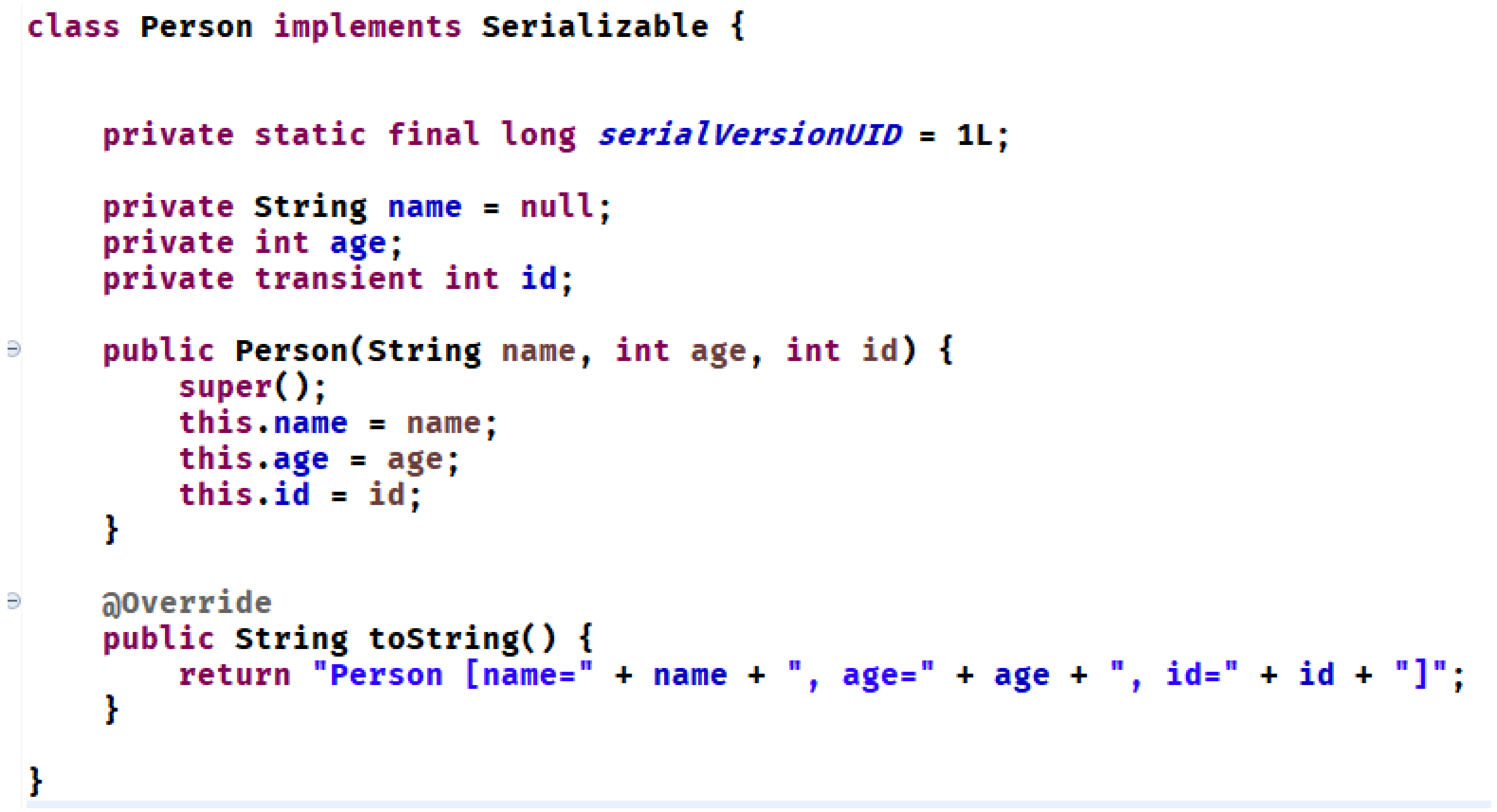
Serialization

- Serialization:
  - ✓ Saves the instance data
  - ✓ Does NOT save static data
  - ✓ Does NOT save instance data marked with the transient keyword
- In order to deserialize an object, the JVM must have access to object's class definition
  - ✓ The methods of an object are not serialized
  - ✓ We usually want to serialize the state of an object which is represented by the instance data
  - ✓ If the wrong class definition is being used during deserialization, then an exception is thrown

Serial UUID

- In order to ensure proper serialization
  - ✓ The class to be serialized has UUID which represents a version of the class
  - ✓ This is automatically generated at the time of serialization
  - ✓ This is generated from the corresponding ‘.class’ file
- There are a number of problems with this
  - ✓ Different Java versions or platforms can create problems
  - ✓ The complexity of computing the UUID can impact performance
- The alternative is to define our own version ID

A Serializable Class



```
class Person implements Serializable {


    private static final long serialVersionUID = 1L;

    private String name = null;
    private int age;
    private transient int id;

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

    @Override
    public String toString() {
        return "Person [name=" + name + ", age=" + age + ", id=" + id + "]";
    }

}
```

Serialization Output

- The serialization is done by an OutputObjectStream
  - ✓ Wraps a FileOutputStream analogous to a BufferedWriter

```
FileOutputStream outfile = new FileOutputStream("person.ser");
ObjectOutputStream out = new ObjectOutputStream(outfile);
out.writeObject(bob);
out.close();
outfile.close();
```

Deserialization Input

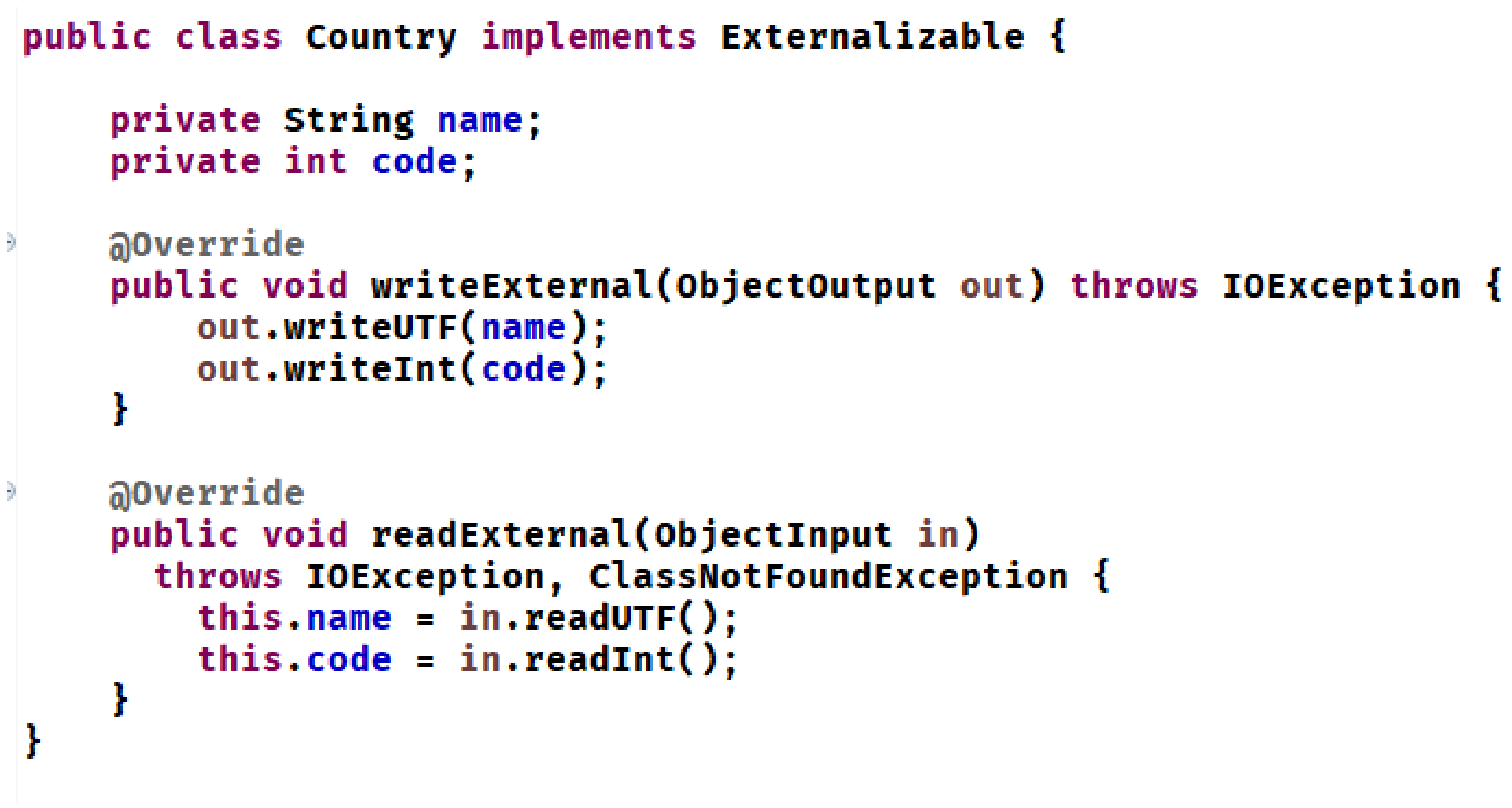
- The serialization is done by an InputObjectStream
  - ✓ Wraps a FileInputStream analogous to a BufferedReader
  - ✓ We must cast the deserialized object to the correct type

```
FileInputStream infile = new FileInputStream("person.ser");
ObjectInputStream in = new ObjectInputStream(infile);
otherBob = (Person) in.readObject();
in.close();
infile.close();
```

Externalizable

- Serialization may not be adequate for some tasks
  - ✓ Certain fields may require special handing
  - ✓ For example, encryption of credentials
- The Externalizable interface can be used to implement customized serialization
  - ✓ Serialization is defined in two methods
  - ✓ “writeExternal()” defines how to serialize
  - ✓ “readInternal()” defines how to deserialize.

An Externalizable Class



```
public class Country implements Externalizable {

    private String name;
    private int code;

    @Override
    public void writeExternal(ObjectOutput out) throws IOException {
        out.writeUTF(name);
        out.writeInt(code);
    }

    @Override
    public void readExternal(ObjectInput in)
      throws IOException, ClassNotFoundException {
        this.name = in.readUTF();
        this.code = in.readInt();
    }
}
```

Questions