Serialization

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming Concepts using Java

Week 9

Reading and writing objects

- We can read and write binary data
  - `DataInputStream`, `DataOutputStream`

Reading and writing objects

- We can read and write binary data
  - `DataInputStream`, `DataOutputStream`
- Read and write low level units
  - Bytes, integers, floats, characters, ...

Reading and writing objects

- We can read and write binary data
  - `DataInputStream`, `DataOutputStream`
- Read and write low level units
  - Bytes, integers, floats, characters, ...
- Can we export and import objects directly?

Reading and writing objects

- We can read and write binary data
  - `DataInputStream`, `DataOutputStream`
- Read and write low level units
  - Bytes, integers, floats, characters, ...
- Can we export and import objects directly?
- Why would we want to do this?
  - Backup objects onto disk, with state
  - Restore objects from disk
  - Send objects across a network

Reading and writing objects

- We can read and write binary data
  - `DataInputStream`, `DataOutputStream`
- Read and write low level units
  - Bytes, integers, floats, characters, ...
- Can we export and import objects directly?
- Why would we want to do this?
  - Backup objects onto disk, with state
  - Restore objects from disk
  - Send objects across a network
- Serialization and deserialization

Reading and writing objects . . .

- To write objects, Java has another output stream type, `ObjectOutputStream`

```
var out = new ObjectOutputStream(
    new FileOutputStream("employee.dat"));
```

Reading and writing objects . . .

- To write objects, Java has another output stream type, `ObjectOutputStream`
- Use `writeObject()` to write out an object

```
var out = new ObjectOutputStream(
    new FileOutputStream("employee.dat"));

var emp = new Employee(...);
var boss = new Manager(...);
out.writeObject(emp);
out.writeObject(boss);
```

Reading and writing objects . . .

- To write objects, Java has another output stream type, `ObjectOutputStream`
- Use `writeObject()` to write out an object
- To read back objects, use `ObjectInputStream`

```
var out = new ObjectOutputStream(
    new FileOutputStream("employee.dat"));

var emp = new Employee(...);
var boss = new Manager(...);
out.writeObject(emp);
out.writeObject(boss);

var in = new ObjectInputStream(
    new FileInputStream("employee.dat"));
```

Reading and writing objects . . .

- To write objects, Java has another output stream type, `ObjectOutputStream`
- Use `writeObject()` to write out an object
- To read back objects, use `ObjectInputStream`
- Retrieve objects in the same order they were written, using `readObject()`

```
var out = new ObjectOutputStream(
    new FileOutputStream("employee.dat"));

var emp = new Employee(...);
var boss = new Manager(...);
out.writeObject(emp);
out.writeObject(boss);

var in = new ObjectInputStream(
    new FileInputStream("employee.dat"));

var e1 = (Employee) in.readObject();
var e2 = (Employee) in.readObject();
```

Reading and writing objects . . .

- To write objects, Java has another output stream type, `ObjectOutputStream`
- Use `writeObject()` to write out an object
- To read back objects, use `ObjectInputStream`
- Retrieve objects in the same order they were written, using `readObject()`
- Class has to allow serialization — implement marker interface `Serializable`

```
var out = new ObjectOutputStream(
    new FileOutputStream("employee.dat"));

var emp = new Employee(...);
var boss = new Manager(...);
out.writeObject(emp);
out.writeObject(boss);

var in = new ObjectInputStream(
    new FileInputStream("employee.dat"));

var e1 = (Employee) in.readObject();
var e2 = (Employee) in.readObject();

public class Employee
       implements Serializable {...}
```

How serialization works

- `ObjectOutputStream` examines all the fields and saves their contents

How serialization works

- `ObjectOutputStream` examines all the fields and saves their contents
- `ObjectInputStream` "reconstructs" the object, effectively calls a constructor

How serialization works

- `ObjectOutputStream` examines all the fields and saves their contents
- `ObjectInputStream` "reconstructs" the object, effectively calls a constructor
- What happens when many objects share the same object as an instance variable?

```
class Manager extends Employee {
   private Employee secretary;
   ....
}
```

  - Two managers have the same secretary
  - How do we avoid duplicating objects when serializing?

How serialization works

- `ObjectOutputStream` examines all the fields and saves their contents
- `ObjectInputStream` "reconstructs" the object, effectively calls a constructor
- What happens when many objects share the same object as an instance variable?

```
class Manager extends Employee {
   private Employee secretary;
   ....
}
```

  - Two managers have the same secretary
  - How do we avoid duplicating objects when serializing?
- Each object is assigned a serial number
  - When first encountered, save the data to output stream
  - If saved previously, record serial number
  - Reverse the process when reading

How serialization works

- `ObjectOutputStream` examines all the fields and saves their contents
- `ObjectInputStream` "reconstructs" the object, effectively calls a constructor
- What happens when many objects share the same object as an instance variable?

```
class Manager extends Employee {
   private Employee secretary;
   ....
}
```

  - Two managers have the same secretary
  - How do we avoid duplicating objects when serializing?
- Each object is assigned a serial number — hence serialization
  - When first encountered, save the data to output stream
  - If saved previously, record serial number
  - Reverse the process when reading

Customizing serialization

- Some objects should not be serialized
  — values of file handles, ...

Customizing serialization

- Some objects should not be serialized — values of file handles, ...
- Mark such fields as `transient`

```
public class LabeledPoint
     implements Serializable{
   private String label;
   private transient Point2D.Double point;
   ...
}
```

Customizing serialization

- Some objects should not be serialized — values of file handles, ...
- Mark such fields as `transient`
- Can override `writeObject()`
  - `defaultWriteObject()` writes out the object with all non-transient fields
  - Then explicitly write relevant details of transient fields

```
private void
      writeObject(ObjectOutputStream out)
      throws IOException{
   out.defaultWriteObject();
   out.writeDouble(point.getX());
   out.writeDouble(point.getY());
}
```

Customizing serialization

- Some objects should not be serialized — values of file handles, ...
- Mark such fields as `transient`
- Can override `writeObject()`
  - `defaultWriteObject()` writes out the object with all non-transient fields
  - Then explicitly write relevant details of transient fields
- ... and `readObject()`
  - `defaultReadObject()` reconstructs object with all non-transient fields
  - Then explicitly reconstruct transient fields

```
private void
      writeObject(ObjectOutputStream out)
      throws IOException{
   out.defaultWriteObject();
   out.writeDouble(point.getX());
   out.writeDouble(point.getY());
}
```

```
private void
      readObject(ObjectInputStream in)
      throws IOException {
   in.defaultReadObject();
   double x = in.readDouble();
   double y = in.readDouble();
   point = new Point2D.Double(x, y);
}
```

Handle with care!

- Serialization is a good option to share data within an application

Handle with care!

- Serialization is a good option to share data within an application
- Over time, older serialized objects may be incompatible with newer versions
  - Some mechanisms for version control, but still some pitfalls possible

Handle with care!

- Serialization is a good option to share data within an application
- Over time, older serialized objects may be incompatible with newer versions
  - Some mechanisms for version control, but still some pitfalls possible
- Deserialization implicitly invokes a constructor
  - Running code from an external source
  - Always a security risk

Summary

- Serialization allows us to export and import objects, with state
  - Backup objects onto disk, with state
  - Restore objects from disk
  - Send objects across a network
- Use `ObjectOutputStream` and `ObjectInputStream` to write and read objects
- Serial numbers are used to ensure only a single copy of each shared object is archived
- Mark fields that should not be serialized as `transient`
  - Customize `writeObject()` and `readObject()`
- Serialization carries risks
  - Version control of objects
  - Running unknown code