Object Serialization

- What is object serialization?
- The *Serializable* interface
- Programming with object serialization
- Example (persistence)
- Security in object serialization
- The *Externalizable* interface

What is Object Serialization?

- Java data I/O classes are not object friendly
- If you want to save the state of an object you should use object serialization
- It is a mechanism that can be used to enable persistence
- It is useful for any application that wants to:
  - Save the state of objects to a file and read those objects later to reconstruct the state of program
  - Send an object over the network (you'll see this next year)

The Serializable interface

- It is a marker interface (empty interface)

  public interface Serializable {

  }
- It is part of the *java.io* package
- It is merely used to inform the JVM that you want the object to be serialized
- Therefore, your Java objects must implement the *Serializable* interface to be serialized

Programming

- Besides an object, you need an I/O stream
- To save the state:
  - Create an instance of ObjectOutputStream (it is a subclass of FilterOutputStream)
  - Use writeObject() to save the state of the object
- To read the state:
  - Create an instance of ObjectInputStream (it is a subclass of FilterInputStream)
  - Use readObject() to read the state of the object
  - You must know what type of object is expected in the stream

Serialization

- Example: here is how you would save a serialized string to a file….

```
FileOutputStream fos = new
   FileOutputStream("file.out");
ObjectOutputStream oos = new
   ObjectOutputStream(fos);
oos.writeObject("this string is being saved");
```


Deserialization

- Example: here is how you would read and reconstruct the objects you have saved:

```
FileInputStream fis = new FileInputStream("file.out");
ObjectInputStream ois = new ObjectInputStream(fis);
String s = (String) ois.readObject();
```

Note: the program that serializes objects should be kept in sync with the program that deserializes them

Persistence

- Example:
  - Employee.java
  - SaveEmp.java
  - ReadEmp.java

Security in Object Serialization

- Consider the following snippet of code:

```
public class PasswordFile implements Serializable {
   private String passwd;
   …
}
```

- If we serialize this object we'll end up writing the password to a file, because:
- Object Serialization has access to all instance variables, including private, within a serializable class.

Security in Object Serialization

- There are two ways to serialize an object without exposing any sensitive data to the world:
  - Mark any sensitive data fields as *transient*
    e.g.: private transient String passwd;
  - Implement the *Externalizable* interface

Note: fields that are marked static are not saved as well.

The Externalizable interface

- The Externalizable interface is defined as:

```
public interface Externalizable extends Serializable {
   public void writeExternal(ObjectOutput out) throws
    IOException;
   public void readExternal(ObjectInput in) throws
    IOException, ClassNotFoundException;
}
```

Note: particularly sensitive classes should not be serialized at all