Serialization

CSCI 201
Principles of Software Development

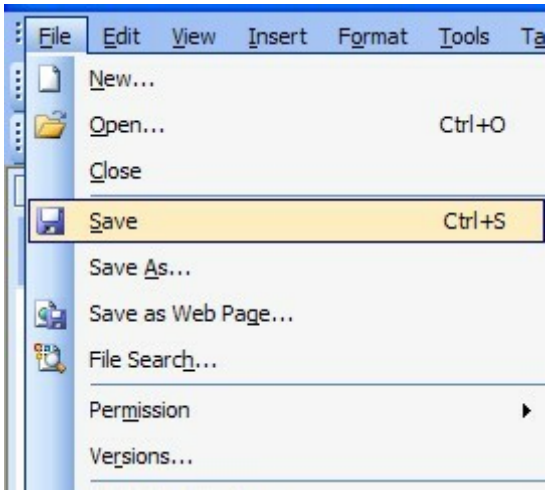
Jeffrey Miller, Ph.D.
jeffrey.miller@usc.edu

USC Viterbi
School of Engineering

Outline

- Serialization and File I/O
- Program

Saving Objects



- When writing objects out to an output stream, you may be tempted to save all of the data that is within the object individually
- Although this may work, you need to make sure to save all of the data from parent classes up the hierarchy as well
  - This is necessary if you want to save the exact state of a program or copy that state to another program
- Reading all of this data back into the exact variables may be challenging if you do not have access to set the values of all of the variables

Saving Object Example

```
1  public class Test {
2   public static void main(String [] args) {
3    Employee emp = new Employee("donald trump", 2_000_000);
4    PrintWriter pw;
5    try {
6     pw = new PrintWriter(new FileWriter("emp.txt"));
7     emp.saveData(pw);
8    } catch (IOException ioe) {
9     System.out.println(ioe.getMessage());
10   } finally {
11    if (pw != null) {
12     try {
13      pw.close();
14     } catch (IOException ioe) {
15       System.out.print(ioe.getMessage());
16     }
17    }
18   }
19  }
20 }
```

```
21  class Employee {
22   private String name;
23   private int salary;
24   Employee(String name, int sal) {
25    this.name = name;
26    this.salary = sal;
27   }
28   void saveData(PrintWriter pw) {
29    pw.println(name + "," + salary);
30    pw.flush();
31   }
32 }
```

Saving Object Example

```
1  public class Test {
2   public static void main(String [] args) {
3    Employee emp = new Employee("donald trump", 2000000);
4    PrintWriter pw;
5    try {
6     pw = new PrintWriter(new FileWriter("emp.txt"));
7     emp.saveData(pw);
8    } catch (IOException ioe) {
9     System.out.println(ioe.getMessage());
10   } finally {
11    if (pw != null) {
12     try {
13      pw.close();
14     } catch (IOException ioe) {
15       System.out.print(ioe.getMessage());
16     }
17    }
18   }
19  }
20 }
```

```
33  class Person {
34   protected String name;
35   Person(String name) {
36    this.name = name;
37   }
38  }
```

```
21  class Employee extends Person {
22   // private String name;
23   private int salary;
24   Employee(String name, int sal) {
25    this.name = name;
26    this.salary = sal;
27   }
28   void saveData(PrintWriter pw) {
29    pw.println(name + "," + salary);
30    pw.flush();
31   }
32 }
```

Saving Object Example

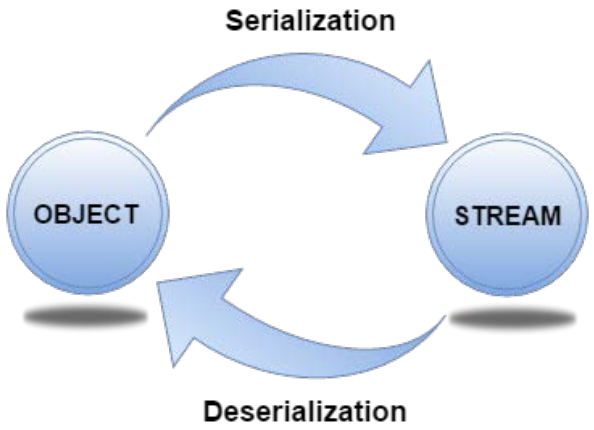
```
1  public class Test {
2   public static void main(String [] args) {
3    Employee emp = new Employee("donald trump", 2000000);
4    PrintWriter pw;
5    try {
6     pw = new PrintWriter(new FileWriter("emp.txt"));
7     emp.saveData(pw);
8    } catch (IOException ioe) {
9     System.out.println(ioe.getMessage());
10   } finally {
11    if (pw != null) {
12     try {
13      pw.close();
14     } catch (IOException ioe) {
15       System.out.print(ioe.getMessage());
16     }
17    }
18   }
19  }
20 }
```

Will line 29 compile?

```
21  class Employee extends Person {
22   // private String name;
23   private int salary;
24   Employee(String name, int sal) {
25    // this.name = name;
26    this.salary = sal;
27   }
28   void saveData(PrintWriter pw) {
29    pw.println(name + "," + salary);
30    pw.flush();
31   }
32 }
```

```
33  class Person {
34   private String name;
35   Person(String name) {
36    this.name = name;
37   }
38  }
```

Serialization Overview



- The `java.io.Serializable` interface allows us to specify to the JVM that the current object can be stored in an `ObjectOutputStream` and then retrieved from `ObjectInputStream`
- The object's state will be restored exactly when retrieved from the `ObjectInputStream`
  - This includes all of the private variables from inherited classes as well
- The `java.io.Serializable` interface does not contain any methods
- If an object does not implement the `java.io.Serializable` interface and tries to be serialized, a `NotSerializableException` will be thrown
- `static` and `transient` variables of the class are not serialized
  - If a variable in the serializable class is another object, that object must be serializable or a `NotSerializableException` will be thrown
  - You can make the variable `transient` instead, which means the variable will not be serialized

serialVersionUID

- The serialization runtime associates a version number with each serializable class, called a `serialVersionUID`, which is used during deserialization to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to serialization.
- If the receiver has loaded a class for the object that has a different `serialVersionUID` than that of the corresponding sender's class, then deserialization will result in an `InvalidClassException`.
- A serializable class can declare its own `serialVersionUID` explicitly by declaring a field named `serialVersionUID` that must be `static`, `final`, and of type `long`

```
public static final long serialVersionUID = 1;
```

From http://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html

Array Serialization Example

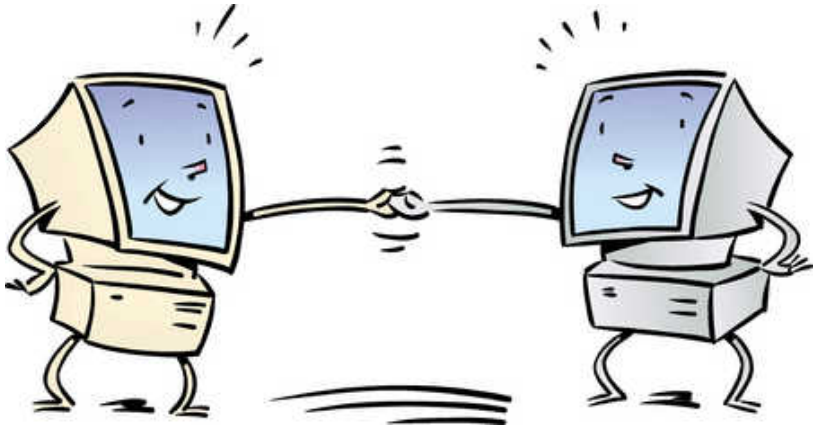
```
1   import java.io.*; // Note that I imported * to save space
2   public class Test {
3     public static void main(String [] args) {
4       try {
5         String [] strarr = {"CSCI", "201", "Spring", "Summer", "Fall"};
6         ObjectOutputStream oos = new ObjectOutputStream(
            new FileOutputStream("o.txt"));
7         oos.writeObject(strarr);
8         oos.flush();
9         oos.close();
10        ObjectInputStream ois = new ObjectInputStream(
            new FileInputStream("o.txt"));
11        String [] sarr = (String[])ois.readObject();
12        ois.close();
13      } catch (FileNotFoundException fnfe) {
14        System.out.println("FileNotFoundException: " + fnfe.getMessage());
15      } catch (IOException ioe) {
16        System.out.println("IOException: " + ioe.getMessage());
17      } catch (ClassNotFoundException cnfe) {
18        System.out.println("ClassNotFoundException: " + cnfe.getMessage());
19      }
20    }
21  }
```

Custom Serialization Example

```
1   import java.io.*;
2   class Employee implements Serializable {
3     public static final long serialVersionUID = 1;
4     private String fname, lname;
5     private transient String pass;
6     public Employee(String fname, String lname, String pass) {
7       this.fname = fname;
8       this.lname = lname;
9       this.pass = pass;
10    }
11    public void printEmployee() {
12      System.out.println(fname + " " + lname + ": " + pass);
13    }
14  }
15  public class EmployeeMain {
16    public static void main(String [] args) {
17      Employee emp = new Employee("donald", "trump", "billionaire");
18      try {
19        ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("out.txt"));
20        oos.writeObject(emp);
21        oos.flush();
22        oos.close();
23        ObjectInputStream ois = new ObjectInputStream(new FileInputStream("out.txt"));
24        Employee emp1 = (Employee)ois.readObject();
25        ois.close();
26        emp.printEmployee();
27        emp1.printEmployee();
28      } catch (Exception e) { // bad coding used to save space
29        System.out.println("e: " + e.getMessage());
34      }
35    }
36  }
```

Networking Serialization

- Just as we have written out to a file and read it back into objects, we can do the same with transmitting data over a network to another program
- We will see this when we talk about networking later in the semester



Outline

- Serialization and File I/O
- Program

Program

- Create a class that contains information about an employee, including name, address, and social security number. For the address, create another class with number, street, city, state, and zip in it.
- All of the data except social security number should be saved to a file and be read back into the same objects.