Announcements
• Final Exam: Tues. Dec. 14 @ 3:25pm

Material
• Serialization
• Cloning
• Frameworks
We have access to all the information about objects with Reflection
  - What can we do with that?
  - How about saving objects to disk?

Serialization
  - Allows a collection of objects to be written and read to/from a stream as a series of bytes.

Serialized streams are stored as binary files. In binary files no conversion from internal byte representation to text representation is performed.
• Example
  Employee[] staff = new Employee[2];
  staff.add(new Employee(...));
  staff.add(new Employee(...));

• Construct an output stream
  ObjectOutputStream out = new ObjectOutputStream(
    new FileOutputStream("staff.dat"));

• Then save the array to the stream
  out.writeObject(staff);
  out.close();

• The array and all of its objects and their dependent objects are saved
• Employee doesn't have to define any method, but needs to implement the Serializable interface
Serializable Interface

- A tagging interface with no methods
  
  ```java
  interface Serializable {
  }
  ```

- Classes that implement this interface do so only as a “tag” so that:

  ```java
  class ABC implements Serializable {
    code for ABC here
  }
  
  ABC o = new ABC();
  if (o instanceof Serializable)
    this will be true
  ```
Serializable Interface

(from the java API) Classes that require special handling during the serialization and deserialization process must implement special methods with these exact signatures:

```java
private void writeObject(java.io.ObjectOutputStream out) throws IOException
private void readObject(java.io.ObjectInputStream in) throws IOException, ClassNotFoundException;
```

Transient fields are NOT serialized, (i.e., not saved or restored).

A class that implements Serializable must mark transient fields of classes that do not support serialization, (e.g., java.awt.geom.*).
How it works

- Each newly encountered object is saved
- Each object gets a serial number in the stream
- No object is saved twice
- Reference to already encountered object saved as "reference to #"

In a Serializable class the programmer must ensure that all fields are of Serializable types or mark them as transient or over-ride & implement writeObject() and readObject().

All primitive types are serializable.
Cloning

- Shallow and Deep Copy
- Assignment (copy = e) makes shallow copy
- Clone makes deep copy
  - Employee cloned = (Employee)e.clone();

Clone Properties:
- x.clone() != x
- x.clone().equals(x)
- x.clone().getClass() == x.getClass();

No, not that kind of cloning

No, not that clone either
Employee e = new Employee(...) e =
Employee o;

//Assignment
o = e

Cloning: shallow vs deep

// shallow clone

// deep clone
Cloning

- `Object.clone` makes a new object and copies (not clones) all fields.
- Cloning is subtle; `Object.clone` is protected and will only clone objects that implement `Cloneable` interface.
- Subclass must redefine `clone` to be public, and implement `Cloneable`.

```java
public class Employee implements Cloneable {
    public Object clone() {
        return super.clone(); // not complete
    }

    ...
}
```
• **Tag interface, no methods defined**
  ```java
  public interface Cloneable {
  }
  ```

• **Used to test if “cloneable” in Object.clone()**
  ```java
  if (x instanceof Cloneable) ...make clone...
  else throws new CloneNotSupportedException()
  ```
Deep Cloning

• Why doesn’t Object make deep clone?
• Does not work in all cases...
• Your job is to clone mutable fields

```java
public class Employee implements Cloneable {
    public Object clone() {
        try {
            Employee cloned = (Employee)super.clone();
            cloned.hireDate = (Date)hiredate.clone();
            return cloned;
        } catch(CloneNotSupportedException e) {
            return null; // won't happen
        }
    }
    ...
}
```

Employee subclasses will inherit Employee clone(). If they do not over-ride it then possible shallow copying at the subclass level may occur!
Frameworks

- Set of cooperating classes
- Structures the essential mechanisms of a problem domain
- Example: Swing is a GUI framework
- Implements services common to a type of applications
- Programmer forms subclasses of framework classes
- Result is an application
- Inversion of control: framework controls execution flow
  - aka Don’t call us, we’ll call you
• Applet: Java program that runs in a web browser
• Programmer forms subclass of Applet or JApplet
• Overwrites as needed
  init/destroy
  start/stop
  paint
• Interacts with ambient browser
  String getParameter(String variable)
  getAppletContext().showDocument(URL url, String target)
  getAppletContext().showStatus(String)
• HTML page contains applet tag and parameters
  <applet code="HelloWord.class" width="300" height="100">
    <param name="message" value="GoodBye World!"/>
  </applet>
Applets as Framework

- Applet programmer uses inheritance
- Applet class deals with generic behavior (browser interaction)
- Inversion of control: applet calls init, start, stop, destroy