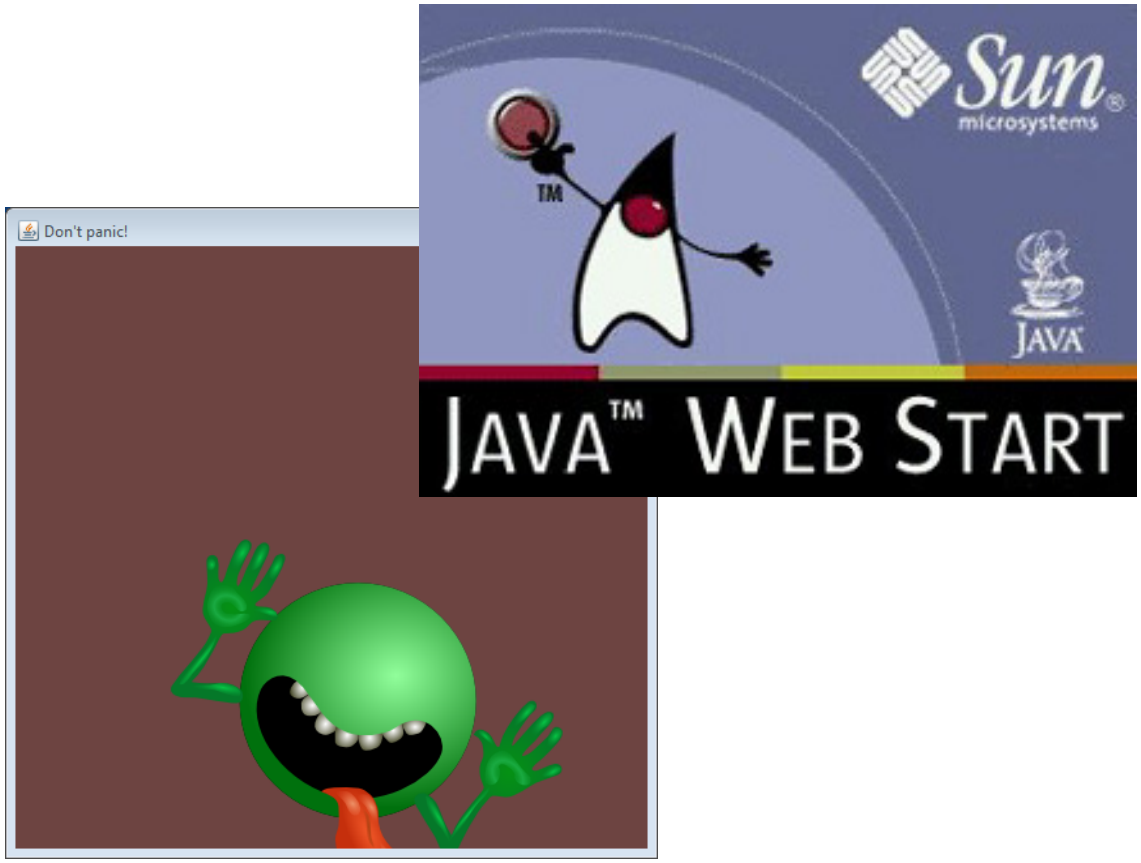
More on deployment, object serialization



Overview

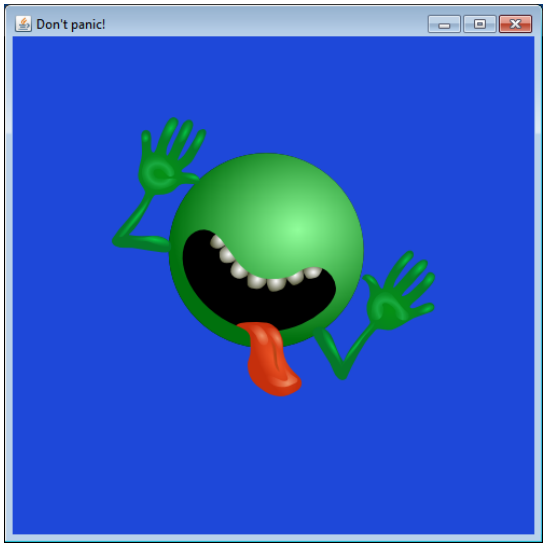
- Java ARchives (JARs)
  - Deploying by sending a single file
  - File contains everything needed to run your app
- Java Web Start (JWS)
  - Deploying JARs on your web site
  - Client automatically updates if you change JAR
- Serialization
  - Saving and reloading the state of objects
  - Built-in support in Java

JAR files

- Java ARchives
  - A compressed zip file
    - Create with the `jar` command line utility
  - Contains all the *.class files needed to run your app
    - Including any inner-classes, e.g. `Panic$PanicPanel.class`
    - Don't need the *.java files
  - Also any images, sounds, and other resources
  - Must have a special file named `mainifest.txt`
    - Defines the class to run if JAR is executed

Don't panic application

- PanicApplet was a viral sensation
- Standalone for offline enjoyment
  - Changed from JApplet to JFrame
  - Changed loading of image/sound file




```
// Load the image
URL imgURL = getClass().getClassLoader().getResource("dont_panic.png");
img = new ImageIcon(imgURL).getImage();

// Load the audio file
URL soundURL = getClass().getClassLoader().getResource("boink.wav");
audio = Applet.newAudioClip(soundURL);
```

Executable JAR: Step 1

- Organize your files
  - My example: everything in a single directory
    - Big projects will have a more complicated folder hierarchy
  - All my *.class files and any images/sounds

```
04/19/2012  07:32 PM    <DIR>          .
04/19/2012  07:32 PM    <DIR>          ..
04/19/2012  07:12 PM               295 .classpath
04/19/2012  07:12 PM               381 .project
04/19/2012  07:12 PM    <DIR>          .settings
04/14/2012  01:06 PM            43,522 boink.wav
09/10/2011  01:59 PM            38,711 dont_panic.png
04/19/2012  07:30 PM             1,008 Panic$PanicPanel.class
04/19/2012  07:30 PM             2,685 Panic.class
04/19/2012  07:30 PM             2,106 Panic.java
               7 File(s)         88,708 bytes
               3 Dir(s)     996,950,016 bytes free
```

Executable JAR: Step 2

- Create the manifest.txt file
  - Single line defining main class

```
Main-Class: Panic
```

manifest.txt

```
04/19/2012  07:41 PM    <DIR>          .
04/19/2012  07:41 PM    <DIR>          ..
04/19/2012  07:12 PM               295 .classpath
04/19/2012  07:12 PM               381 .project
04/19/2012  07:12 PM    <DIR>          .settings
04/14/2012  01:06 PM            43,522 boink.wav
09/10/2011  01:59 PM            38,711 dont_panic.png
04/19/2012  07:25 PM                19 manifest.txt
04/19/2012  07:30 PM             1,008 Panic$PanicPanel.class
04/19/2012  07:30 PM             2,685 Panic.class
04/19/2012  07:30 PM             2,106 Panic.java
               8 File(s)         88,727 bytes
               3 Dir(s)     988,528,640 bytes free
```

Executable JAR: Step 3

- Create JAR using "jar -cvmf" command

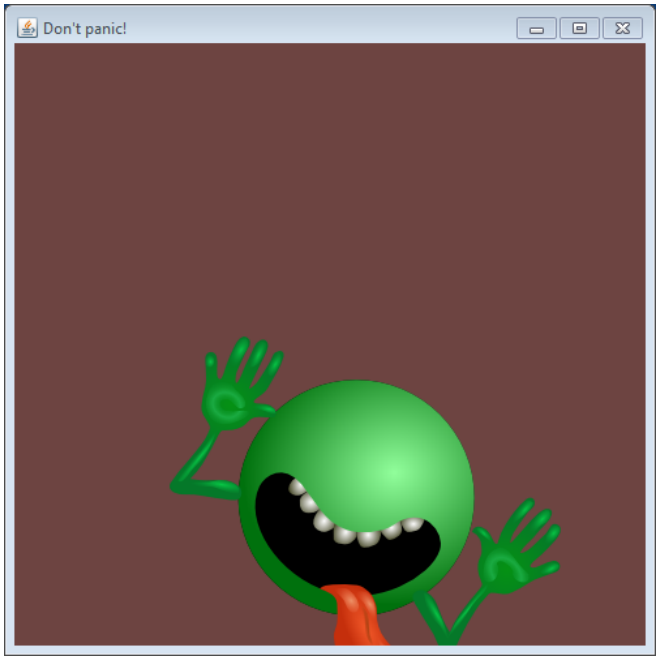
```
c:\workspace\Panic>jar -cvmf manifest.txt Panic.jar *.class *.png *.wav
added manifest
adding: Panic$PanicPanel.class(in = 1008) (out= 602)(deflated 40%)
adding: Panic.class(in = 2685) (out= 1472)(deflated 45%)
adding: dont_panic.png(in = 38711) (out= 38681)(deflated 0%)
adding: boink.wav(in = 43522) (out= 36937)(deflated 15%)
```

```
04/19/2012  07:42 PM    <DIR>          .
04/19/2012  07:42 PM    <DIR>          ..
04/19/2012  07:12 PM               295 .classpath
04/19/2012  07:12 PM               381 .project
04/19/2012  07:12 PM    <DIR>          .settings
04/14/2012  01:06 PM            43,522 boink.wav
09/10/2011  01:59 PM            38,711 dont_panic.png
04/19/2012  07:25 PM                19 manifest.txt
04/19/2012  07:30 PM             1,008 Panic$PanicPanel.class
04/19/2012  07:30 PM             2,685 Panic.class
04/19/2012  07:42 PM            78,535 Panic.jar
04/19/2012  07:30 PM             2,106 Panic.java
               9 File(s)        167,262 bytes
               3 Dir(s)     991,420,416 bytes free
```

Executable JAR: Step 4

- Executing the JAR
  - From command line using `java` command
    - Use the `-jar` switch
  - Clicking in your operating system explorer
    - Assuming your file associations are setup right

```
c:\workspace\Panic>java -jar panic.jar
```



Java Classpath

- Classpath command line switch
  - Use switch: `-classpath` or `-cp`
  - Tells tools (javac, java) where to look for user classes
  - e.g. Allow you to have one copy of `StdDraw`, `StdAudio`, …

```
 Directory of c:\java\Std

04/20/2012  09:07 PM    <DIR>          .
04/20/2012  09:07 PM    <DIR>          ..
01/08/2012  04:58 PM             9,037 StdAudio.java
01/08/2012  04:58 PM            39,797 StdDraw.java
09/29/2011  03:36 PM             5,864 StdIn.java
02/13/2012  09:11 PM             9,462 StdRandom.java
```

```
 Directory of c:\java\WavPlayer

04/20/2012  09:06 PM    <DIR>          .
04/20/2012  09:06 PM    <DIR>          ..
04/10/2012  02:51 PM            44,264 frog.wav
04/11/2012  04:20 PM             4,498 WavPanel.java
04/10/2012  07:38 PM             3,801 WavPlayer.java
```

```
c:\java\WavPlayer>javac -classpath .;c:\java\Std *.java

c:\java\WavPlayer>java -classpath .;c:\java\Std WavPlayer
```

Java Classpath

- Classpath environment variable
  - Set per command shell:

```
c:\java\WavPlayer>set CLASSPATH=.;c:\java\Std

c:\java\WavPlayer>echo %CLASSPATH%
.;c:\java\Std
```

```
vertanen@katie:~$ CLASSPATH=.:~/java/Std

vertanen@katie:~$ echo $CLASSPATH
.:/home/staff/vertanen/java/Std
```

  - Or set permanently in
    - Windows: Control Panel -> System Variables
    - Linux: startup file such as .bashrc

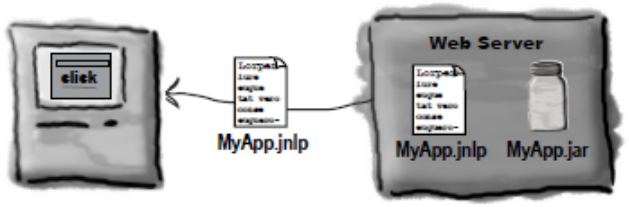
Java Web Start

- Java Web Start (JWS)
  - Advertise your fabulous app on your web site
  - Visitor clicks a link and downloads
  - Installs JAR locally on visitor's computer
    - So visitor can run even when offline
  - But application will detect if you change the app
    - Visitor's computer will update automagically

Client clicks a web page link to your JWS aplication (a .jnlp file):

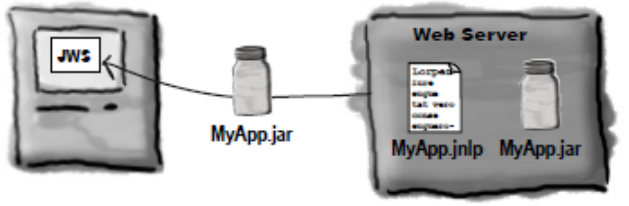
```
<a href="Panic.jnlp">Don't panic!</a>
```

Web server responds by sending the .jnlp file (an XML text file, not the actual app JAR)



Java Web Start plugin starts, reads .jnlp file, then requests the JAR file from server

Web server delivers the JAR file



Java Web Start starts the application by calling the specified main method

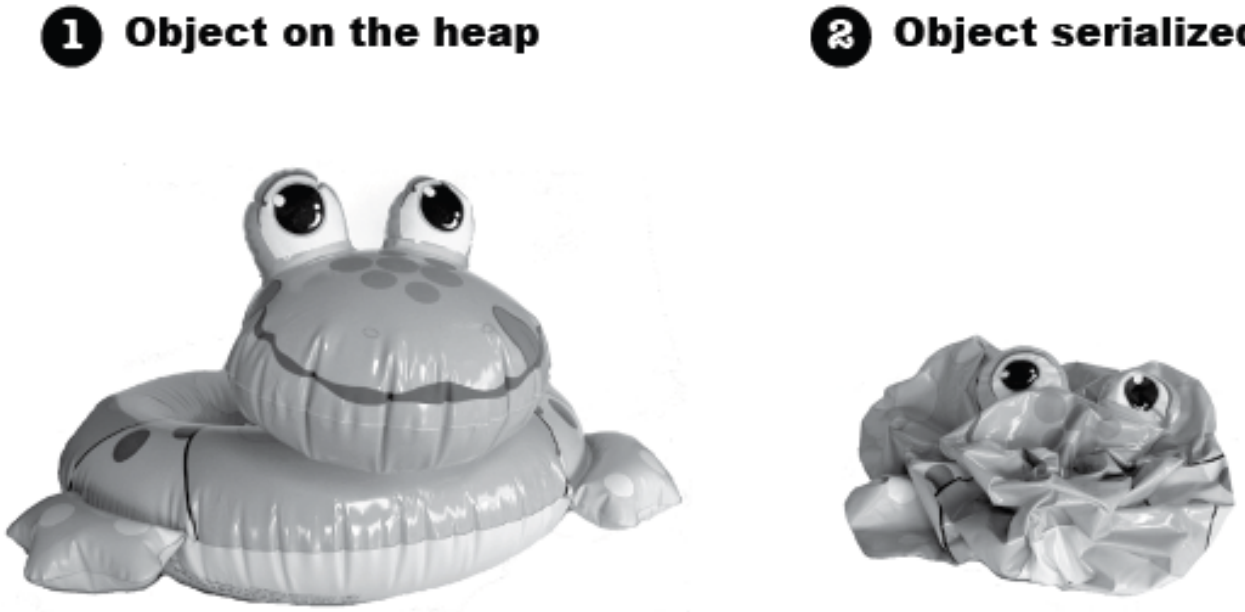
```
<?xml version="1.0" encoding="utf-8"?>

<jnlp spec="0.2 1.0"
      codebase="http://katie.mtech.edu/classes/csci136/"
      href="examples/panic.jnlp">

    <information>
        <title>Don't Panic!</title>
        <vendor>Keith Vertanen</vendor>
        <homepage href="index.php" />
        <description>Don't Panic Java Web Start Edition</description>
        <icon href="examples/dont_panic_40.png" />
        <offline-allowed />
    </information>

    <resources>
        <j2se version="1.3+" />
        <jar href="examples/Panic.jar" />
    </resources>

    <application-desc main-class="Panic" />

</jnlp>
```

Saving program state

- Common problem:
  - Save the state of your program
  - Load back at a later time
- Option 1:
  - Invent a text or binary file format
  - Can be loaded by your Java program
  - Can be loaded by other programs that knows the format
- Option 2:
  - Write serialized version of your Java objects to disk
  - Read back in an reconstitute state of object
  - Only can be read back in by your own program

Serialization

- Option 2: Saving an object instance to disk
  - Class must `implement Serializable`
    - No methods to implement, just a marker/tag interface
  - Any classes referenced by class must also `implement Serializable`
  - Write out to a binary format file



```
public class Pond implements Serializable
{
   public Duck duck = new Duck();

   public static void main(String [] args)
   {
      Pond myPond = new Pond();
      try
      {
         FileOutputStream   fs = new FileOutputStream("Pond.ser");
         ObjectOutputStream os = new ObjectOutputStream(fs);

         os.writeObject(myPond);
         os.close();
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
    }
}
```

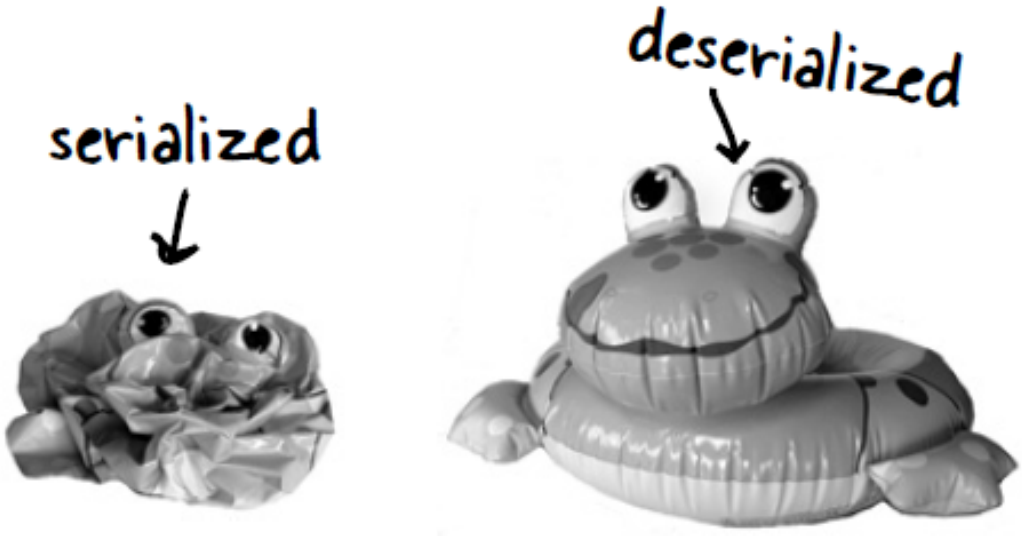
```
public class Duck implements Serializable
{
   public String name = "Daffy";
}
```

% more pond.ser

¼ϕ
♣sr
♦Pond&'‖±+┴£☻
☺L
♦duckt
♂LPond$Duck;xpsr
Pond$Duck#T↔↔♂Σ≥î☻
☻L
♦namet
↕Ljava/lang/String;L
♠this$0t
♠LPond;xpt
♣Daffyq

Serialization

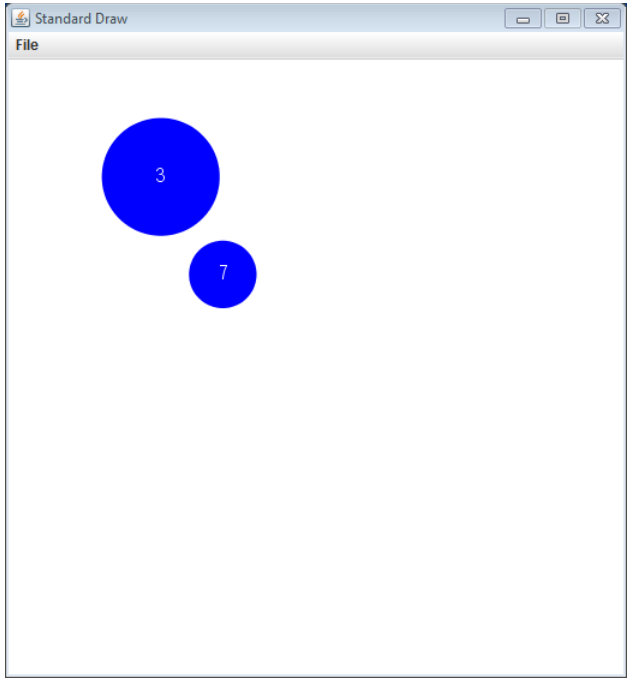
- Read an object instance from disk
  - Open the serialized file from disk
  - Read the object
    - Java reads in as a generic Object
    - Cast back to the original class type



```
public class LoadPond implements Serializable
{
   public static void main(String [] args)
   {
      Pond myPond = null;
      try
      {
         FileInputStream   fs = new FileInputStream("Pond.ser");
         ObjectInputStream is = new ObjectInputStream(fs);

         myPond = (Pond) is.readObject();
         is.close();

         System.out.println("Duck name = " + myPond.duck.name);
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
   }
}
```

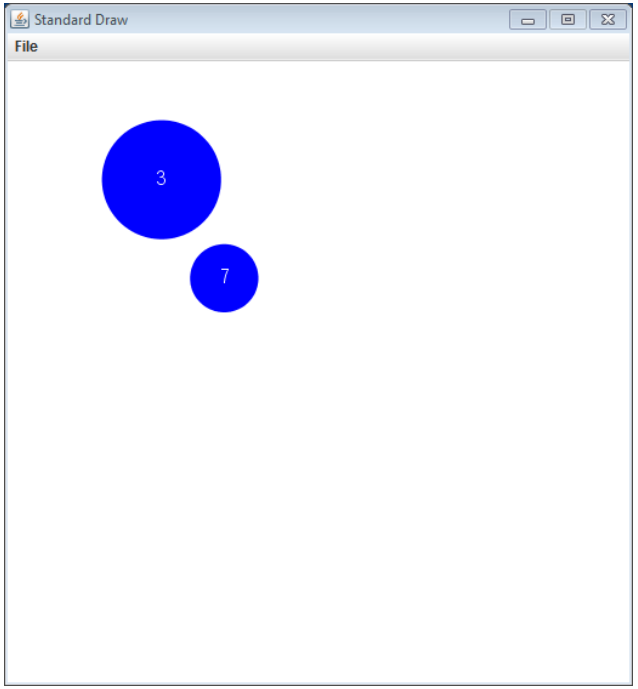
Click the circle game

- Two circles, each storing:
  - Location (x, y)
  - Radius r
  - Number of times it has been clicked
- User hits 'q' to quit
  - Save state to `ClickGame.ser`
- On startup
  - Reload state from `ClickGame.ser`
  - If no previous state file, start new game



Click the circle deluxe

- Circles jiggle around randomly
  - Managed by a thread
  - Thread object stored as instance variable
    - So we can wait for graceful shutdown on quit
  - But `Thread` does not `implement Serializable`
    - Doesn't make sense to serialize it
  - Mark as **`transient`**
    - Not stored in serialized version



Summary

- Java JAR files
  - Zip with your *.class files and other files
- Classpath
  - Allows common user classes to be stored in one place
  - Set via command line argument or system variable
- Java Web Start
  - Easy deployment of packaged JAR via web
- Saving objects
  - Serialization: easy way to save/retrieve objects from disk
  - Every instance variable must be serializable
    - Or marked as transient