Programming Assignment 6 (100 Points)
Due: 11:59pm Thursday, November 13th

README (10 points)

You are required to provide a text file named **README,** NOT Readme.txt, README.pdf, or README.doc, with your assignment in your pa6 directory. There should be no file extension after the file name "**README**". Your README should include the following sections:

Program Description (4 points) : Provide a *high level* description of both programs. Describe what your programs do and how you can interact with them. Make these explanations such that your grandmother or uncle or someone you know who has no programming experience can understand what this program does and how to use it.

Write your READMEs as if it was intended for a 5 year old. **Do not assume your reader is a computer science major.**

Short Response (6 points) : Answer the following questions:

Vim related questions:

1. In vim, how do you move the cursor to the end of a line with a single command? To the beginning of a line with a single command?
2. How do you highlight/select a line in vim?

Java related questions:

3. What does the keyword static mean in regards to variables?
4. What does the keyword static mean in regards to methods? Provide an example use of a static method.
5. What is overriding? Give an example from a previous/current assignment.

Unix related questions:

6. Using the *cut* command, how do you extract out only the columns 5 through 13 in a file named foo?

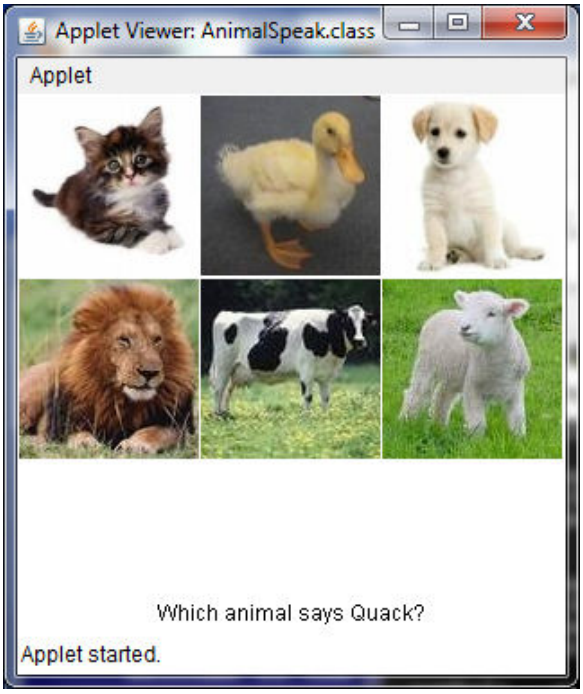
STYLE (20 points)

Please see previous programming assignments. A full file header for your README is not needed, but still need to put your name and cs11f login at the top. However, all other **style guidelines apply to all the files** (e.g. over 80 characters apply to READMEs as well.)

You will be specifically graded on commenting, file headers, class and method headers, meaningful variable names, sufficient use of blank lines, not using more than 80 characters on a line, perfect indentation, no magic numbers/hard-coded numbers other than zero, and use of accessor/mutator methods to access private fields (getters and setters) where specified.

CORRECTNESS (70 points)

For this assignment, you are asked to create two different games for pre-schooler children to learn what sounds different animals make when they speak – **AnimalSpeak** and **Memory**.



Program 1: AnimalSpeak (35 points)

Part 1: Use provided interface types

1.The Speakable interface:

```
import objectdraw.*;
public interface Speakable {
  public String speak();
  public boolean contains( Location point );
}
```

2.The Highlightable interface:

```
import java.awt.Color;
public interface Highlightable {
  public void showHighlight( Color color );
  public void hideHighlight();
  public Color getHighlightColor();
}
```

3.The AnimalCard interface, which basically extends the two interfaces above:

```
interface AnimalCard extends Speakable, Highlightable { }
```

Part 2: Animal Classes

You will create 6 animal classes: **Puppy, Kitty, Lion, Lamb, Cow, Duck**.

Each of these animal classes will implement the AnimalCard interface. For example:

```
public class Puppy implements AnimalCard { ... }
```

Six animal images are available for you in **~/../public/PA6-images/**. They are cow.jpg, duck.jpg, kitty.jpg, lamb.jpg, lion.jpg, and puppy.jpg.

Each animal class will hold the specifics about that animal:

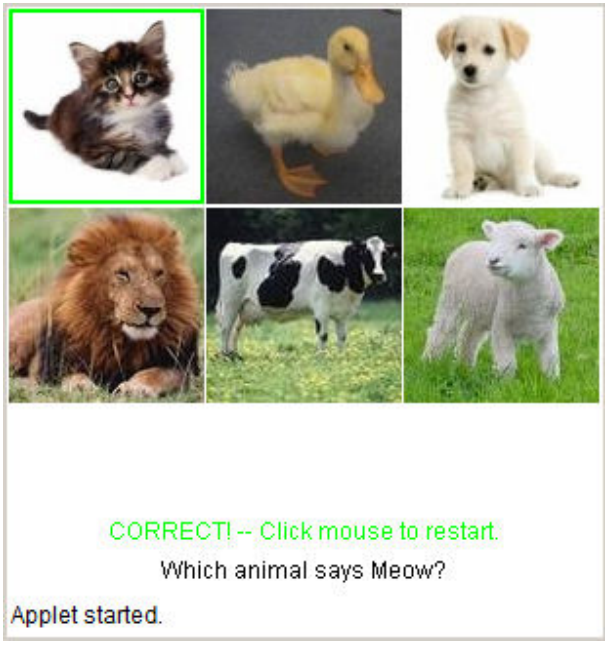
- Animal's image (a VisibleImage created from passing an Image object to its constructor)
- contains() method to determine whether some point is contained in the VisibleImage associated with that (Hint: check VisibleImage's contains() method)
- speak() method to print what sound this animal makes when it speaks Be sure to use a constant (private static final) for the String this animal speaks. Use the following generic animal sounds for each animal:

  Puppy: "Woof"
  Lion: "Roar"
  Lamb: "Baaa"

Kitty: “Meow”
Cow: “Moo”
Duck: “Quack”

- Specifics about animal’s highlighted status. You can use a FramedRect object to create the highlighting around an image. You should access this FramedRect object inside the three methods showHighlight(), hideHighlight(), and getHighlightColor().

Part 3: AnimalSpeak Applet



Create the following applet in AnimalSpeak.java:

```
public class AnimalSpeak
```

This is the controller class (extends WindowController) for this first program.

In order to integrate the jpg images into each of the Animal classes (Cow.java, Duck.java, etc), you need to use the getImage method. For example:

getImage(“cow.jpg”)

This method will return an Image object. You should pass this object as a parameter to the Animal’s constructor, and inside that Animal’s class, convert that image into a VisibleImage object.

This class will handle layout of the animal images in rows with three images per row, as shown in the image above.

It will then call a method named pickAnAnimal(), which picks a random int between 0 and (number of images - 1), uses a **switch** statement to assign a **generic AnimalCard reference** to a specific animal object, and uses that generic reference to build the following String:

"Which animal says " + animal.speak() + "?"

The above string should be displayed in a Text object centered horizontally and 20 pixels up from the bottom edge of the canvas/window.

This controller class also handles the mouse click events by checking if the user clicked on the correct image (remember the contains() method in each animal class).

Below are different scenarios that your AnimalSpeak program must handle:

AnimalSpeak Scenario 1: User clicks on CORRECT image

If the user DOES click on the correct image, display the message:

"CORRECT! -- Click mouse to restart."

The above string should be a Text object centered horizontally and 40 pixels up from the bottom edge of the canvas/window. Additionally, create a green highlight border two pixels thick (two FramedRects) around the correct animal's image. *You might want to think about which class would be most convenient to keep track of these FramedRect highlights.*

If the user clicked on the correct image, set the **generic AnimalCard reference** used in this check to **null**. You can use this to key on the next mouse click event that should blank out the correct/incorrect Text and call pickAnAnimal() again to randomly pick an animal to ask the user what sound that animals speaks.

Important:
After a correct image has been selected, the user should be able to click *anywhere* on canvas (including clicking on other images or previously selected correct image or white space) to reset the program. Resetting means that correct message and highlighting should be cleared, and a new animal should be chosen.

If this click to reset the program is on an animal image, the click should not be registered as a click to choose that animal for the next round. For example, after a correct animal is clicked and the user makes another click on a different animal image to reset, no highlighting or correct/incorrect message should appear. Only the prompt displaying "Which animal says…" should be updated with new randomly chosen animal.

Assume applet will NOT be resized.

AnimalSpeak Scenario 2: User clicks on **INCORRECT** image

If the user DOES NOT click on the correct image, display the message

"WRONG - Try Again!"

The above string should be a Text object centered horizontally and 40 pixels up from the bottom edge of the canvas/window, same as the Text object for correct message. Additionally, create a red highlight border two pixels thick (two FramedRects) around the wrong animal's image the user clicked on.

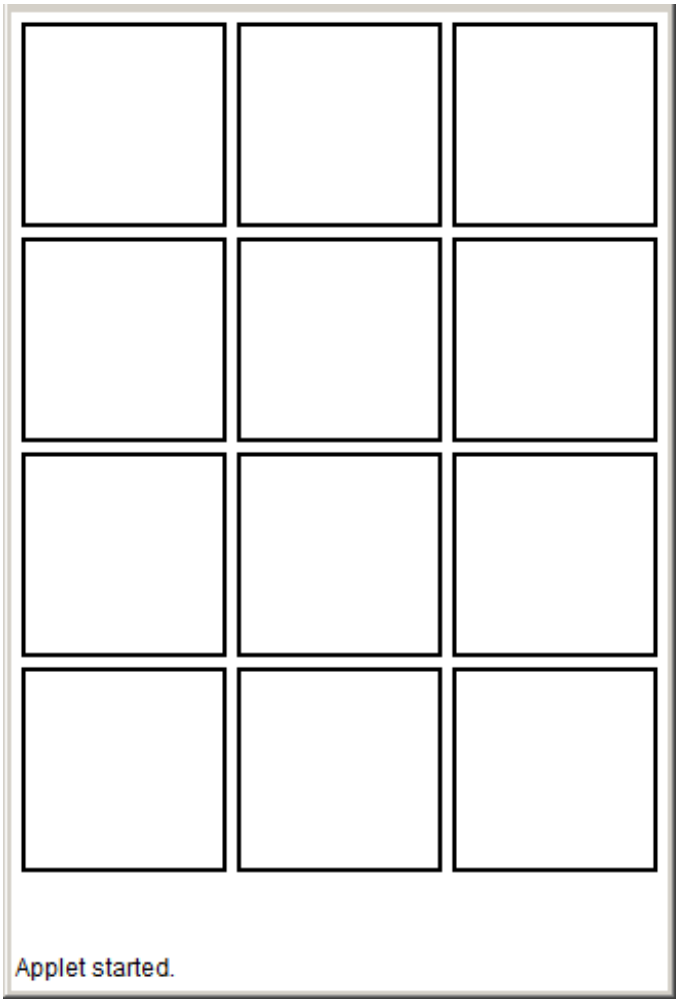
AnimalSpeak Scenario 3: User clicks OUTSIDE of any image

If the user clicks **outside** of *any* image, all highlighting should be cleared, and the CORRECT or WRONG message should be **blanked out**.

Make sure to complete Program 1: Animal Speak
BEFORE
starting Program 2: Memory.

Program 2: Memory (35 points)

The second program is a memory concentration game.



Part 1: Layout

The applet will be a playing board containing 12 tiles arranged in a 3x4 grid. These tiles are place holders for images of animals. Initially, each image will be hidden, and only a black highlight border (one pixel thick) for each image will be displayed. This information is summarized by the example picture of the applet to the right:

Part 2: Hideable Interface

In order to allow an animal image to be hidden, you will need the following Hideable interface:

```
public interface Hideable
{
   public abstract void show();
   public abstract void hide();
}
```

Part 3: Modify AnimalCard Interface

AnimalCard interface you created in the first program will need to be modified in two ways:

1. Add the new Hideable interface to the list of interfaces that are extended by AnimalCard.
2. Add an equals() method to the AnimalCard interface. Both of these changes are shown below:

```
interface AnimalCard extends Speakable,
Highlightable, Hideable
{
   @Override
   public abstract boolean equals(Object
o);
}
```

Be sure to add the "**@Override**" before the equals() method definition in each of the classes that implements

the AnimalCard interface. This way the compiler will warn you if you are not actually overriding the superclass's method.

Part 4: Modify animal classes

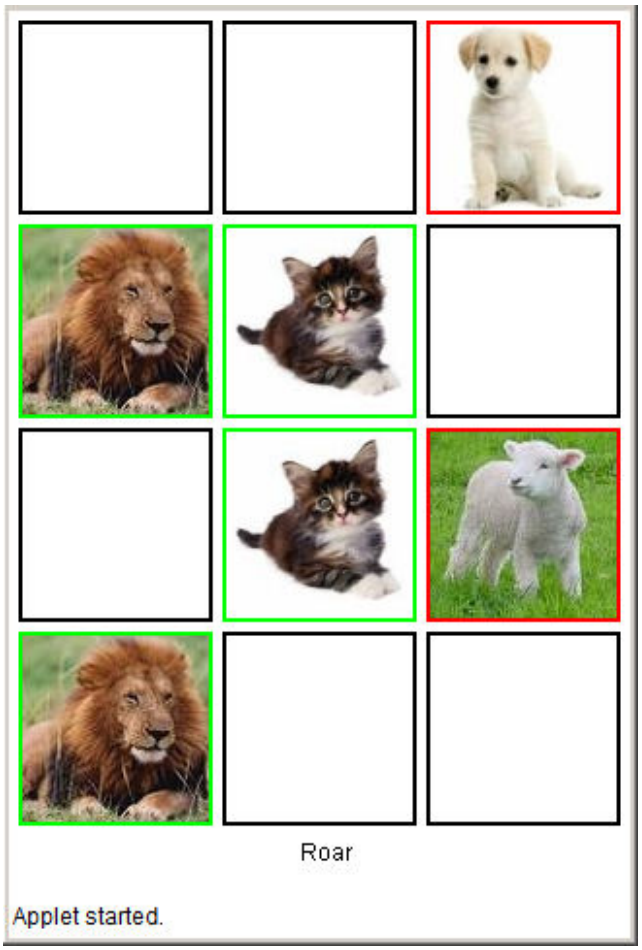
Next, you will need to implement the show(), hide(), and equals() method for each animal that implements the AnimalCard interface.

The show/hide methods should simply show or hide the VisibleImage object associated with animal. The equals() method should check whether its argument is the same animal/type as the animal on which the equals() method is called.

Part 5: Memory.java

Create the following applet:

```
public class Memory
```



This is the controller class (extends WindowController) for this second program, and should create the layout as described and illustrated above in Part 1.

This controller class will re-use much of the same code that you have written for your 6 animal classes. The only additions are the show(), hide(), and equals() methods just mentioned, along with the logic of this game itself.

For the 12 tiles on the Memory board, there will be **6 unique** images, and each image will be placed in **two *randomly selected*** tiles on the board. Then the user should be allowed to uncover two images at a time. The state of the program should change in the fashion described below:

1. When the applet is loaded, all 12 images should be initially hidden.

2. When the user clicks on any hidden tile, simply uncover the image for that tile (make the image visible using the show() method you defined in the animal classes)

3. If the user clicks on the same tile they just uncovered, **nothing should happen**. If the user clicks on a different hidden tile, also uncover that image, and then take one of the following two actions:

    a. If the two images uncovered DO NOT match, then both images should be highlighted with a red border two pixels thick (two FramedRects). Now, when the user clicks **anywhere** on the board, both of these images **should be re-hidden** (using the hide() method of the relevant animal class),

and both images' borders should be turned back to **black** (one pixel thick). To check whether two images match, use the equals() method of one image, passing in the second image as an argument. (HINT: use getClass().)

b. If the two images uncovered DO match, then both images should be highlighted with a green border two pixels thick (two FramedRects). **Additionally**, use the speak() method in the animal classes to display the animal sound in a Text object 20 pixels up from the bottom edge of the canvas. Now these two images should **remain visible** with the same green highlight border for the **remainder of the game**.

4. If there are still any uncovered (unmatched) tiles remaining, the program starts from state (2) again. Otherwise, the program ends in the final state, state (5). NOTE: If the user clicks on a tile after having uncovered two non-matching images, then the **same click** should first cause the non-matching images to be hidden, **AND** the same click should then cause the clicked tile to become visible (if it was a hidden tile).

5. At this point, all tiles have been matched and are currently highlighted green. The game is over, and there is nothing else left to do.

Important:
Unlike in AnimalSpeak, clicking on whitespace for Memory program does *nothing.* The only way to reset Memory program is by restarting the applet.

When two matching images are picked, the sound related to the animal should remain until next pair of correctly selected animal. For example, if two kittens were matched *correctly*, the Text "Meow" should stay until next pair of correctly selected animal for which, the Text should update accordingly.

There *never* should be a case where there is a single image of an animal with highlighted border.

Clicking on already uncovered correct image should not do anything, and should not be checked for correct or incorrect pairing with other images.

Clicking on the same image multiple times after it has been uncovered (but unmatched) should not count as comparisons with itself. For example, if a click has been made on a single kitten image and the user repeatedly clicks on the same image, the kitten image should not compare to itself and have green highlighted border.

Assume applet will NOT be resized.

Running

To run (and view) your AnimalSpeak applet, first create an html file named **AnimalSpeak.html**, which contains the following code:

```
<html>
   <body>
          <applet code="AnimalSpeak.class" archive="objectdraw.jar" width="305"
          height="300"></applet>
   </body>
</html>
```

Then use the appletviewer command specifying this html file:

```
> appletviewer AnimalSpeak.html
```

You must have all the files in the pa6 directory and run appletviewer in the pa6 directory for it to display correctly.

To run (and view) your Memory applet, you will have to resize the applet's width and height values in **Memory.html**. Try to make the applet the same general proportion as the pictures above.

Turnin

To turnin your code, navigate to your home directory and run the following command:

```
> turnin pa6
```

You may turn in your programming assignment as many times as you like. The last submission you turn in before the deadline is the one that we will collect.

Verify

To verify a previously turned in assignment,

```
> verify pa6
```

If you are unsure your program has been turned in, use the verify command. We will not take any late files you forgot to turn in. Verify will help you check which files you have successfully submitted. **It is your responsibility to make sure you properly turned in your assignment.**

Files to be collected

Source Files		Images	Misc.
AnimalCard.java	Cow.java	cow.jpg	README
AnimalSpeak.html	Duck.java	duck.jpg	objectdraw.jar
AnimalSpeak.java	Kitty.java	kitty.jpg	
Speakable.java	Lamb.java	lamb.jpg	
Hideable.java	Lion.java	lion.jpg	
Highlightable.java	Puppy.java	puppy.jpg	
Memory.html			
Memory.java			

Extra Credit (5 points)

1. Early Turn-In [2pts total]

    Turn in by **Tuesday, November 11th** to receive full 2 pts
    Turn in by **Wednesday, November 12th** to receive 1 pt

2. Adding Extra Animals [3pts total]

    Add at **least 3 more animals** and their images to both programs. Find an appropriate image, crop to a square, resize to 100x100 pixels.

    For **AnimalSpeak (1 pt)**, this will require an extra row of 3 images.

    For **Memory (2 pts)**, this will require two extra rows of 3 images each.

    Change the height attribute in your .html files appropriately so these new rows display properly.

NO LATE ASSIGNMENTS ACCEPTED!

START EARLY!

…and # HAVE FUN!