PA5

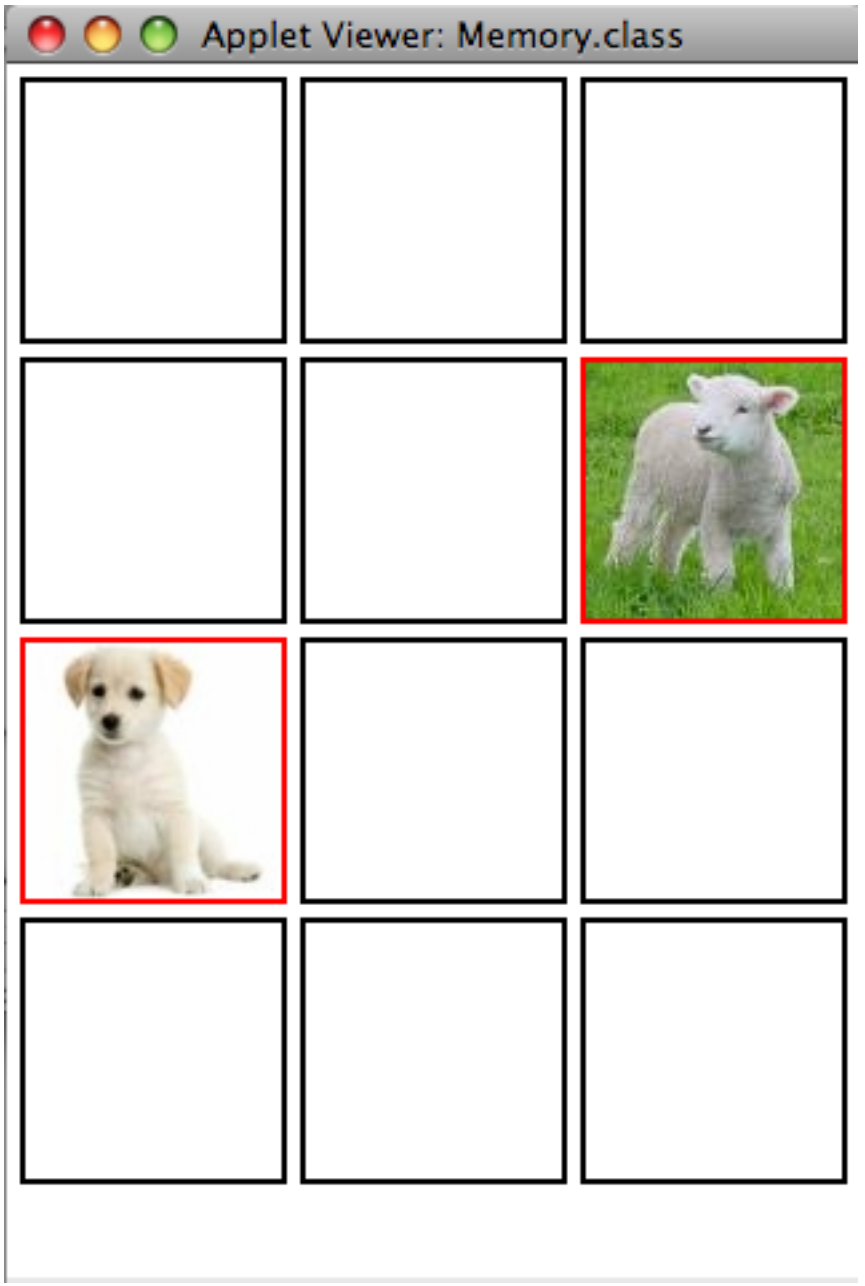
Animal Speak and Memory

Demo – Animal Speak

Show 6 different animals and prompt user to choose which animal makes the given sound.

Demo – Memory

Using the same animal cards created for the previous game, also create a memory game.

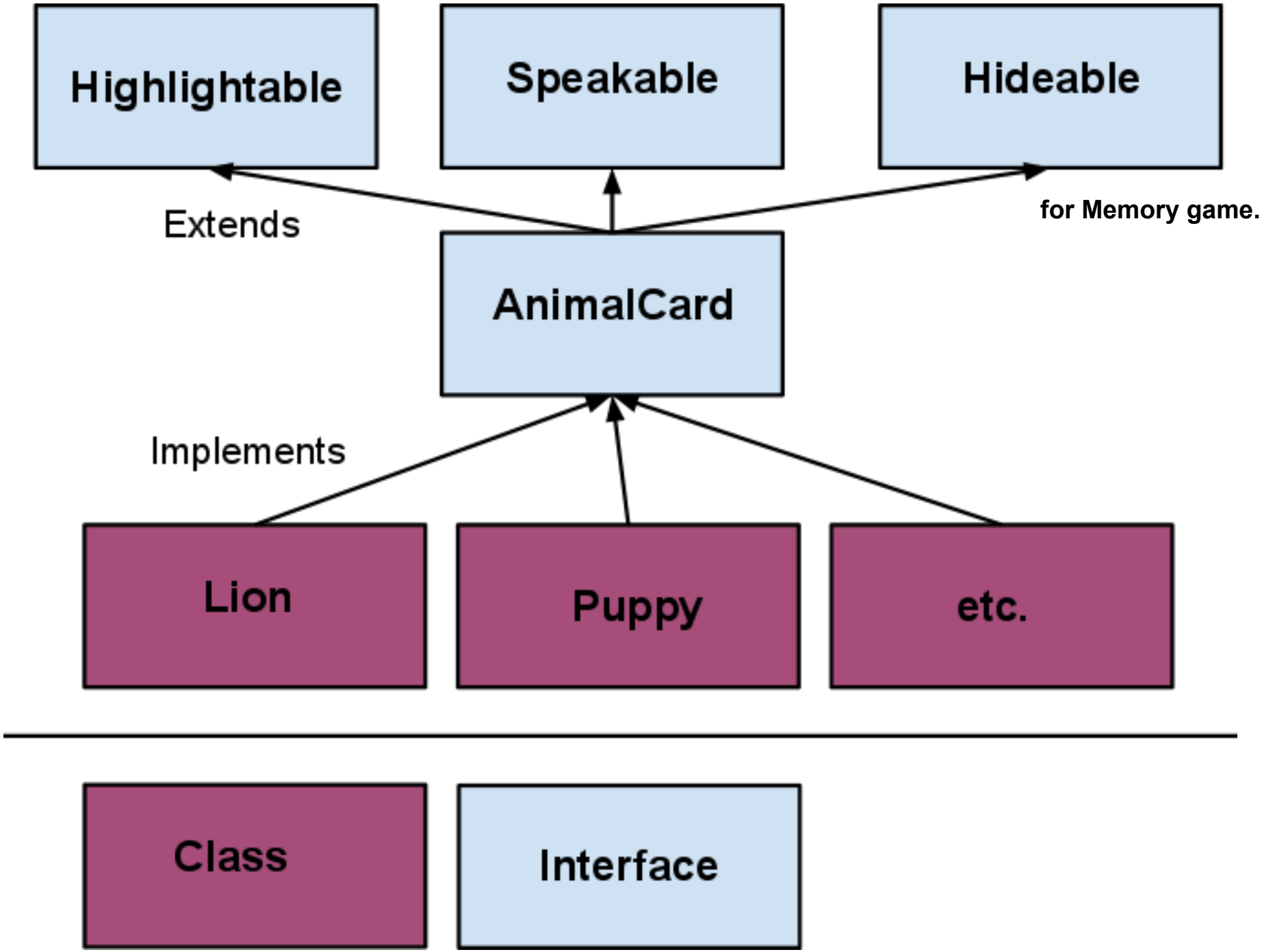


The .java files you need

- `AnimalCard.java`
- `Speakable.java (interface)`
- `Highlightable.java (interface)`
- `Hideable.java (interface)`
- `AnimalSpeak.java (controller of first game)`
- `Memory.java (controller of second game)`
- `Cow.java, Puppy.java, etc.`

Other files

- `AnimalSpeak.html`
- `Memory.html`
- `cow.jpg, puppy.jpg, etc.`


Highlightable
Speakable
Hideable
Extends
for Memory game.
AnimalCard
Implements
Lion
Puppy
etc.
Class
Interface

Interface Code (all given in write up)

Speakable

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

Highlightable

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

Highlightable, Speakable, AnimalCard

- Code for these given in assignment
  - Just need to copy, paste, and put them in their own .java files

- All three of these .java files define interfaces: not classes
  - What is an interface? How is that different? What can be included in an interface?

- All cards: Puppy.java, Lion.java, etc, will **implement** the AnimalCard interface, which in turn **extends** the other interfaces

- SO, **every** method listed in any of these interfaces MUST be implemented in Puppy.java, etc.

Benefits of implementing AnimalCard ?

Why is this helpful?
Because, then, when you declare these objects, you can declare all of them as AnimalCards, and you know that they will have at least some functionality.

Example:

```
AnimalCard card1 = new Puppy();
card1.show();
card1 = new Lion();
card1.speak();
```

These common method calls work no matter what, since card1's type is an AnimalCard.

Implementation for one animal card

```
public class Puppy implements AnimalCard {

    private static String PUPPY_SPEAK = "Woof";
    //Any other instance variables

    public Puppy(Image pic, Location loc, DrawingCanvas
canvas){
        //Save parameters in instance variables
        //Set up image, filled rect for border, etc.
    }

    // Implementations of methods from interfaces including:
    // speak(), contains(), hide(), show(), showhighlight()
    // and the others.
    // You can add more methods if you think is will be
helpful

}
```

Cards

- The images for the assignment are in the public directory, so grab them from there

- Make one class (say, Puppy.java), test it thoroughly to make sure it's perfect, and then you can copy and paste as necessary to make the other ones
  - *Just make sure you copy and paste carefully and make the **correct changes***

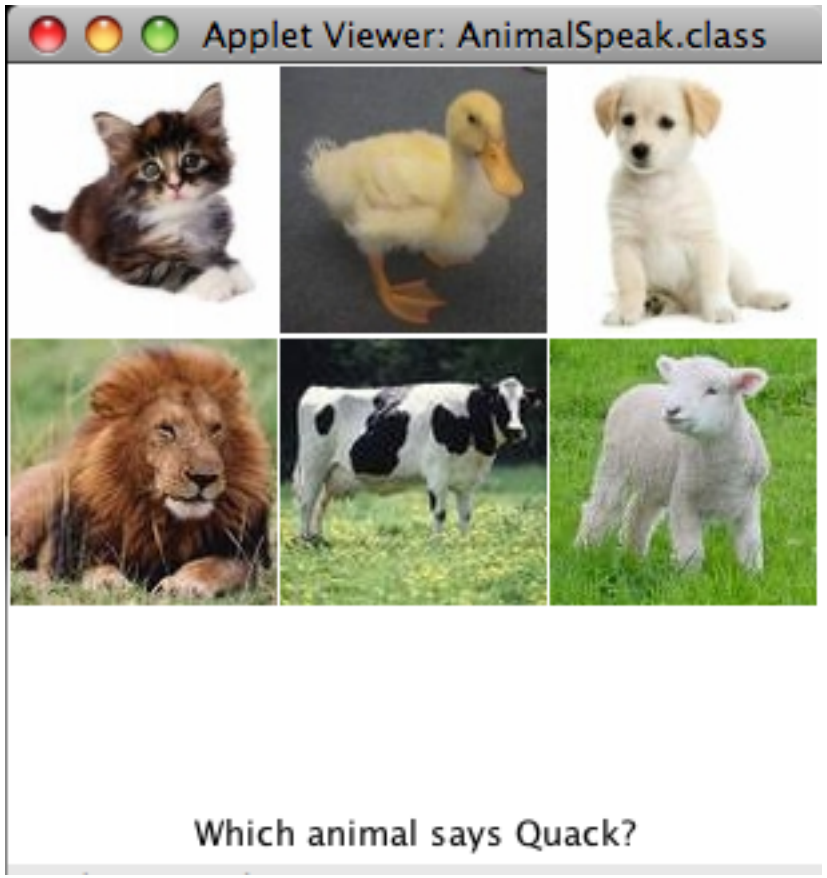
AnimalSpeak

- Now you know how to make your 6 AnimalCards

- AnimalSpeak is the first game you make using these cards

- Define a class, `AnimalSpeak.java`, that extends WindowController
  - Will set up the canvas
  - `onMouse...()` methods
  - `begin()` to set up UI elements

AnimalSpeak UI



- 6 cards displayed on canvas by calling their constructors (new Puppy(), etc.)
- Prompt text at the bottom
- Colored text above
  - WRONG - Try Again!
  - CORRECT! - click mouse to restart

AnimalSpeak.java

- Need to keep track of:
  - If user is currently correct (what might be a good way to do this?)
  - If a card's highlight should be displayed, and if so, what color

- Create a method called **pickAnAnimal()** that picks a new animal for the prompt each time the user wins or the game begins:
  - Use a **switch statement** inside this method to set the correct animal (as seen on the next slide)
  - Will also need to use a **RandomIntGenerator** (check objectdraw docs for details)

AnimalSpeak.java

```
// instantiate in begin method:
randInt = new RandomIntGenerator(0, 5);
...

private void pickAnAnimal(){
  int x = randInt.nextValue();
  switch(x){
    case 0: animal = kitty;
      break;
    case 1: animal = puppy;
      break; ...}
  prompt.setText("Which animal says " +
                 animal.speak() + "?");
  // move prompt text to be centered using
  // canvas.getWidth(), prompt.getWidth(), and math
}
```

AnimalSpeak – onMouseClick()

- Need to keep track of which image (if any), the user clicked on last
- Use that information to decide what to do on the next click
- Checking if the correct animal was selected: use the **contains**() method on your generic AnimalCard reference
- First check if the click was on the correct animal image. Then check whether or not the click was inside any image at all

AnimalSpeak – Resetting the Board

- After a WRONG image has been selected: if the user's click is not on any image, the red highlighting and the "WRONG" prompt should go away

- After a CORRECT image has been selected: if the user clicks anywhere on the board, the green highlighting and the "CORRECT" prompt should go away. ALSO, the next random animal should be selected by calling pickAnAnimal()

Memory – Hideable Interface

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

Aside from this interface, also need to add **`equals`**`()` method to AnimalCard interface described on the PA5 assignment page, and used for checking if an animal is the same as another animal

Memory – AnimalCard equals()

- Use Java objects' **`getClass`**() method to test if any two objects are instances of the same animal class

Placing images on board

• Every time the applet loads, the set of 12 images need to be **randomly** placed in the 12 tiles of the board. (Think about how to solve this problem).

• The **`RandomIntGenerator`** will be helpful here.

• Each of the six images should be placed **exactly twice –** so during placement, need to keep track how many times each image has already been placed.

• Suggestion: put this functionality in its own method (for example, `placeNextAnimal`()), calling that function 12 times from begin().

• In `placeNextAnimal`(), you will most likely need a while() loop to allow skipping over images that have been placed twice already.

Memory – onMouseClick()

If a user clicks on a card, the following conditions need to be checked:

- Has the card already been matched (has green highlight border)?
- Is the card currently uncovered already?

If any of those two cases are true, nothing should happen.

This implies that you will need to somehow keep track of the “first” card that was uncovered.

To remove red highlighting for non-matching pairs after the next click, you will also need to keep track of the “second” card as well.

Memory – onMouseClick()

- Be very, VERY cautious about when you use ==, !=, or equals() to compare two card objects!!

- `equals`() will compare **classes** of objects == and != will compare **actual instances** of objects

Memory – onMouseClick()

- Remember: when a matching pair is uncovered, the uncovered animal's sound should be displayed in a Text object on the bottom

- Call `speak`() on either the first or second card, since either will give the same sound

Other things to remember...

- Fill out ALL the README questions
  - **high-level description**

- **STYLE** is worth 30% of the assignment grade, so make sure you use good style

- Make sure you test all possible cases! When you find errors, fix them. And if your code throws exceptions, points will be docked
  - `NullPointerException`

- Refer to your book and the javadocs for objectdraw questions **first**