Lecture 6
CSE 11 Fall 2013

Classes

- Classes are *abstractions*
- They help us group actions (methods) and state (data) in sensible ways
- Well defined classes make it easier to understand and reason about a program

What is “State”?

- This information (or date) about an object that is specific to a particular instance
  - It can be used to differentiate one instance from another
- Example: Your Drivers license
  - State of Issue (String, e.g. “California”)
  - First Name, Last Name (String)
  - Date of Birth (Integer)
  - Address (String)
  - Restrictions (like requiring corrective lenses) (boolean)
  - Height (integer or double)
  - License Number (String)

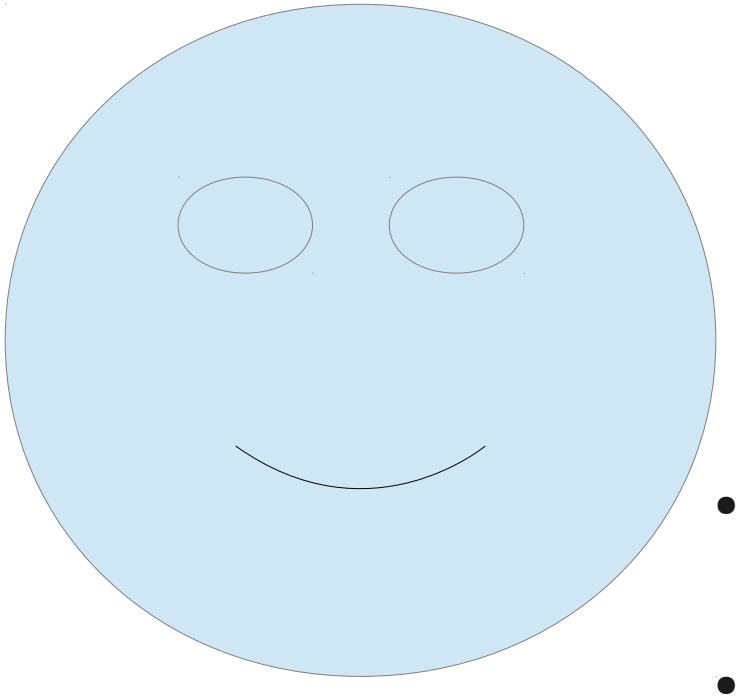
These are called state variables

- One can define state that is specific to an entire class (every instance in the class shares the same data)

Methods

- Procedures, Functions, Actions
  - Information that can retrieved from an instance
    - Accessor Methods (retrieve the value of internal variable)
      - You can read the magnetic stripe on a license to get the information in a digital form
  - Actions that can change some or all of the state variables (also called internal state)
    - Mutator methods
      - None defined on license itself.

FaceDrag Example from Book

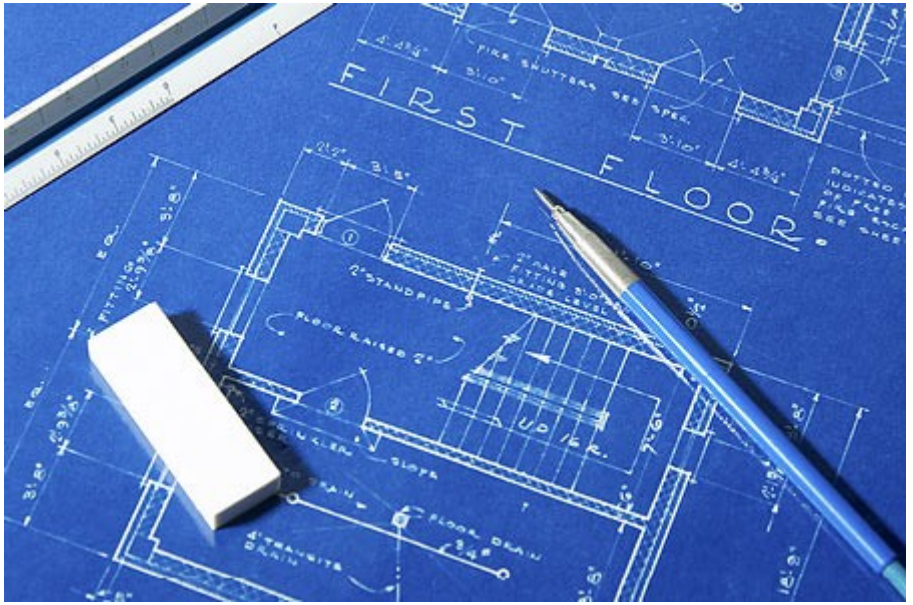


- Two choices for how to move/show/etc a face in your code
  - As four individual pieces (face, eyes, mouth)
  - Logically, as a single face (and leave the details to something else)
- Suppose you had 10 faces to keep track of on your canvas
- What if the face were more complicated (eyelashes, ears, hat,..)

Code Walk Through

- FaceDrag – no face “class”
  - http://eventfuljava.cs.williams.edu/sampleProgs/ch6/textbook/FaceDrag/FaceDrag.java
- FunnyFace – a face class
  - http://eventfuljava.cs.williams.edu/sampleProgs/ch6/textbook/RevFaceDrag/FunnyFace.java

Understanding Instance Variables

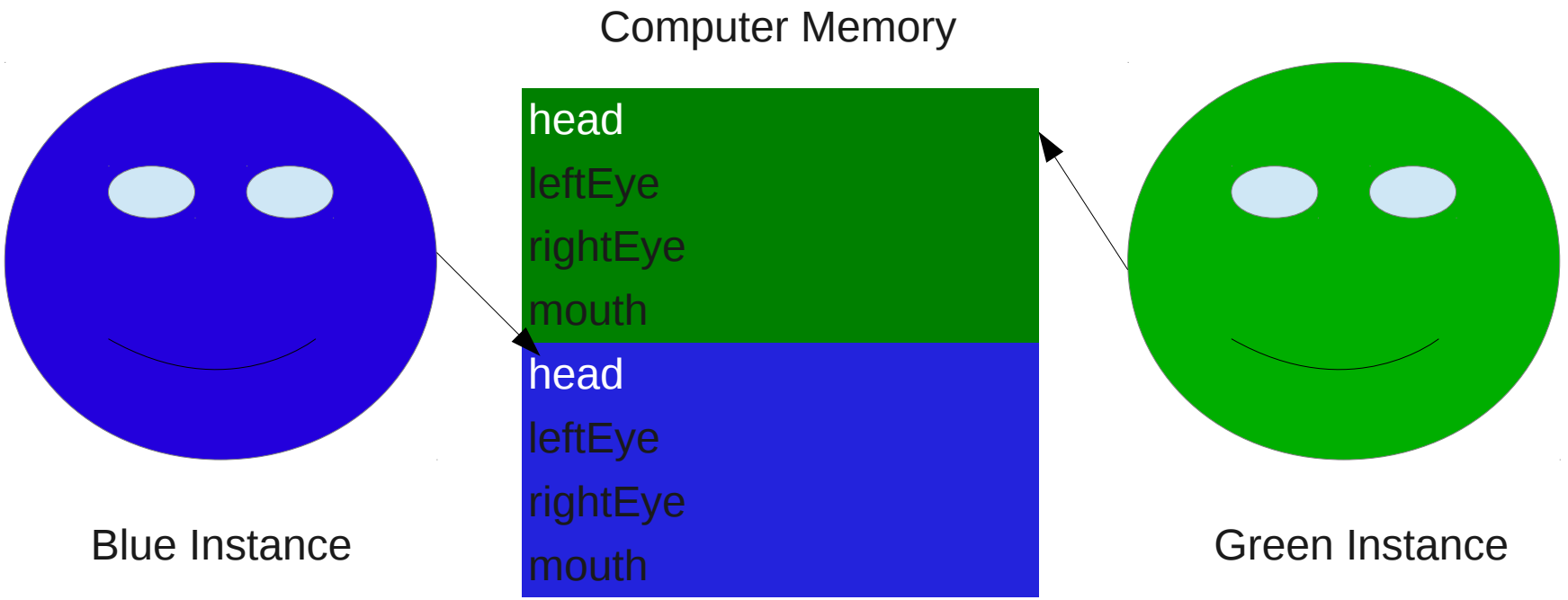


- A class definition is a blueprint for HOW to create objects

```
public class FunnyFace {
        private FramedOval head,eye;
        …
```

`private FramedOval head;`

- Each *instance* of FunnyFace will have it's own head object.
  - There is underlying storage (state) associated with each instance



Why do you use instance variables?

- Each individual object needs some information specific to it.
  - location, size, number of elements, etc.
- That information needs to *persist* from the time you construct the instance
  - Can be used to communicate state among methods
- Use temporary variables for things that do not have to persist beyond a particular method

Methods and Parameters

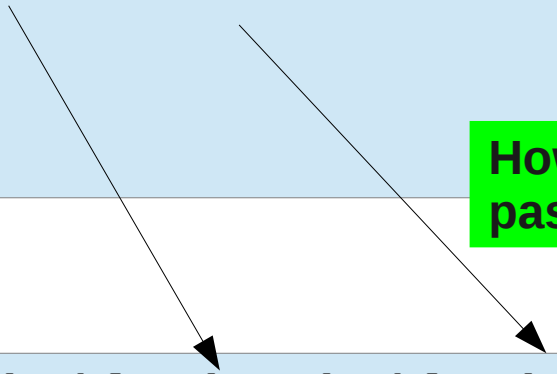
```
public void move (double dx, double dy) {
     head.move ( dx, dy );
     leftEye.move (dx, dy );
     rightEye.move (dx, dy );
     mouth.move (dx, dy);
}
```

- dx and dy are the names you (as the programmer) choice to call parameters passed to you
- `double dx;` double is the type of the parameter

Methods and Parameters

```
FunnyFace myface;

double deltaX = 45.0;
double deltaY = 75.0
     myface.move(deltaX, deltaY);
```



How are deltaX and deltaY passed to the move method?

```
public void move (double dx, double dy) {
     head.move ( dx, dy );
     leftEye.move (dx, dy );
     rightEye.move (dx, dy );
     mouth.move (dx, dy);
}
```

Temporary variables in methods

```
public void moveHalfWay (double dx, double dy) {
double halfX, halfY;

     halfX = dx/2;
     halfY = dy/2;
     head.move ( halfX, halfY);
     leftEye.move (halfX, halfY );
     rightEye.move (halfX, halfY );
     mouth.move (halfX, halfY);
}
```

- If you do not need the variable when a method ends, MAKE IT TEMPORARY!
- In this example, halfX and halfY are defined by the parameters passed to the moveHalfWay method
  - They are not needed once the method completes

Accessor Methods

- They provide information about a particular object.
- Use accessor methods to
  - Retrieve the value of an instance variable
  - Provide a logical operation about the state
    - e.g. Contains() in various examples, hidden(), etc.
  - Does NOT change the state (value stored) in any instance variable.

Constructors

- Instances do not exist unless they have been *constructed*
  - Constructors tell the Java runtime system to allocate memory specific to a new instance
  - Allows the programmer to initialize instance variables based upon parameters passed to the constructor
  - A constructor can only be called with a `new` statement.
- Java does not have a destructor (C++ does)

Constructors

```
public class Sun {
    private Location initialLocation;
    private FilledOval sunShape;

    public Sun(Location initial, double diameter,
               DrawingCanvas canvas)  {
        initialLocation = new Location(initial); // record location
        sunShape= new FilledOval(initial, diameter, diameter, canvas);
    }
...
}
```

- Are methods that have the same name as the class
- Multiple constructor definitions are allowed as long as their argument types differ from each other