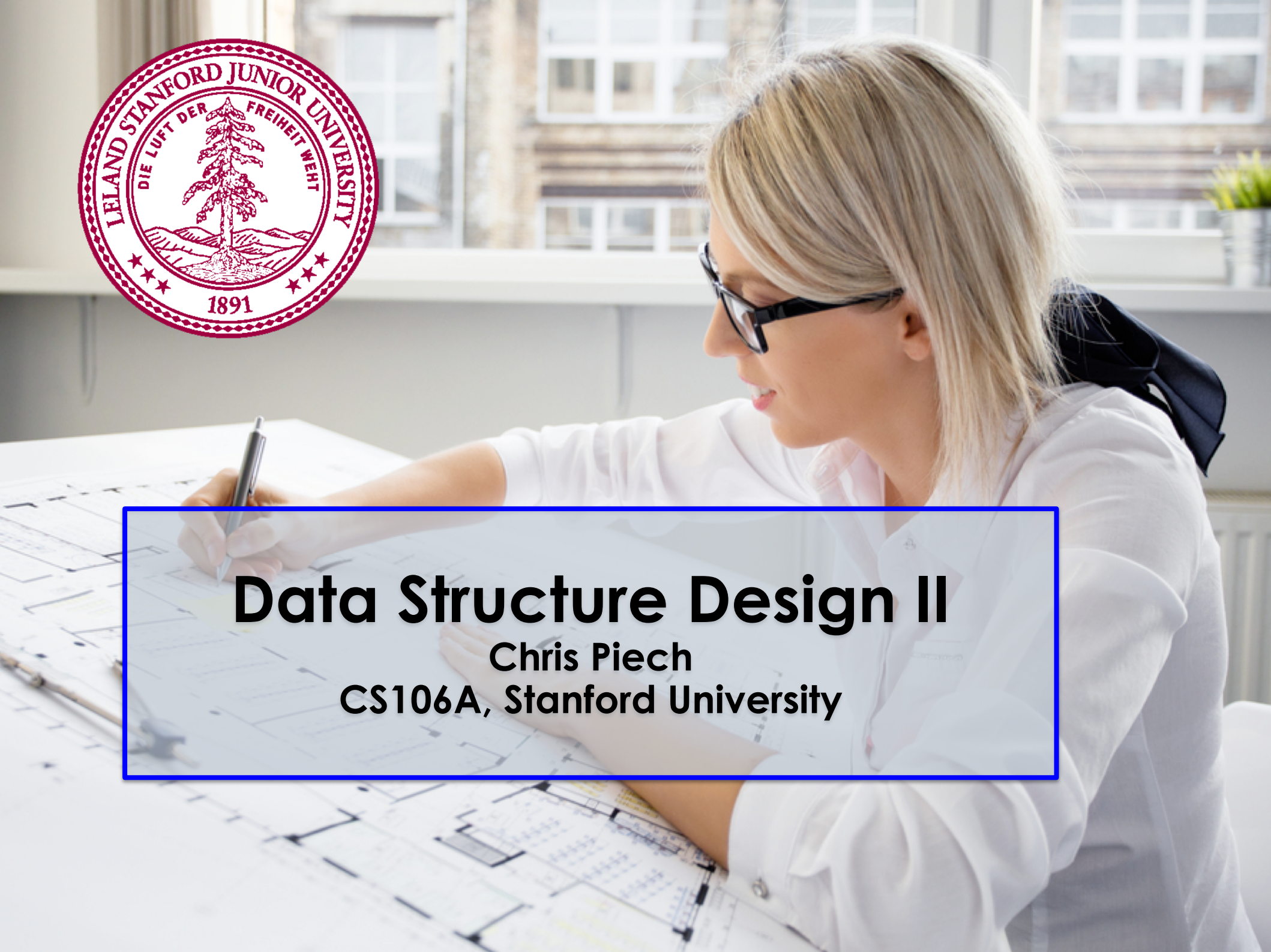
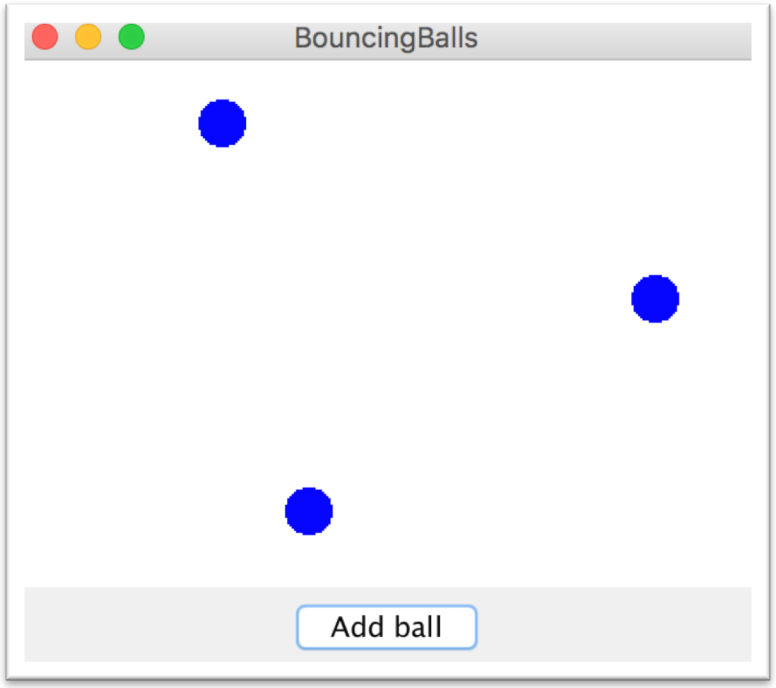


Data Structure Design II

Chris Piech
CS106A, Stanford University

Today in lecture



We have *used* many variable types

E.g. GRect

E.g. String

E.g. AudioSample

Today we learn how to define our own

We use Classes (written in new files) to defined new variable types

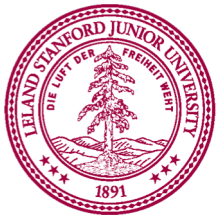
You must define three things

1. What **variables** does each instance store?

2. What **methods** can you call on an instance?

3. What happens when you make a **`new`** one?

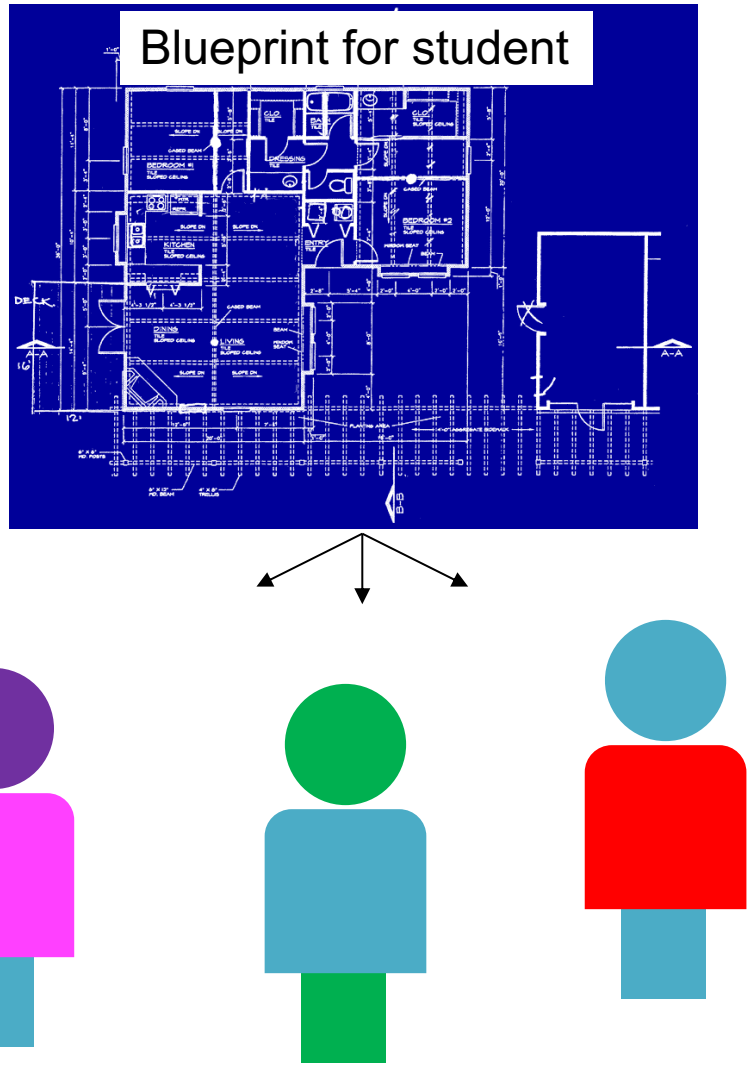
*details on how to define these three things coming soon



Classes are like blueprints

class: A template for a new type of variable.

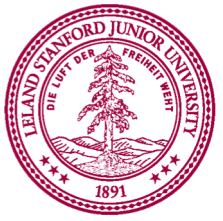
A blueprint is a helpful analogy



What Does GRect Look Like?

```
public class GRect extends GObject {
```

GRect.java

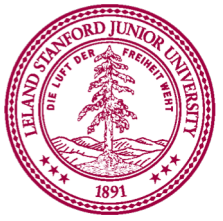


What Does GRect Look Like?

GRect.java

```
public class GRect extends GObject {

    // Instance variables
    private double width = 0;
    private double height = 0;
    private double yc = 0;
    private double xc = 0;
    private boolean isFilled = false;
    private boolean isVisible = false;
```

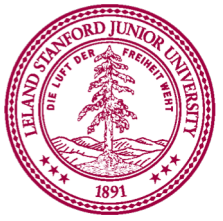
...



What Does GRect Look Like?

GRect.java

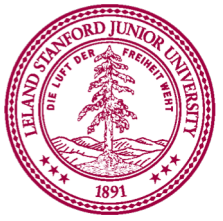
```
public class GRect extends GObject {

   // Instance variables
   private double width = 0;
   private double height = 0;
   private double yc = 0;
   private double xc = 0;
   private boolean isFilled = false;
   private boolean isVisible = false;


   // Constructors
   public GRect(double width, double height) {
     this.width = width;
     this.height = height;
   }
   public GRect(double x, double y, double width, double height) {
       this.xc = x;
       this.yc = y;
       this.width = width;
       this.height = height;
   }

   ...
```

What Does GRect Look Like?

GRect.java

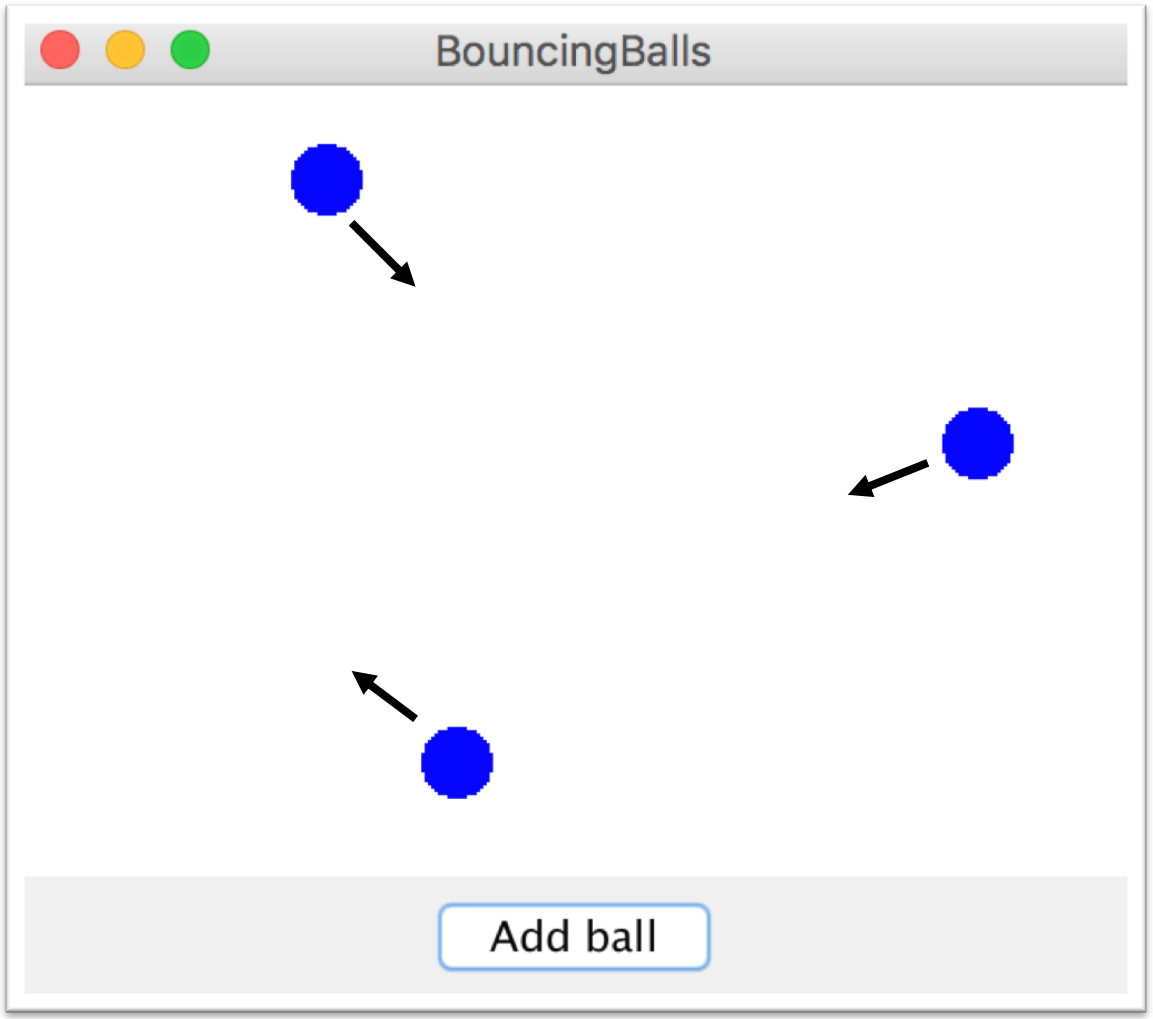
```
   ...

   // Public methods
   public double getWidth() {
      return this.width;
   }

   public double getHeight() {
      return this.height;
   }

   public void setFilled(boolean newIsFilled) {
      this.isFilled = newIsFilled;
   }

   public void move(double dx, double dy) {
      this.xc += dx;
      this.yc += dy;
   }

   ... and so on ...

}
```

What does a class do?

A class defines a new variable type

Bouncing Balls



A Ball Variable Type

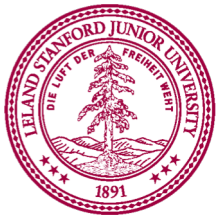
The Ball class

1. What **variables** does each instance store?
   - Each ball has its own Goval (lets call it shape)
   - Each ball has its own dx
   - Each ball has its own dy

2. What **methods** can you call on an instance?
   - `heartbeat();`
   - `getShape();`

3. What happens when you make a **`new`** one?
   - Sets initial values for all the ”instance” vars

*details on how to define these three things coming soon



Ball.java

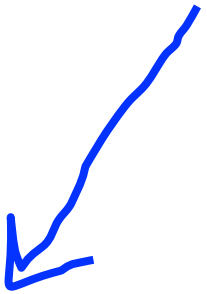
```
1
2
3  public class Ball {
4      /* instance vars! */
5
6      // each ball has a "shape"
7      private GOval shape = null;
8
9
10     // each ball has a dx
11     private double dx = 0.0;
12
13
14     // each ball has a dy
15     private double dy = 0.0;
16
17
18     ...
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
```

1. Instance vars define what makes up a variable of type Ball

Ball.java

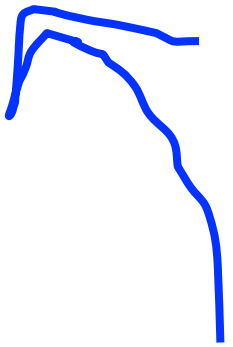
```
public class Ball {
    /* instance vars! */

    // each ball has a "shape"
    private GOval shape = null;

    // each ball has a dx
    private double dx = 0.0;

    // each ball has a dy
    private double dy = 0.0;

    // This defines what happens when you make a new ball
    public Ball(int screenWidth, int screenHeight) {
        RandomGenerator rg = RandomGenerator.getInstance();
        double x = rg.nextInt(screenWidth - BALL_SIZE);
        double y = rg.nextInt(screenHeight - BALL_SIZE);
        shape = new GOval(x, y, BALL_SIZE, BALL_SIZE);
        shape.setFilled(true);
        shape.setColor(Color.BLUE);
        dx = getRandomSpeed();
        dy = getRandomSpeed();
    }

    ...
```

2. The constructor defines what happens when you call new



Ball.java

```
50
51      public void heartbeat(int screenWidth, int screenHeight) {
52          shape.move(dx, dy);
53          reflectOffWalls(screenWidth, screenHeight);
54      }
55
56      public GOval getShape() {
57          return shape;
58      }
59
60      ...
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
```



3. Public methods define what methods the "client" can call on instances

Ball.java

```
50
51      public void heartbeat(int screenWidth, int screenHeight) {
52          shape.move(dx, dy);
53          reflectOffWalls(screenWidth, screenHeight);
54      }
55
56
57      public GOval getShape() {
58          return shape;
59      }
60
61
62      private void reflectOffWalls(int sWidth, int sHeight) {
63          if(shape.getY() < 0) {
64              dy *= -1;
65          }
66          if(shape.getY() > sHeight - BALL_SIZE) {
67              dy *= -1;
68          }
69          if(shape.getX() < 0) {
70              dx *= -1;
71          }
72          if(shape.getX() > sWidth - BALL_SIZE) {
73              dx *= -1;
74          }
75      }
76
77
78
79
80
81
82
```

4. Private methods are allowed

What does a class do?

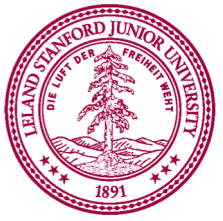
A class defines a new variable type

You must define three things

1. What **variables** does each instance store?

2. What **methods** can you call on an instance?

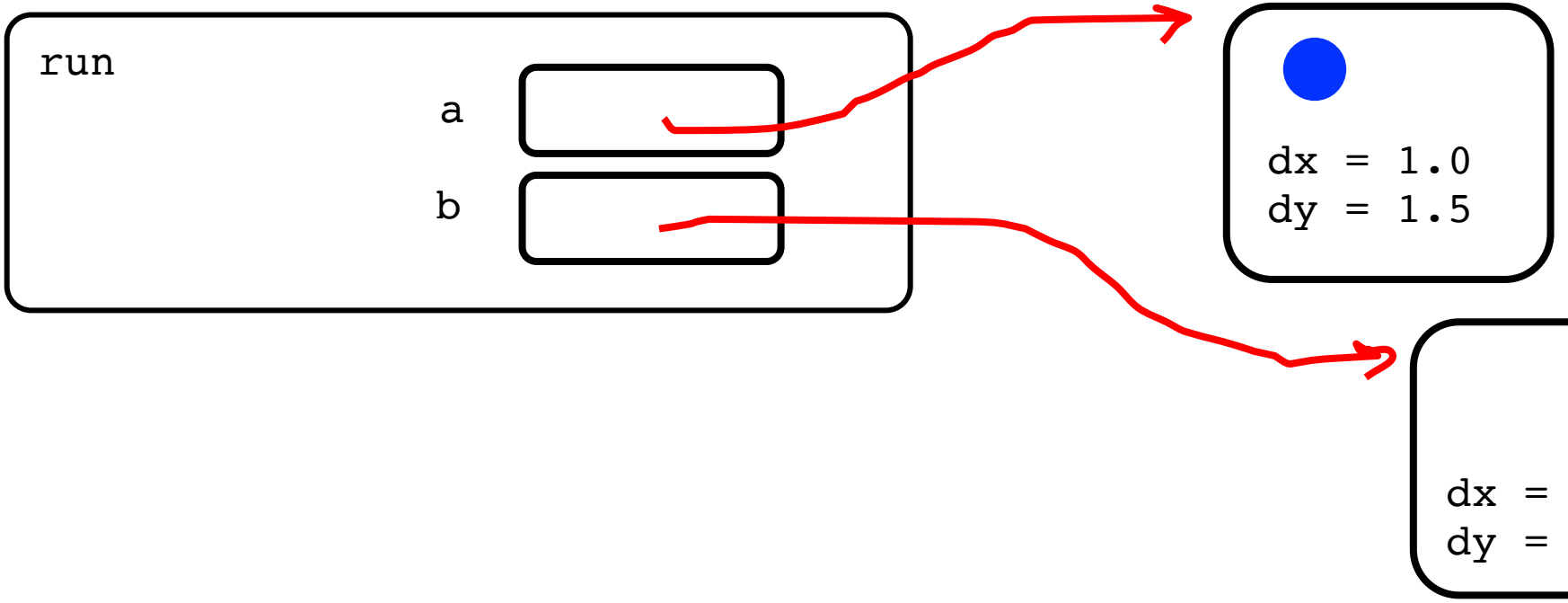
3. What happens when you make a **`new`** one?



Wait… if each instance has a copy of each instance variable. How does Java know which one to use?

Ball.java | BouncingBalls.java

```
1
2   public class BouncingBalls extends GraphicsProgram {
3       public void run() {
4           // make a few new balls
5           Ball a = new Ball(getWidth(), getHeight());
6           Ball b = new Ball(getWidth(), getHeight());
7
8           // call a method on one of the balls
9           a.heartbeat(getWidth(), getHeight());
10      }
11  }
12
13
14
15
```

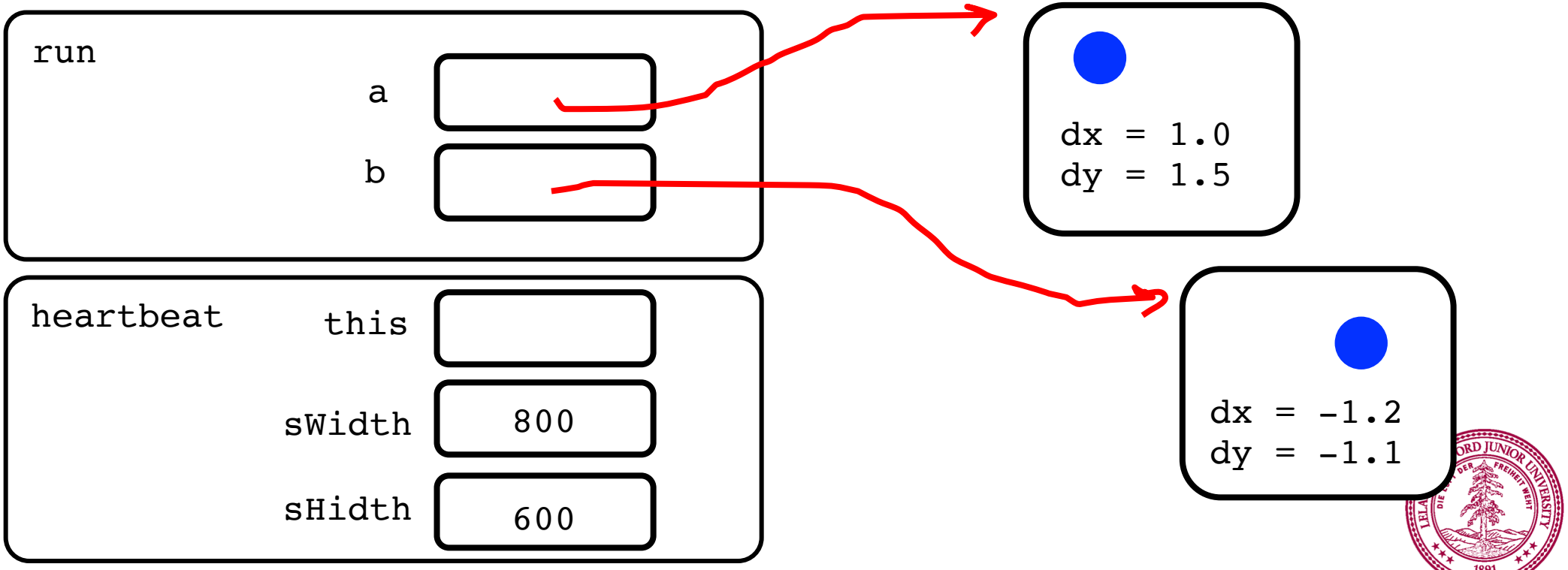


Ball.java | BouncingBalls.java

```
1
2   public class BouncingBalls extends GraphicsProgram {
```

Ball.java

```
50
51  public void heartbeat(int screenWidth, int screenHeight) {
52      shape.move(dx, dy);
53      reflectOffWalls(screenWidth, screenHeight);
54  }
55
56
57
58
```

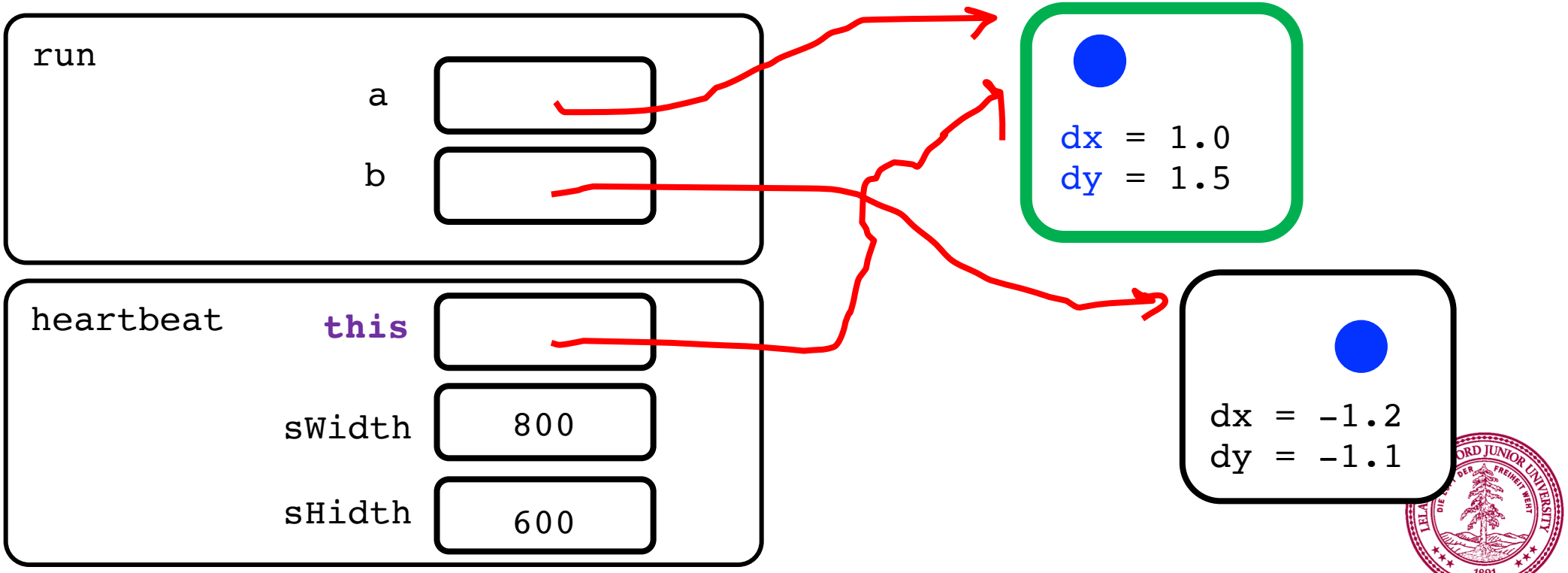


Ball.java | BouncingBalls.java

```
1
2   public class BouncingBalls extends GraphicsProgram {
```

Ball.java

```
50
51  public void heartbeat(int screenWidth, int screenHeight) {
52      shape.move(dx, dy);
53      reflectOffWalls(screenWidth, screenHeight);
54  }
55
56
57
58
```

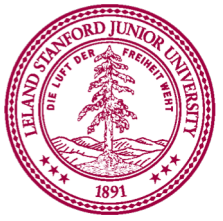


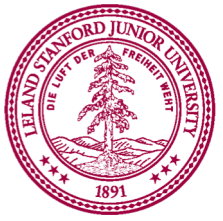
Tl;dr: Java knows which Ball you called heartbeat on

What Does GRect Look Like?

GRect.java

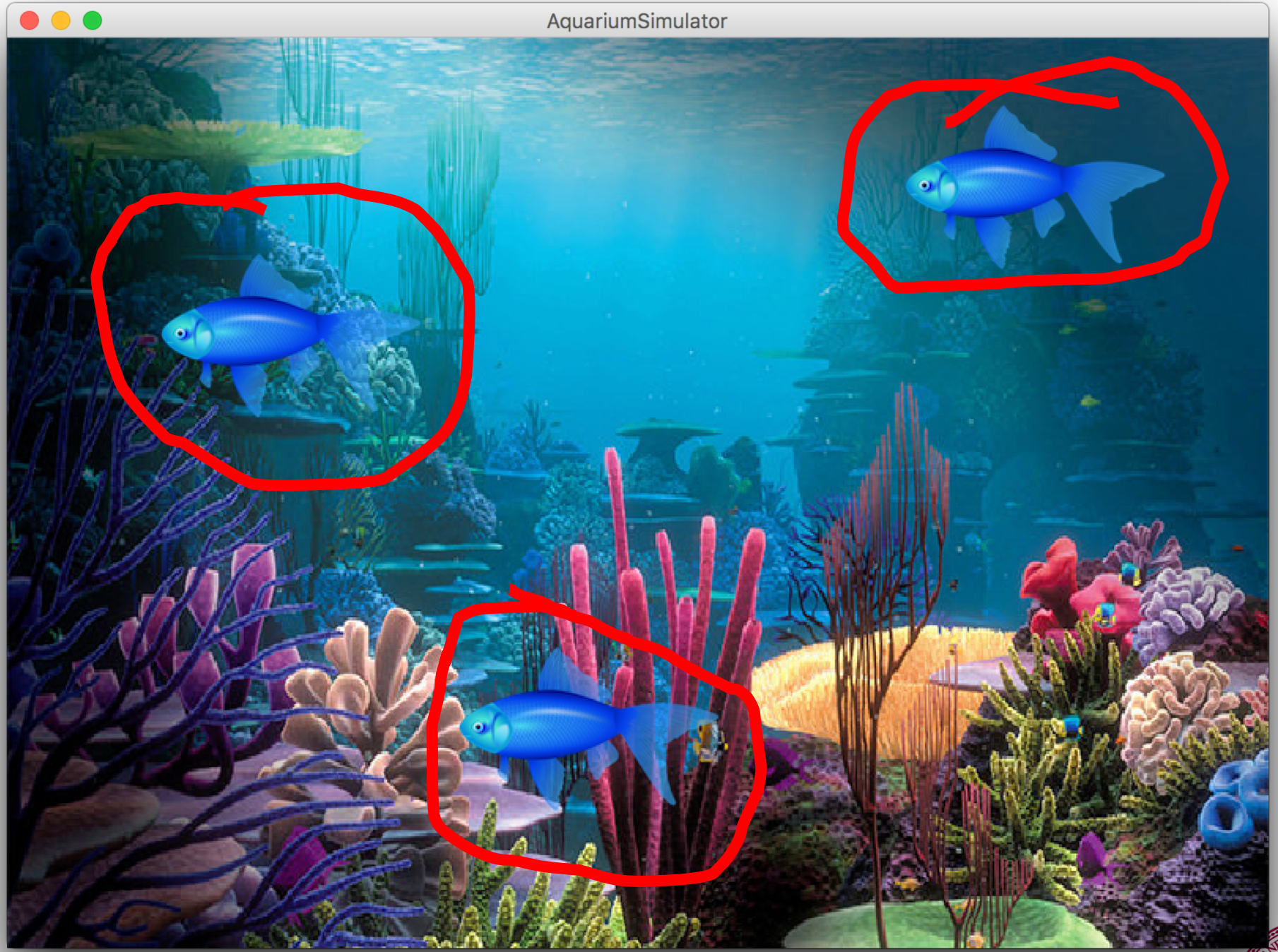
```
public class GRect extends GObject {

   // Instance variables
   private double width = 0;
   private double height = 0;
   private double yc = 0;
   private double xc = 0;
   private boolean isFilled = false;
   private boolean isVisible = false;


   // Constructors
   public GRect(double width, double height) {
     this.width = width;
     this.height = height;
   }
   public GRect(double x, double y, double width, double height) {
       this.xc = x;
       this.yc = y;
       this.width = width;
       this.height = height;
   }

   ...
```

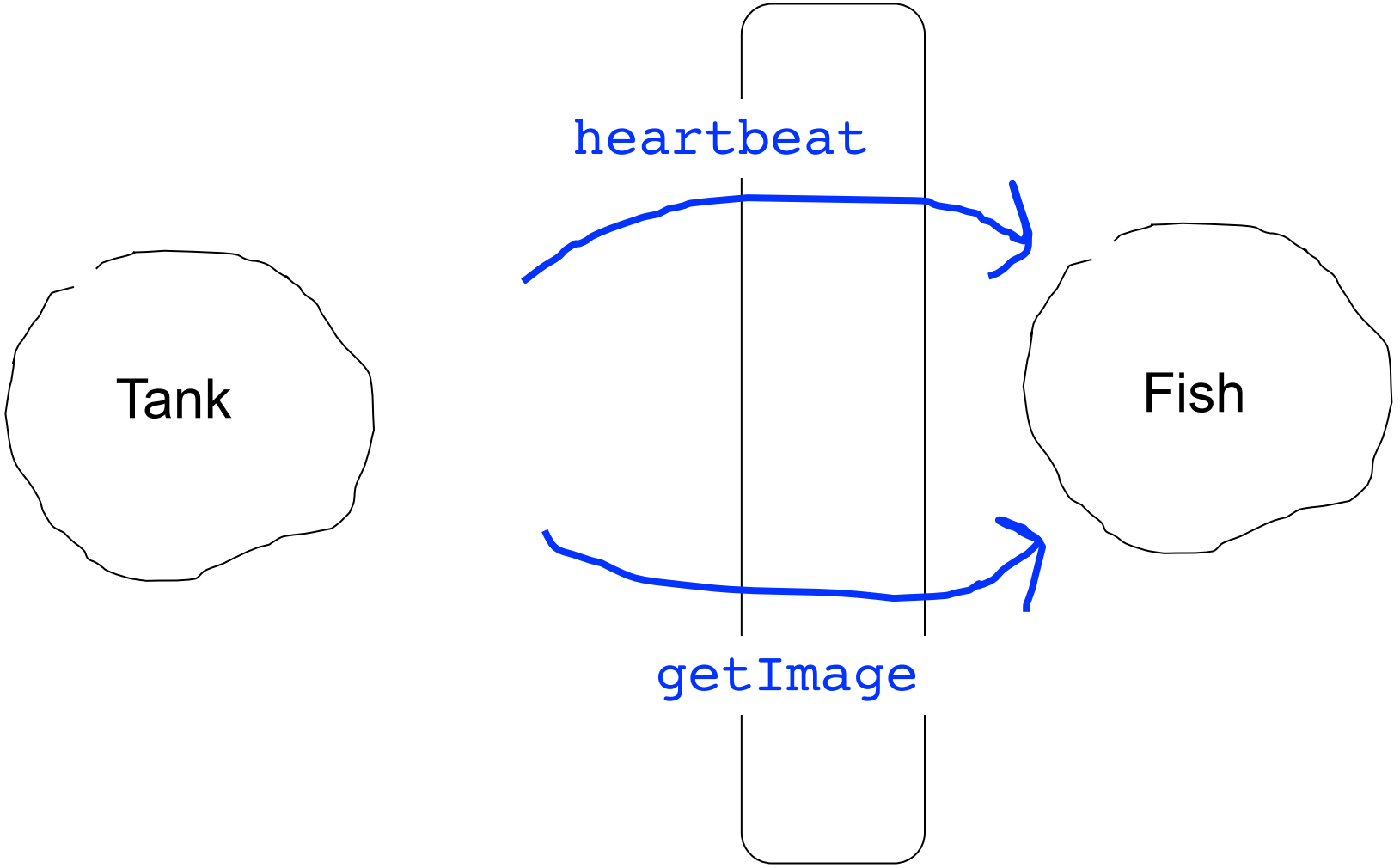




LELAND STANFORD JUNIOR UNIVERSITY
DIE LUFT DER FREIHEIT WEHT
1891

SpreadTheWord

```
To: [redacted]@stanford.edu
Subject: Hello from lecture
Text:
Dear [redacted]

I hope this email finds you well.

As you know, CS106A is a huge class with many wonderful people in it. In lecture today we built a
program to help you meet a few fellow students. Here are five random people in CS106A. You can
(optionally) introduce yourself:
 Omar, [redacted]i@stanford.edu
 Micah, [redacted]@stanford.edu
 Gianfranco, [redacted]e@stanford.edu
 Noam, [redacted]v@stanford.edu
 Dylan, [redacted]@stanford.edu

All the best,
Chris

P.S. Today we covered 'classes' which introduces a whole new way of thinking about programs
```


AquariumSimulator



Wall of abstraction

Adding Privacy

```
private boolean isLeftImgShown;
```

- **encapsulation**: Hiding implementation details of an object from its clients.
  - Encapsulation provides *abstraction*.
    - separates external view (behavior) from internal view (state)
  - Encapsulation protects the integrity of an object's data.

- A class's instance variables should be declared *private*.
  - No code outside the class can access or change it.

What does a class do?

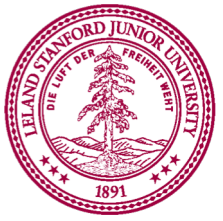
A class defines a new variable type

You must define three things

1. What **variables** does each instance store?

2. What **methods** can you call on an instance?

3. What happens when you make a **`new`** one?



More Practice

See Days Until