Announcements

- Assignment 5 due Monday August 5th at 10AM
Plan for Today

- Review: Classes
- Bouncing Ball
- Emailer
What do we know about classes?
A class defines a new variable type.
Classes Are Like Blueprints

**Hedgehog Class (blueprint)**

State:  
- Has name
- Has color
- Has cuteness level

Behavior:  
- Can eat
- Can run*
- Can curl up

---

**Hedgehog #1 (variable)**

State:  
- name = “Walnoot”
- color = Brown
- cuteness = 10 (Very cute)

Behavior:  
- Can eat
- Can run
- Can curl up

**Hedgehog #2 (variable)**

State:  
- name = “Nutmeg”
- color = Snowflake
- cuteness = 15 (VERY cute)

Behavior:  
- Can eat
- Can run
- Can curl up

**Hedgehog #3 (variable)**

State:  
- name = “Ruffles”
- color = Beige
- cuteness = 50 (speechless)

Behavior:  
- Can eat
- Can run
- Can curl up
Making a Class ~ 3 Ingredients

1. Define the **variables** each instance stores (state)
2. Define the **constructor** used to make a new instance
3. Define the **methods** you can call on an instance (behaviors)
You’ve seen them before...
GRect `square` = new GRect(10, 10);

- `type` our object (variable)
- It's an instance of the GRect class!
double x = square.getX()

Method defined in GRect class that we can call on our object
Unpacking GRect

public class GRect {  
    private double width;  
    public GRect(double width, double height) {  
        ...  
    }  
    ...  
}
public class GRect {

3 Ingredients:
public class GRect {

    // 1. 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;

    GRect.java

3 Ingredients:

1. Define the variables each instance stores
public class GRect {

    // 1. 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;

    // 2. Constructor(s)
    public GRect(double width, double height) {
        this.width = width;
        this.height = height;
    }

    3 Ingredients:

    1. Define the **variables** each instance stores

    2. Define the **constructor** used to make a **new** instance
public class GRect {

    // 1. 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;

    // 2. Constructor(s)
    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;
    }
}

3 Ingredients:

1. Define the variables each instance stores

2. Define the constructor used to make a new instance
public class GRect {

    // 1. 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;

    // 2. Constructor(s)
    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;
    }

    // 3. 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;
    }
}

3 Ingredients:

1. Define the variables each instance stores
2. Define the constructor used to make a new instance
3. Define the methods you can call on an instance
Making our own classes
Bouncing Ball
Making a Ball variable type

1. Define the **variables** each instance stores (think: state/properties)
   - Each ball has its own GOval (let’s call it circle)
   - Each ball has its own dx
   - Each ball has its own dy

2. Define the **constructor** used to make a **new** instance
   - Set initial values for all the instance vars

3. Define the **methods** you can call on an instance (think: behaviors)
   - heartbeat()
   - getGOval()
public class Ball {

    private static final int BALL_SIZE = 20;

    // 1: what variables make up a ball?
    private GOval circle; // each ball has a GOval shape
    private double dx; // each ball has a dx
    private double dy; // each ball has a dy

    // 2. what happens when you make a new ball?
    public Ball() {
        this.circle = new GOval(0, 0, BALL_SIZE, BALL_SIZE);
        this.circle.setFilled(true);
        this.circle.setColor(Color.BLUE);
        this.dx = getRandomSpeed();
        this.dy = getRandomSpeed();
    }

    1. Instance variables define what makes up a variable of type Ball
public class Ball {

    private static final int BALL_SIZE = 20;

    // 1: what variables make up a ball?
    private GOval circle; // each ball has a GOval shape
    private double dx; // each ball has a dx
    private double dy; // each ball has a dy

    // 2. what happens when you make a new ball?
    public Ball() {
        // make the ball's circle
        this.circle = new GOval(0, 0, BALL_SIZE, BALL_SIZE);
        this.circle.setFilled(true);
        this.circle.setColor(Color.BLUE);

        // gets a random dx and a random dy
        this.dx = getRandomSpeed();
        this.dy = getRandomSpeed();
    }
}
3. Public methods define what the “client” can call on instances

```java
// 3. what methods can you call on a ball?
public GOval getGOval() {
    return this.circle;
}

public void heartbeat(int screenWidth, int screenHeight) {
    this.circle.move(this.dx, this.dy);
    reflectOffWalls(screenWidth, screenHeight);
}
```

private void reflectOffWalls(int screenWidth, int screenHeight) {
    if (this.circle.getY() < 0) {
        this.dy *= -1;
    }
    if (this.circle.getY() > screenHeight - BALL_SIZE) {
        this.dy *= -1;
    }
    if (this.circle.getX() < 0) {
        this.dx *= -1;
    }
    if (this.circle.getX() > screenWidth - BALL_SIZE) {
        this.dx *= -1;
    }
}
// private methods are allowed
private void reflectOffWalls(int screenWidth, int screenHeight) {
    if (this.circle.getY() < 0) {
        this.dy *= -1;
    }
    if (this.circle.getY() > screenHeight - BALL_SIZE) {
        this.dy *= -1;
    }
    if (this.circle.getX() < 0) {
        this.dx *= -1;
    }
    if (this.circle.getX() > screenWidth - BALL_SIZE) {
        this.dx *= -1;
    }
}

private double getRandomSpeed() {
    RandomGenerator rg = RandomGenerator.getInstance();
    double speed = rg.nextDouble(1,3);
    if (rg.nextBoolean()) {
        speed *= -1;
    }
    return speed;
}
But if each Ball instance has a copy of each instance variable...

... how does Java know which one to use?
* all class methods and constructors have access to a this reference
public class BouncingBall extends GraphicsProgram {
    public void run() {
        // make a few new bouncing balls
        Ball a = new Ball();
        Ball b = new Ball();

        // call a method on one of the balls
        a.heartbeat(getWidth(), getHeight());
    }
}
public class BouncingBall extends GraphicsProgram {
    public void run() {
        // make a few new bouncing balls
        Ball a = new Ball();
        Ball b = new Ball();

        // call a method on one of the balls
        a.heartbeat(getWidth(), getHeight());
    }
}
public class BouncingBall extends GraphicsProgram {
    public void run() {
        // make a few new bouncing balls
        Ball a = new Ball();
        Ball b = new Ball();

        // call a method on one of the balls
        a.heartbeat(getWidth(), getHeight());
    }
}
public class BouncingBall extends GraphicsProgram {
    public void run() {
        // make a few new bouncing balls
        Ball a = new Ball();
        Ball b = new Ball();

        // call a method on one of the balls
        a.heartbeat(getWidth(), getHeight());
    }
}
public class BouncingBall extends GraphicsProgram {
    public void run() {
        // make a few new bouncing balls
        Ball a = new Ball();
        Ball b = new Ball();

        // call a method on one of the balls
        a.heartbeat(getWidth(), getHeight());
    }
}

public void heartbeat(int sWidth, int sHeight) {
    this.circle.move();
    reflectOffWalls(sWidth, sHeight);
}

Stack frames

run()
    a 42
    b 52

heartbeat()
    this
    sWidth 800
    sHeight 600

heap

circle
    dx = 1.0
    dy = 1.5

circle
    dx = -1.2
    dy = -1.1
public class BouncingBall extends GraphicsProgram {
    public void run() {
        // make a few new bouncing balls
        Ball a = new Ball();
        Ball b = new Ball();

        // call a method on one of the balls
        a.heartbeat(getWidth(), getHeight());
    }
}

public void heartbeat(int sWidth, int sHeight) {
    this.circle.move();
    reflectOffWalls(sWidth, sHeight);
}

heartbeat() was called on ball a
⇒ So, this refers to a

Stack frames

- run()
  - a 
    - 42
  - b
    - 52

heartbeat()
- this
  - sWidth
    - 800
  - sHeight
    - 600

- this.circle
  - dx = 1.0
  - dy = 1.5

- ball a
  - circle
    - dx = -1.2
    - dy = -1.1
public class BouncingBall extends GraphicsProgram {
    public void run() {
        // make a few new bouncing balls
        Ball a = new Ball();
        Ball b = new Ball();

        // call a method on one of the balls
        a.heartbeat(getWidth(), getHeight());
    }
}

public void heartbeat(int sWidth, int sHeight) {
    this.circle.move();
    reflectOffWalls(sWidth, sHeight);
}

heartbeat() was called on ball a
⇒ So, this refers to a
public class BouncingBall extends GraphicsProgram {
    public void run() {
        // make a few new bouncing balls
        Ball a = new Ball();
        Ball b = new Ball();

        // call a method on one of the balls
        a.heartbeat(getWidth(), getHeight());
    }
}

public void heartbeat(int sWidth, int sHeight) {
    this.circle.move();
    reflectOffWalls(sWidth, sHeight);
}

heartbeat() was called on ball a ⇒ So, this refers to a
Java knows **which instance** you called a method on
One more note

```java
ArrayList<Ball> balls

Index 0  Index 1  Index 2
42       52       62

42: circle
dx = -1.2
dy = -1.1

52: circle
dx = 2.0
dy = 1.5

62: circle
dx = 1.8
dy = -2.2
```

Array List of Ball objects named `balls`
```java
ArrayList<Ball> balls = new ArrayList<>();

// Add balls to the list
balls.add(new Ball(42, -1.2, -1.1));
balls.add(new Ball(52, 2.0, 1.5));
balls.add(new Ball(62, 1.8, -2.2));

// Create a new ball
Ball newBall = new Ball();
```
balls.append(newBall)
balls.append(newBall)
Let’s build something bigger
Sending emails...

toAddress: [email protected]

subject: Greetings from Lecture

body: Dear [Name],

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:

- Elena, [email protected]
- Moritz, [email protected]
- Georgiana, [email protected]
- Yazan, [email protected]
- Jose, [email protected]

All the best,
Laura & Sarai

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

- Review: Classes
- Bouncing Ball
- Emailer

Next Time: Interactors