Outline

- **Objects revisited**
  - Instance variables
  - Instance methods
  - Declaring and creating
- **Primitives variables**
  - Different size bit patterns in memory
- **Reference variables**
  - Remote control to an object
  - Of aliases and orphans
Classes and Objects

Class = object blueprint

Objects = different instances of a class
Hey objects, what do you know?

"I'm in the center towards the bottom. I'm pretty big and orange, like a big pumpkin!"

"I'm over to the left, midway up, I'm kinda small, and blue 😞"

"I'm way over on the right, higher than the other guys. I'm also totally mauve! 😊"

```java
public class Ball {
    private double posX = 0.0;
    private double posY = 0.0;
    private double radius = 0.0;
    private Color color = new Color(0.88f, 0.68f, 1.0f);
}
```

Instance variables = what an object knows (state)
"Hey objects, what can you do?"

"We can draw ourselves, print our data, change our color, move around, and even see if we overlap with some other guy!"

- **0.1, 0.5, r=0.1**
- **0.5, 0.3, r=0.2**
- **0.9, 0.6, r=0.15**

---

**Instance methods** = what an object can do (behavior)

```java
public void draw()
{
    ...
}

public String toString()
{
    ...
}

public void setColor(double r, double g, double b)
{
    ...
}

public void move(double deltaX, double deltaY)
{
    ...
}

public boolean overlap(Ball other)
{
    ...
}
```
import java.awt.*;

public class Ball {
    private double posX = 0.0;
    private double posY = 0.0;
    private double radius = 0.0;
    private Color color = new Color(0.88f, 0.68f, 1.0f);

    public void move(double deltaX, double deltaY) {
        posX += deltaX;
        posY += deltaY;
    }
    ...
}
Hey objects, where did you come from?

"Dude, don't you know where objects come from?!? The object stork totally dropped us off."

```java
public Ball(double x, double y, double r) {
    posX = x;
    posY = y;
    radius = r;
}
```

Constructor = the object stork
public class BallSoapOpera
{
    public static void main(String [] args)
    {
        Ball bluey;
    }
}
An object soap opera

```java
public class BallSoapOpera {
    public static void main(String[] args) {
        Ball bluey;
        bluey = new Ball(0.1, 0.5, 0.1);
    }
}
```

"Yay thank you object stork. At long last, I'm finally a real Ball! Though I seem to be invisible."
public class BallSoapOpera
{
    public static void main(String [] args)
    {
        Ball bluey;
        bluey = new Ball(0.1, 0.5, 0.1);
        bluey.draw();
    }
}

"That's great! Now everybody can see me. But my color is a little girly..."
public class BallSoapOpera
{
    public static void main(String[] args)
    {
        Ball bluey;
        bluey = new Ball(0.1, 0.5, 0.1);
        bluey.setColor(0.0, 0.0, 1.0);
        bluey.draw();
    }
}
public class BallSoapOpera
{
    public static void main(String [] args)
    {
        Ball bluey;
        bluey = new Ball(0.1, 0.5, 0.1);
        bluey.setColor(0.0, 0.0, 1.0);
        bluey.draw();
        Ball mauevy = new Ball(0.1, 0.9, 0.15);
        mauevy.draw();
    }
}

"Well hello there, what's your name? Why don't you come over here?"
public class BallSoapOpera
{
    public static void main(String [] args)
    {
        Ball bluey;
        bluey = new Ball(0.1, 0.5, 0.1);
        bluey.setColor(0.0, 0.0, 1.0);
        bluey.draw();

        Ball mauvey = new Ball(0.1, 0.9, 0.15);
        mauvey.draw();

        while (!mauvey.overlap(bluey))
        {
            mauvey.move(0.0, -0.01);
            mauvey.draw();
        }
    }
}
public class BallSoapOpera
{
    public static void main(String [] args)
    {
        Ball bluey;
        bluey = new Ball(0.1, 0.5, 0.1);
        bluey.setColor(0.0, 0.0, 1.0);
        bluey.draw();

        Ball mauvey = new Ball(0.1, 0.9, 0.15);
        mauvey.draw();

        while (!mauvey.overlap(bluey))
        {
            mauvey.move(0.0, -0.01);
            mauvey.draw();
        }
    }
}
public class BallSoapOpera
{
    public static void main(String [] args)
    {
        Ball bluey;
        bluey = new Ball(0.1, 0.5, 0.1);
        bluey.setColor(0.0, 0.0, 1.0);
        bluey.draw();

        Ball mauevy = new Ball(0.1, 0.9, 0.15);
        mauevy.draw();

        while (!mauevy.overlap(bluey))
        {
            mauevy.move(0.0, -0.01);
            mauevy.draw();
        }
        bluey.setColor(1.0, 0.0, 0.0);
        bluey.draw();
    }
}
Declaring a variable

- **All variables must have a type**
  - **Primitive types:** hold fundamental values
    - integers, booleans, floating-point values
    - type name is all lowercase
    - `int`, `double`, `boolean`, `char`, `byte`, `short`, `long`, `float`
  - **Object reference types:** refer to an object
    - May contain several values
    - Type name starts in uppercase (by our style convention)
    - `e.g. String, Color, Ball, Dog, Giraffe, ...`
Primitive type sizes

- **Primitive types**
  - Just a block of memory in your computer
  - Size of block measured in bits (number of 0s or 1s)
  - Integers:

<table>
<thead>
<tr>
<th>type</th>
<th>bits</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8</td>
<td>0110 1110</td>
</tr>
<tr>
<td>short</td>
<td>16</td>
<td>0110 1110 1101 1101</td>
</tr>
<tr>
<td>int</td>
<td>32</td>
<td>0101 1001 0000 0001 0111 1101 0110 0010</td>
</tr>
<tr>
<td>long</td>
<td>64</td>
<td>1101 0011 1001 0001 1101 0101 1010 0101 0111 1010 0011 1010 1011 1100 1111 1111</td>
</tr>
</tbody>
</table>
Creating a primitive variable

byte x = 7;
Creating a primitive variable

```java
byte x = 7;
x = x + 1;
```

<table>
<thead>
<tr>
<th>x</th>
<th>00001000</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>primitive value</td>
</tr>
</tbody>
</table>

01101010101010010101010111010101010101010101011101010101111010101010101010101010101001001101010101010110000010000101010101011101010100010101010111110101010101010101010101011000100010101101010101111010101101010101010101010101001001101010101010010101010111010101010101110101010101010101001001101010101010010101010111010101010101110101010101010101001001101010101010010101010111010101010101110101010001010101010101011110101010101010101010100101010011010101010010101010111010101010101110101010001010101010101111010101010101010101010010011010101010100101010
Creating a primitive variable

byte x = 7;
x = x + 1;
short y = 7;
You can't put a big cup into a small one

You may know 7 can fit in a byte, but compiler doesn't!

byte  x = 7;
  x = x + 1;
short y = 7;
  x = y;

You can't put a big cup into a small one
Declaring a reference variable

Ball b;

Currently b is equal to null.

References variables always need to be used new to create an actual object.

Analogy: unprogrammed universal remote
Creating a reference variable

Ball b = new Ball(0.0, 0.0, 0.5);

bits for Ball at (0,0) r=0.5

bits that get us to a Ball object
Creating a reference variable

```
Ball b = new Ball(0.0, 0.0, 0.5);

Color c = new Color(0.0f, 0.0f, 1.0f);
```
References variables can't switch types

```java
Ball b = new Ball(0.0, 0.0, 0.5);

Color c = new Color(0.0f, 0.0f, 1.0f);

b = c;
```

You can't put a Color object into a Ball reference variable!
Two references of same type

```java
Ball b = new Ball(0.0, 0.0, 0.5);
Ball b2 = new Ball(0.0, 0.0, 0.7);
```
Two references of same type

Ball b = new Ball(0.0, 0.0, 0.5);
Ball b2 = new Ball(0.0, 0.0, 0.7);

b = b2;
Two names: one object

```java
Ball b = new Ball(0.0, 0.0, 0.5);
Ball b2 = new Ball(0.0, 0.0, 0.7);

b = b2;
b.setColor(1.0, 0.0, 0.0);
```
Two names: one object

```java
Ball b = new Ball(0.0, 0.0, 0.5);
Ball b2 = new Ball(0.0, 0.0, 0.7);
b = b2;
b2.setColor(1.0, 0.0, 0.0);
```

Currently `b` and `b2` are just aliases: different names for controlling the same object. Calling a method on `b` is same as calling the same method on `b2`. 

```
bits for Ball at (0,0) r=0.5
```

```
bits for Ball at (0,0) r=0.7
```

```
b2
bits that get us to a Ball object
```

```
b
bits that get us to a Ball object
```
Ball b = new Ball(0.0, 0.0, 0.5);
Ball b2 = new Ball(0.0, 0.0, 0.7);

b = b2;
b2.setColor(1.0, 0.0, 0.0);

The Ball object at (0,0) r=0.5 has become an orphan (no one can control it anymore). The Java garbage collector eventually frees up the memory.
Reference variables can be reprogrammed

```java
Ball b = new Ball(0.0, 0.0, 0.5);
Ball b2 = new Ball(0.0, 0.0, 0.7);

b = b2;
b2.setColor(1.0, 0.0, 0.0);

b2 = new Ball(1.0, 1.0, 0.5);
```

b2 now refers to a brand new Ball object at a new location (1,1). b2 forgets how to control Ball at (0,0). But b still can.
Alias bug'o'rama

- **Instance variables have a name**
  - So do parameters to methods
  - So do local variables
  - **Be careful:** Java lets you use the same name!

```java
public class Ball {
    private double posX = 0.0;
    private double posY = 0.0;
    private double radius = 0.0;

    public Ball(double x, double y, double r) {
        posX = x;
        posY = y;
        radius = r;
    }
    ...
}
```

This class works just fine.

The instance variables and the parameters to the constructor method `Ball()` all use different names.

No confusion!
Alias bug'o'rama

- **Local variables**
  - If same name as instance variable → Java uses the local variable

```java
public class Ball {
    private double posX = 0.0;
    private double posY = 0.0;
    private double radius = 0.0;

    public Ball(double x, double y, double r) {
        double posX = x;
        double posY = y;
        double radius = r;
    }
    ...
}
```

This will compile and run, but the instance variables will all remain 0.0.

In the Ball() constructor, posX means the local variable not the instance variable.
**Parameter to method**
- If **same name as instance variable** → Java uses the parameter variable

```java
public class Ball {
    private double posX = 0.0;
    private double posY = 0.0;
    private double radius = 0.0;

    public Ball(double posX, double posY, double radius) {
        posX = posX;
        posY = posY;
        radius = radius;
    }
    ...  
}
```

This will compile and run, but the instance variables will all remain 0.0.

In the `Ball()` constructor, `posX` means the parameter variable not the instance variable.
this to the rescue

- **this**
  - Refers to the instance of the object running the method
  - Use instance variable instead of local variable

```java
public class Ball {
    
    private double posX = 0.0;
    private double posY = 0.0;
    private double radius = 0.0;

    public Ball(double posX, double posY, double radius) {
        this.posX = posX;
        this.posY = posY;
        this.radius = radius;
    }

    ...
}
```

This works just fine. Using **this** allows you to have the same parameter variables names as your instance variables (if you want).
Multiple `main()` methods

- Every Java class can have a `main()`
  - java MyClass → runs `main()` in MyClass.java
  - Often used to test and debug a class

```java
public class Ball {
    ...
    public static void main(String [] args) {
        Ball a = new Ball(0.5, 0.5, 0.2);
        Ball b = new Ball(0.5, 0.5, 0.2);
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("a overlaps b = " + a.overlap(b));
        a.move(0.5, 0.0);
        System.out.println("a = " + a);
        System.out.println("a overlaps b = " + a.overlap(b));
        a.draw();
        b.draw();
    }
    ...
}
```
• Classes and objects

Class =

Object =

Instance variables =

Instance methods =

Constructor =
Summary – Part 1

- Classes and objects
  - Class = object blueprint
  - Object = instances of a class
  - Instance variables = what an object knows
  - Instance methods = what an object can do
  - Constructor = object stork

- Primitive and reference variables
  - Aliased objects, orphaned objects

- Alias bugs

- Every class can have main()
Summary

- **Objects revisited**
  - Instance variables
  - Instance methods
  - Declaring and creating

- **Primitives variables**
  - Different size bit patterns in memory

- **Reference variables**
  - Remote control to an object
  - Of aliases and orphans
Your Turn

- Create a class called Dog.java. Dogs know their name, breed and the sound they make. Dogs know how to make a sound (“Bark”). You don’t need to use actual sounds for this – just print out the string for that sound. You will need some way to construct a dog, and some way to print out its sound.

- Create a DogClient.java that creates three dogs of different breeds, names and sounds.

- Three points possible – 1 for turning something in, 1 for correct Dog class, and 1 for correct DogClient class.

- Submit your files to the Moodle Activity 06 dropbox for this week.