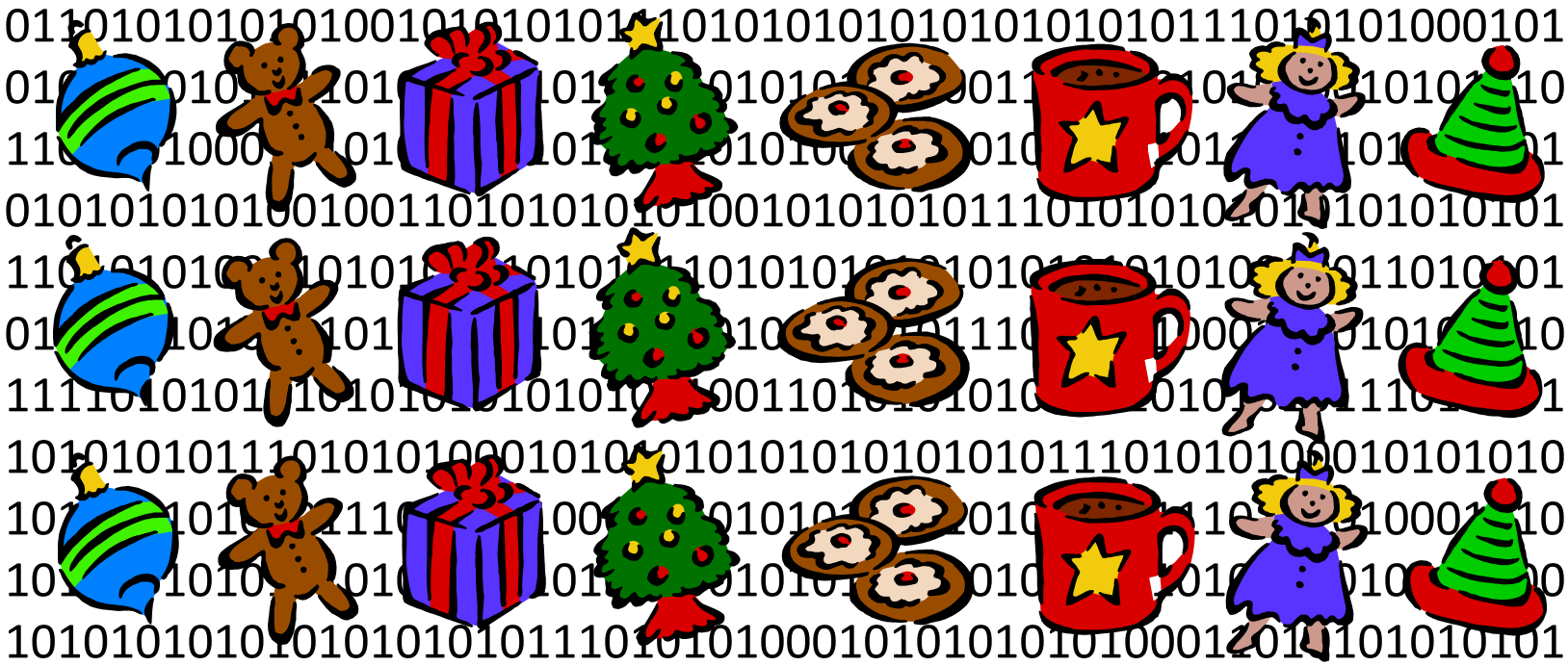
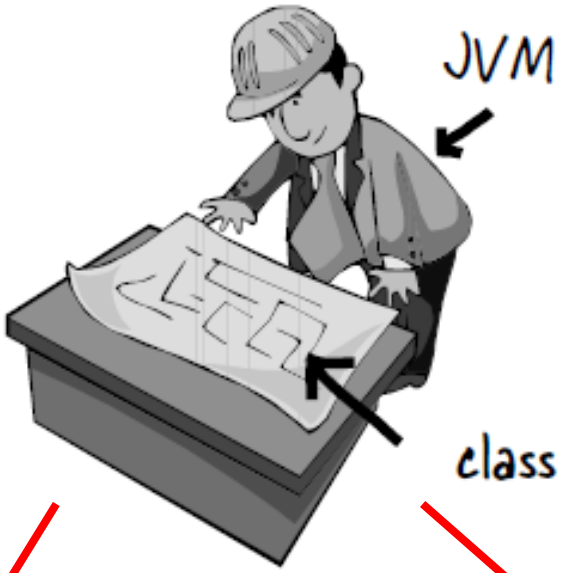
Objects, primitives and references



Overview

- 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

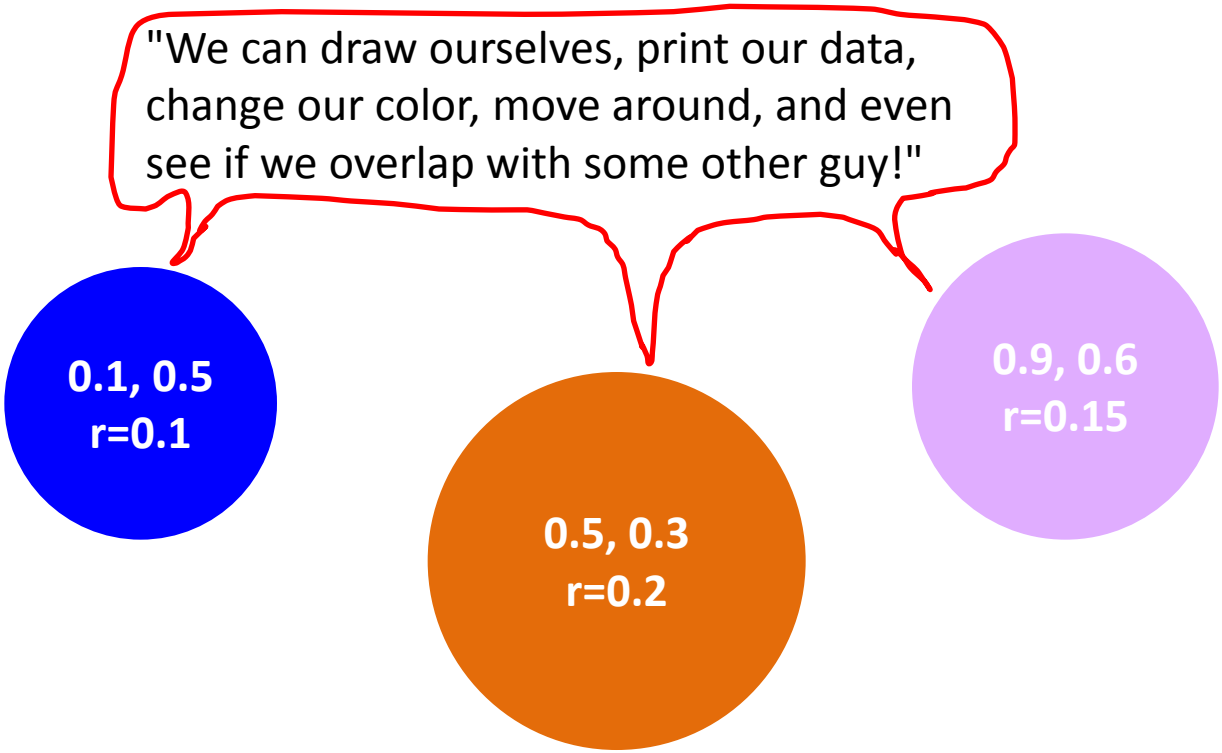
Objects = different instances of a class

Hey objects, what do you know?


```
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

Hey objects, what can you do?




```
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)                          {...}
```

Instance methods = what an object can do

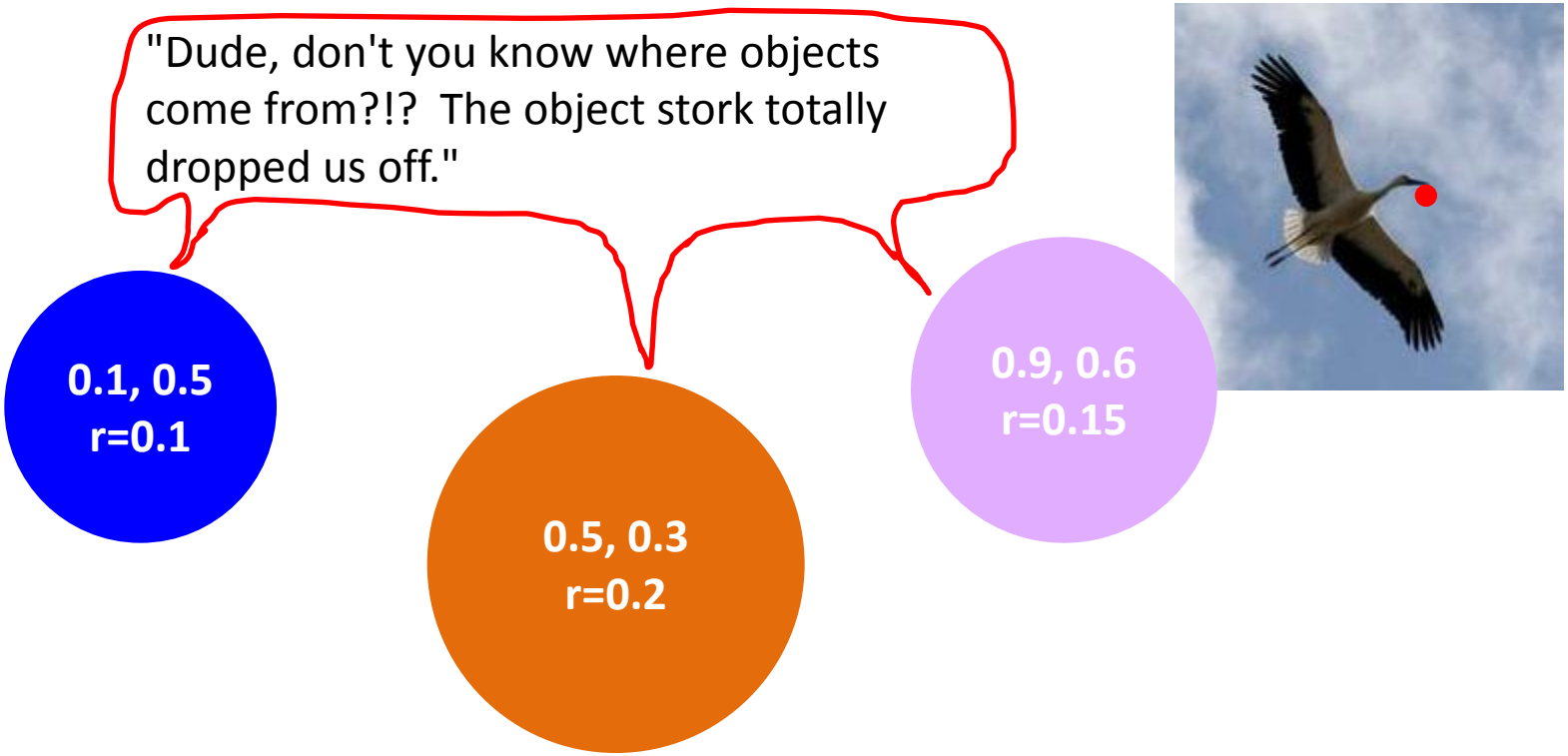
Bonus move method

```
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?




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

Constructor = the object stork

An object soap opera

```
public class BallSoapOpera
{
   public static void main(String [] args)
   {
      Ball bluey;
   }
}
```

"Cruel cruel world. I'm a variable, but I have no purpose in life. I'm so worthless, a sad empty vessel…"

An object soap opera

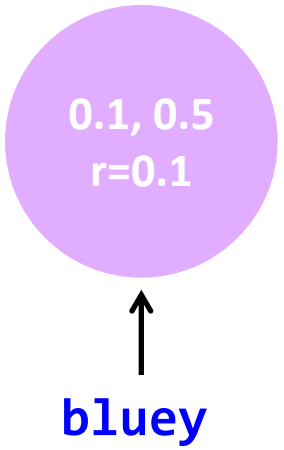
```
public class BallSoapOpera
{
   public static void main(String [] args)
   {
      Ball bluey;
      bluey = new Ball(0.1, 0.5, 0.1);
   }
}
```

bluey

"Yay thank you object stork. At long last, I'm finally a real Ball! Though I seem to be invisible."

An object soap opera

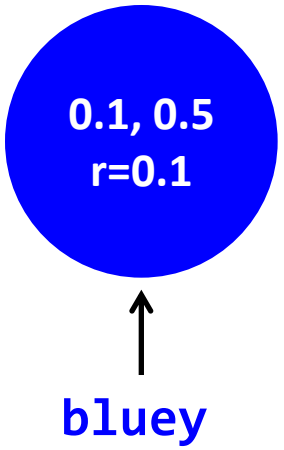
```
public class BallSoapOpera
{
   public static void main(String [] args)
   {
      Ball bluey;
      bluey = new Ball(0.1, 0.5, 0.1);
      bluey.draw();
   }
}
```



"That's great! Now everybody can see me. But my color is a little girly…"

An object soap opera

```
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();
   }
}
```

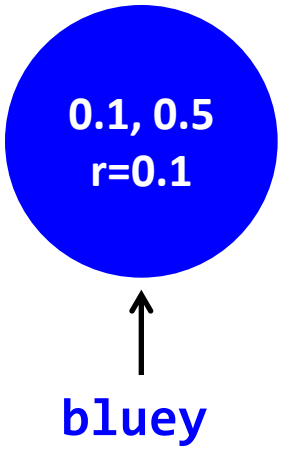


" Awh that's better, a nice manly blue, just like I like it!"

An object soap opera

```
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 beautiful? Why don't you come over here?"

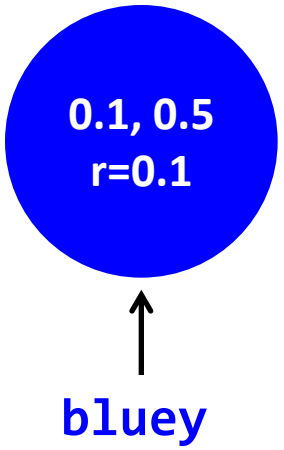
An object soap opera

```
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();
      }
   }
}
```

"<giggle> Well hello to you too handsome…"

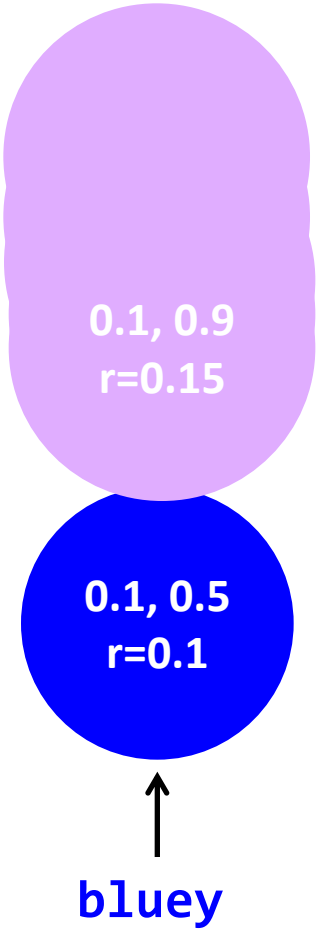


An object soap opera

```
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();
      }
   }
}
```

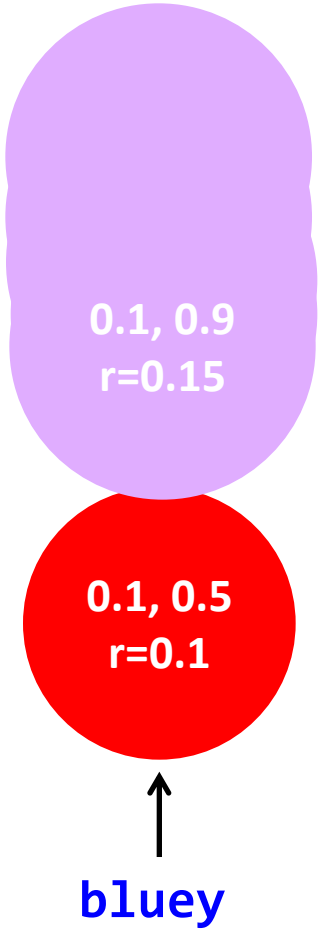


An object soap opera

```
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
    - e.g. `String`, `Color`, `Ball`, `Dog`, `Giraffe`, `...`

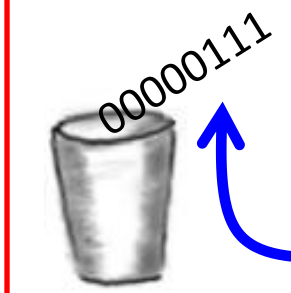
Primitive type sizes

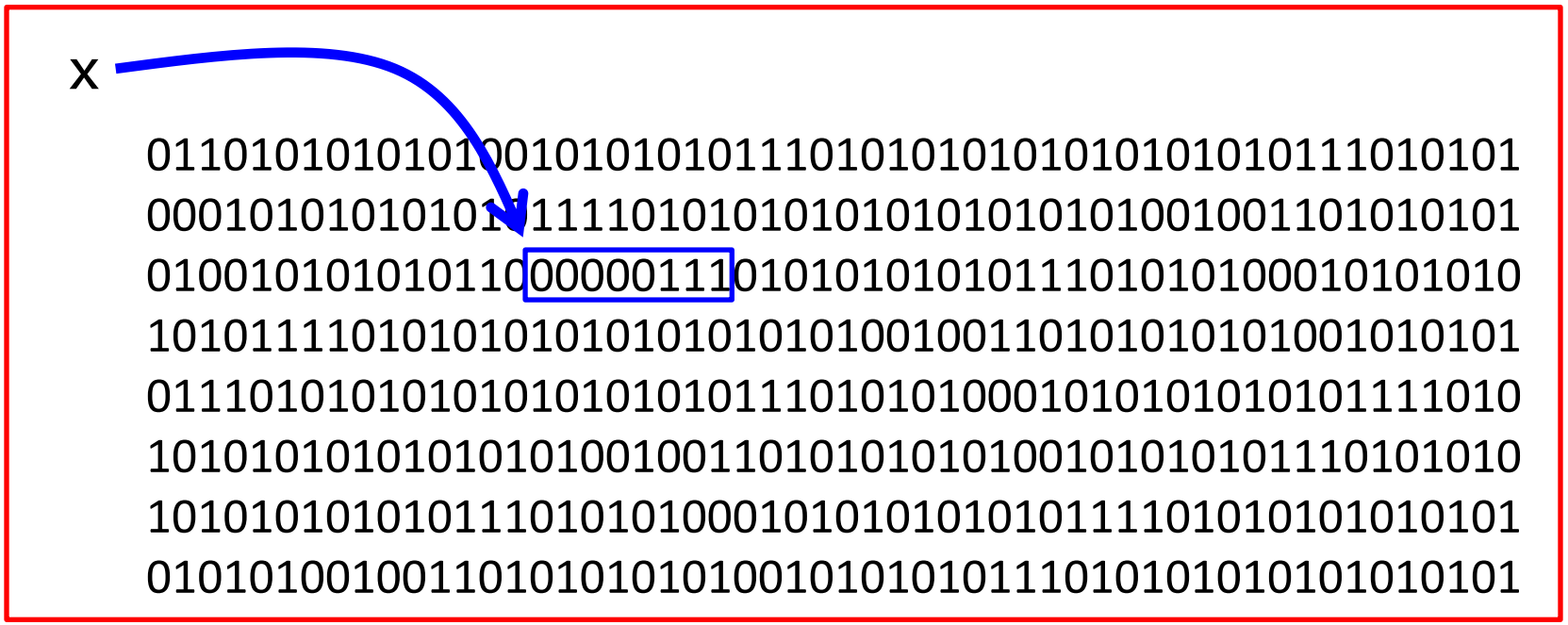
- Primitive types
  - Just a block of memory in your computer
  - Size of block measured in bits (number of 0s or 1s)
  - Integers:

type	bits	example
`byte`	8	0110 1110
`short`	16	0110 1110 1101 1101
`int`	32	0101 1001 0000 0001 0111 1101 0110 0010
`long`	64	1101 0011 1001 0001 1101 0101 1010 0101 0111 1010 0011 1010 1011 1100 1111 1111

Creating a primitive variable

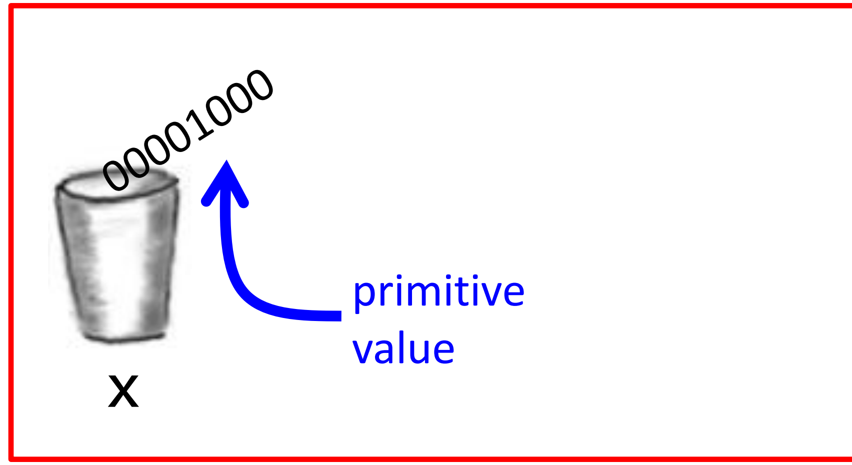
```
byte x = 7;
```

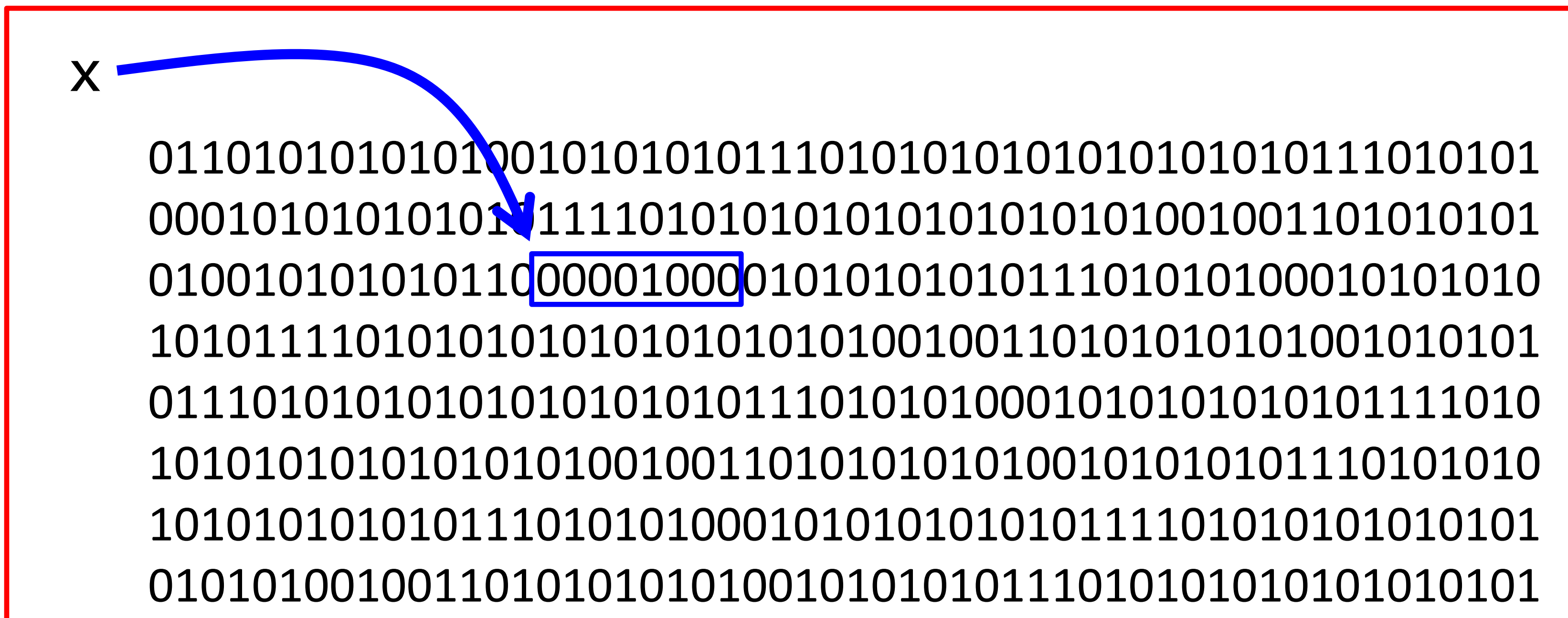




Creating a primitive variable

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

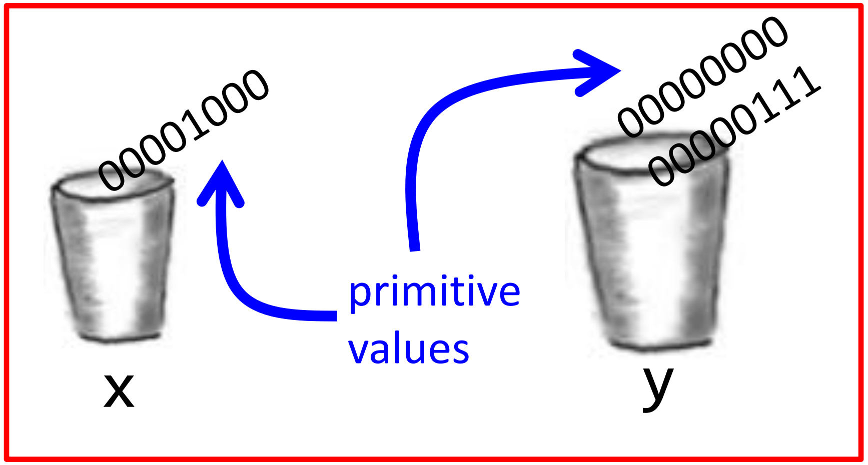


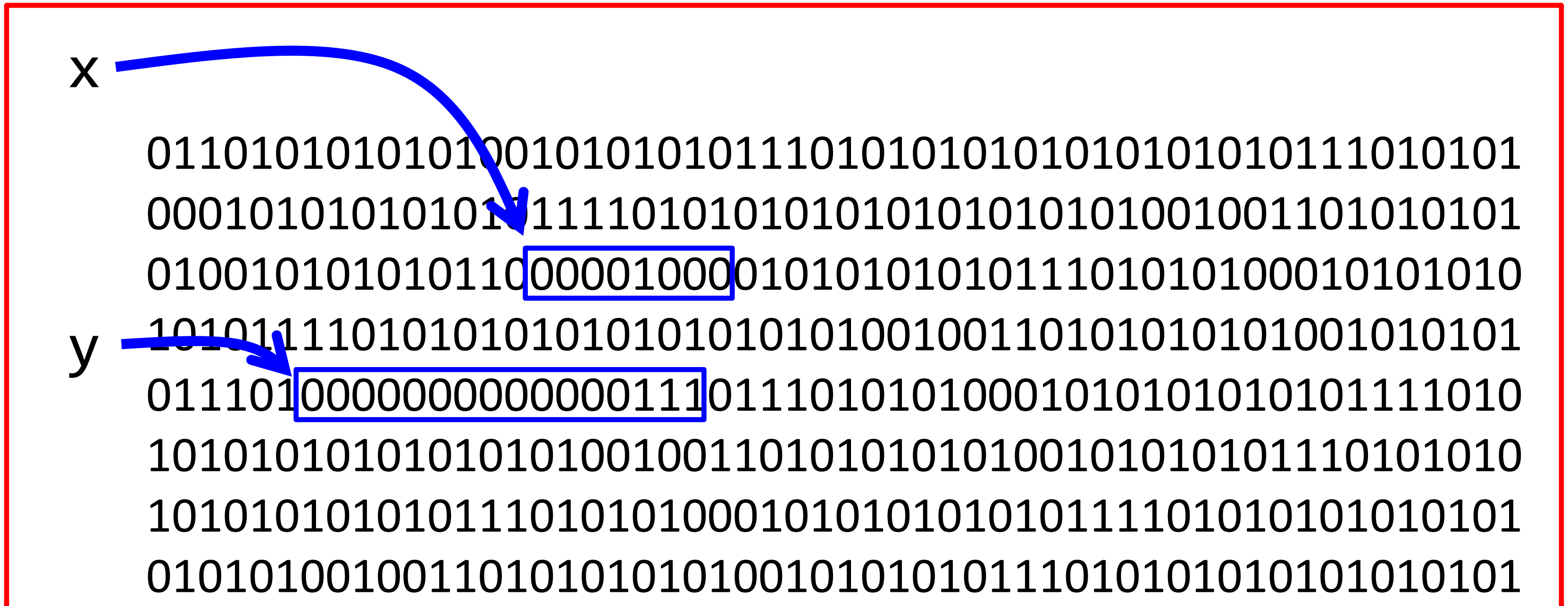


Creating a primitive variable

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

short y = 7;
```





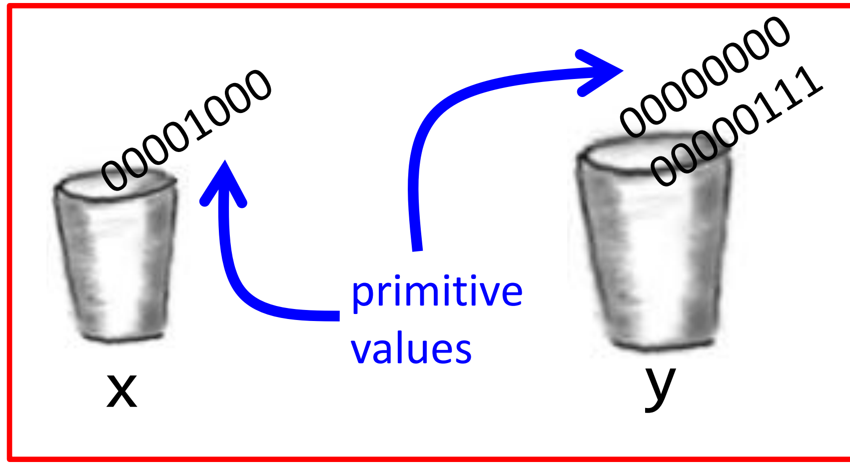
You can't put a big cup into a small one

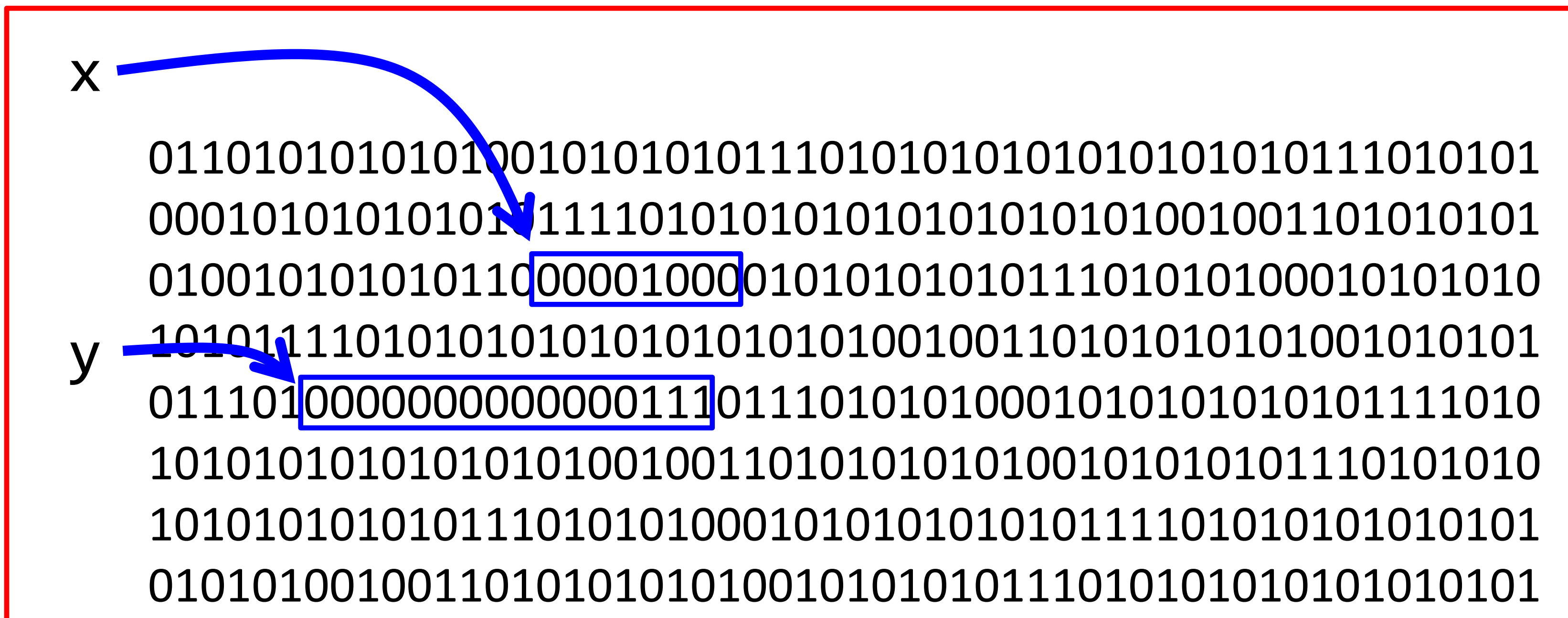
You may know 7 can fit in a byte, but compiler doesn't!

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

short y = 7;

x = y;
```



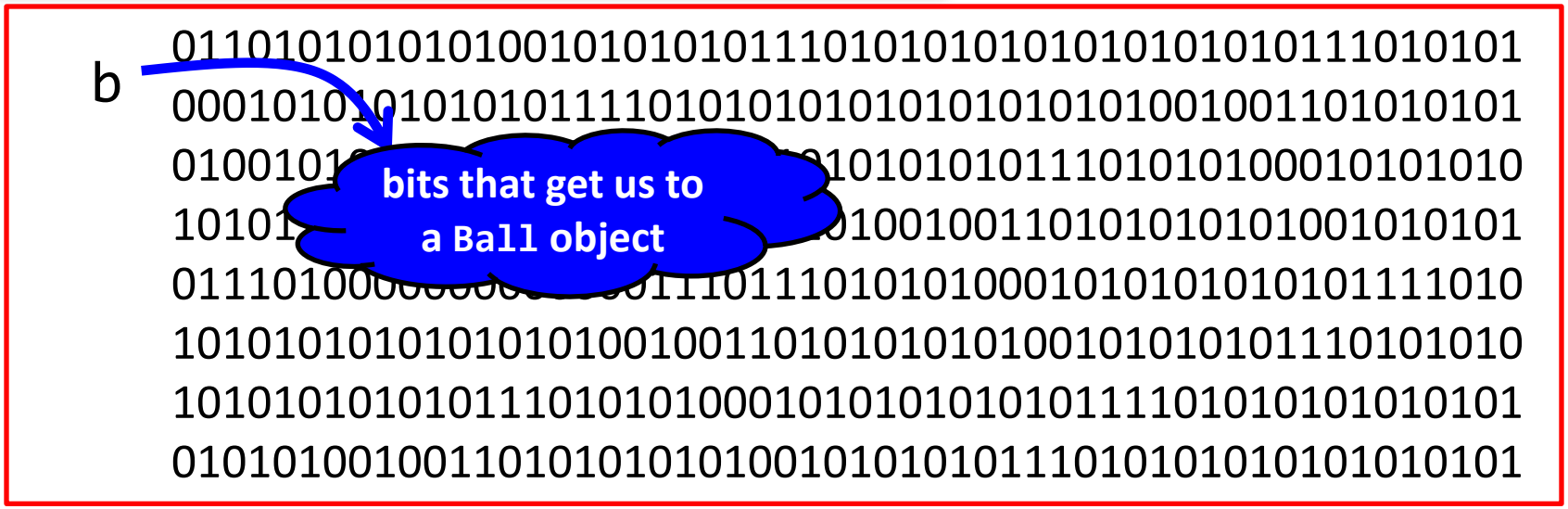


Declaring a reference variable

```
Ball b;
```

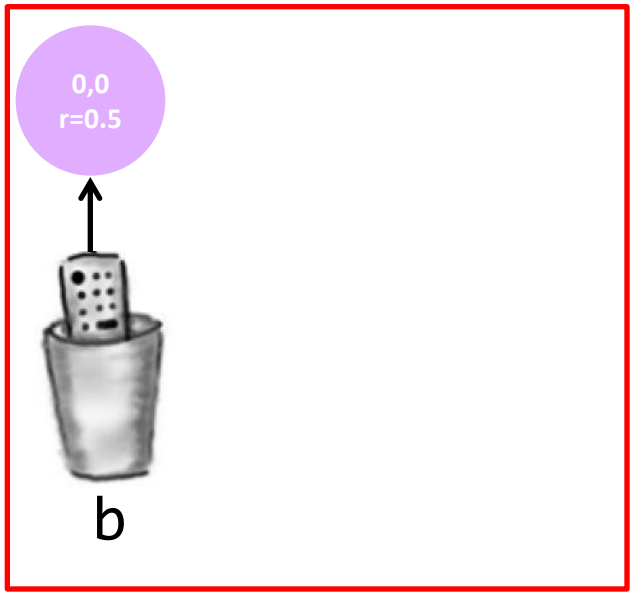
Currently b is equal to `null`.
References variables always need to be use `new` to create an actual object.

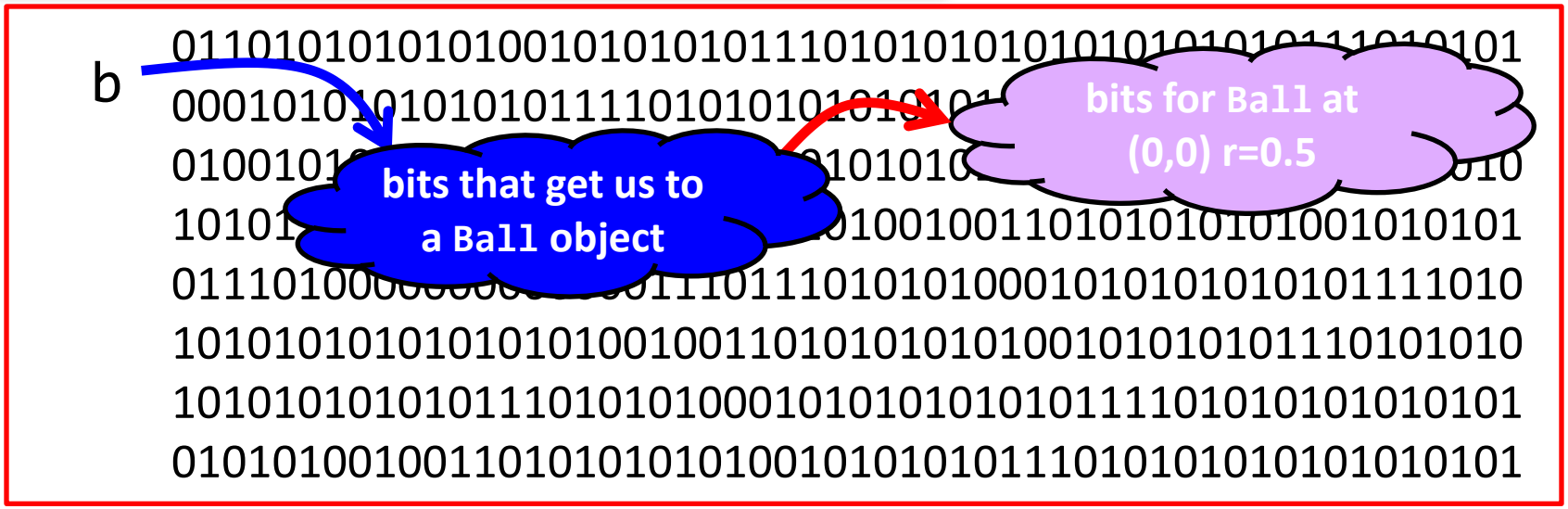
b



Creating a reference variable

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

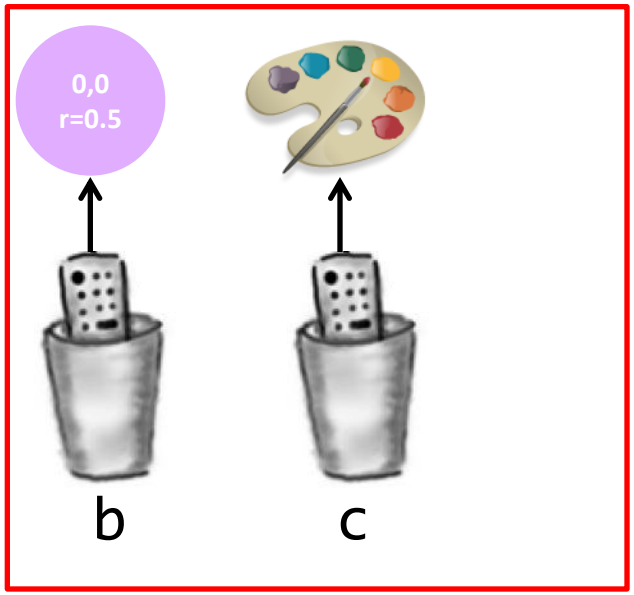


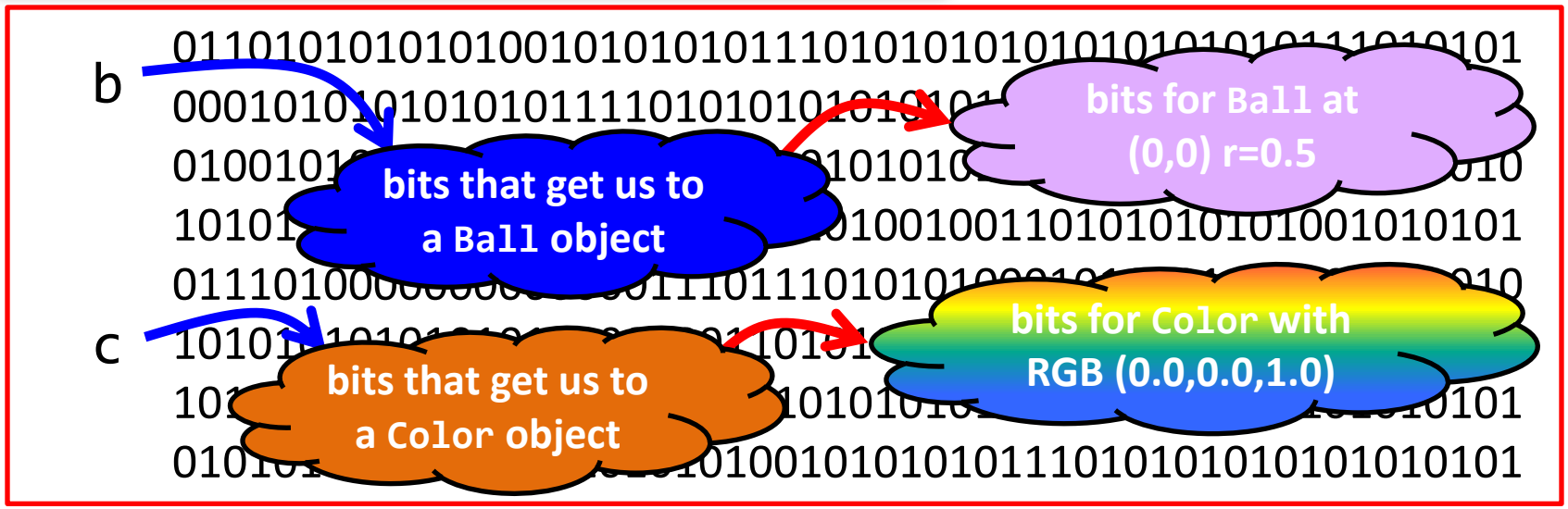


Creating a reference variable

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

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





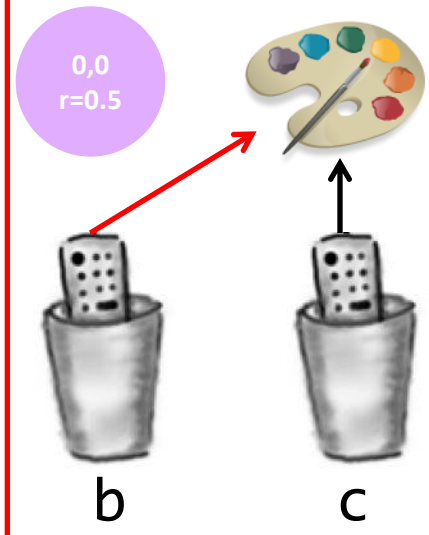
References variables can't switch types

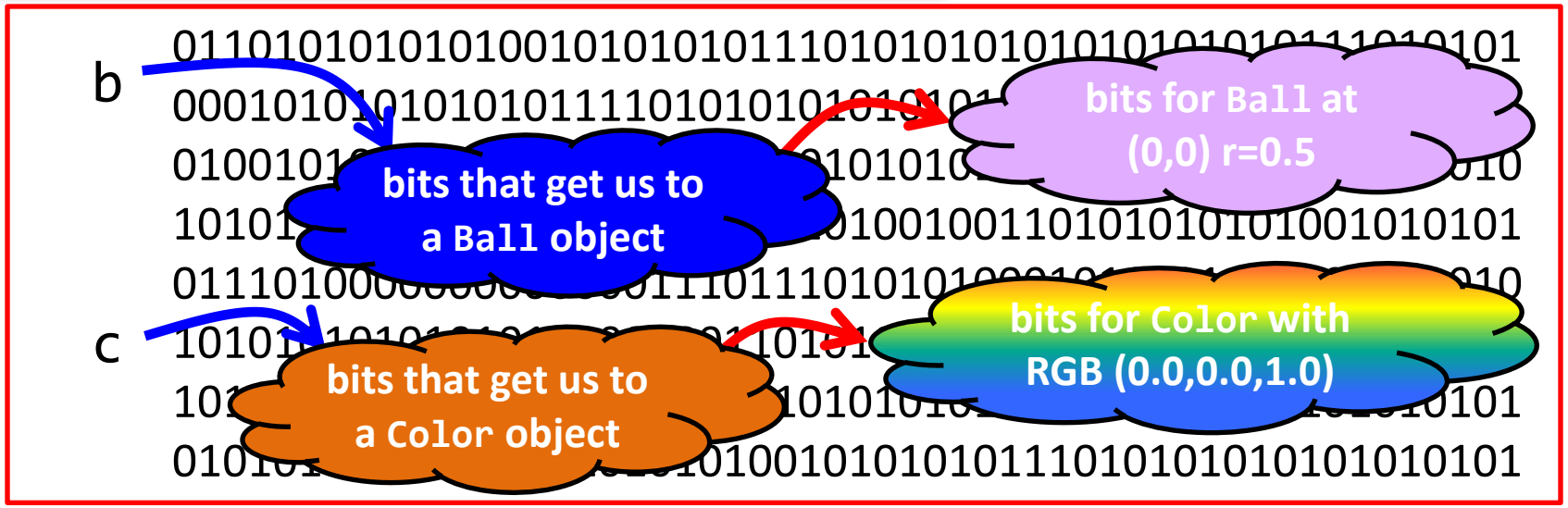
```
Ball b = new Ball(0.0, 0.0, 0.5);

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

b = c;
```

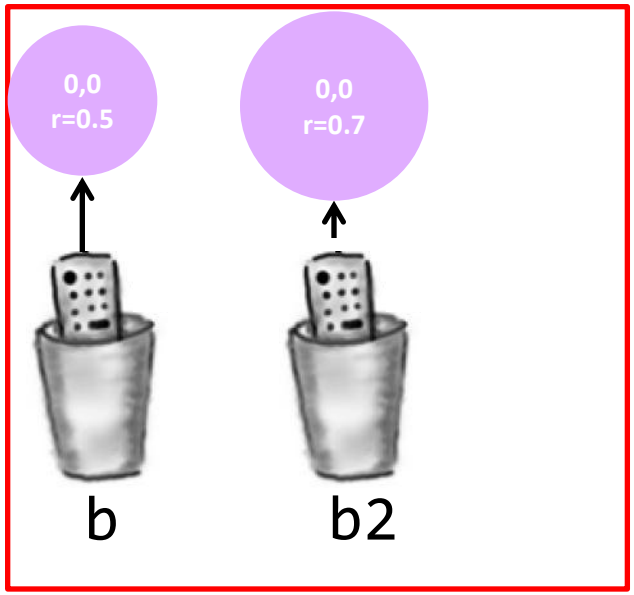
You can't put a Color object into a Ball reference variable!

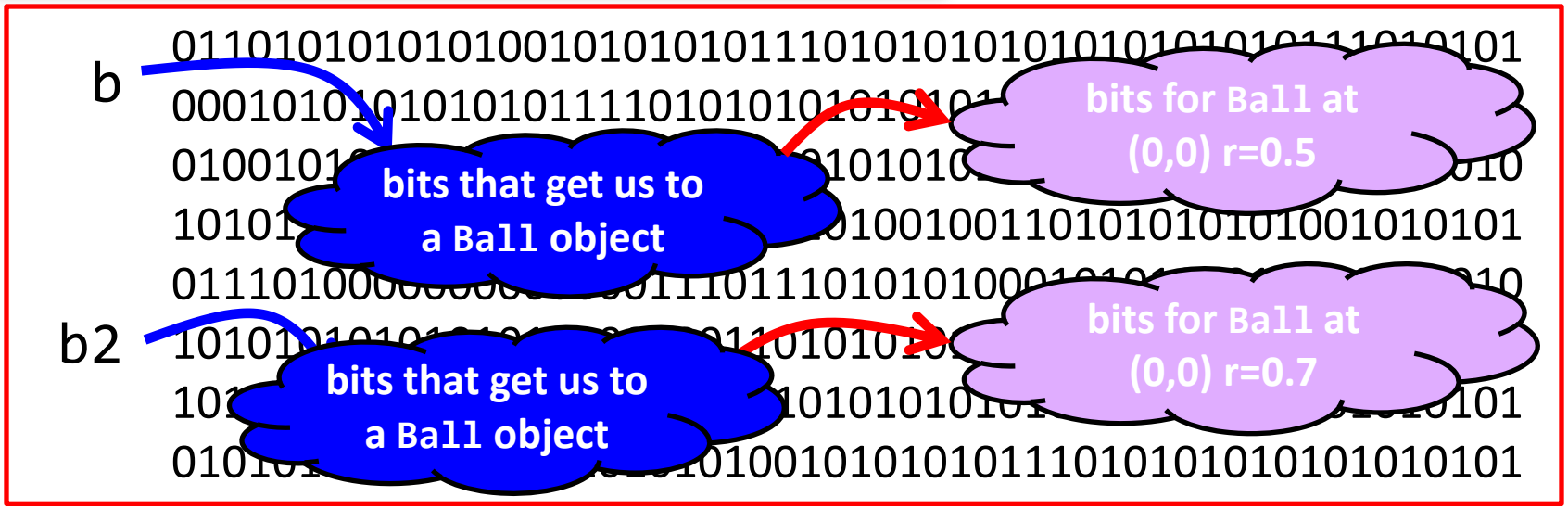




Two references of same type

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

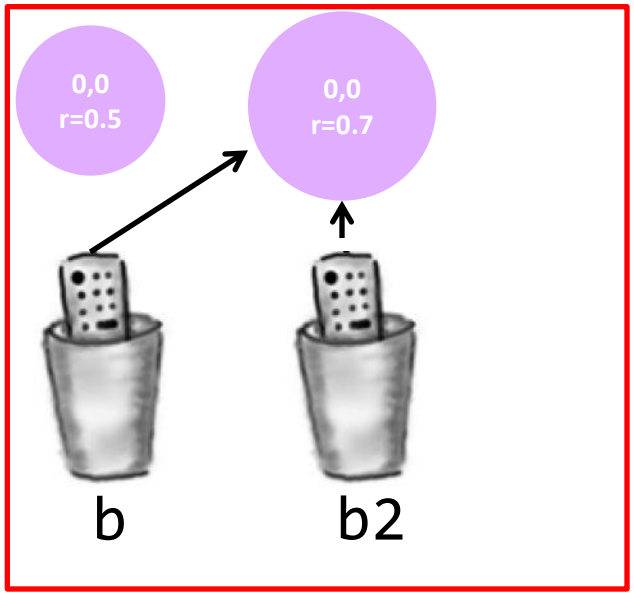


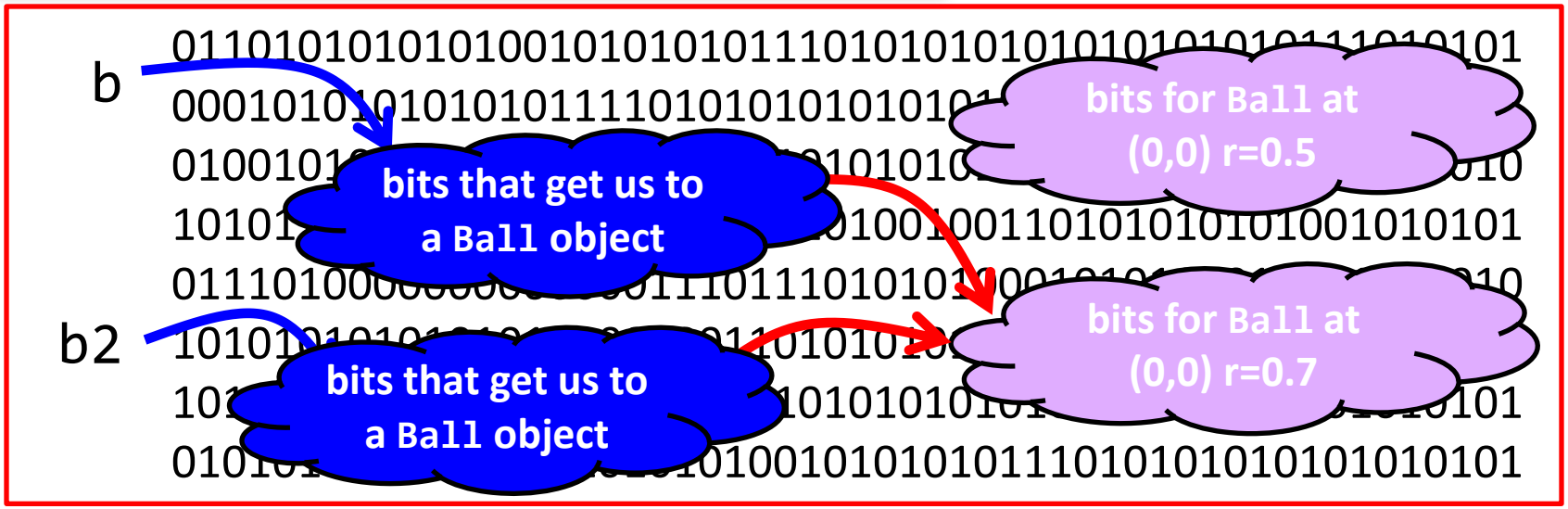


Two references of same type

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

b = b2;
```

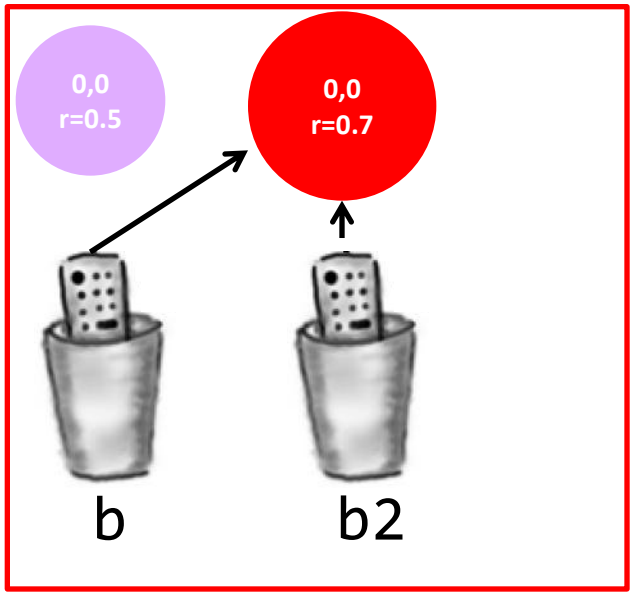


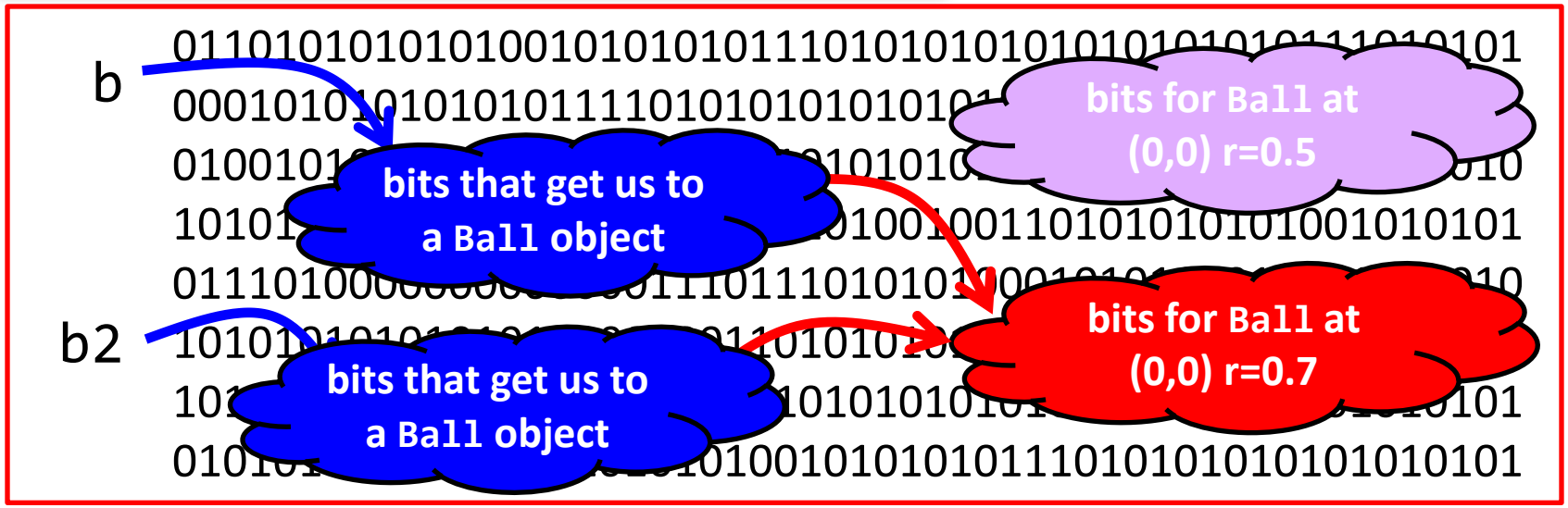


Two names: one object

```
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);
```



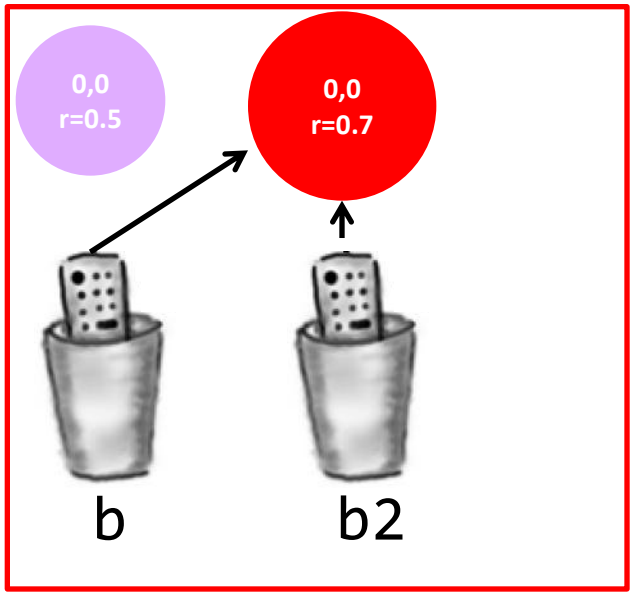


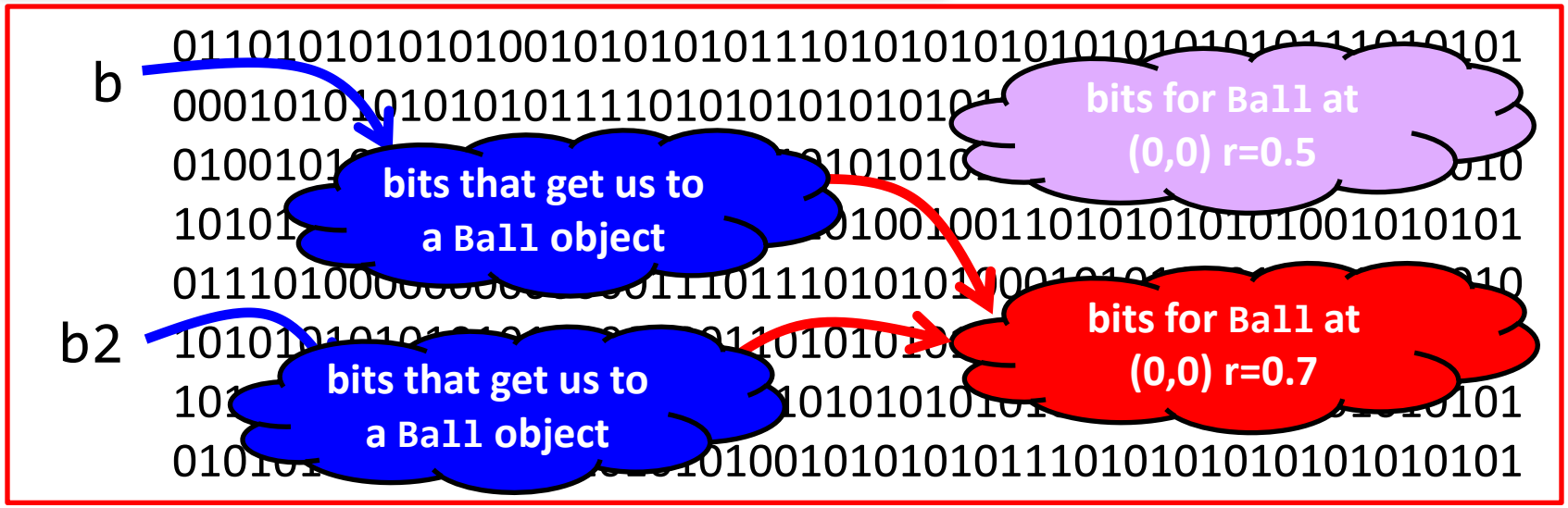
Two names: one object

```
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.



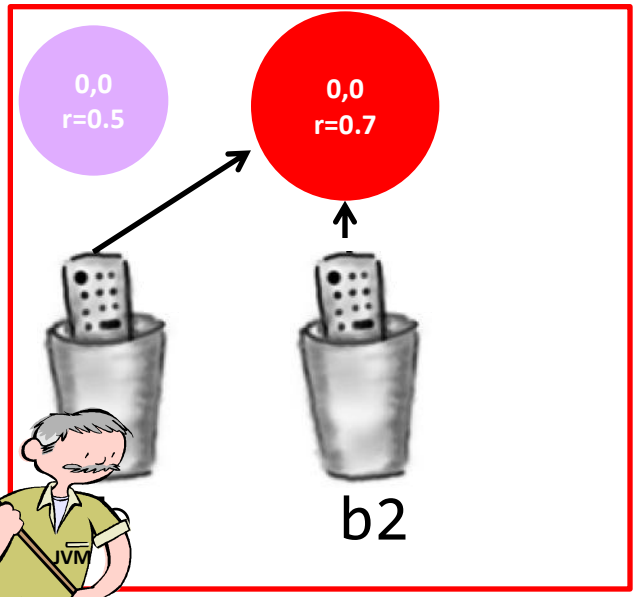


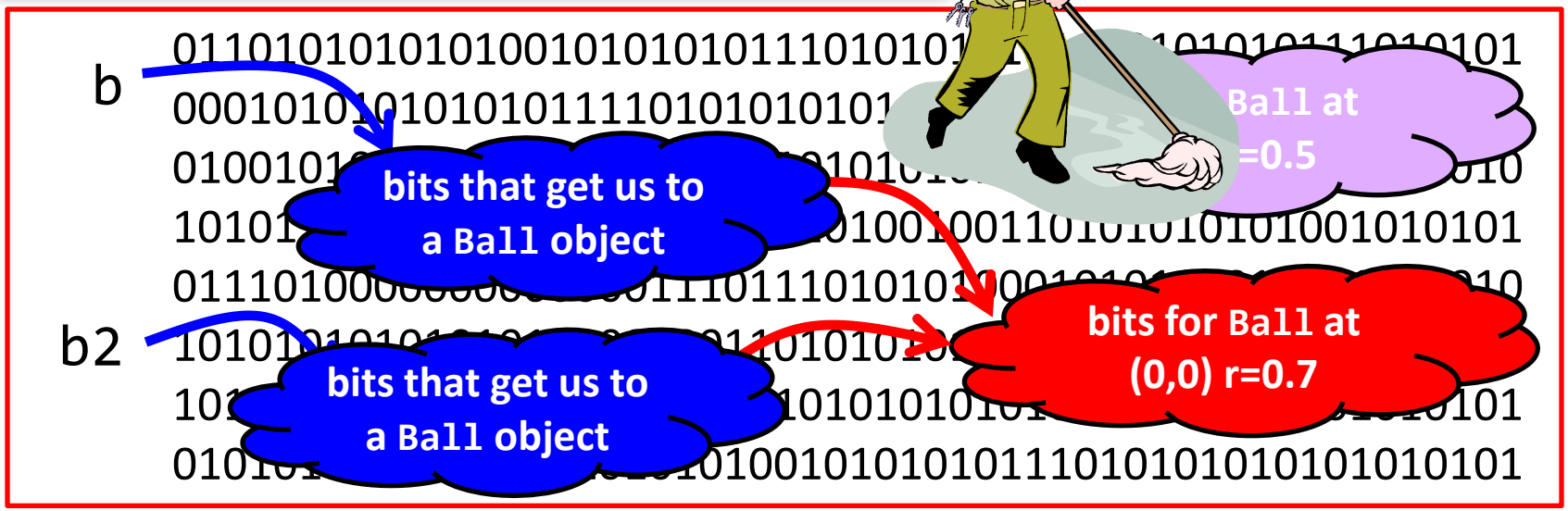
Orphaned objects

```
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.





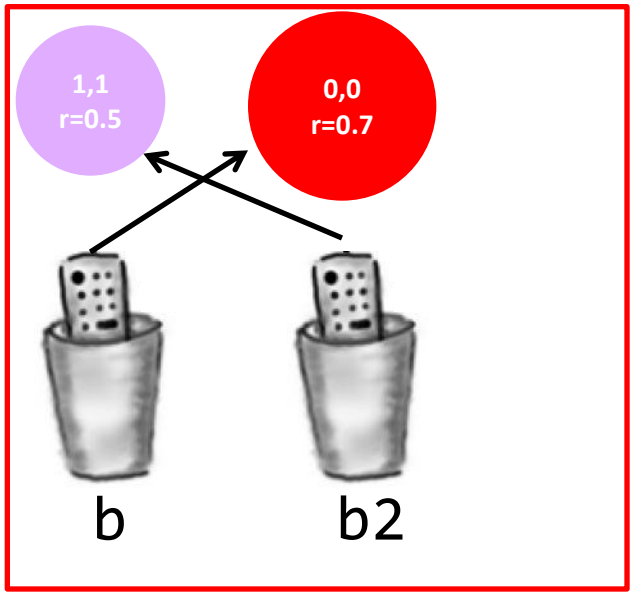
Reference variables can be reprogrammed

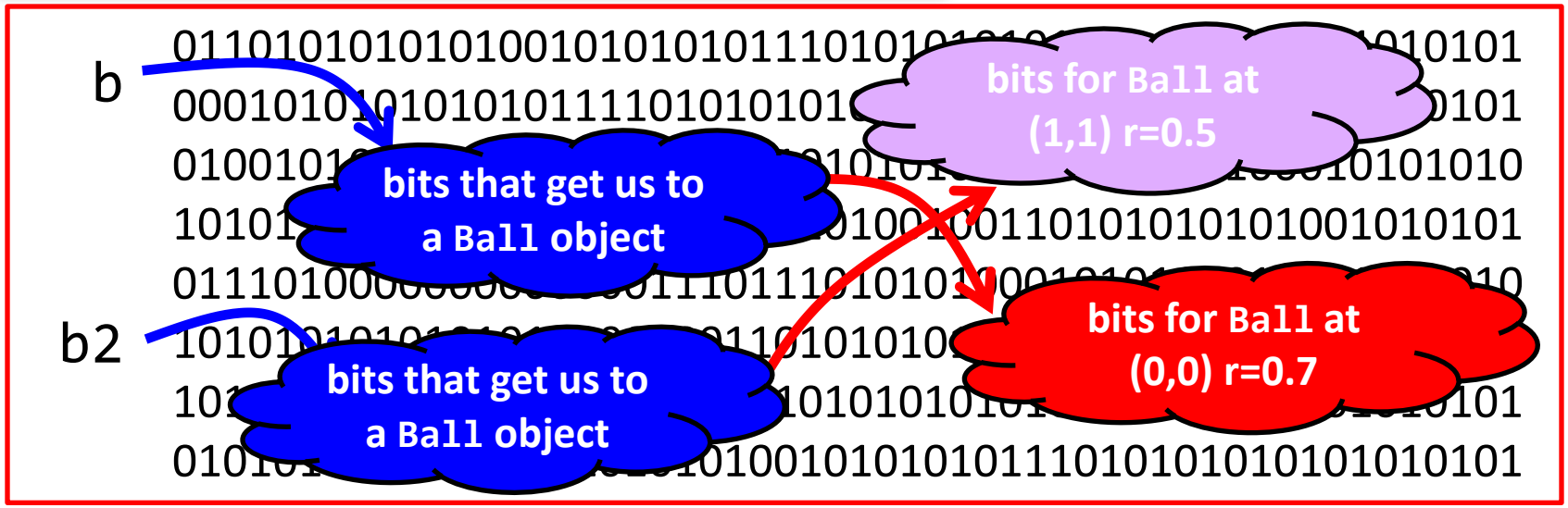
```
Ball b = new Ball(0.0, 0.0, 0.5);
Ball b2 = new Ball(0.0, 0.0, 0.7);

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!

```
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

```
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.

Alias bug'o'rama

- Parameter to method
  - If same name as instance variable → Java uses the parameter variable

```
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

```
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

```
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();
   }
   ...
}
```

Quiz

- Classes and objects

Class =

Object =

Instance variables =

Instance methods =

Constructor =

Summary

- 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()`