Objects Recap

and

Pointers!

AKA: Why is .equals different than ==?

Find a partner for the day!

Syntax

```
public class Robot {

}
```

Instance variables and *methods* go inside the curly braces (so that they "belong to" the class)

And Sales Tax, and Income Tax

Syntax

```
public class Robot {

}
```

Instance variables and *methods* go inside the curly braces (so that they "belong to" the class)

By yourself:

- Add the code that defines three instance variables: one int, one String, and one of whatever type you'd like. Name them whatever you want.

With a partner:

- Compare your results. Do you agree on the syntax?

Syntax

```
public class Robot {
   private int numberOfWheels;
   private String name;
   private double speed;
}
```

private TYPE NAME;

(In general. Note that there's no value yet!)

Private things can only be used inside the class:
that is, between the curly braces

To let code outside Robot use these data, we need *getter methods*.

And robotax. Sort of.

Syntax

```
public class Robot {
   private int numberOfWheels;
   private String name;
   private double speed;
}
```

private TYPE NAME;

(In general. Note that there's no value yet!)

Private things can only be used inside the class:
that is, between the curly braces

To let code outside Robot use these data, we need *getter methods*.

By yourself: add a getter for one of your instance variables. Then compare with your partner.

And robotax. Sort of.

Syntax

```
public class Robot {
   private int numberOfWheels;
   private String name;
   private double speed;

   public double getSpeed() {
      return speed;
   }

   public String getName() {
      return name;
   }
}
```

By convention, getters are named getWhatever().

Note "public"

public return type name(parameters)

Because getSpeed() is *inside* Robot, it can use the (private) speed instance variable

Maybe "robotix"?

Syntax

```
public class Robot {
   private int numberOfWheels;
   private String name;
   private double speed;

   public double getSpeed() {
      return speed;
   }

   public String getName() {
      return name;
   }
}
```

By convention, getters are named getWhatever().

Note "public"

public return type name(parameters)

Because getSpeed() is *inside* Robot, it can use the (private) speed instance variable

By yourself: add a setter for one of your instance variables. Then compare with your partner.

Syntax

```
public class Robot {
   private int numberOfWheels;
   private String name;
   private double speed;

   public double getSpeed() {
      return speed;
   }
```

void means "doesn't return anything"

public return type name(argtype1 argname1, argtype2 argname2, ...)

```
   public void setSpeed(double newSpeed) {
      speed = newSpeed;
   }
}
```

Setters change the *internal state* of an object.

Syntax

```
public class Robot {
   private int numberOfWheels;
   private String name;
   private double speed;

   public double getSpeed() {
      return speed;
   }
```

void means "doesn't return anything"

public return type name(argtype1 argname1, argtype2 argname2, ...)

```
   public void setSpeed(double newSpeed) {
      speed = newSpeed;
   }
}
```

Setters change the *internal state* of an object.

By yourself: write a constructor for your class. Then compare with your partner.

Syntax

```
public class Robot {
   private int numberOfWheels;
   private String name;
   private double speed;

   public Robot(int w, String n, double s) {
      numberOfWheels = w;
      name = n;
      speed = s;
   }
}
```

public ClassName(parameters)

Constructors fill in instance variables.

```
public class Robot {
   private int numberOfWheels;
   private String name;
   private double speed;

   public Robot(int w,
                String n,
                double s) {
      numberOfWheels = w;
      name = n;
      speed = s;
   }

   double getSpeed() {
      return speed;
   }

   void setSpeed(double s) {
      speed = s;
   }
}
```

```
pr2.setSpeed(10.0);
```

```
System.out.println(
        "pr2 goes " +
        pr2.getSpeed());
```

```
Robot pr2 = new Robot(8,
                      "PR2",
                      0.5);
```

In the usual way: match lefts to rights.

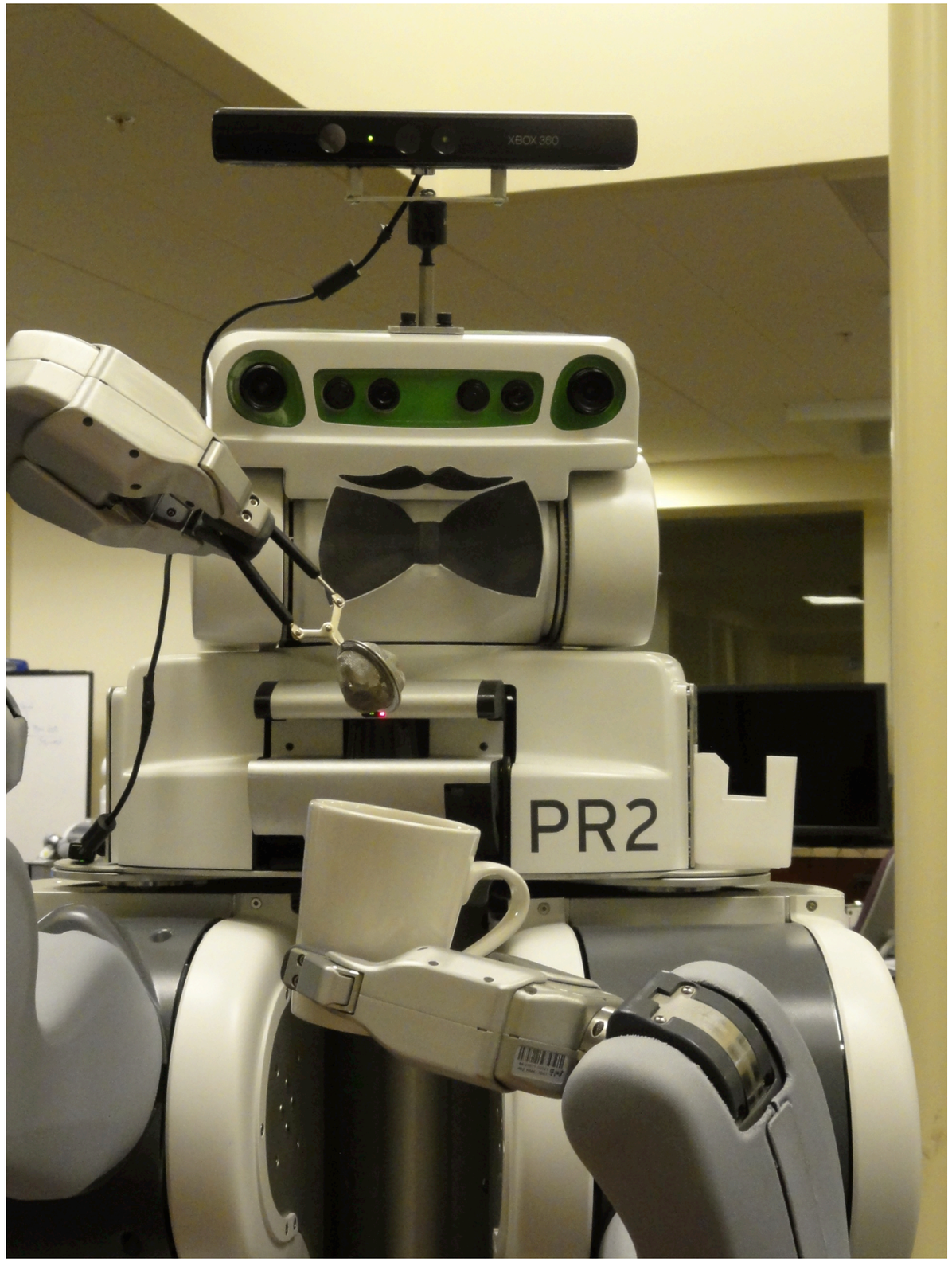
And then: put the rights into an order that will run. And figure out what it prints.

```
public class Robot {
   private int numberOfWheels;
   private String name;
   private double speed;

   public Robot(int w,
                String n,
                double s) {
      numberOfWheels = w;
      name = n;
      speed = s;
   }

   double getSpeed() {
      return speed;
   }

   void setSpeed(double s) {
      speed = s;
   }
}
```

```
pr2.setSpeed(10.0);

System.out.println(
        "pr2 goes " +
        pr2.getSpeed());

Robot pr2 = new Robot(8,
                      "PR2",
                      0.5);
```

In the usual way: match lefts to rights.

And then: put the rights into an order that will run. And figure out what it prints.

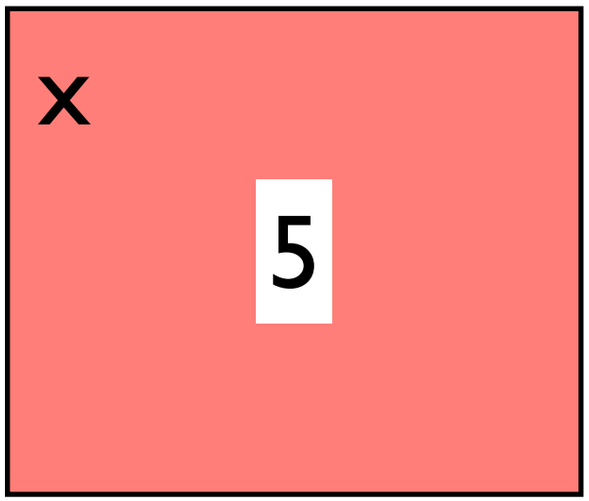

XBOX 360
PR2

Pointers!

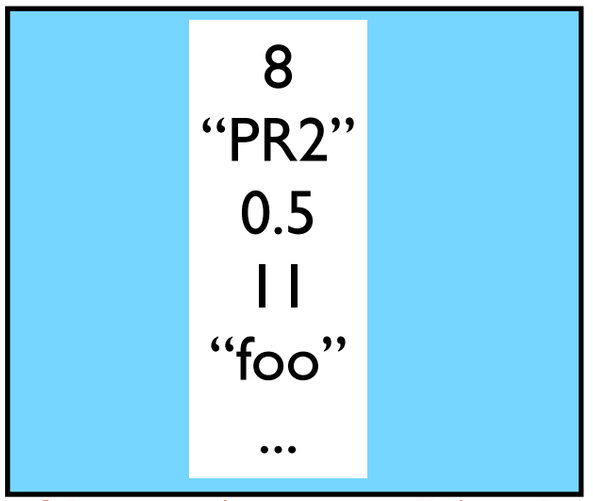
```
int x = 5;
Robot pr2 = new Robot(8, "PR2", 0.5);
```

Pointers!

```
int x = 5;
Robot pr2 = new Robot(8, "PR2", 0.5);
```



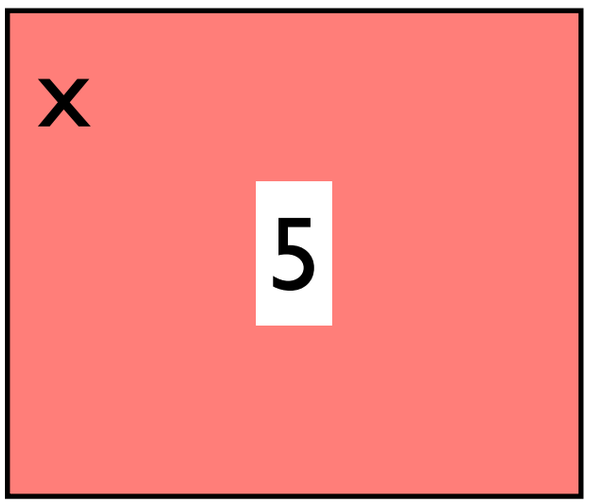
Primitives take up very little memory (each)



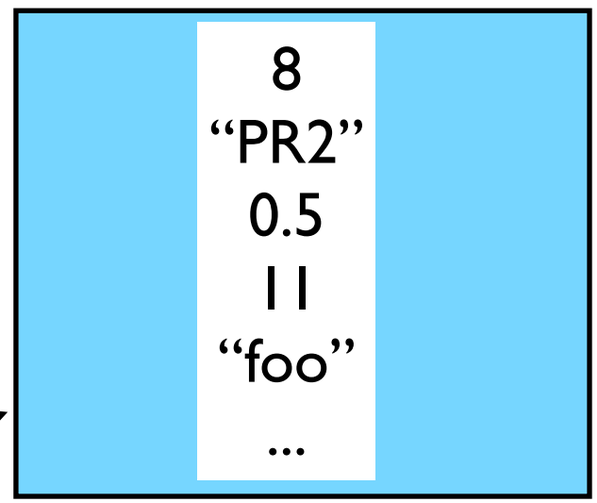
Objects (potentially) take up lots of memory (each). (So do arrays!)

Pointers!

```
int x = 5;
Robot pr2 = new Robot(8, "PR2", 0.5);
```



Primitives take up very little memory (each)



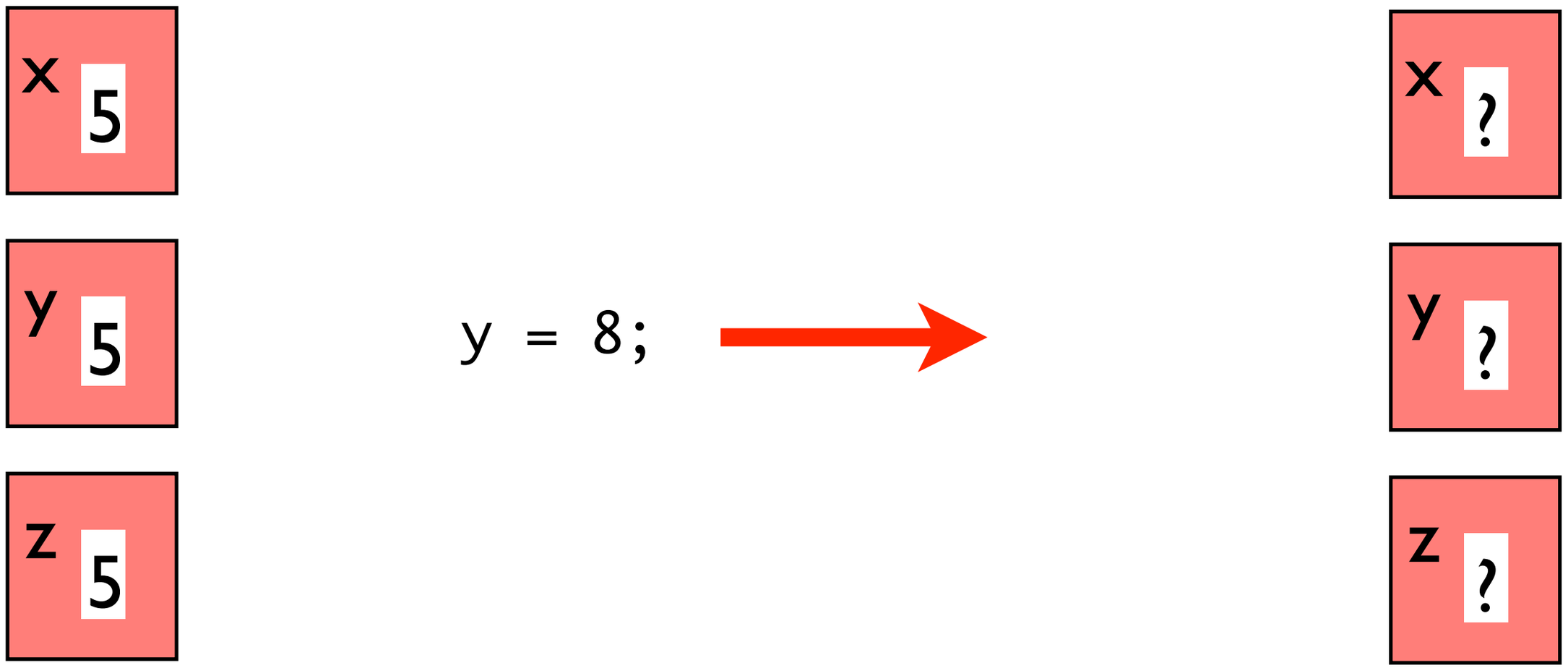
Objects (potentially) take up lots of memory (each).

A *pointer. (also memory-cheap)*

Every object variable is storing a pointer.

Pointers!

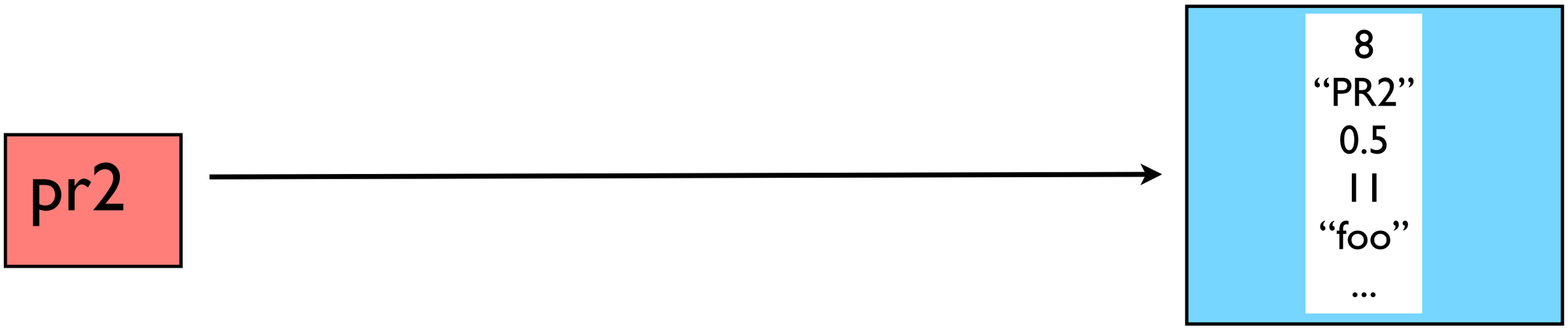
```
int x = 5;
int y = x;
int z = x;
```



In the usual way: fill the boxes in. Yes, it's really easy.

Pointers!

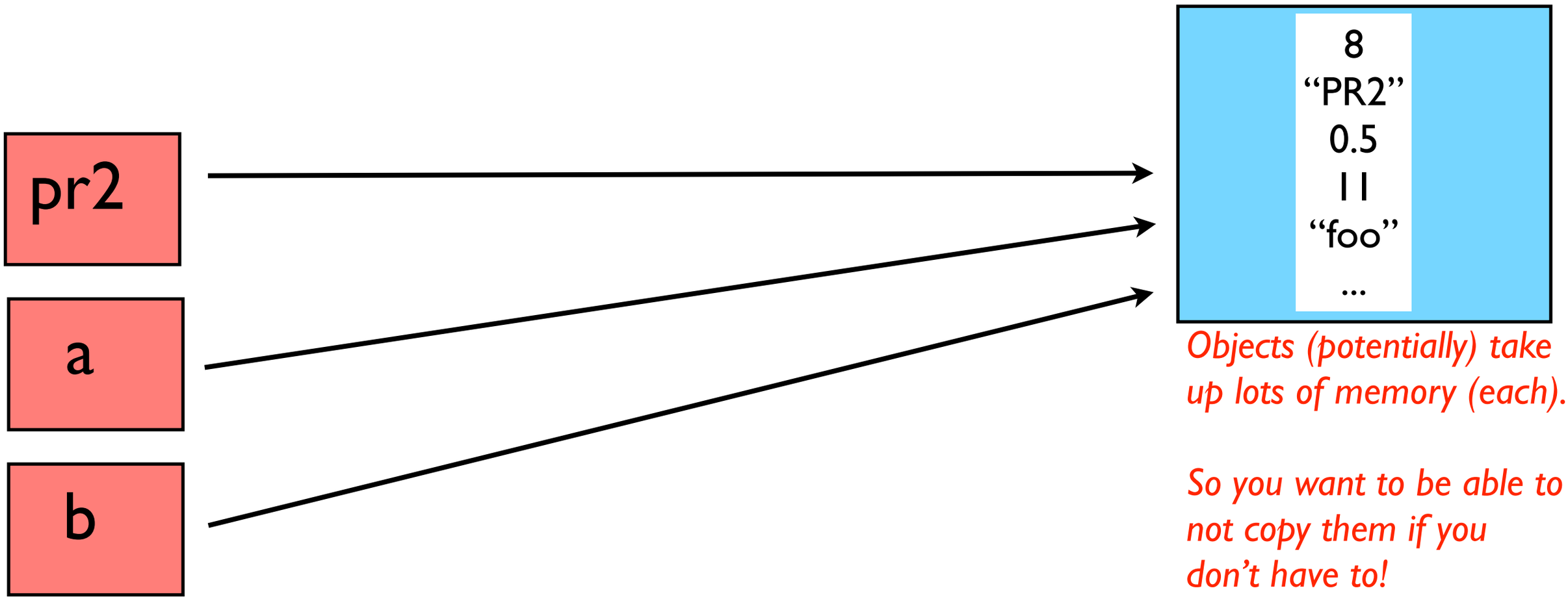
```
Robot pr2 = new Robot(8, "PR2", 0.5);
Robot a = pr2;
Robot b = pr2;
```



In the usual way: finish the picture.

Pointers!

```
Robot pr2 = new Robot(8, "PR2", 0.5);
Robot a = pr2;
Robot b = pr2;
```



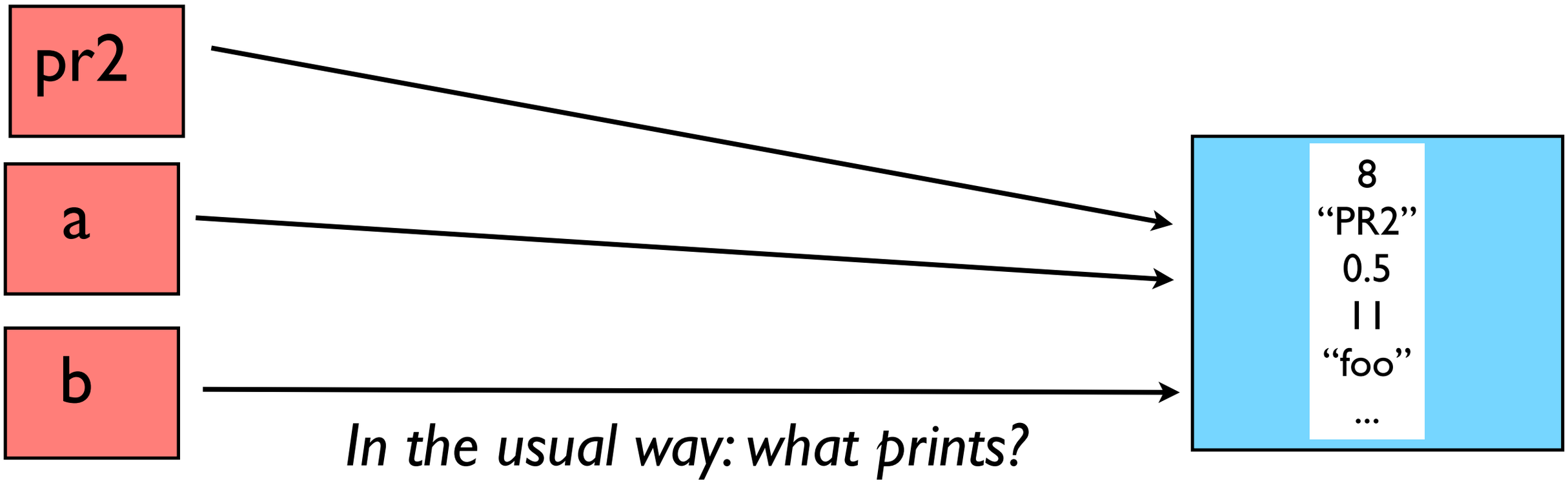
Objects (potentially) take up lots of memory (each).

So you want to be able to not copy them if you don’t have to!

In the usual way: finish the picture.

Pointers!

```
System.out.println("PR2 speed: " + pr2.getSpeed());
a.setSpeed(10);
System.out.println("PR2 speed: " + pr2.getSpeed());
```



In the usual way: what prints?

.equals

Snarf `Strings`

Before you run it, predict what will print.
Compare that with your partner.

Then run it, and see if you were right.

.equals

Snarf `Strings`

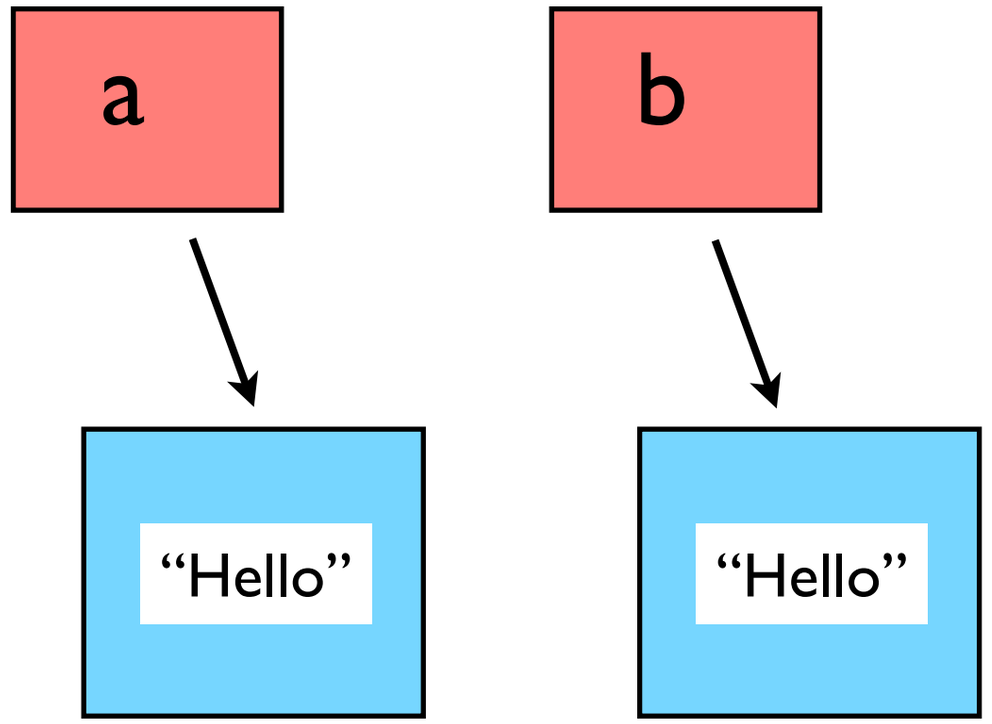
Before you run it, predict what will print.
Compare that with your partner.

Then run it, and see if you were right.

I got:

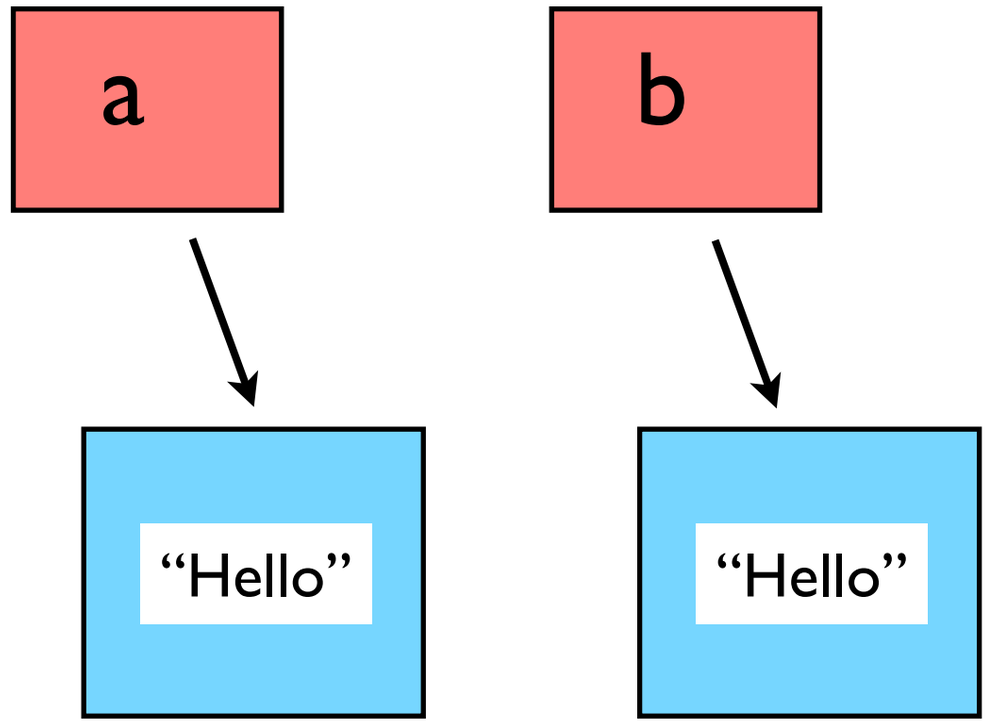
```
!=
.equals
c == b
c.equals(b)
```

```
String a = new String("Hello");
String b = new String("Hello");

if (a == b) {
  System.out.println("==");
} else {
   System.out.println("!=");
}

if (a.equals(b)) {
  System.out.println(".equals");
} else {
   System.out.println("not .equals");
}

String c = b;

if (c == b) {
  System.out.println("c == b");
} else {
   System.out.println("c != b");
}

if (c.equals(b)) {
  System.out.println("c.equals(b)");
```



== compares *pointers*.

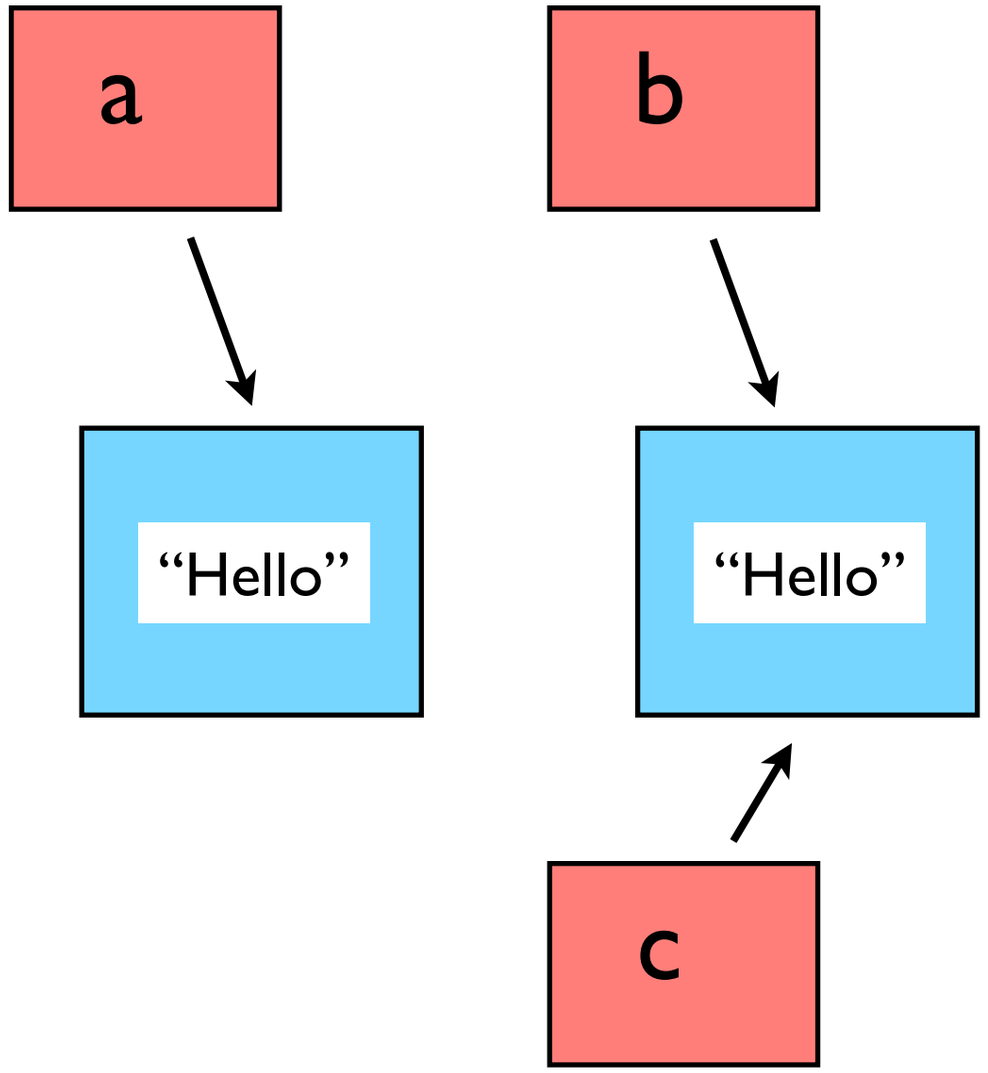
```
String a = new String("Hello");
String b = new String("Hello");

if (a == b) {
  System.out.println("==");
} else {
   System.out.println("!=");
}

if (a.equals(b)) {
  System.out.println(".equals");
} else {
   System.out.println("not .equals");
}

String c = b;

if (c == b) {
  System.out.println("c == b");
} else {
   System.out.println("c != b");
}

if (c.equals(b)) {
  System.out.println("c.equals(b)");
```



== compares *pointers*.

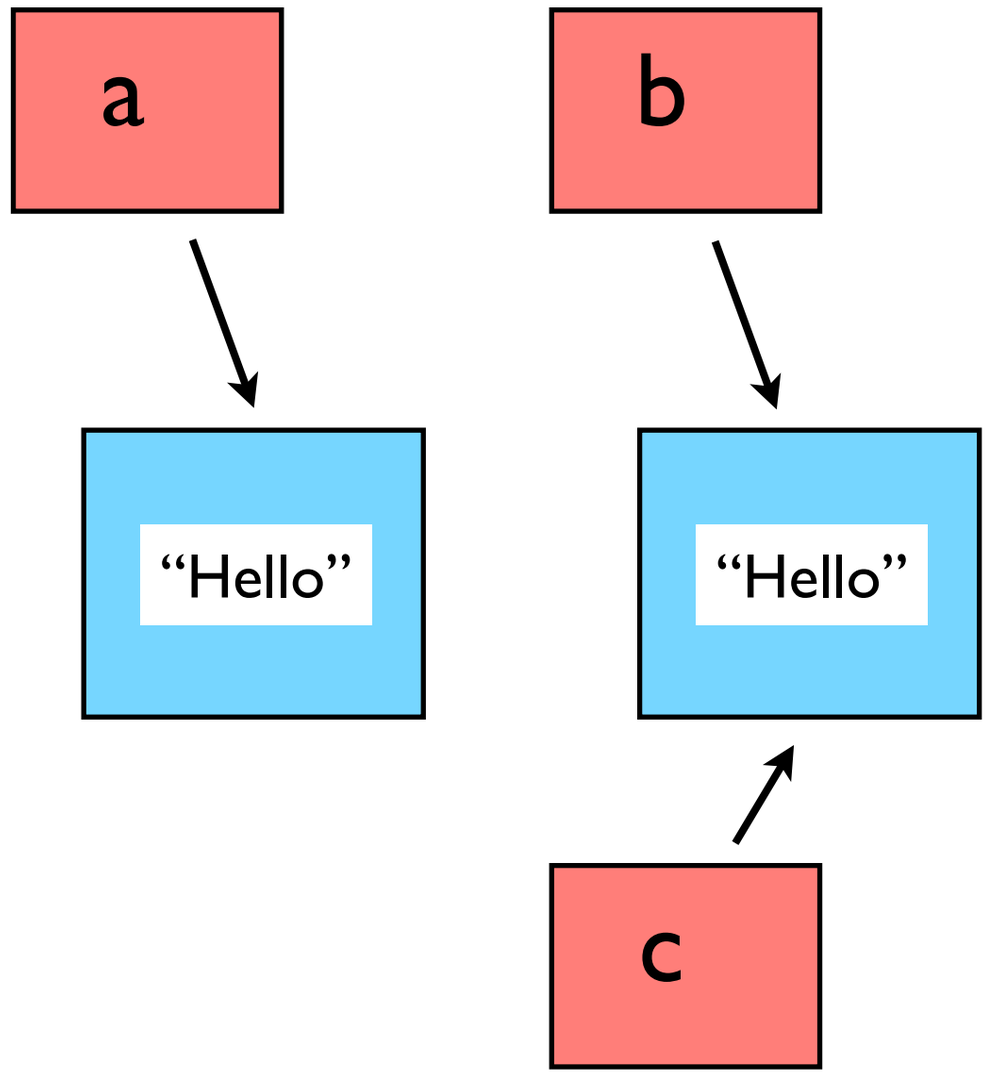
.equals compares *values*

```
String c = b;

if (c == b) {
   System.out.println("c == b");
} else {
    System.out.println("c != b");
}

if (c.equals(b)) {
   System.out.println("c.equals(b)");
} else {
    System.out.println("not c.equals(b)");
}
```



= copies *pointers*.

```
String c = b;

if (c == b) {
  System.out.println("c == b");
} else {
   System.out.println("c != b");
}

if (c.equals(b)) {
  System.out.println("c.equals(b)");
} else {
   System.out.println("not c.equals(b)");
}
```



= copies *pointers*.

.clone copies *values*.

Jotto!

Hangman due at midnight.

New assignment coming out today: Jotto!

(Due Monday)

Three new APTs coming out today!

(Due Friday)

Demo time!

http://goo.gl/9tx4P