

Reference vs Object vs Instance vs Class

In this video, I'm going to discuss, and hopefully clear up any confusion about references, vs. objects, vs. instances, vs. classes.

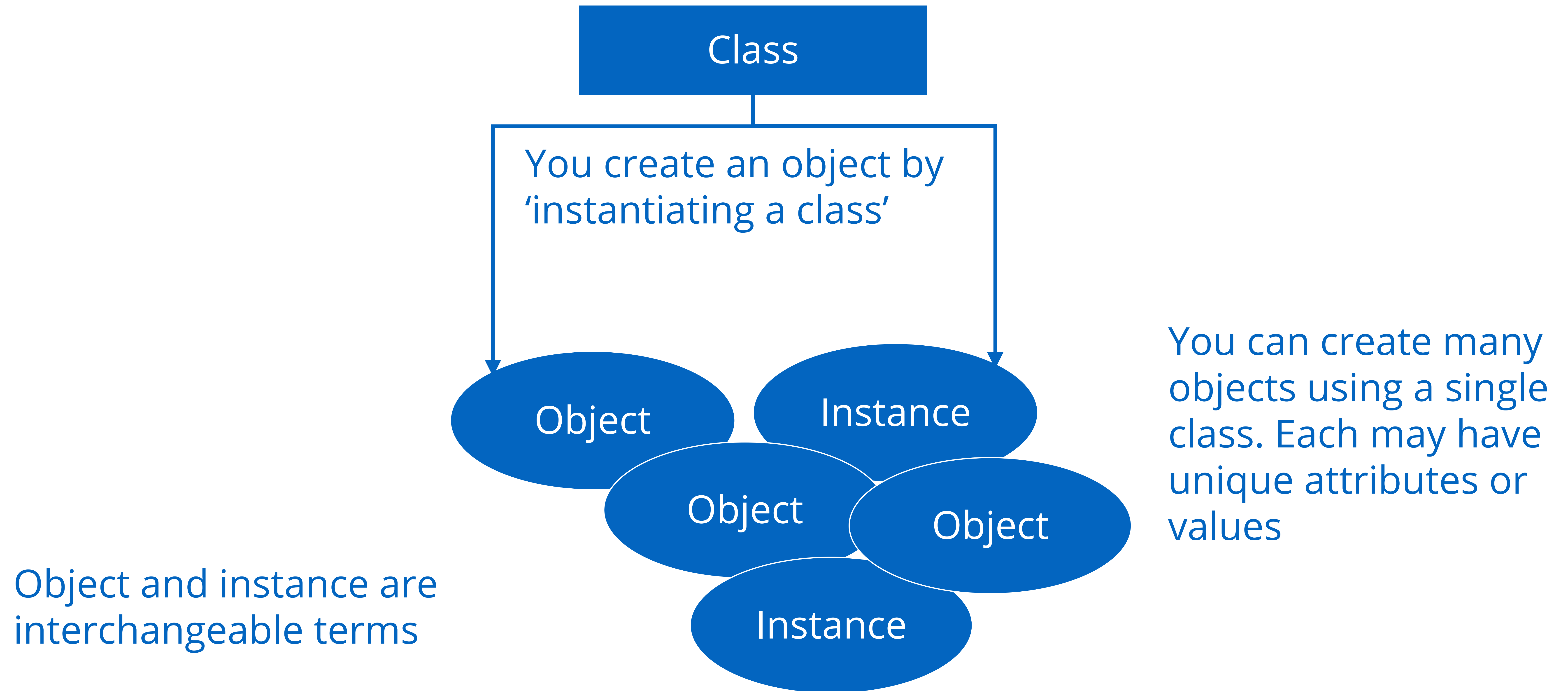
Reference vs Object vs Instance vs Class

By now, you've probably noticed that I use the words **reference**, **object**, **instance** and **class** frequently.

These new concepts may well be confusing at first.

In this video, I'm going to go through all these terms, and show you exactly what each of these words mean in the context of Java programming.

Object vs Instance vs Class



Reference vs Object vs Instance vs Class

Let's use the analogy of building a house to understand classes.

A class is basically a blueprint for the house.

Using the blueprint, we can build as many houses as we like based on those plans.

Each house we build (in other words, going back to programming terms, each house we instantiate using the new operator) is an object.

This object can also be known as an instance. Often, we'll say it's an instance of the class. So, we would have an instance of house in this example.

Getting back to the physical world, each house we build has an address (it's built at a physical location).

In other words, if we want to tell someone where we live, we give them our address (perhaps written on a piece of paper). This is known as a reference.

Reference vs Object vs Instance vs Class

We can copy that reference as many times as we like, but there is still just one house that we're referring to.

In other words, we're copying the paper that has the address on it, not the house itself.

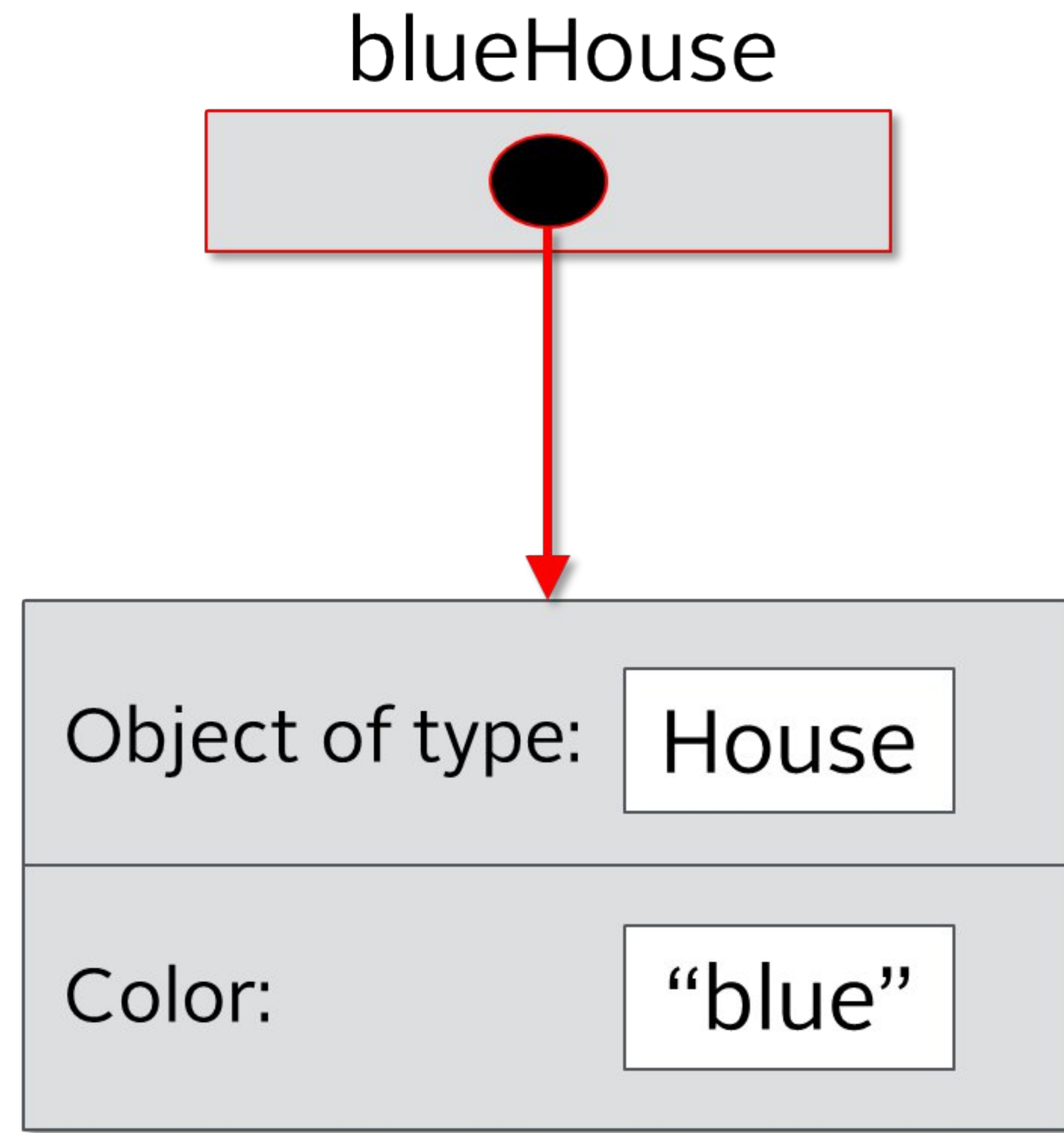
We can pass references as parameters to constructors and methods.

Reference vs Object vs Instance vs Class

```
public class House {  
  
    private String color;  
  
    public House(String color) {  
        this.color = color;  
    }  
  
    public String getColor() {  
        return color;  
    }  
  
    public void setColor(String color) {  
        this.color = color;  
    }  
}
```

```
public class Main {  
  
    public static void main(String[] args) {  
  
        House blueHouse = new House("blue");  
        House anotherHouse = blueHouse;  
  
        System.out.println(blueHouse.getColor()); // prints blue  
        System.out.println(anotherHouse.getColor()); // blue  
  
        anotherHouse.setColor("red");  
        System.out.println(blueHouse.getColor()); // red  
        System.out.println(anotherHouse.getColor()); // red  
  
        House greenHouse = new House("green");  
        anotherHouse = greenHouse;  
  
        System.out.println(blueHouse.getColor()); //red  
        System.out.println(greenHouse.getColor()); // green  
        System.out.println(anotherHouse.getColor()); // green  
  
    }  
}
```


Reference vs Object vs Instance vs Class



```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        House blueHouse = new House("blue");
```

```
        House anotherHouse = blueHouse;
```

```
        System.out.println(blueHouse.getColor()); // prints blue
```

```
        System.out.println(anotherHouse.getColor()); // blue
```

```
        anotherHouse.setColor("red");
```

```
        System.out.println(blueHouse.getColor()); // red
```

```
        System.out.println(anotherHouse.getColor()); // red
```

```
        House greenHouse = new House("green");
```

```
        anotherHouse = greenHouse;
```

```
        System.out.println(blueHouse.getColor()); //red
```

```
        System.out.println(greenHouse.getColor()); // green
```

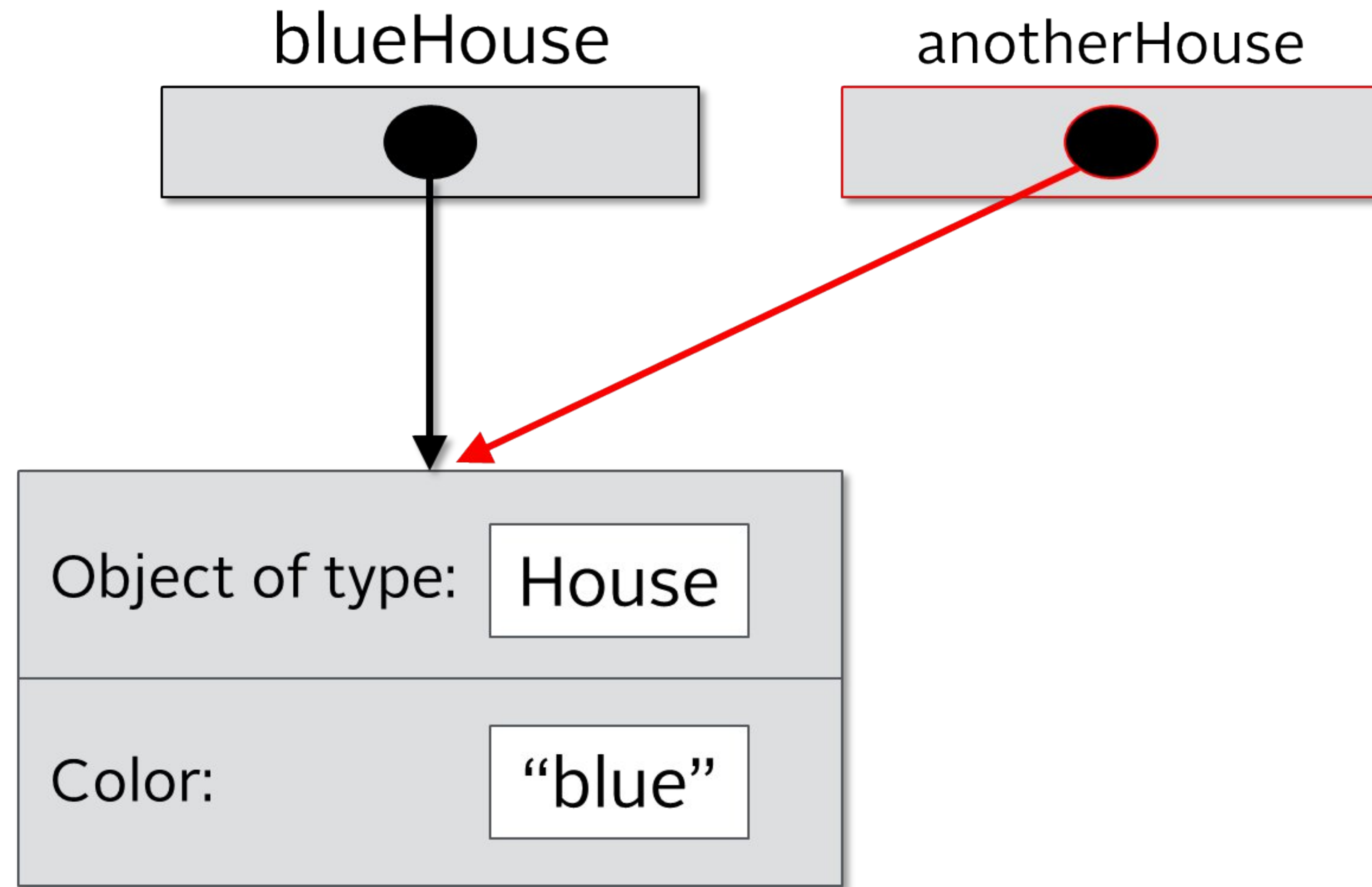
```
        System.out.println(anotherHouse.getColor()); // green
```

```
    }
```

```
}
```

The line **`House blueHouse = new House("blue");`** creates a new **instance** of the **House class**. Remember **House** is a blueprint, and we are assigning it to the **blueHouse variable**. In other words it is a **reference** to the **object** in memory. The image on the left hopefully makes sense to you now.

Reference vs Object vs Instance vs Class



```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        House blueHouse = new House("blue");  
        House anotherHouse = blueHouse;
```

```
        System.out.println(blueHouse.getColor()); // prints blue  
        System.out.println(anotherHouse.getColor()); // blue
```

```
        anotherHouse.setColor("red");  
        System.out.println(blueHouse.getColor()); // red  
        System.out.println(anotherHouse.getColor()); // red
```

```
        House greenHouse = new House("green");  
        anotherHouse = greenHouse;
```

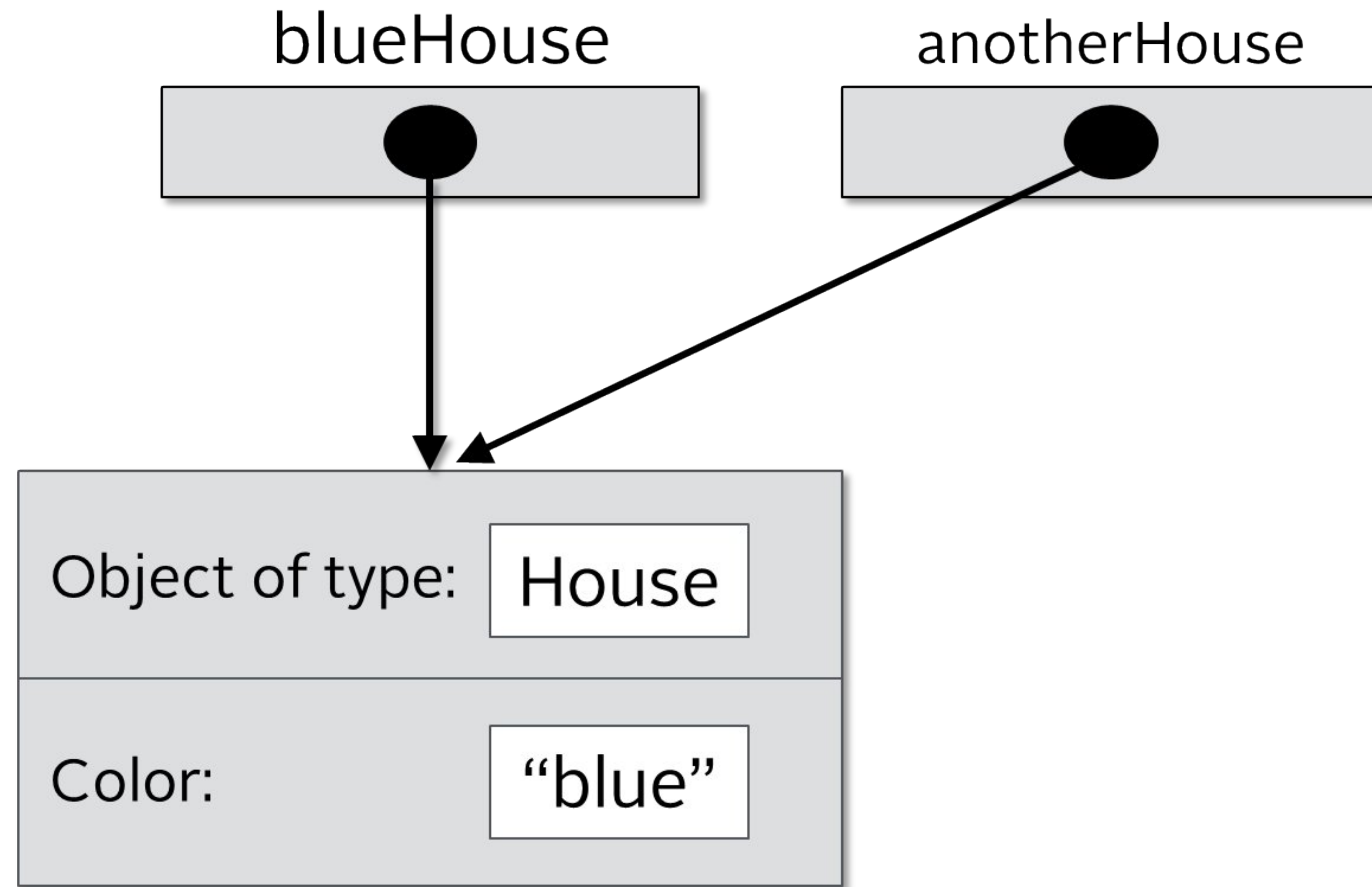
```
        System.out.println(blueHouse.getColor()); //red  
        System.out.println(greenHouse.getColor()); // green  
        System.out.println(anotherHouse.getColor()); // green
```

```
    }
```

```
}
```

The next line **House anotherHouse = blueHouse;** creates another **reference** to the same **object** in memory. Here we have two **references** pointing to the same object in memory. There is still one house, but two **references** to that one **object**. In other words we have two pieces of paper with the physical address of where the house is built (going back to our real world example).

Reference vs Object vs Instance vs Class



```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        House blueHouse = new House("blue");
        House anotherHouse = blueHouse;
```

```
        System.out.println(blueHouse.getColor()); // prints blue
        System.out.println(anotherHouse.getColor()); // blue
```

```
        anotherHouse.setColor("red");
        System.out.println(blueHouse.getColor()); // red
        System.out.println(anotherHouse.getColor()); // red
```

```
        House greenHouse = new House("green");
        anotherHouse = greenHouse;
```

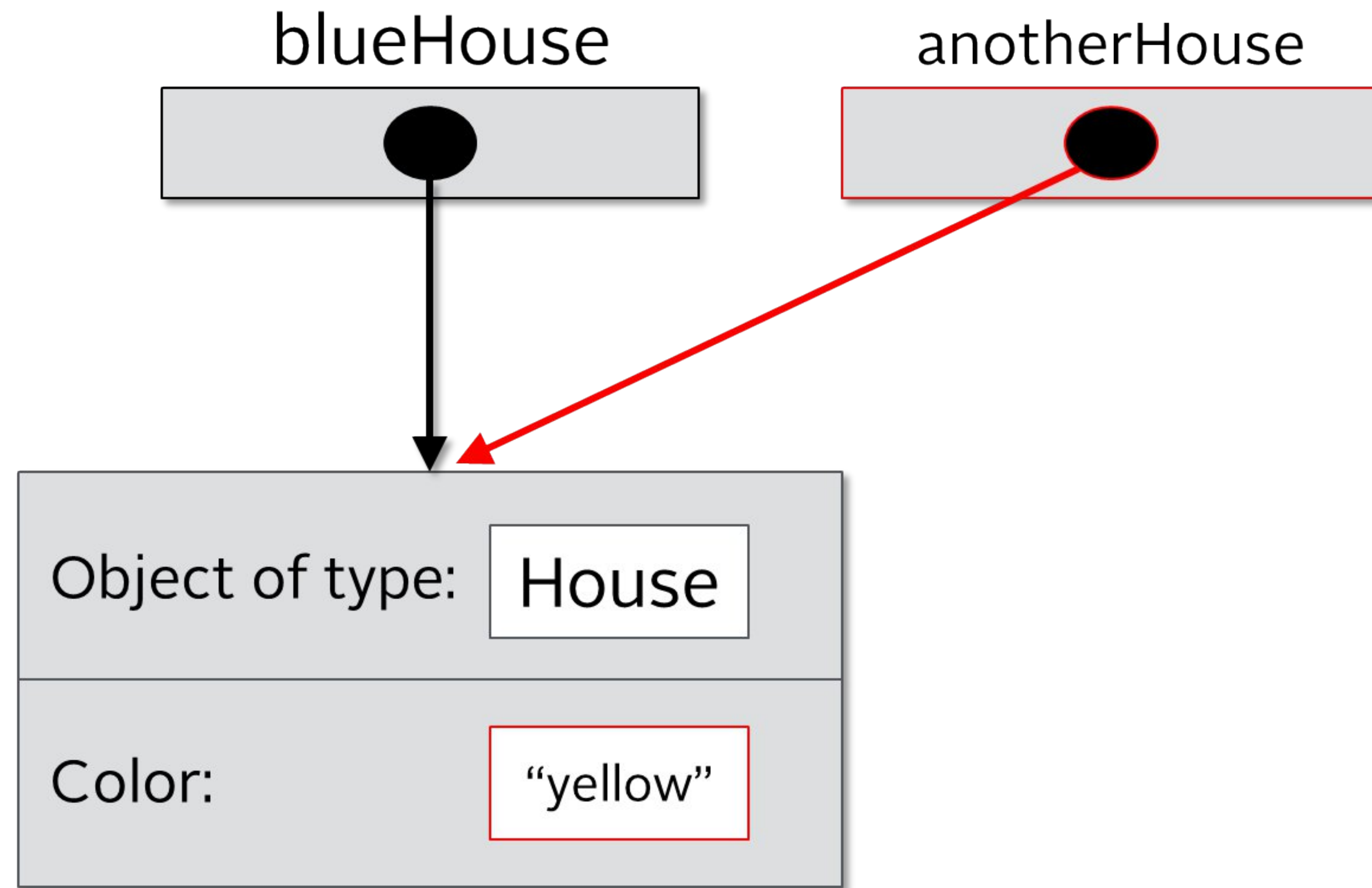
```
        System.out.println(blueHouse.getColor()); //red
        System.out.println(greenHouse.getColor()); // green
        System.out.println(anotherHouse.getColor()); // green
```

```
    }
```

```
}
```

Next we have two println statements that print the blueHouse color and anotherHouse color. Both will print “blue” since we have two **references** to the same **object**.

Reference vs Object vs Instance vs Class



```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        House blueHouse = new House("blue");
        House anotherHouse = blueHouse;
```

```
        System.out.println(blueHouse.getColor()); // prints blue
        System.out.println(anotherHouse.getColor()); // blue
```

```
        anotherHouse.setColor("yellow");
```

```
        System.out.println(blueHouse.getColor()); // yellow
        System.out.println(anotherHouse.getColor()); // yellow
```

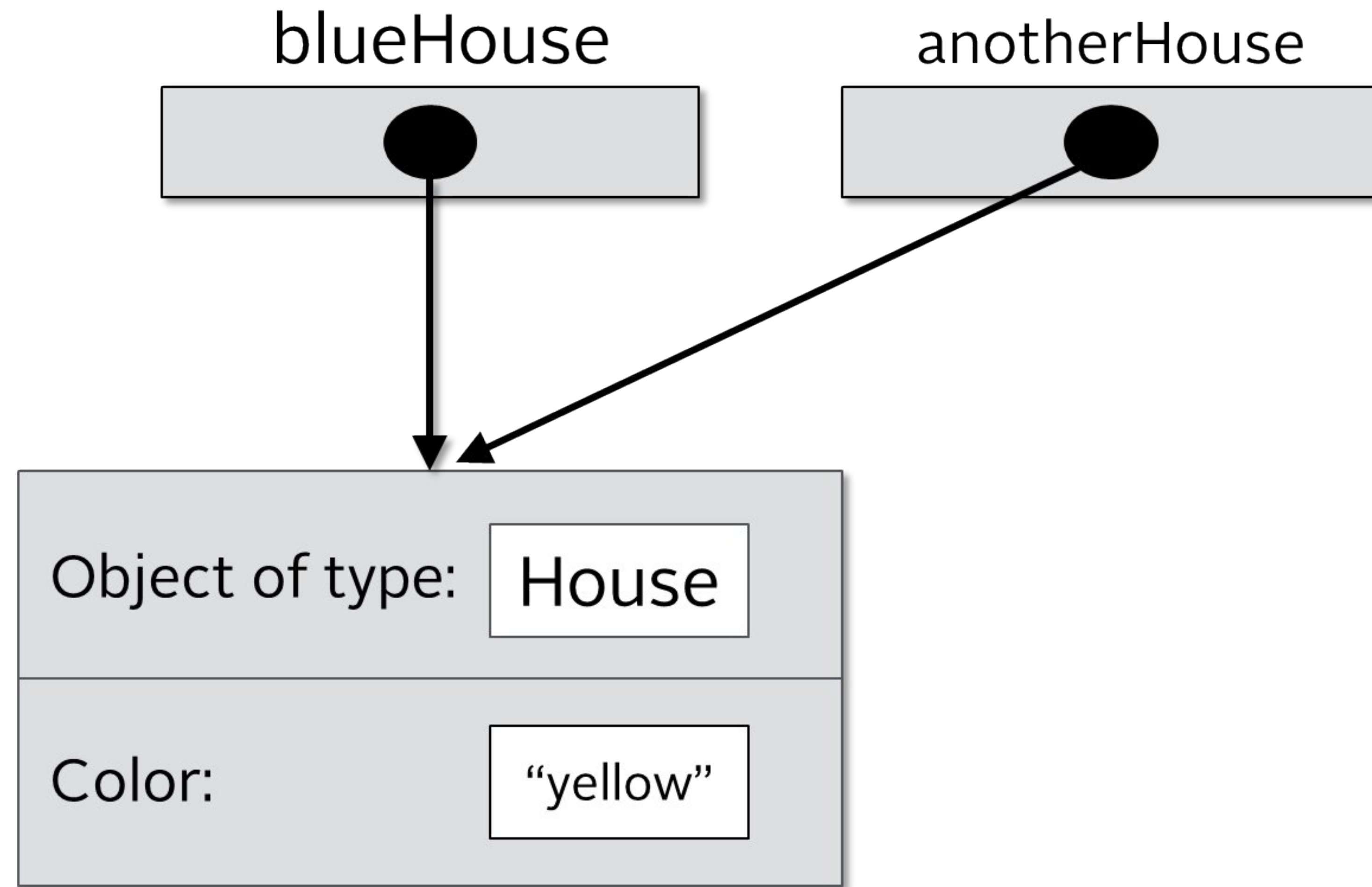
```
        House greenHouse = new House("green");
        anotherHouse = greenHouse;
```

```
        System.out.println(blueHouse.getColor()); //yellow
        System.out.println(greenHouse.getColor()); // green
        System.out.println(anotherHouse.getColor()); // green
```

```
    }
}
```

The next line calls the method `setColor` and sets the color to yellow. To the left you can see that both `blueHouse` and `anotherHouse` have the same color now. Why? Remember we have two **references** that point to the same **object** in memory. Once we change the color, of one, **both references** still point to the same **object**. In our real world example, there is still just one physical house at that one address, even though we have written the same address on two pieces of paper.

Reference vs Object vs Instance vs Class



```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        House blueHouse = new House("blue");  
        House anotherHouse = blueHouse;
```

```
        System.out.println(blueHouse.getColor()); // prints blue  
        System.out.println(anotherHouse.getColor()); // blue
```

```
        anotherHouse.setColor("yellow");
```

```
        System.out.println(blueHouse.getColor()); // yellow  
        System.out.println(anotherHouse.getColor()); // yellow
```

```
        House greenHouse = new House("green");  
        anotherHouse = greenHouse;
```

```
        System.out.println(blueHouse.getColor()); //yellow  
        System.out.println(greenHouse.getColor()); // green  
        System.out.println(anotherHouse.getColor()); // green
```

```
    }
```

```
}
```

Here we have two println statements that are printing the color. Both now print "yellow" since we still have two **references** that point to the same **object** in memory. Notice the arrows on the left hand side.

Reference vs Object vs Instance vs Class

```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        House blueHouse = new House("blue");
        House anotherHouse = blueHouse;
```

```
        System.out.println(blueHouse.getColor()); // prints blue
        System.out.println(anotherHouse.getColor()); // blue
```

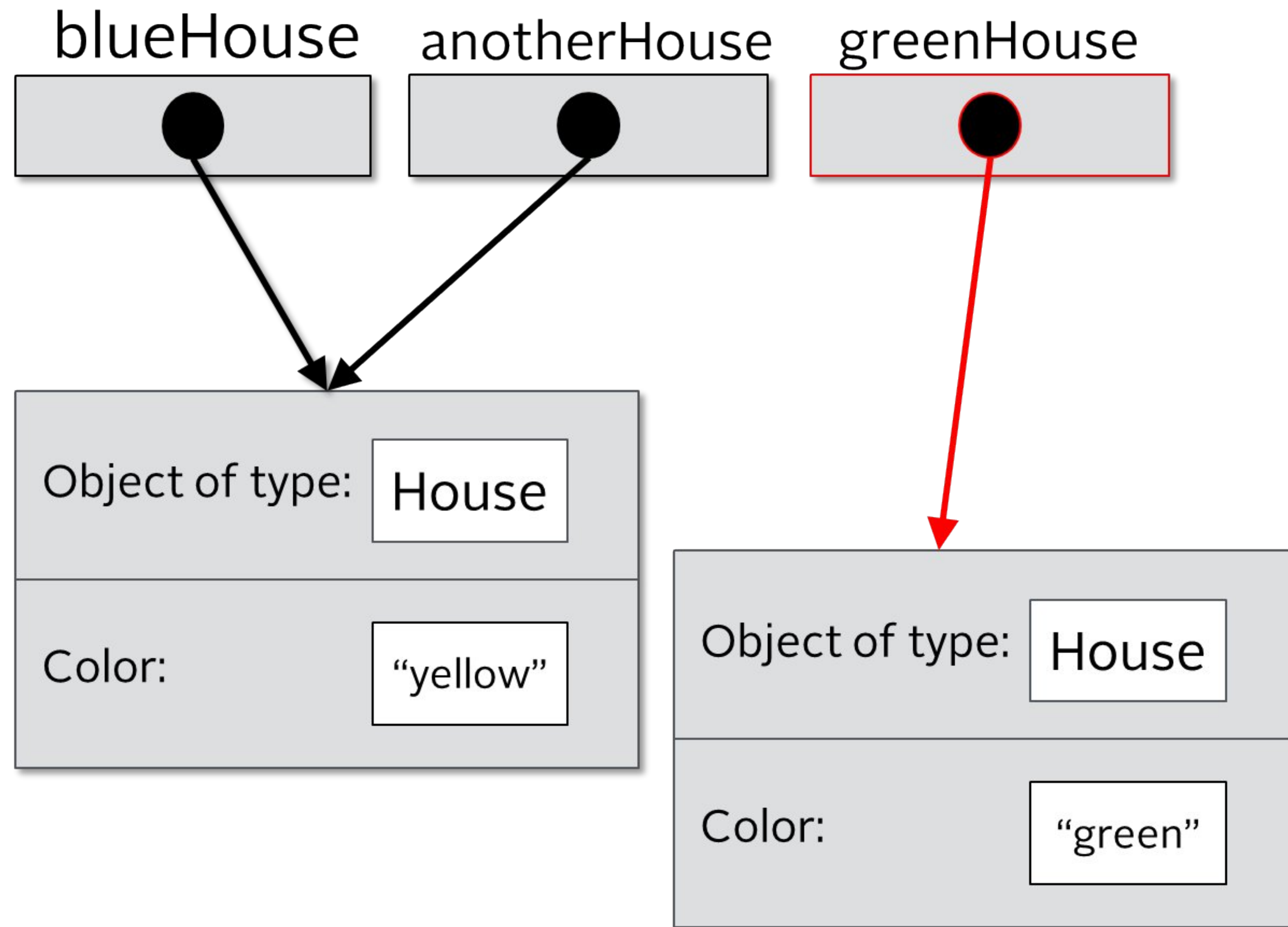
```
        anotherHouse.setColor("yellow");
        System.out.println(blueHouse.getColor()); // yellow
        System.out.println(anotherHouse.getColor()); // yellow
```

```
        House greenHouse = new House("green");
        anotherHouse = greenHouse;
```

```
        System.out.println(blueHouse.getColor()); //yellow
        System.out.println(greenHouse.getColor()); // green
        System.out.println(anotherHouse.getColor()); // green
```

```
    }
```

```
}
```



Here we are creating another new instance of the House class with the color set to "green". Now we have two **objects** in memory but we have three **references** which are `blueHouse`, `anotherHouse` and `greenHouse`. The variable (**reference**) `greenHouse` points to a different **object** in memory, but `blueHouse` and `anotherHouse` point to the same object in memory.

Reference vs Object vs Instance vs Class

```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        House blueHouse = new House("blue");  
        House anotherHouse = blueHouse;
```

```
        System.out.println(blueHouse.getColor()); // prints blue  
        System.out.println(anotherHouse.getColor()); // blue
```

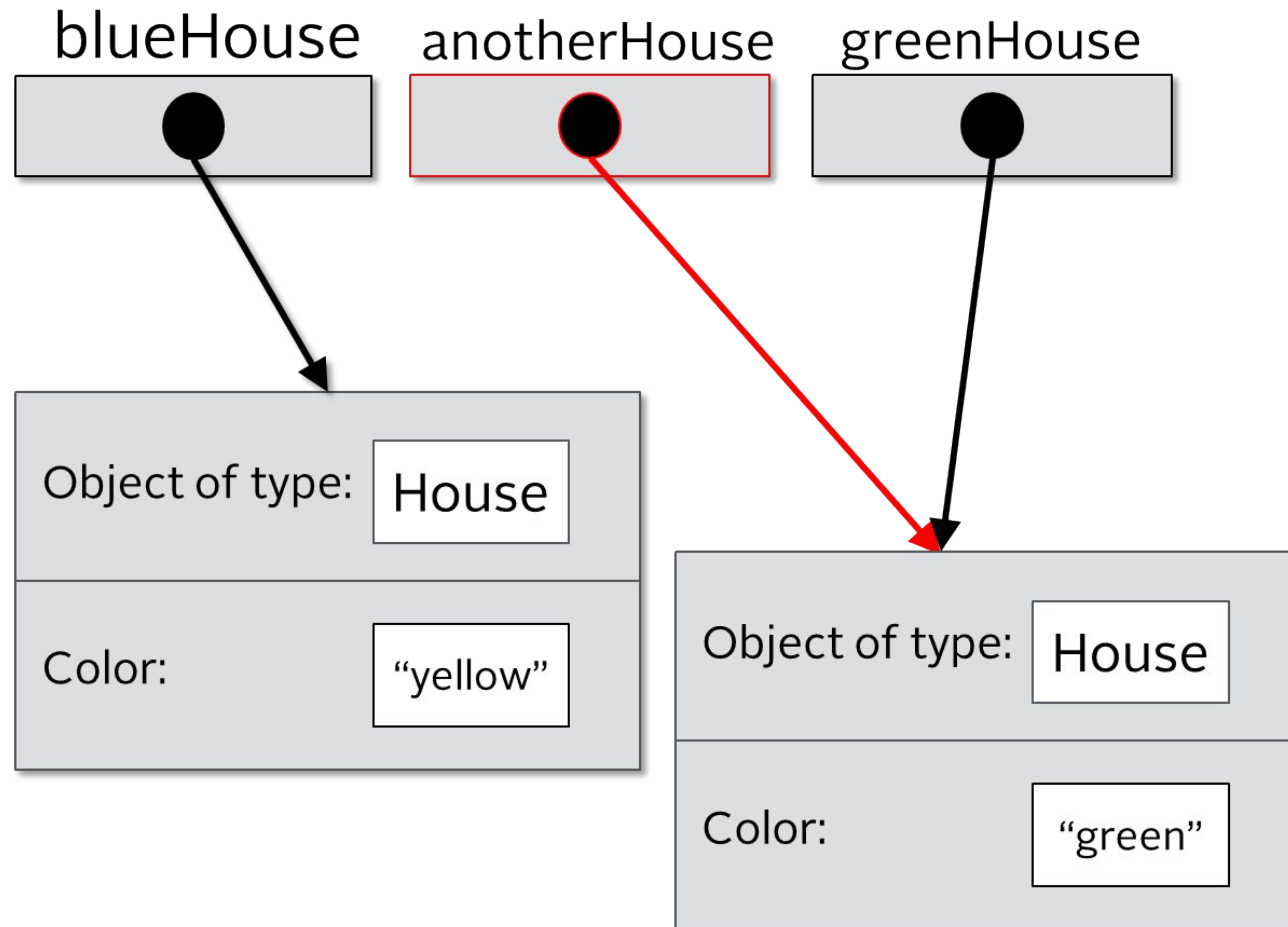
```
        anotherHouse.setColor("yellow");  
        System.out.println(blueHouse.getColor()); // yellow  
        System.out.println(anotherHouse.getColor()); // yellow
```

```
        House greenHouse = new House("green");  
        anotherHouse = greenHouse;
```

```
        System.out.println(blueHouse.getColor()); //yellow  
        System.out.println(greenHouse.getColor()); // green  
        System.out.println(anotherHouse.getColor()); // green
```

```
    }
```

```
}
```



Here we assign `greenHouse` to `anotherHouse`. In other words, we are dereferencing `anotherHouse`. It will now point to a different **object** in memory. Before it was pointing to a house that had the "yellow" color, now it points to the house that has the "green" color. In this scenario, we still have three **references** and two **objects** in memory, but `blueHouse` points to one **object** while `anotherHouse` and `greenHouse` point to the same **object** in memory.

Reference vs Object vs Instance vs Class

```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        House blueHouse = new House("blue");  
        House anotherHouse = blueHouse;
```

```
        System.out.println(blueHouse.getColor()); // prints blue  
        System.out.println(anotherHouse.getColor()); // blue
```

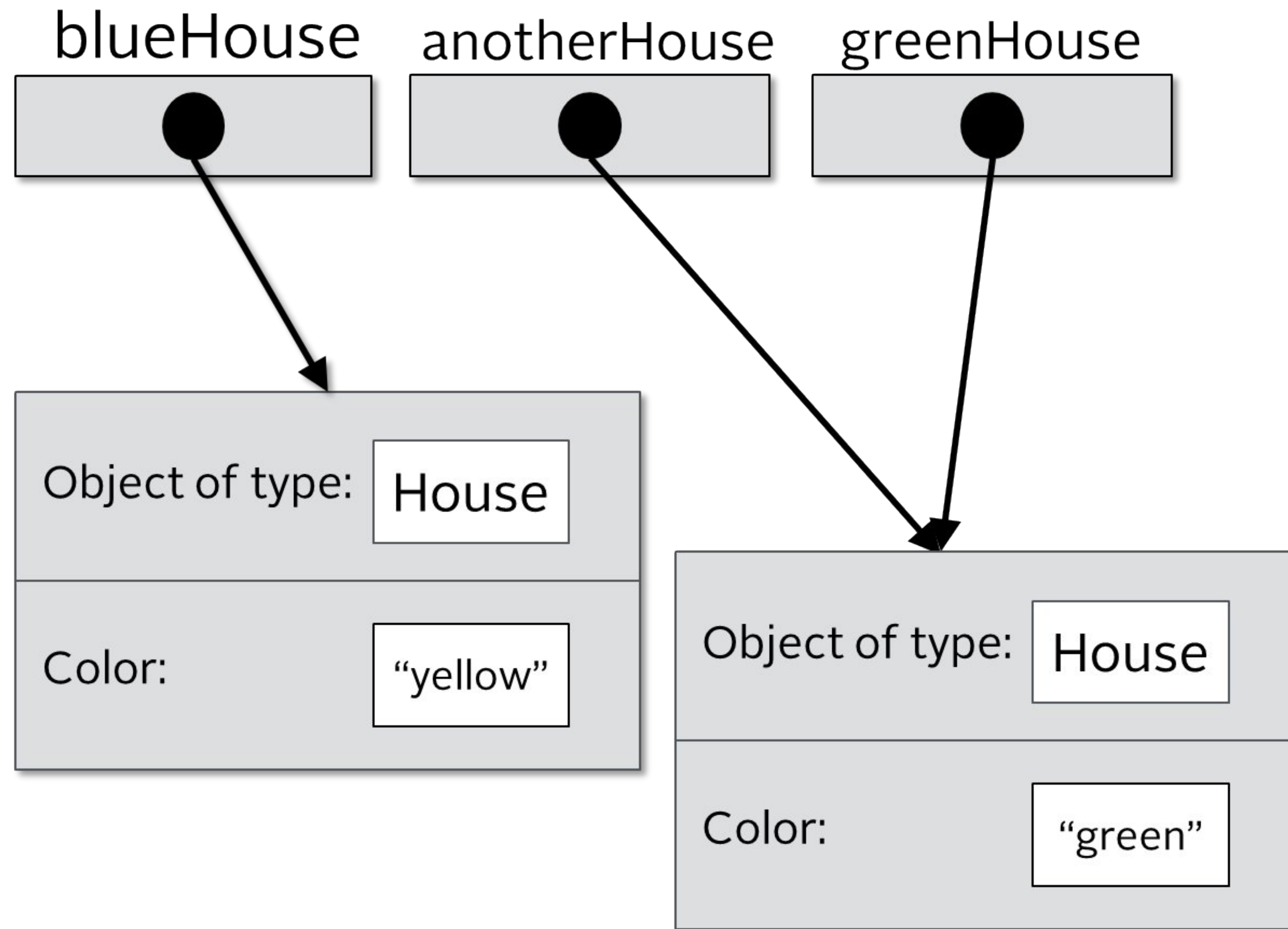
```
        anotherHouse.setColor("yellow");  
        System.out.println(blueHouse.getColor()); // yellow  
        System.out.println(anotherHouse.getColor()); // yellow
```

```
        House greenHouse = new House("green");  
        anotherHouse = greenHouse;
```

```
        System.out.println(blueHouse.getColor()); //yellow  
        System.out.println(greenHouse.getColor()); // green  
        System.out.println(anotherHouse.getColor()); // green
```

```
    }
```

```
}
```



Finally we have three `println` statements. The first will print `"yellow"` since the `blueHouse` **variable(reference)** points to the **object** in memory that has the `"yellow"` color, while the next two lines will print `"green"` since both `anotherHouse` and `greenHouse` point to same **object** in memory.

The reference vs The object

Consider the code on this slide.

1	<code>new</code> House("red");	<i>// house object gets created in memory</i>
2	House myHouse = <code>new</code> House("beige");	<i>// house object gets created in memory</i> <i>// and it's location (reference) is</i> <i>// assigned to myHouse</i>
3	House redHouse = <code>new</code> House("red");	<i>// house object gets created in memory</i> <i>// and it's location (reference) is</i> <i>// assigned to redHouse</i>

On the first line, I create a new House and make it red.

But I am not assigning this to any variable.

The reference vs The object

```
1 new House("red"); // house object gets created in memory
```

This compiles fine and you can do this.

This object is created in memory, but after that statement completes, my code has no way to access it.

The object exists in memory, but I can't communicate with it after that statement is executed.

That's because I didn't create a reference to it.

The reference vs The object

On the second line, I do create a reference to the house object I created.

```
2 House myHouse = new House("beige");    // house object gets created in memory
                                           // and it's location (reference) is
                                           // assigned to myHouse
```

My reference, the variable I call myHouse, lets me have access to that beige house as long as my variable, myHouse, stays in scope. Or until it gets reassigned to reference a different object.

The reference vs The object

I'm creating a red house again, but this is a different object altogether from the red house I created on line one.

```
3 House redHouse = new House("red");           // house object gets created in memory
                                                // and it's location (reference) is
                                                // assigned to redHouse
```

This third statement is creating yet another house object in memory which has no relationship to the one I created on the first line.

```
1 new House("red");                             // house object gets created in memory
2 House myHouse = new House("beige");           // house object gets created in memory
                                                // and it's location (reference) is
                                                // assigned to myHouse
```


The reference vs The object

```
1  new House("red");           // house object gets created in memory
2  House myHouse = new House("beige"); // house object gets created in memory
                                     // and it's location (reference) is
                                     // assigned to myHouse
3  House redHouse = new House("red"); // house object gets created in memory
                                     // and it's location (reference) is
                                     // assigned to redHouse
```

This code has three instances of house but only two references.

That first object is said to be eligible for garbage collection immediately after that first statement.

It's useless to the code because It's no longer accessible.

The reference vs The object

```
1 new House("red"); // house object gets created in memory
2 House myHouse = new House("beige"); // house object gets created in memory
                                     // and it's location (reference) is
                                     // assigned to myHouse
3 House redHouse = new House("red"); // house object gets created in memory
                                     // and it's location (reference) is
                                     // assigned to redHouse
```

There are times we might want to instantiate an object and immediately call a method on it.

99 percent of the time, we'll want to reference the objects we create.