ALIASING!

WHAT IS ALIASING???

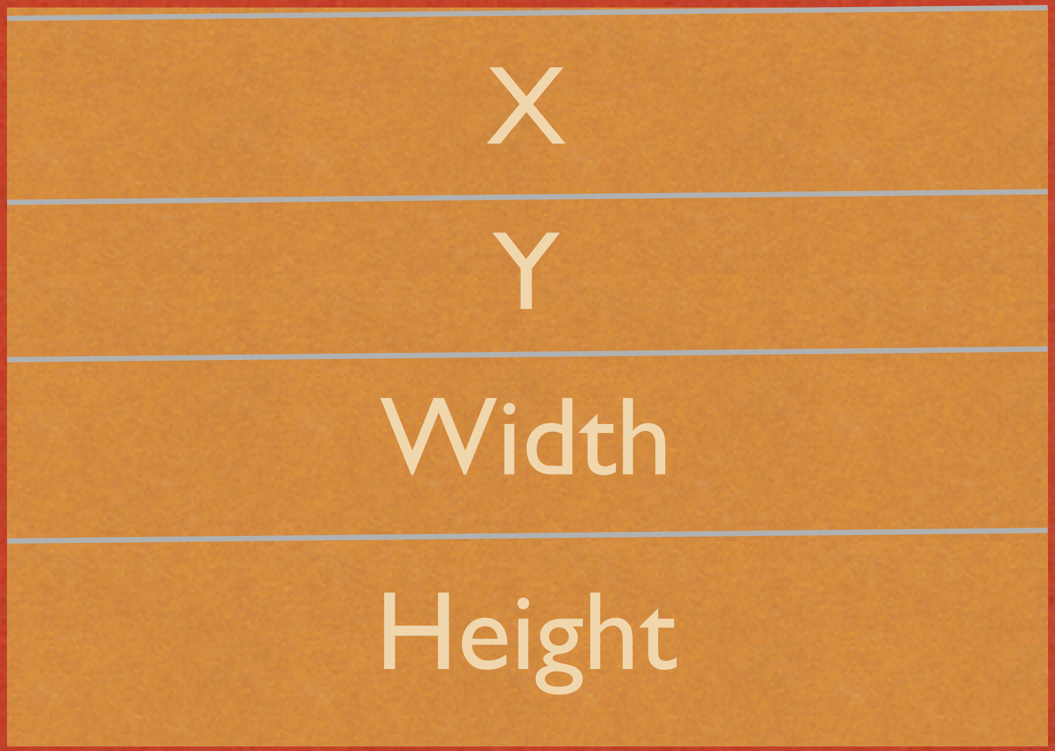
- Aliasing is when two **object** variables refer to the same thing!

SOME BACKGROUND KNOWLEDGE

Let's suppose a class named **Rectangle** has 4 private instance variables:

x
y
width
height

This is a **visual** representation of a single Rectangle object!



SOME BACKGROUND KNOWLEDGE

Let's define the rectangle's constructor like this:

```
public Rectangle(int setX, int setY,
                    int setWidth, int setHeight) {
        x = setX;
        y = setY;
        width = setWidth;
        height = setHeight;
}
```

(Oh, and we have setters and getters for each variable)

SOME BACKGROUND KNOWLEDGE

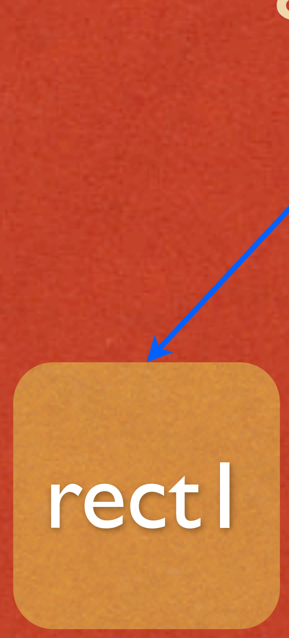
What's really happening when we do this??

```
Rectangle rect1 = Rectangle(0, 0, 50, 50);
```

SOME BACKGROUND KNOWLEDGE

What's really happening when we do this??

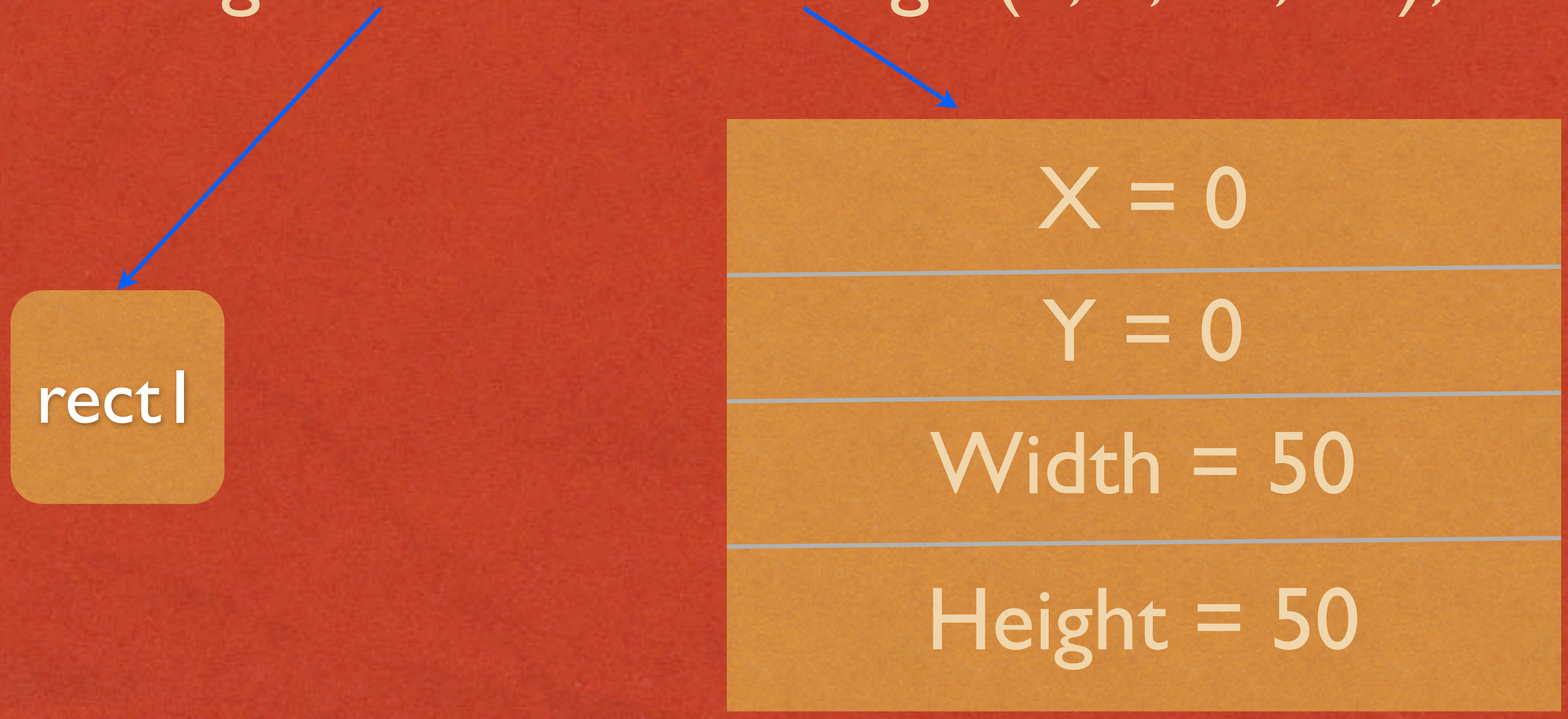
```
Rectangle rect1 = Rectangle(0, 0, 50, 50);
```



SOME BACKGROUND KNOWLEDGE

What's really happening when we do this??

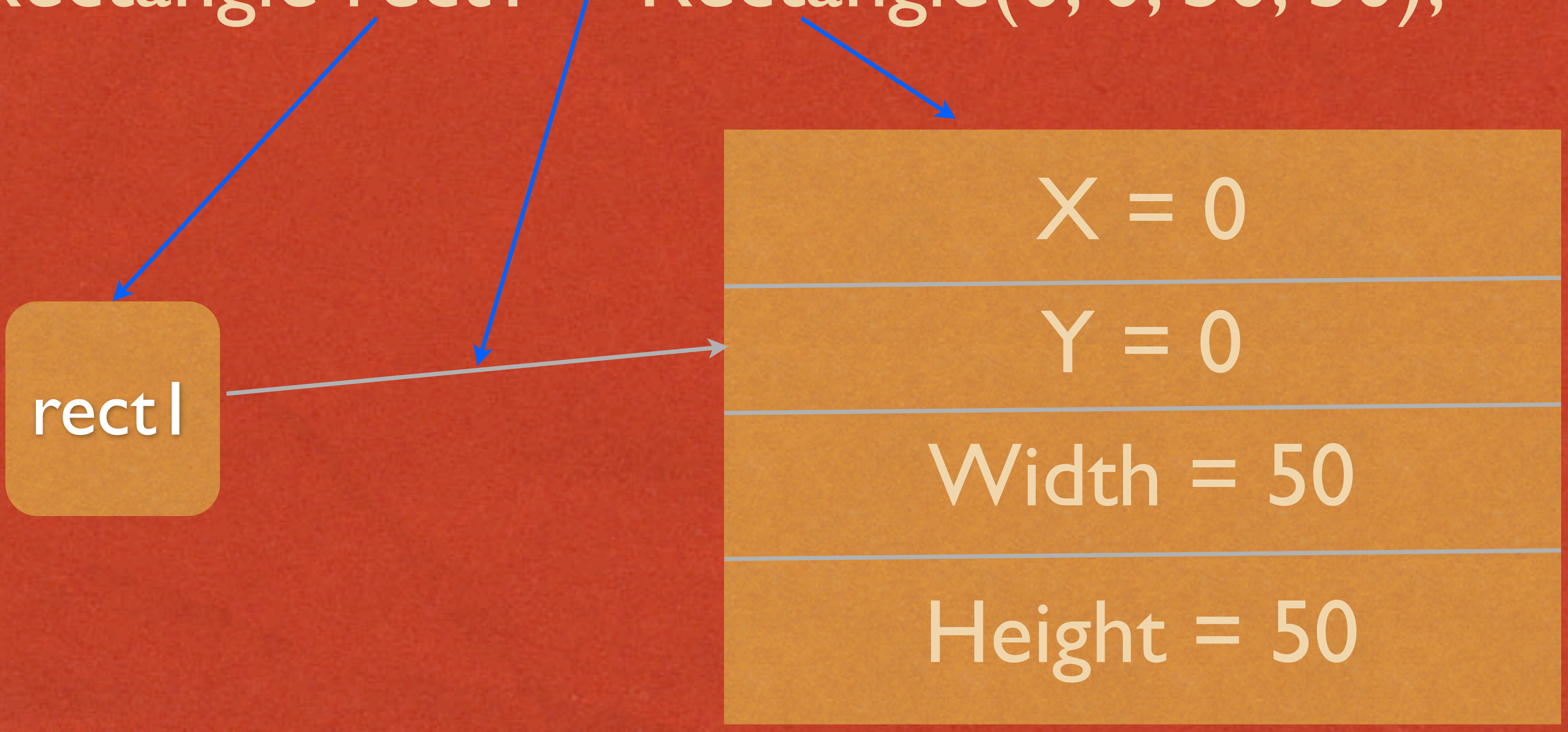
```
Rectangle rect1 = Rectangle(0, 0, 50, 50);
```



SOME BACKGROUND KNOWLEDGE

What's really happening when we do this??

```
Rectangle rect1 = Rectangle(0, 0, 50, 50);
```

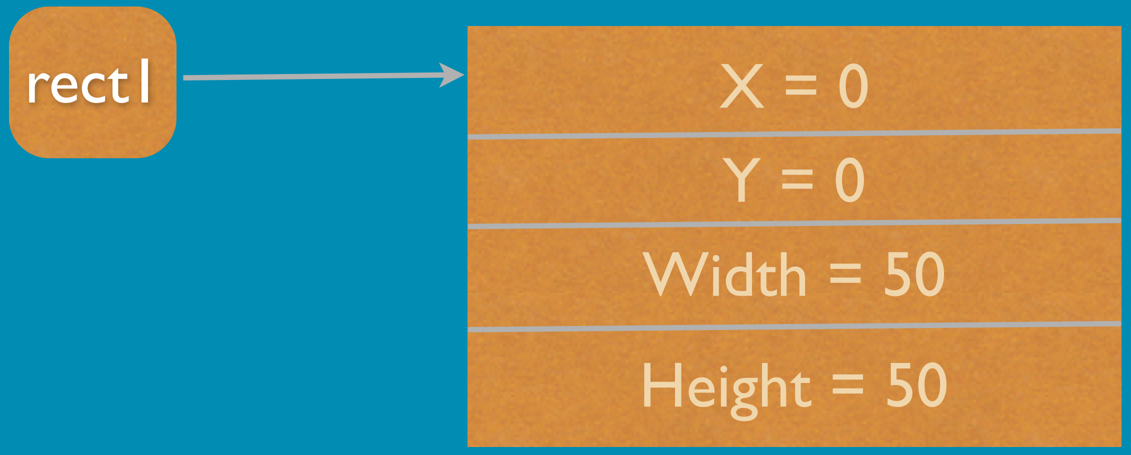


ALIASING EXAMPLE

Our current state

Inside our main:

```
Rectangle rect1 = Rectangle(0, 0, 50, 50);
```

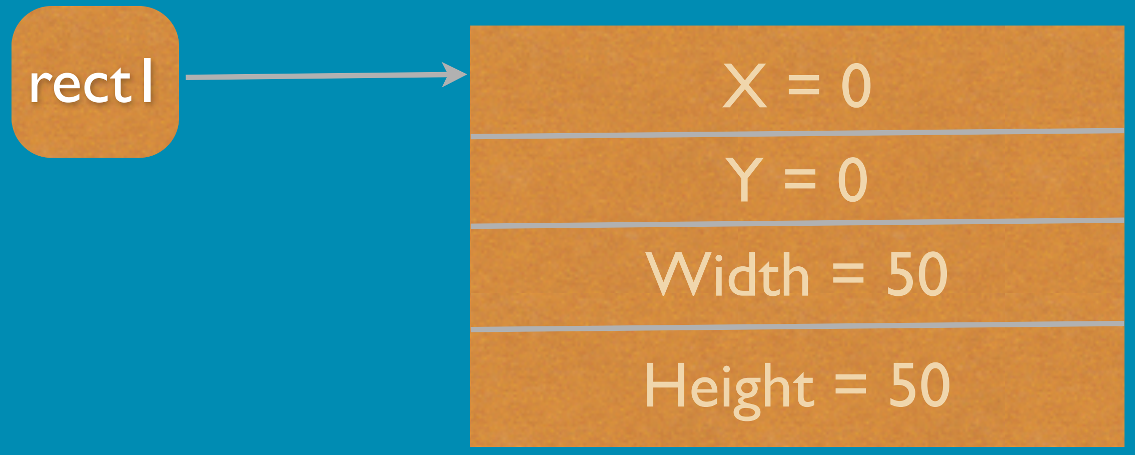


Let's declare another rectangle object called rect2

ALIASING EXAMPLE

Inside our main:

```
Rectangle rect1 = Rectangle(0, 0, 50, 50);
Rectangle rect2;
```

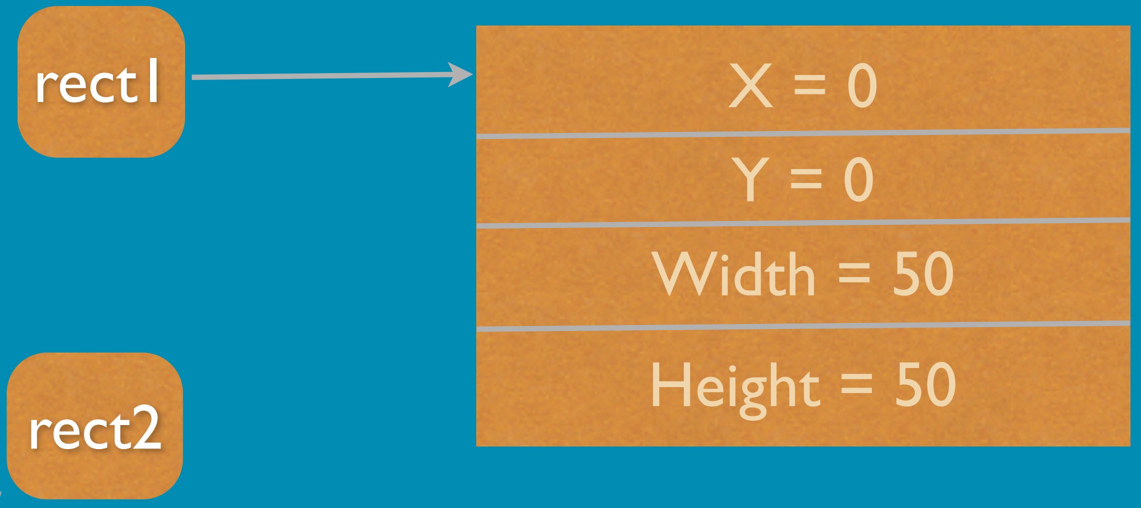


There's our newly declared rect2

ALIASING EXAMPLE

Inside our main:

```
Rectangle rect1 = Rectangle(0, 0, 50, 50);
Rectangle rect2;
```



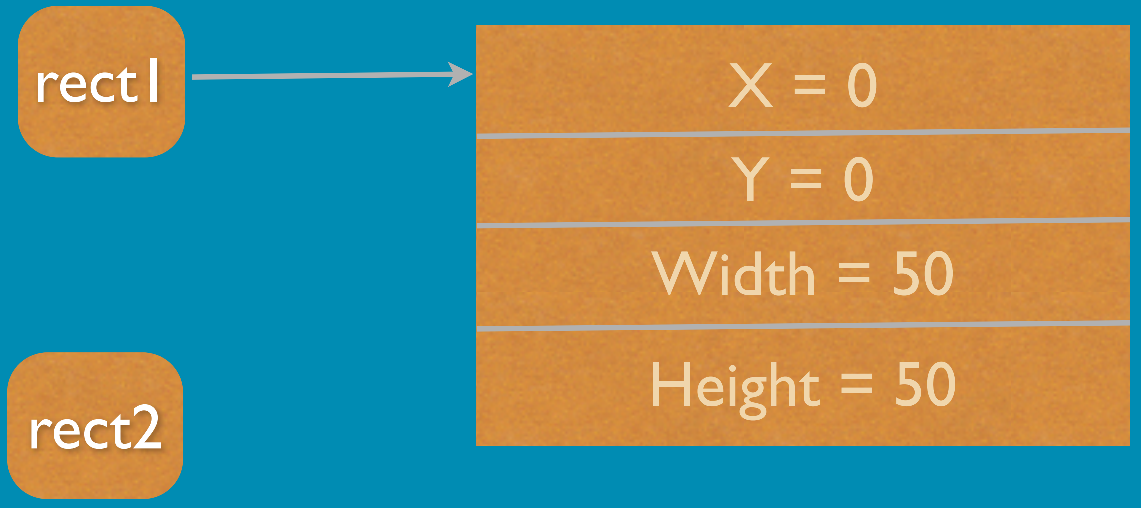
This is what it looks like graphically

There's our newly declared rect2

ALIASING EXAMPLE

Inside our main:

```
Rectangle rect1 = Rectangle(0, 0, 50, 50);
Rectangle rect2;
rect2 = rect1;
```

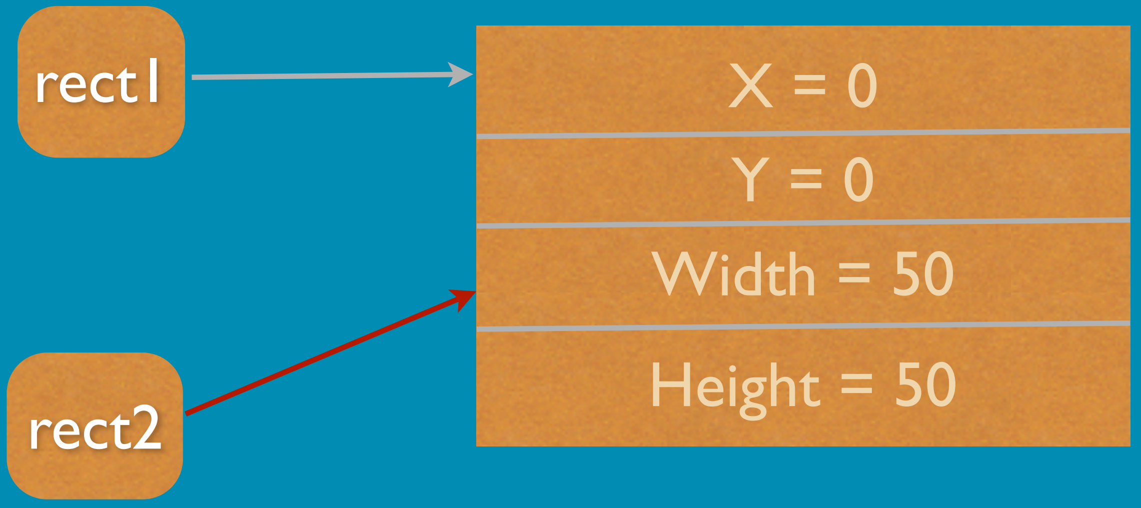


What's happening when we assign rect2 to rect1??

ALIASING EXAMPLE

Inside our main:

```
Rectangle rect1 = Rectangle(0, 0, 50, 50);
Rectangle rect2;
rect2 = rect1;
```

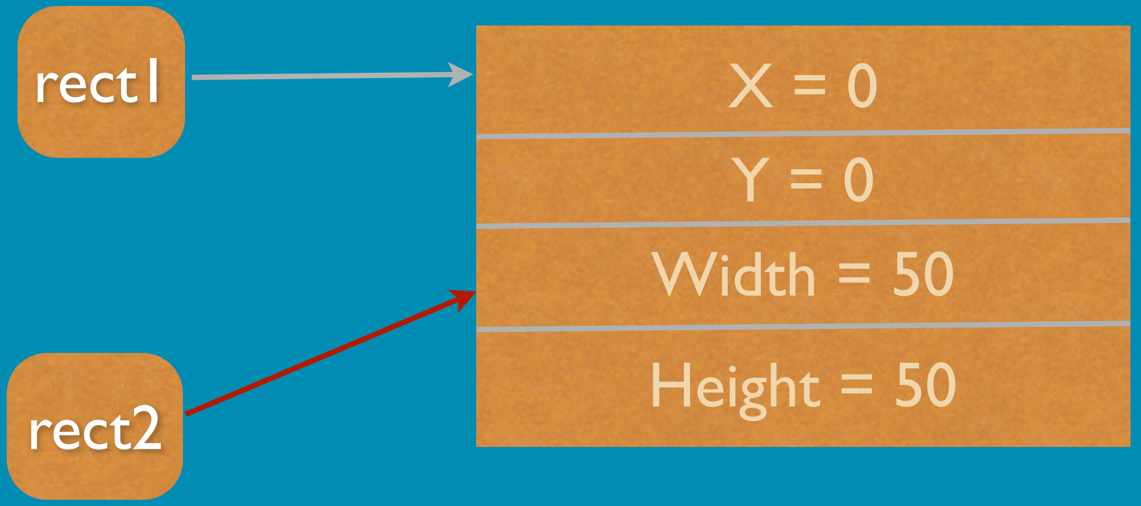


Turns out rect2 now "points" to the same block as rect1

ALIASING EXAMPLE

Inside our main:

```
Rectangle rect1 = Rectangle(0, 0, 50, 50);
Rectangle rect2;
rect2 = rect1;
```

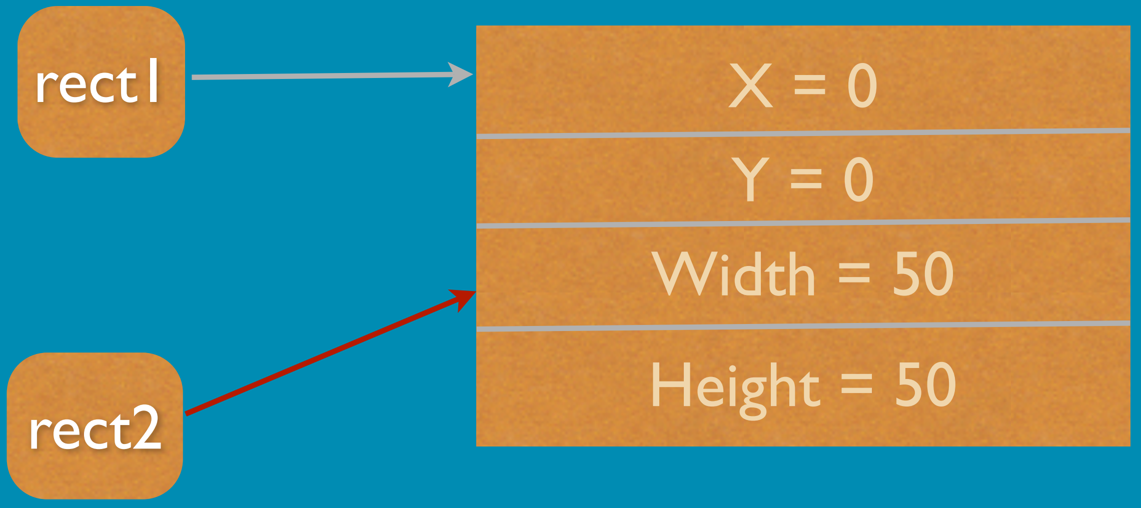


That's aliasing! It's when two **Object** variables refer to the same thing

ALIASING EXAMPLE

Inside our main:

```
Rectangle rect1 = Rectangle(0, 0, 50, 50);
Rectangle rect2;
rect2 = rect1;
```

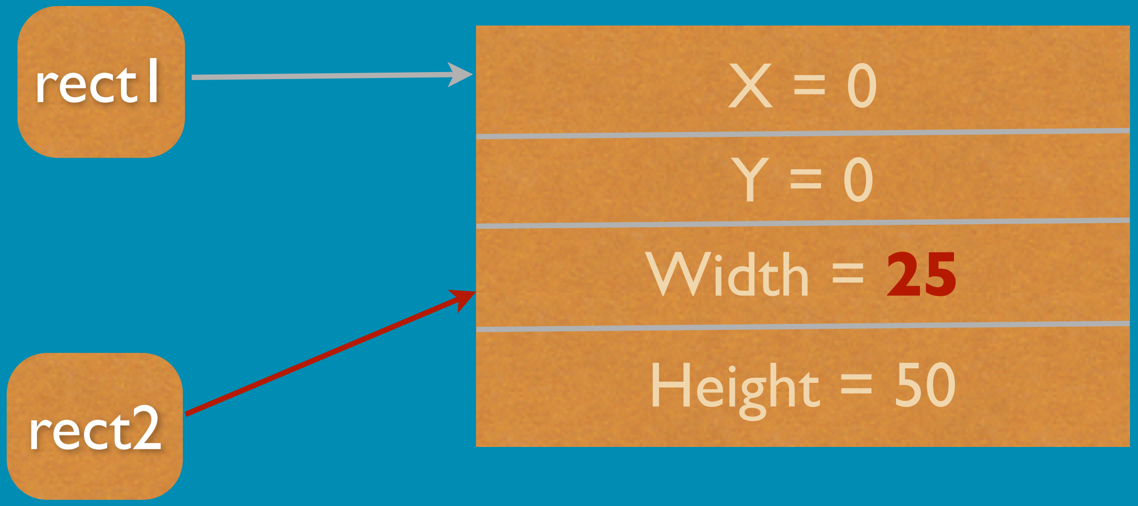


Ok, so if rect1 and rect2 refer to the same thing, what happens when you modify rect1?

ALIASING EXAMPLE

Inside our main:

```
Rectangle rect1 = Rectangle(0, 0, 50, 50);
Rectangle rect2;
rect2 = rect1;
rect1.setWidth(25);
```



Ok, so if rect1 and rect2 refer to the same thing, what happens when you modify rect1?

So it turns out that rect2 is also implicitly modified since rect2 and rect1 are the exact same thing!!

A SMALL DETAIL...

- Aliasing **only** works for **objects**
- Ints, bools, doubles are **primitives**
- So aliasing does **not** work for ints, bools or doubles

A SMALL DETAIL... EXAMPLE

Inside our main:

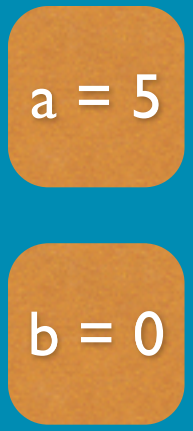
```
int a = 5;
```

a = 5

A SMALL DETAIL... EXAMPLE

Inside our main:

```
int a = 5;
int b;
```



A SMALL DETAIL... EXAMPLE

Inside our main:

```
int a = 5;
int b;
b = a;
```

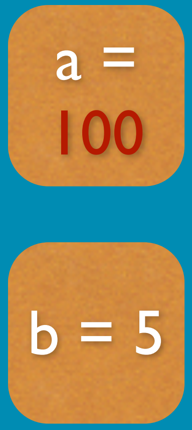
a = 5

b = 5

A SMALL DETAIL... EXAMPLE

Inside our main:

```
int a = 5;
int b;
b = a;
a = 100;
```



No aliasing occurred. So when a was modified, b remained unchanged!

ADVANCED ALIASING!~

So we have this method:

```
public static void modifyRectangle(Rectangle someRect) {
    someRect.setX(100);
    someRect.setY(100);
}
```

ADVANCED ALIASING!~

So we have this method:

```
public static void modifyRectangle(Rectangle someRect) {
    someRect.setX(100);
    someRect.setY(100);
}
```

And inside our main:

```
Rectangle rect1 = Rectangle(0, 0, 50, 50);
modifyRectangle(rect1);
System.out.println(rect1.getX());
System.out.println(rect1.getY());
```

ADVANCED ALIASING!~

So we have this method:

```
public static void modifyRectangle(Rectangle someRect) {
    someRect.setX(100);
    someRect.setY(100);
}
```

And inside our main:

```
Rectangle rect1 = Rectangle(0, 0, 50, 50);
modifyRectangle(rect1);
System.out.println(rect1.getX());
System.out.println(rect1.getY());
```

What will this print out??

ADVANCED ALIASING!~

So we have this method:

```
public static void modifyRectangle(Rectangle someRect) {
    someRect.setX(100);
    someRect.setY(100);
}
```

And inside our main:

```
Rectangle rect1 = Rectangle(0, 0, 50, 50);
modifyRectangle(rect1);
System.out.println(rect1.getX());
System.out.println(rect1.getY());
```

What will this print out?? It will print out:

```
100
100
```

ADVANCED ALIASING!~

BUT WHY???

ADVANCED ALIASING!~

BUT WHY???

Because the rectangle was modified by aliasing!

ADVANCED ALIASING!~

BUT WHY???

Because the rectangle was modified by aliasing!

The method

```
public static void modifyRectangle(Rectangle someRect)
{
    someRect.setX(100);
    someRect.setY(100);
}
```

ADVANCED ALIASING!~

BUT WHY???

Because the rectangle was modified by aliasing!

The method

```
public static void modifyRectangle(Rectangle someRect)
{
    someRect.setX(100);
    someRect.setY(100);
}
```

The variable **someRect** will "point" to the original copy of the object passed to it!

So when you modify someRect, you're modifying the original object as well!

DOES NOT WORK FOR PRIMITIVES

So we have this method:

```
public static void modifyInt(int someInt) {
        someInt = 314159;
}
```

And inside our main:

```
int x = 5;
modifyInt(x);
System.out.println(x);
```

DOES NOT WORK FOR PRIMITIVES

So we have this method:

```
public static void modifyInt(int someInt) {
        someInt = 314159;
}
```

And inside our main:

```
int x = 5;
modifyInt(x);
System.out.println(x);
```

Will print 5

THE END

- Now you know what aliasing is