Memory

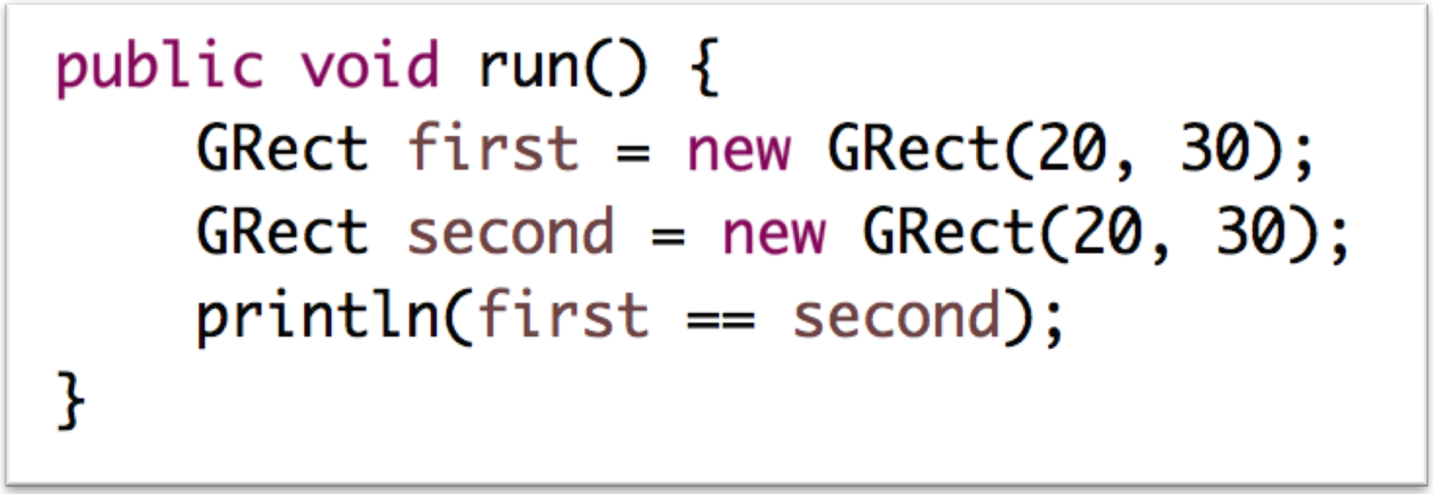
Chris Piech

CS106A, Stanford University

Learning Goals

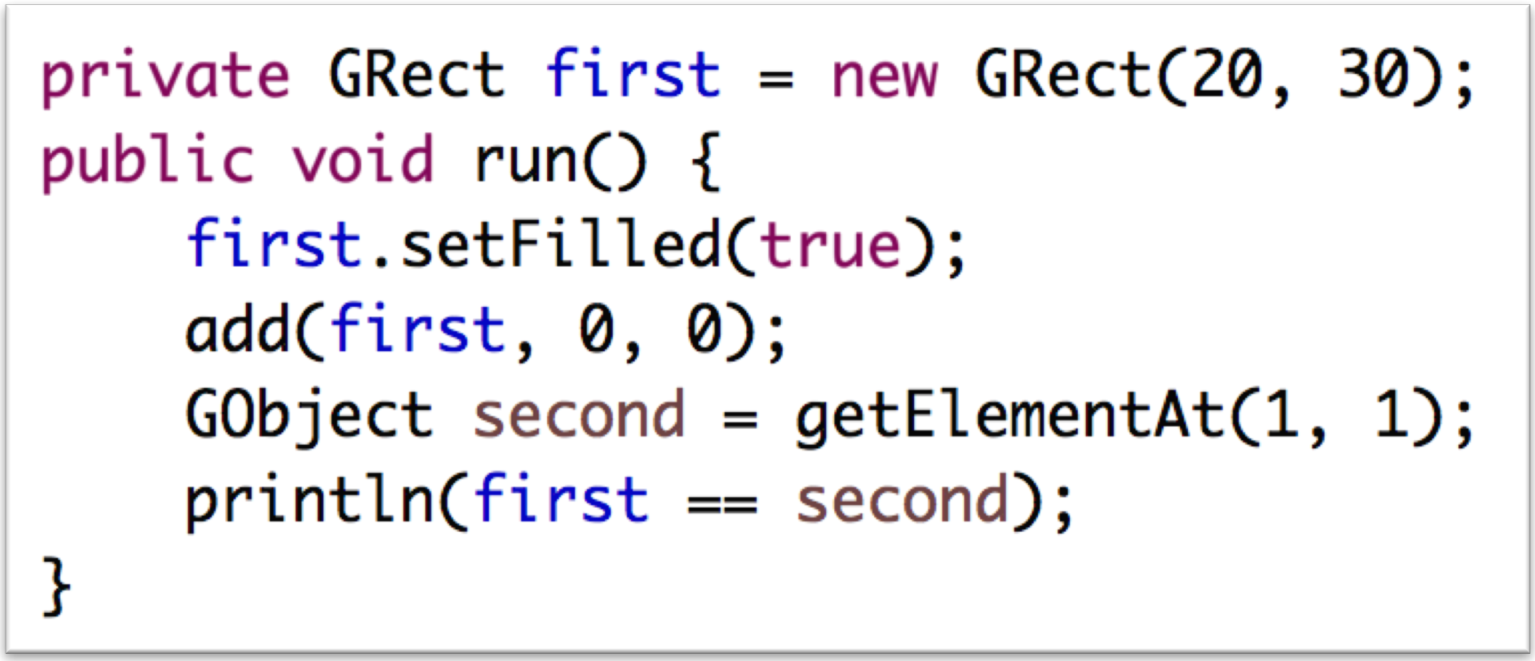
1. Be able to trace memory for objects
2. Be able to trace memory with instance variables

Who thinks this prints **`true`**?



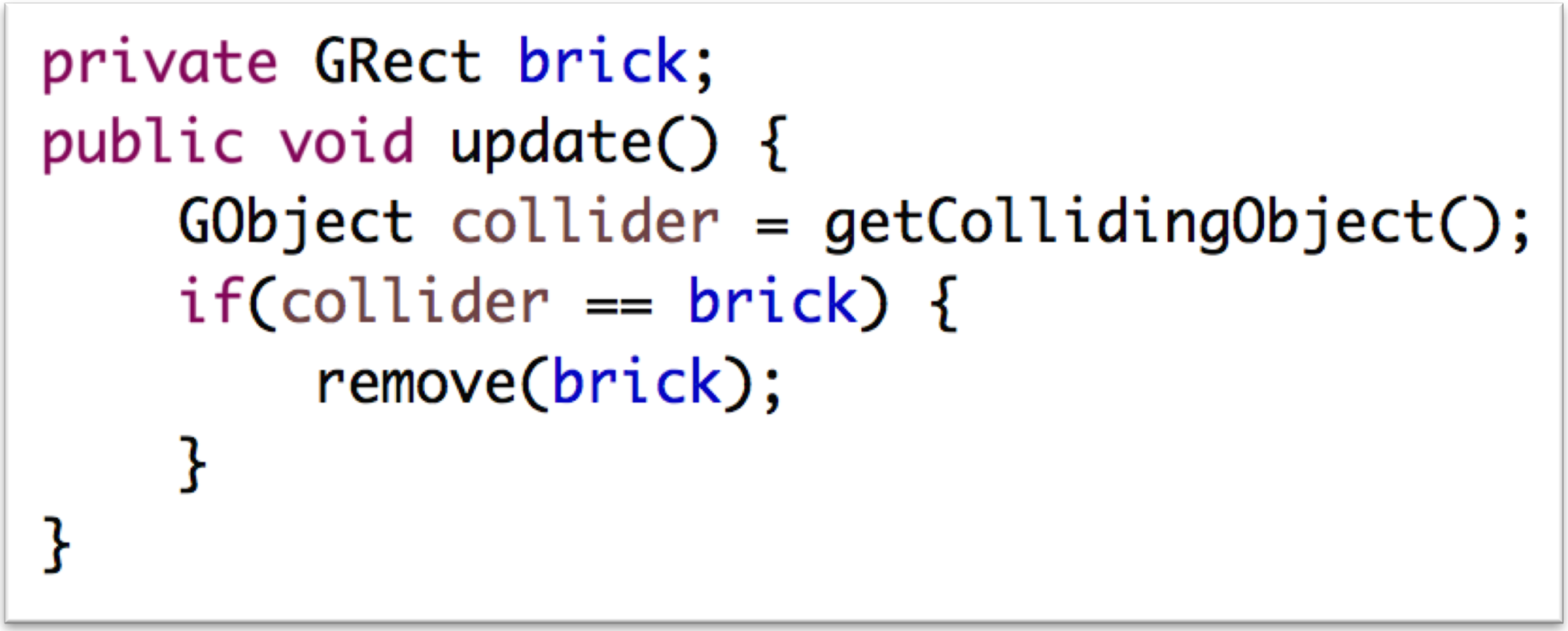
```
public void run() {
    GRect first = new GRect(20, 30);
    GRect second = new GRect(20, 30);
    println(first == second);
}
```

Who thinks this prints **`true`**?



```
private GRect first = new GRect(20, 30);
public void run() {
    first.setFilled(true);
    add(first, 0, 0);
    GObject second = getElementAt(1, 1);
    println(first == second);
}
```

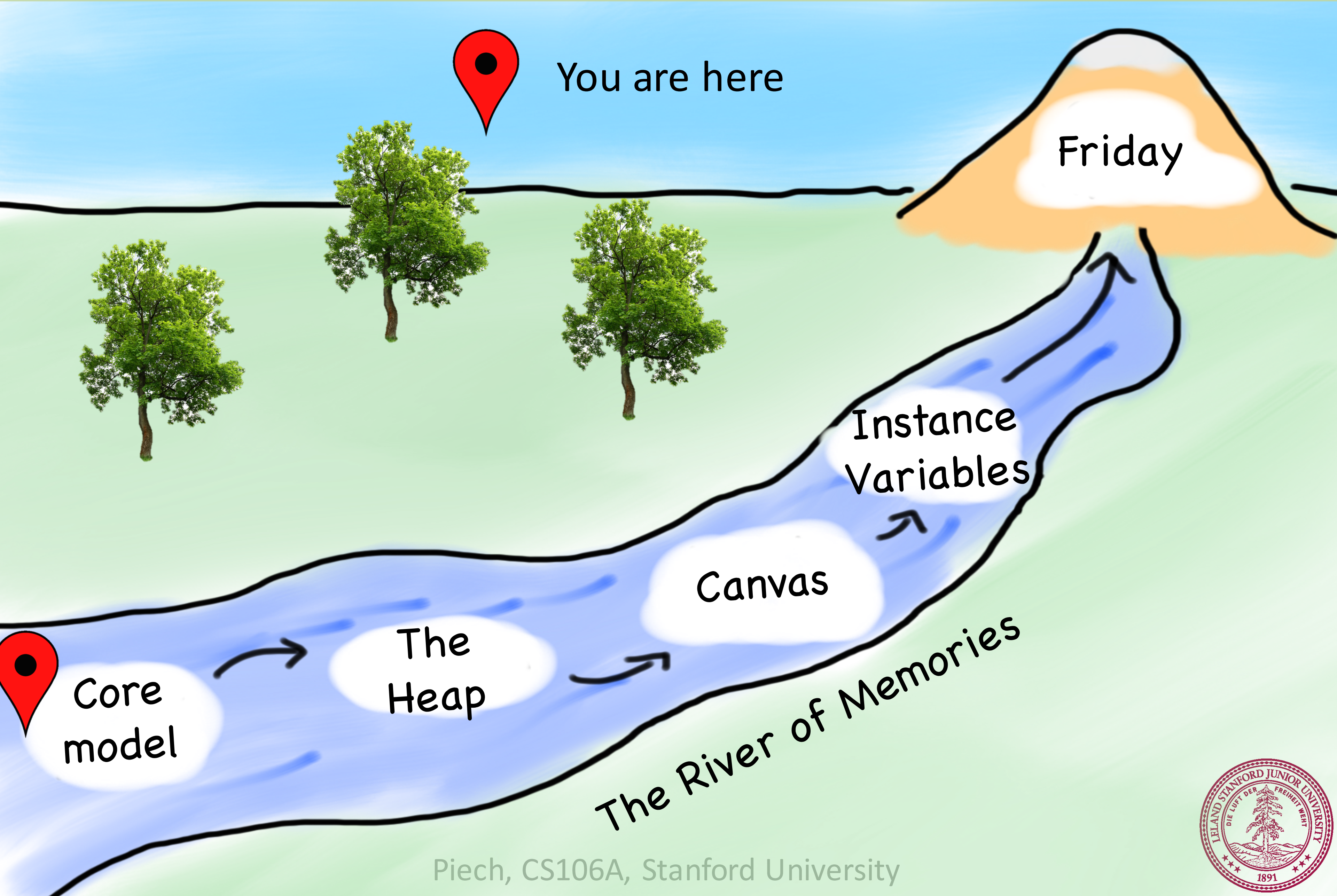
Deep Understanding is Key



```
private GRect brick;
public void update() {
    GObject collider = getCollidingObject();
    if(collider == brick) {
        remove(brick);
    }
}
```

[suspense]

Today's Route



Core memory model

Stack Diagrams

```
public void run() {
   println(toInches(5));
}

private int toInches(int feet){
   int result = feet * 12;
   return result;
}
```

Stack Diagrams

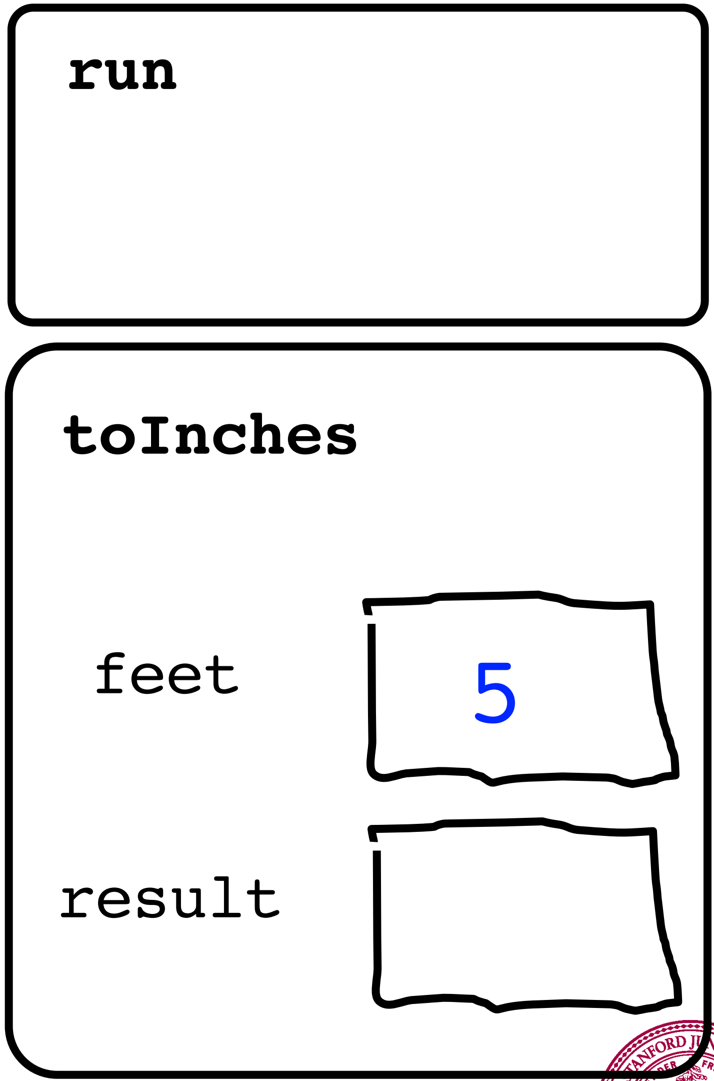
```
public void run() {
   println(toInches(5));
}

private int toInches(int feet){
   int result = feet * 12;
   return result;
}
```

Stack Diagrams

```
public void run() {
   println(toInches(5));
}

private int toInches(int feet){
   int result = feet * 12;
   return result;
}
```

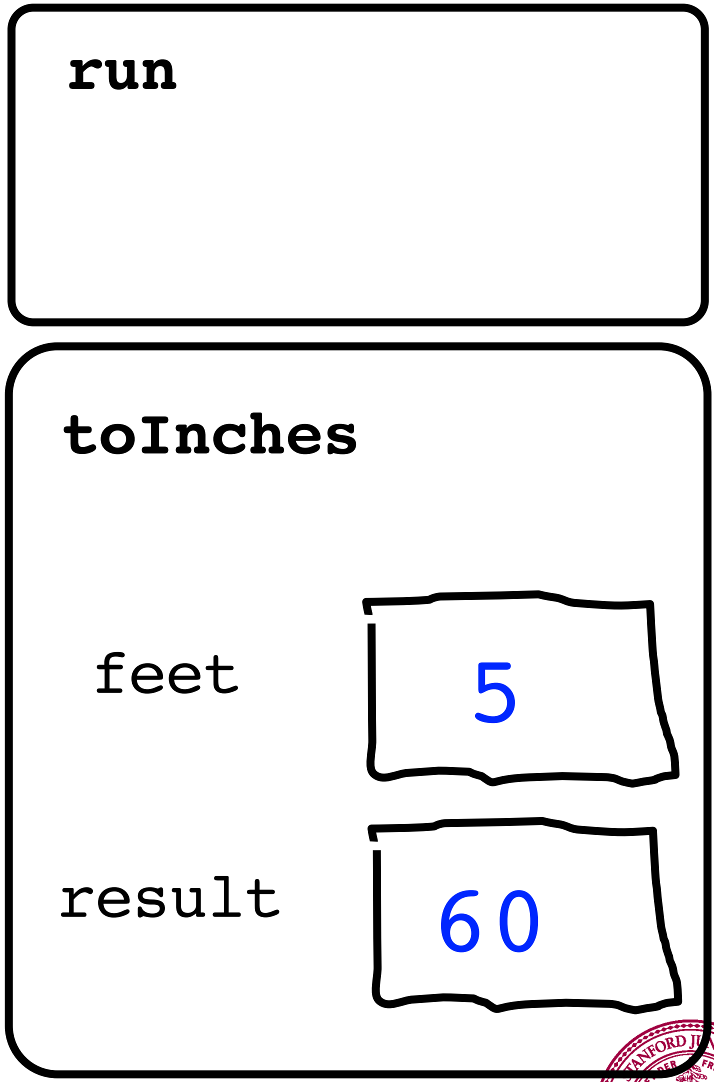
Stack Diagrams

```
public void run() {
   println(toInches(5));
}

private int toInches(int feet){
   int result = feet * 12;
   return result;
}
f
```



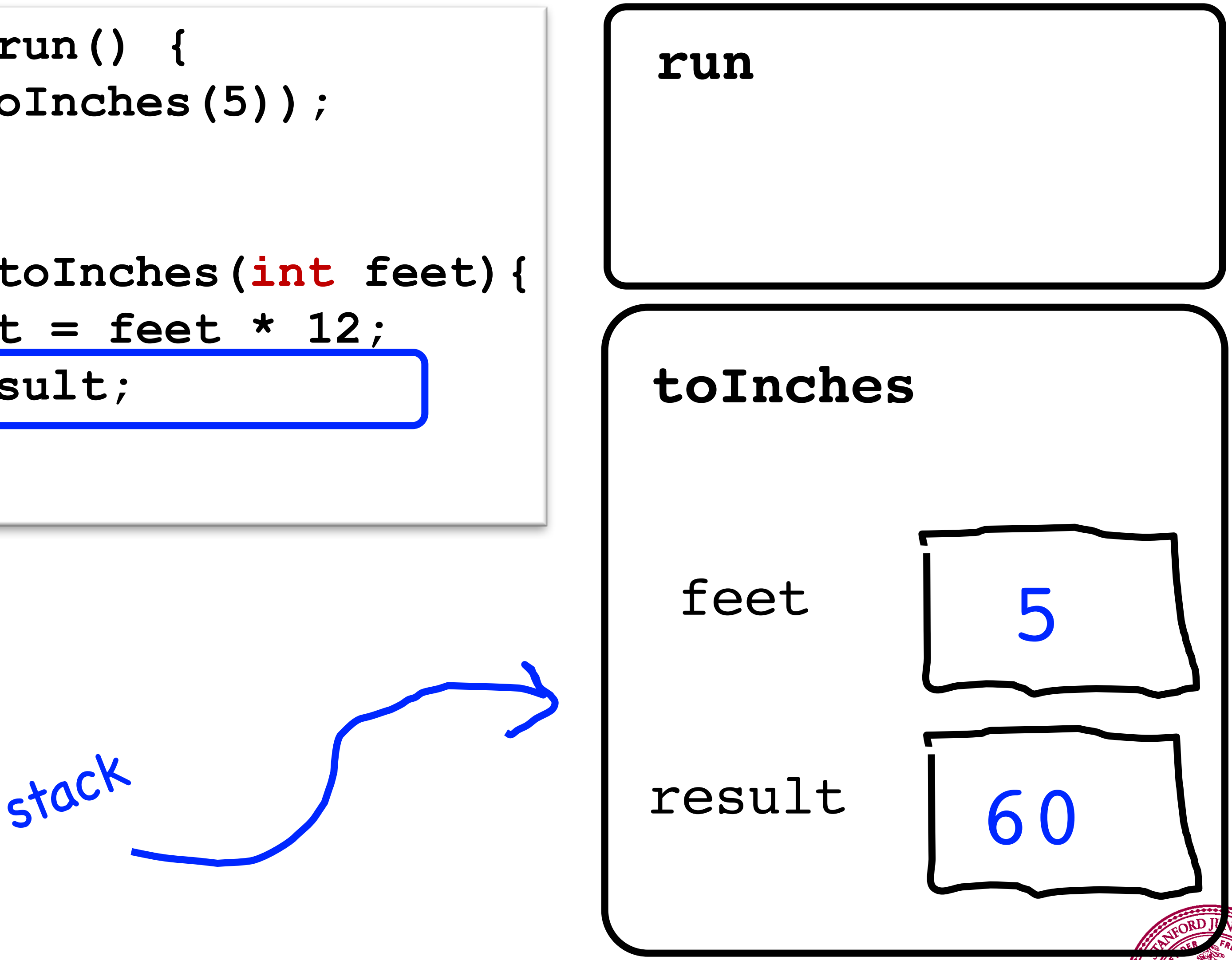
Stack Diagrams

```
public void run() {
   println(toInches(5));
}

private int toInches(int feet){
   int result = feet * 12;
   return result;
}
f
```



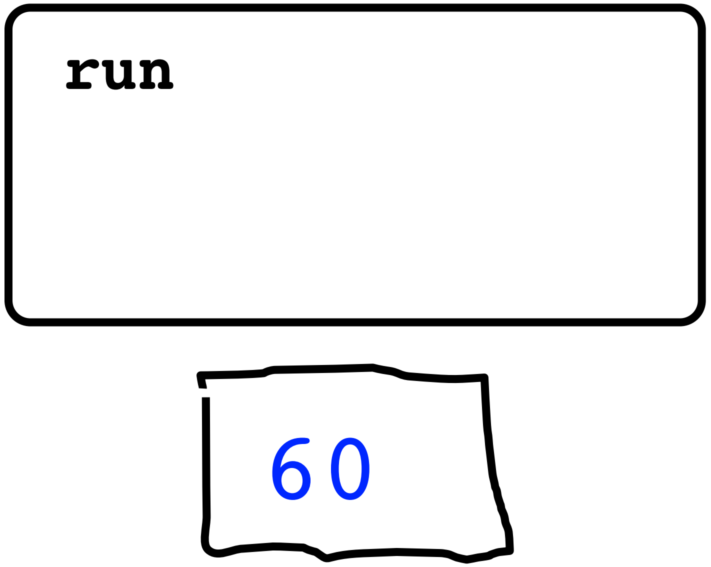
Stack Diagrams

```
public void run() {
   println(toInches(5));
}

private int toInches(int feet){
   int result = feet * 12;
   return result;
}
```



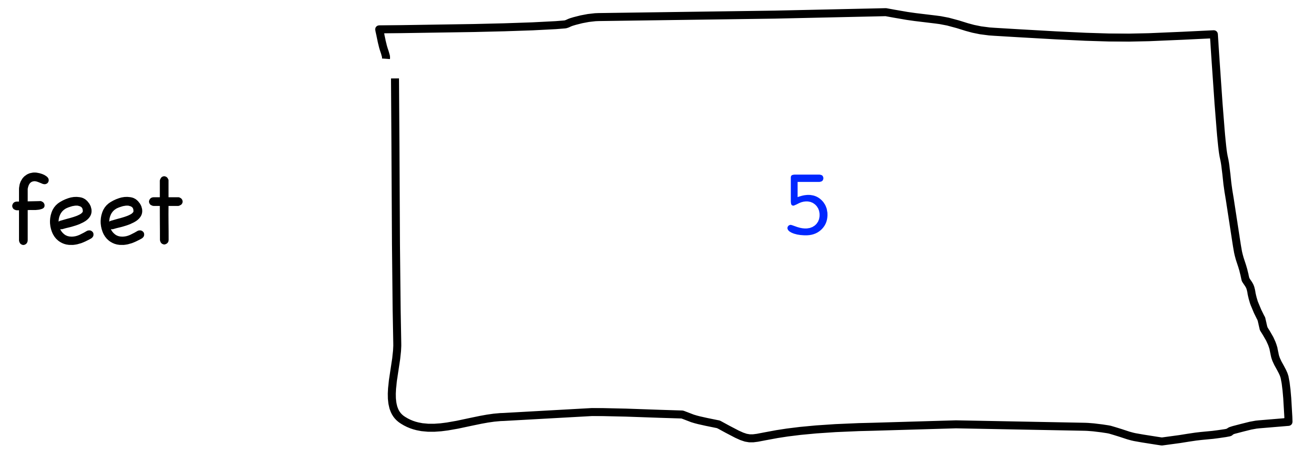
Stack Diagrams

```
public void run() {
    println(toInches(5));
}

private int toInches(int feet){
    int result = feet * 12;
    return result;
}
f
```

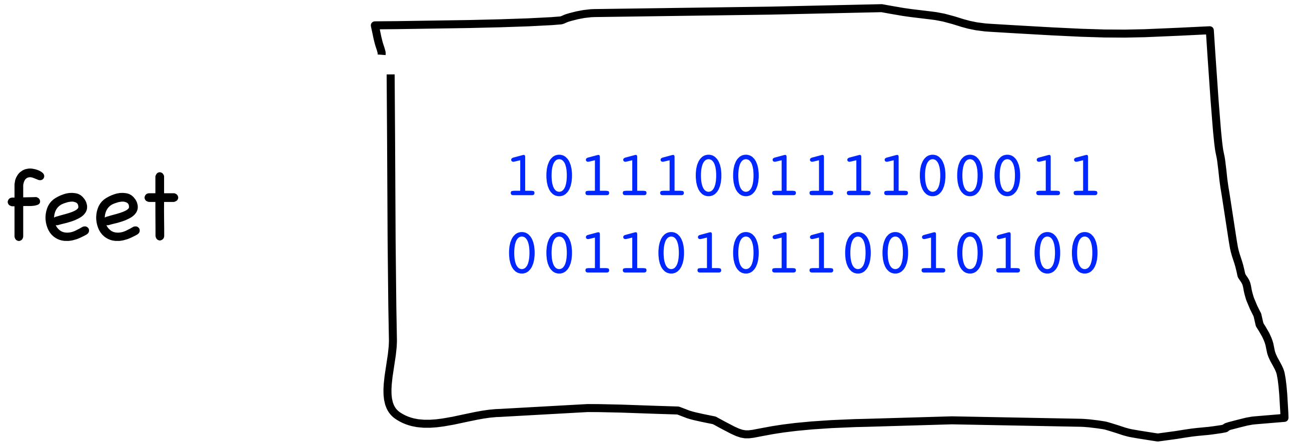


Aside: Actual Memory

What is a bucket



What is a bucket

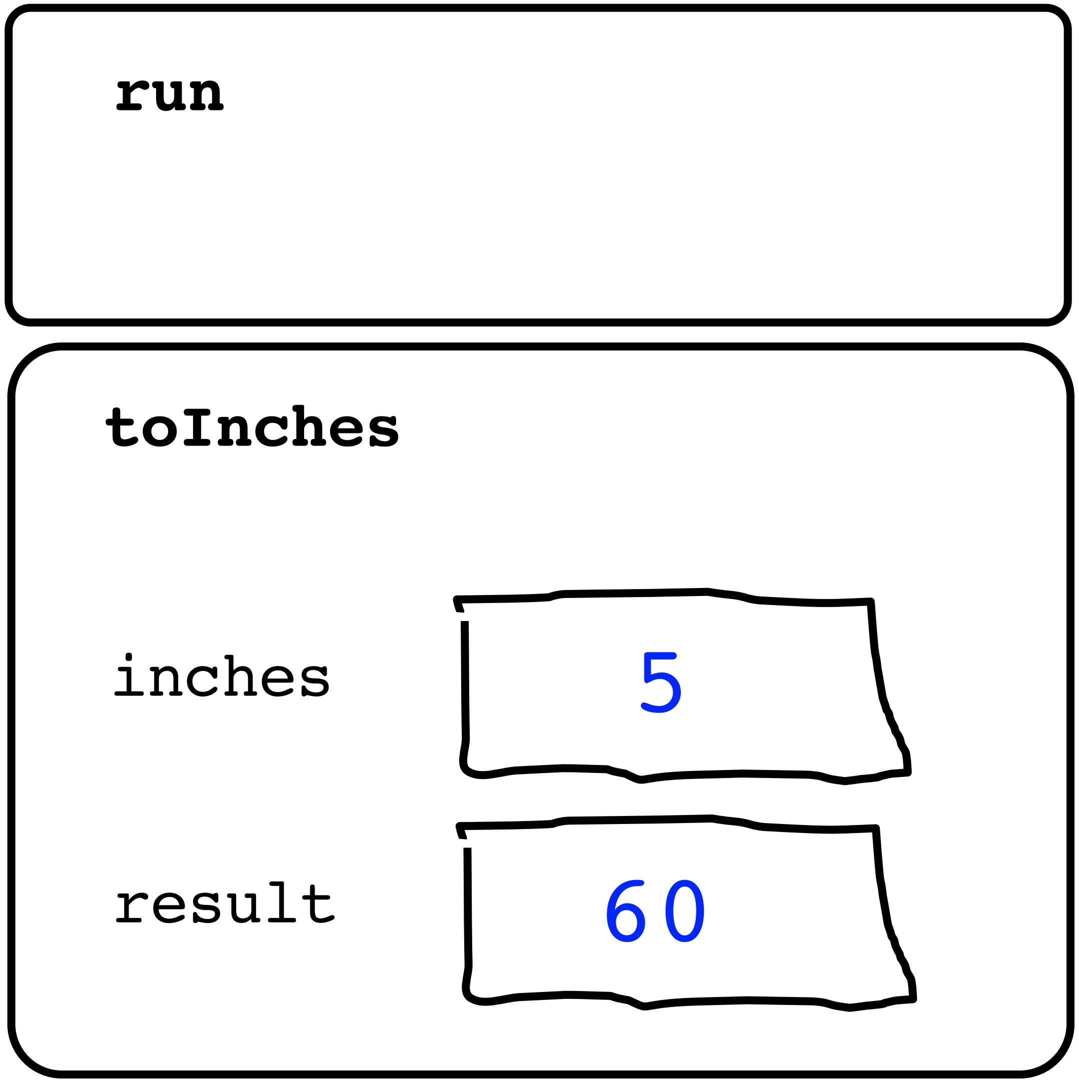


* Each bucket or "word" holds 64 bits

What does memory look like?

```
public void run() {
    println(toInches(5));
}

private int toInches(int feet){
    int result = feet * 12;
    return result;
}
```

Stack Diagrams



Actual Memory

run overhead	0011010011010001 1010110011010111
toInches overhead	1111100000111100 0000111100001111
feet	1011100111100011 0011010110010100
result	0101101110111101 1111011111101111
	?
	?

#0: don’t think on the binary level (yet)

Primitives vs Classes

Primitive Variable Types	Class Variable Types
`int`	`GRect`
`double`	`GOval`
`char`	`Gline`
`boolean`	`Color`

Class variables (aka objects)

1. Have upper camel case types
2. You can call methods on them
3. Are constructed using **`new`**
4. Are stored in a special way

Objects

```
GRect myRect = new GRect(20, 20);
```

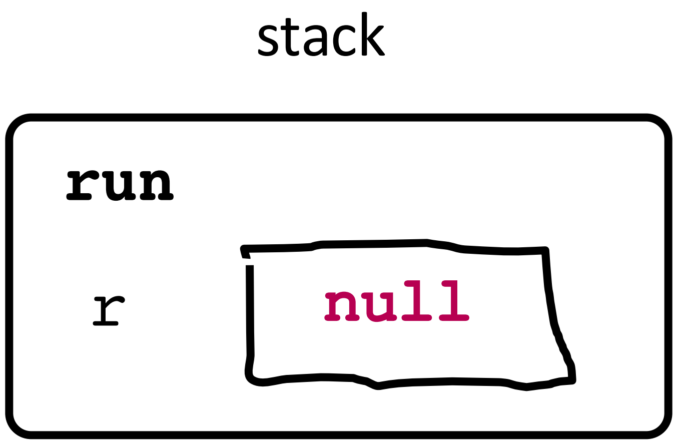
Object

The Heap

```
public void run() {
    GRect r = null;
}
```

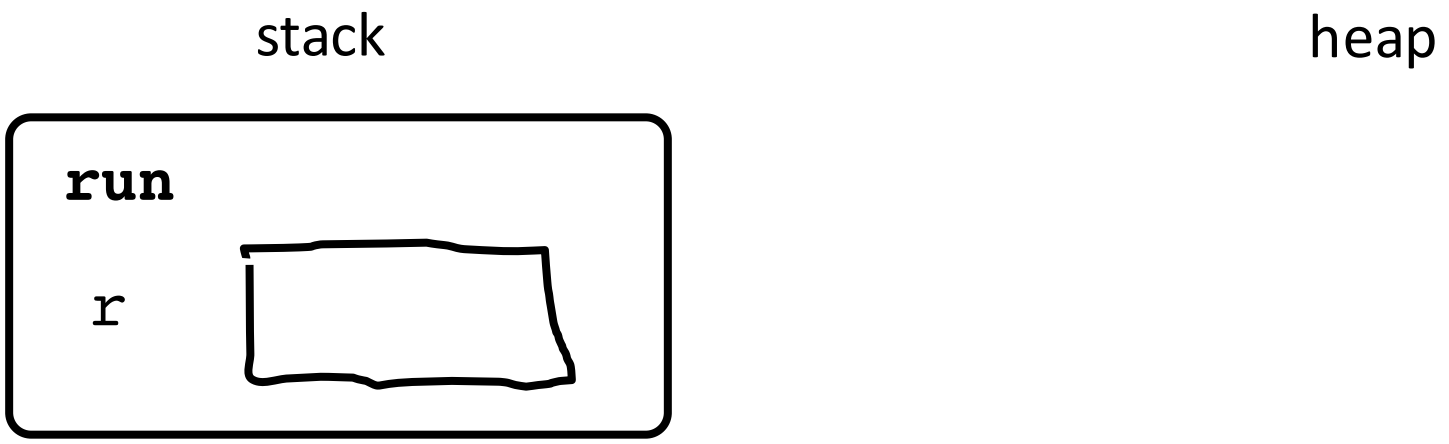
stack

```
public void run() {
    GRect r = null;
}
```

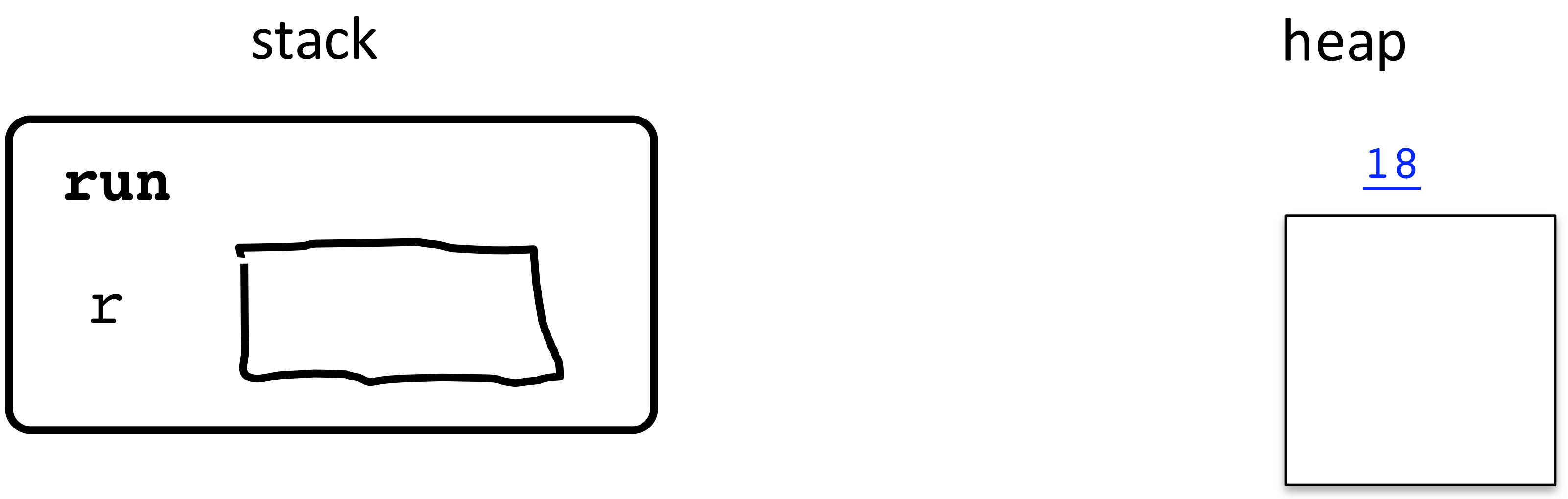


Wahoo!

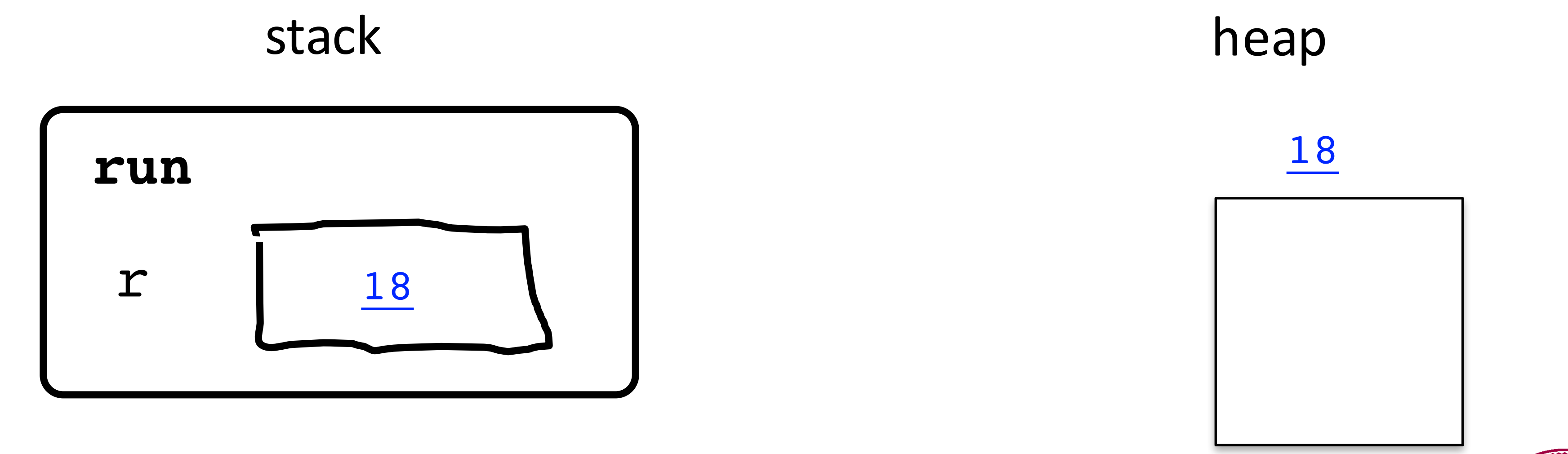
```
public void run() {
    GRect r = new GRect(50, 50);
}
```



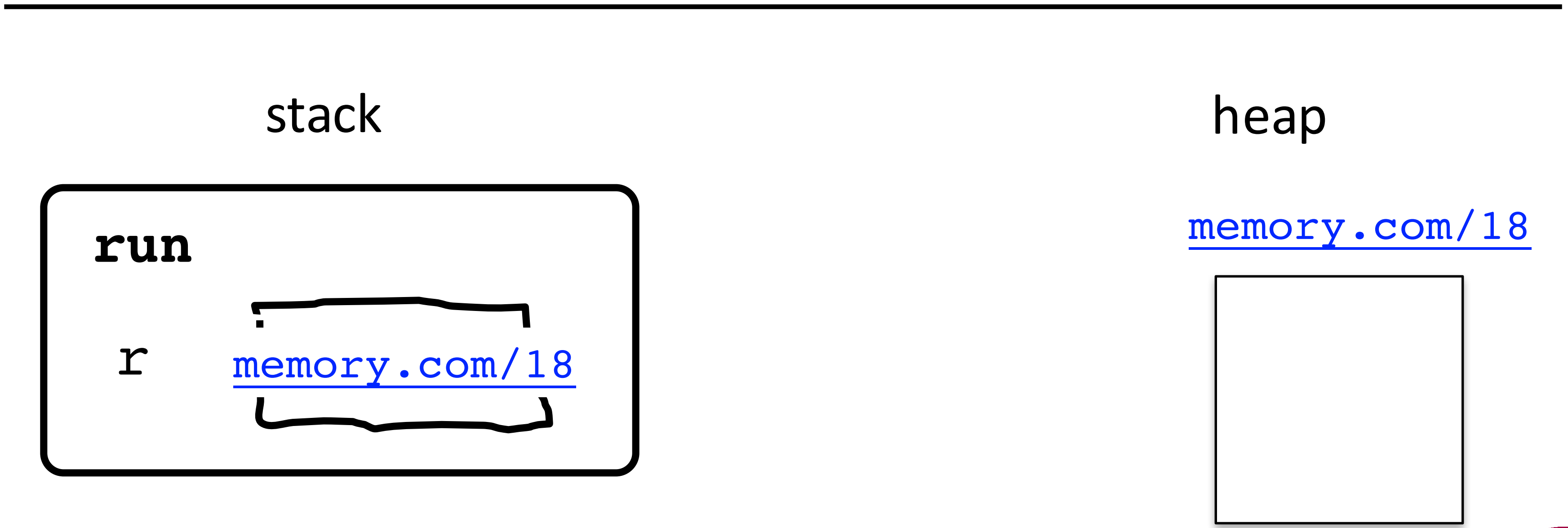
```
public void run() {
    GRect r = new GRect(50, 50);
}
```



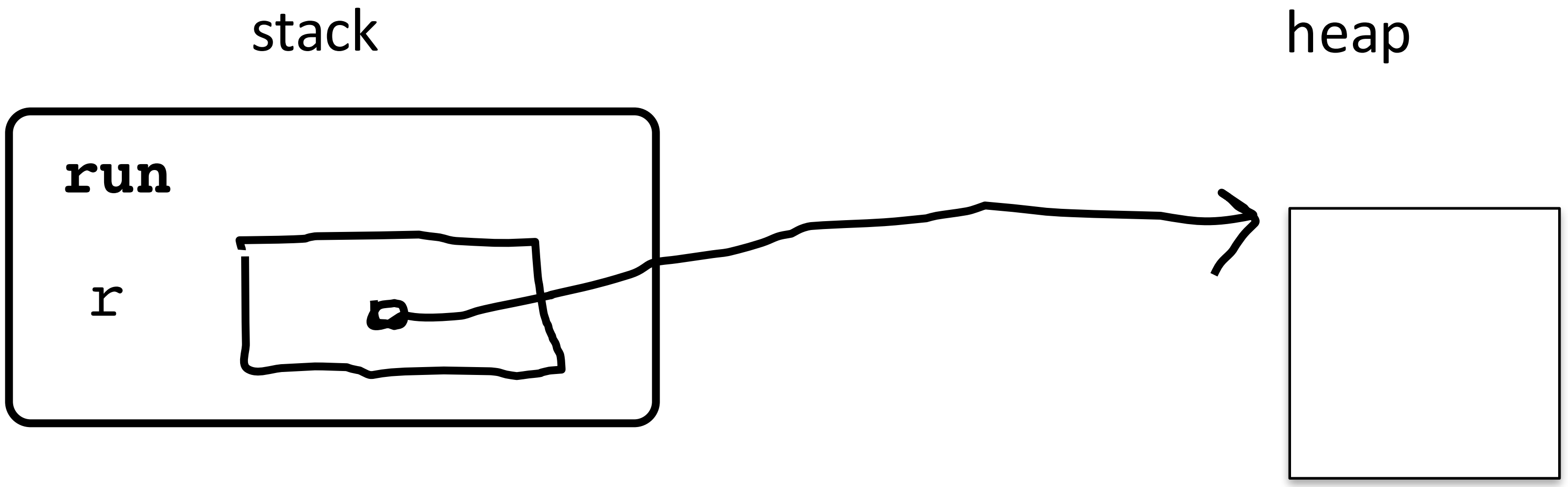
```
public void run() {
    GRect r = new GRect(50, 50);
}
```



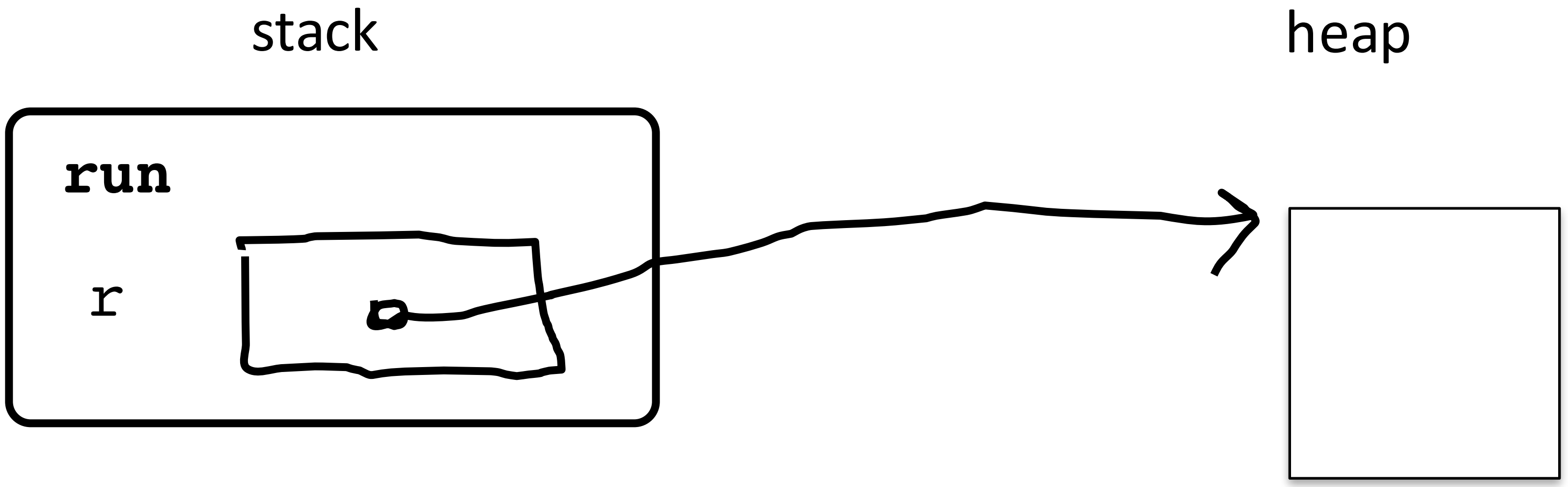
```
public void run() {
    GRect r = new GRect(50, 50);
}
```



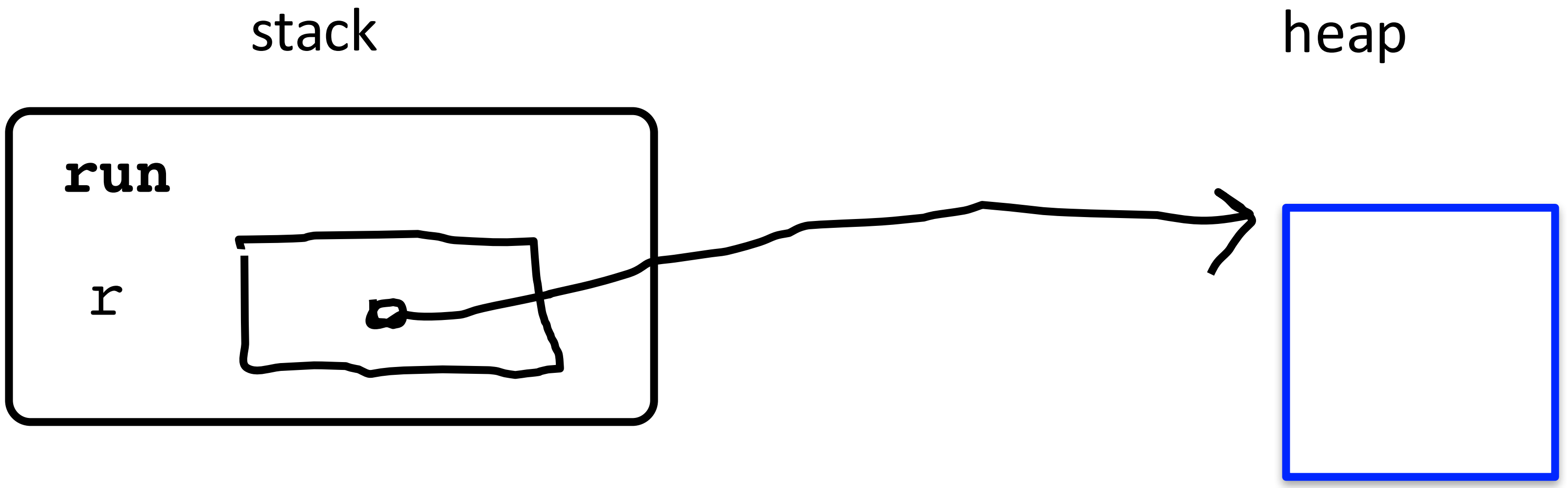
```
public void run() {
   GRect r = new GRect(50, 50);
}
```



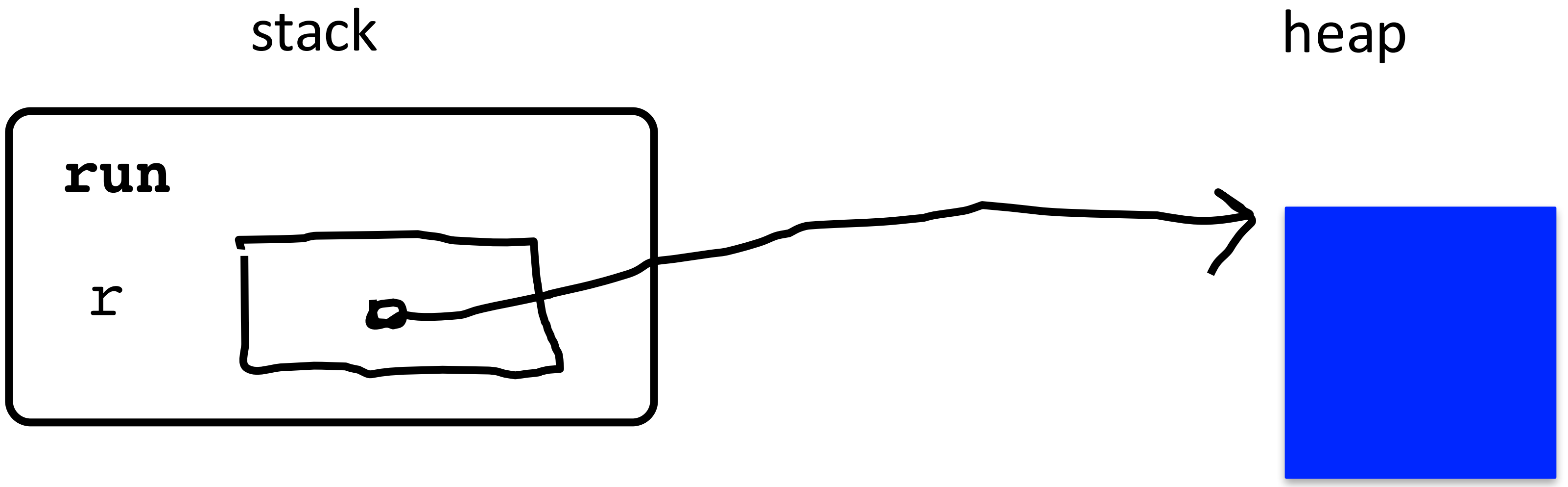
```
public void run() {
    GRect r = new GRect(50, 50);
    r.setColor(Color.BLUE);
    r.setFilled(true);
}
```



```
public void run() {
    GRect r = new GRect(50, 50);
    r.setColor(Color.BLUE);
    r.setFilled(true);
}
```



```
public void run() {
    GRect r = new GRect(50, 50);
    r.setColor(Color.BLUE);
    r.setFilled(true);
}
```



#1: `new` allocates memory on the heap

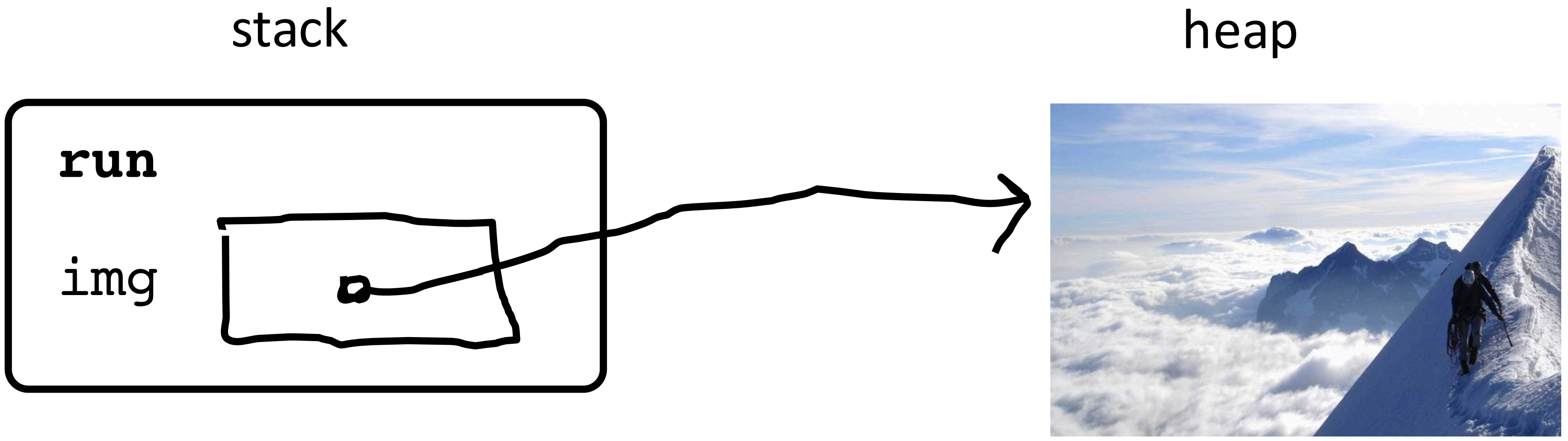
#2: object variables store heap addresses

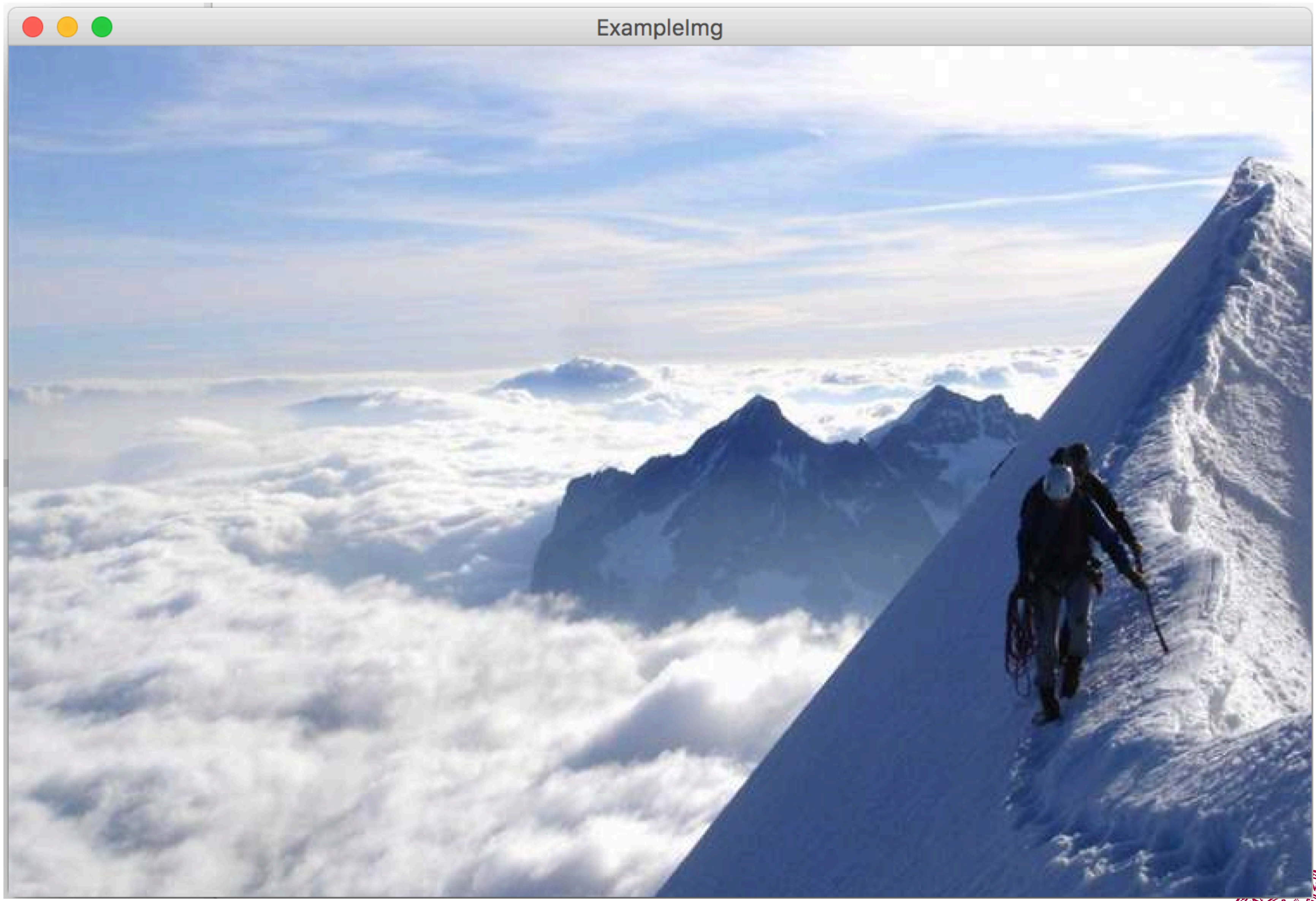
#ultimatekey

What does an object store?

An object stores a memory address!

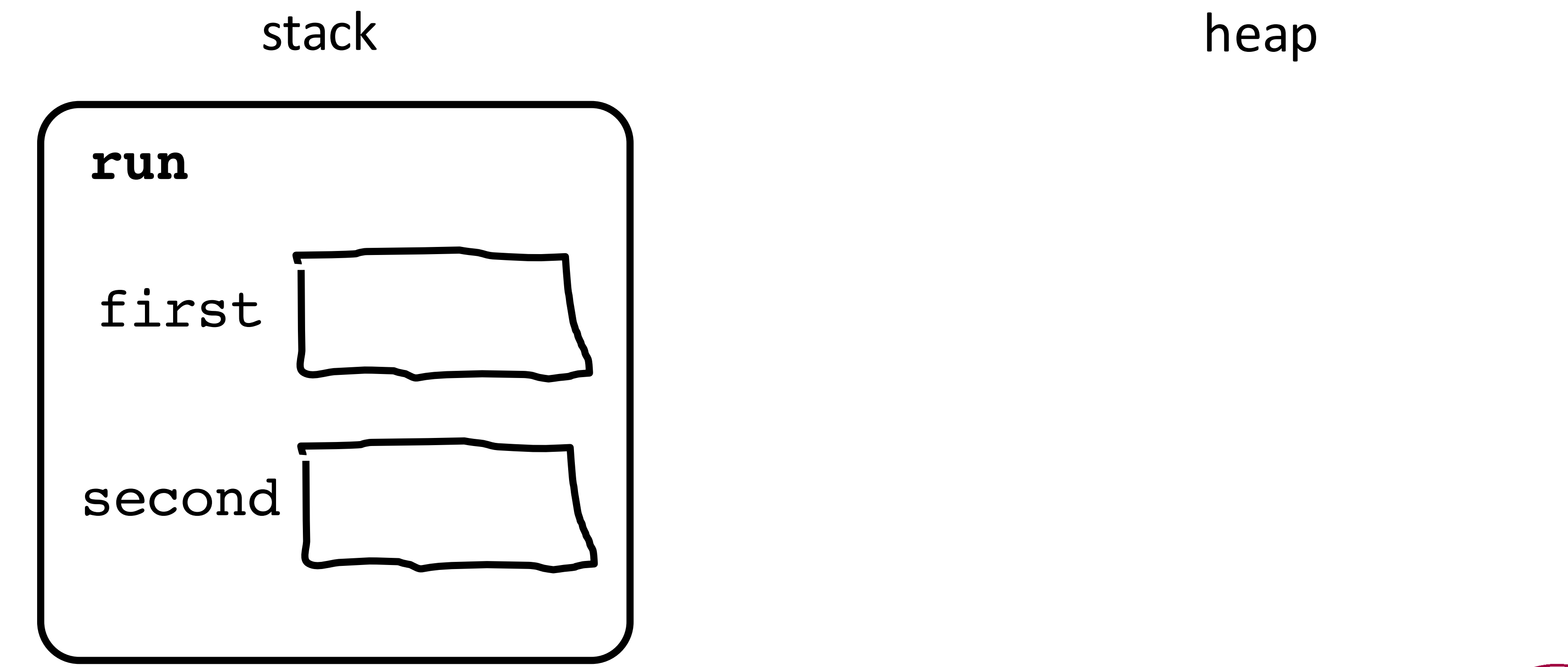
```
public void run() {
    GImage img = new GImage("mountain.jpg");
    add(img, 0, 0);
}
```



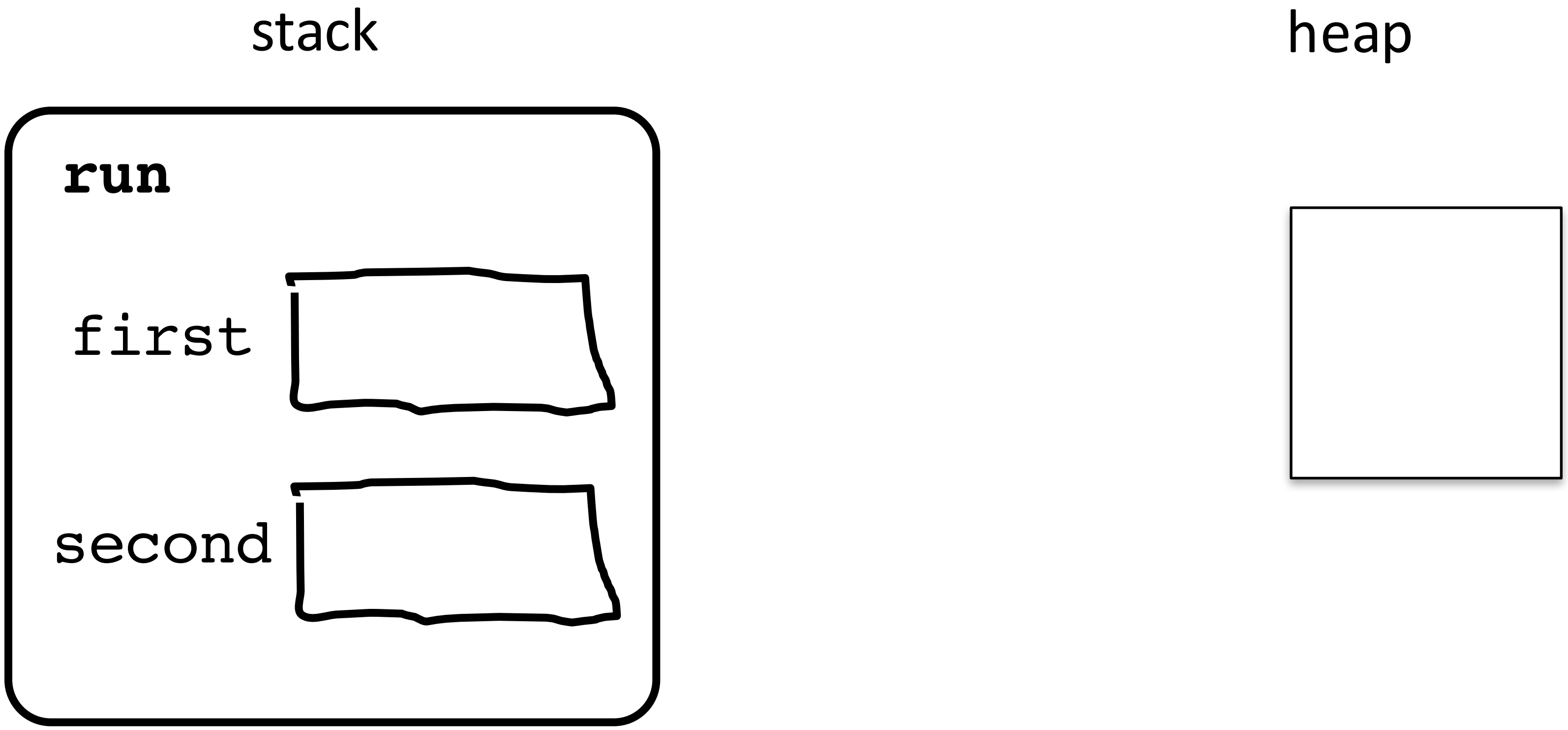

ExampleImg

#3: `GImage`s look impressive but don't take much extra work

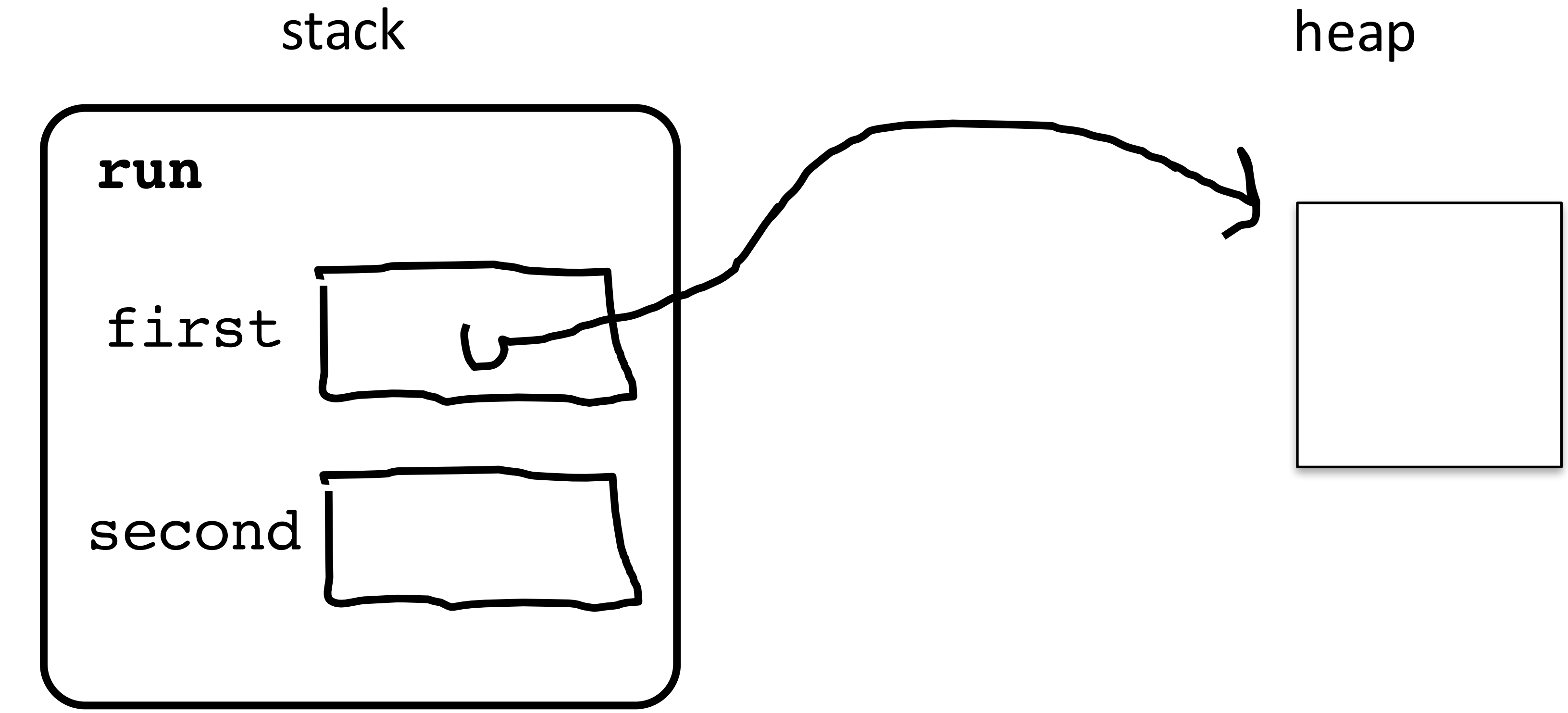
```
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}
```



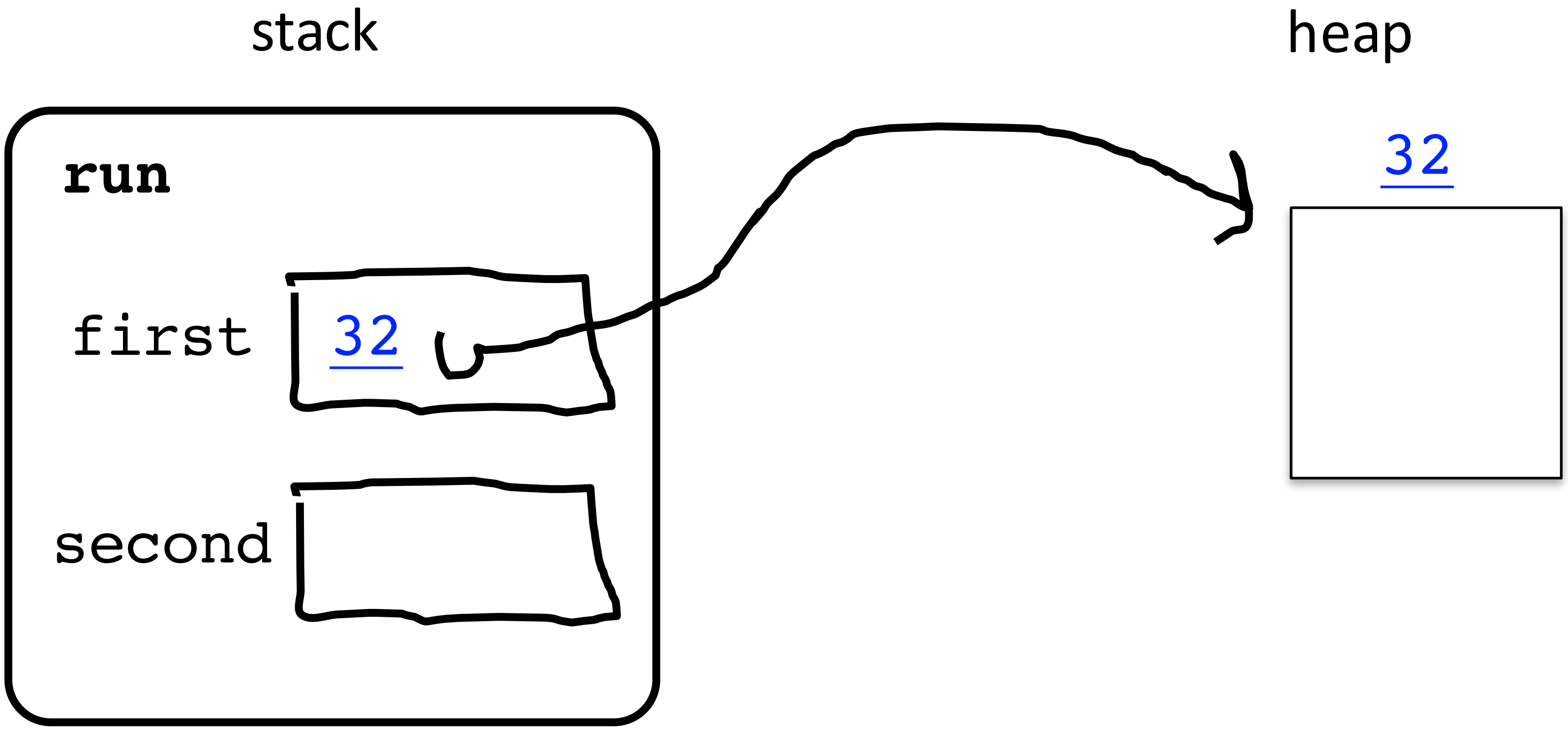
```
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}
```



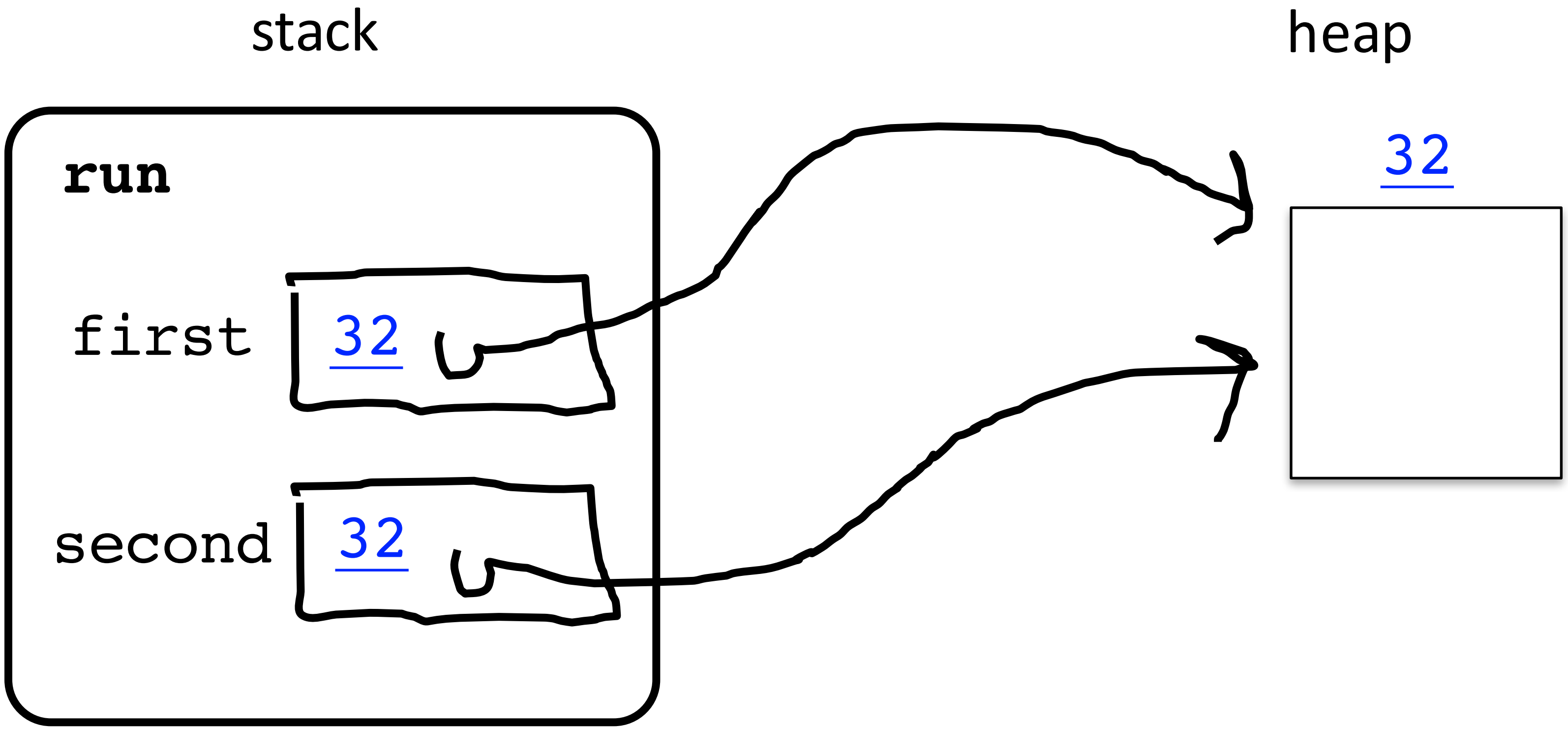
```
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}
```



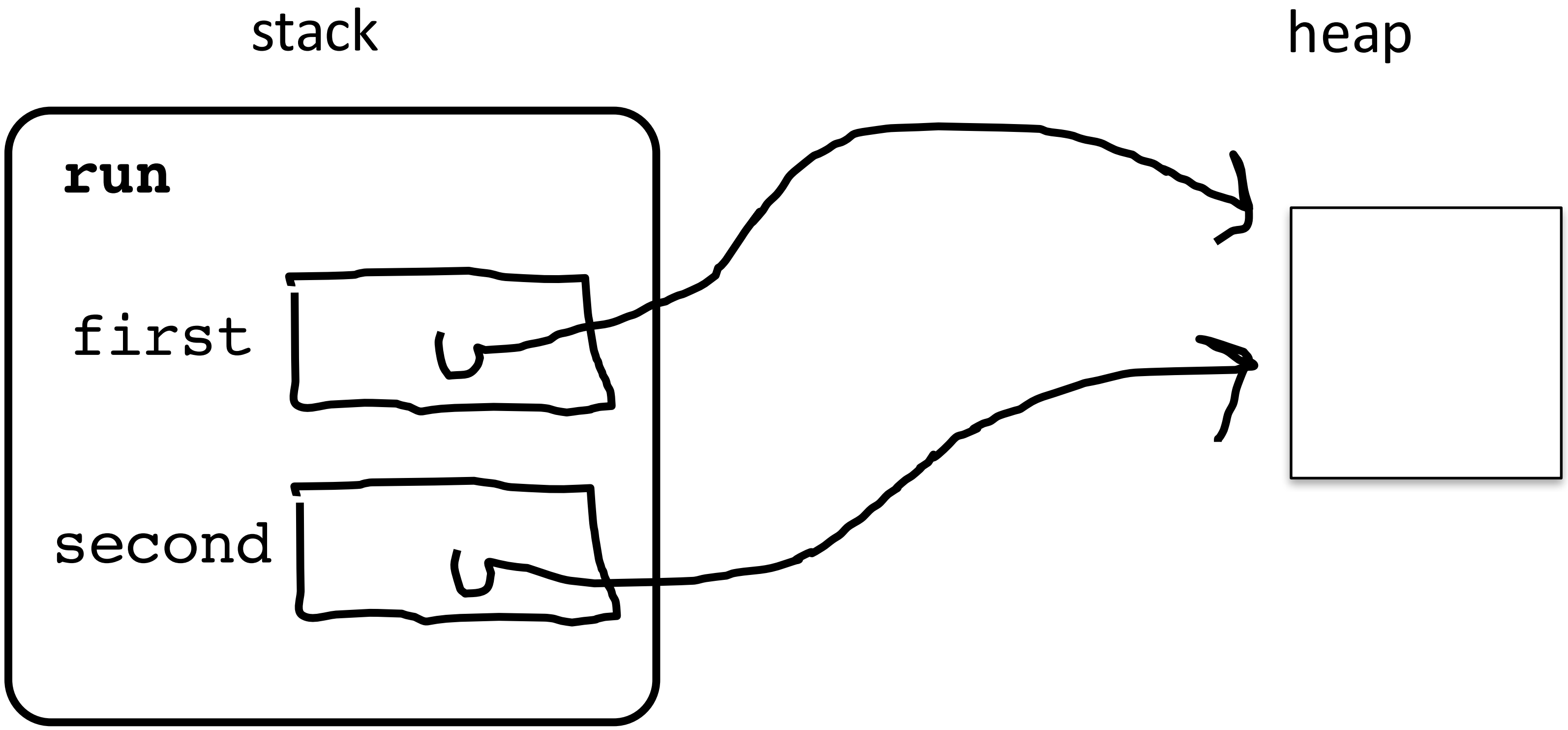
```
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}
```



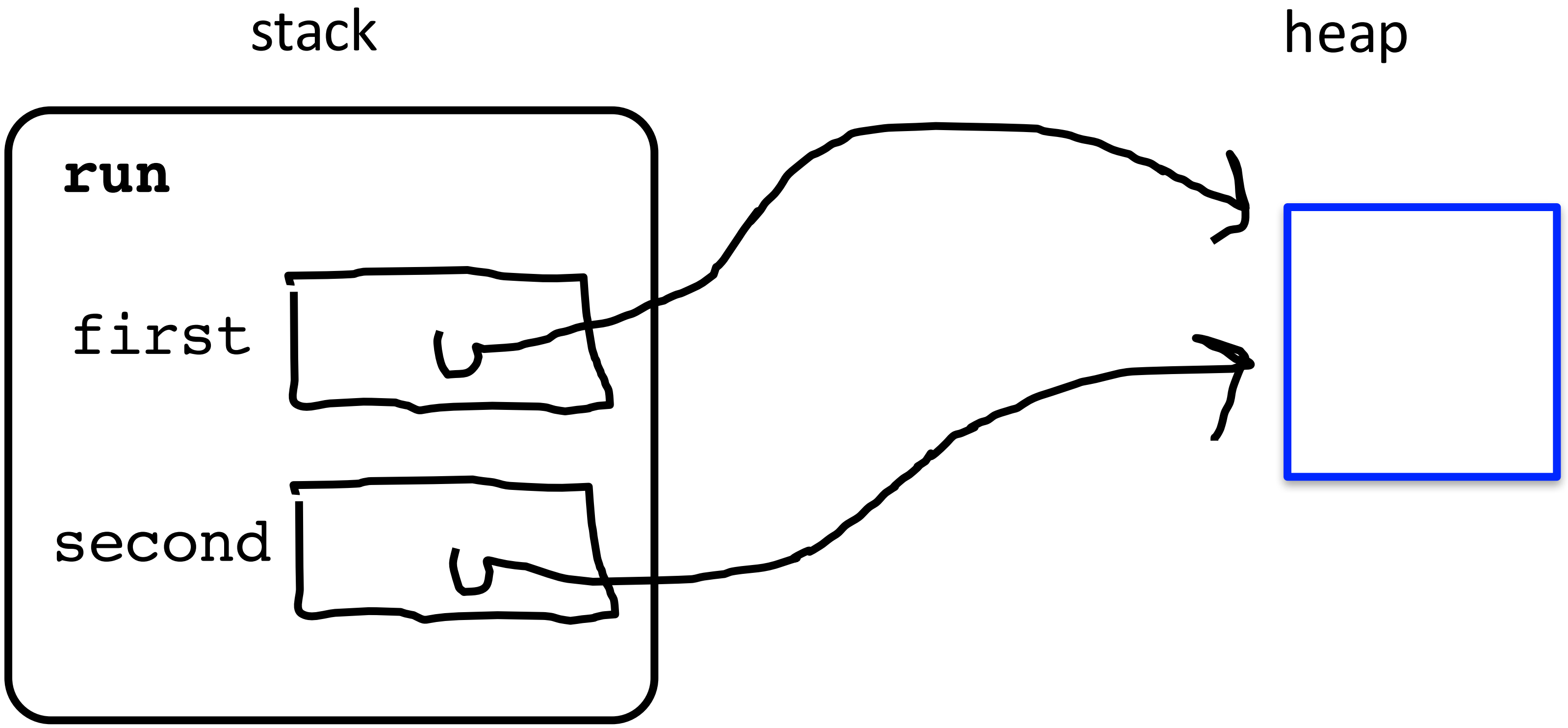
```
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}
```



```
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}
```



```
public void run() {
    GRect first = new GRect(20, 20);
    GRect second = first;
    second.setColor(Color.BLUE);
    add(first, 0, 0);
}
```



#4: when you use the = operator with objects, it copies the *address*

Passing by “Reference”

Primitives pass by value

```
// NOTE: This program is buggy!!

public void run() {
   int x = 3;
   addFive(x);
   println("x = " + x);
}

private void addFive(int x) {
   x += 5;
}
```

* This is probably the single more important example to understand in CS106A

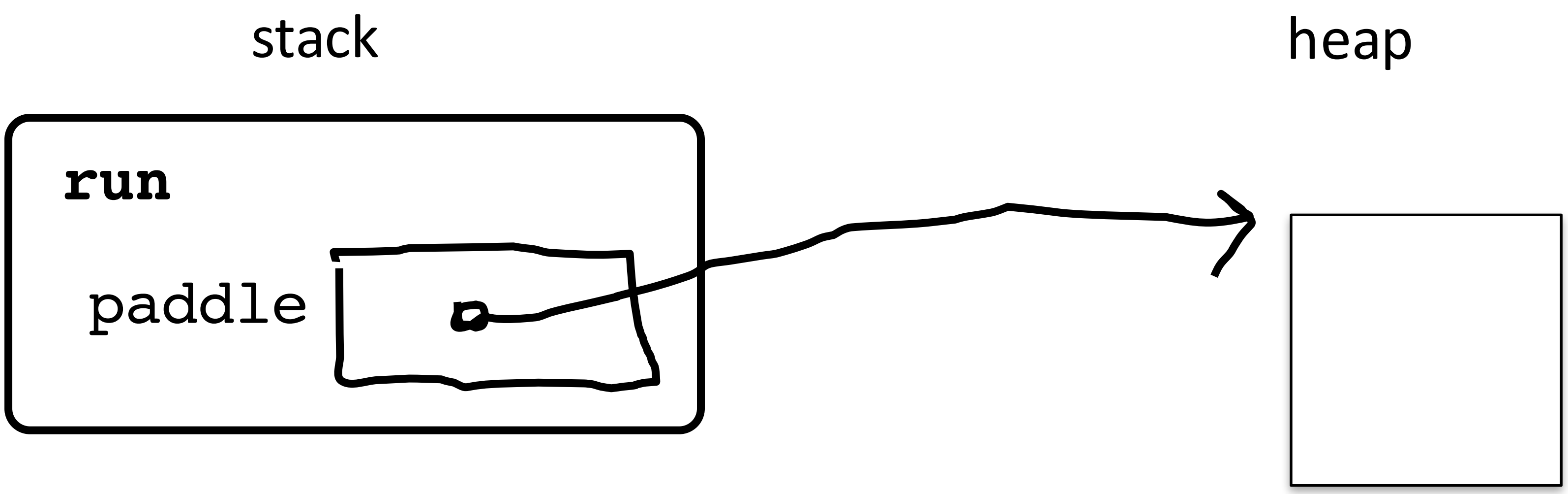
Objects pass by reference

```
// NOTE: This program is awesome!!

public void run() {
   GRect paddle = new GRect(50, 50);
   makeBlue(paddle);
   add(paddle, 0, 0);
}

private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```

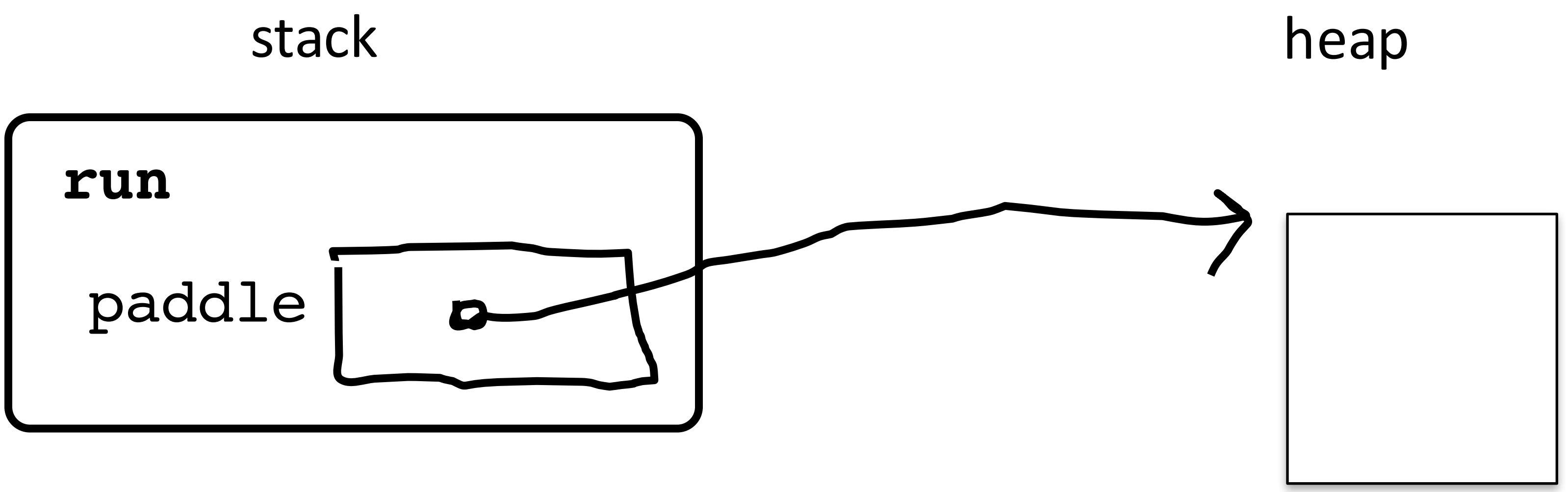
* This is probably the single more important example to understand in CS106A

```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}
private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```

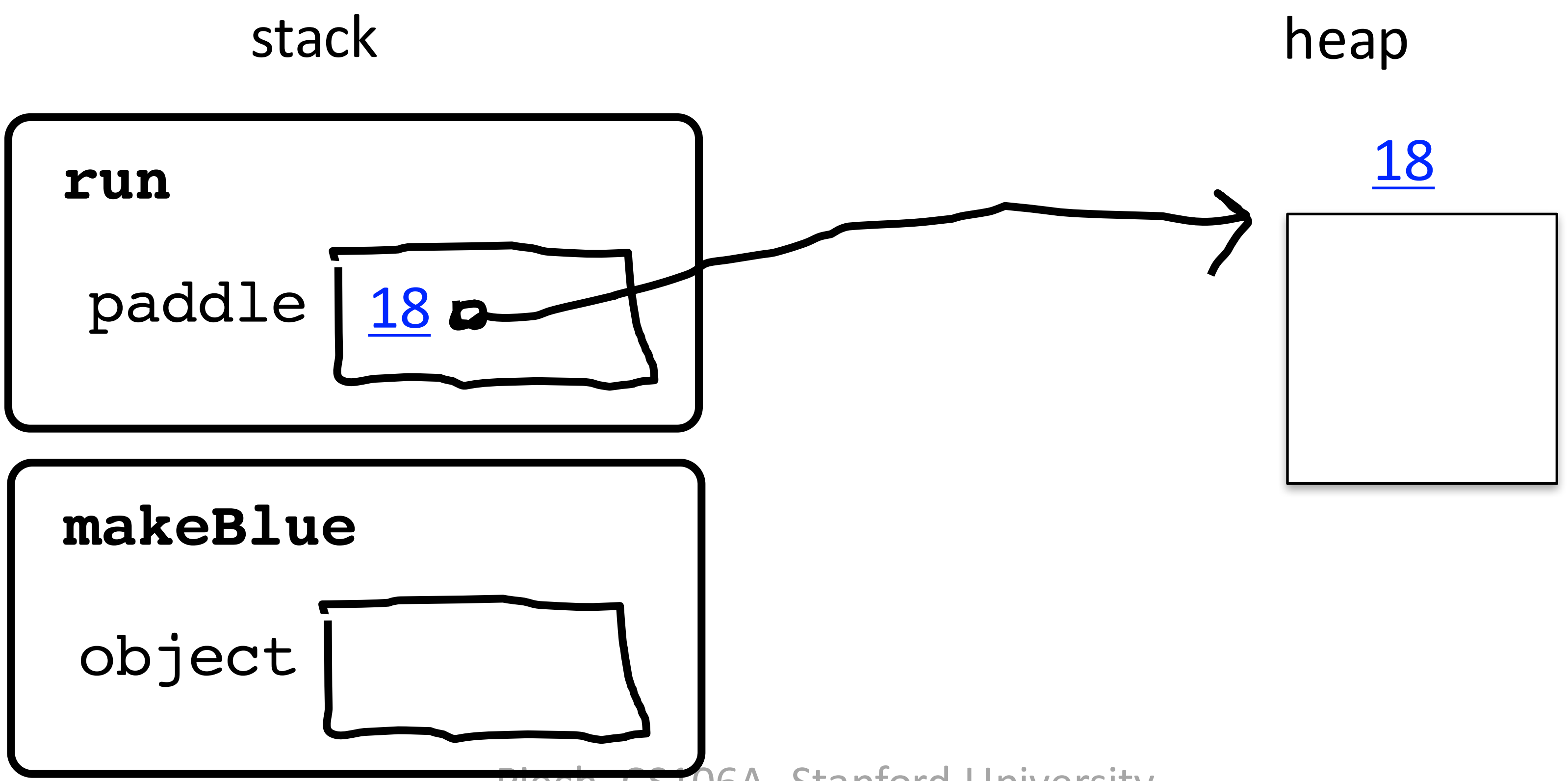
```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}
private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```



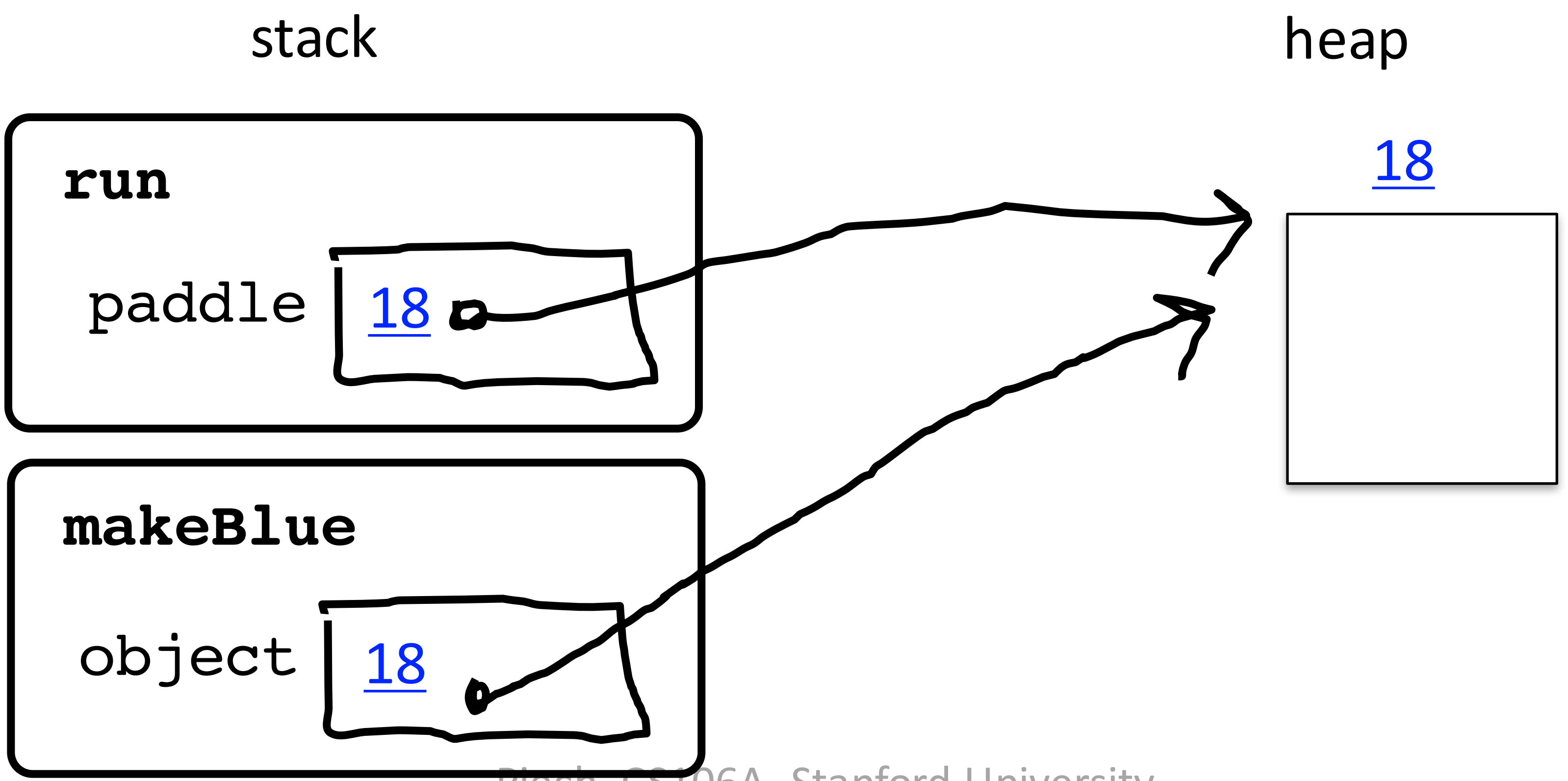
```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}
private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```



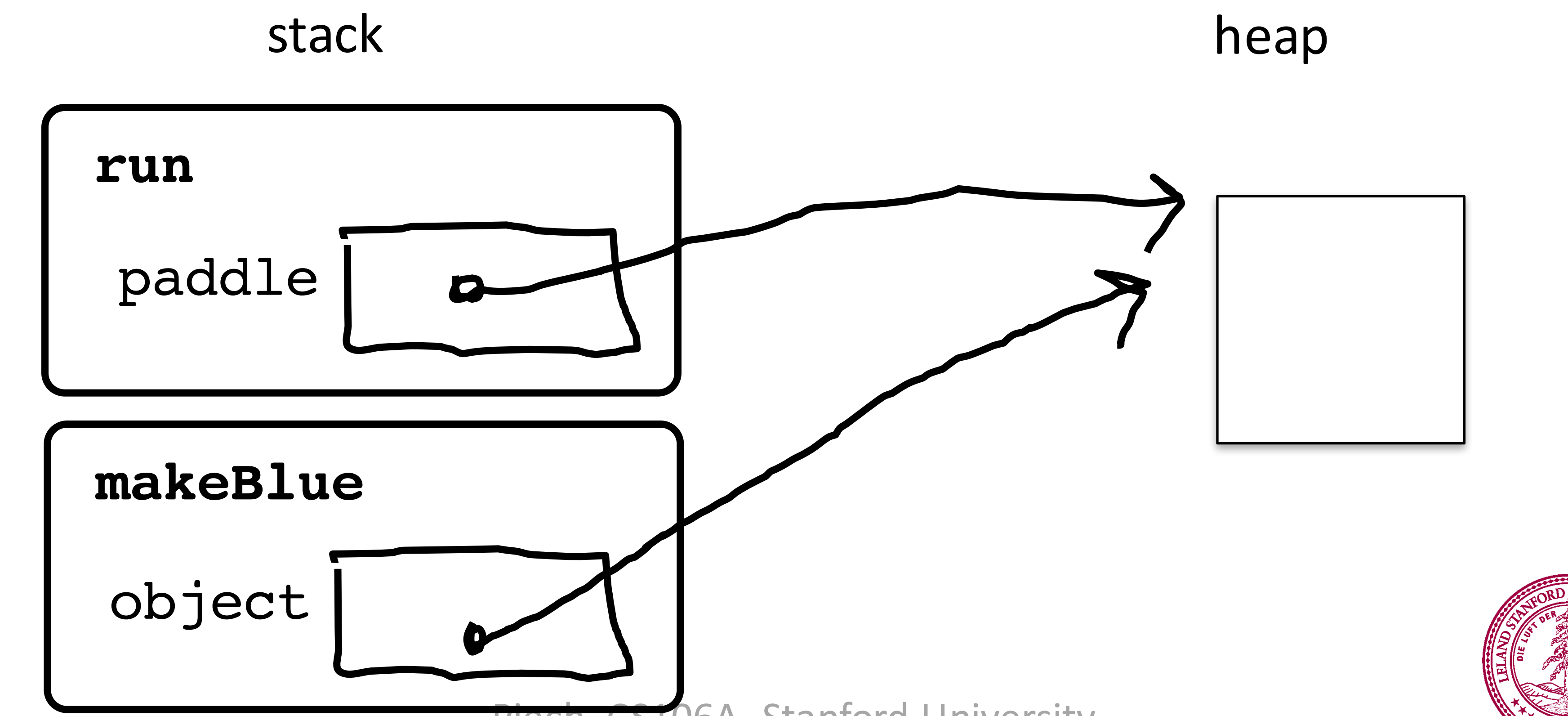
```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}
private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```



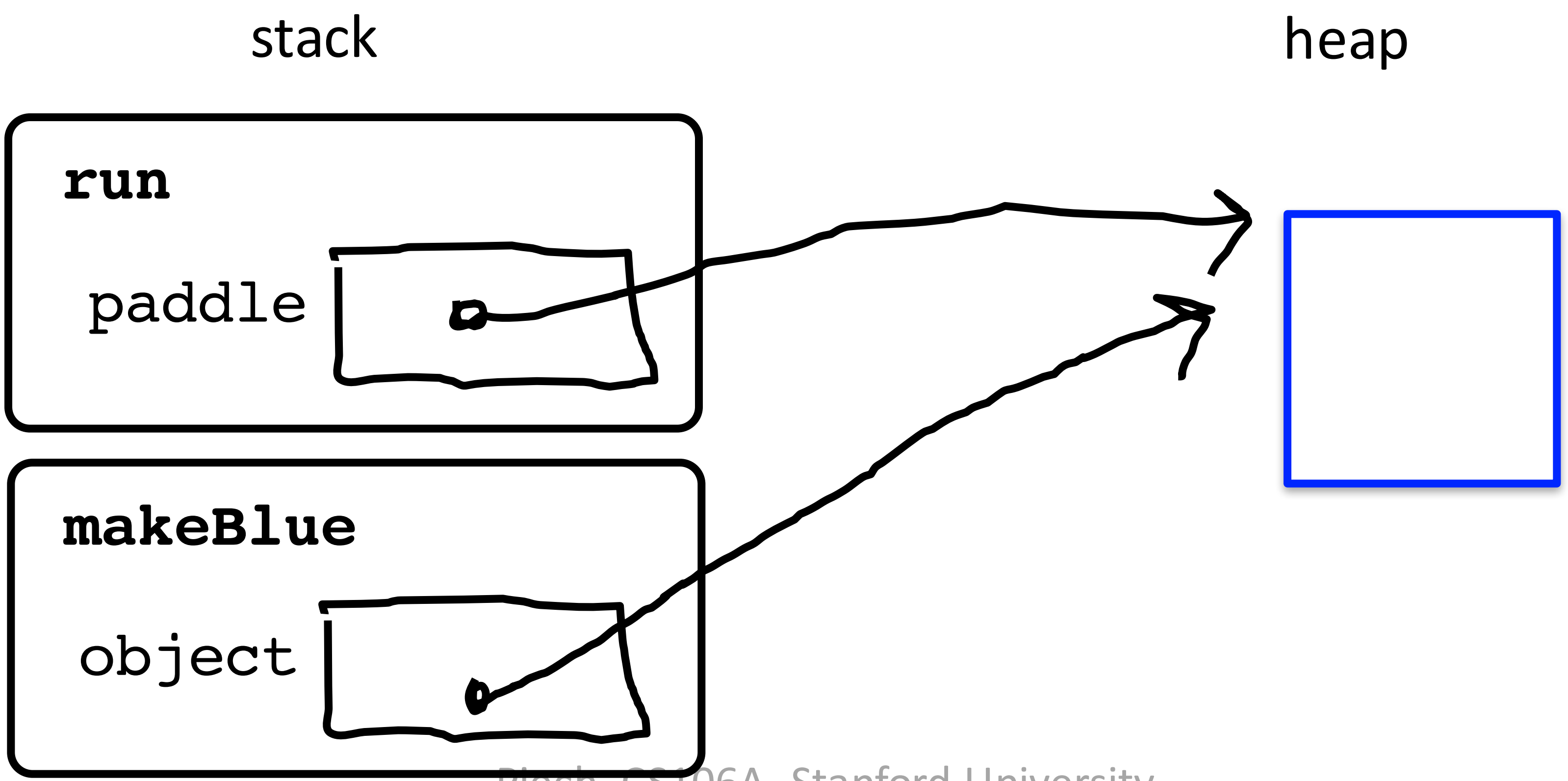
```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}
private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```



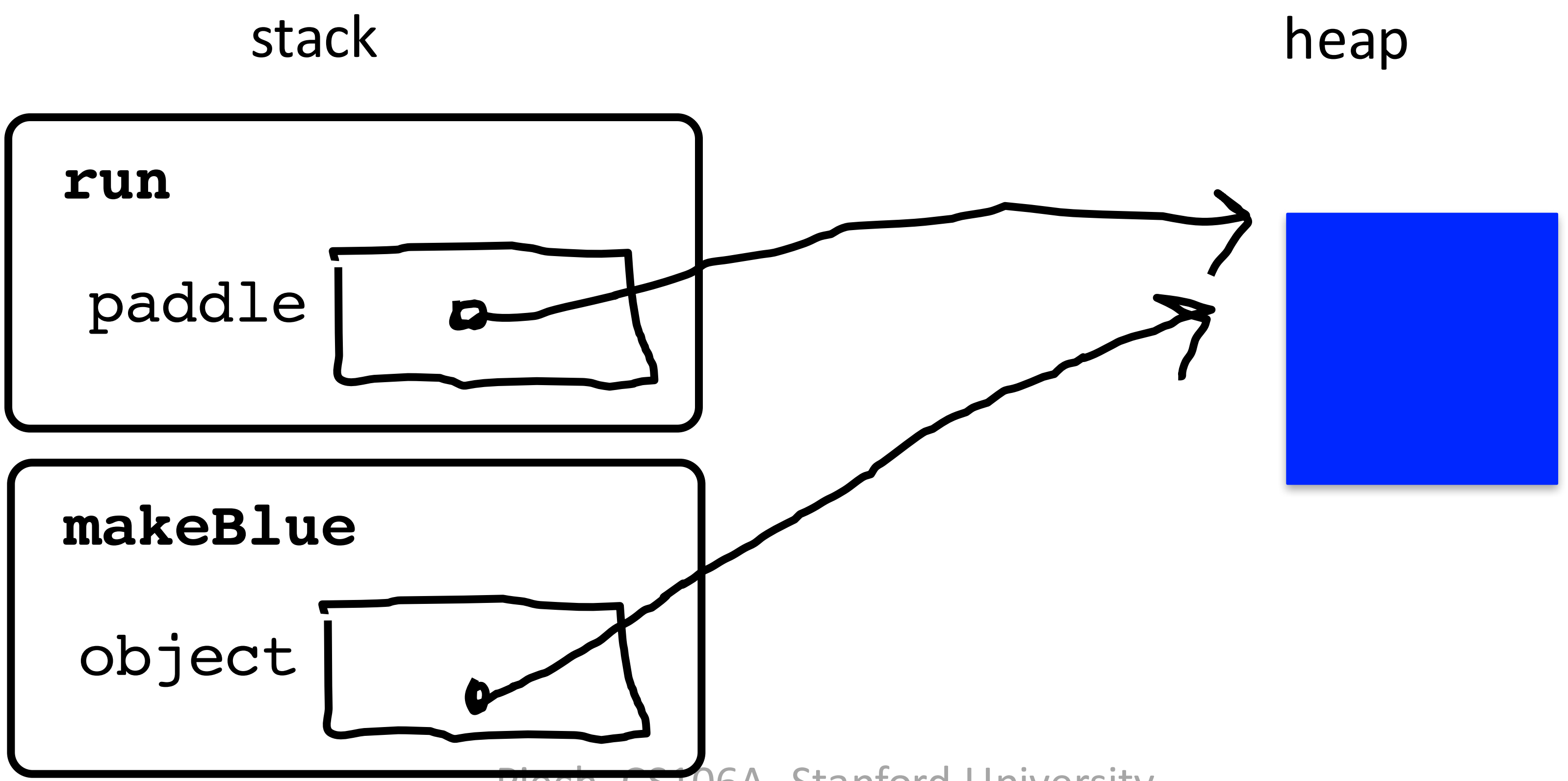
```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}
private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```



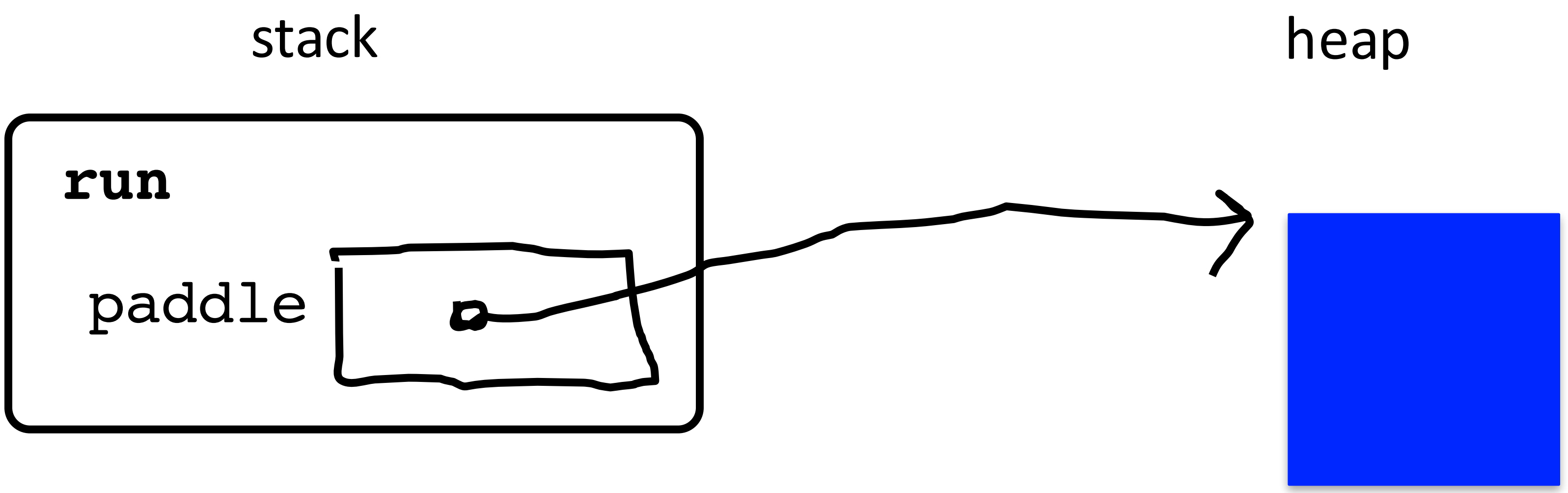
```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}
private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```



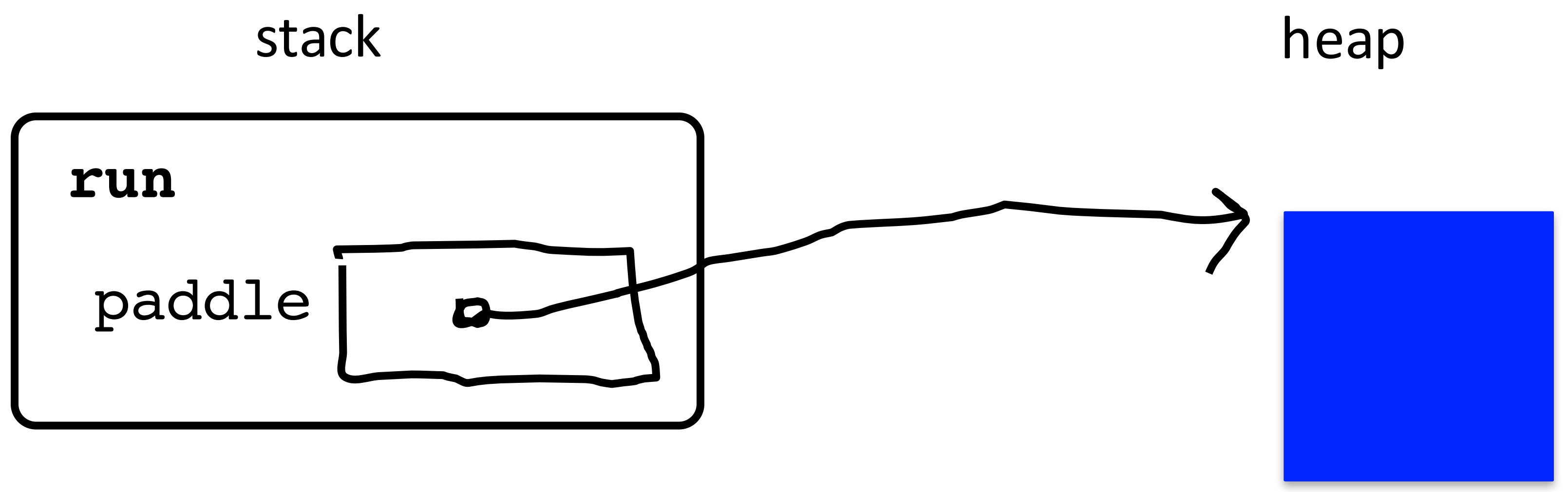
```
public void run() {
     GRect paddle = new GRect(50, 50);
     makeBlue(paddle);
     add(paddle, 0, 0);
}
private void makeBlue(GRect object) {
     object.setColor(Color.BLUE);
     object.setFilled(true);
}
```



```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}
private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```



```
public void run() {
    GRect paddle = new GRect(50, 50);
    makeBlue(paddle);
    add(paddle, 0, 0);
}
private void makeBlue(GRect object) {
    object.setColor(Color.BLUE);
    object.setFilled(true);
}
```



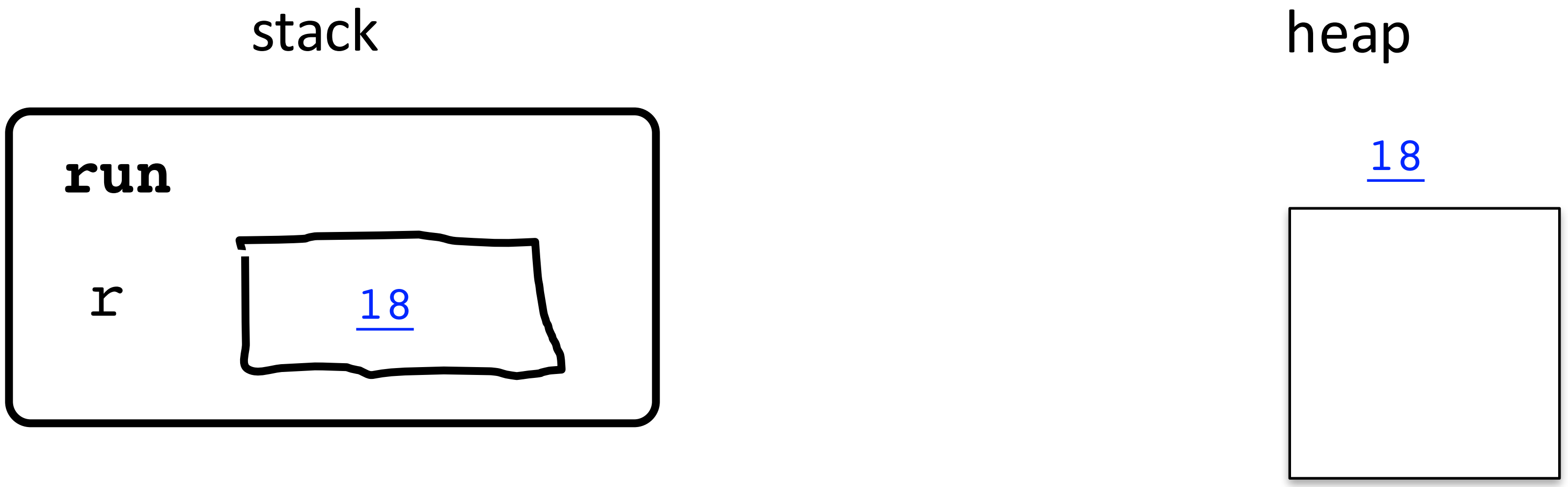
#5: when you pass (or return) an object, the address is passed.

Aka reference

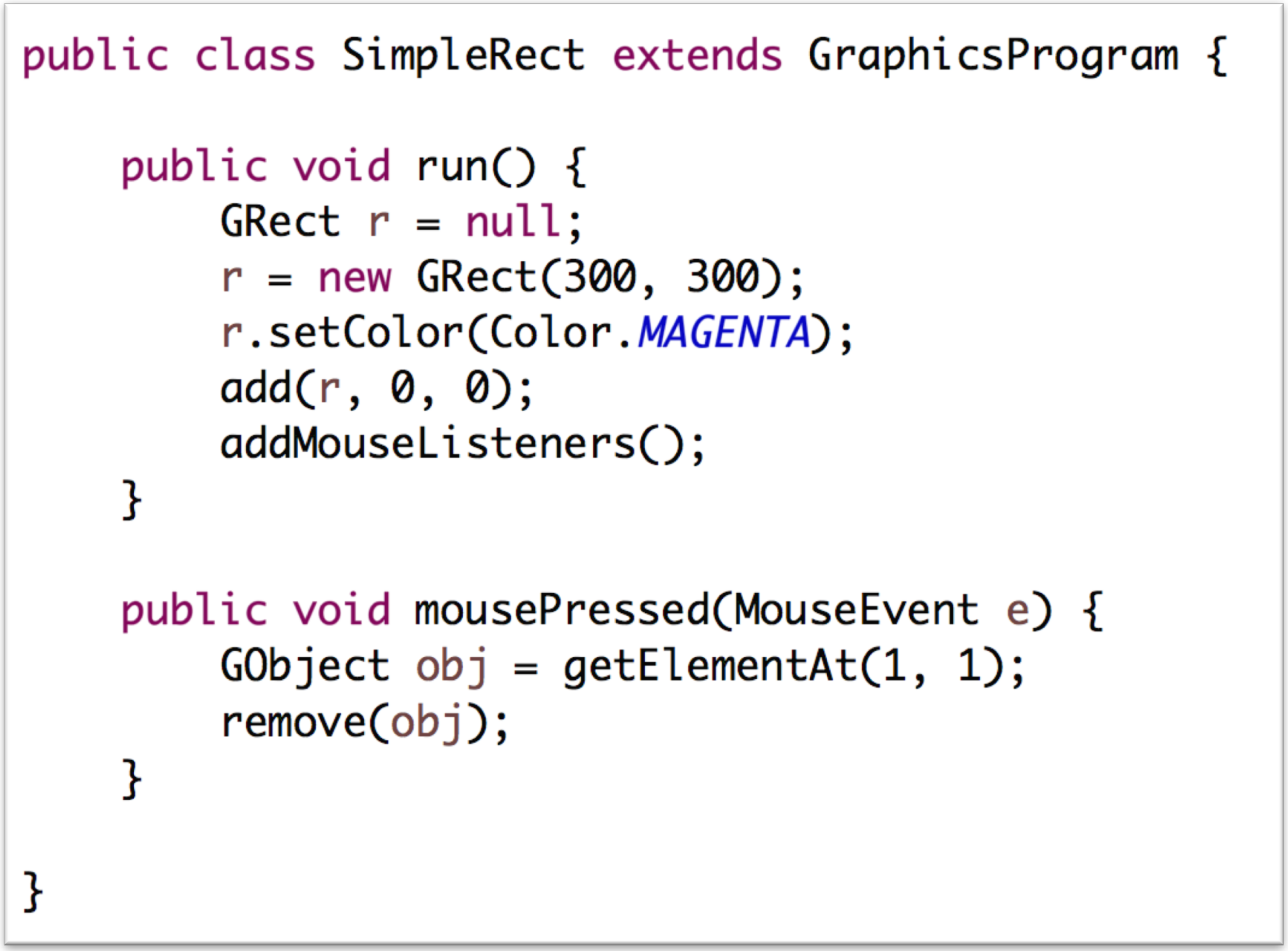
What does an object store?

An object stores a memory address!

```
public void run() {
    GRect r = new GRect(50, 50);
}
```

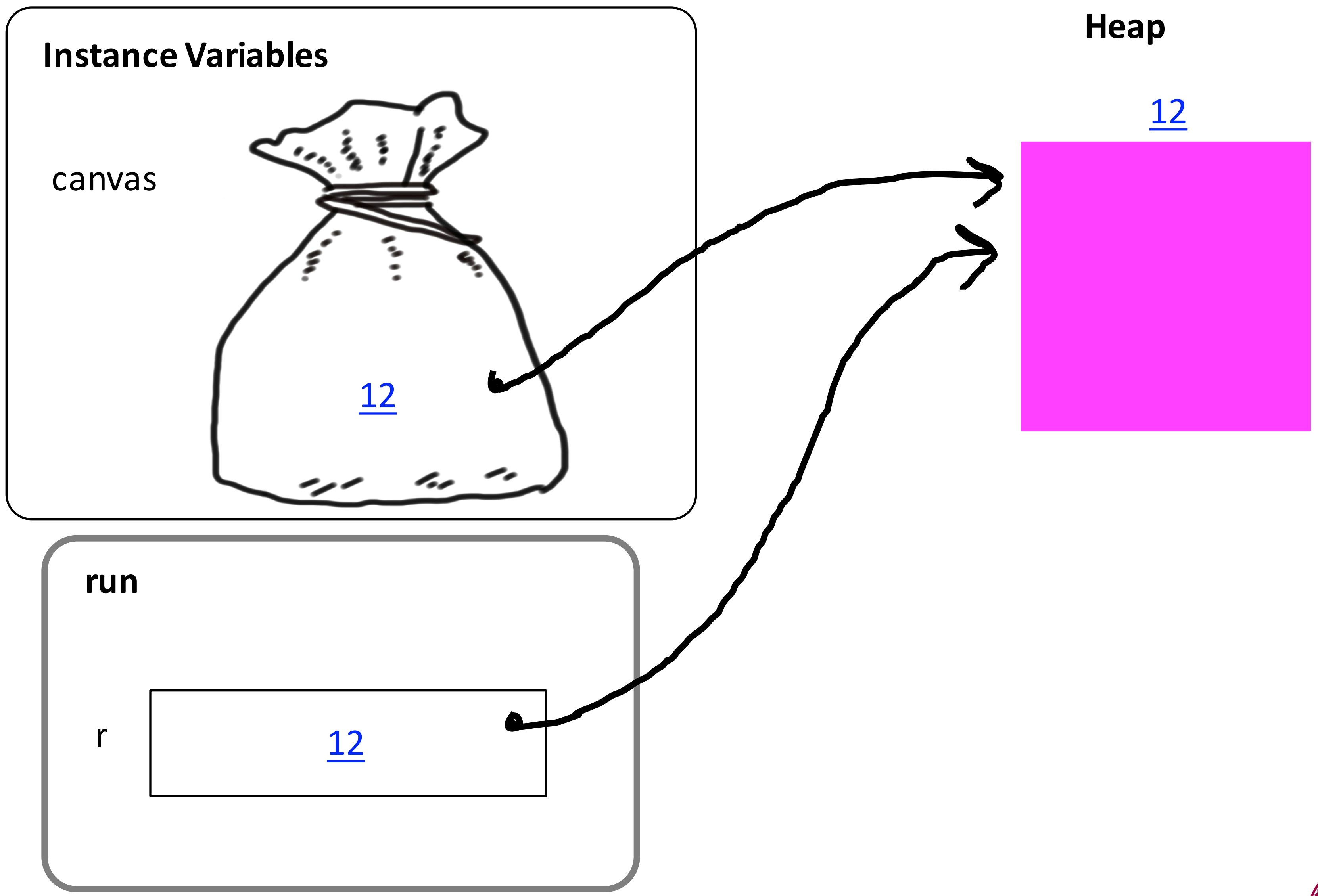


Canvas



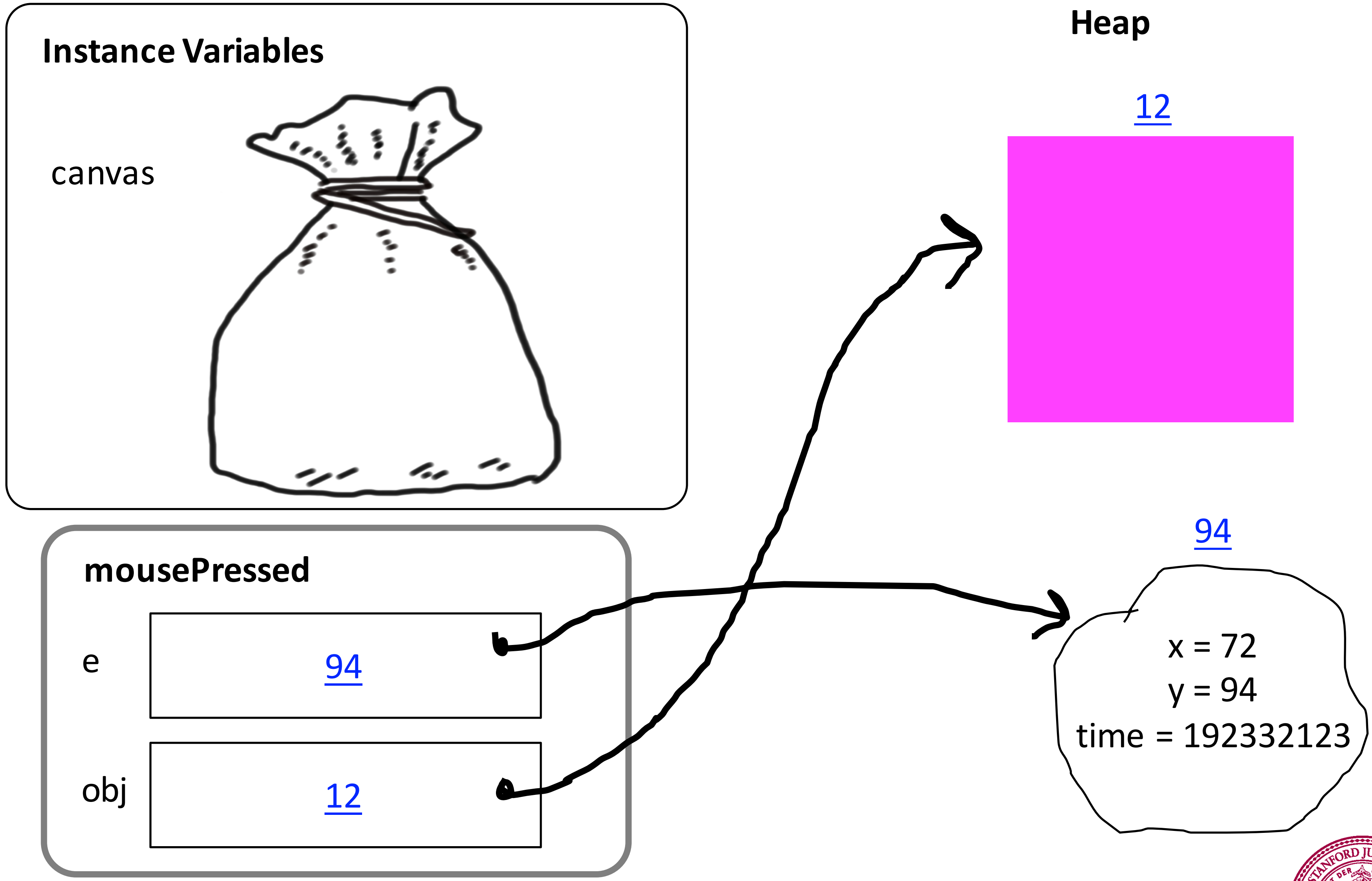
```
public class SimpleRect extends GraphicsProgram {

    public void run() {
        GRect r = null;
        r = new GRect(300, 300);
        r.setColor(Color.MAGENTA);
        add(r, 0, 0);
        addMouseListeners();
    }

    public void mousePressed(MouseEvent e) {
        GObject obj = getElementAt(1, 1);
        remove(obj);
    }

}
```

Canvas

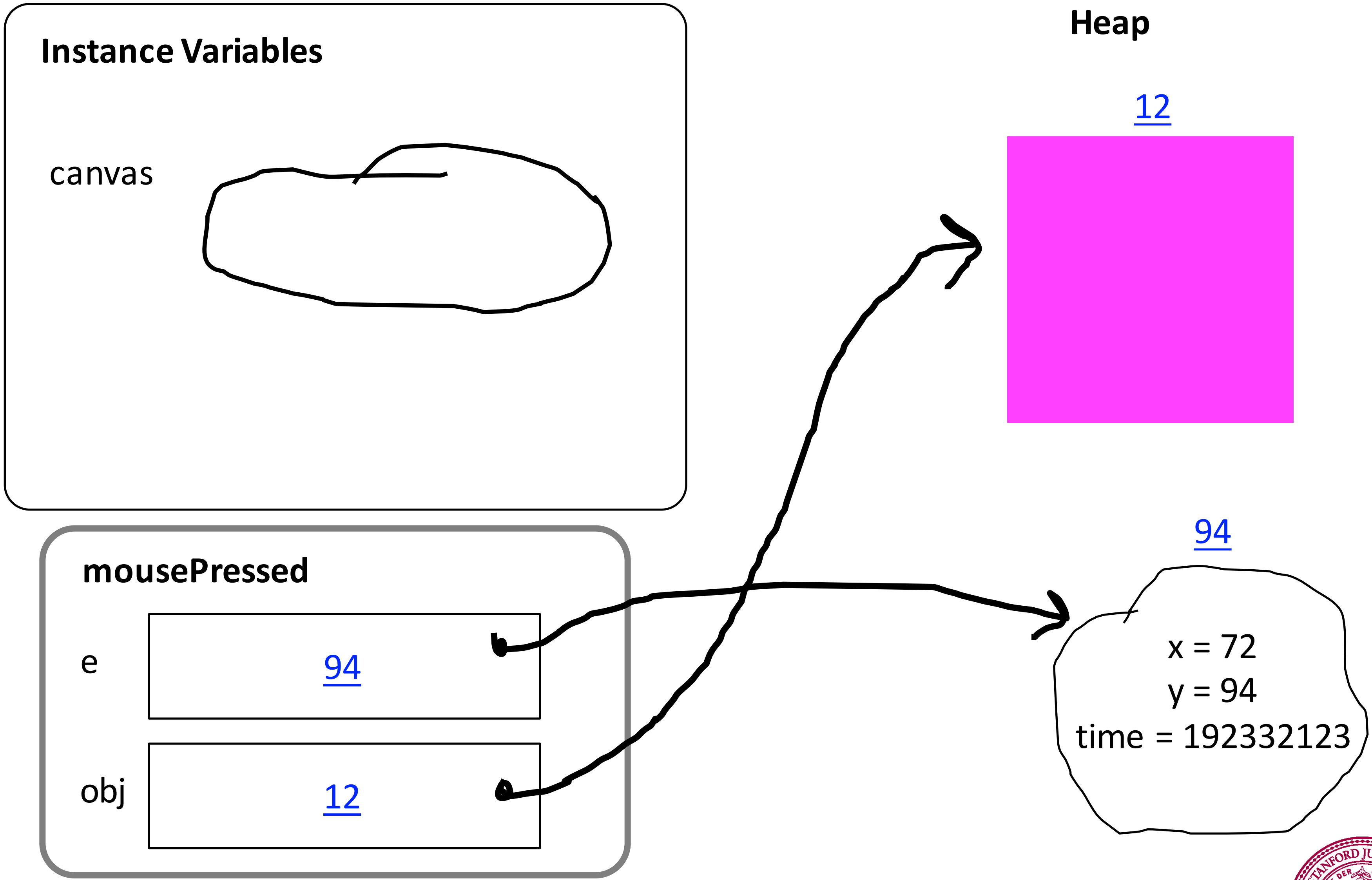


```
public class SimpleRect extends GraphicsProgram {

    public void run() {
        GRect r = null;
        r = new GRect(300, 300);
        r.setColor(Color.MAGENTA);
        add(r, 0, 0);
        addMouseListeners();
    }

    public void mousePressed(MouseEvent e) {
        GObject obj = getElementAt(1, 1);
        remove(obj);
    }

}
```

Canvas


Instance Variables
canvas
Heap
12
94
x = 72
y = 94
time = 192332123
mousePressed
e
94
obj
12

Canvas


Instance Variables
canvas
Heap
12
mousePressed
e
94
obj
12
94
x = 72
y = 94
time = 192332123

#6: graphics programs all have a “canvas” which keeps track of the objects on the screen

```
public void run() {
    GRect first = new GRect(50, 50);
    GRect second = first;
    add(first, 0, 0);
    add(second, 20, 20);
}
```

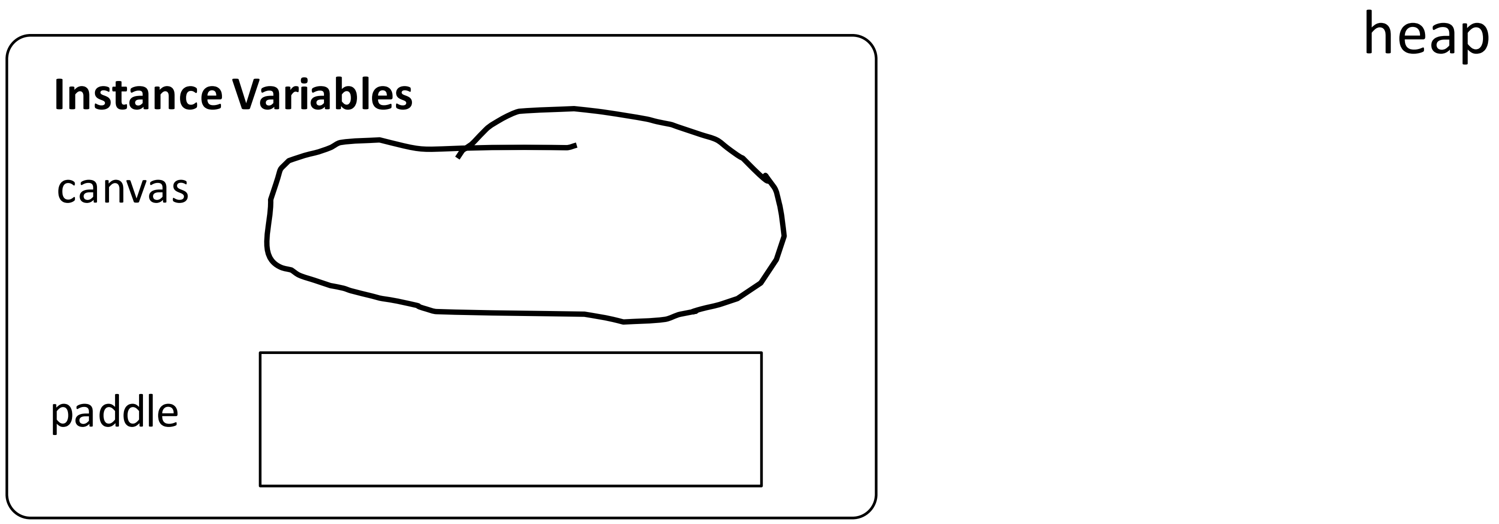
Intentionally left blank so that we can fill it in during lecture

What does an object store?

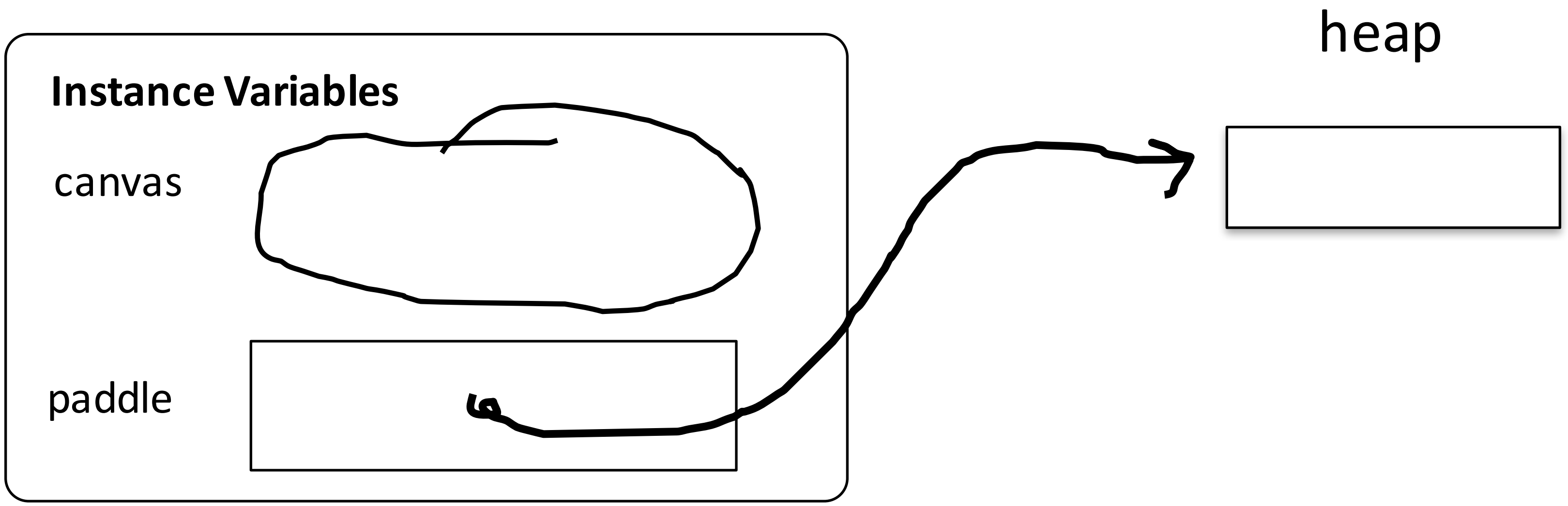
An object stores a memory address!

Instance Variables

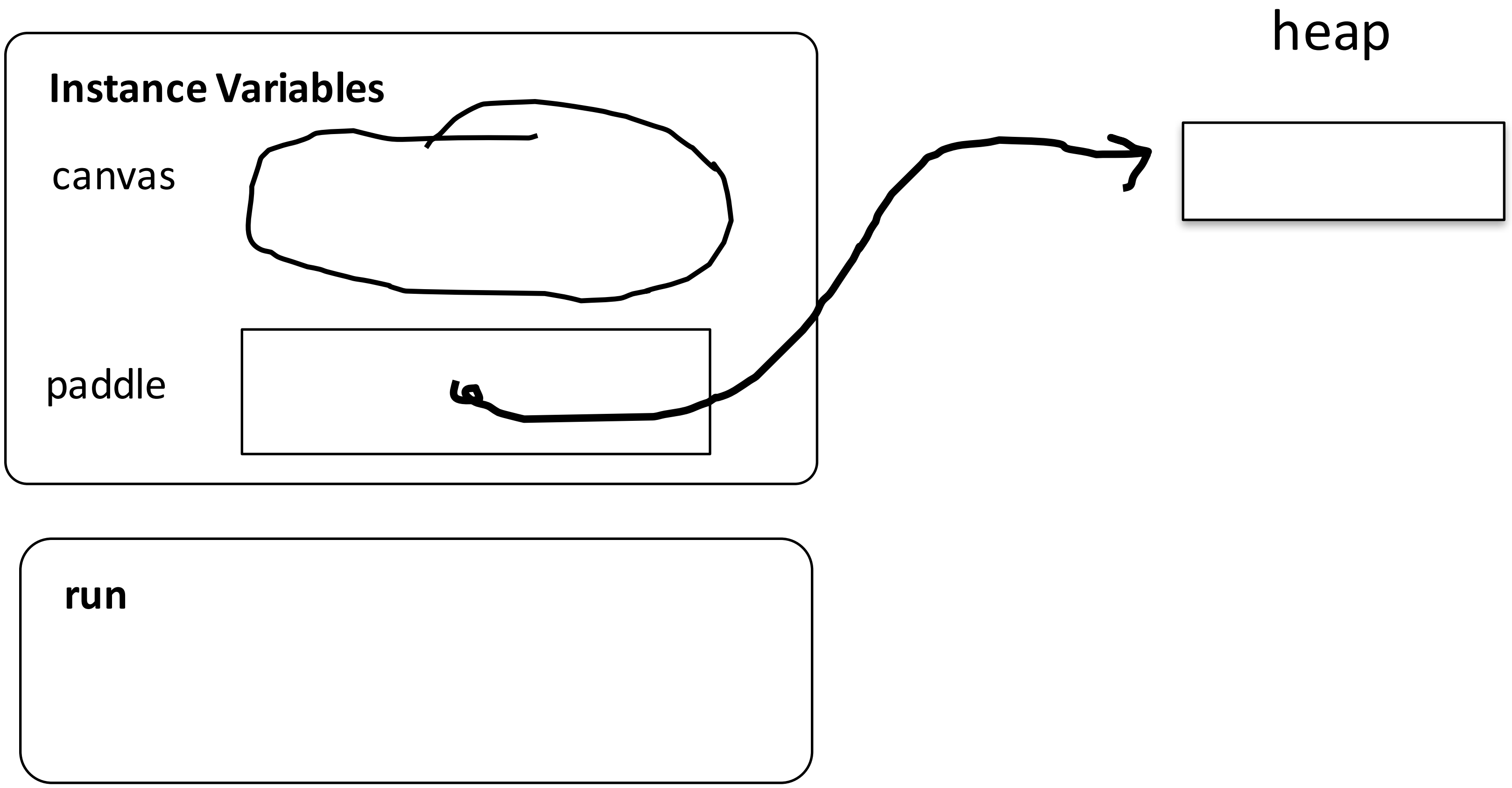
```
GRect paddle = new GRect(20, 30);
public void run() {
    paddle.setColor(Color.BLUE);
    add(paddle, 0, 0);
}
```



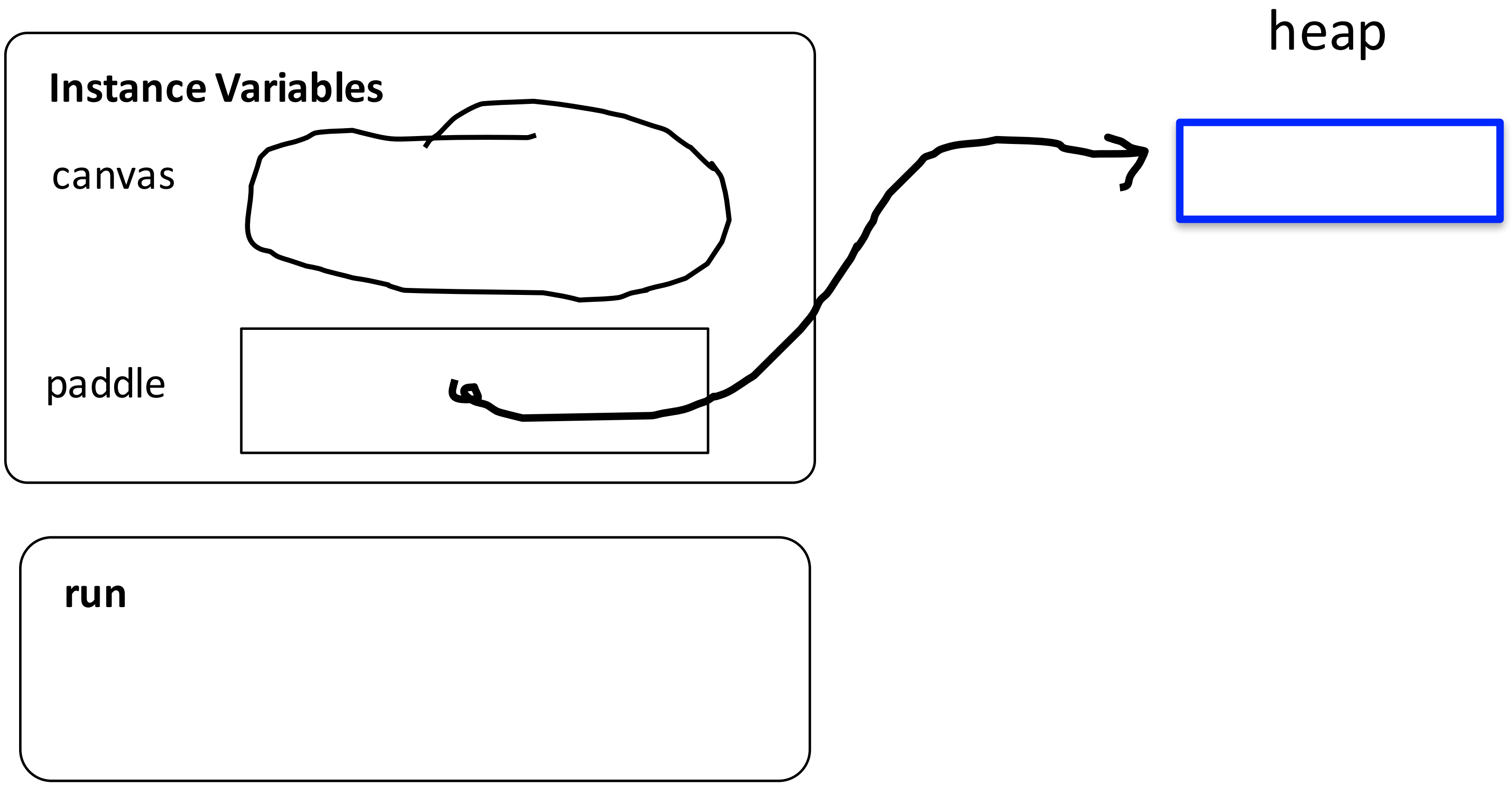
```
GRect paddle = new GRect(20, 30);
public void run() {
     paddle.setColor(Color.BLUE);
     add(paddle, 0, 0);
}
```



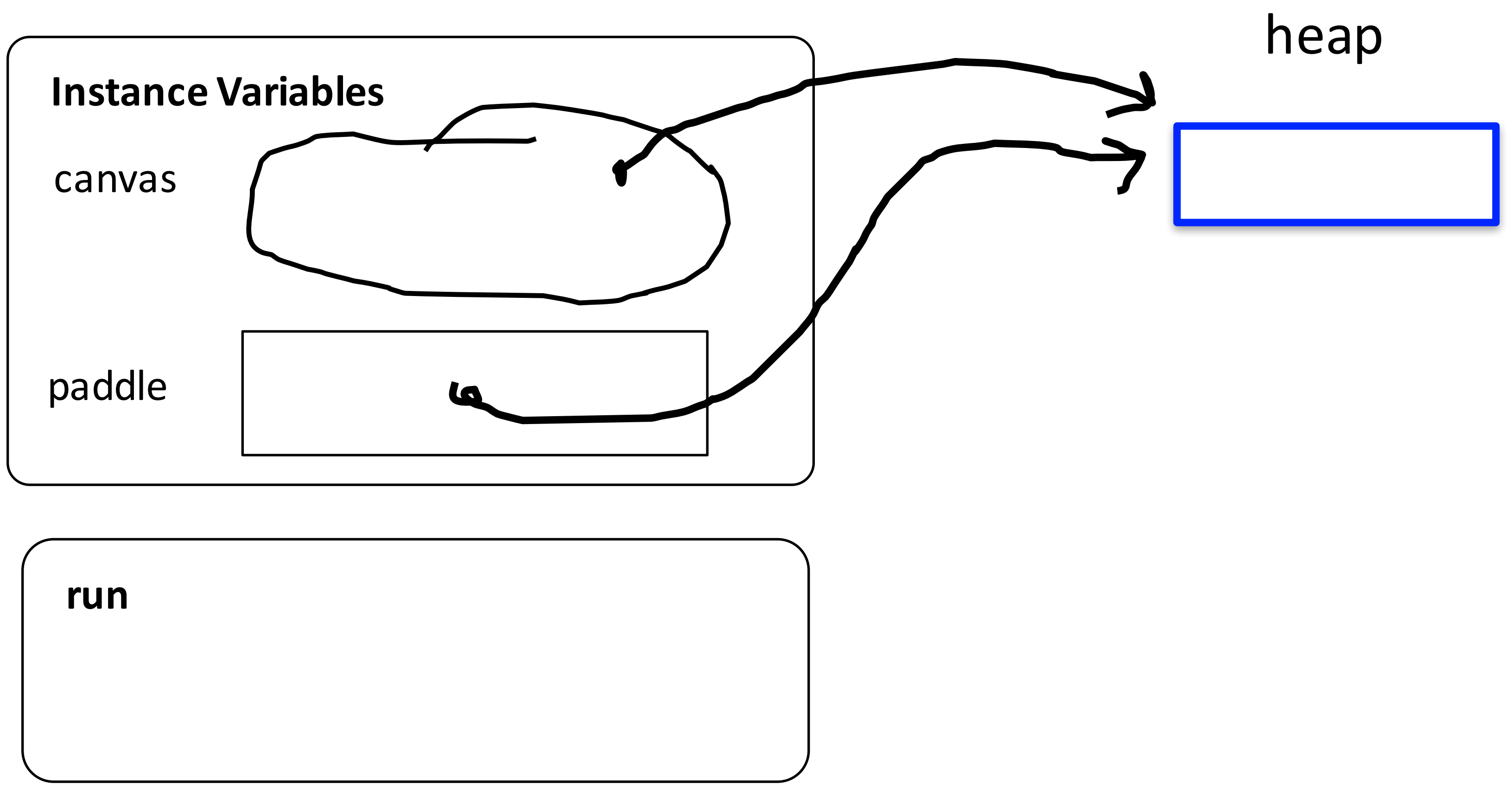
```
GRect paddle = new GRect(20, 30);
public void run() {
     paddle.setColor(Color.BLUE);
     add(paddle, 0, 0);
}
```



```
GRect paddle = new GRect(20, 30);
public void run() {
    paddle.setColor(Color.BLUE);
    add(paddle, 0, 0);
}
```



```
GRect paddle = new GRect(20, 30);
public void run() {
    paddle.setColor(Color.BLUE);
    add(paddle, 0, 0);
}
```

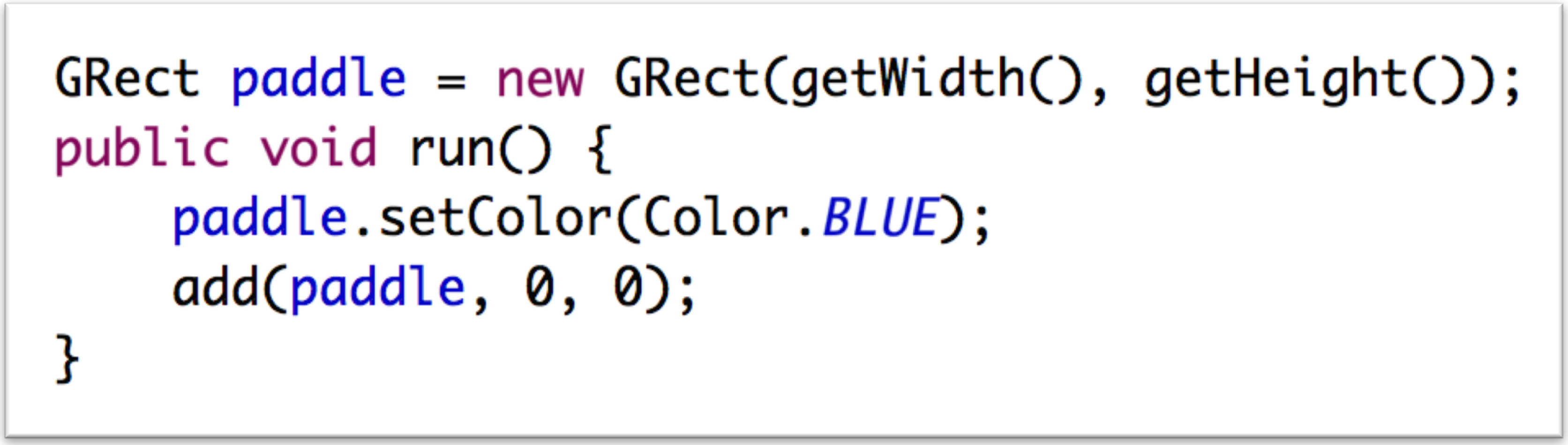


#7: there is space for all instance variables. They are accessible by the entire class

#8: instance variables are initialized before run is called

Common Bug

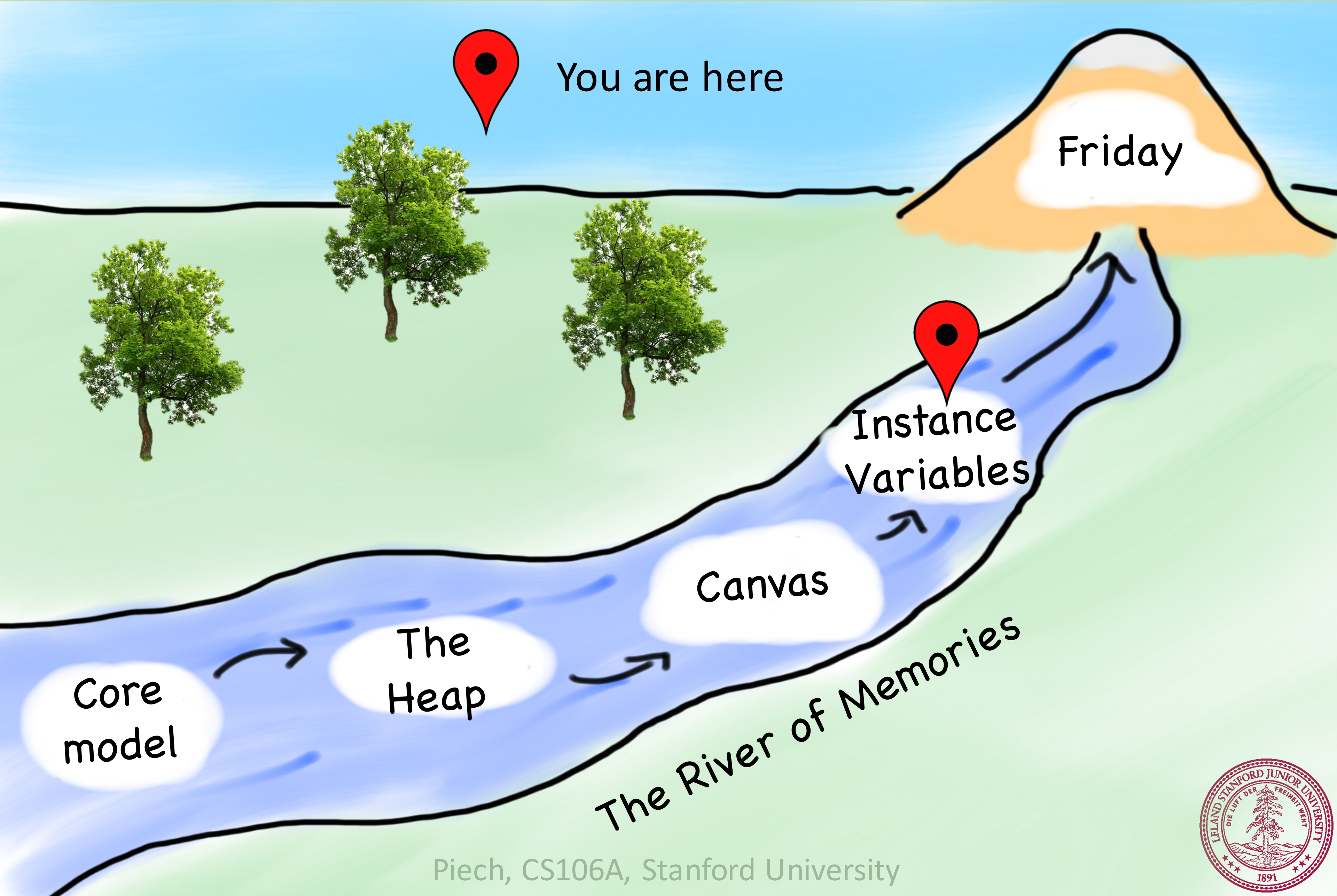
Question: what does this program do?



```
GRect paddle = new GRect(getWidth(), getHeight());
public void run() {
    paddle.setColor(Color.BLUE);
    add(paddle, 0, 0);
}
```

Answer: makes a square that is 0 by 0 since **`getWidth`** is called before the screen has been made.

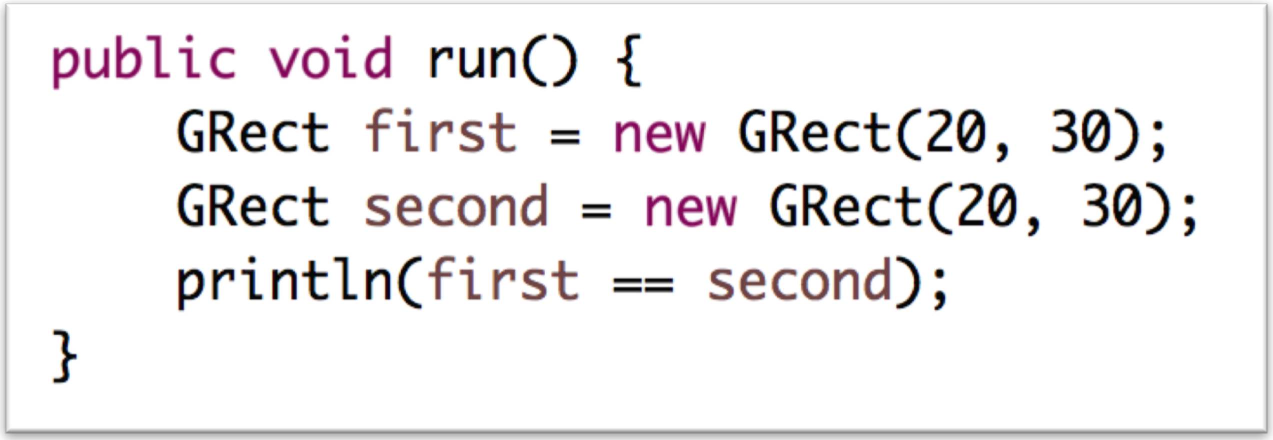
Today's Route



#9: for objects, == checks if the variables store the same address

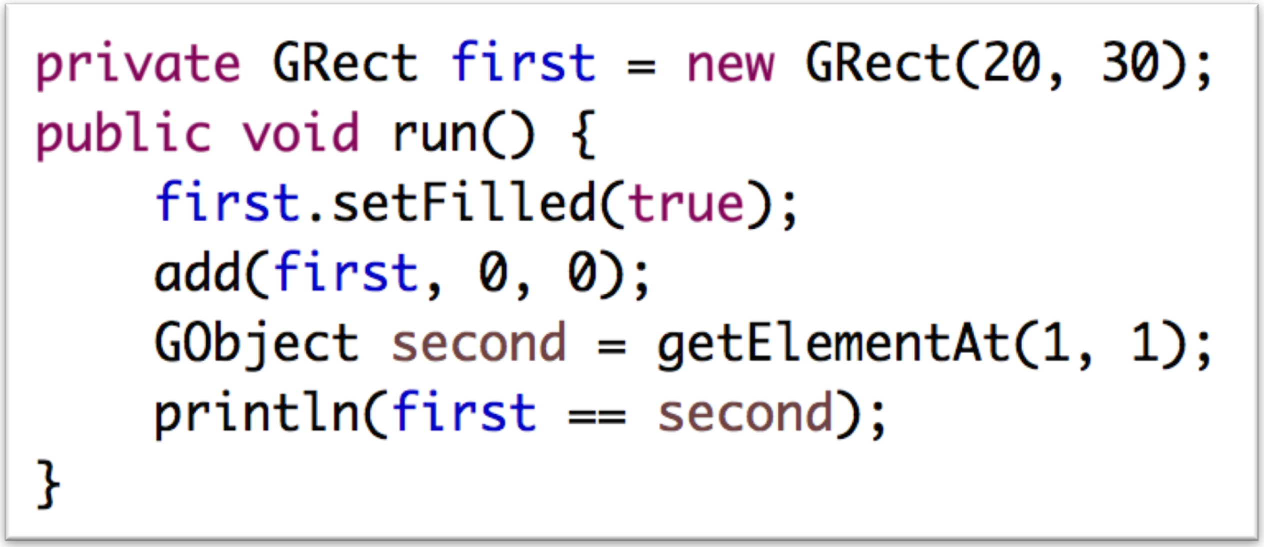
Recall the start of class?

Who thinks this prints **`true`**?



```
public void run() {
    GRect first = new GRect(20, 30);
    GRect second = new GRect(20, 30);
    println(first == second);
}
```

Who thinks this prints **`true`**?



```java
private GRect first = new GRect(20, 30);
public void run() {
    first.setFilled(true);
    add(first, 0, 0);
    GObject second = getElementAt(1, 1);
    println(first == second);
}
```

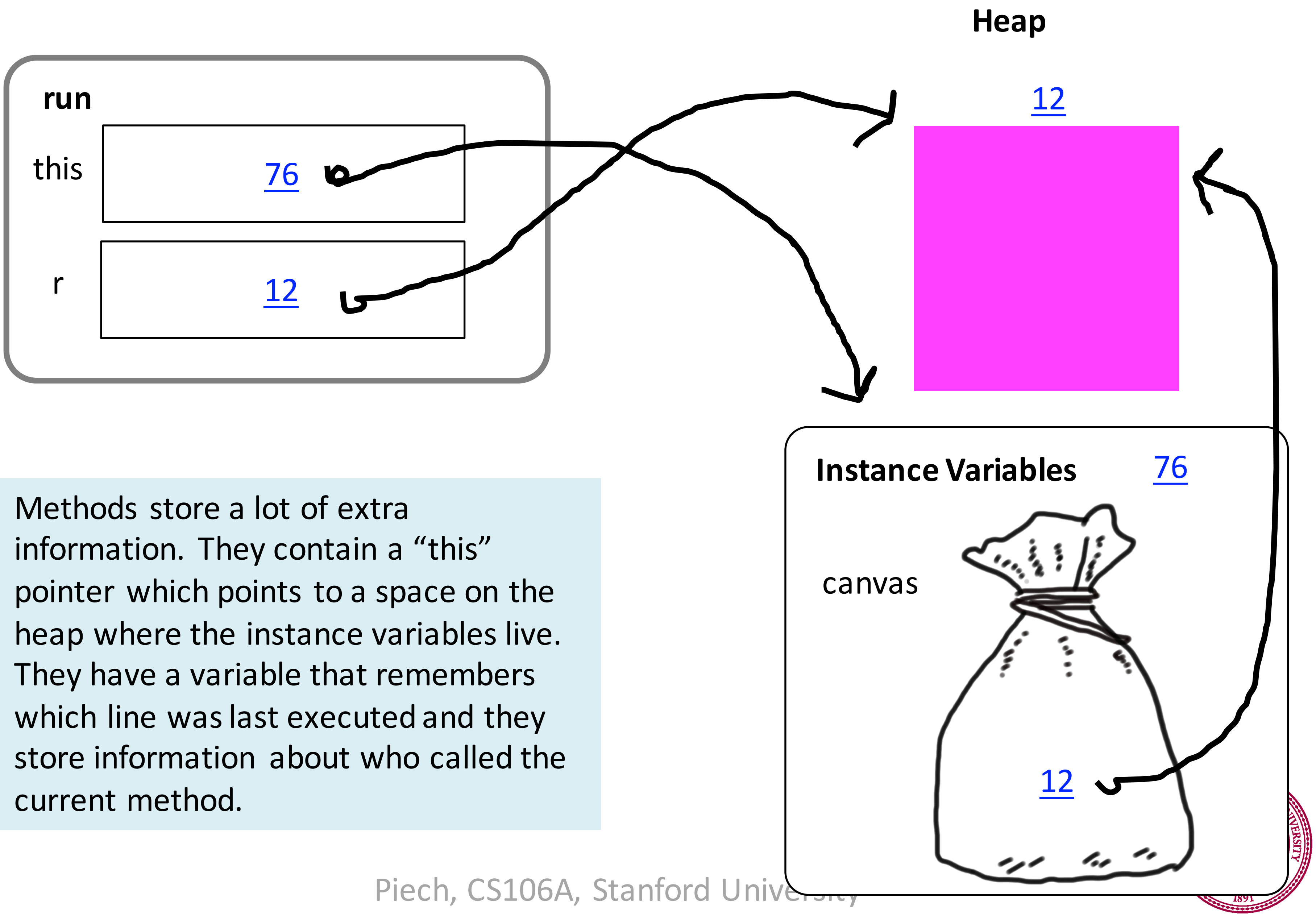
What does an object store?

An object stores a memory address!

Learning Goals

1. Be able to write a large program
2. Be able to trace memory with references

Beyond CS106A



Methods store a lot of extra information. They contain a “this” pointer which points to a space on the heap where the instance variables live. They have a variable that remembers which line was last executed and they store information about who called the current method.