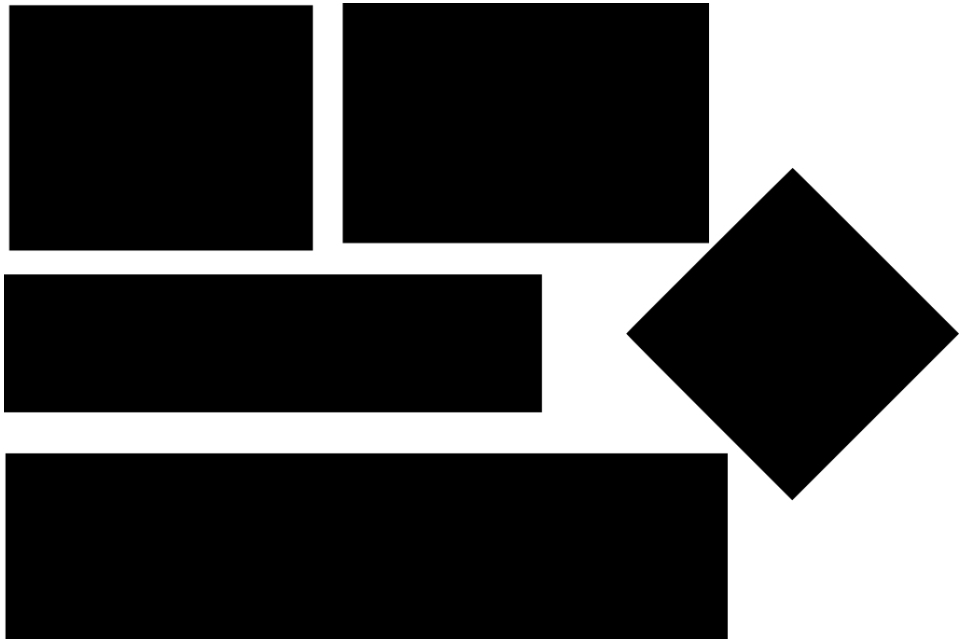


Rectangle Class

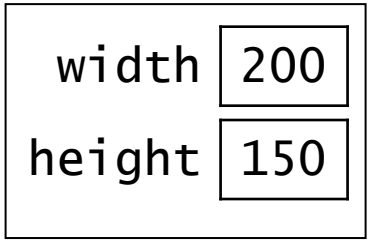
Computer Science OOD
Boston University

Christine Papadakis-Kanaris

Case Study: A Rectangle Class

- Let's say that we want to create a data type for objects that represent rectangles.

- Every Rectangle object should have two variables inside it (*`width`* and *`height`*) for the rectangle's dimensions.
  - these variables are referred to as *fields*
  - also known as: attributes, instance variables



- We'll also put functions/methods inside the object.

Sample Rectangle Class

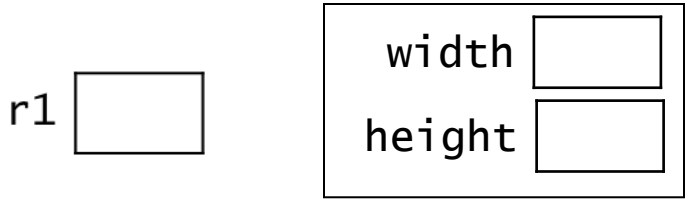
```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public Rectangle(int dim) {
        width = height = dim;
    }
    public Rectangle() {
        width = height = 0;
    }


    .
    .
    .
    .



}
```

Constructor

- The constructor has the same name as the class.
  - it is non-static
  - it has no return type
- The purpose of the constructor is to initialize the members.
- Constructors can be overloaded.
- A constructor that defines no parameters is referred to as the a no-arg constructor.
- If a class does not define ***any*** constructors, Java will provide a ***default*** *no-arg* constructor for the class.

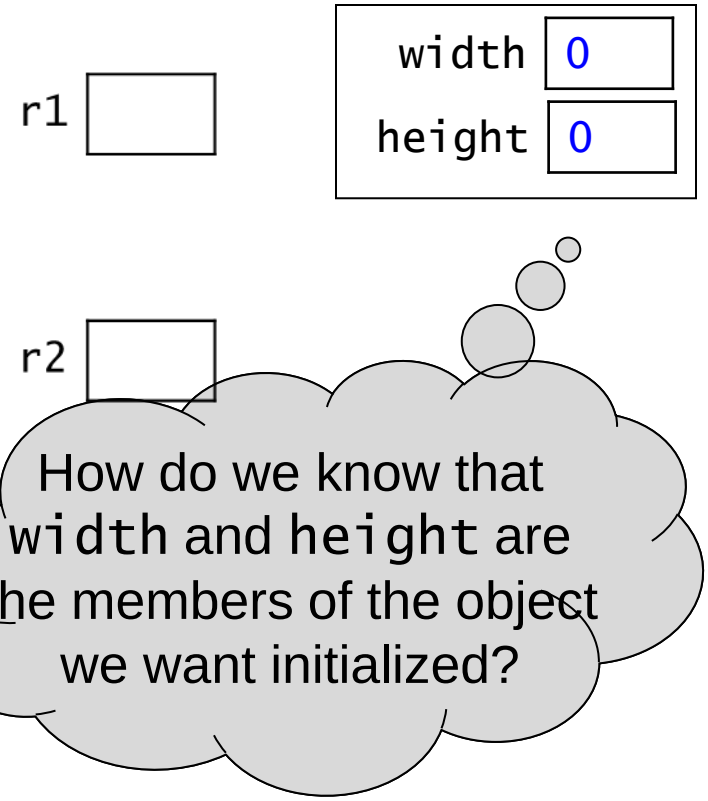
Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public Rectangle(int dim) {
        width = height = dim;
    }
    public Rectangle() {
        width = height = 0;
    }
    .
    .
    .
    public static void main( String [] args ) {

        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle(5, 10);
    }

}
```



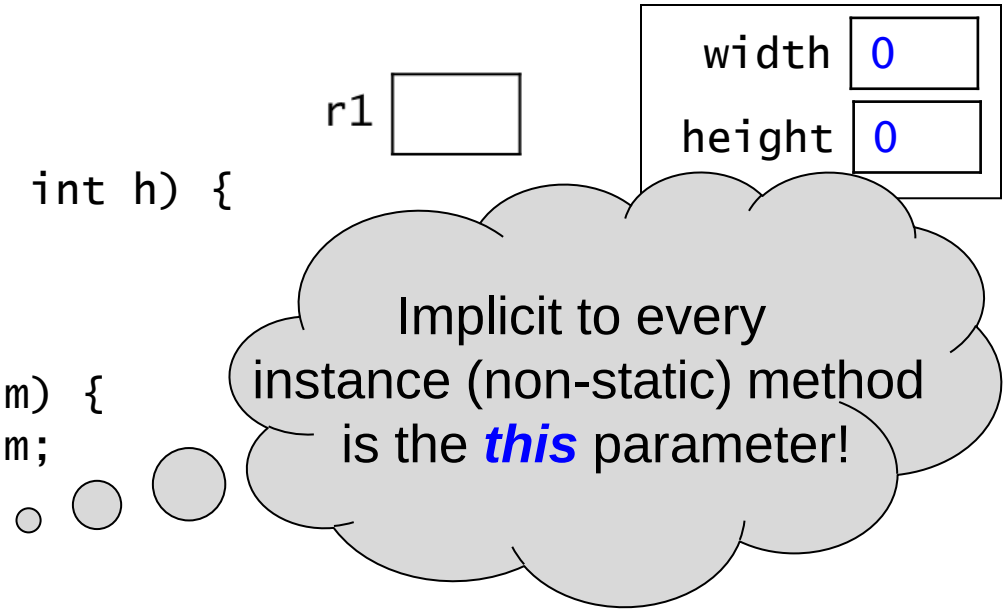
Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public Rectangle(int dim) {
        width = height = dim;
    }
    public Rectangle() {
        width = height = 0;
    }
    .
    .
    .
    public static void main( String [] args ) {

        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle(5, 10);
    }

}
```



Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public Rectangle(int dim) {
        width = height = dim;
    }
    public Rectangle() {
        width = height = 0;
    }
    .
    .
    .
    public static void main( String [] args ) {

        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle(5, 10);
    }

}
```



Sample Rectangle Class

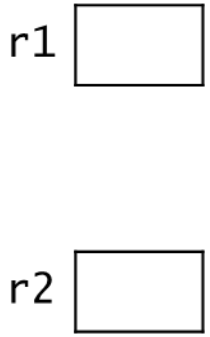
```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public Rectangle(int dim) {
        width = height = dim;
    }
    public Rectangle() {
        width = height = 0;
    }
    .
    .
    .
    public static void main( String [] args ) {

        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle(5, 10);
    }

}
```

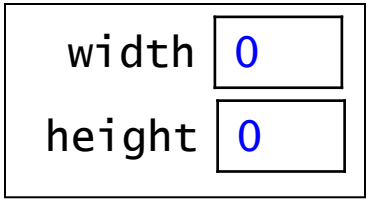
r2

*The **this*** paramater contains the address location of the object the method was called on.

Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public Rectangle(int dim) {
        width = height = dim;
    }
    public Rectangle() {
        this.width = this.height = 0;
    }
    .
    .
    .
    public static void main( String [] args ) {

        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle(5, 10);
    }

}
```





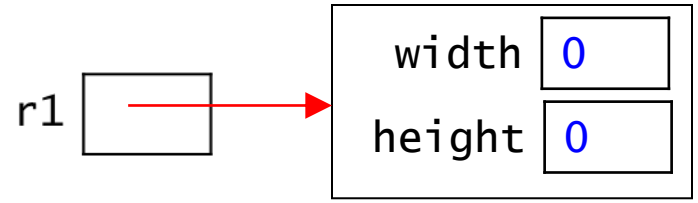
Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public Rectangle(int dim) {
        width = height = dim;
    }
    public Rectangle() {
        this.width = this.height = 0
    }
    .
    .
    .
    public static void main( String [] args ) {

        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle(5, 10);
    }

}
```

Note that this call is part of an assignment statement.

Sample Rectangle Class

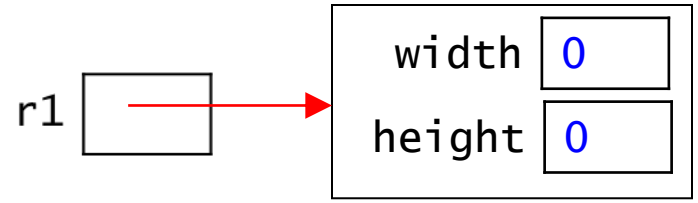
```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public Rectangle(int dim) {
        width = height = dim;
    }
    public Rectangle() {
        this.width = this.height = 0
    }
    .
    .
    .
    public static void main( String [] args ) {

        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle(5, 10);
    }

}
```



Constructors return the address location of the object constructed via the *this* parameter!

Sample Rectangle Class

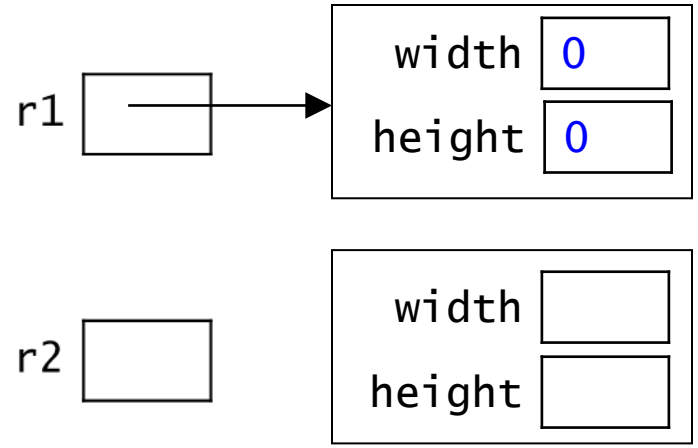
```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public Rectangle(int dim) {
        width = height = dim;
    }
    public Rectangle() {
        this.width = this.height = 0
    }
    .
    .
    .
    public static void main( String [] args ) {

        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle(5, 10);
    }

}
```



This is why constructors cannot be declared to be void methods!

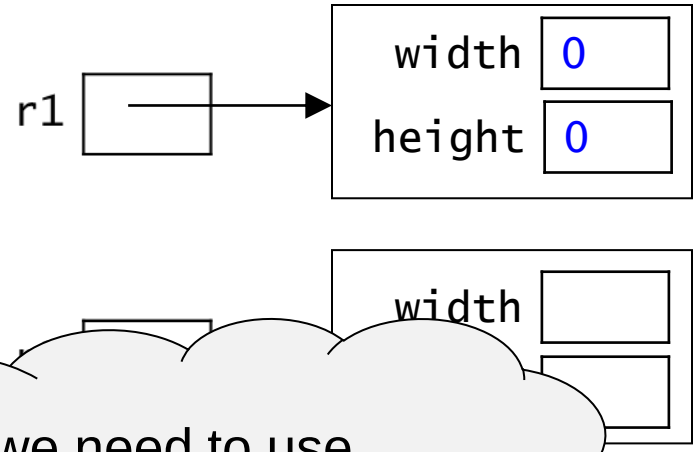
Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public Rectangle(int dim) {
        width = height = dim;
    }
    public Rectangle() {
        width = height = 0;
    }
    .
    .
    .
    public static void main( String [] args ) {

        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle(5, 10);
    }

}
```



Sample Rectangle Class

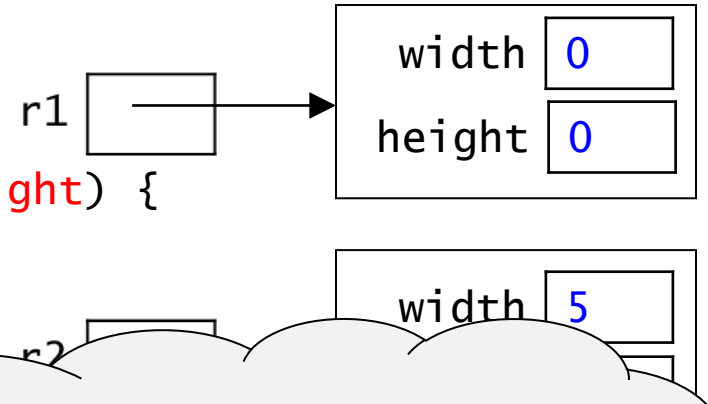
```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public Rectangle(int dim) {
        width = height = dim;
    }
    public Rectangle() {
        width = height = 0;
    }
    .
    .
    .
    public static void main( String [] args ) {

        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle(5, 10);
    }

}
```

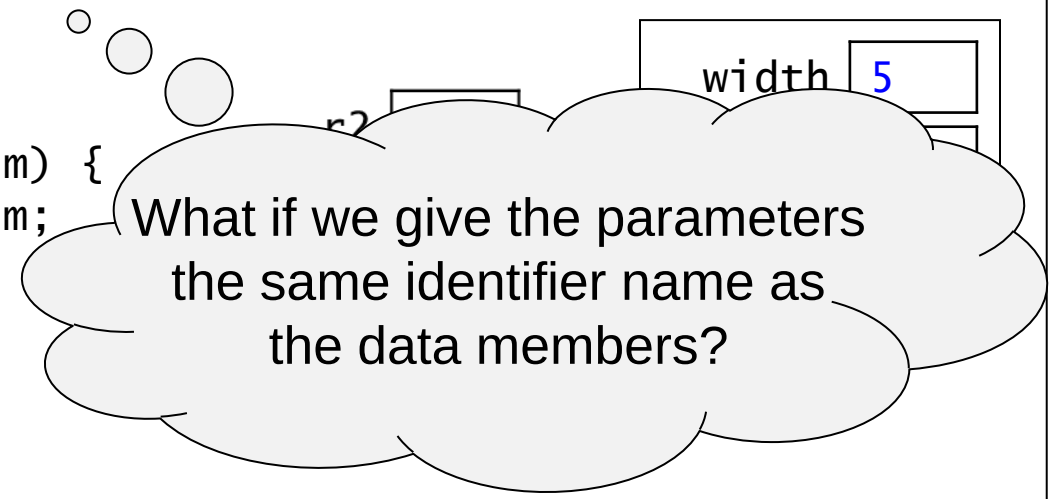


Do we need to use the *this* reference to access the data members?

Sample Rectangle Class

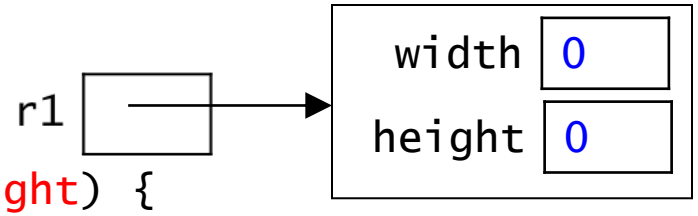
```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int width, int height) {
        width = width;
        height = height;
    }
    public Rectangle(int dim) {
        width = height = dim;
    }
    public Rectangle() {
        width = height = 0;
    }
    .
    .
    .
    public static void main( String [] args ) {

        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle(5, 10);
    }

}
```

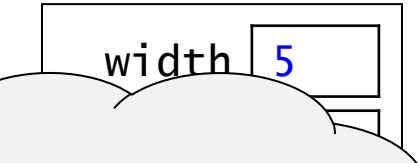




Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int width, int height) {
        width = width;
        height = height;
    }
    public Rectangle(int dim) {
        width = height = dim;
    }
    public Rectangle() {
        width = height = 0;
    }
    .
    .
    .
    public static void main( String [] args ) {

        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle(5, 10);
    }

}
```

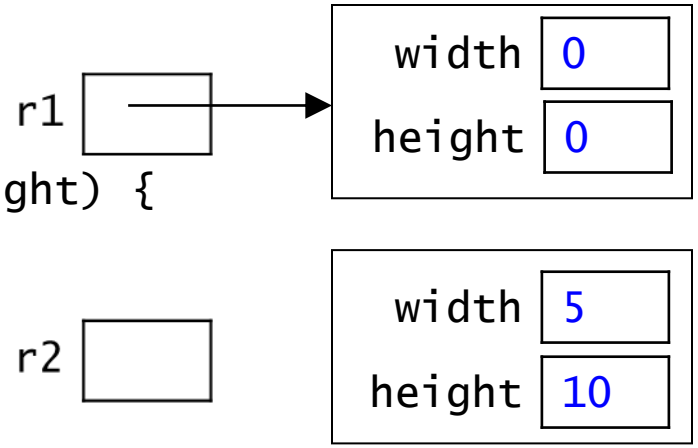




Now we have a scope issue!

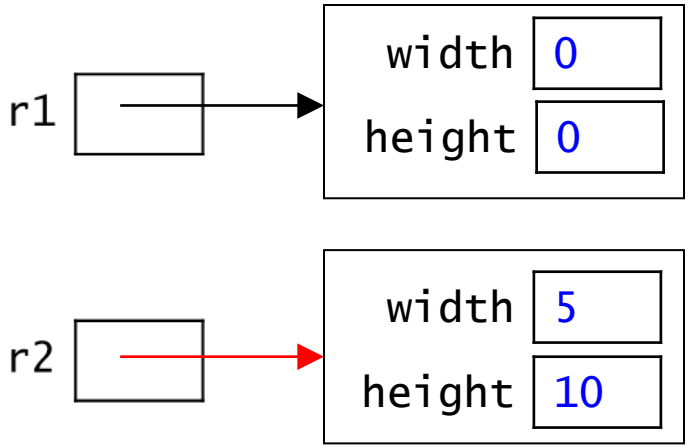
Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }
    public Rectangle(int dim) {
        width = height = dim;
    }
    public Rectangle() {
        width = height = 0;
    }
    .
    .
    .
    public static void main( String [] args ) {

        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle(5, 10);
    }

}
```



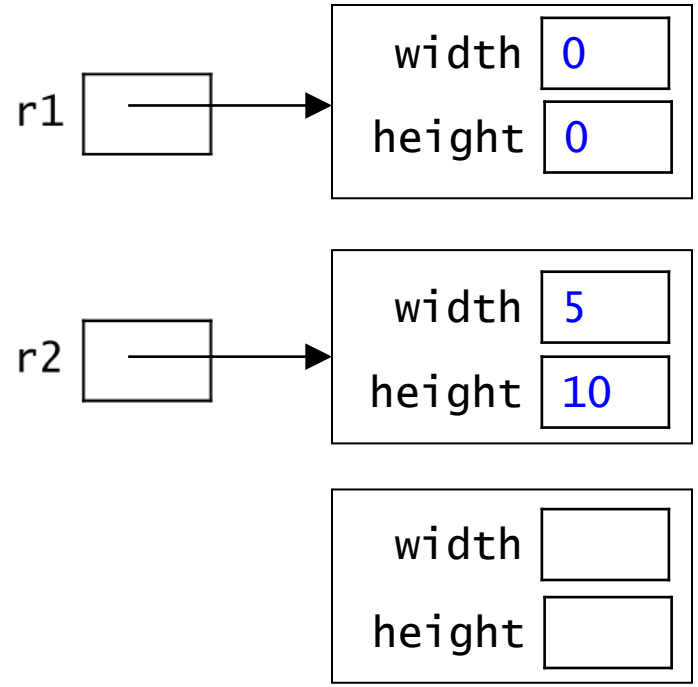
Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public Rectangle(int dim) {
        width = height = dim;
    }
    public Rectangle() {
        width = height = 0;
    }
    .
    .
    .
    public static void main( String [] args ) {

        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle(5, 10);
    }


}
```



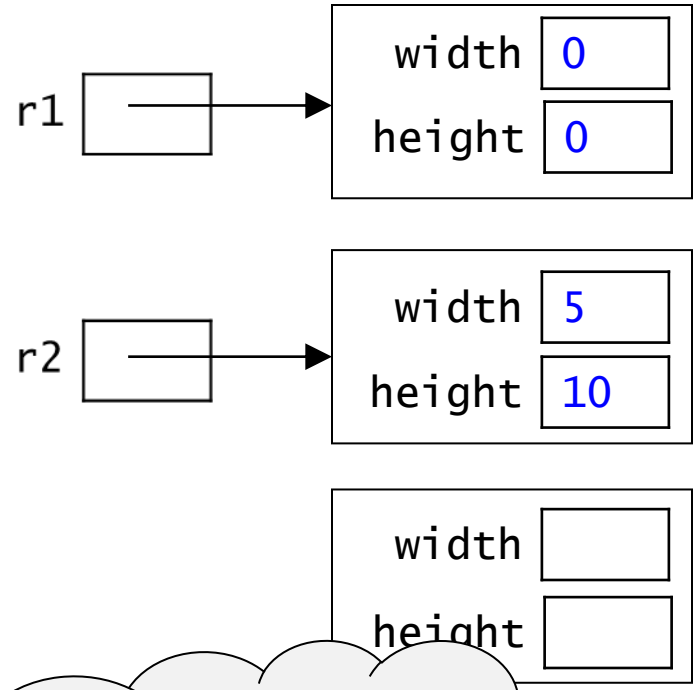
Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public Rectangle(int dim) {
        width = height = dim;
    }
    public Rectangle() {
        width = height = 0;
    }
    .
    .
    .
    public static void main( String [] args ) {

        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle(5, 10);
        Rectangle r3 = new Rectangle(7);
    }

}
```



Sample Rectangle Class

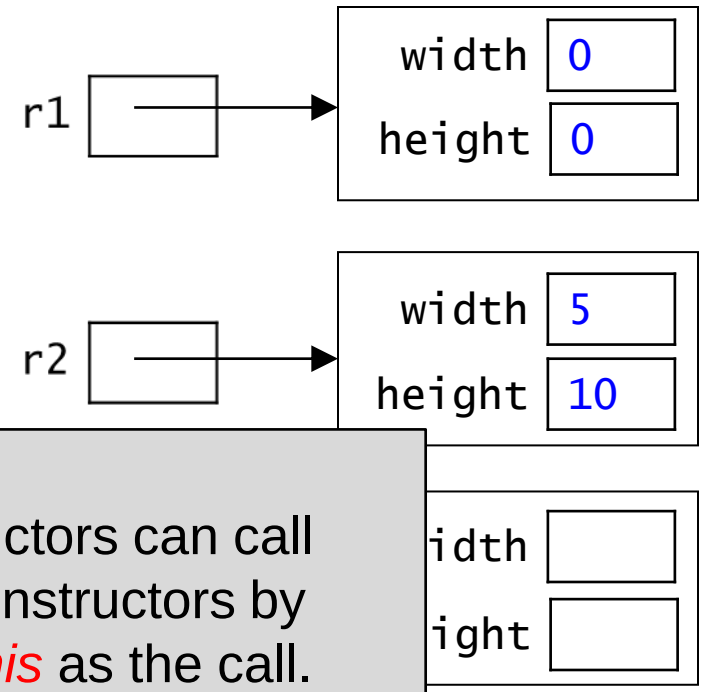
```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public Rectangle(int dim) {
        width = height = dim;
    }
    public Rectangle() {
        width = height = 0;
    }
    .
    .
    .
    public static void main( String

        Rectangle r1 = new Rectangle(),
        Rectangle r2 = new Rectangle(5, 10),
        Rectangle r3 = new Rectangle(7);
    }

}
```



Note that both constructors are doing the same thing.

Sample Rectangle Class

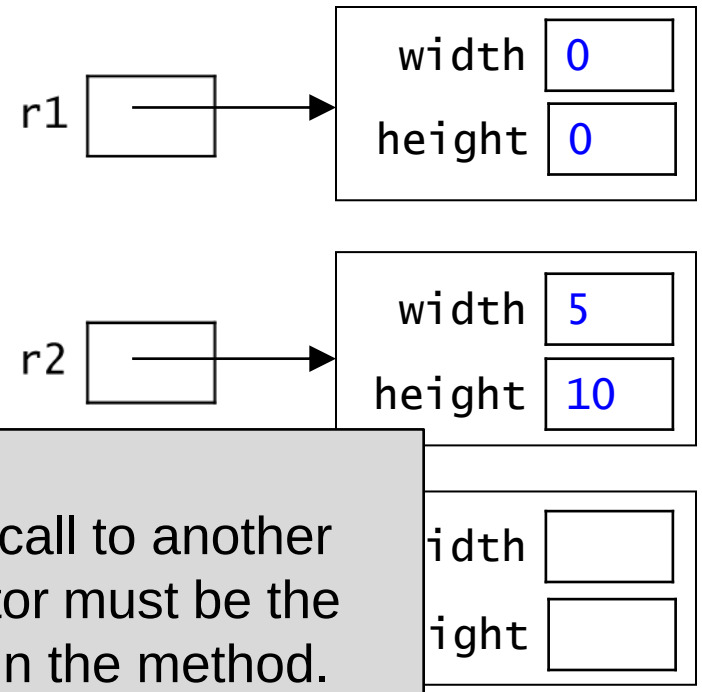
```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public Rectangle(int dim) {
        this(dim, dim);
    }
    public Rectangle() {
        width = height = 0;
    }
    .
    .
    .
    public static void main( String [] args ) {

        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle(5, 10);
        Rectangle r3 = new Rectangle(7);
    }

}
```



Constructors can call *other* constructors by using *this* as the call.

Sample Rectangle Class

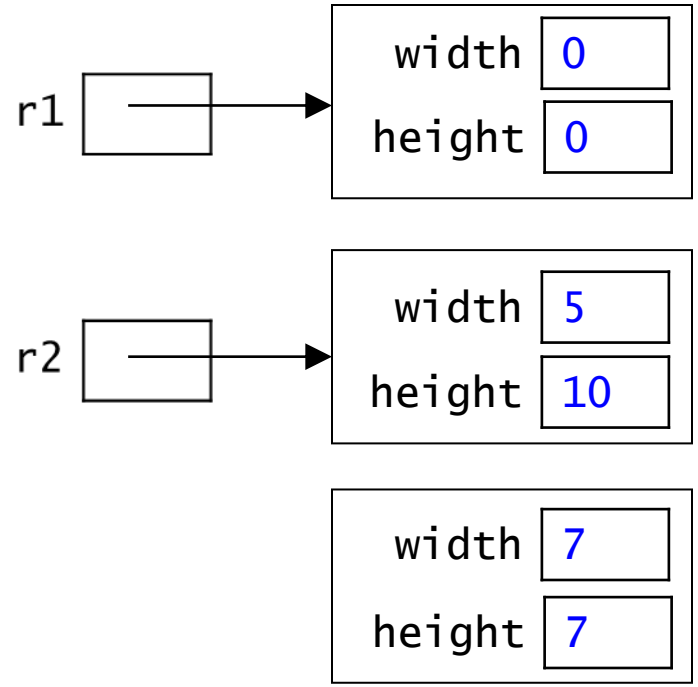
```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public Rectangle(int dim) {
        this(dim, dim);
    }
    public Rectangle() {
        width = height = 0;
    }
    .
    .
    .
    public static void main( String [] args ) {

        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle(5, 10);
        Rectangle r3 = new Rectangle(7);
    }

}
```



The *this* call to another constructor must be the first call in the method.

Sample Rectangle Class

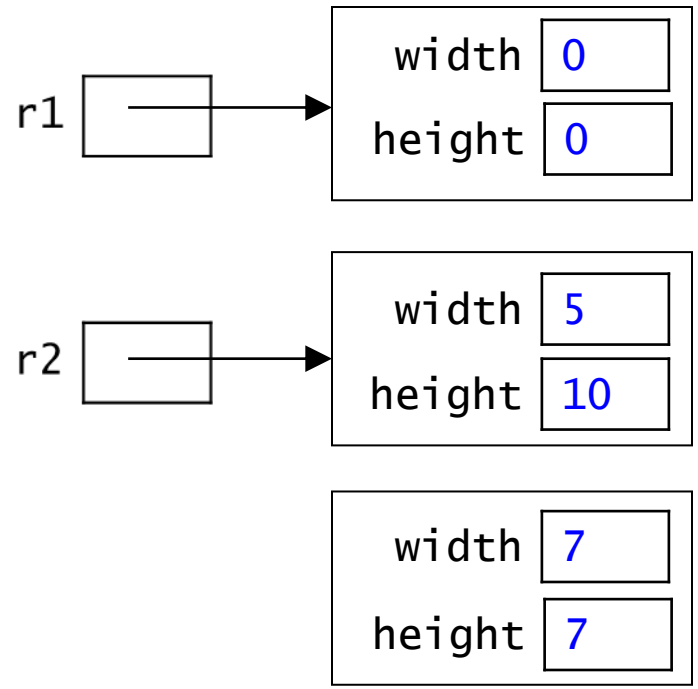
```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public Rectangle(int dim) {
        this(dim, dim);
    }
    public Rectangle() {
        width = height = 0;
    }
    .
    .
    .
    public static void main

        Rectangle r1 = new
        Rectangle r2 = new Re
        Rectangle r3 = new Rectang
    }

}
```



This constructor is also doing the same as the other two constructors.

Sample Rectangle Class

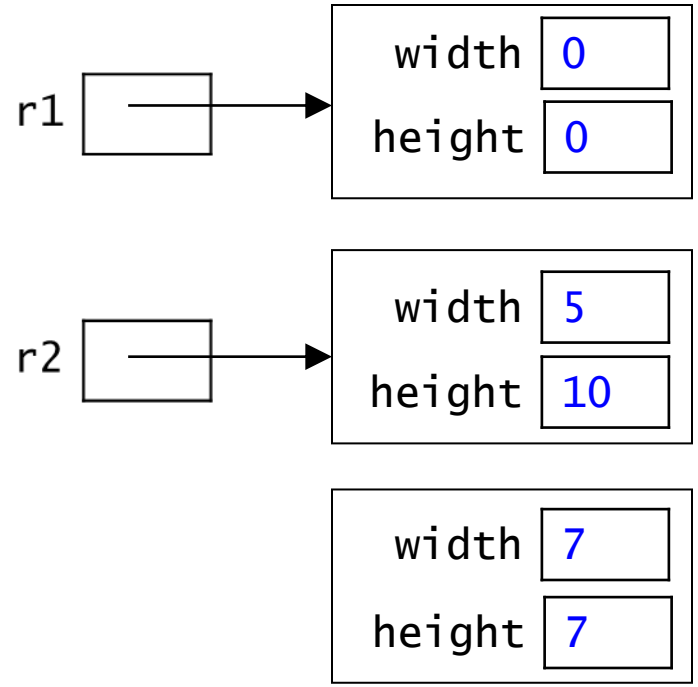
```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public Rectangle(int dim) {
        this(dim, dim);
    }
    public Rectangle() {
        this(0, 0);
    }
    .
    .
    .
    public static void main

        Rectangle r1 = new
        Rectangle r2 = new Re
        Rectangle r3 = new Rectang
    }

}
```



Can use *this* to call one of the other constructors.

Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public Rectangle(int dim) {
        this(dim, dim);
    }
    public Rectangle() {
        this(0);
    }
    .
    .
    .
    public static void main

        Rectangle r1 = new
        Rectangle r2 = new Re
        Rectangle r3 = new Rectang
    }

}
```



Can use *this* to call one of the other constructors.

Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }

    ...

}
```

Accessor Methods

Mutator Methods

Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public int getWidth() {
        return width;
    }
    public int getHeight() {
        return height;
    }
    public void grow(int dw, int dh) {
        width += dw;
        height += dh;
    }
    public double area() {
        return( width*height );
    }

    ...

}
```

Accessor Methods

- Allow *applications* or *client methods* to gain access to the data stored in private data members!

Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public int getWidth() {
        return width;
    }
    public int getHeight() {
        return height;
    }
    public void grow(int dw, int dh) {
        width += dw;
        height += dh;
    }
    public double area() {
        return( width*height );
    }

    ...

}
```

Accessor Methods

- Allow *applications* or *client methods* to gain access to the data stored in private data members!
- Or perform a necessary operation of the class without altering the values of the data members.

Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    public int getWidth() {
        return width;
    }
    public int getHeight() {
        return height;
    }
    public void grow(int dw, int dh) {
        width += dw;
        height += dh;
    }
    public double area() {
        return( width*height );
    }

    ...

}
```

Mutator Methods

- Alter the values of the data members.

Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int
        width = w;
        height = h;
    }
    .
    .
    .
```

```
public void grow(int width, int height) {
      this.width += width;
      this.height += height;
  }
```

```
    .
    .
    .
}
```

What if the variables passed to the method result in inappropriate dimensions, a negative width or height?

Allowing Appropriate Changes

- To allow for appropriate changes to an object, we add whatever mutator methods make sense.

- These (*setter*) methods can prevent inappropriate changes:

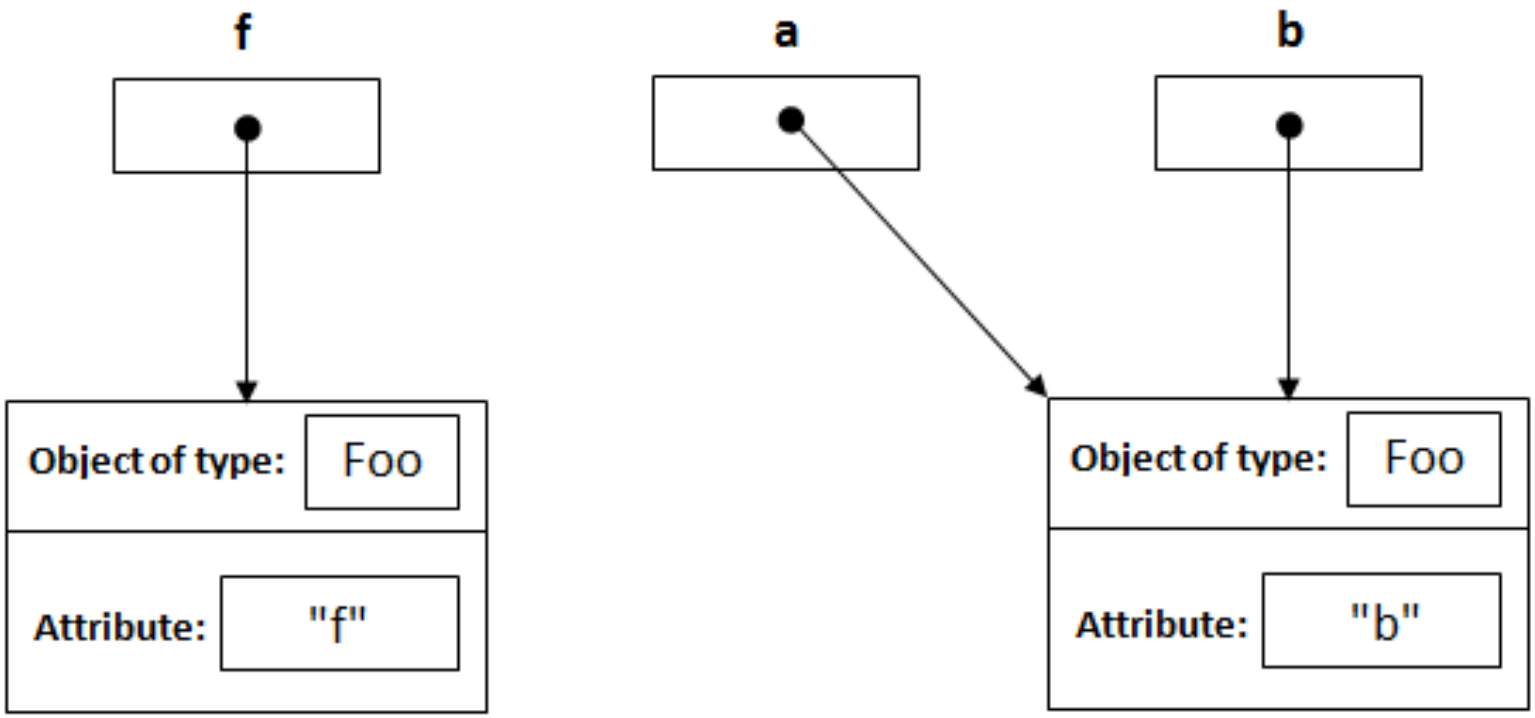
```
public void setWidth(int w) {
    if (w <= 0) {
        throw new IllegalArgumentException();
    }

    this.width = w;
}

public void setHeight(int h) {
    if (h <= 0) {
        throw new IllegalArgumentException();
    }

    this.height = h;
}
```

Throwing an exception ends the method call.

Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;
    public Rectangle(int w, int h) {
        this.setWidth(w);
        this.setHeight(h);
    }
    .
    .
    .
    public void grow(int dw, int dh) {
        this.setWidth(width+dw);
        this.setHeight(height+dh);
    }
    .
    .
    .
}
```

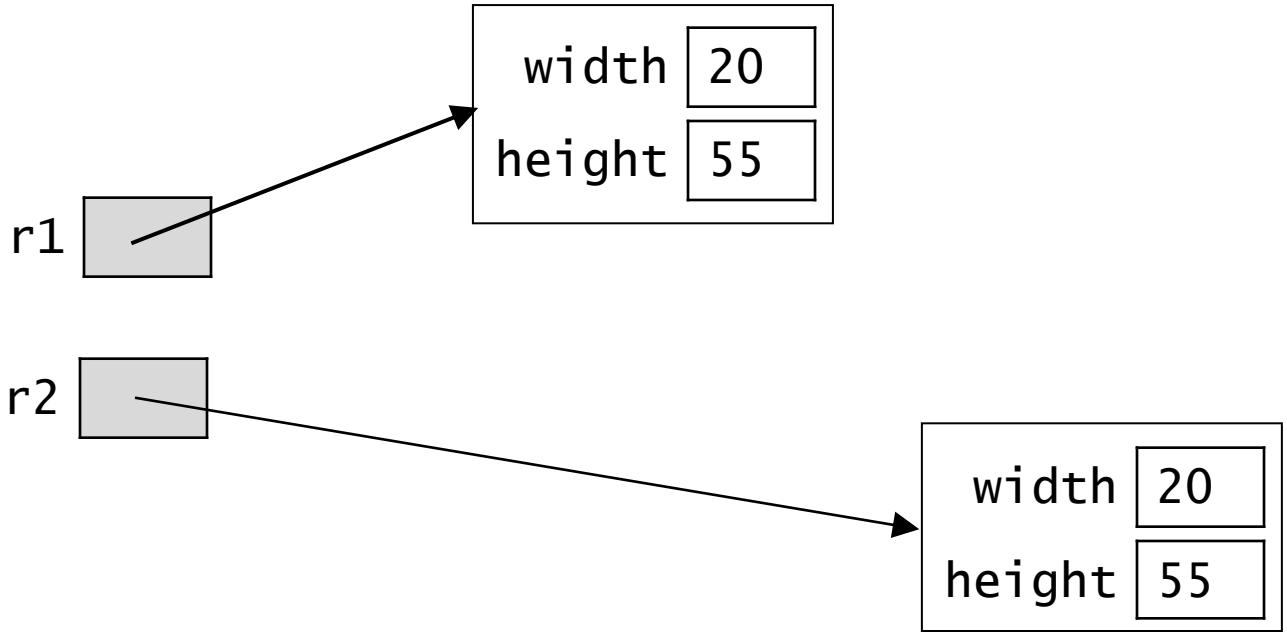
Objects are *Reference* types



Testing for Equivalent Objects

- Let's say that we have two different `Rectangle` objects, both of which represent the same rectangle:

```
Rectangle r1 = new Rectangle(20, 55);
Rectangle r2 = new Rectangle(20, 55);
```



- What is the value of the following condition?

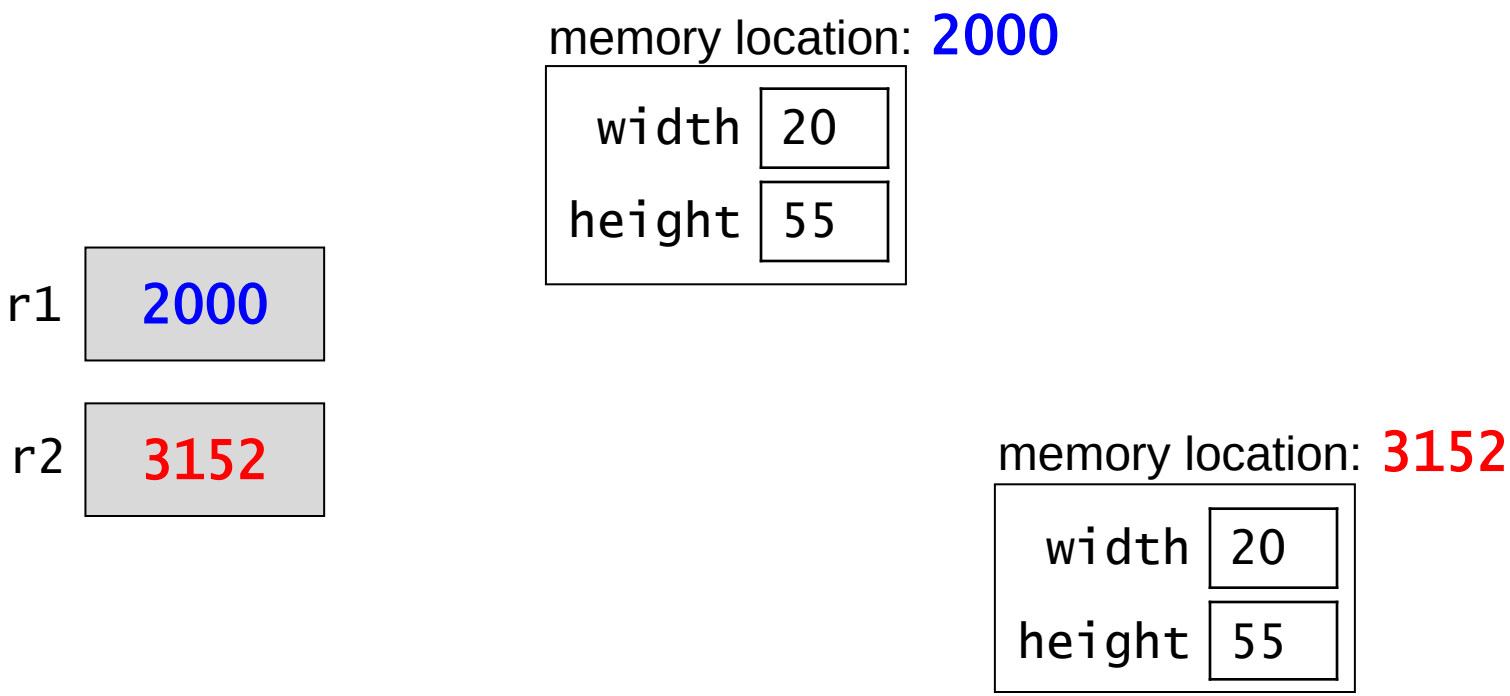
```
r1 == r2
```

Testing for Equivalent Objects (cont.)

- The condition

  `r1 == r2`

  compares the *references* stored in `r1` and `r2`.



- It doesn't compare the objects themselves.

Testing for Equivalent Objects (cont.)

- To test for equivalent objects, we need to use the `equals` method:

```
r1.equals(r2) // commutative
```

Testing for Equivalent Objects (cont.)

- To test for equivalent objects, we need to use the `equals` method:

```
r2.equals(r1) // commutative
```

Testing for Equivalent Objects (cont.)

- To test for equivalent objects, we need to use the `equals` method:

```
r1.equals(r2)
```

- Java's built-in classes have an *equals* methods that:
  - returns `true` if the two objects are equivalent to each other
  - returns `false` otherwise

```
String s1 = "CS112";
String s2 = "CS611";
if ( s1.equals(s2) )
   System.out.println("I am not doing my job!");
```

Default `equals()` Method

- If we don't write an `equals()` method for a class, objects of that class get a default version of this method.

- The default `equals()` just tests if the memory addresses of the two objects are the same.
  - the same as what == does!

- To ensure that we're able to test for equivalent objects, we need to write our own `equals()` method.

`equals()` Method for Our `Rectangle` Class (cont.)

- Here's an alternative version:

```
public boolean equals(Rectangle other) {
    return (other != null
            && this.width == other.width
            && this.height == other.height);
}
```

Converting an Object to a String

```
Rectangle r1 = new Rectangle(10, 20);
System.out.println(r1.toString());
```

Java makes an *implicit* call to the `toString` method

Converting an Object to a String

- The `toString()` method allows objects to be displayed in a human-readable format.
  - it returns a string representation of the object

- This method is called *implicitly* when you attempt to print an object or when you perform string concatenation:

```
Rectangle r1 = new Rectangle(10, 20);
System.out.println(r1);
```

equivalent to:

```
System.out.println(r1.toString());
```

Converting an Object to a String

- The `toString()` method allows objects to be displayed in a human-readable format.
  - it returns a string representation of the object

- This method is called implicitly when you attempt to print an object or when you perform string concatenation:

```
Rectangle r1 = new Rectangle(10, 20);
System.out.println(r1);

// the second line above is equivalent to:
System.out.println(r1.toString());
```

- If we don't write a `toString()` method for a class, objects of that class get a default version of this method.
  - here again, it usually makes sense to write our own version

`toString()` Method for Our `Rectangle` Class

```
public String toString() {
    return width + " x " + height;
}
```

- Note: the method does not do any printing. It returns a `String` that can then be printed.

Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        setWidth(w);
        setHheight(h);
    }
    .
    .
    .
    public void grow(int dw, int dh) {
        setWidth(width+dw);
        setHeight(height+dh);
    }
    public double area() {
        return(width*height);
    }
    public boolean equals(Rectangle other) {
        return (other != null && this.width == other.width
                                && this.height == other.height );
    }
    public String toString() {
       return (width + " x " + height);
    }
```

A sample Client Program

```
public class RectangleClient {

    public static void main(String[] args) {

        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        System.out.println("r1's area = " + r1.area() );

        System.out.println("r2's area = " + r2.area() );

        // grow both rectangles
        r1.grow(50, 10);
        r2.grow(5, 30);

        System.out.println("r1: " + r1);
        System.out.println("r2: " + r2);



    }
}
```

A sample Client Program

```
public class RectangleClient {

    public static void main(String[] args) {

        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        System.out.println("r1's

        System.out.println("r

        // grow both rectangl
        r1.grow(50, 10);
        r2.grow(5, 30);

        System.out.println("r1: " + r1);
        System.out.println("r2: " + r2);


        System.out.println("Number of Rectangles = " +  ?   );
    }
}
```

What if you wanted to keep track of the total number of rectangles created?

Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    private static int rectanglesCreated = 0;

    public Rectangle(int w, int h) {
        setWidth(w);
        setHheight(h);

        rectanglesCreated++;
    }


    .
    .
    public void grow(int dw, int dh) {
        setWidth(width+dw);
        setHeight(height+dh);
    }
    public int area() {
        return(width*height);
    }
    public boolean equals(Rectangle other) {
        return (other != null && this.width == other.width
                              && this.height == other.height );
```

A sample Client Program

```
public class RectangleClient {

    public static void main(String[] args) {

        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        System.out.println("r1's

        System.out.println("r

        // grow both rectangl
        r1.grow(50, 10);
        r2.grow(5, 30);

        System.out.println("r1: " + r1);
        System.out.println("r2: " + r2);


        System.out.println("Number of Rectangles = " +  );
    }
}
```

How do we print it out?

?

Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    private static int rectanglesCreated = 0;

    public Rectangle(int w, int h) {
        setWidth(w);
        setHheight(h);

        rectanglesCreated++;
    }


    .
    .
    public void grow(int dw, int dh) {
        setWidth(width+dw);
        setHeight(height+dh);
    }
    public int area() {
        return(width*height);
    }
    public boolean equals(Rectangle other) {
        return (other != null && this.width == other.width
                                && this.height == other.height );
```

A sample Client Program

```
public class RectangleClient {

    public static void main(String[] args) {

        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        System.out.println("r1's area = " + r1.area() );

        System.out.println("r2's area = " + r2.area() );

        // grow both rectangles
        r1.grow(50, 10);
        r2.grow(5, 30);

        System.out.println("r1: " + r1);
        System.out.println("r2: " + r2);

        System.out.println("Number of Rectangles = " +
                                        Rectangle.rectanglesCreated );

    }
}
```

Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    public static int rectanglesCreated = 0;

    public Rectangle(int w, int h) {
        setWidth(w);
        setHheight(h);

        rectanglesCreated++;
    }


    .
    .
    public void grow(int dw, int dh) {
        setWidth(width+dw);
        setHeight(height+dh);
    }
    public int area() {
        return(width*height);
    }
    public boolean equals(Rectangle other) {
        return (other != null && this.width == other.width
                                && this.height == other.height );
```

A sample Client Program

```
public class RectangleClient {

    public static void main(String[] args) {

        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        System.out.println("r1's area = " + r1.area() );

        System.out.println("r2's area = " + r2.area() );

        // grow both rectangles
        r1.grow(50, 10);
        r2.grow(5, 30);

        System.out.println("r1: " + r1);
        System.out.println("r2: " + r2);

        System.out.println("Number of Rectangles = " +
                                        Rectangle.rectanglesCreated );
        Rectangles.rectanglesCreated = 0;
    }
}
```

Sample Rectangle Class

```
public class Rectangle {
    private int width;
    private int height;

    private static int rectanglesCreated = 0;

    public Rectangle(int w, int h) {
        setWidth(w);
        setHheight(h);

        rectanglesCreated++;
    }
    .
    .
    public static int numRectanglesCreated() {
       return( rectanglesCreated );
    }
    public void grow(int dw, int dh) {
        setWidth(width+dw);
        setHeight(height+dh);
    }
    public int area() {
        return(width*height);
    }
```

A sample Client Program

```
public class RectangleClient {

    public static void main(String[] args) {

        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        System.out.println("r1's area = " + r1.area() );

        System.out.println("r2's area = " + r2.area() );

        // grow both rectangles
        r1.grow(50, 10);
        r2.grow(5, 30);

        System.out.println("r1: " + r1);
        System.out.println("r2: " + r2);

        System.out.println("Number of Rectangles = " +
                                Rectangle.numRectanglesCreated() );

    }
}
```

A sample Client Program

```
public class RectangleClient {

    public static void main(String[] args) {
        System.out.println("Number of Rectangles = " +
                                    Rectangle.numRectanglesCreated() );

        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangl  0, 80);

        System.out.println("r1's area  "

        System.out.println("r2's ar

        // grow both rectangles
        r1.grow(50, 10);
        r2.grow(5, 30);

        System.out.println("r1: " + r1);
        System.out.println("r2: " + r2);

        System.out.pri  r of Rectangles = " +
                                    Rectangle.numRectanglesCreated() );
    }
}
```

It is a static variable and,
it can be accessed even before
any objects are created!

A sample Client Program:

memory trace

```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + r2.area() );
      // grow both rectangles
      r1.grow(50, 10);
      r2.grow(5, 30);
      System.out.println("r1: " + r1);
      System.out.println("r2: " + r2);
      ...
}
```

Memory Stack **Memory Heap** **Static**

Rectangles Created	0

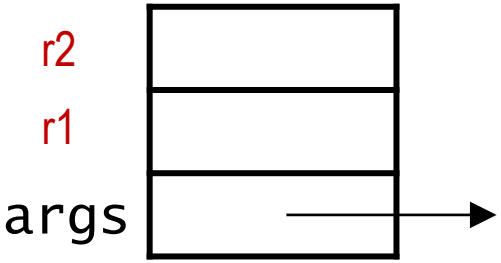
A sample Client Program:

memory trace

```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + r2.area() );
      // grow both rectangles
      r1.grow(50, 10);
      r2.grow(5, 30);
      System.out.println("r1: " + r1);
      System.out.println("r2: " + r2);
      ...
}
```

Memory Stack **Memory Heap** **Static**

Rectangles Created	0



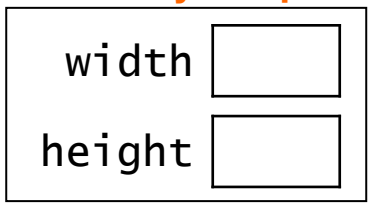
A sample Client Program:

memory trace

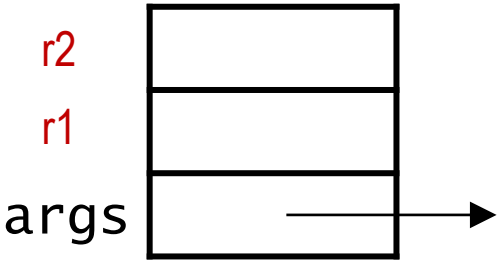
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + r2.area() );
      // grow both rectangles
      r1.grow(50, 10);
      r2.grow(5, 30);
      System.out.println("r1: " + r1);
      System.out.println("r2: " + r2);
      ...
}
```

Memory Stack

Memory Heap



Static



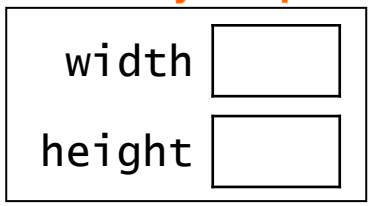
A sample Client Program:

memory trace

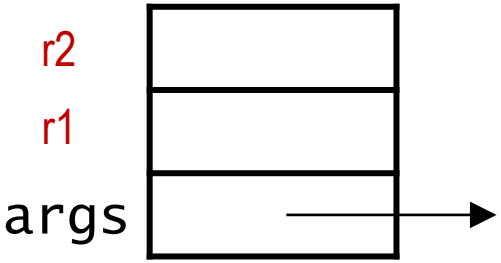
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + r2.area() );
      // grow both rectangles
      r1.grow(50, 10);
      r2.grow(5, 30);
      System.out.println("r1: " + r1);
      System.out.println("r2: " + r2);
      ...
}
```

Memory Stack

Memory Heap



Static

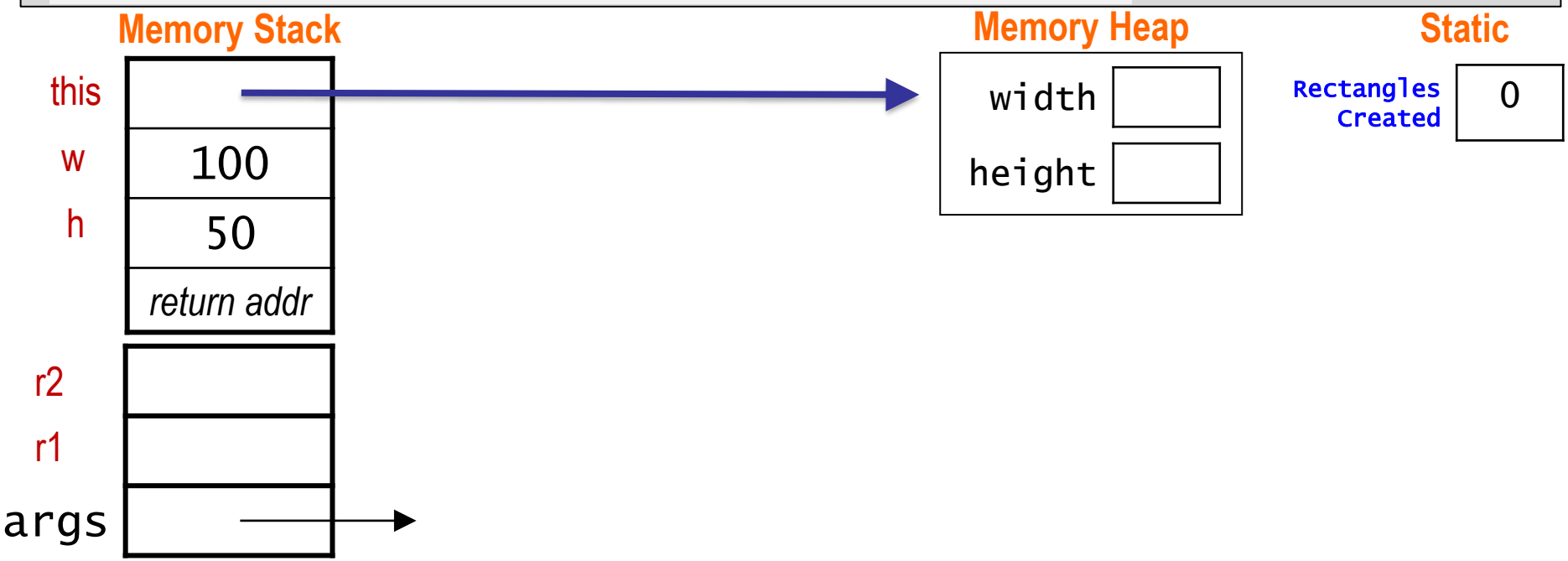


A sample Client Program:

memory trace

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        this.width = w;
        this.height = h;
        rectanglesCreated++;
    }
}
```

Note: for purposes of this example, showing the simplified method that directly updates the variables.

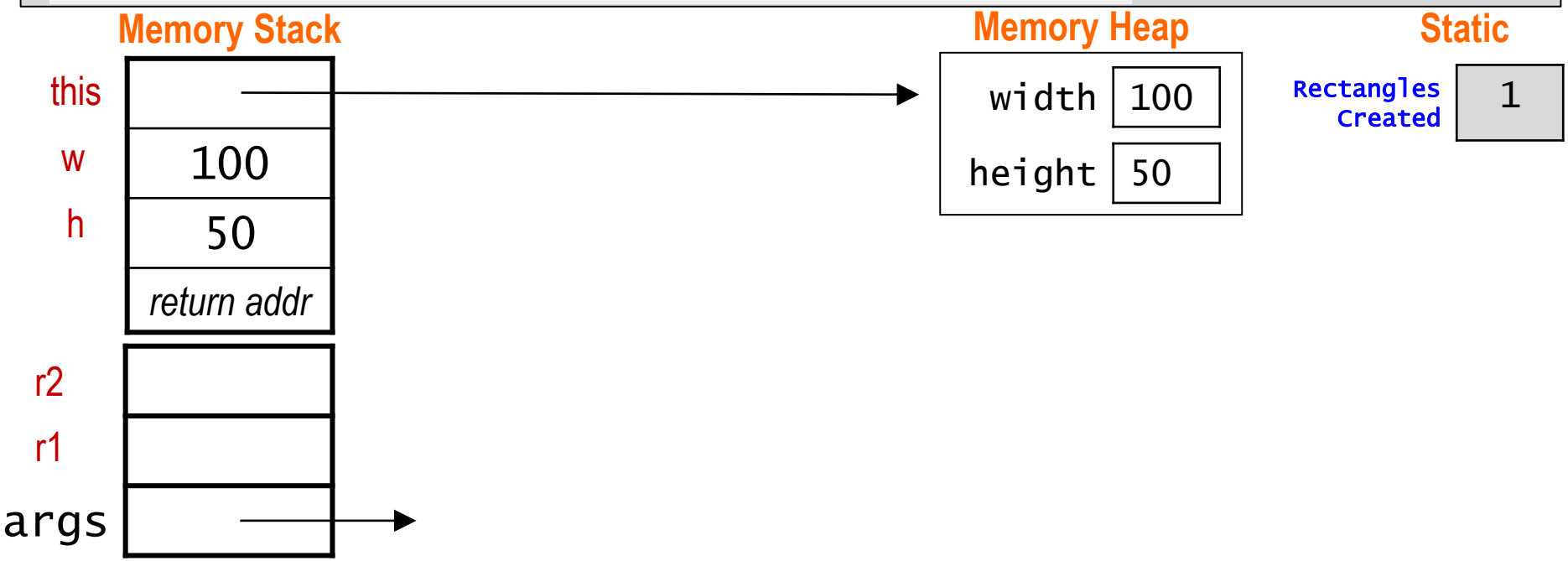


A sample Client Program:

memory trace

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
        rectanglesCreated++;
    }
}
```

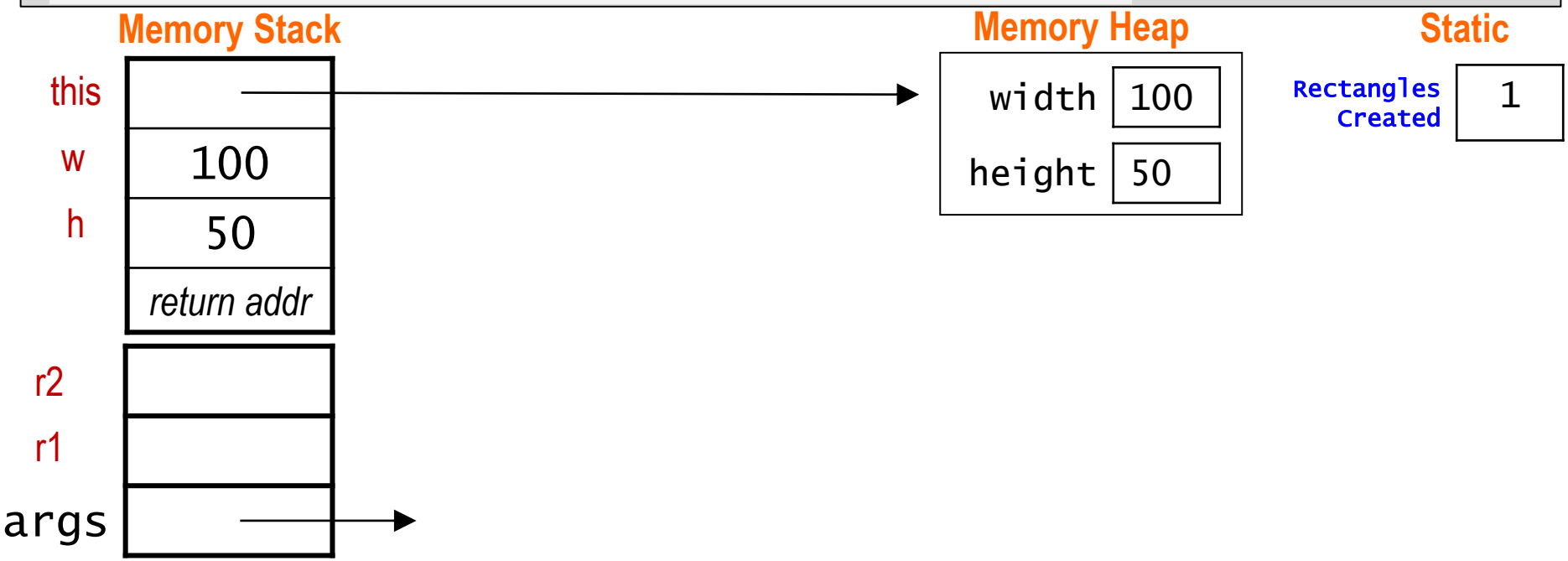
Note: for purposes of this example, showing the simplified method that directly updates the variables.



A sample Client Program:

memory trace

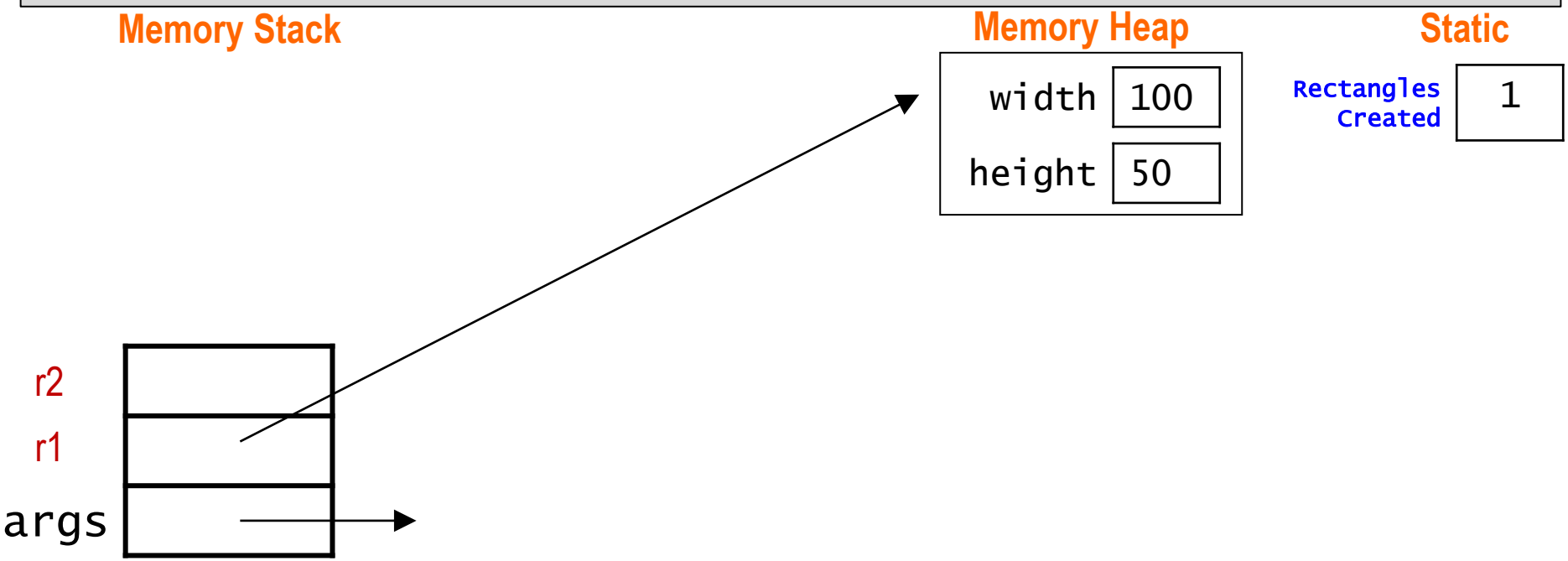
```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
        rectanglesCreated++;
    } // returning from method
}    .
```



A sample Client Program:

memory trace

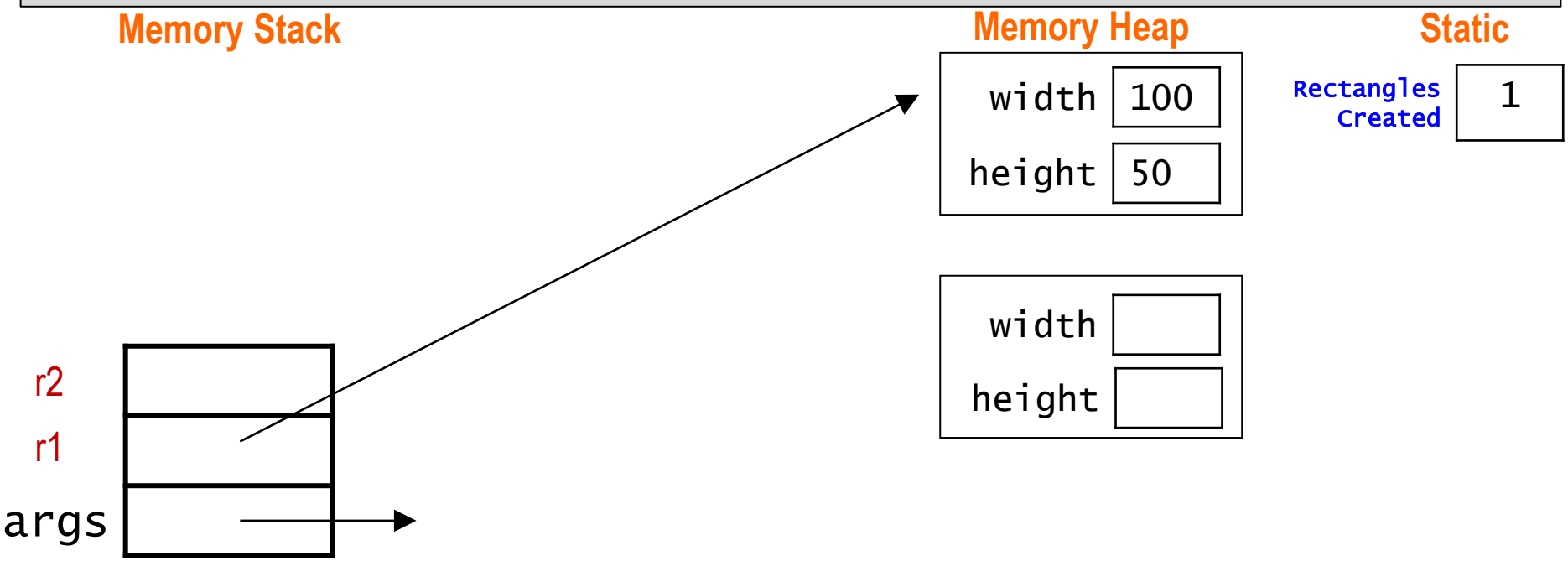
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + r2.area() );
      // grow both rectangles
      r1.grow(50, 10);
      r2.grow(5, 30);
      System.out.println("r1: " + r1);
      System.out.println("r2: " + r2);
      ...
}
```



A sample Client Program:

memory trace

```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + r2.area() );
      // grow both rectangles
      r1.grow(50, 10);
      r2.grow(5, 30);
      System.out.println("r1: " + r1);
      System.out.println("r2: " + r2);
      ...
}
```

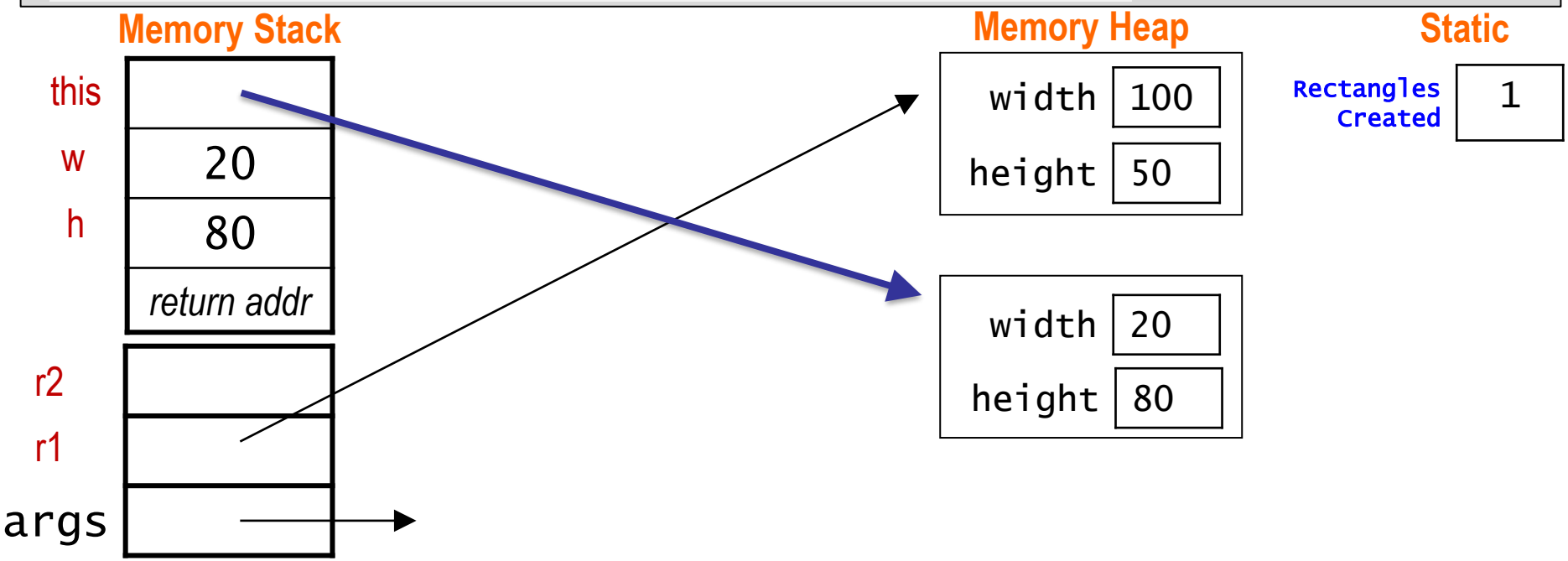


A sample Client Program:

memory trace

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
        rectanglesCreated++;
    }
}
```

Note: for purposes of this example, showing the simplified method that directly updates the variables.

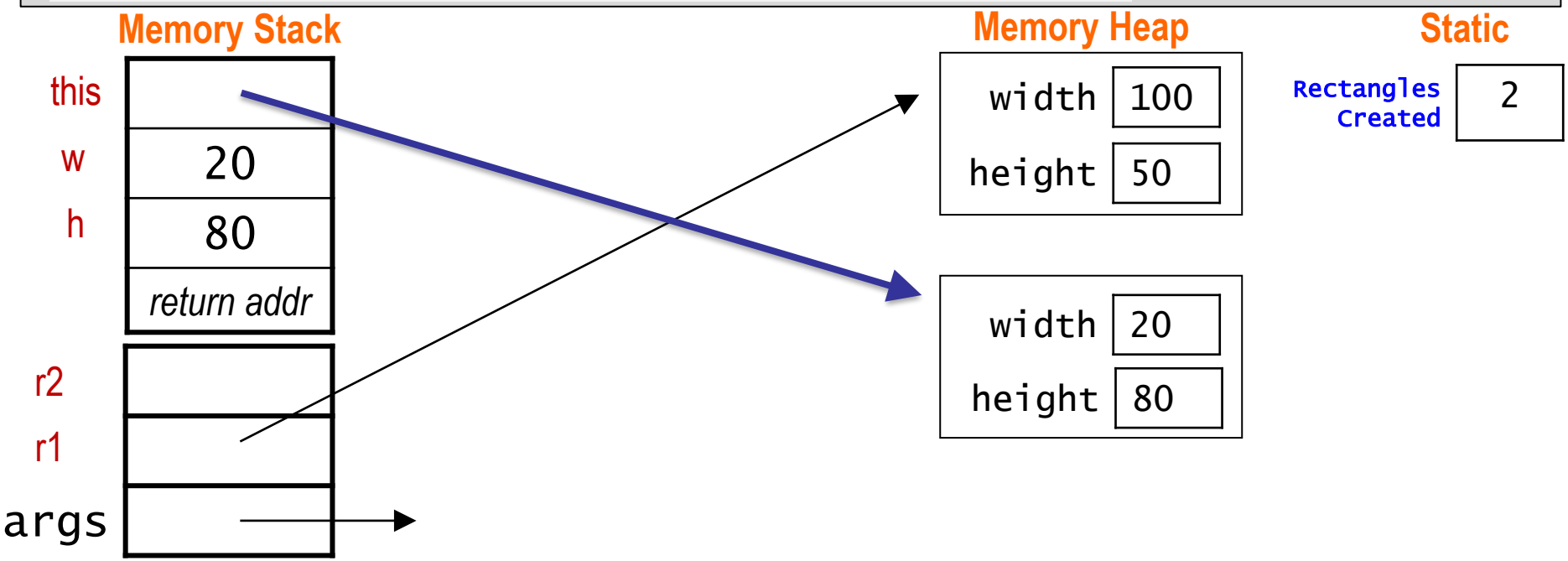


A sample Client Program:

memory trace

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
        rectanglesCreated++;
    }
}
```

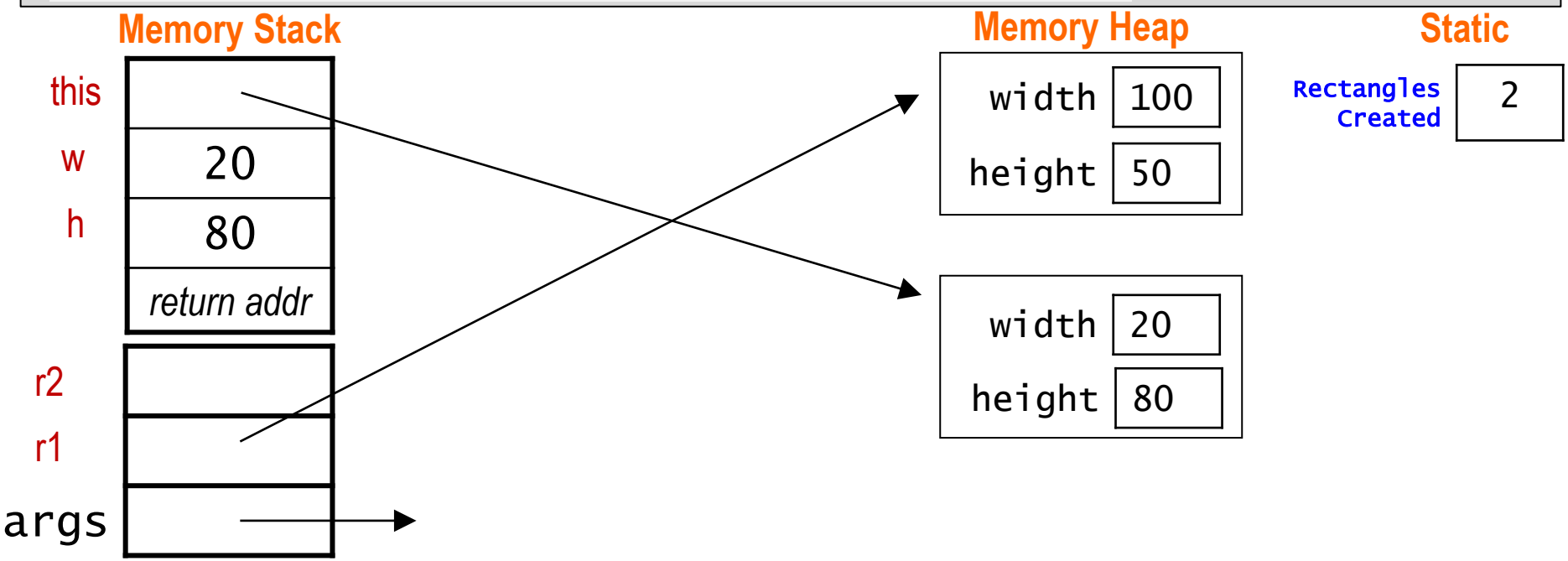
Note: for purposes of this example, showing the simplified method that directly updates the variables.



A sample Client Program:

memory trace

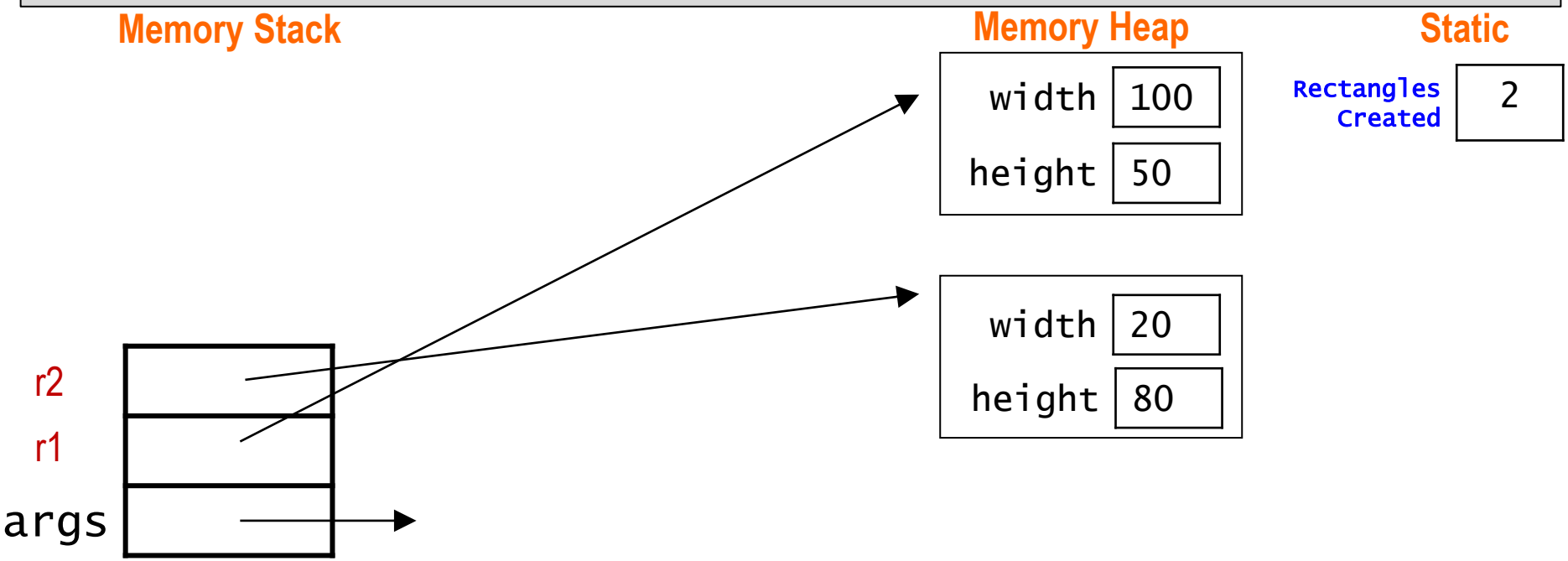
```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        width = w;
        height = h;
        rectanglesCreated++;
    } // return from method
}    .
```



A sample Client Program:

memory trace

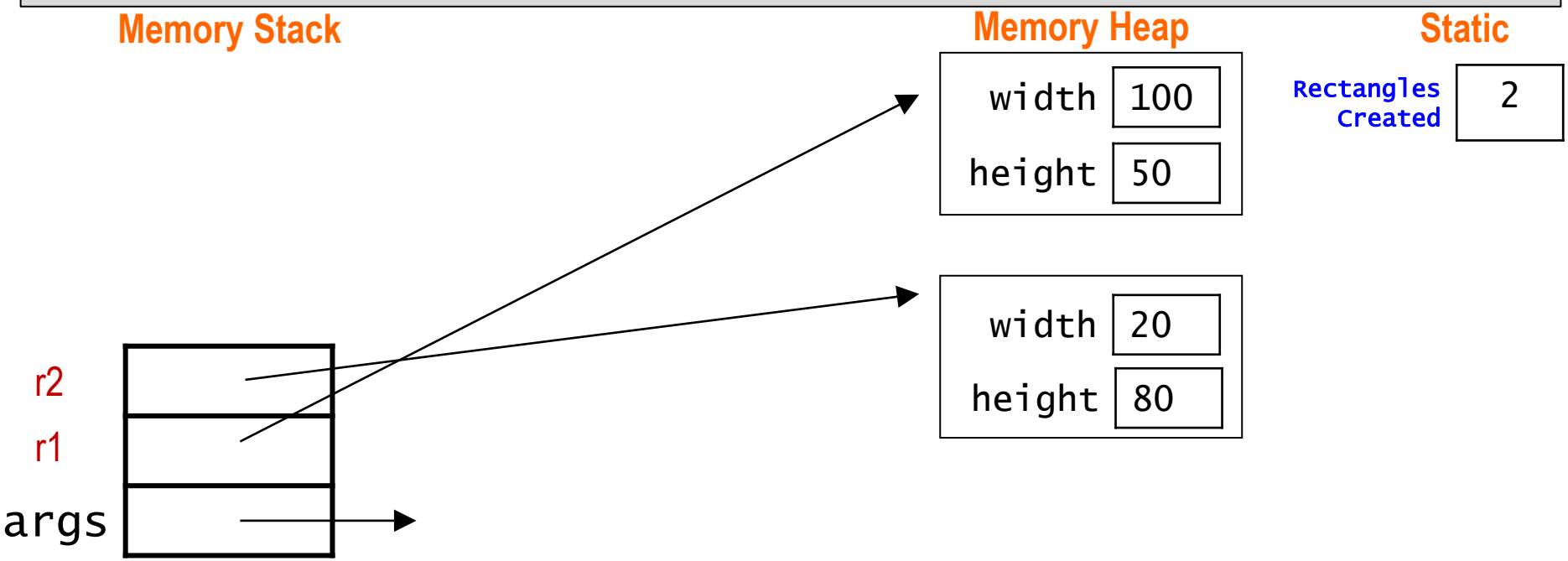
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + r2.area() );
      // grow both rectangles
      r1.grow(50, 10);
      r2.grow(5, 30);
      System.out.println("r1: " + r1);
      System.out.println("r2: " + r2);
      ...
}
```



A sample Client Program:

memory trace

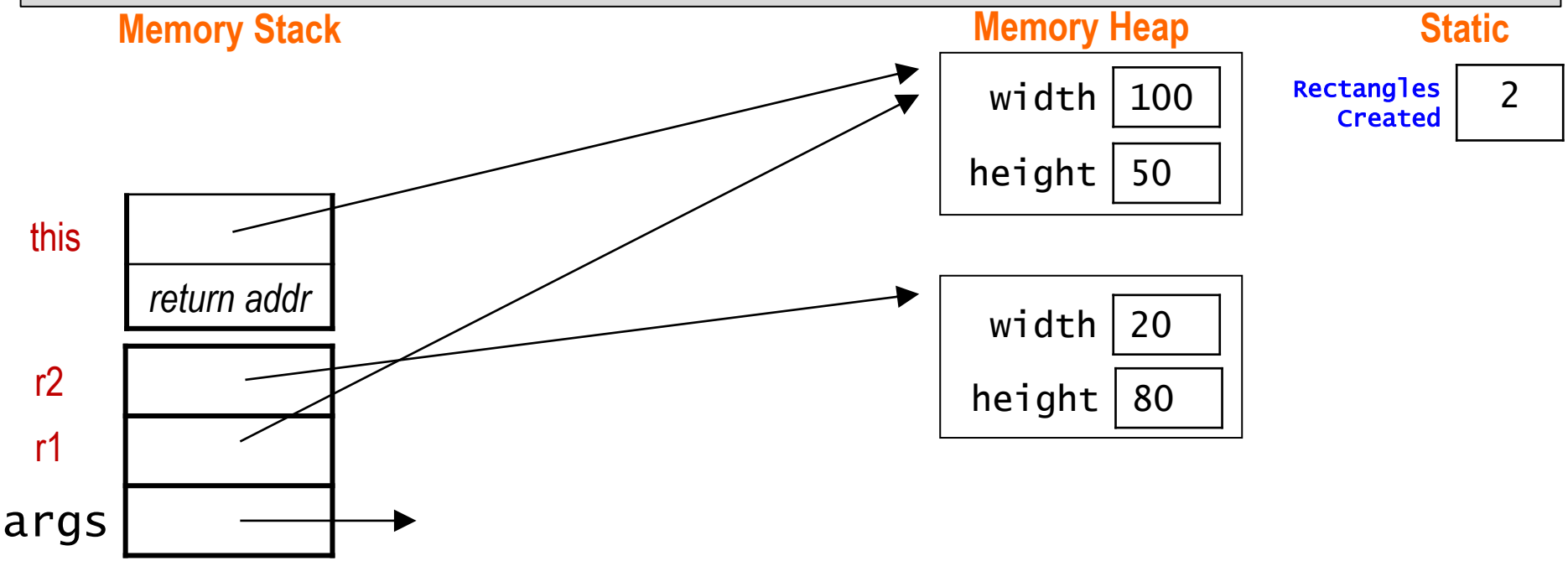
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + r2.area() );
      // grow both rectangles
      r1.grow(50, 10);
      r2.grow(5, 30);
      System.out.println("r1: " + r1);
      System.out.println("r2: " + r2);
      ...
}
```



A sample Client Program:

memory trace

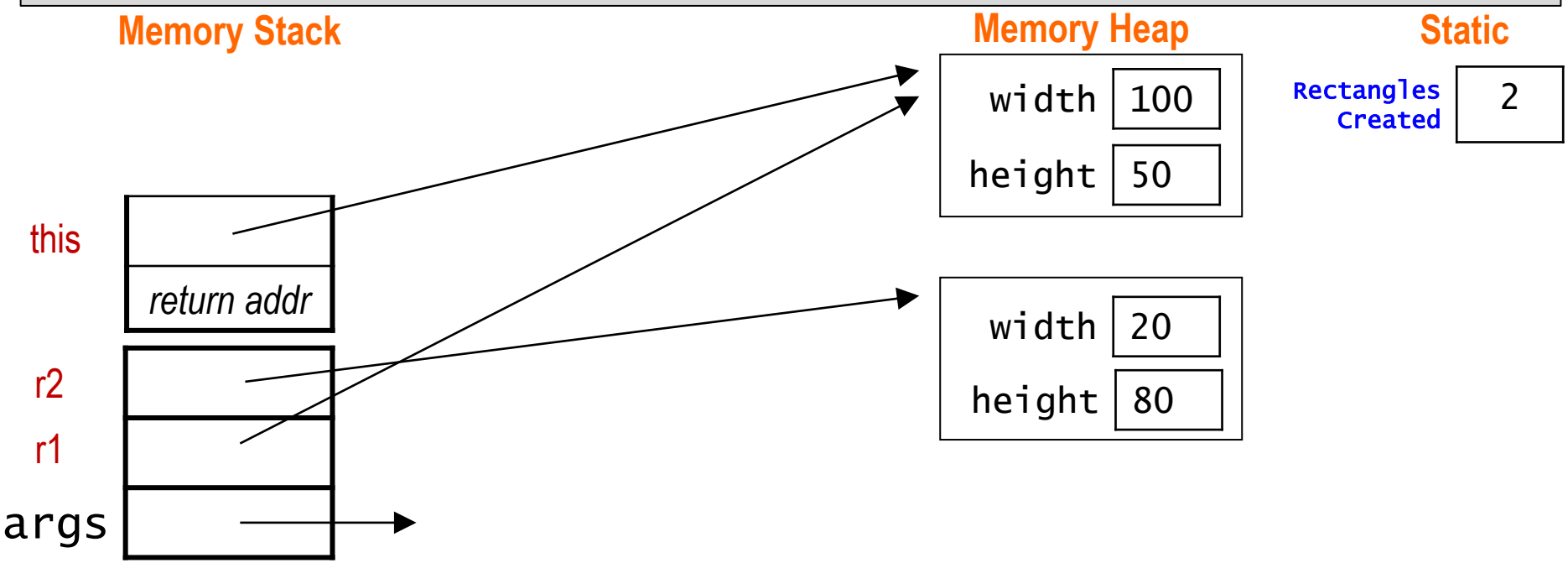
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public int area() {
        return( width*height );
    }
    .
}
```



A sample Client Program:

memory trace

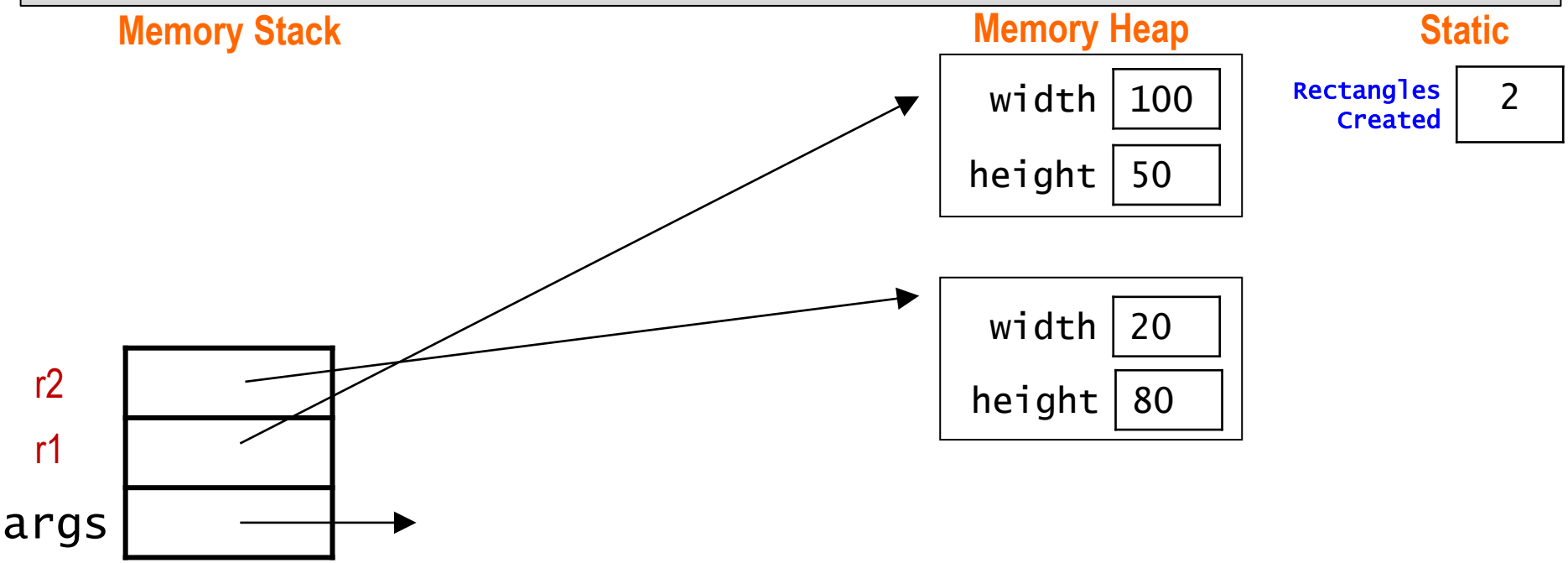
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public int area() {
        return( this.width*this.height ); // returns 5000
    }
    .
}
```



A sample Client Program:

memory trace

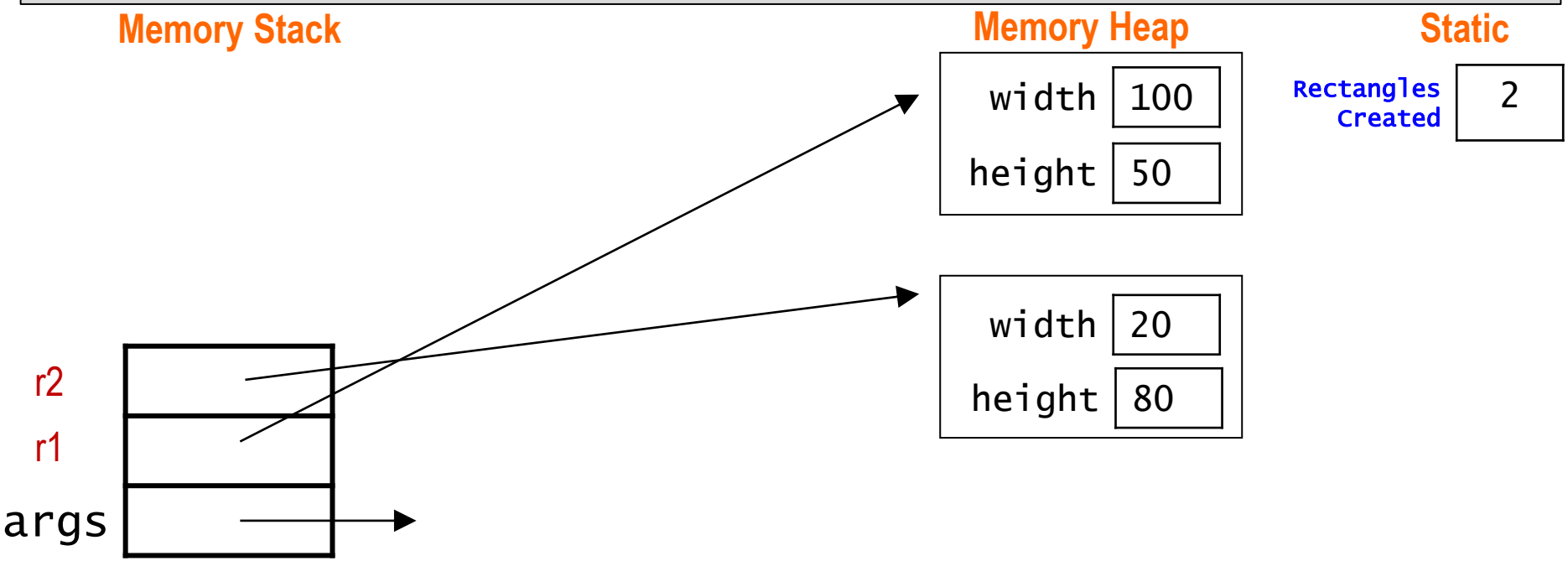
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + 5000 ); // outputs area = 5000
      System.out.println("r2's area = " + r2.area() );
      // grow both rectangles
      r1.grow(50, 10);
      r2.grow(5, 30);
      System.out.println("r1: " + r1);
      System.out.println("r2: " + r2);
      ...
}
```



A sample Client Program:

memory trace

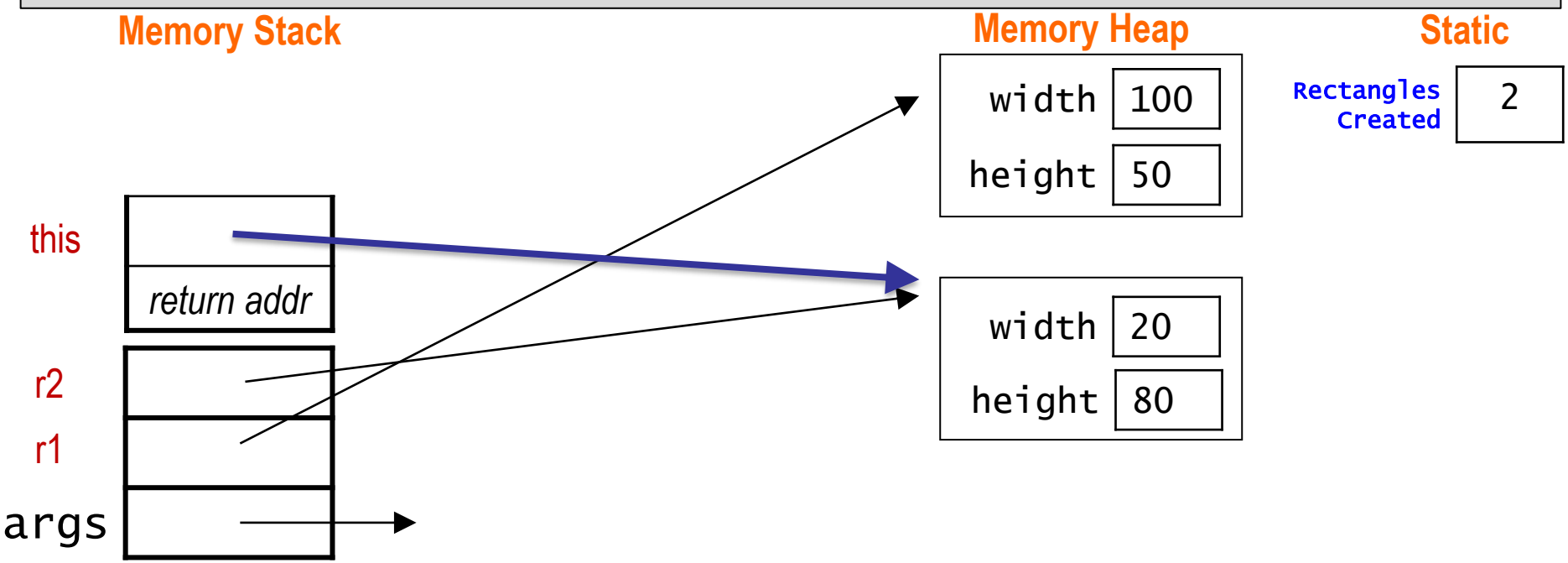
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + 5000 ); // outputs area = 5000
      System.out.println("r2's area = " + r2.area() );
      // grow both rectangles
      r1.grow(50, 10);
      r2.grow(5, 30);
      System.out.println("r1: " + r1);
      System.out.println("r2: " + r2);
      ...
}
```



A sample Client Program:

memory trace

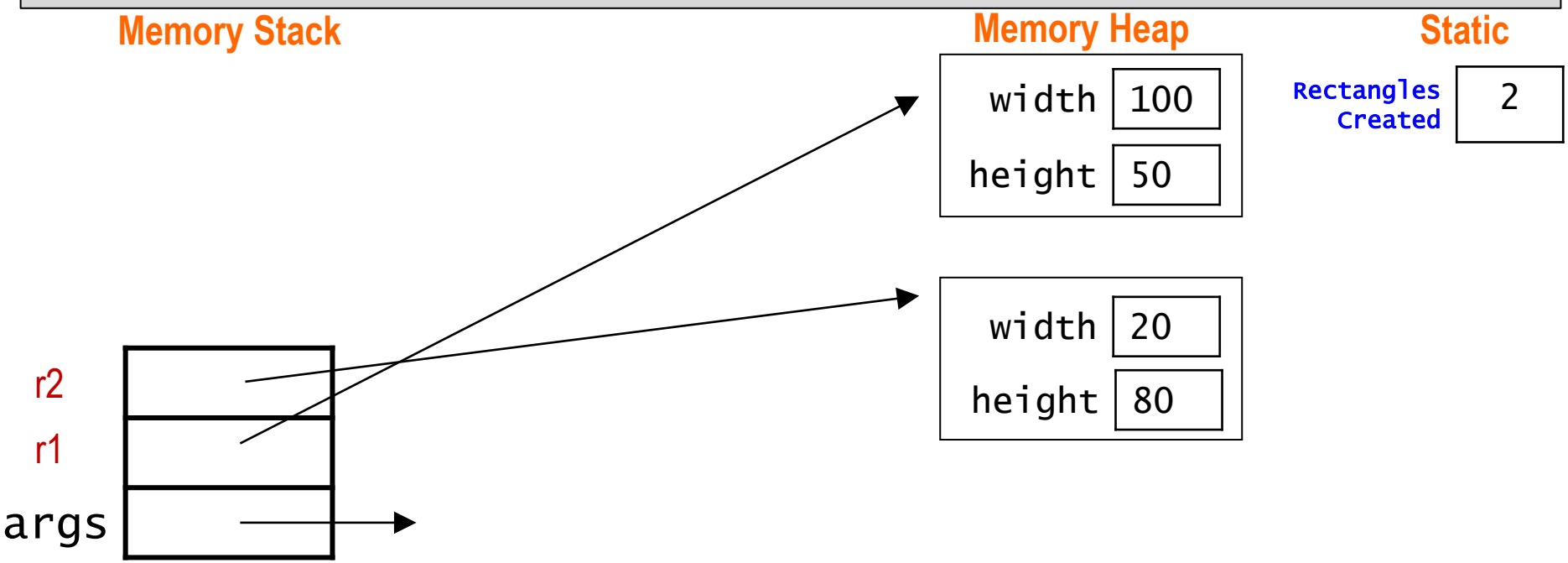
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public int area() {
        return( width*height ); // returns 1600
    }
    .
}
```



A sample Client Program:

memory trace

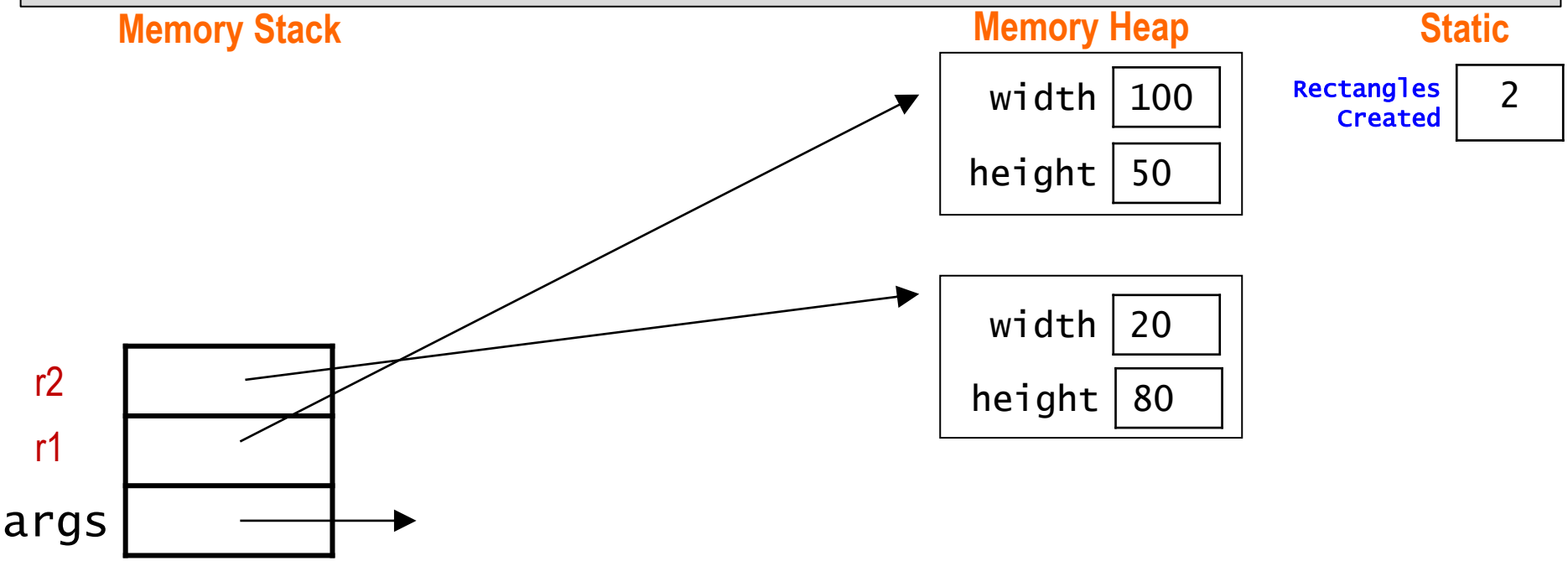
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600
      // grow both rectangles
      r1.grow(50, 10);
      r2.grow(5, 30);
      System.out.println("r1: " + r1);
      System.out.println("r2: " + r2);
      ...
}
```



A sample Client Program:

memory trace

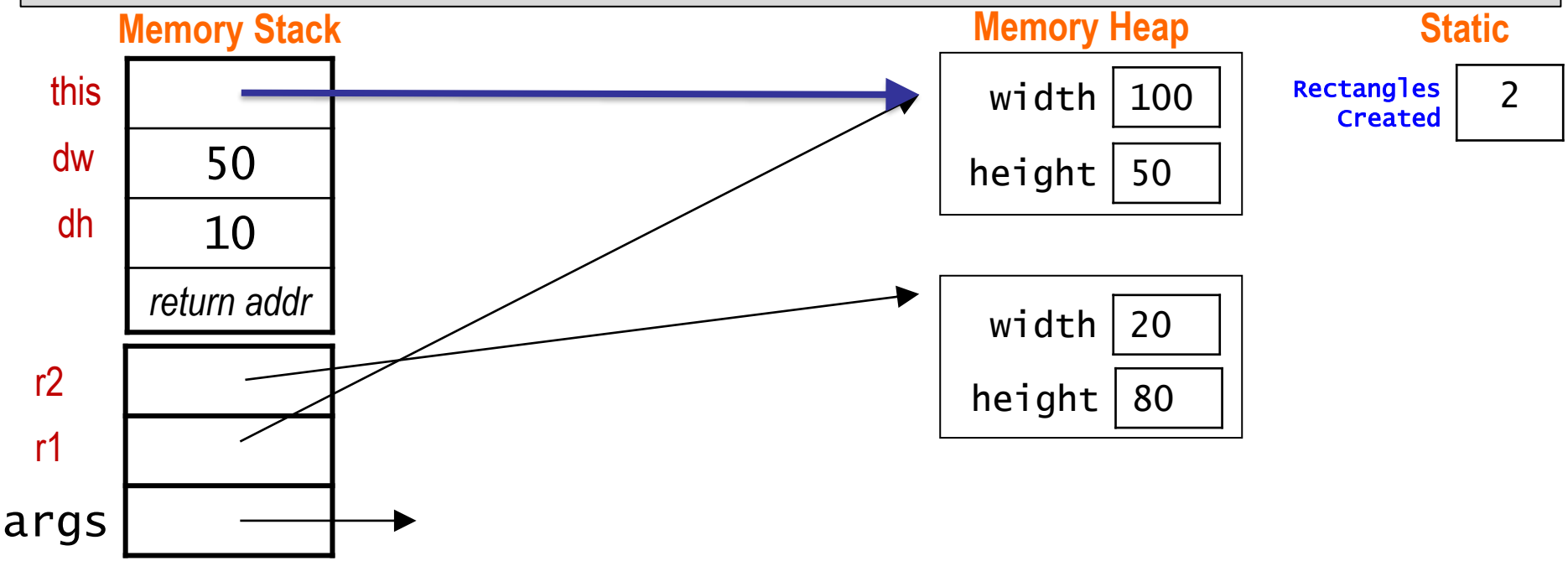
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600
      // grow both rectangles
      r1.grow(50, 10);
      r2.grow(5, 30);
      System.out.println("r1: " + r1);
      System.out.println("r2: " + r2);
      ...
}
```



A sample Client Program:

memory trace

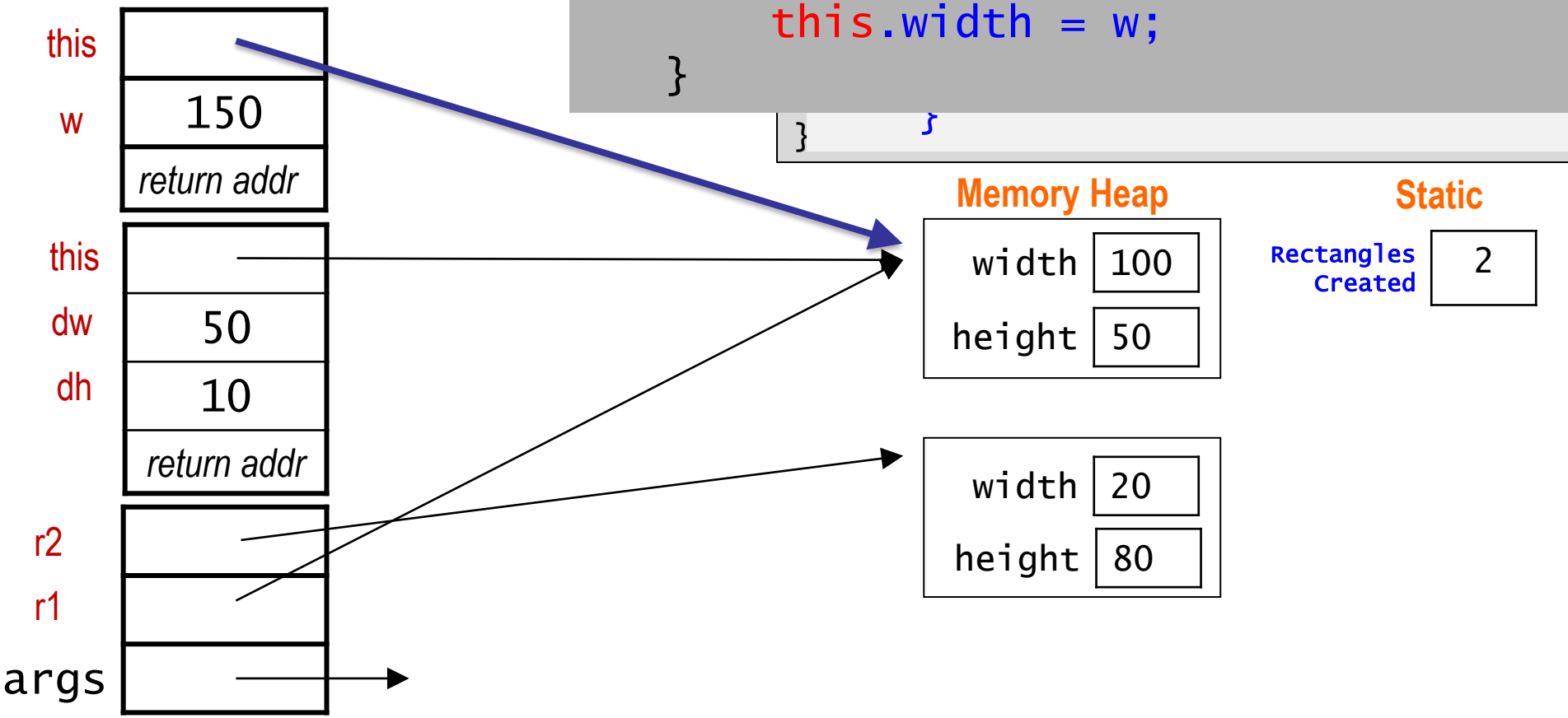
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public void grow( int dw, int dh ) {
        this.setWidth( width+dw );
        setHeight( height+dh );
    }
}
```



A sample Client Program:

memory trace

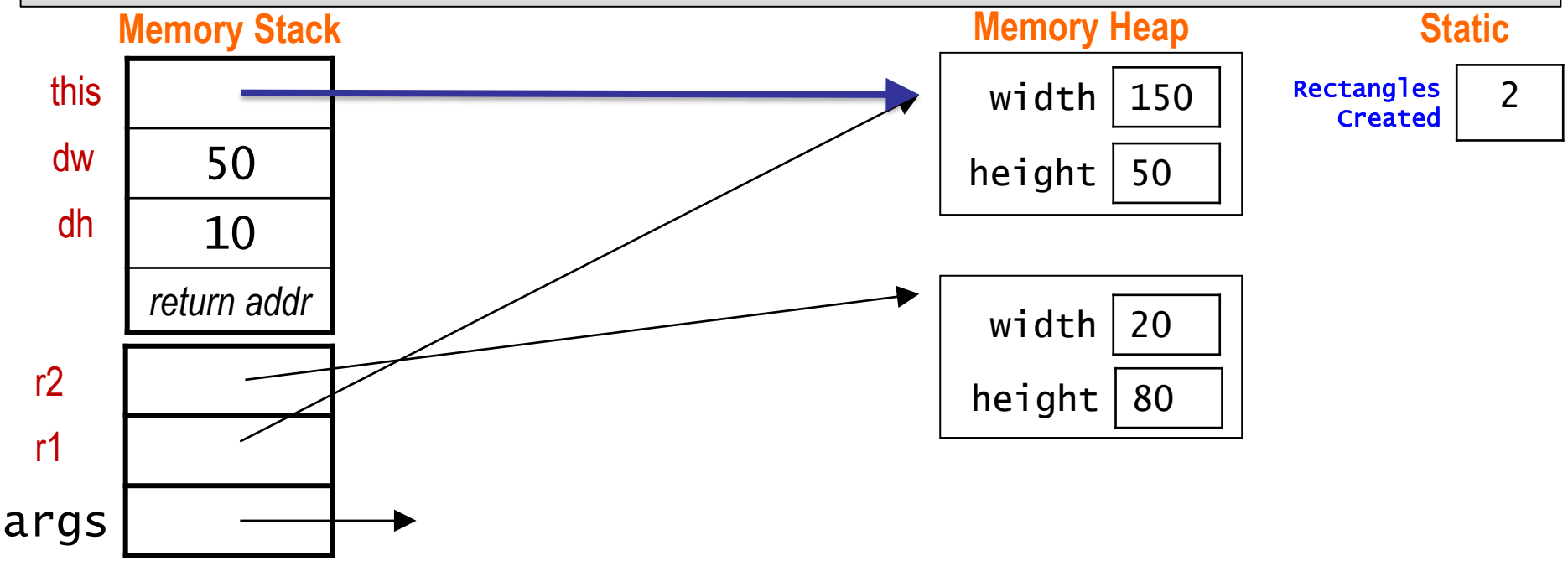
```
public class Rectangle {
    public void setWidth(int w) {
        if (w <= 0) {
            throw new IllegalArgumentExc
        }

        this.width = w;
    }
}
```



A sample Client Program:

memory trace

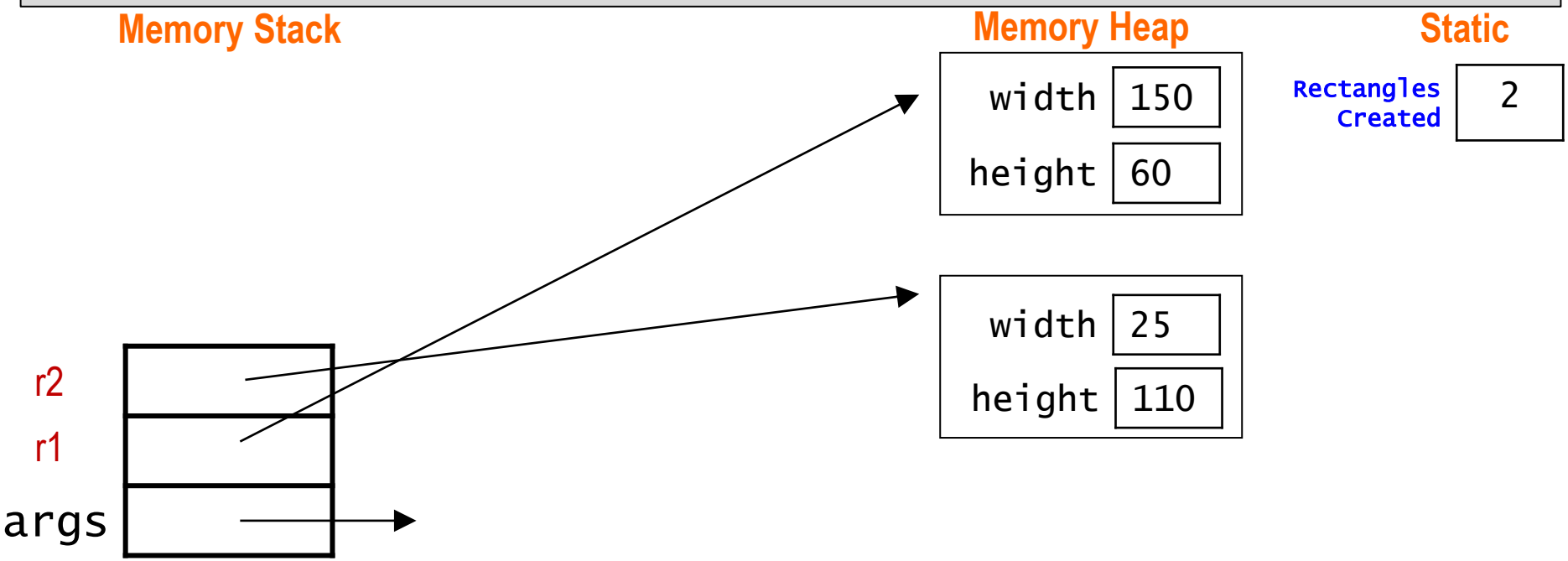
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public void grow( int dw, int dh ) {
        setWidth( width+dw );
        setHeight( height+dh );
    }
}
```



A sample Client Program:

memory trace

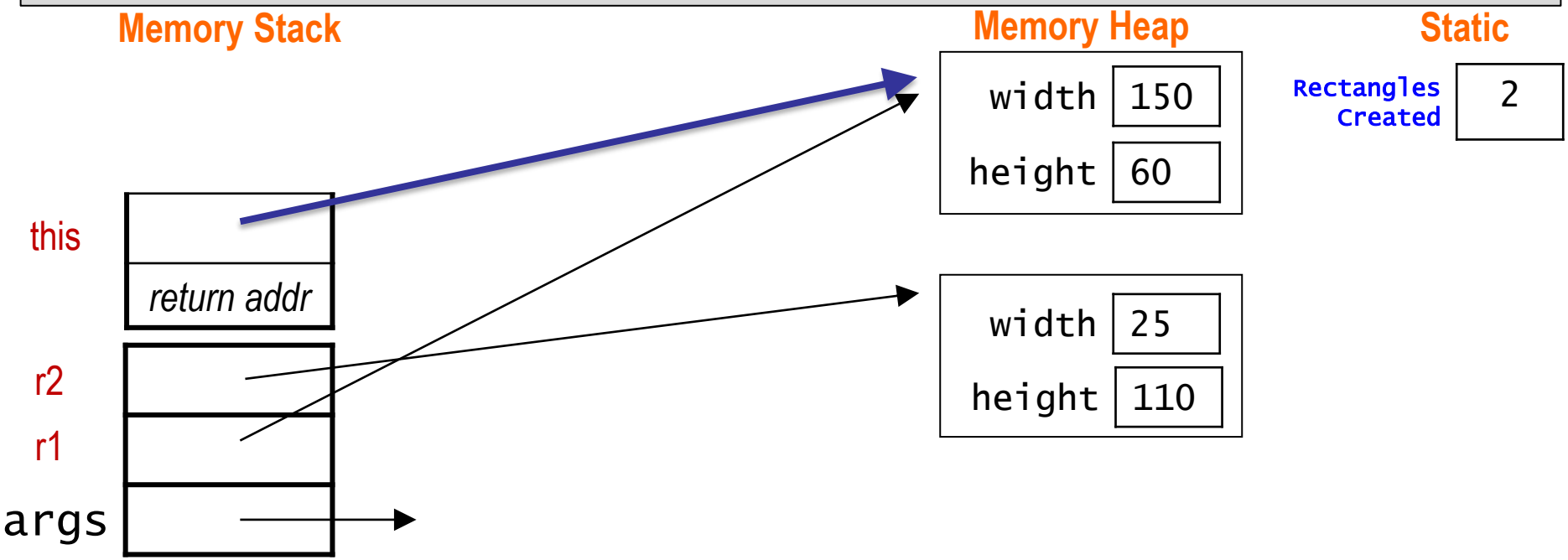
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600
      // grow both rectangles
      r1.grow(50, 10);
      r2.grow(5, 30);
      System.out.println("r1: " + r1.toString());
      System.out.println("r2: " + r2);
      ...
}
```



A sample Client Program:

memory trace

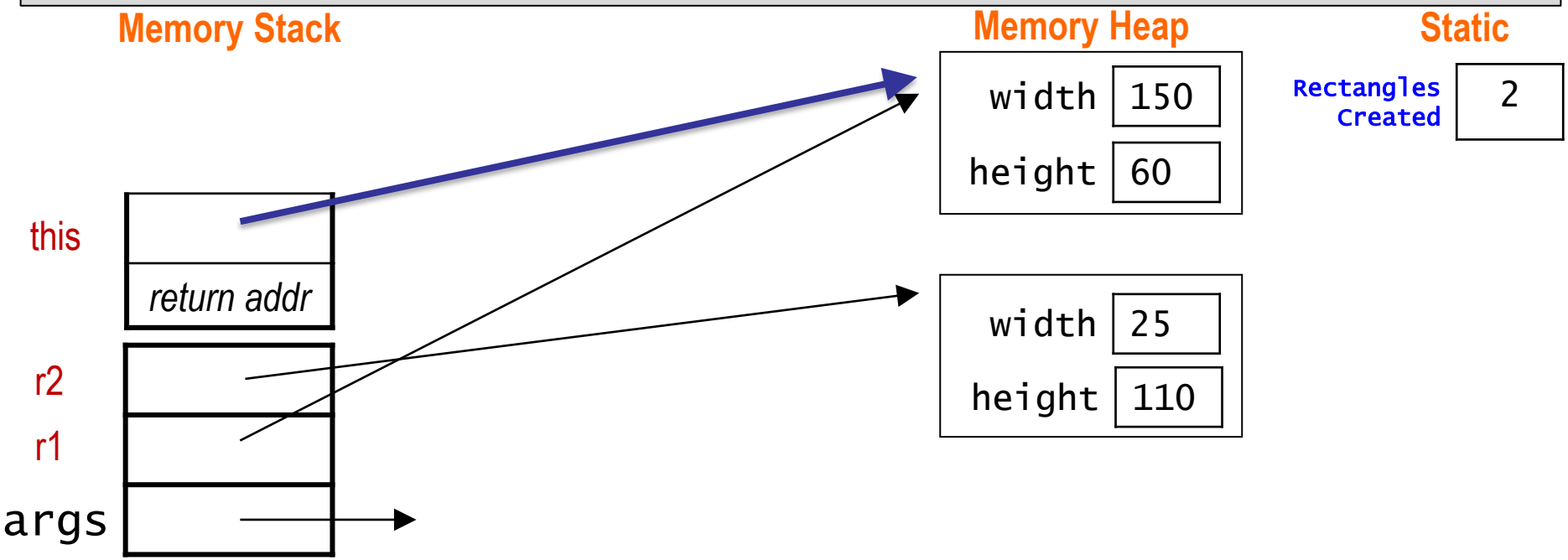
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public String toString() {
        return this.width + " x " + this.height;
    }
  }
}
```



A sample Client Program:

memory trace

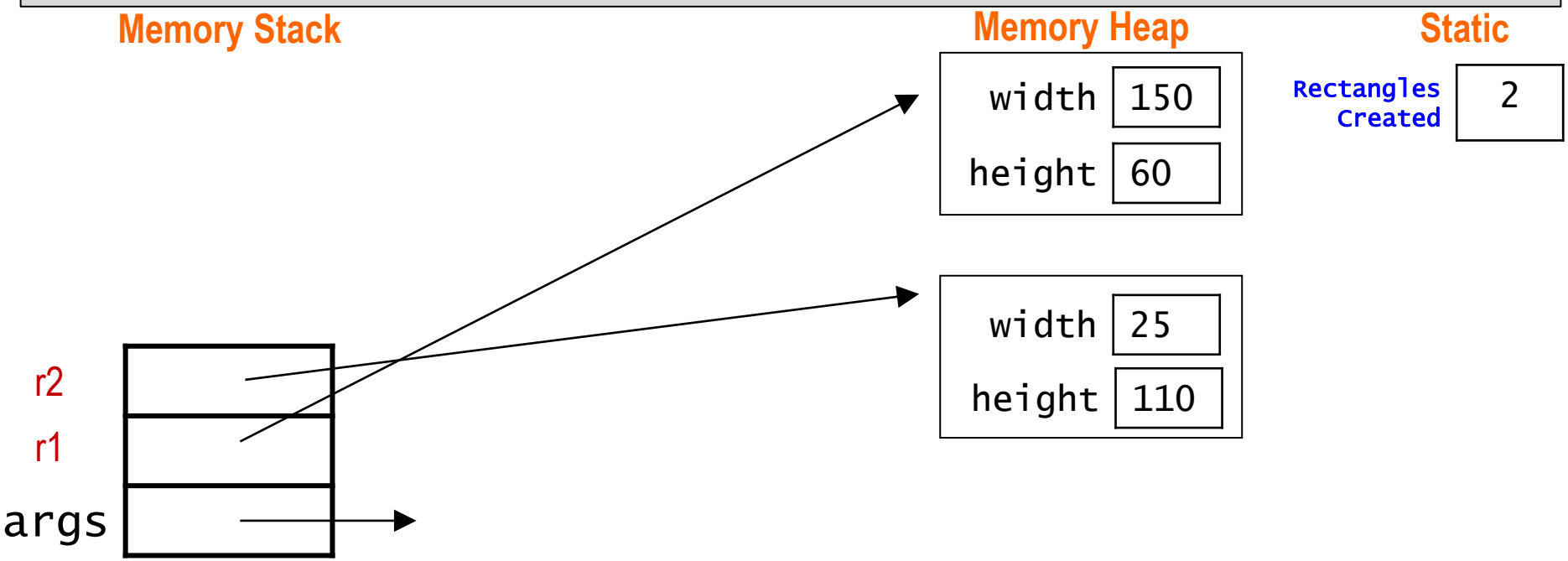
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public String toString() {
        return width + " x " + height; // returns “150 x 60"
    }
  }
}
```



A sample Client Program:

memory trace

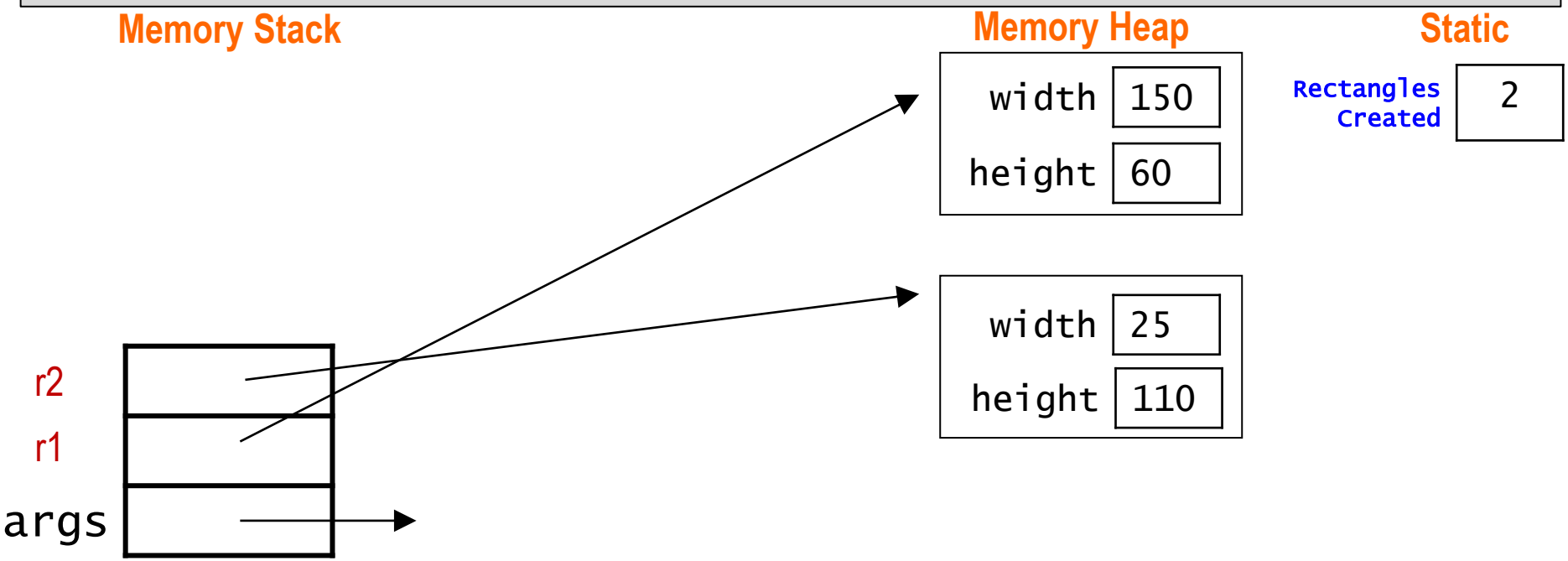
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600
      // grow both rectangles
      r1.grow(50, 10);
      r2.grow(5, 30);
      System.out.println("r1: " + r1); // outputs r1: 150 x 60
      System.out.println("r2: " + r2);
      ...
}
```



A sample Client Program:

memory trace

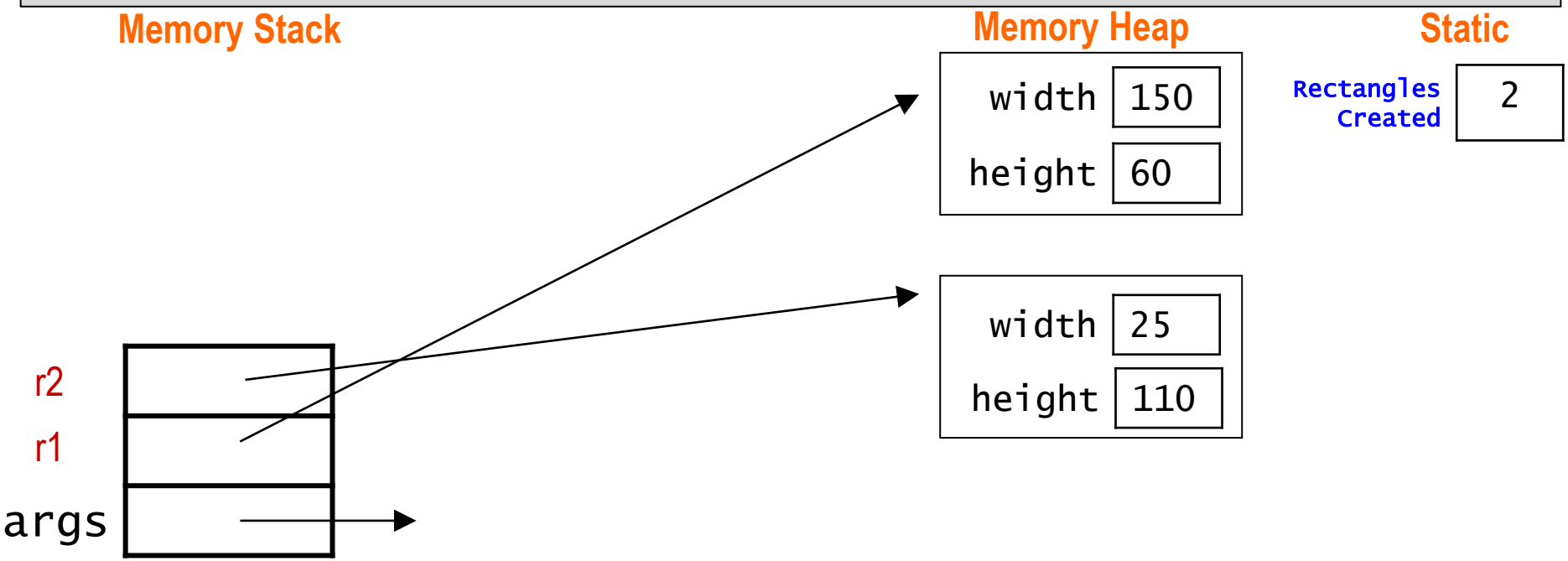
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600
      // grow both rectangles
      r1.grow(50, 10);
      r2.grow(5, 30);
      System.out.println("r1: " + r1);
      System.out.println("r2: " + r2); // output r2: 25 x 110
      ...
}
```



A sample Client Program:

memory trace

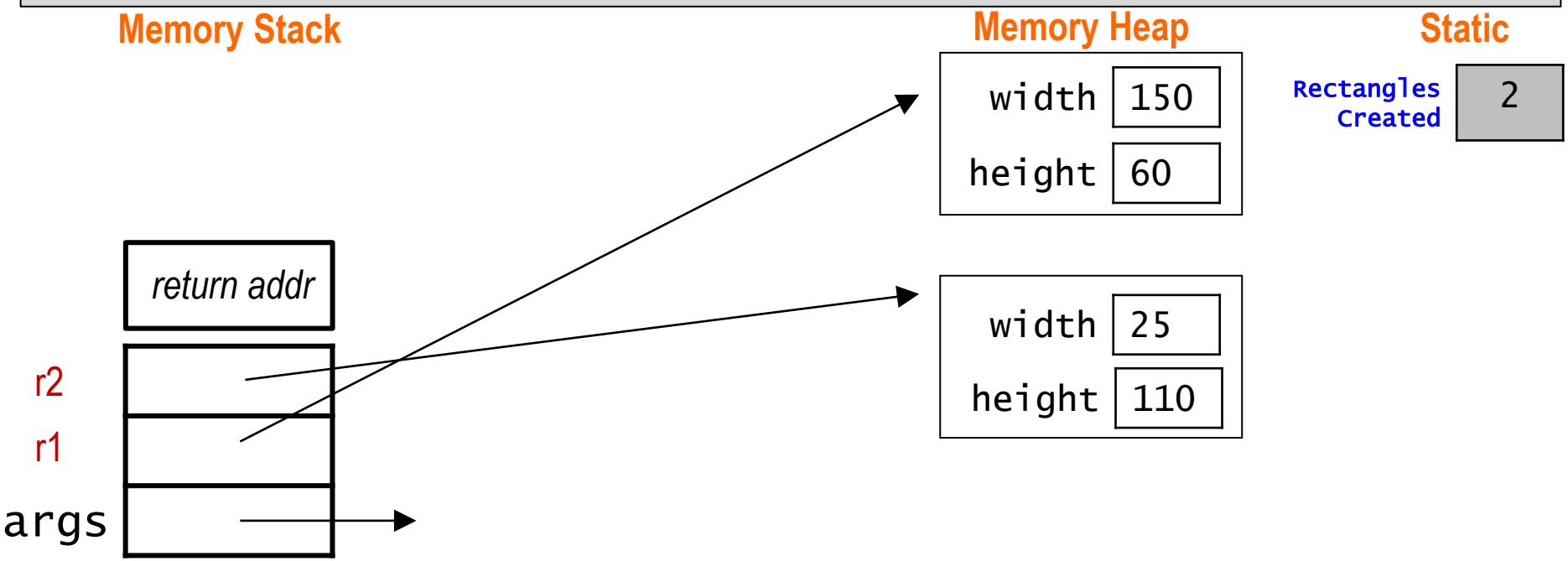
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600

      ...

      System.out.println("Number of Rectangles = " +
                                    Rectangle.numRectanglesCreated() );
}
```



A sample Client Program:

memory trace

```
public class Rectangle {
    private int width;
    private int height;
    private static int rectanglesCreated;
    .
    .
    public static int numRectanglesCreated() {
        return( rectanglesCreated ); // return 2
    }
}
```

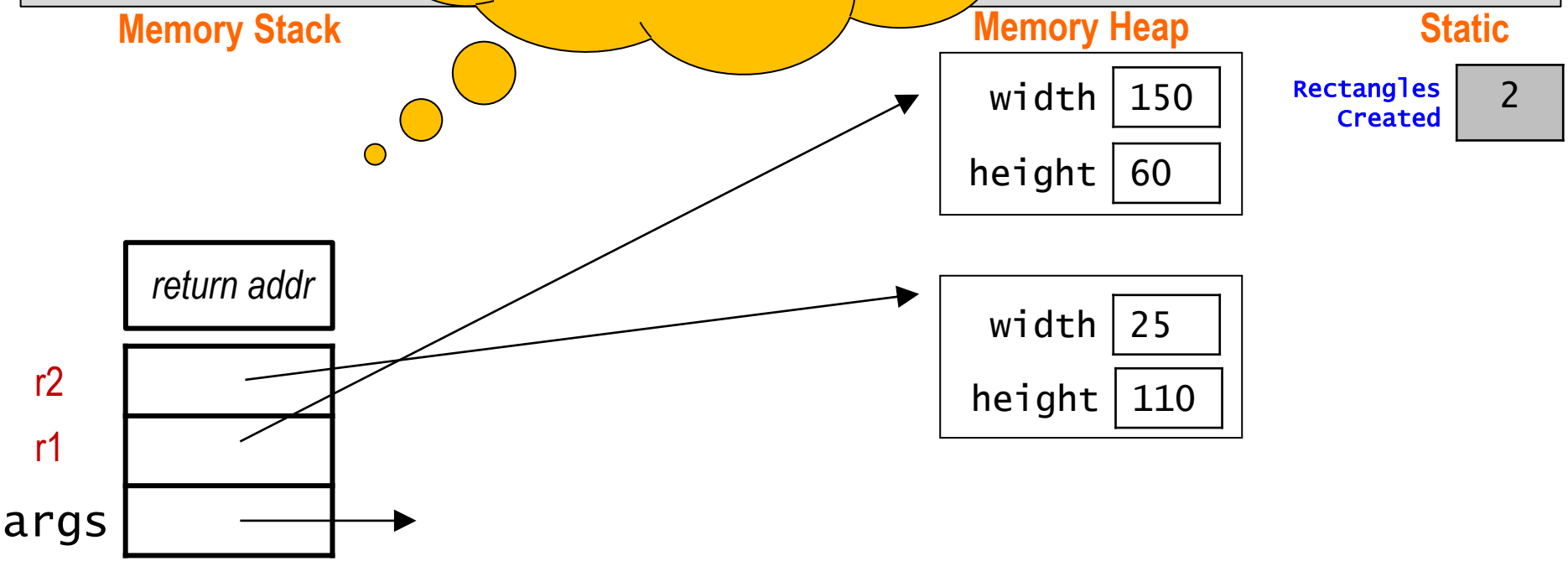


A sample Client Program:

memory trace

```
public class Rectangle {
    private int width;
    private int height;
    private static int rectanglesCreated;
    .
    .
    public sta                    l() {
        return                    urn 2
    }
}
```

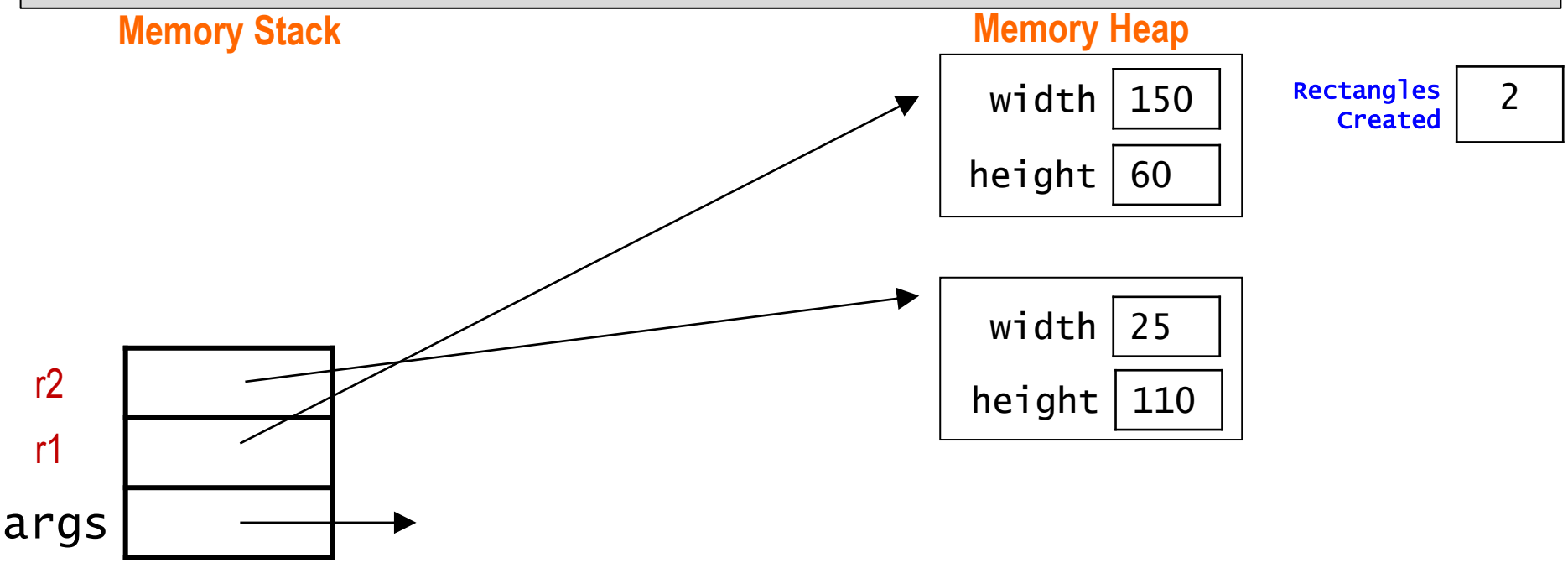
*Note the absence of the **this** reference!*



A sample Client Program:

memory trace

```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600

      ...

      System.out.println("Number of Rectangles = " +
                                    Rectangle.numRectanglesCreated() ); // 2
} // end of the method
```

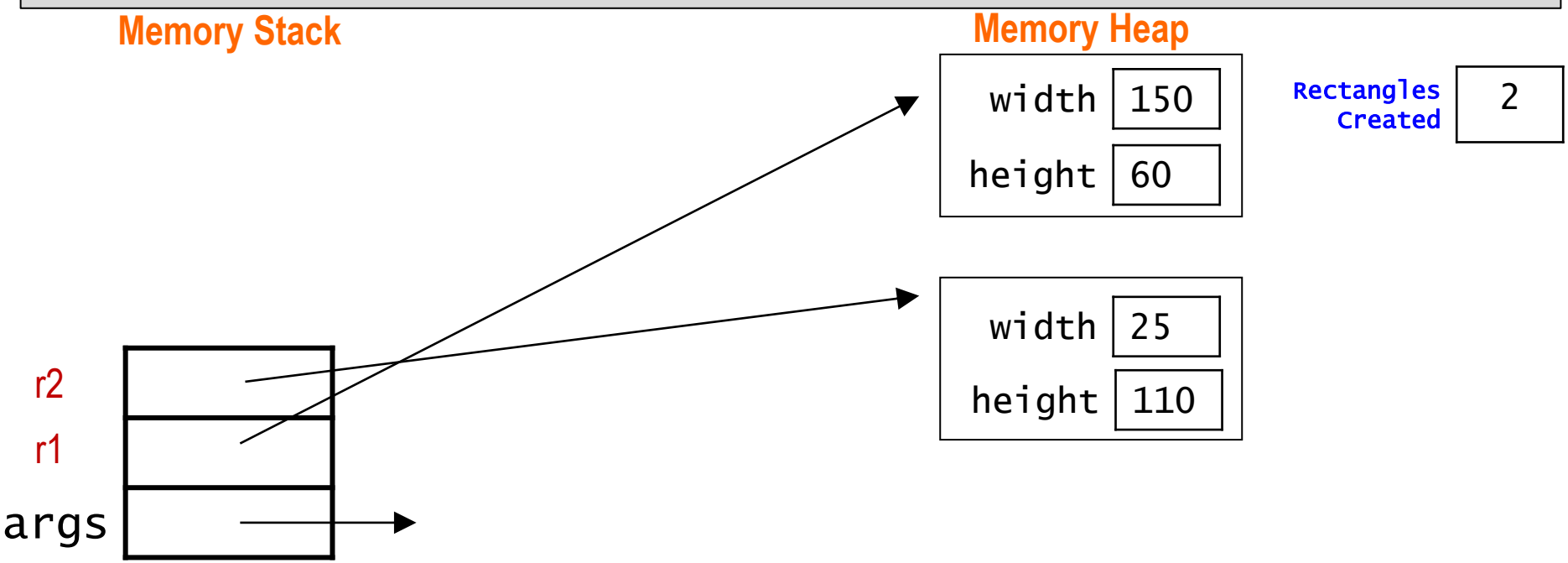


A sample Client Program

memory trace

```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.a
      System.out.println("r2's area = " + 1600 ); //

      ...

      System.out.println("Number of Rectangles = " +
                                 r1.numRectanglesCreated() ); // ?
} // end of the method
```

*What if the call to the static method was made **through** the object?*

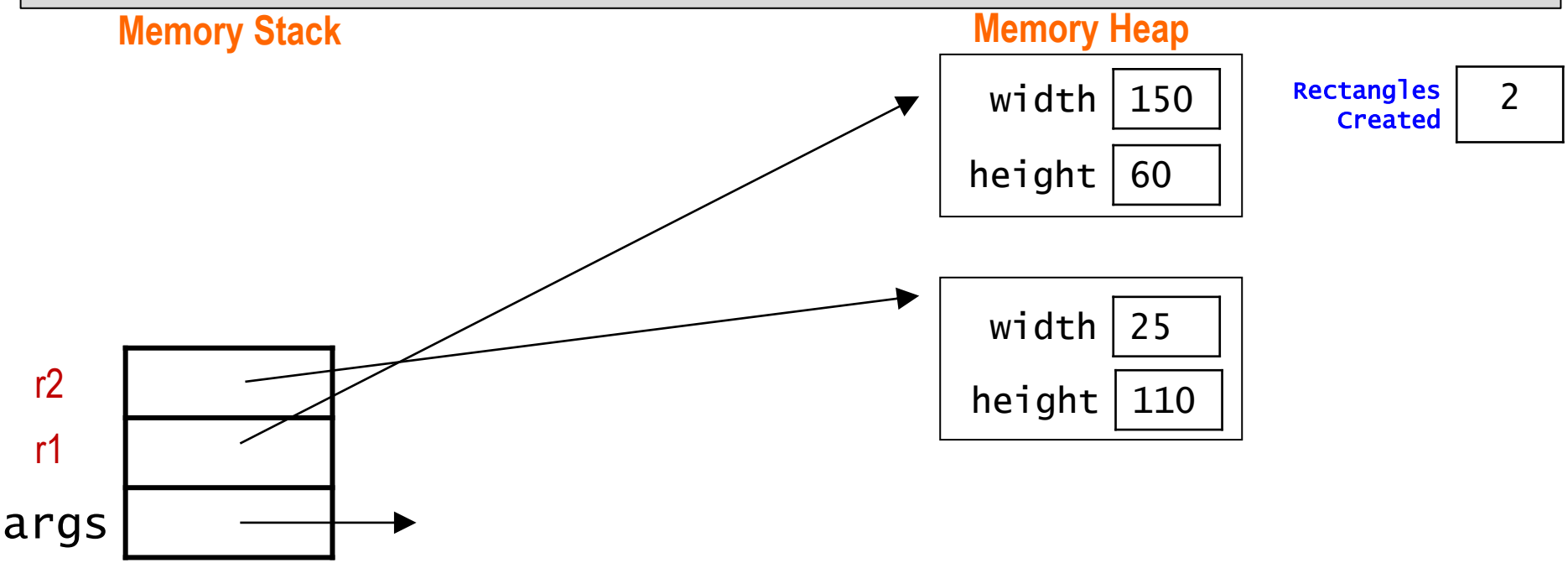


A sample Client Program

memory trace

```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.a
      System.out.println("r2's area = " + 1600 ); //

      ...

      System.out.println("Number of Rectangles = " +
                                 r1.numRectanglesCreated() ); // 2
} // end of the method
```

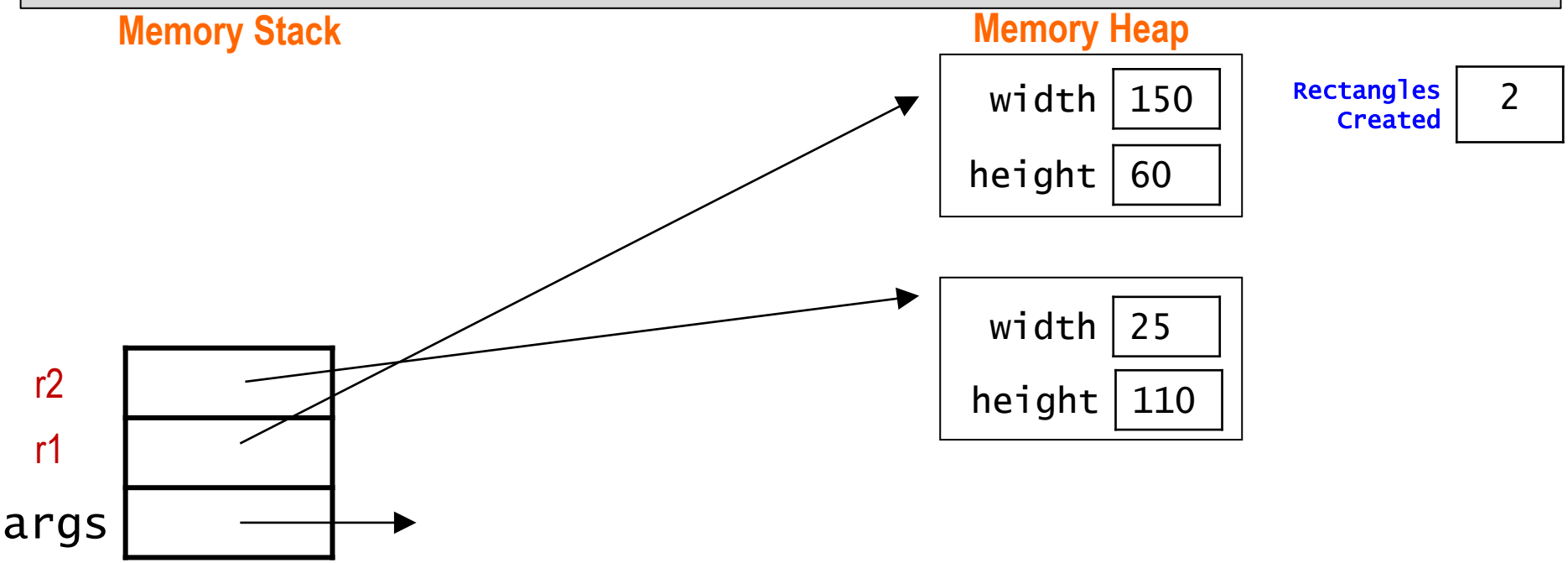
No change, the static method is called ***through*** *the object!*



A sample Client Program:

memory trace

```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600

      ...

      System.out.println("Number of Rectangles = " +
                                 Rectangle.numRectanglesCreated() ); // 2
} // end of the method
```



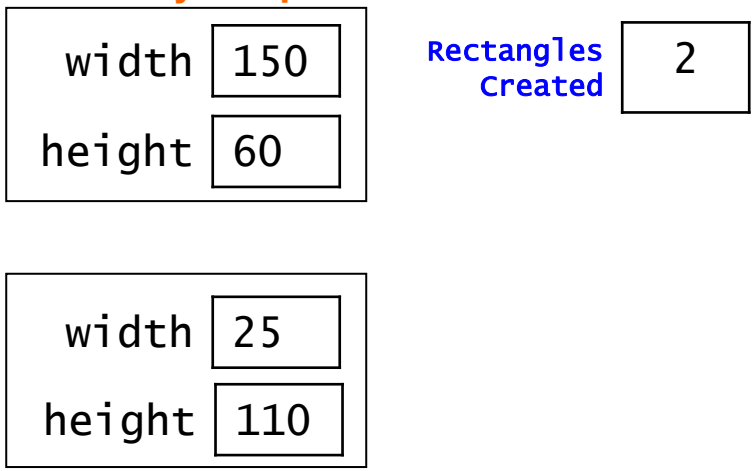
A sample Client Program:

memory trace

```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600

      ...

      System.out.println("Number of Rectangles = " +
                                    Rectangle.numRectanglesCreated() );
} // end of the method
```

Memory Stack

Memory Heap



A sample Client Program:

memory trace

```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600

      ...

      System.out.println("Number of Rectangles = " +
                                    Rectangle.numRectanglesCreated() );
} // end of the method
```

Memory Stack

Memory Heap

width	150
height	60

width	25
height	110

A sample Client Program:

memory trace

```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600

      ...

      System.out.println("Number of Rectangles = " +
                                    Rectangle.numRectanglesCreated() );
} // end of the method
```

Memory Stack

Memory Heap

Classes as Custom Data Type:

a summary

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

      int x;                      // no-arg constructor
      int x = 5;                  // custom contructor
      x = 12;                     // mutator/setter method
      int j = x;                  // accessor method
      System.out.println(x);      // toString method
      if ( x == j ) {             // equals method

      }
      if ( j == x ) {             // equals method

      }
   } // end of main method
} // end of class TestClass
```

Classes as Custom Data Type:

a summary

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

      int x;                      // Rectangle r = new Rectangle();
      int x = 5;                  // Rectangle r2 = new Rectangle(5);
      x = 12;                     // r.setWidth(12);
      int j = x;                  // r.getHeight();
      System.out.println(x); // System.out.println(r);
      if ( x == j ) {             // r.equals(r2);

      }
      if ( j == x ) {             // r2.equals( r );

      }
   } // end of main method
} // end of class TestClass
```

Sample Rectangle Class (C++)

C++

```
class Rectangle {
   private:
      int width;
      int height;


   public:
      Rectangle(int w, int h) {
        setWidth(w);
        setHheight(h);
      }
      void grow(int dw, int dh) {
        setWidth(width+dw);
        setHeight(height+dh);
      }
      double area() {
        return(width*height);
      }
      boolean operator==(Rectangle other) {
        return (other != null && width == other.width
                              && height == other.height );
      }
}
```

```
// Destructor
~Rectangle() {
   // object clean-up
}
```

Sample Rectangle Class (C++)

C++

```
class Rectangle {
   private:
      int width;
      int height;


   friend ostream &operator<<(ostream &out, Rectangle &r);
   public:
      Rectangle(int w, int h) {
        setWidth(w);
        setHheight(h);
      }
      void grow(int dw, int dh) {
        this.setWidth(width+dw);
        this.setHeight(height+dh);
      }
      double area() {
        return(this.width*this.height);
      }
      boolean operator==(Rectangle other) {
        return (other != null && this.width == other.width
                              && this.height == other.height );
      }
}
```

Classes as Custom Data Type:

C++

```
// C++ allows operator overloading

int main( ) {

      int x;                    // Rectangle r;
      int x = 5;                // Rectangle r2 = new Rectangle(5,10);
      x = 12;                   // r = 12;
      int j = x;                // j = r;
      cout << x;                // cout << r;
      if ( x == j ) {           // r == r2);

      }
      if ( j == x ) {           // r2 == r;

      }

   } // end of main method

} // end of class TestClass
```

Sample Rectangle Class (Python)

Python

```
class Rectangle:
    def __init__(self, init_width=15, init_height=15):
        self.x = 0
        self.y = 0
        self.width = init_width
        self.height = init_height

    def grow(self, dwidth, dheight):
        self.width += dwidth
        self.height += dheight

    def area(self):
         return self.width * self.height

    ...

    def __eq__(self, other):
         if self.width == other.width and self.height == other.height:
             return True
        else:
             return False

    def __repr__(self):
         return str(self.width) + ' x ' + str(self.height)
```

Sample Rectangle Class (Python)

Python

```
class Rectangle:
    def __init__(self, init_width, init_height):
        self.x = 0
        self.y = 0
        self.width = init_width
        self.height = init_height

    def grow(self, dwidth, dheight):
        self.width += dwidth
        self.height += dheight

    def area(self):
         return self.width * self.height

    ...

    def __eq__(self, other):
         if self.width == other.width and self.height == other.height:
             return True
        else:
             return False

    def __repr__(self):
         return str(self.width) + ' x ' + str(self.height)
```

Sample Rectangle Class (Python)

Python

```
class Rectangle:
    def __init__(self, init_width, init_height):
        self.x = 0
        self.y = 0
        self.width = init_width
        self.height = init_height

    def grow(self, dwidth, dheight):
        self.width += dwidth
        self.height += dheight

    def area(self):
         return self.width * self.height

    ...

    def __eq__(self, other):
         if self.width == other.width and self.height == other.height:
             return True
        else:
             return False

    def __repr__(self):
         return str(self.width) + ' x ' + str(self.height)
```

Classes as Custom Data Type:

Python

```
// Pyton allows operator overloading


x = 5                       // r = Rectangle(5, 10)
x = 12                      // object data members
j = x                       // can be directly accessed
print(x)                    // r.repr() or r.str()
if ( x == j ) {             // r == other


}

if ( j == x ) {             // other == r


}
```

Classes as Custom Data Type:

Python

```
// Pyton allows operator overloading


x = 5                    // r = Rectangle(
x = 12                   // object data me
j = x                    // can be directly access
print(x)                 // print(x)
if ( x == j ) {          // r == other


}

if ( j == x ) {          // other == r


}
```

Assume some *other* rectangle object

Classes as Custom Data Type:

Python

```
// Pyton allows operator overloading


x = 5                    // r = Rectangle(
x = 12                   // object data me
j = x                    // can be directly access
print(x)                 // print(r)
if ( x == j ) {          // r.__eq__(other)


}

if ( j == x ) {          // other.__eq__(r)


}
```

Assume some *other* rectangle object

Classes as Custom Data Type:

Python

```
// Pyton allows operator overloading


x = 5                       // r = Rectangle(
x = 12                      // object data me
j = x                       // can be directly access
print(x)                    // print(r)
if ( x.__eq__(j) ) {        // r.__eq__(other)


}

if ( j.__eq__(x) ) {        // other.__eq__(r)


}
```

Assume some *other* rectangle object

Your TURN!!!

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int w, int h) {
        setWidth(w);
        setHeight(h);
    }
    public Rectangle(int dim) {
        this(dim, dim);
    }
    public Rectangle() {
        this(0);
    }
    .
    .
    .

    public static void main( String [] args ) {

        Rectangle r1 = new Rectangle();
    }

}
```