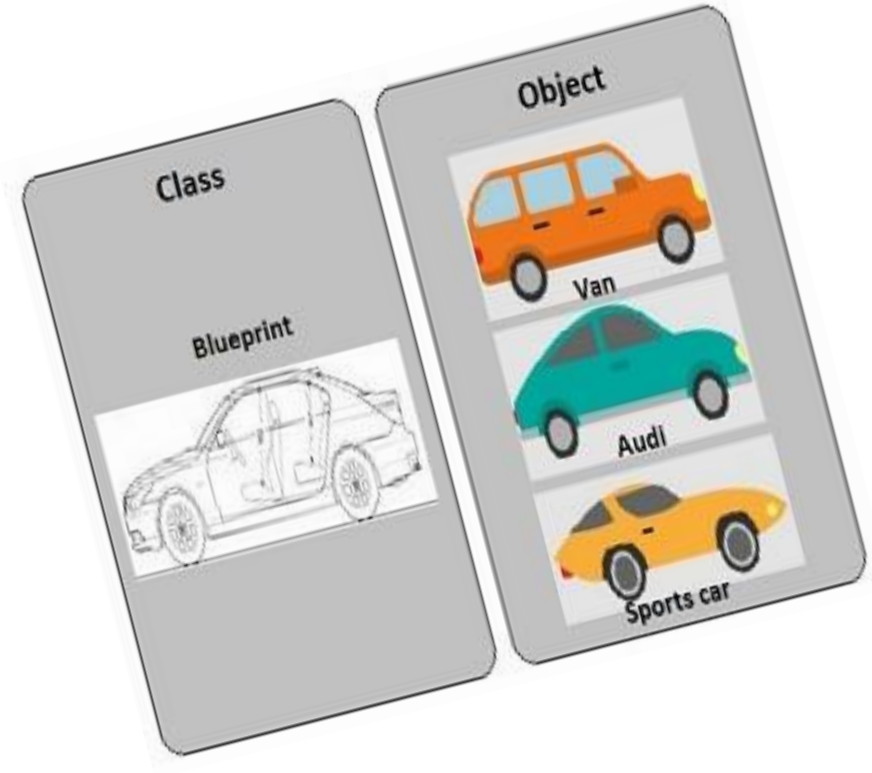


Writing our own classes to build *custom* data types

Computer Science OOD
Boston University

Christine Papadakis-Kanaris

Class Definition

```
public class className {

    public static variables    // class scope

    private instance members   // object scope

            Instance Methods of the class

    // constructor(s) initialize the object

    // mutator(s) modify the object's data

    // accessor(s) retrieve the object's data

    // ... methods to print, compare, etc.

            Static Methods of the class

}   // methods called at the class level
```

Access modifiers

Allows the class to control access of data members and methods.

Class Definition

```
public class className {

    public static variables    // class scope

    private instance members   // object scope

            Inst   e Meth

    // constructor(s)

    // mutator(s) m

    // accessor(s) m

    // ... methods to prin

            Static Methods of the class

    // methods called at the class level
}
```

The *public* and *private* access modifiers are used to encapsulate the data within the object and establish the public interface!

Access modifiers

ows the lass to ntrol cess of data members and methods.

Class Definition

```
public class className {

    public static variables    // class scope

    private instance members   // object scope

            Instance Methods of the class

    // constructor(s)

    // mutator(s) m

    // accessor(s) m

    // ... methods to print, p

            Static Methods of the class

    // methods called at the class level
}
```

In general data members are declared private and …

Access modifiers

ows the lass to ntrol cess of data members and methods.

Class Definition

```
public class cla

    public static varia

    private instance  ember

            Instance Methods of the class

    public className() { ... };

    public datatype setMethod() { ... };

    public datatype getMethod() { ... };

    // ... methods to print, compare, etc.

            Static Methods of the class

}   // methods called at the class level
```

… and methods are public. But not always. Anything can be declared public and anything can declared to be private!

Access modifiers

Allows the class to control access of data members and methods.

Sample Rectangle Class

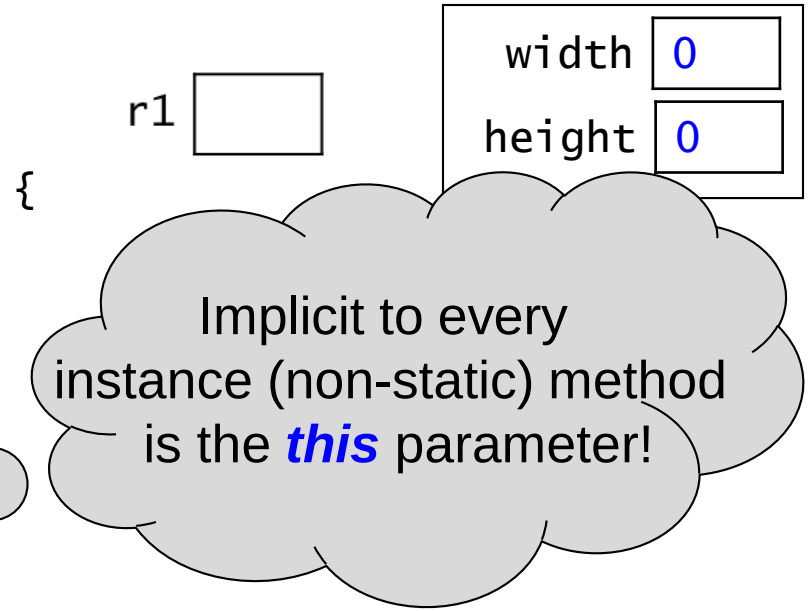
```
public class Rectangle {
    private int width;
    private int height;

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


    .
    .
    .
    .



}
```

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.

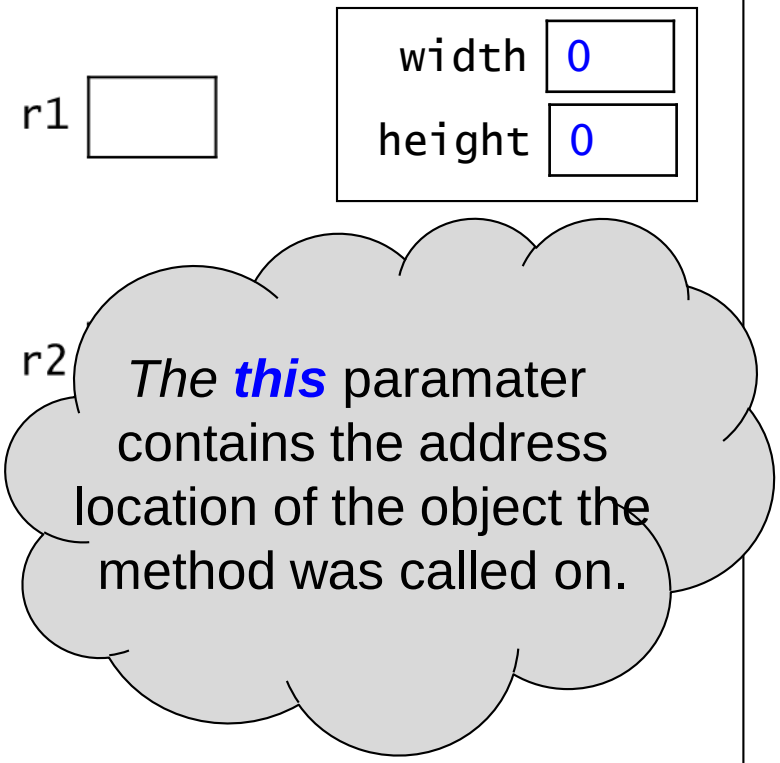
Sample Rectangle Class

```
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);
    }

}
```



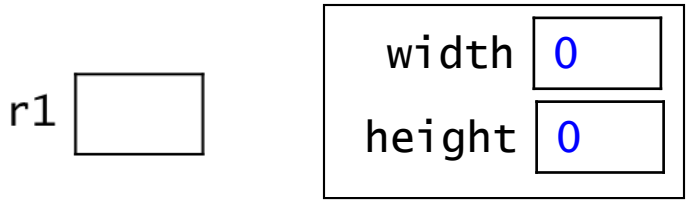
Sample Rectangle Class

```
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);
    }

}
```



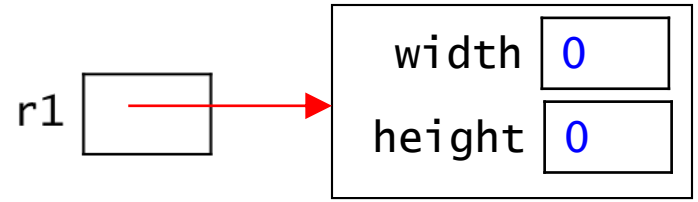
Sample Rectangle Class

```
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() {
        this.width = this.height = 0;
    }
    .
    .
    .
    public static void main( String [] args ) {

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

}
```



Sample Rectangle Class

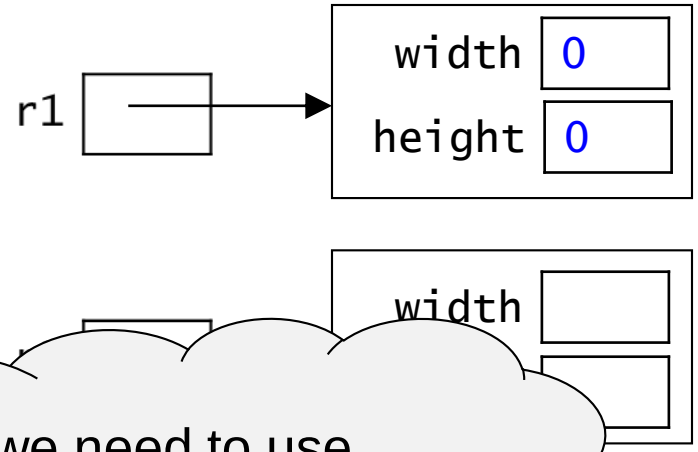
```
public class Rectangle {
    private int width;
    private int height;

    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);
    }

}
```



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

Sample Rectangle Class

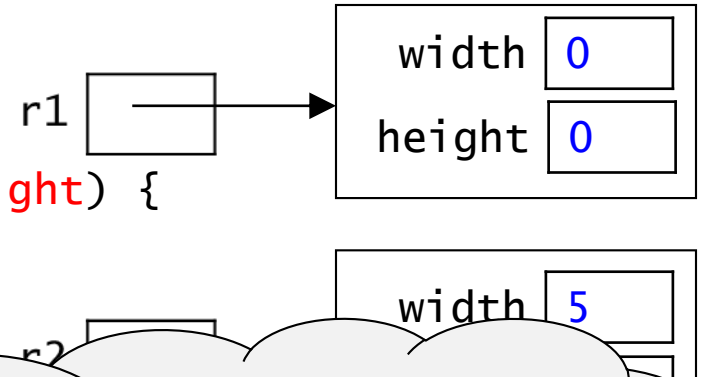
```
public class Rectangle {
    private int width;
    private int height;

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

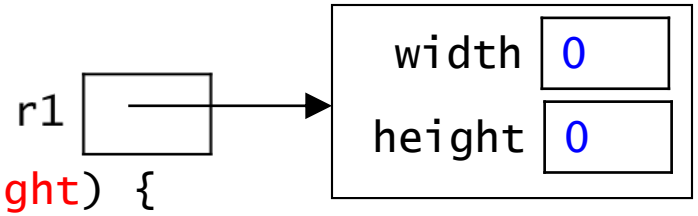
```
public class Rectangle {
    private int width;
    private int height;

    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);
    }

}
```



What if we give the parameters the same identifier name as the data members?

Sample Rectangle Class

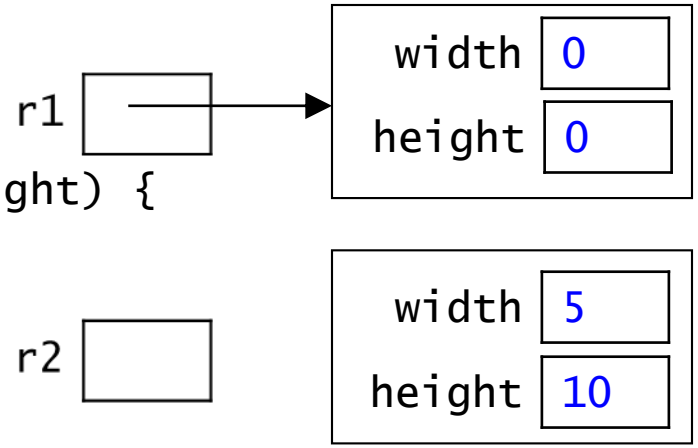
```
public class Rectangle {
    private int width;
    private int height;

    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);
    }

}
```



Now we have a scope issue!

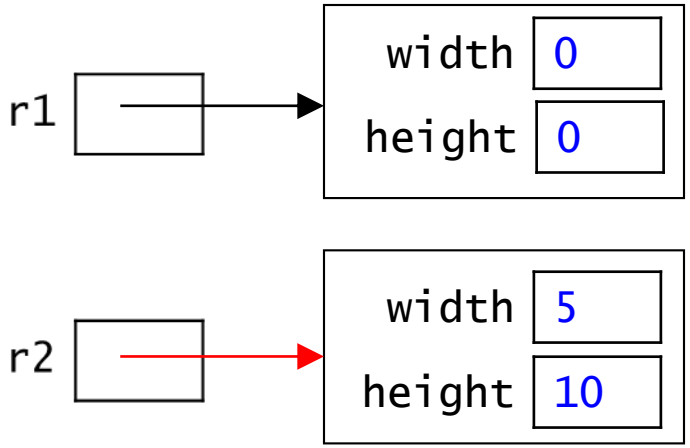
Sample Rectangle Class

```
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

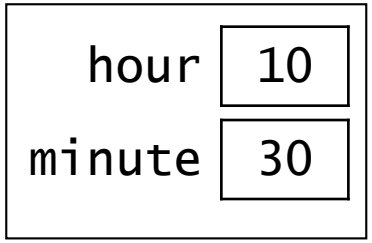
```
public class Rectangle {
    private int width;
    private int height;

    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);
    }


}
```



Another Example: A Class for `Time` Objects

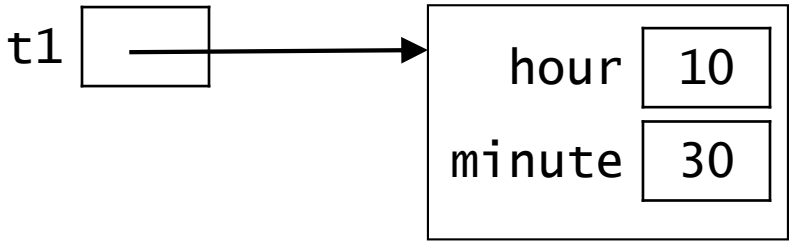
- Let's say that we want to create a data type for objects that represent military times (e.g., 10:30 or 17:50).

- A `Time` object for 10:30 would look like this:



- We would create it as follows:

```
Time t1 = new Time(10, 30);
```



Which of these is a valid initial `Time` class?

A.

```
public class Time {
    public Time(int h, int m) {
        self.hour = h;
        self.min = m;
    }
}
```

B.

```
public class Time {
    int hour; // must declare fields
    int min;

    public Time(int h, int m) {
        this.hour = h;
        this.min = m;
    }
}
```

no return type!

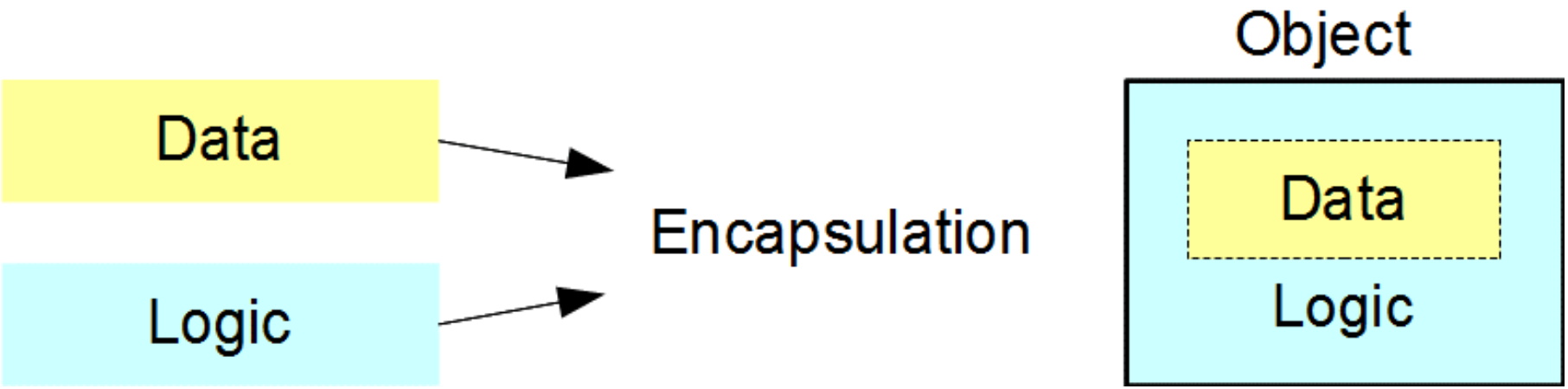
C.

```
public class Time {
    public void Time(int h, int m){
        self.hour = h;
        self.min = m;
    }
}
```

D.

```
public class Time {
    int hour;
    int min;

    public void Time(int h, int m) {
        this.hour = h;
        this.min = m;
    }
}
```

Encapsulation



Client Programs

- Our `Rectangle` class is *not* a program.
  - it has no `main` method

- Instead, it will be used by code defined in other classes.
  - referred to as *client programs* or *client code*

Blueprint Class vs. Client Program:

understanding the principle of Encapsulation

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

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

```
public class RectangleClient {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        int area1 = r1.width * r1.height;
        System.out.println("r1's area = " + area1);

        int area2 = r2.width * r2.height;
        System.out.println("r2's area = " + area2);

        // grow both rectangles
        r1.width += 50;  r1.height += 10;
        r2.width += 5;   r2.height += 30;

        System.out.println("r1: " + r1.width + " x " + r1.height);
        System.out.println("r2: " + r2.width + " x " + r2.height);
    }
}
```

Blueprint Class vs. Client Program:

understanding the principle of Encapsulation

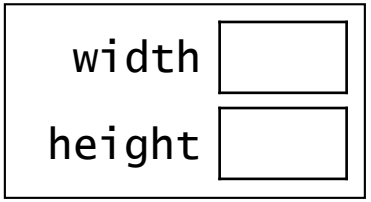
```
public class Rectangle {
    int width;
    int height;

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

```
public class RectangleClient {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        int area1 = r1.width * r1.height;
        System.out.println("r1's area = " + area1);

        int area2 = r2.width * r2.height;
        System.out.println("r2's area = " + area2);

        // grow both rectangles
        r1.width += 50;  r1.height += 10;
        r2.width += 5;   r2.height += 30;

        System.out.println("r1: " + r1.width + " x " + r1.height);
        System.out.println("r2: " + r2.width + " x " + r2.height);
    }
}
```

Blueprint Class vs. Client Program:

understanding the principle of Encapsulation

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

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



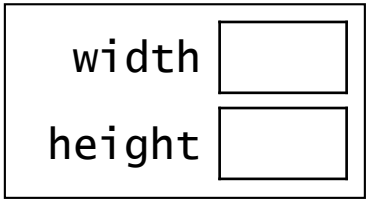

```
public class RectangleClient {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        int area1 = r1.width * r1.height;
        System.out.println("r1's area = " + area1);

        int area2 = r2.width * r2.height;
        System.out.println("r2's area = " + area2);

        // grow both rectangles
        r1.width += 50;  r1.height += 10;
        r2.width += 5;   r2.height += 30;

        System.out.println("r1: " + r1.width + " x " + r1.height);
        System.out.println("r2: " + r2.width + " x " + r2.height);
    }
}
```

Blueprint Class vs. Client Program:

understanding the principle of Encapsulation

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

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

```
w = 100;
h = 50;
```



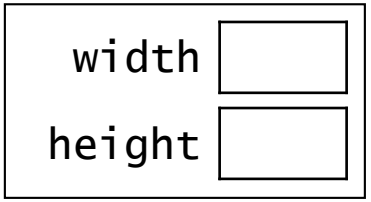

```
public class RectangleClient {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        int area1 = r1.width * r1.height;
        System.out.println("r1's area = " + area1);

        int area2 = r2.width * r2.height;
        System.out.println("r2's area = " + area2);

        // grow both rectangles
        r1.width += 50;  r1.height += 10;
        r2.width += 5;   r2.height += 30;

        System.out.println("r1: " + r1.width + " x " + r1.height);
        System.out.println("r2: " + r2.width + " x " + r2.height);
    }
}
```

Blueprint Class vs. Client Program:

understanding the principle of Encapsulation

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

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

```
w = 100;
h = 50;
```




```
public class RectangleClient {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        int area1 = r1.width * r1.height;
        System.out.println("r1's area = " + area1);

        int area2 = r2.width * r2.height;
        System.out.println("r2's area = " + area2);

        // grow both rectangles
        r1.width += 50;  r1.height += 10;
        r2.width += 5;   r2.height += 30;

        System.out.println("r1: " + r1.width + " x " + r1.height);
        System.out.println("r2: " + r2.width + " x " + r2.height);
    }
}
```

Blueprint Class vs. Client Program:

understanding the principle of Encapsulation

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

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

```
w = 100;
h = 50;
```

width	100
height	

```
public class RectangleClient {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        int area1 = r1.width * r1.height;
        System.out.println("r1's area = " + area1);

        int area2 = r2.width * r2.height;
        System.out.println("r2's area = " + area2);

        // grow both rectangles
        r1.width += 50;  r1.height += 10;
        r2.width += 5;   r2.height += 30;

        System.out.println("r1: " + r1.width + " x " + r1.height);
        System.out.println("r2: " + r2.width + " x " + r2.height);
    }
}
```

Blueprint Class vs. Client Program:

understanding the principle of Encapsulation

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

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

```
w = 100;
h = 50;
```

width	100
height	

```
public class RectangleClient {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        int area1 = r1.width * r1.height;
        System.out.println("r1's area = " + area1);

        int area2 = r2.width * r2.height;
        System.out.println("r2's area = " + area2);

        // grow both rectangles
        r1.width += 50;  r1.height += 10;
        r2.width += 5;   r2.height += 30;

        System.out.println("r1: " + r1.width + " x " + r1.height);
        System.out.println("r2: " + r2.width + " x " + r2.height);
    }
}
```

Blueprint Class vs. Client Program:

understanding the principle of Encapsulation

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

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

```
w = 100;
h = 50;
```

width	100
height	50

```
public class RectangleClient {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        int area1 = r1.width * r1.height;
        System.out.println("r1's area = " + area1);

        int area2 = r2.width * r2.height;
        System.out.println("r2's area = " + area2);

        // grow both rectangles
        r1.width += 50;  r1.height += 10;
        r2.width += 5;   r2.height += 30;

        System.out.println("r1: " + r1.width + " x " + r1.height);
        System.out.println("r2: " + r2.width + " x " + r2.height);
    }
}
```

Blueprint Class vs. Client Program:

understanding the principle of Encapsulation

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

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

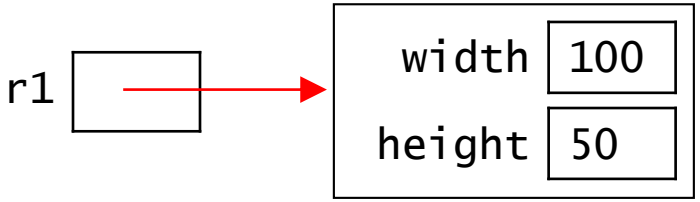
width	100
height	50

```
public class RectangleClient {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        int area1 = r1.width * r1.height;
        System.out.println("r1's area = " + area1);

        int area2 = r2.width * r2.height;
        System.out.println("r2's area = " + area2);

        // grow both rectangles
        r1.width += 50;  r1.height += 10;
        r2.width += 5;   r2.height += 30;

        System.out.println("r1: " + r1.width + " x " + r1.height);
        System.out.println("r2: " + r2.width + " x " + r2.height);
    }
}
```

Blueprint Class vs. Client Program:

understanding the principle of Encapsulation

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

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

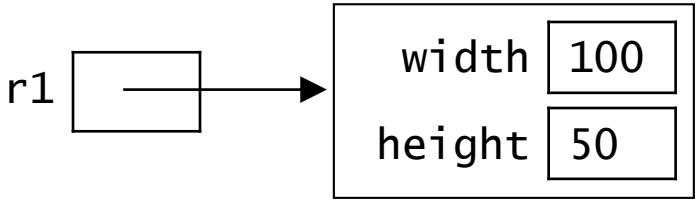



```
public class RectangleClient {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        int area1 = r1.width * r1.height;
        System.out.println("r1's area = " + area1);

        int area2 = r2.width * r2.height;
        System.out.println("r2's area = " + area2);

        // grow both rectangles
        r1.width += 50;  r1.height += 10;
        r2.width += 5;   r2.height += 30;

        System.out.println("r1: " + r1.width + " x " + r1.height);
        System.out.println("r2: " + r2.width + " x " + r2.height);
    }
}
```

Blueprint Class vs. Client Program:

understanding the principle of Encapsulation

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

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

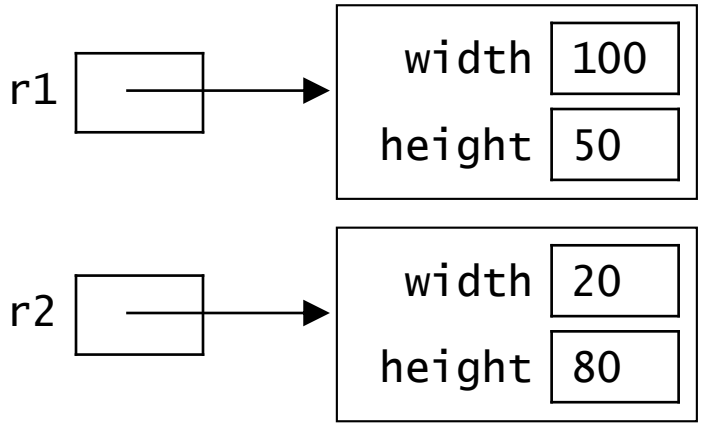



```
public class RectangleClient {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        int area1 = r1.width * r1.height;
        System.out.println("r1's area = " + area1);

        int area2 = r2.width * r2.height;
        System.out.println("r2's area = " + area2);

        // grow both rectangles
        r1.width += 50;  r1.height += 10;
        r2.width += 5;   r2.height += 30;

        System.out.println("r1: " + r1.width + " x " + r1.height);
        System.out.println("r2: " + r2.width + " x " + r2.height);
    }
}
```

Blueprint Class vs. Client Program:

understanding the principle of Encapsulation

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

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

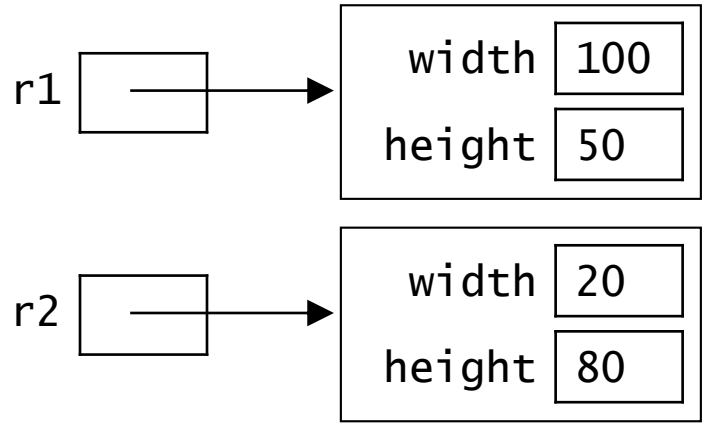



```
public class RectangleClient {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        int area1 = r1.width * r1.height;
        System.out.println("r1's area = " + area1);

        int area2 = r2.width * r2.height;
        System.out.println("r2's area = " + area2);

        // grow both rectangles
        r1.width += 50;  r1.height += 10;
        r2.width += 5;   r2.height += 30;

        System.out.println("r1: " + r1.width + " x " + r1.height);
        System.out.println("r2: " + r2.width + " x " + r2.height);
    }
}
```

Blueprint Class vs. Client Program:

understanding the principle of Encapsulation

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

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




```
public class RectangleClient {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        int area1 = r1.width * r1.height;
        System.out.println("r1's area = " + area1);

        int area2 = r2.width * r2.height;
        System.out.println("r2's area = " + area2);

        // grow both rectangles
        r1.width += 50;  r1.height += 10;
        r2.width += 5;   r2.height += 30;

        System.out.println("r1: " + r1.width + " x " + r1.height);
        System.out.println("r2: " + r2.width + " x " + r2.height);
    }
}
```

Blueprint Class vs. Client Program:

understanding the principle

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

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

Note the implication of not using class *access* modifiers and not specifying that the data members be private!

```
public class RectangleClient {
    public static void main(Stri  args) {
        Rectangle r1 = new R  angle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        int area1 = r1.width * r1.height;
        System.out.println("r1's area = " + area1);

        int area2 = r2.width * r2.height;
        System.out.println("r2's area = " + area2);

        // grow both rectangles
        r1.width += 50;  r1.height += 10;
        r2.width += 5;   r2.height += 30;

        System.out.println("r1: " + r1.width + " x " + r1.height);
        System.out.println("r2: " + r2.width + " x " + r2.height);
    }
}
```

Blueprint Class vs. Client Program:

understanding the principle of Encapsulation

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

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

The client program is directly accessing the *attributes* or data members of our rectangle objects to perform the necessary functions.

150

110

```
public class RectangleClient {
    public static void main(Str[] args) {
        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        int area1 = r1.width * r1.height;
        System.out.println("r1's area = " + area1);

        int area2 = r2.width * r2.height;
        System.out.println("r2's area = " + area2);

        // grow both rectangles
        r1.width += 50;  r1.height += 10;
        r2.width += 5;   r2.height += 30;

        System.out.println("r1: " + r1.width + " x " + r1.height);
        System.out.println("r2: " + r2.width + " x " + r2.height);
    }
}
```

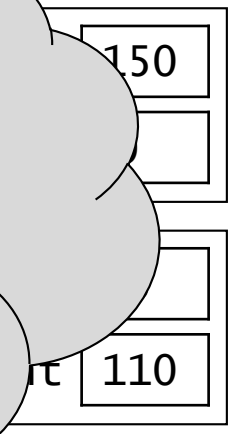
Blueprint Class vs. Client Program:

understanding the principle of Encapsulation

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

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

We want to embed functionality within our class through methods of the class!

```
public class RectangleClient {
    public static void main(Strin    args) {
        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        int area1 = r1.width * r1.height;
        System.out.println("r1's area = " + area1);

        int area2 = r2.width * r2.height;
        System.out.println("r2's area = " + area2);

        // grow both rectangles
        r1.width += 50;  r1.height += 10;
        r2.width += 5;   r2.height += 30;

        System.out.println("r1: " + r1.width + " x " + r1.height);
        System.out.println("r2: " + r2.width + " x " + r2.height);
    }
}
```



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 );
    }

    ...

}
```

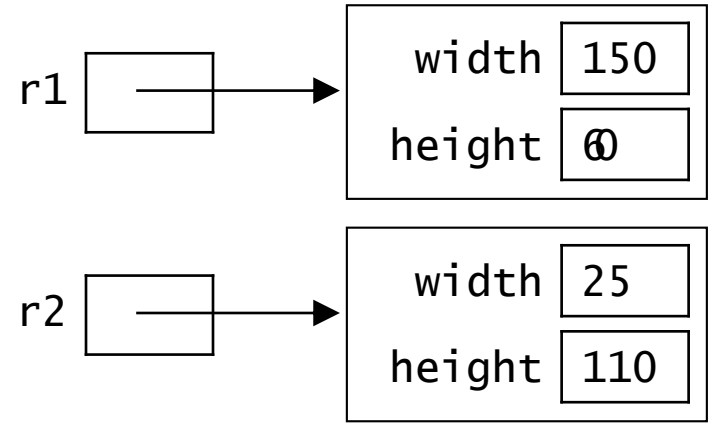
Accessor Methods

- 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.

Blueprint Class vs. Client Program:

understanding the principle of Encapsulation

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

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

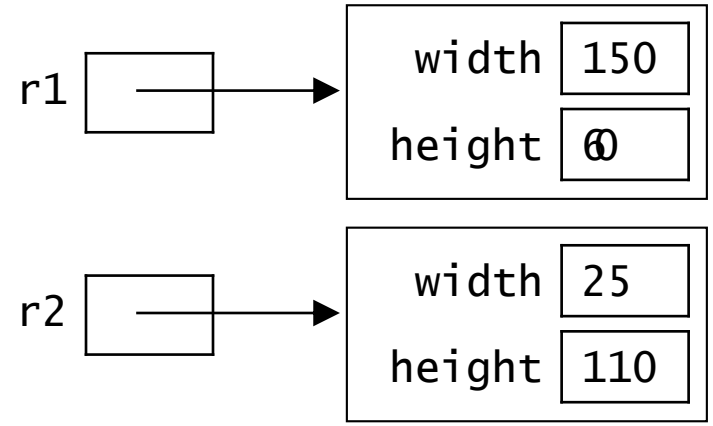



```
public class RectangleClient {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        int area1 = r1.width * r1.height;
        System.out.println("r1's area = " + area1);

        int area2 = r2.width * r2.height;
        System.out.println("r2's area = " + area2);

        // grow both rectangles
        r1.width += 50;  r1.height += 10;
        r2.width += 5;   r2.height += 30;

        System.out.println("r1: " + r1.width + " x " + r1.height);
        System.out.println("r2: " + r2.width + " x " + r2.height);
    }
}
```

Blueprint Class vs. Client Program:

understanding the principle of Encapsulation

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

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

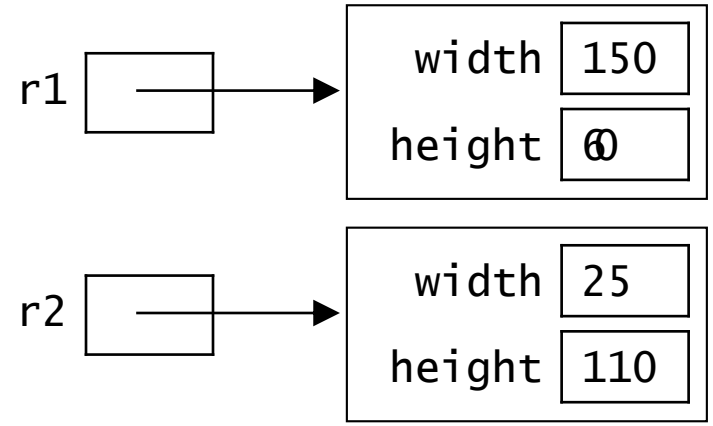



```
public class RectangleClient {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        int area1 = r1.area();
        System.out.println("r1's area = " + area1);

        int area2 = r2.area();
        System.out.println("r2's area = " + area2);

        // grow both rectangles
        r1.width += 50;  r1.height += 10;
        r2.width += 5;   r2.height += 30;

        System.out.println("r1: " + r1.width + " x " + r1.height);
        System.out.println("r2: " + r2.width + " x " + r2.height);
    }
}
```

Blueprint Class vs. Client Program:

understanding the principle of Encapsulation

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

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




```
public class RectangleClient {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle(100, 50);
        Rectangle r2 = new Rectangle(20, 80);

        int area1 = r1.area();
        System.out.println("r1's area = " + area1);

        int area2 = r2.area();
        System.out.println("r2's area = " + area2);

        // grow both rectangles
        r1.width += 50;  r1.height += 10;
        r2.width += 5;   r2.height += 30;

        System.out.println("r1: " + r1.width + " x " + r1.height);
        System.out.println("r2: " + r2.width + " x " + r2.height);
    }
}
```

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

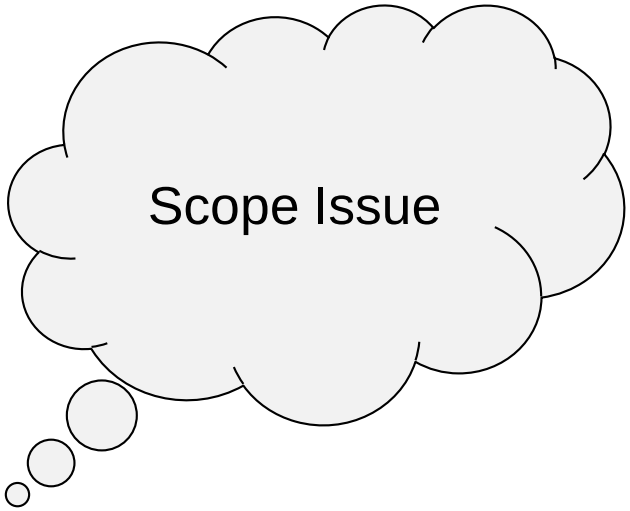
- Alter the values of the data members.

Sample Rectangle Class

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

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    .
    .
    .
    public void grow(int dw, int dh) {
        width += dw;
        height += dh;
    }
    .
    .
    .
}
```

Sample Rectangle Class

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

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



Sample Rectangle Class

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

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

Using the *this* reference resolves the scope issue.

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

    .
    .
    .
}
```

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?

Sample Rectangle Class

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

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

```
public void grow(int width, int height) {
        // perform error checking before
        // making assignments?

        this.width += width;
        this.height += height;
  }
```

Sample Rectangle Class

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

    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    .
    .
    .
    public void gro
        // perform
        // making a

        this.width += width;
        this.height += height;
    }
```

Shouldn't we perform
the same error checking
in the constructor as well?

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.

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();
    }

    width = w;
}

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

    height = h;
}
```

Sample Rectangle Class

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

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

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);
    }

    .
    .
    .
}
```

Rectangle Class 2.0 + A New Client

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

   public Rectangle(int w, int h) {
       this.width = w;
       this.height = h;
   }
   public void grow(int dw, int dh) {
       this.width += dw;
       this.height += dh;
   }
   public int area() {
       return this.width * this.height;
   }
}
```

```
public class MyClient {
    public static void
    main(String[] args) {
        Rectangle r = new Rectangle(10, 15);
        ______________________;

    }
}
```

Which method call increases r's height by 5?

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

   public Rectangle(int w, int h) {
       this.width = w;
       this.height = h;
   }
   public void grow(int dw, int dh) {
       this.width += dw;
       this.height += dh;
   }
   public int area() {
       return this.width * this.height;
   }
}
```

A. `r.grow(0, 5);`

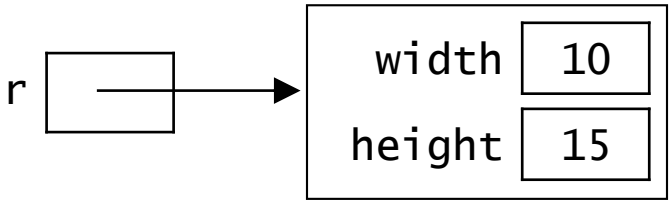
B. `r.grow(5, 0);`

C. `grow(r, 5, 0);`

D. `Rectangle.grow(0, 5);`

E. more than one works

```
public class MyClient {
    public static void
    main(String[] args) {
        Rectangle r = new Rectangle(10, 15);
        _________???_________;

    }
}
```



Which method call increases r's height by 5?

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

   public Rectangle(int w, int h) {
       this.width = w;
       this.height = h;
   }
   public void grow(int dw, int dh) {
       this.width += dw;
       this.height += dh;
   }
   public int area() {
       return this.width * this.height;
   }
}
```

A. `r.grow(0, 5);`

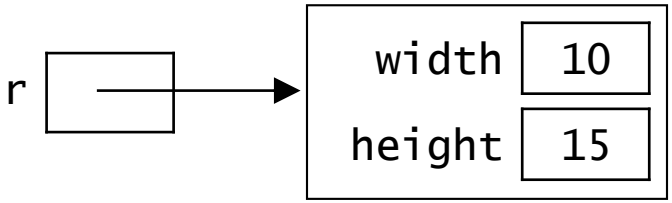
B. `r.grow(5, 0);`

C. `grow(r, 5, 0);`

D. `Rectangle.grow(0, 5);`

E. more than one works

```
public class MyClient {
    public static void
    main(String[] args) {
        Rectangle r = new Rectangle(10, 15);
        _________???_________;

    }
}
```



Types of Instance Methods:

a summary

- There are two main types of instance methods:
  - *mutators* – methods that change an object's internal state
  - *accessors* – methods that retrieve information from an object without changing its state

- Examples of mutators:
  - `grow()` in our `Rectangle` class

- Examples of accessors:
  - `area()` in our `Rectangle` class
  - `String` methods: `length()`, `substring()`, `charAt()`

Practice Defining Instance Methods

- Add a *mutator* method that *scales* the rectangle's dimensions by a specified factor (an integer).

```
public void scale(int factor) {
    this.width *= factor;
    this.height *= factor;
}
```

- Add an *accessor* method that determines if the rectangle is a square (true or false).

```
public boolean isSquare() {   // no inputs!
    if (this.width == this.height) {
        return true;
    } else {
        return false;
    }
}
```

Practice Defining Instance Methods

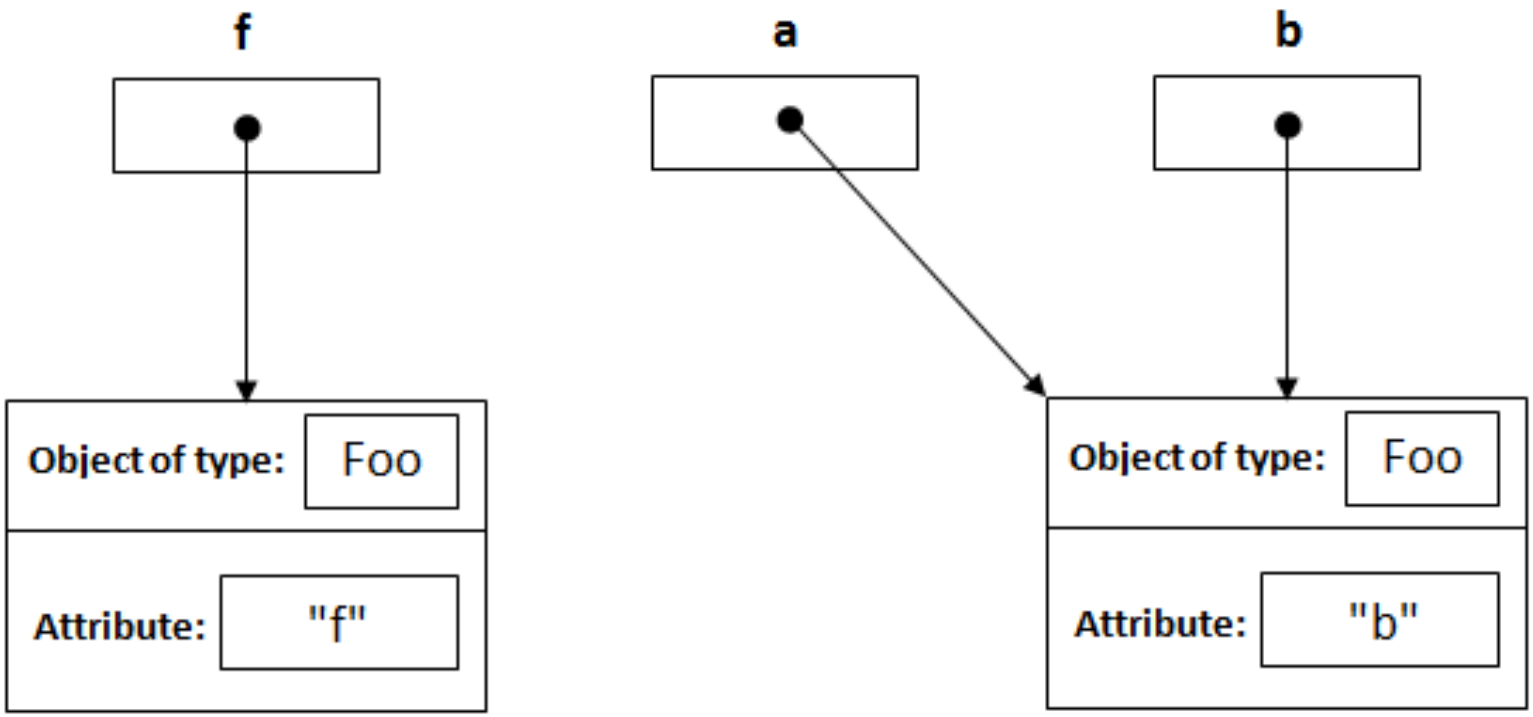
- Add a *mutator* method that *scales* the rectangle's dimensions by a specified factor (an integer).

```
public void scale(int factor) {
    this.width *= factor;
    this.height *= factor;
}
```

- Add an *accessor* method that determines if the rectangle is a square (true or false).

```
public boolean isSquare() {   // no inputs!
    return (this.width == this.height);
}
```

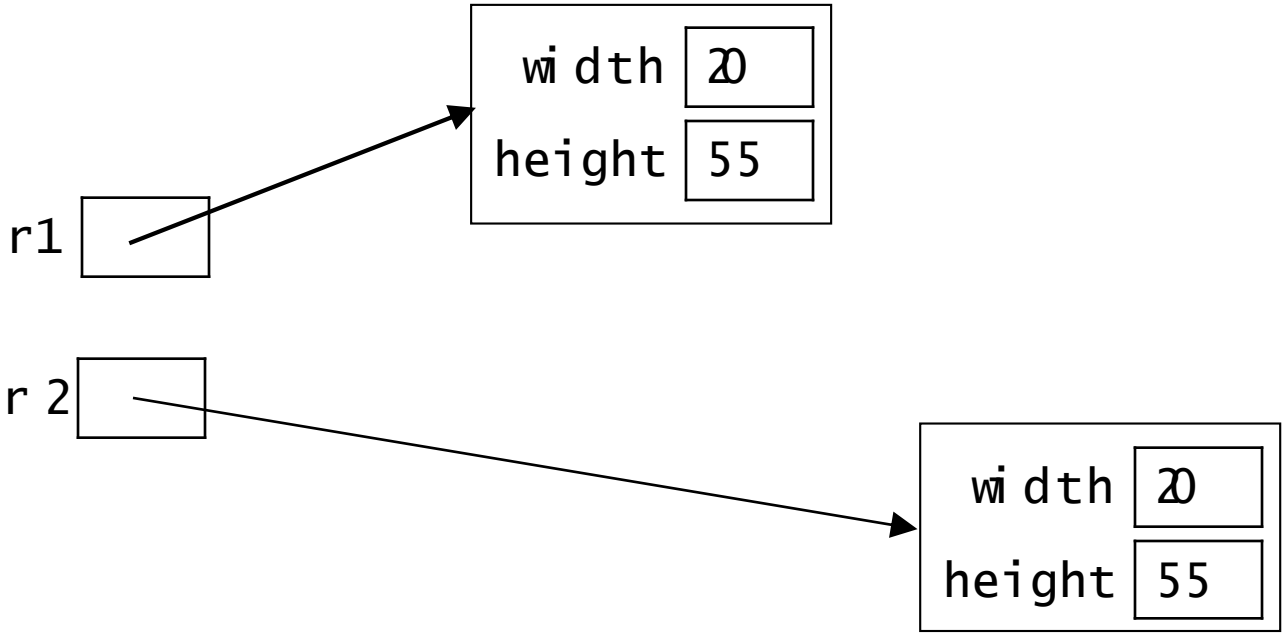
Objects are *Reference* types



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);
```



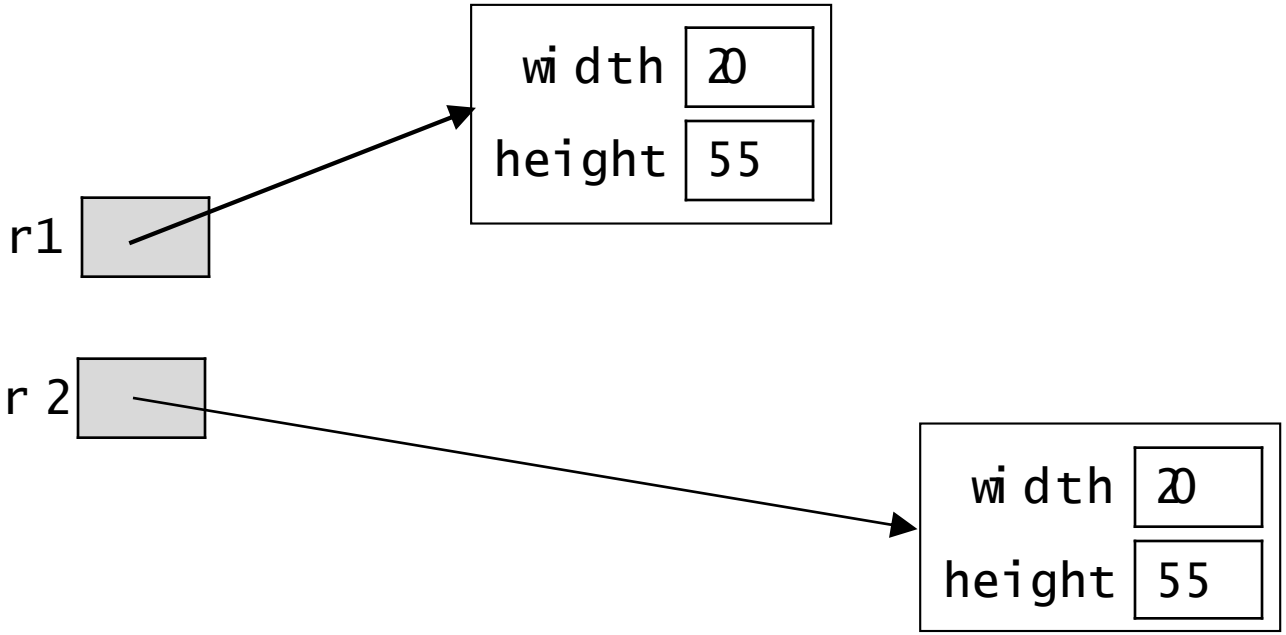
- What is the value of the following condition?

```
r1 == r2
```

Testing for Equivalent Objects

- 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?

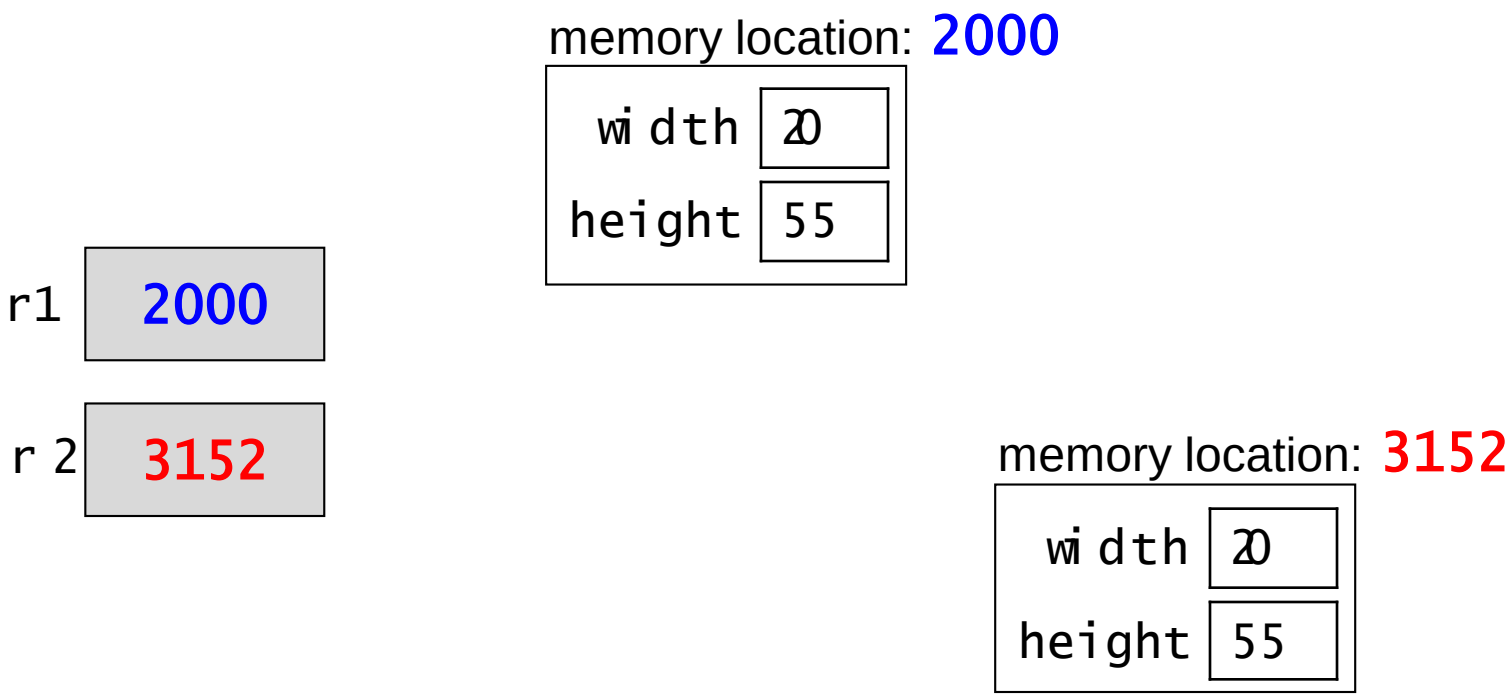
```
r1 == r2
```

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 = “CS111”;
String s2 = “CS112”;
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

```
public boolean equals(Rectangle other) {
    boolean isEqual = true; // assume equality
    // conditional logic to test for in-equality
    if (other == null) {
        isEqual = false;
    } else
      if (this.width != other.width) {
        isEqual = false;
    } else
      if (this.height != other.height) {
        isEqual = false;
    }
    return( isEqual );  // return the value assigned
}
```

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

```
public boolean equals(Rectangle other) {
    boolean isEqual = true;

    if (other == null) {
        isEqual = false;
    } else
      if (this.width != other.width) {
        isEqual = false;
    } else
      if (this.height != other.height) {
        isEqual = false;
    }
    return( isEqual );
}
```

- **Note:** The method is able to access the fields in `other` directly (without using accessor methods).
- Instance methods can access the private data members of *any* object from the same class.

`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);
```

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

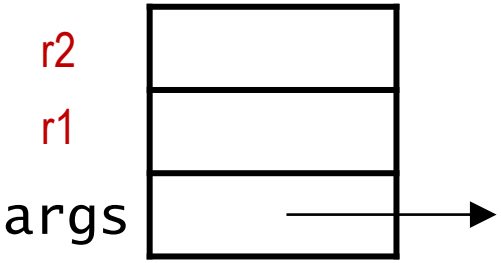
```
public class RectangleClient {

    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:

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**



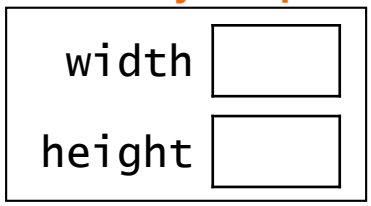
A sample Client Program:

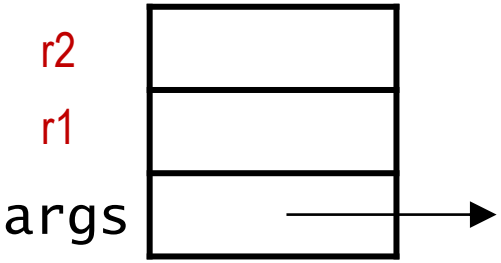
memory trace

```
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





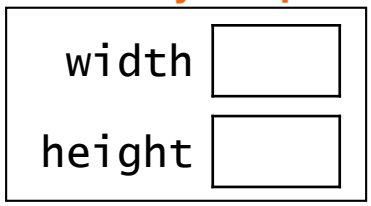
A sample Client Program:

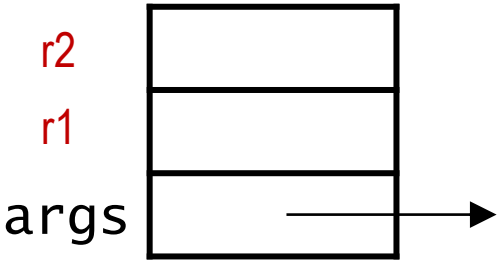
memory trace

```
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





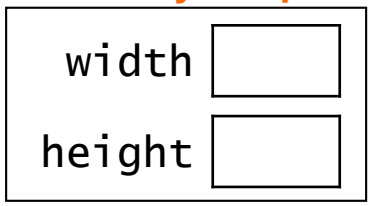
A sample Client Program:

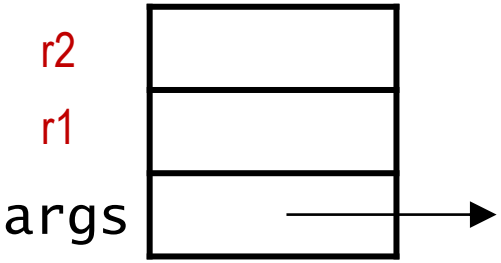
memory trace

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

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

Memory Stack

Memory Heap

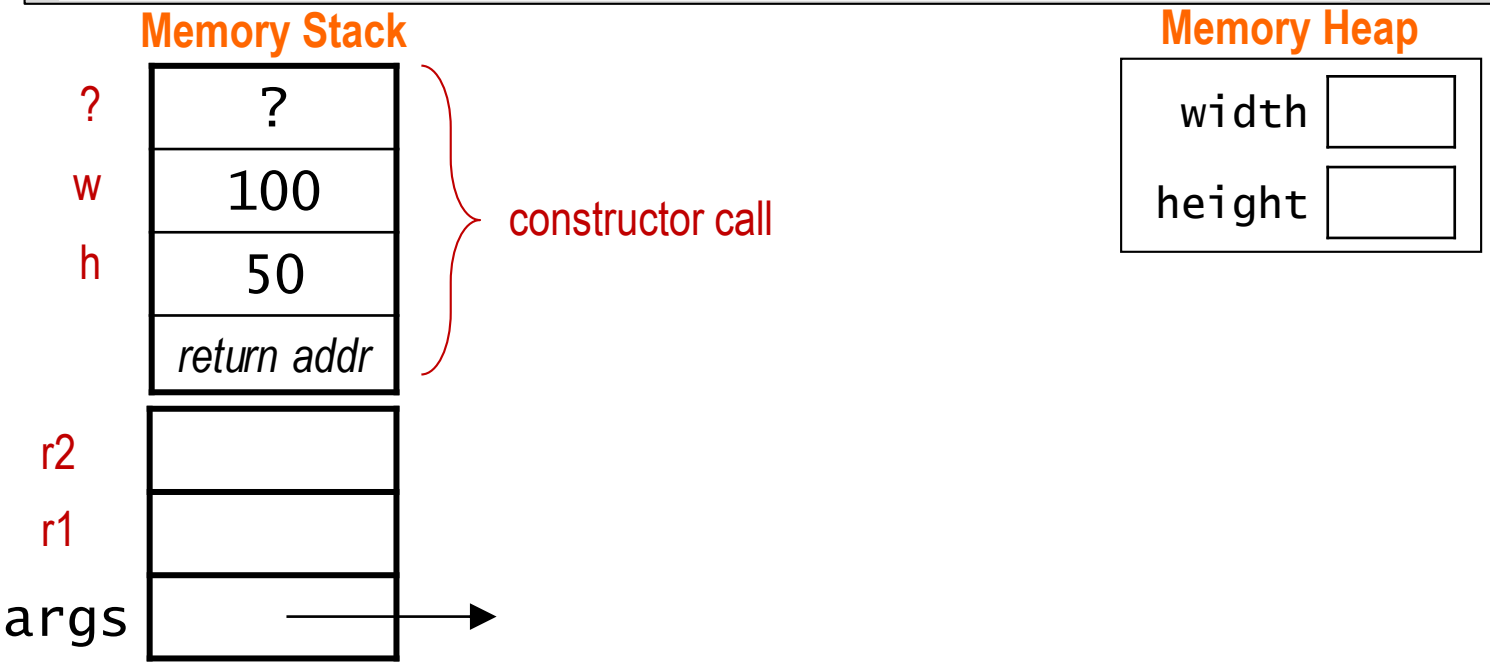




A sample Client Program:

memory trace

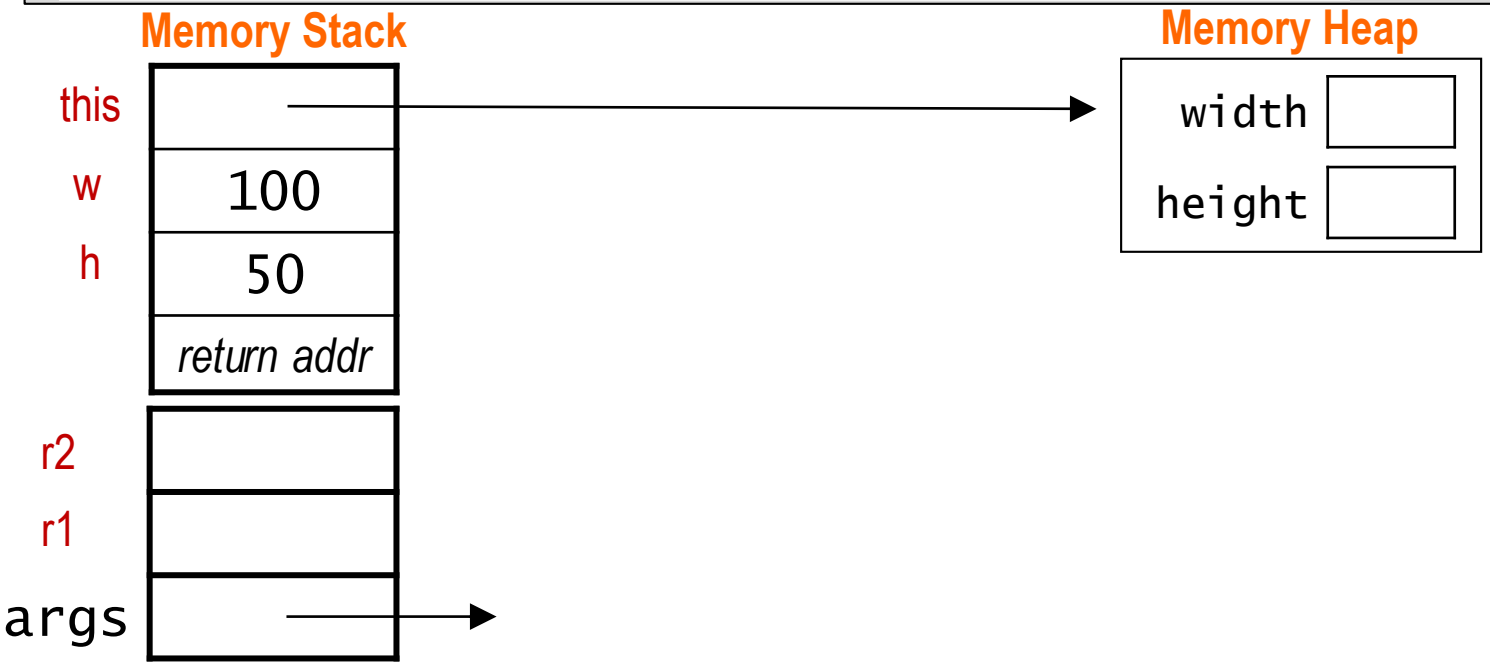
```
public class Rectangle {
    private int width;
    private int height;

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



A sample Client Program:

memory trace

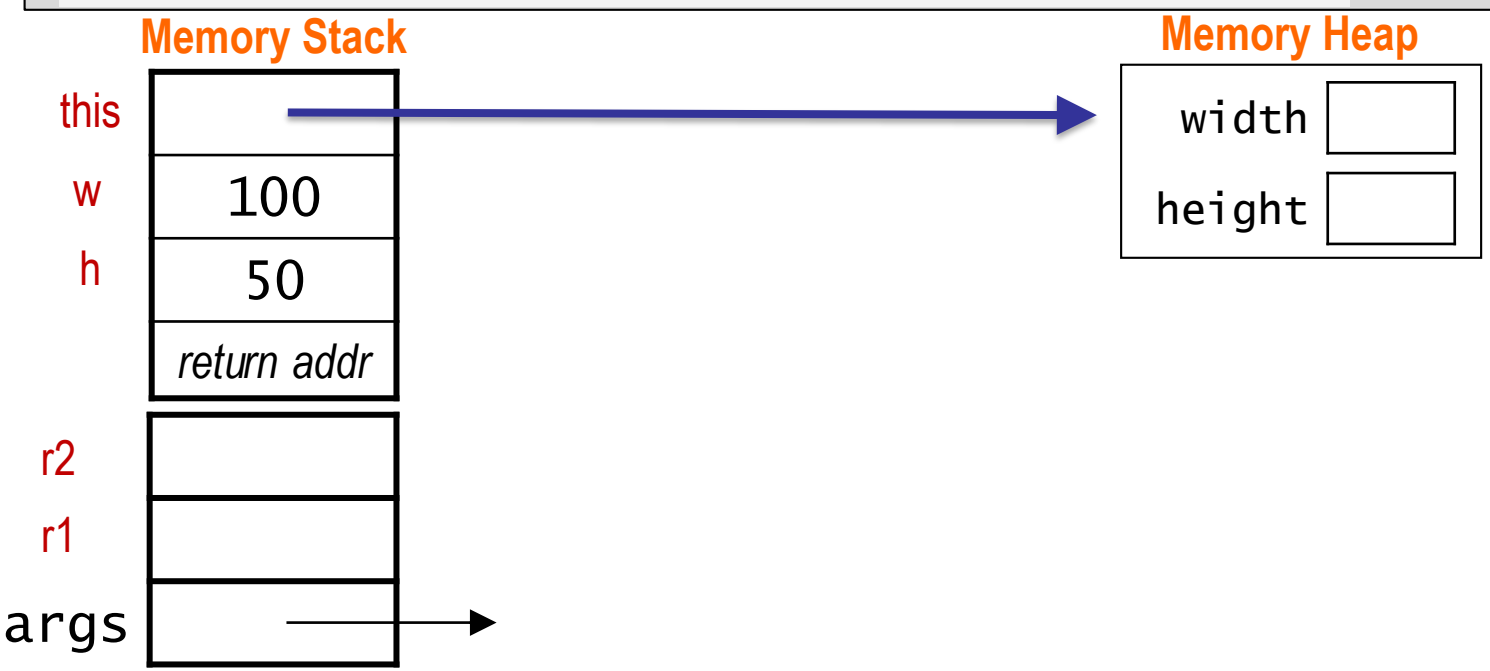
```
public class Rectangle {
    private int width;
    private int height;

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



A sample Client Program:

memory trace

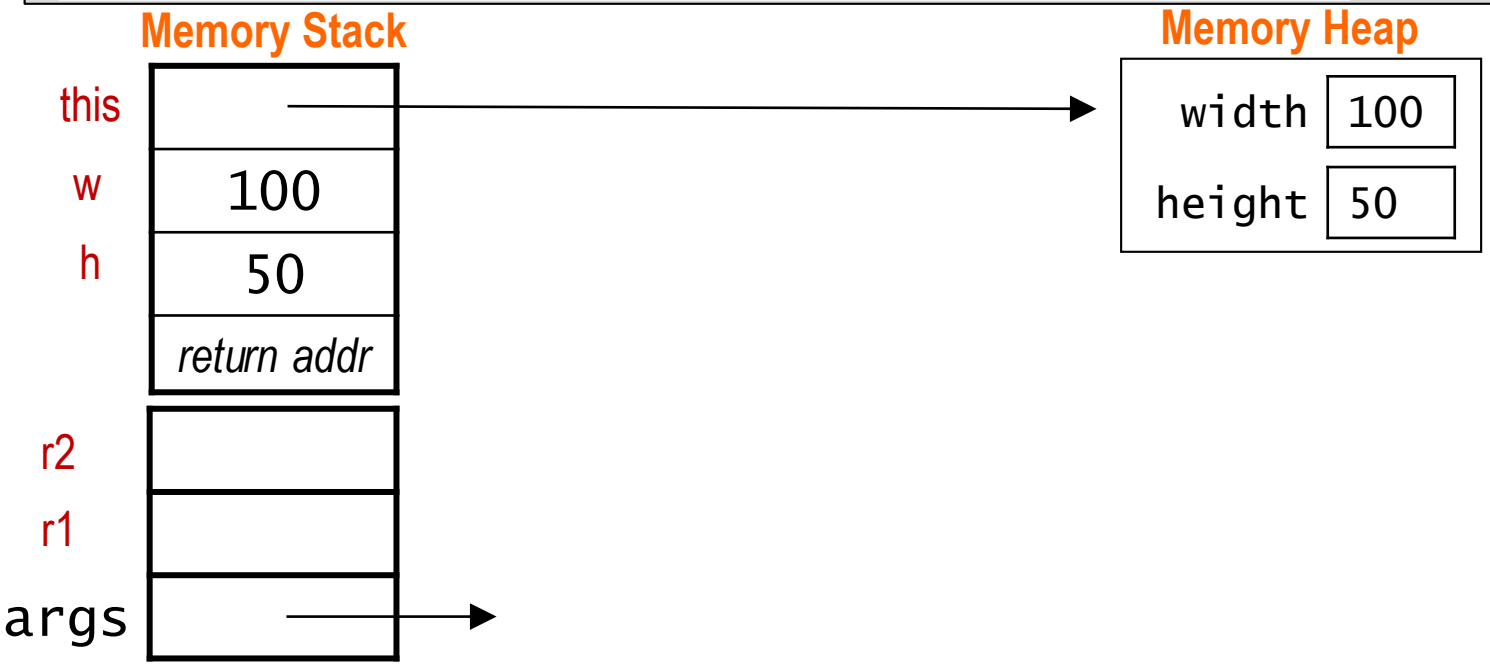
```
public class Rectangle {
    private int width;
    private int height;

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



A sample Client Program:

memory trace

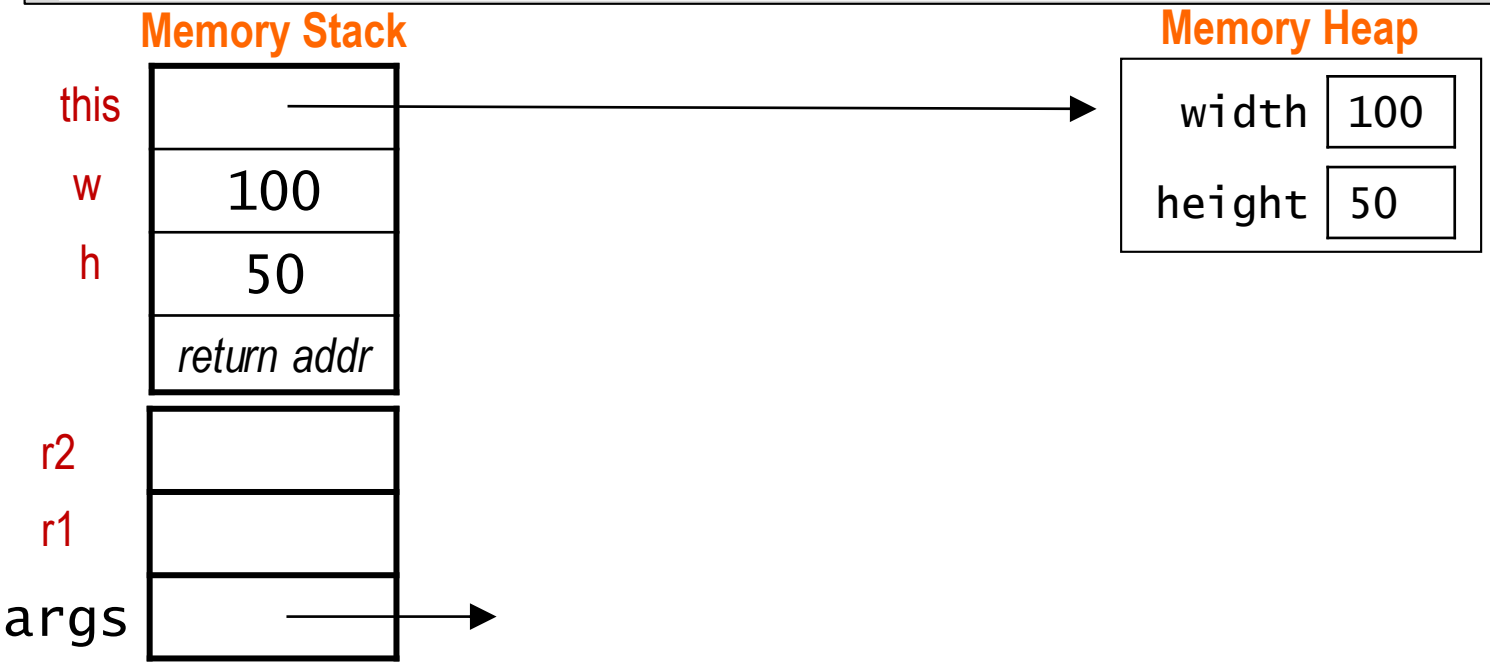
```
public class Rectangle {
    private int width;
    private int height;

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



A sample Client Program:

memory trace

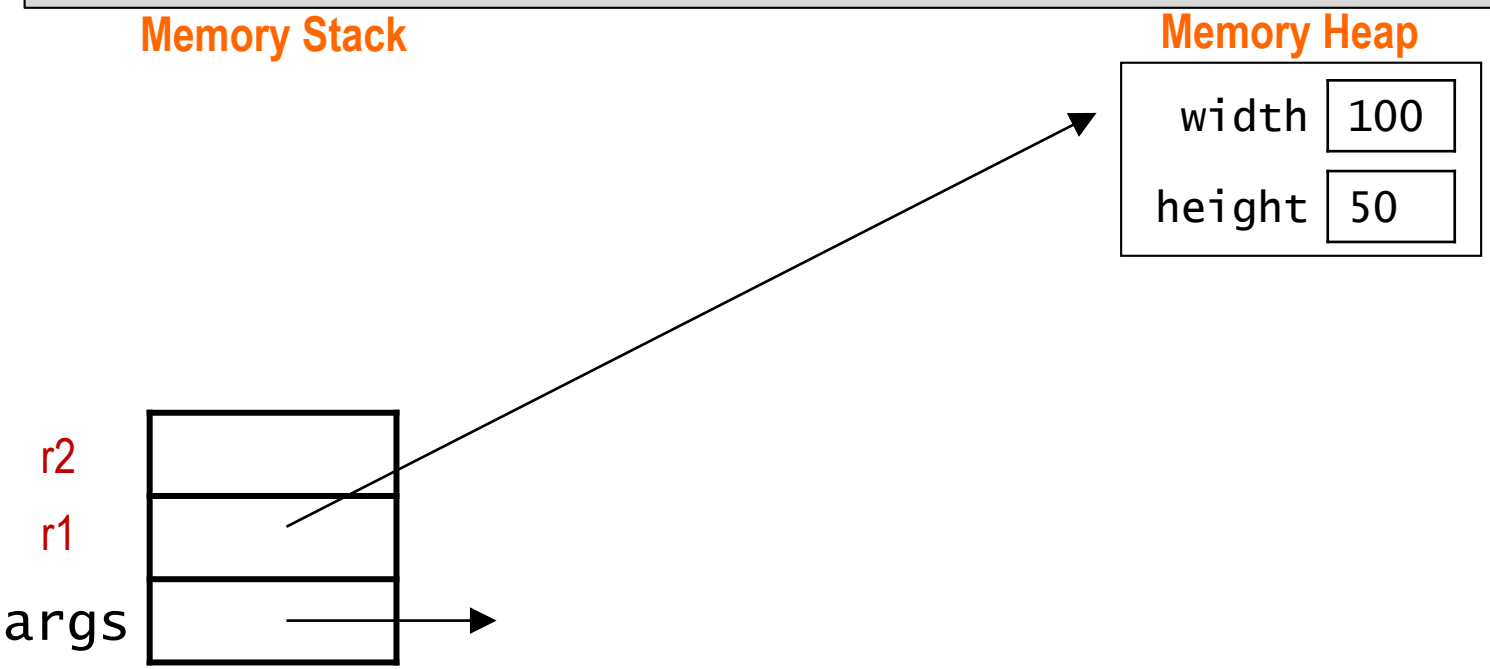
```
public class Rectangle {
    private int width;
    private int height;

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



A sample Client Program:

memory trace

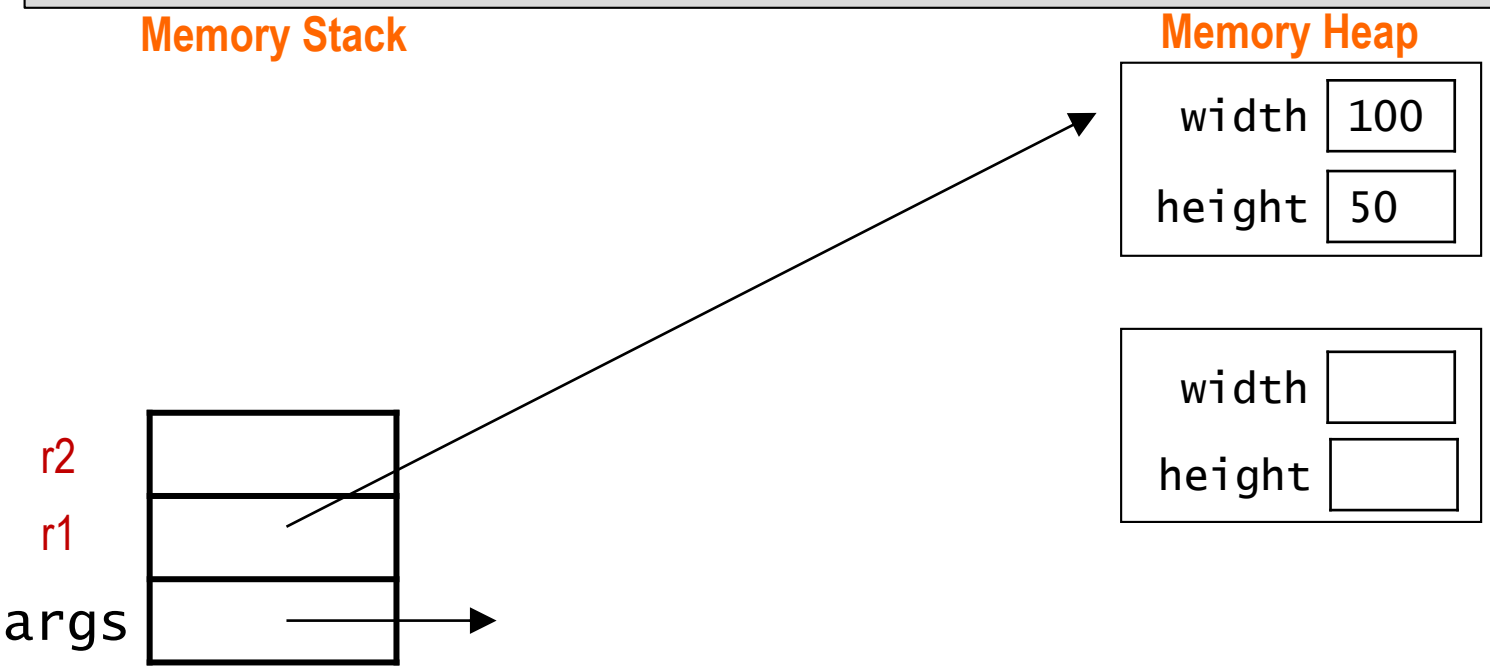
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + r2.area() );
      // grow both rectangles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



A sample Client Program:

memory trace

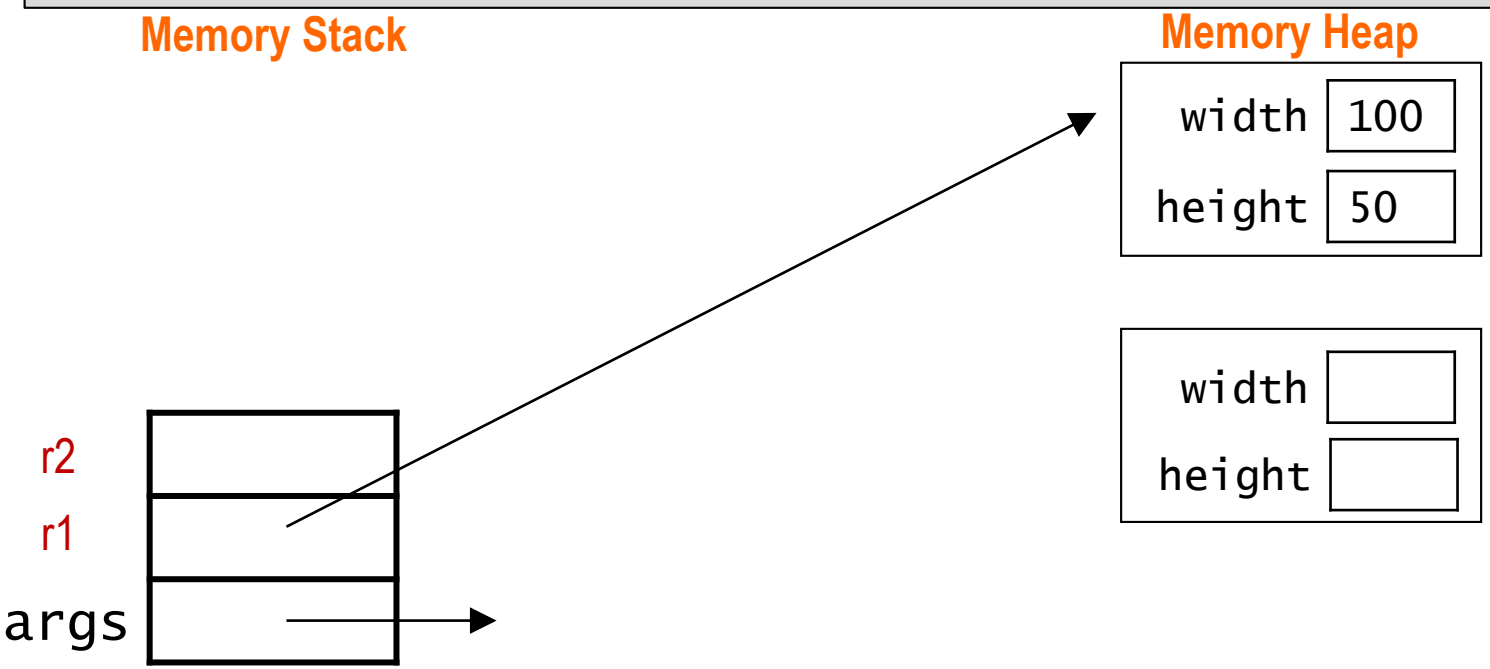
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + r2.area() );
      // grow both rectangles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



A sample Client Program:

memory trace

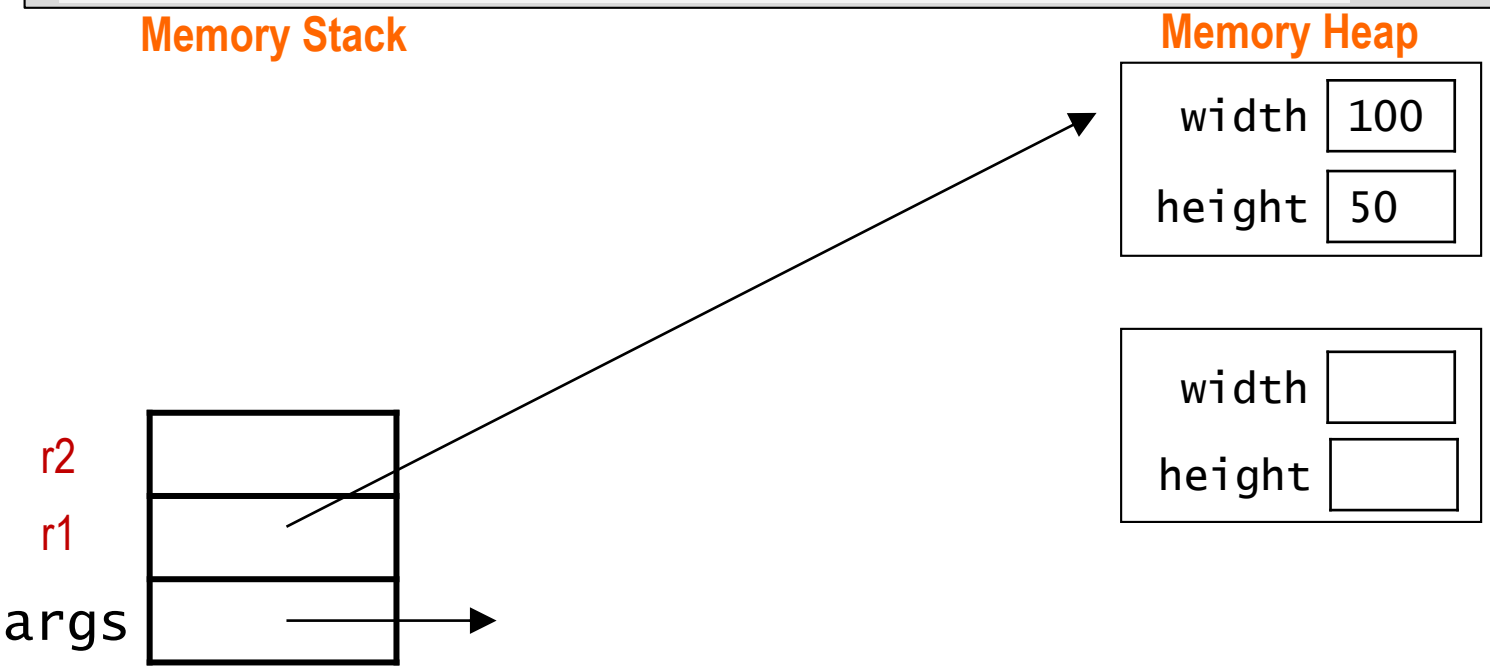
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + r2.area() );
      // grow both rectangles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



A sample Client Program:

memory trace

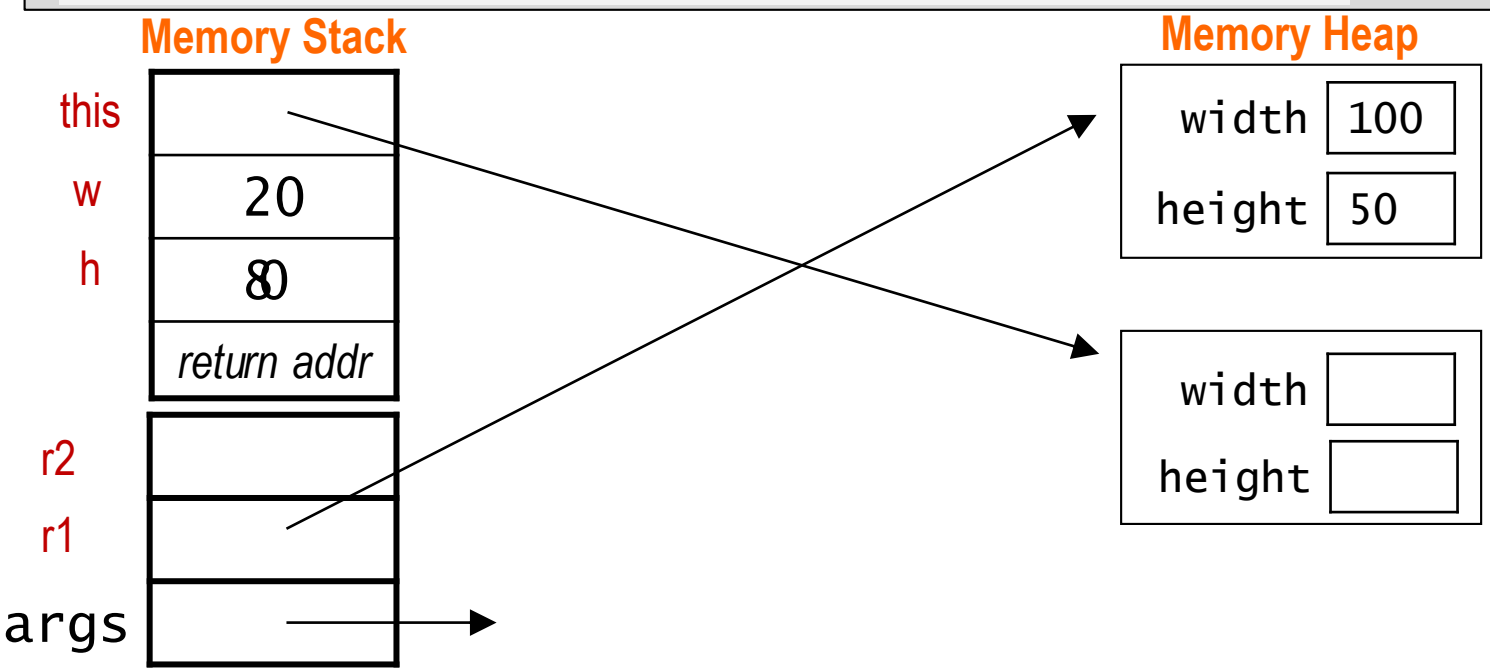
```
public class Rectangle {
    private int width;
    private int height;

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



A sample Client Program:

memory trace

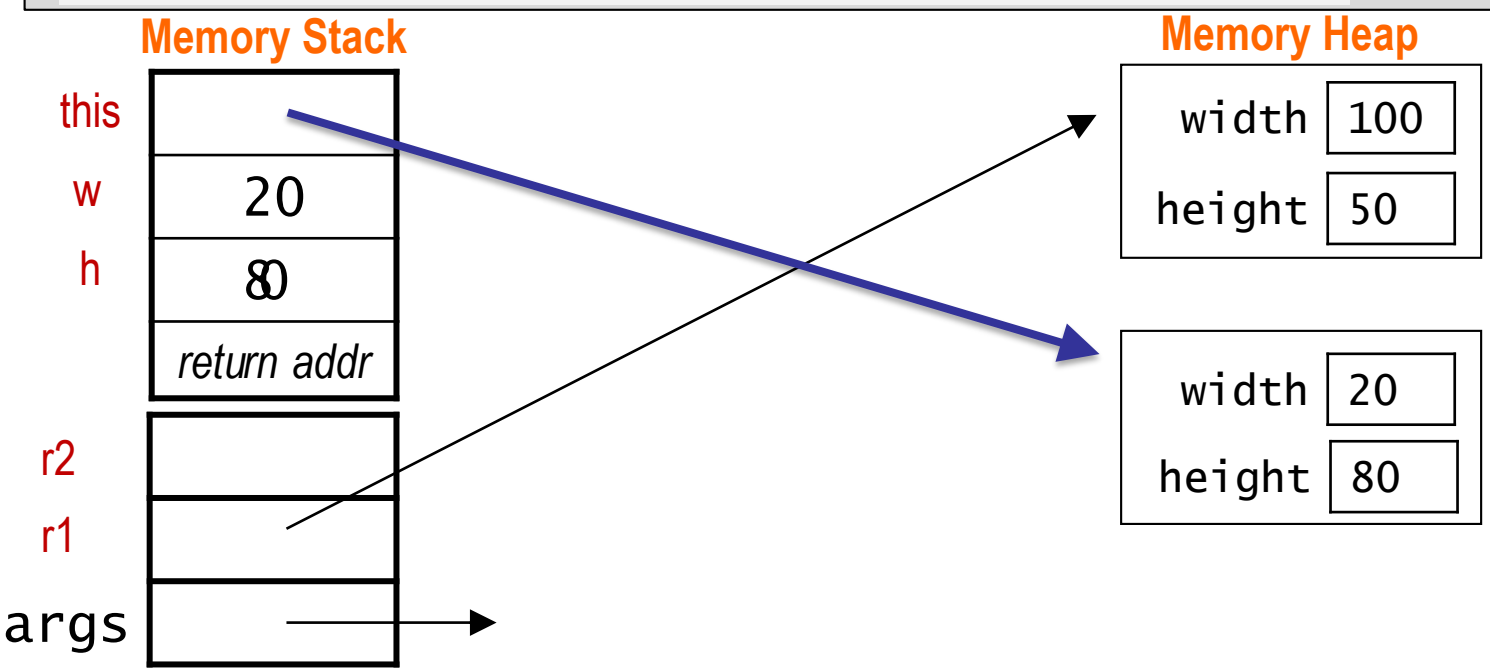
```
public class Rectangle {
    private int width;
    private int height;

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



A sample Client Program:

memory trace

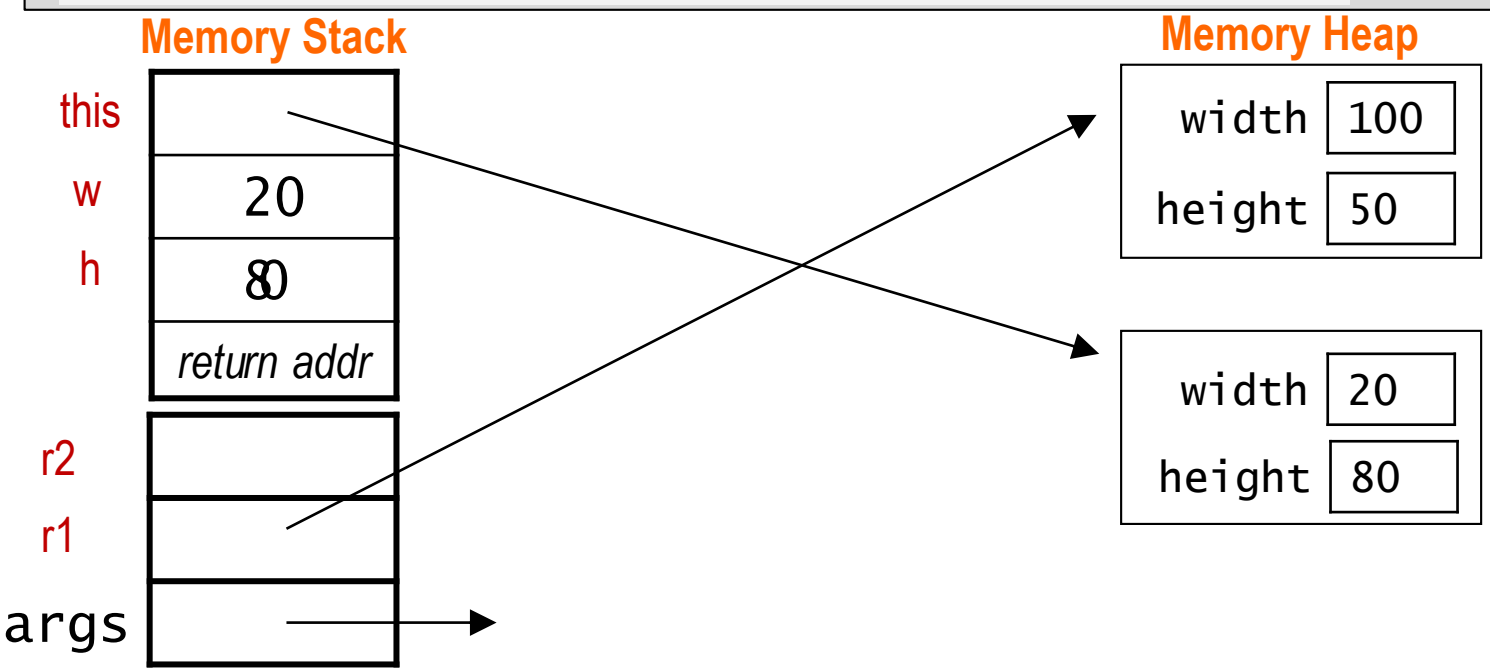
```
public class Rectangle {
    private int width;
    private int height;

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



A sample Client Program:

memory trace

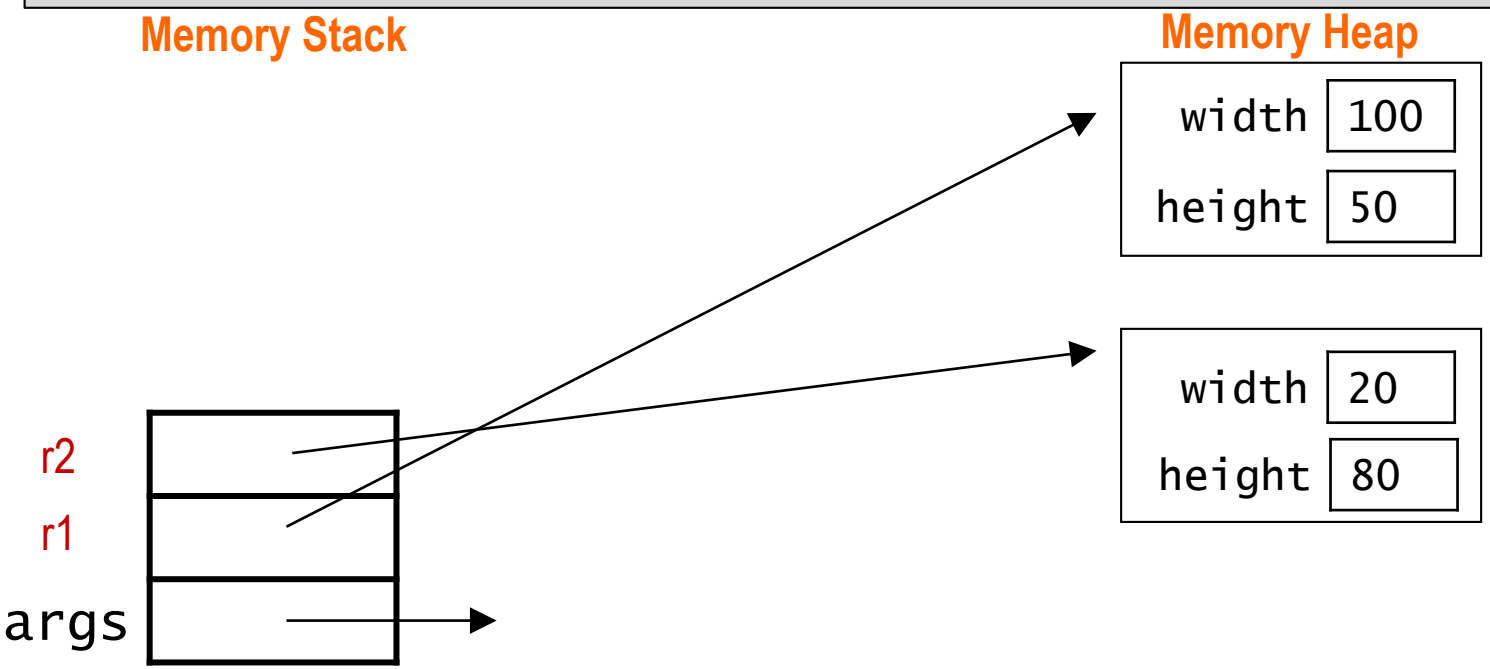
```
public class Rectangle {
    private int width;
    private int height;

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



A sample Client Program:

memory trace

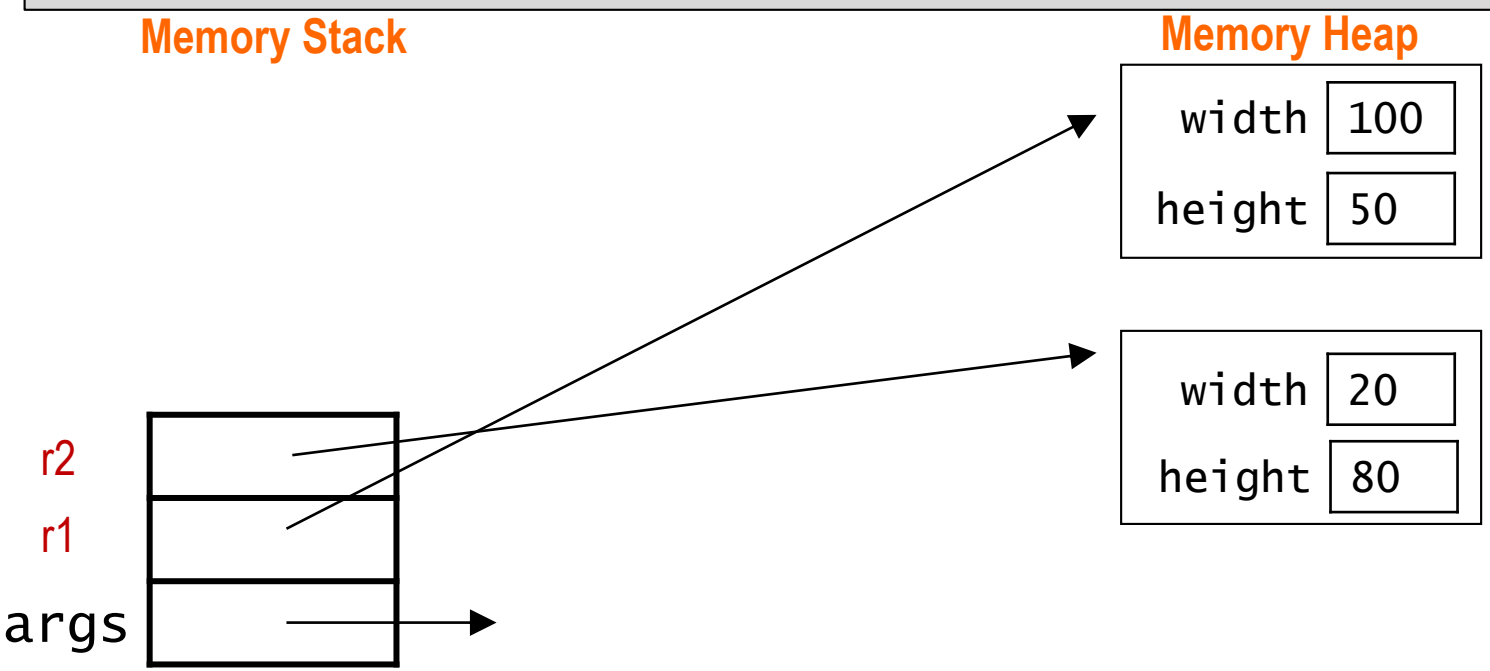
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + r2.area() );
      // grow both rectangles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



A sample Client Program:

memory trace

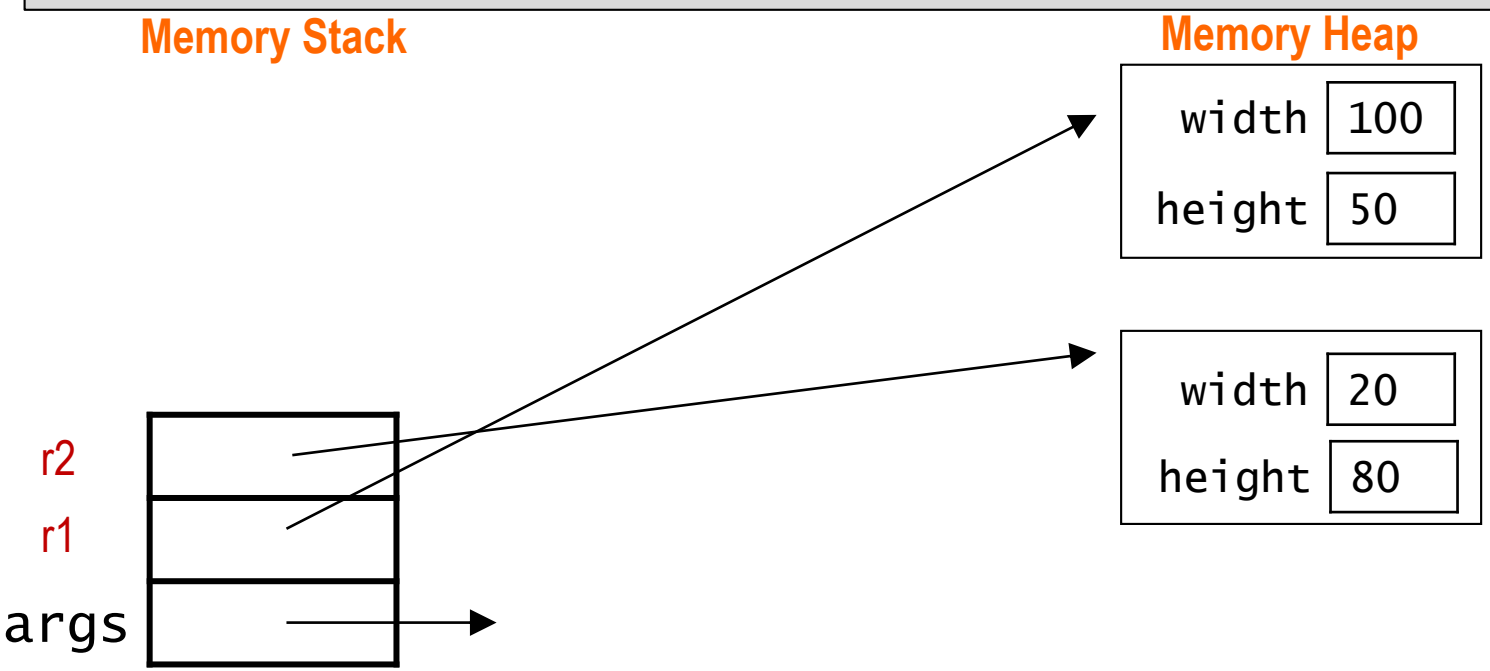
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + r2.area() );
      // grow both rectangles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



A sample Client Program:

memory trace

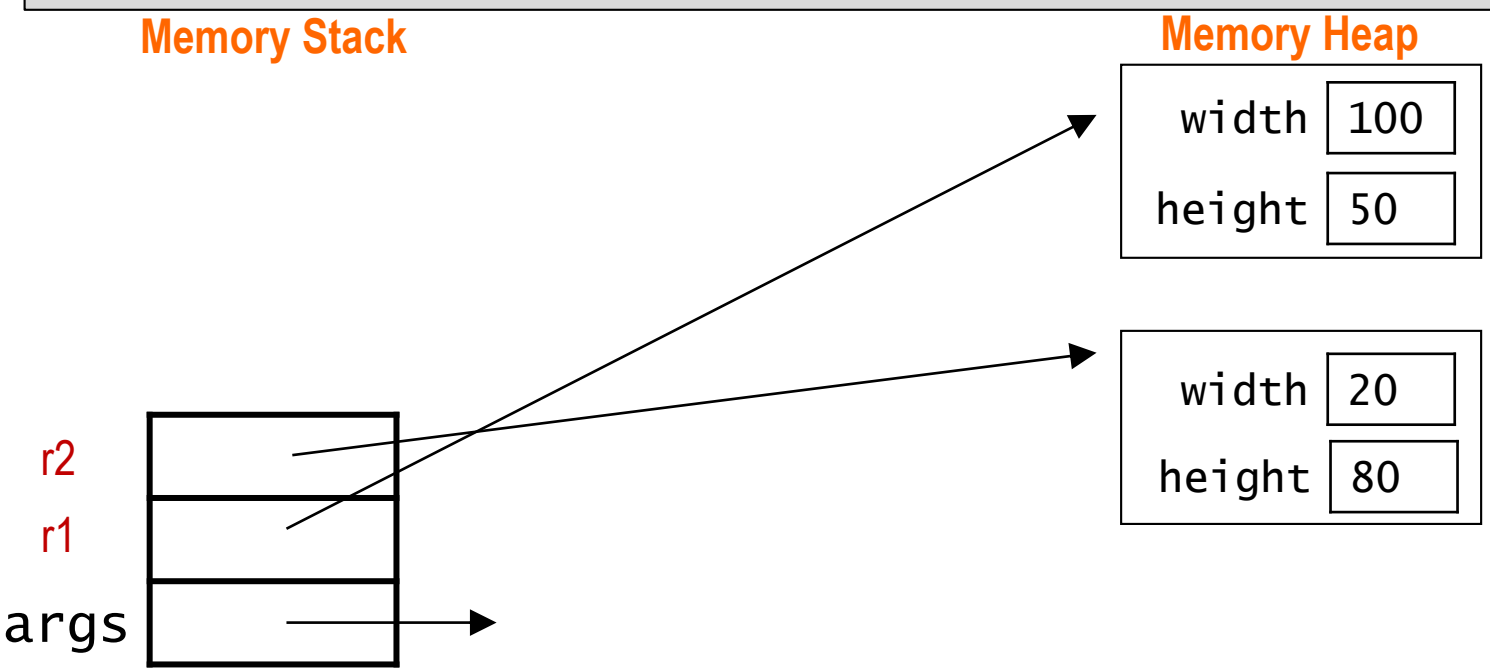
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + r2.area() );
      // grow both rectangles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



A sample Client Program:

memory trace

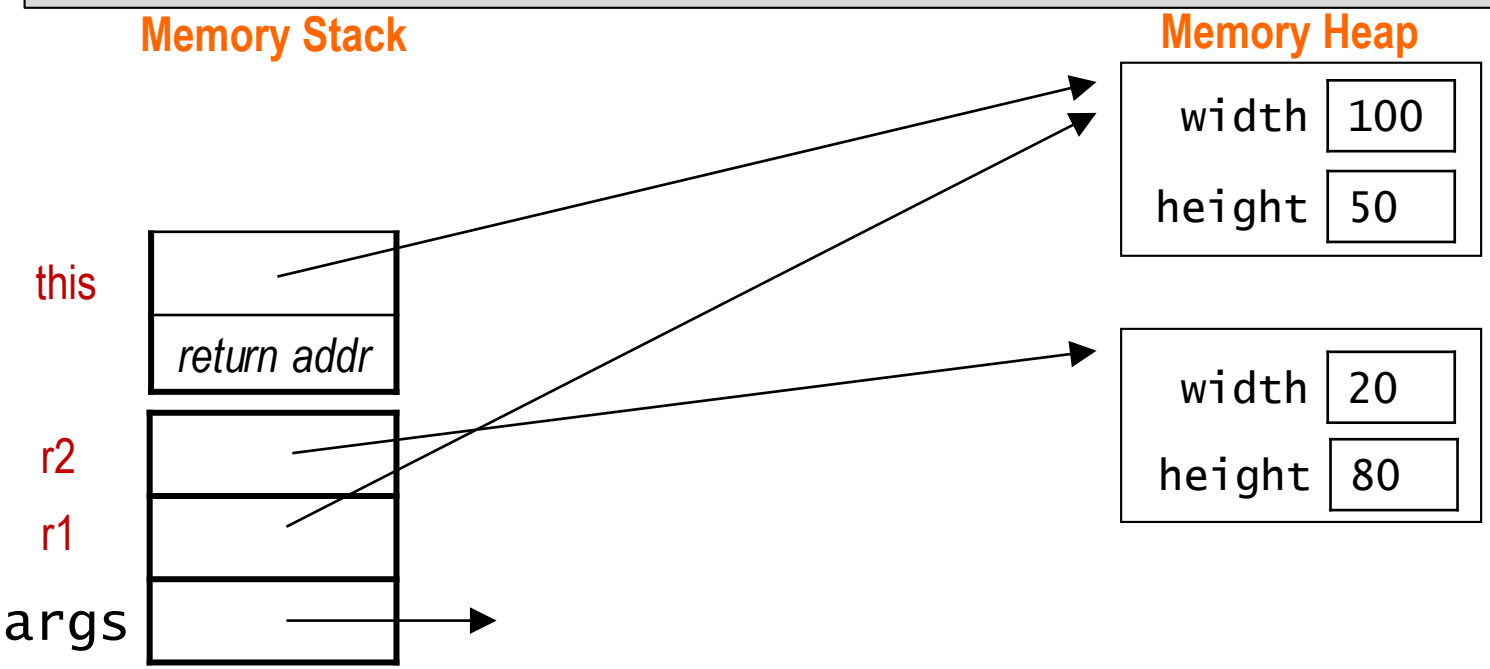
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public double area() {
        return( width*height );
    }
    .
}
```



A sample Client Program:

memory trace

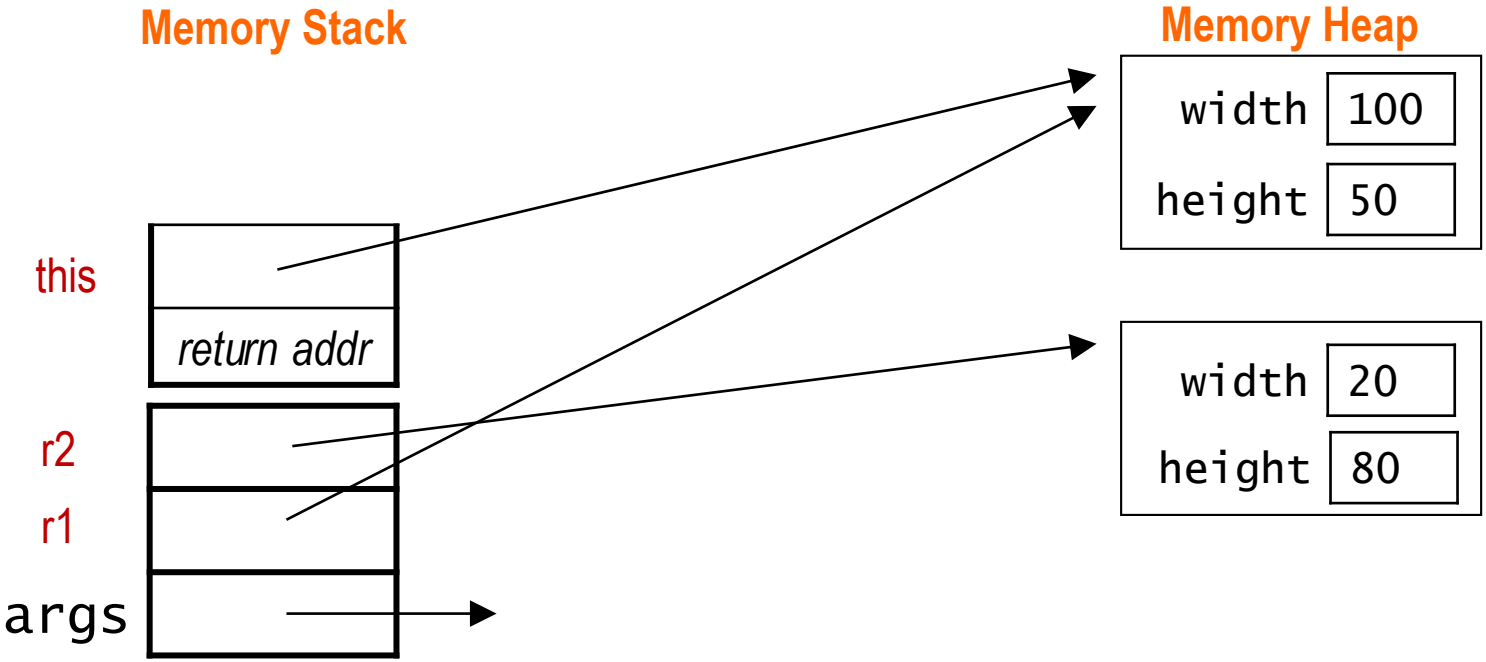
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public double area() {
        return( width*height );
    }
    .
}
```



A sample Client Program:

memory trace

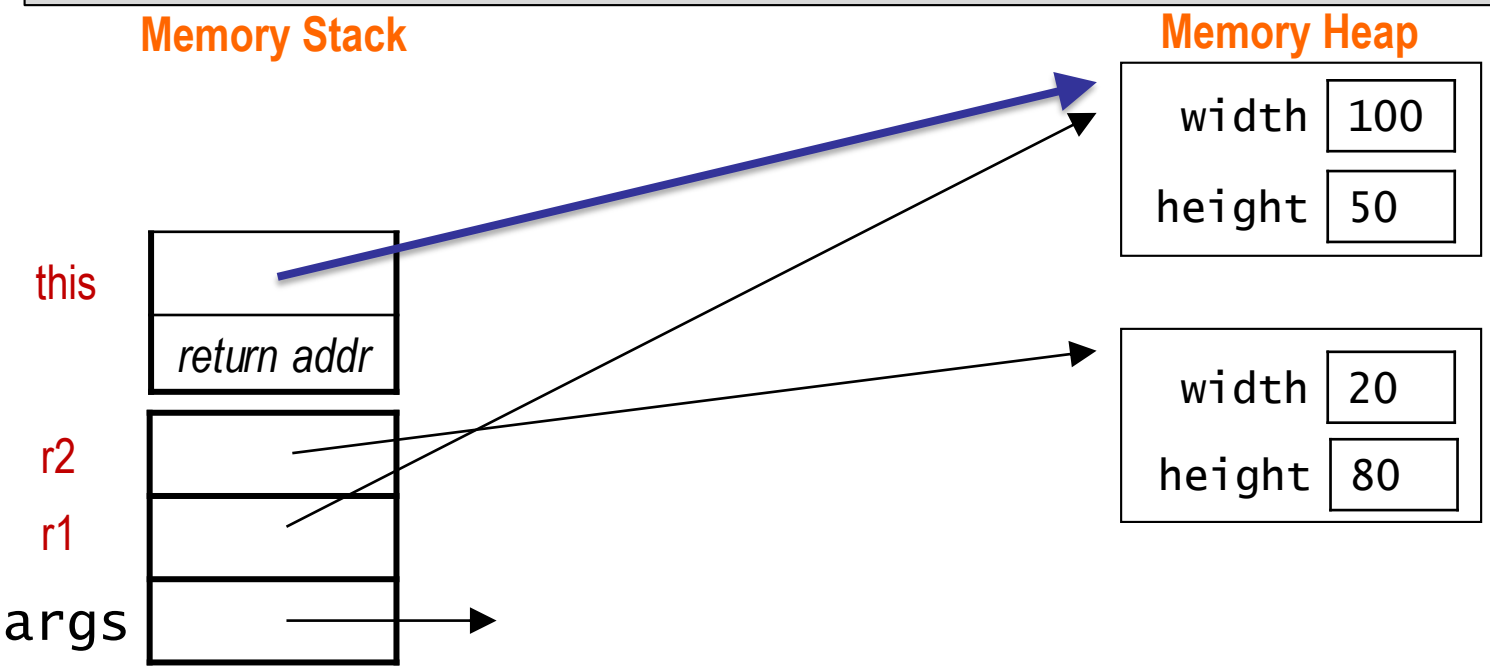
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public double area() {
        return( width*height );
    }
    .
}
```



A sample Client Program:

memory trace

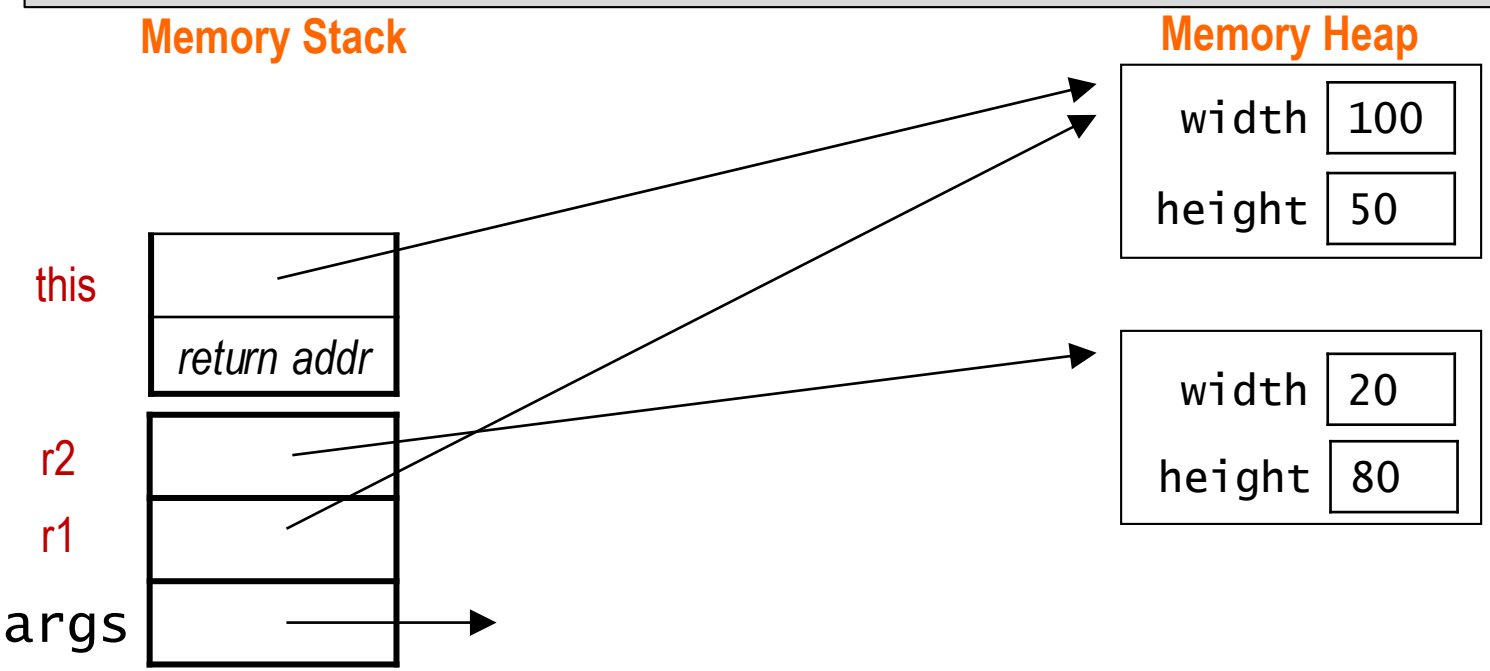
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public double area() {
        return( this.width*this.height );
    }
    .
}
```



A sample Client Program:

memory trace

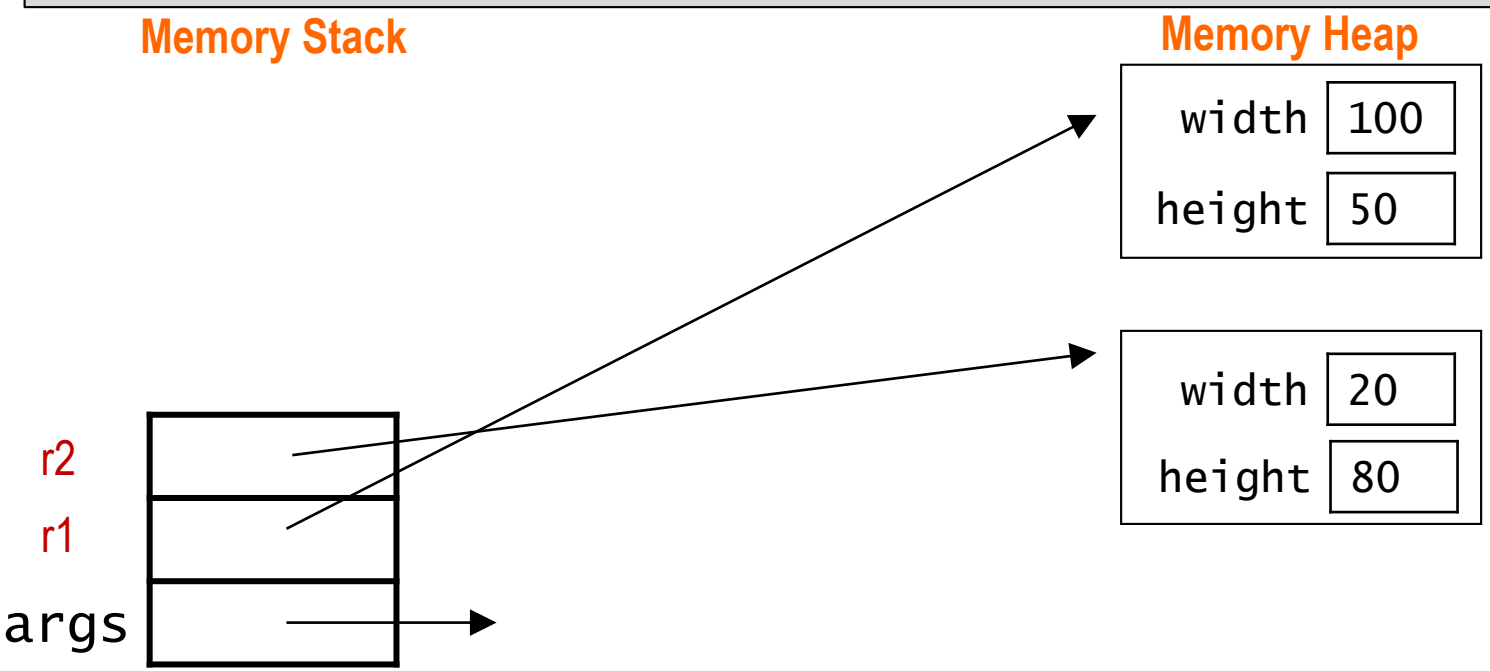
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public double area() {
        return( this.width*this.height ); // returns 5000
    }
    .
}
```



A sample Client Program:

memory trace

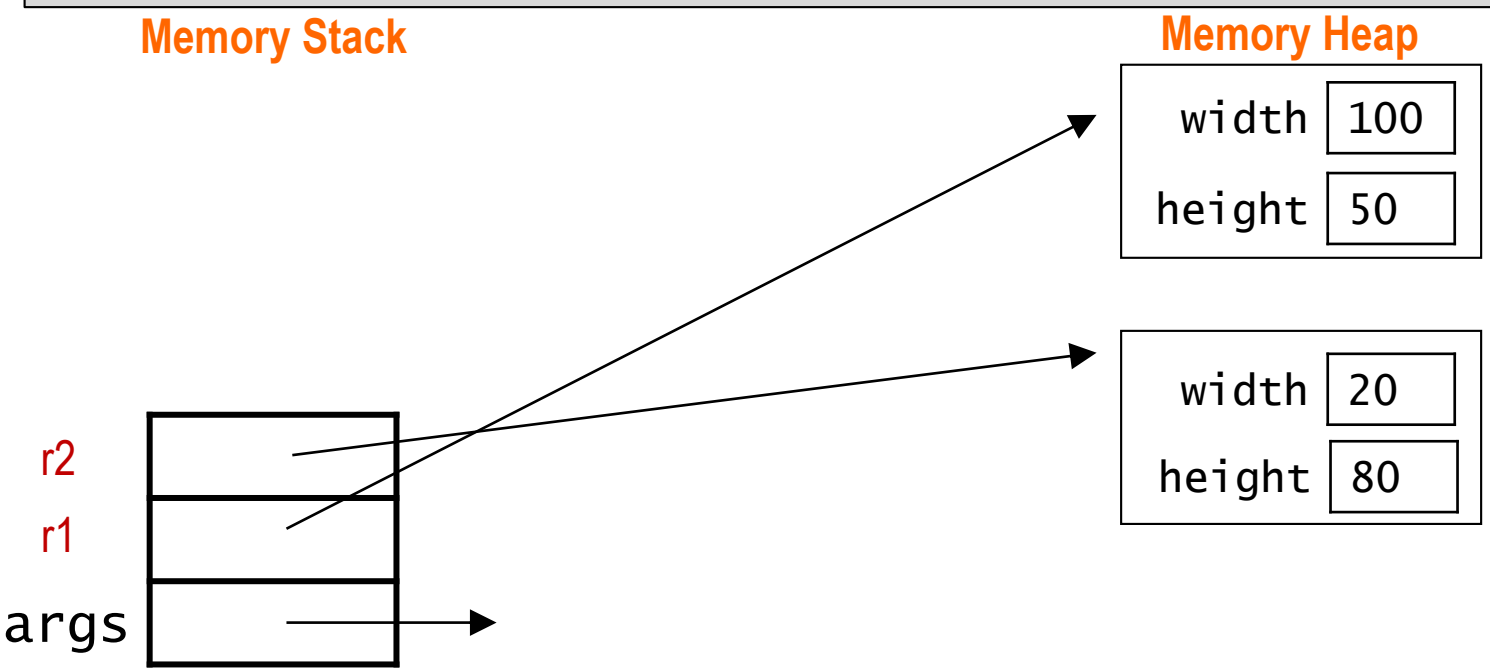
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + 5000 ); // outputs area = 5000
      System.out.println("r2's area = " + r2.area() );
    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

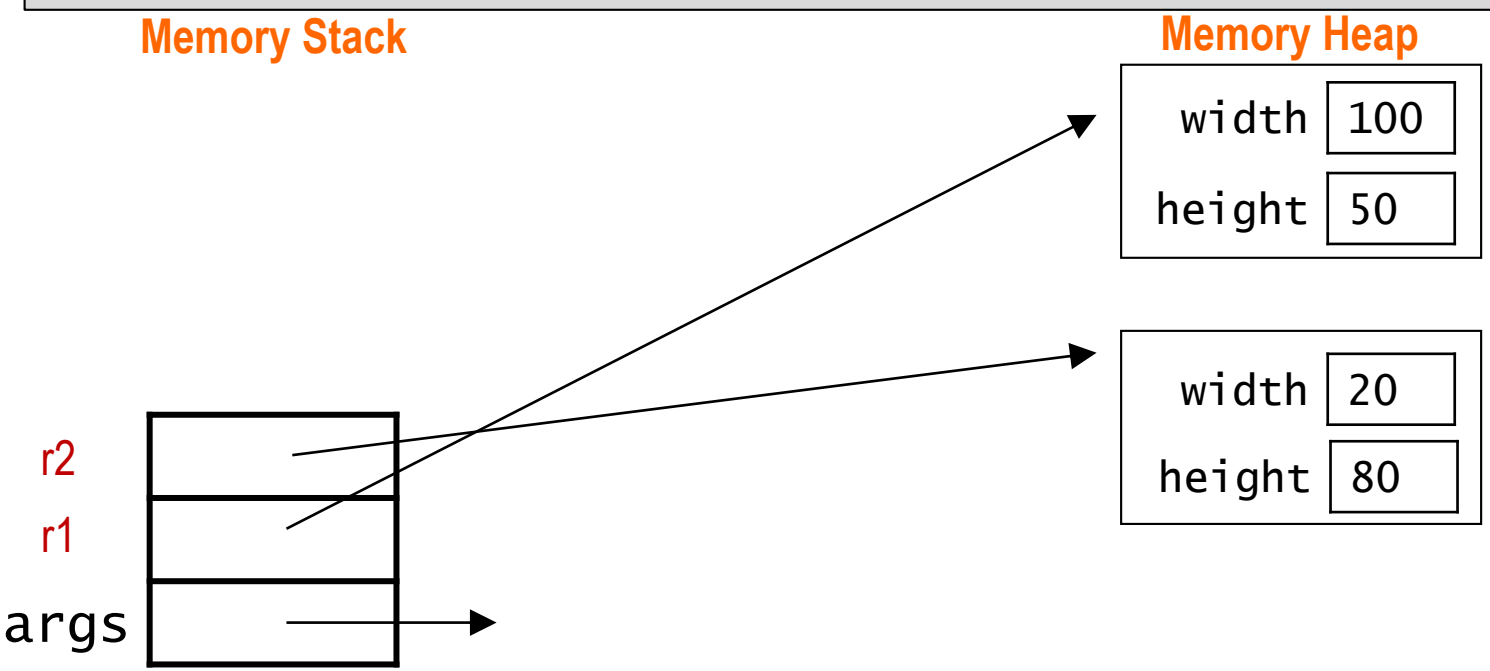
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + 5000 ); // outputs area = 5000
      System.out.println("r2's area = " + r2.area() );
    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

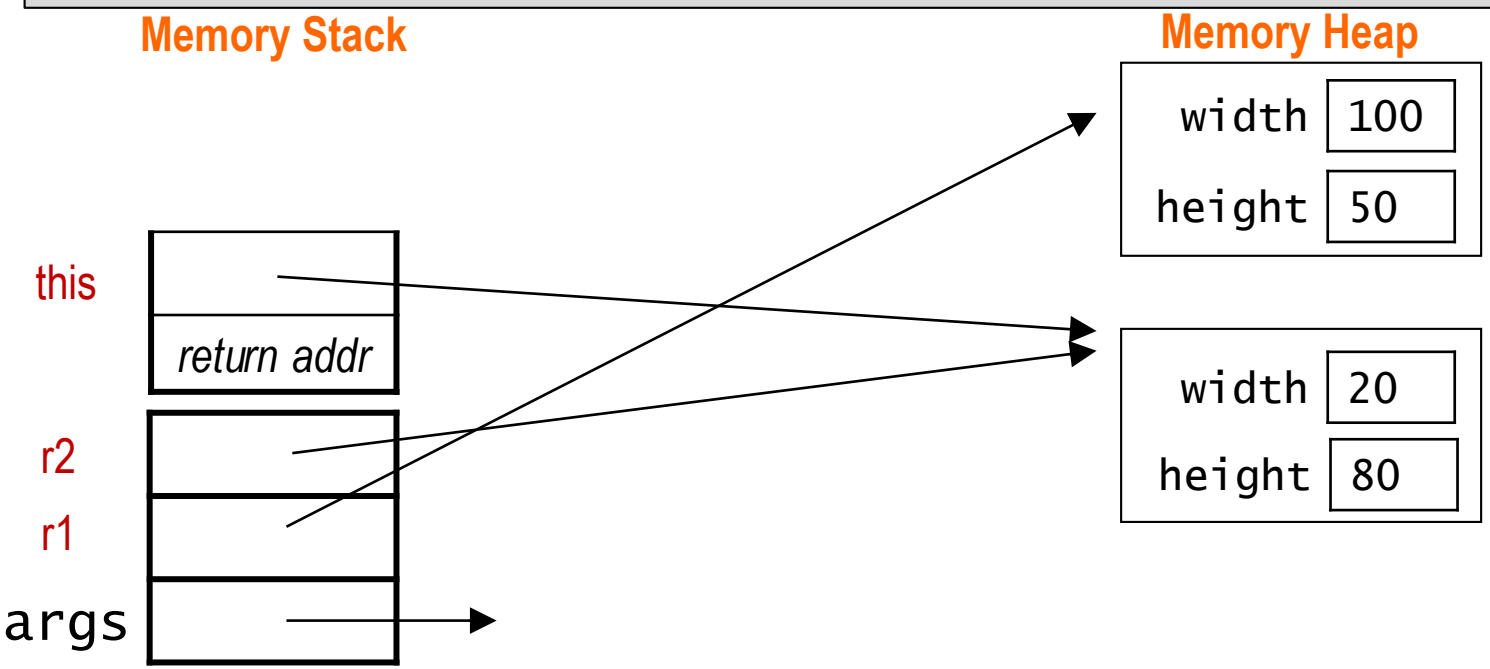
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public double area() {
        return( width*height );
    }
    .
}
```



A sample Client Program:

memory trace

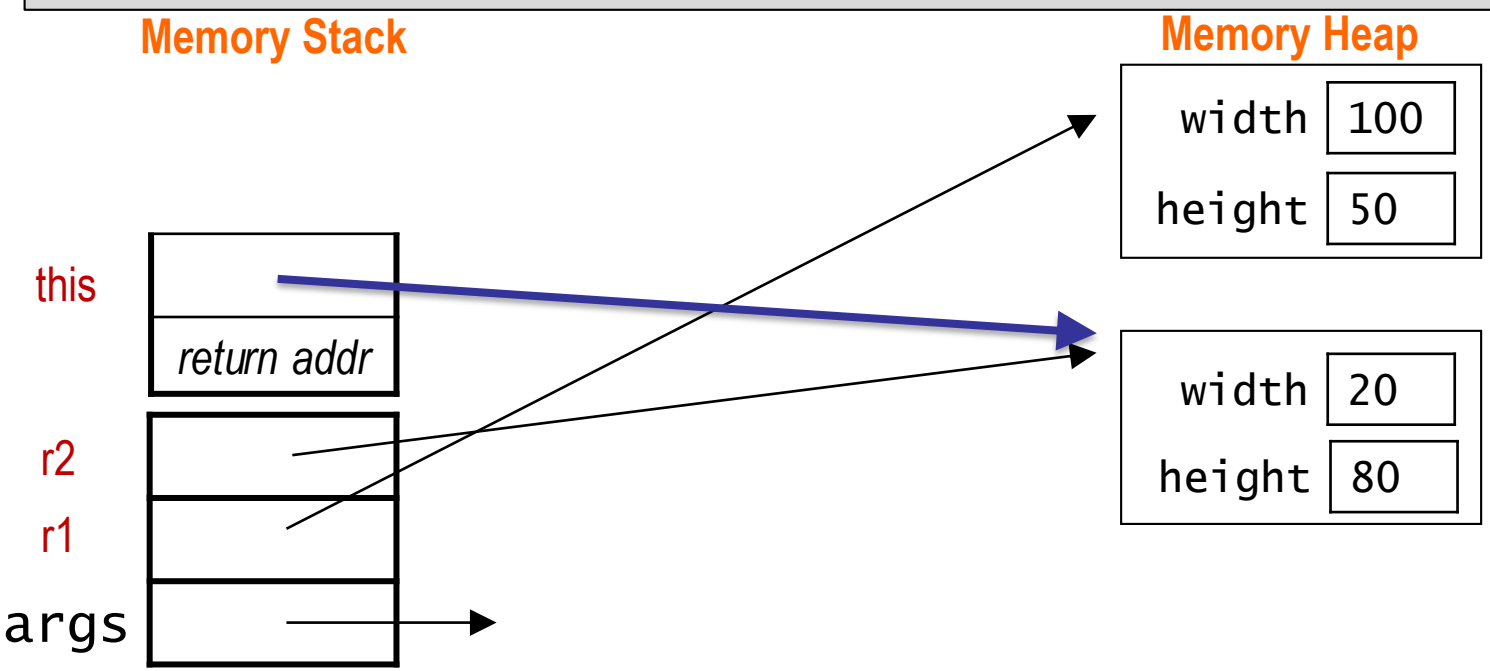
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public double area() {
        return( width*height );
    }
    .
}
```



A sample Client Program:

memory trace

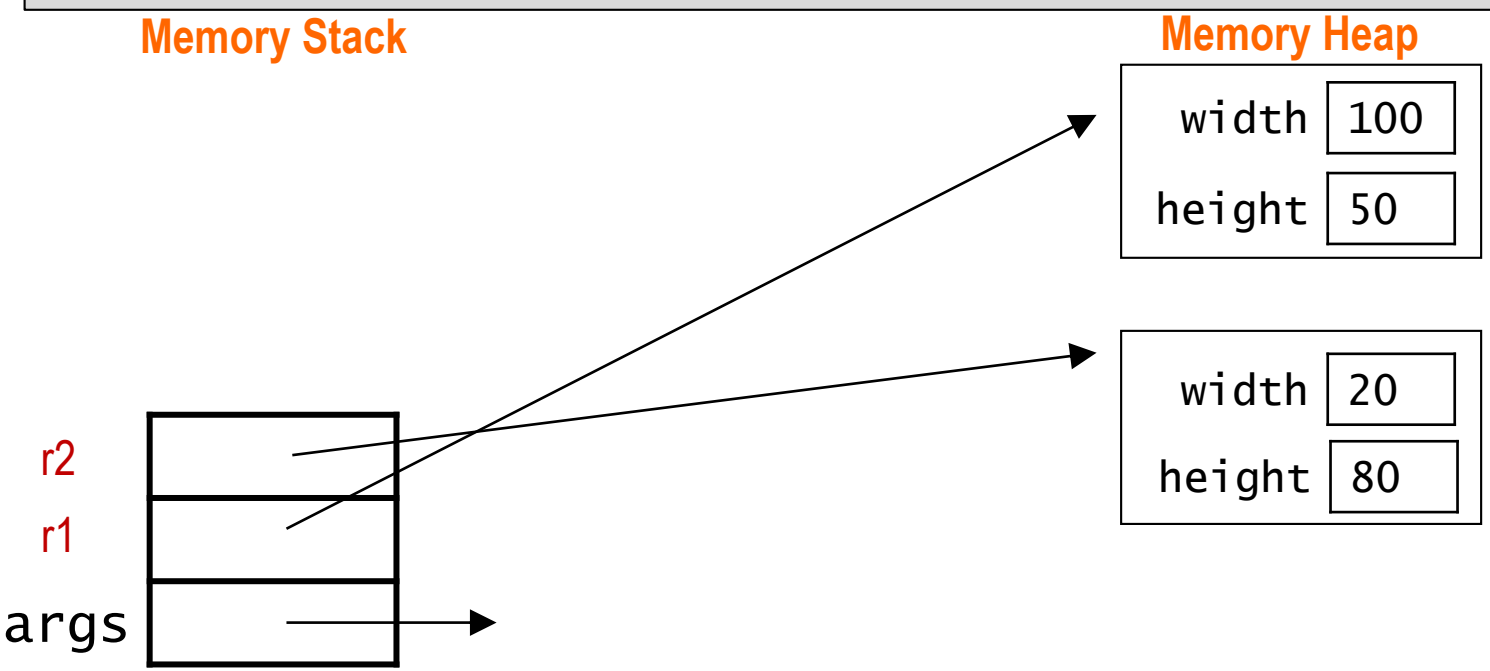
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public double area() {
        return( width*height ); // returns 1600
    }
    .
}
```



A sample Client Program:

memory trace

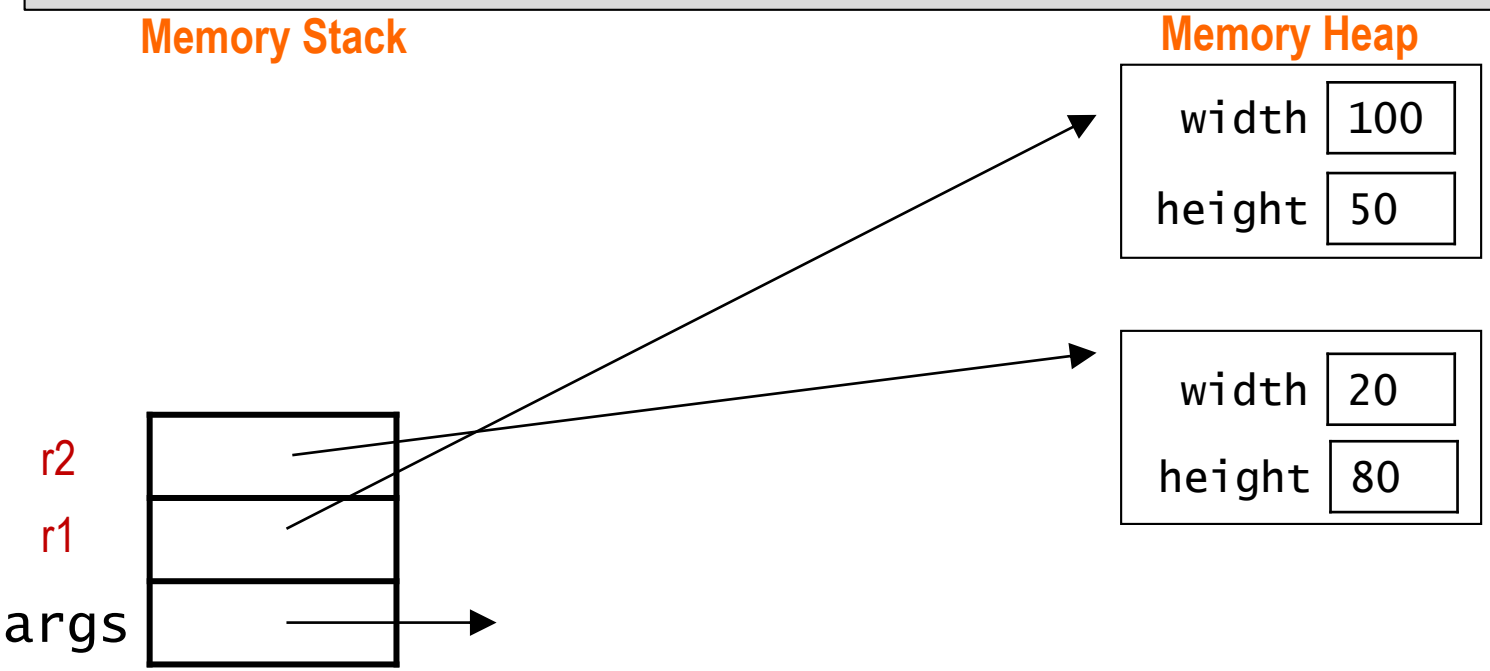
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600
      // grow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

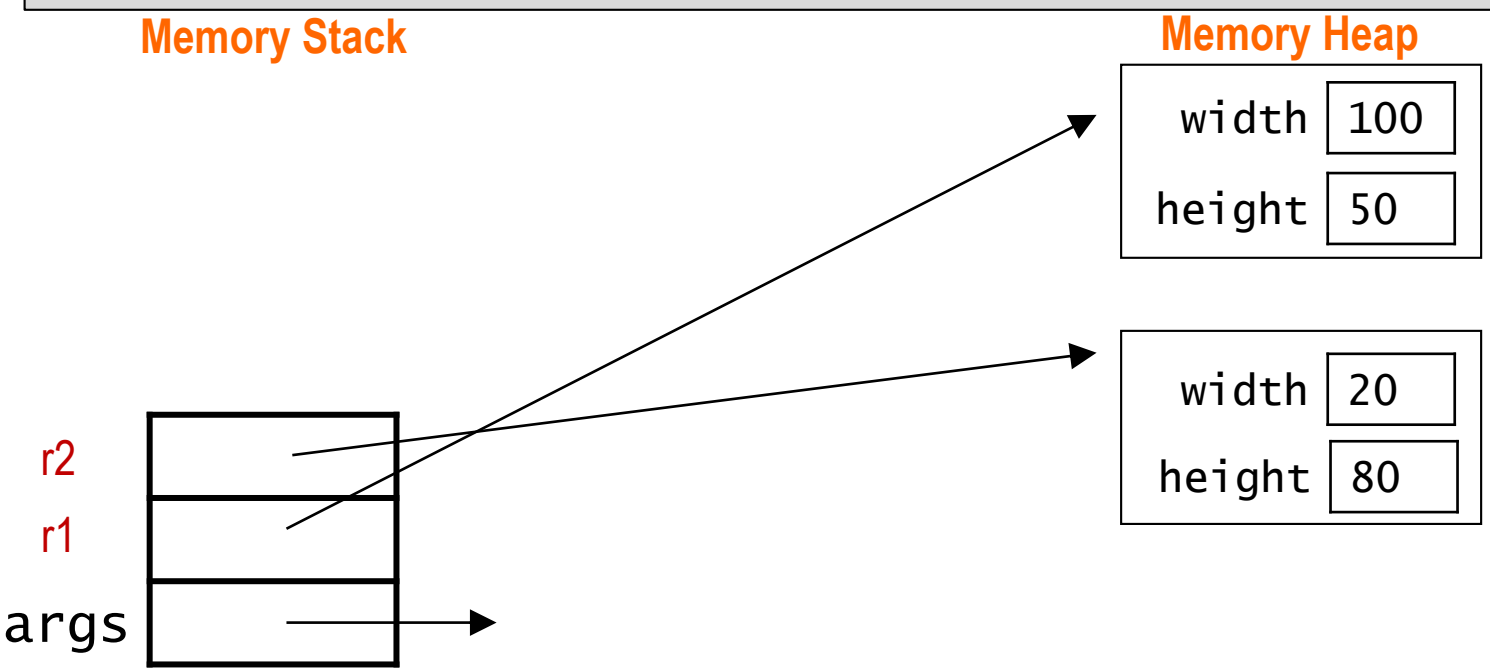
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600
      // grow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

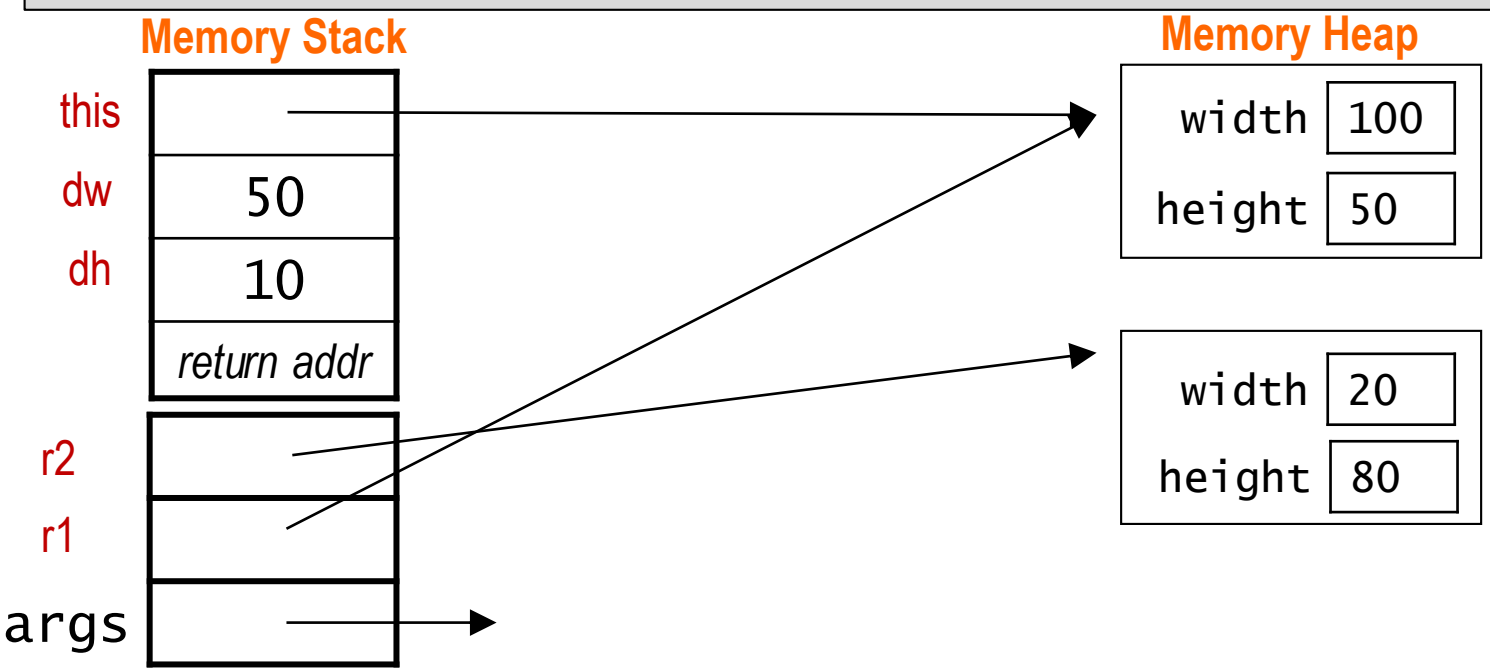
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public double grow( int dw, int dh ) {
        setWidth( width+dw );
        setHeight( height+dh );
    }
}
```



A sample Client Program:

memory trace

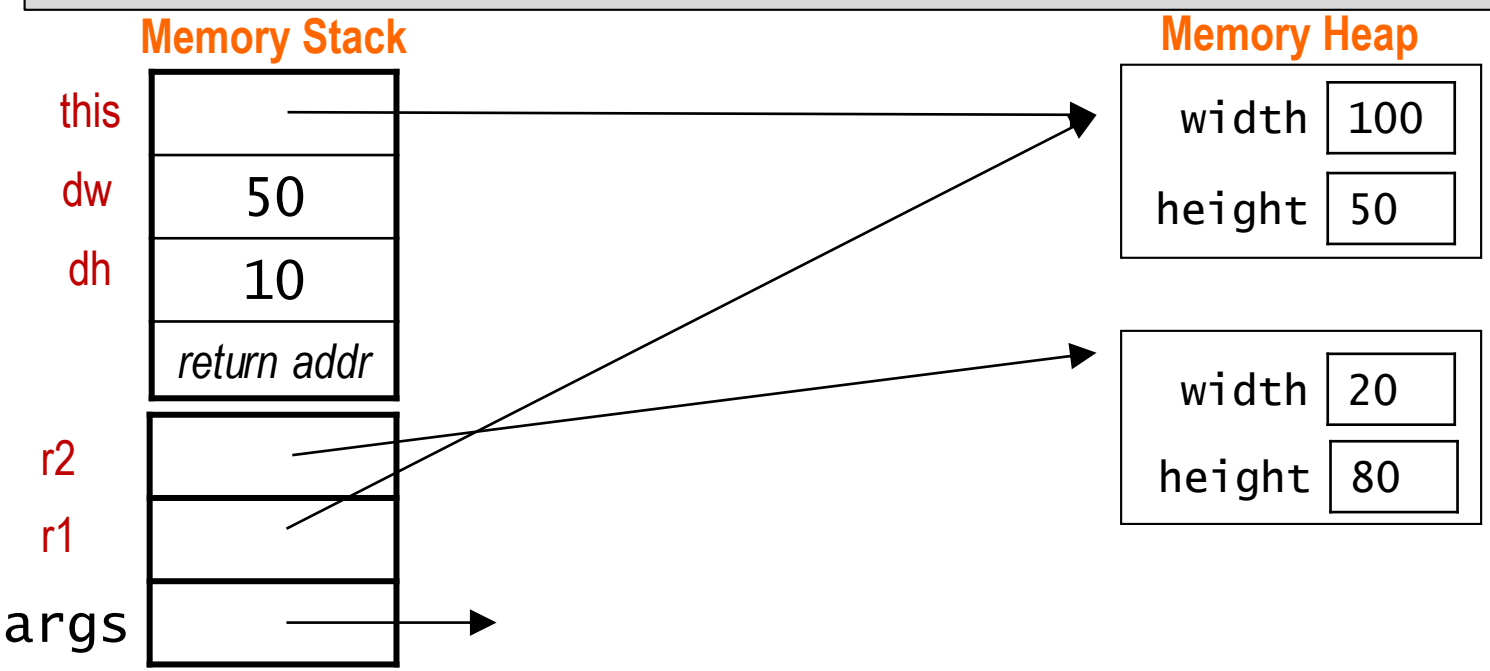
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public double grow( int dw, int dh ) {
        setWidth( width+dw );
        setHeight( height+dh );
    }
}
```



A sample Client Program:

memory trace

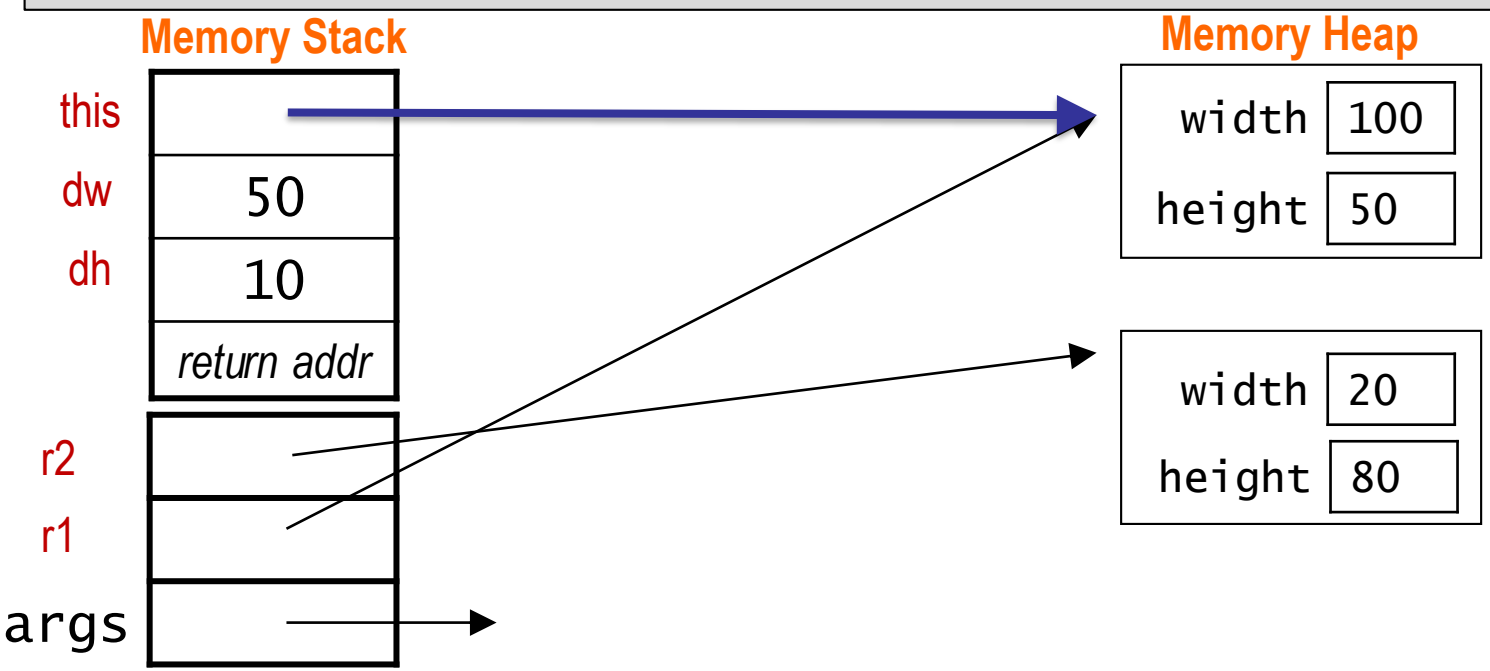
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public double grow( int dw, int dh ) {
        setWidth( width+dw );
        setHeight( height+dh );
    }
}
```



A sample Client Program:

memory trace

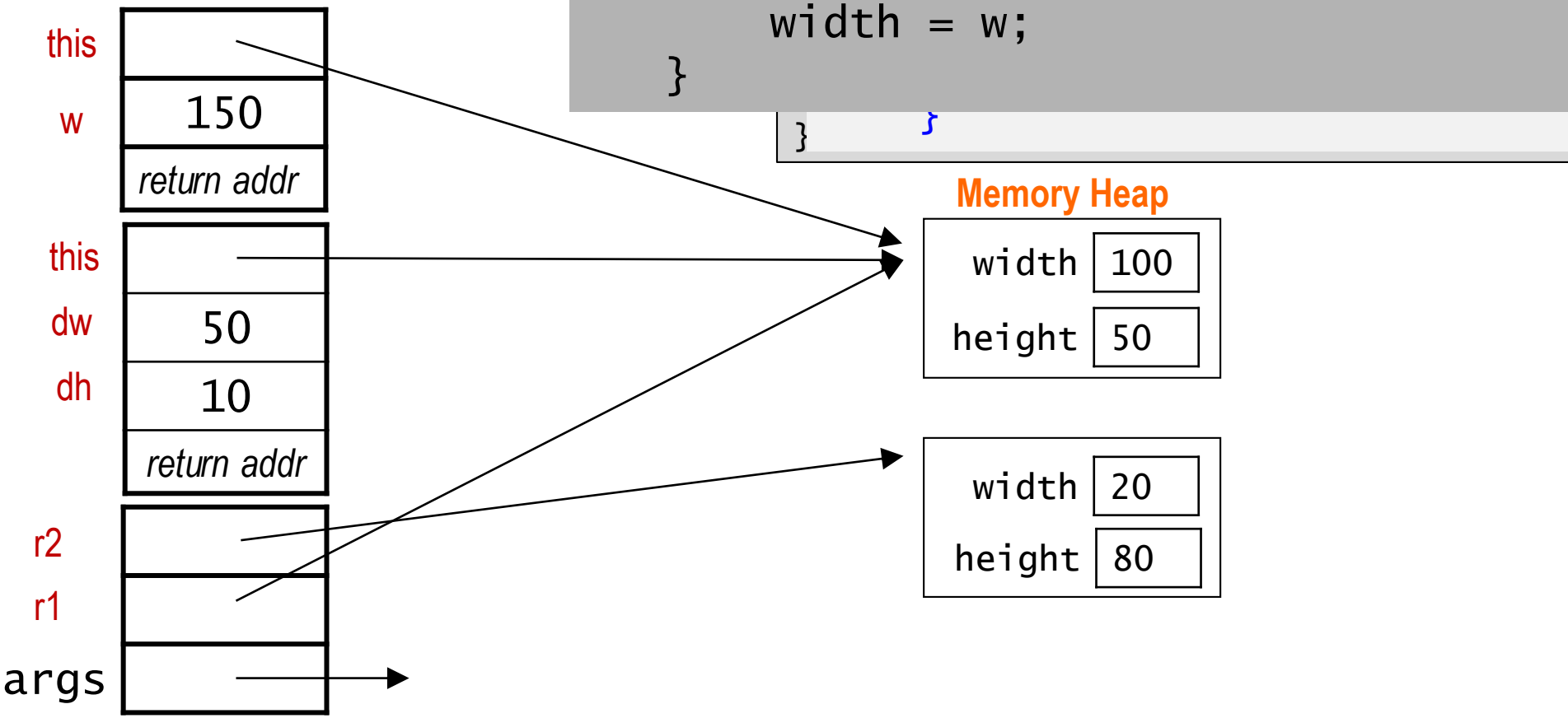
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public double grow( int dw, int dh ) {
        this.setWidth( width+dw );
        setHeight( height+dh );
    }
}
```



A sample Client Program:

memory trace

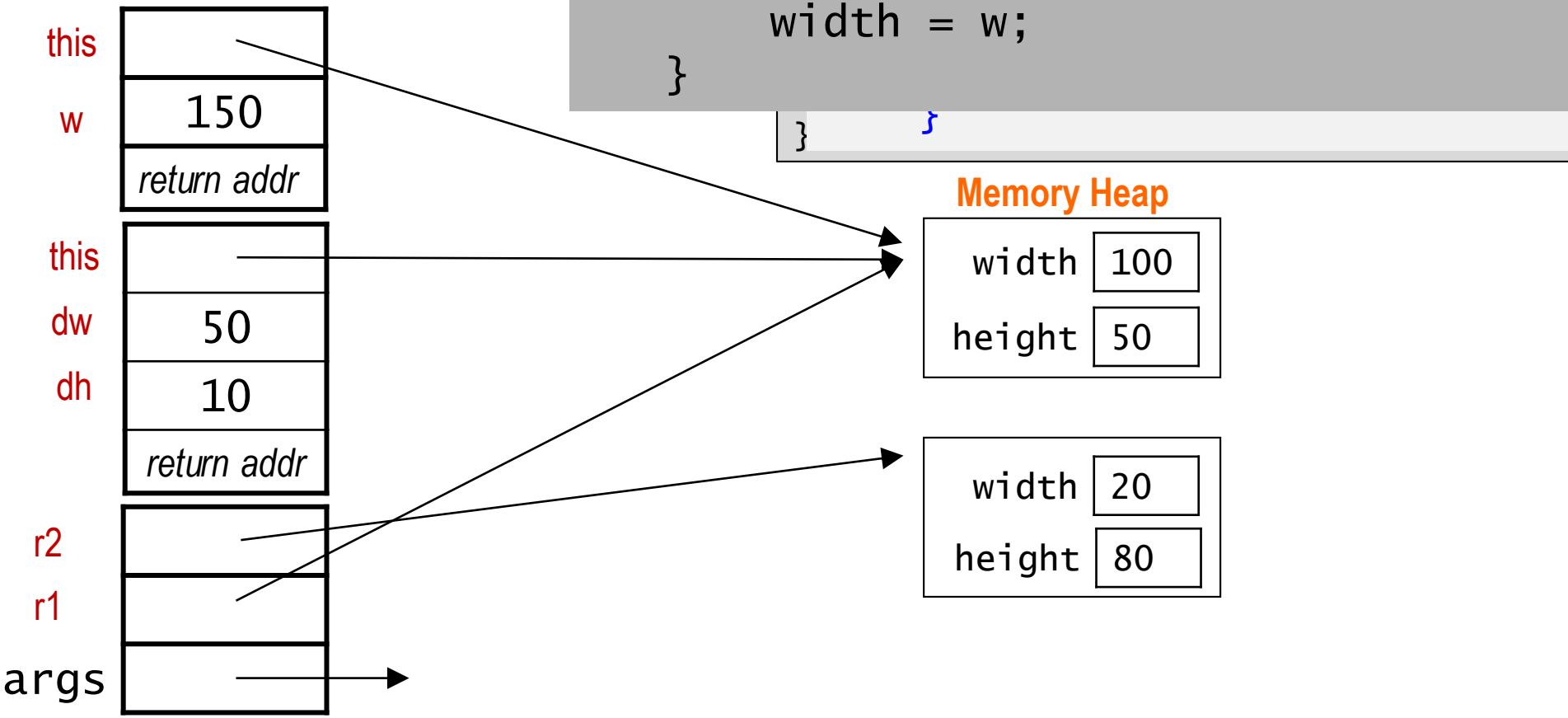
```
public class Rectangle {
    public void setWidth(int w) {
        if (w <= 0) {
            throw new IllegalArgumentExc
        }

        width = w;
    }
}
```



A sample Client Program:

memory trace

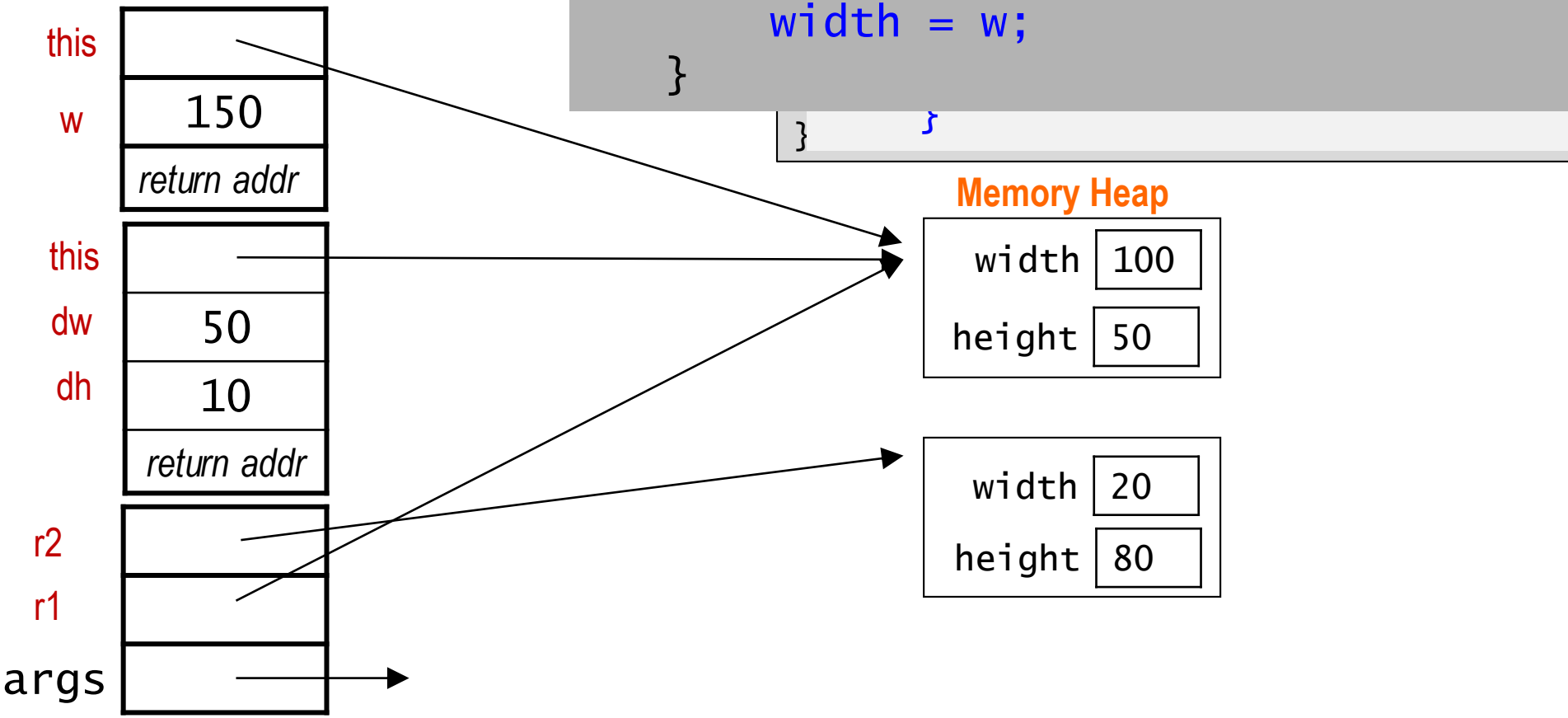
```
public class Rectangle {
    public void setWidth(int w) {
        if (w <= 0) {
            throw new IllegalArgumentExc
        }

        width = w;
    }
}
```



A sample Client Program:

memory trace

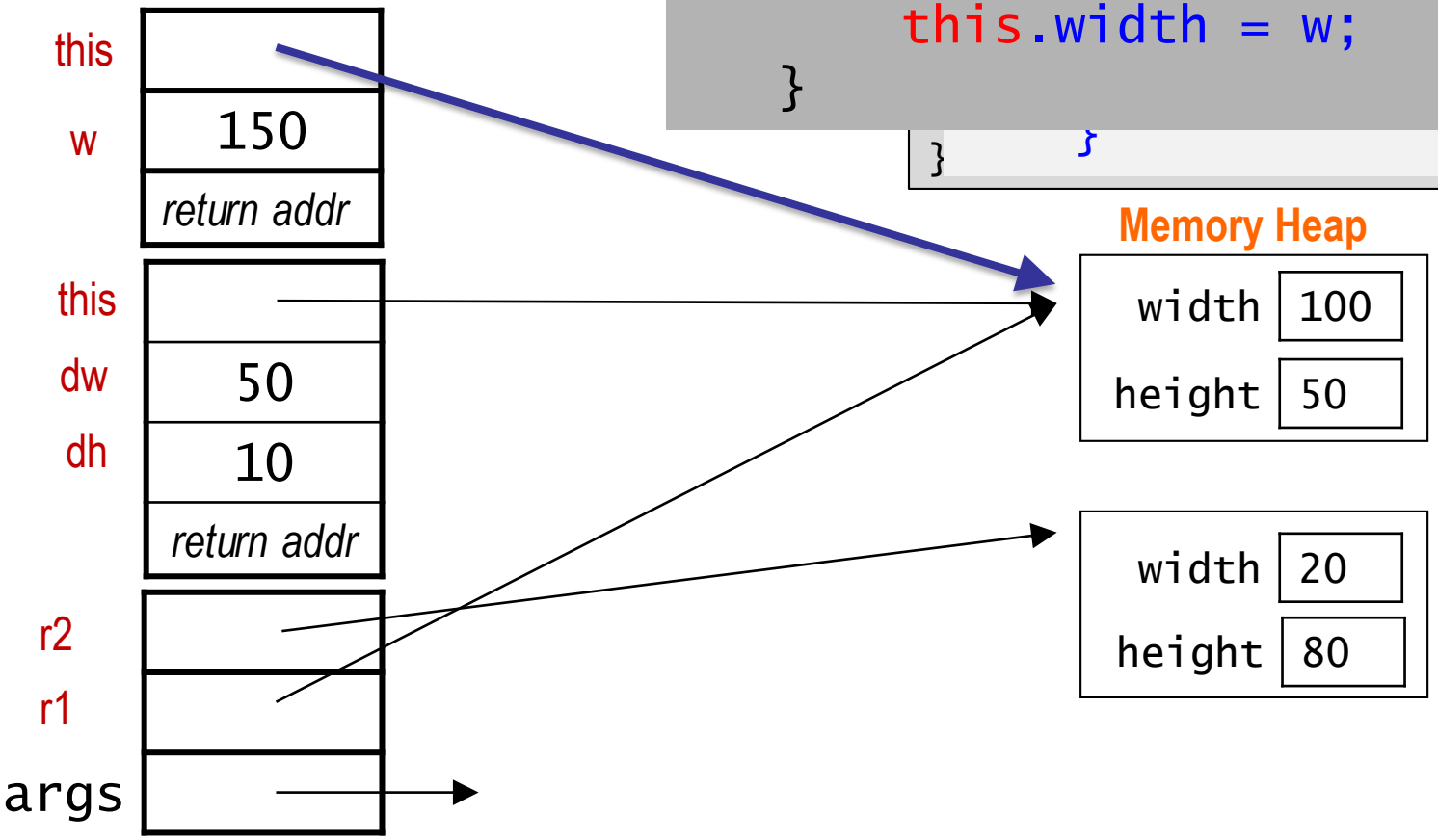
```
public class Rectangle {
    public void setWidth(int w) {
        if (w <= 0) {
            throw new IllegalArgumentExc
        }

        width = w;
    }
}
```



A sample Client Program:

memory trace

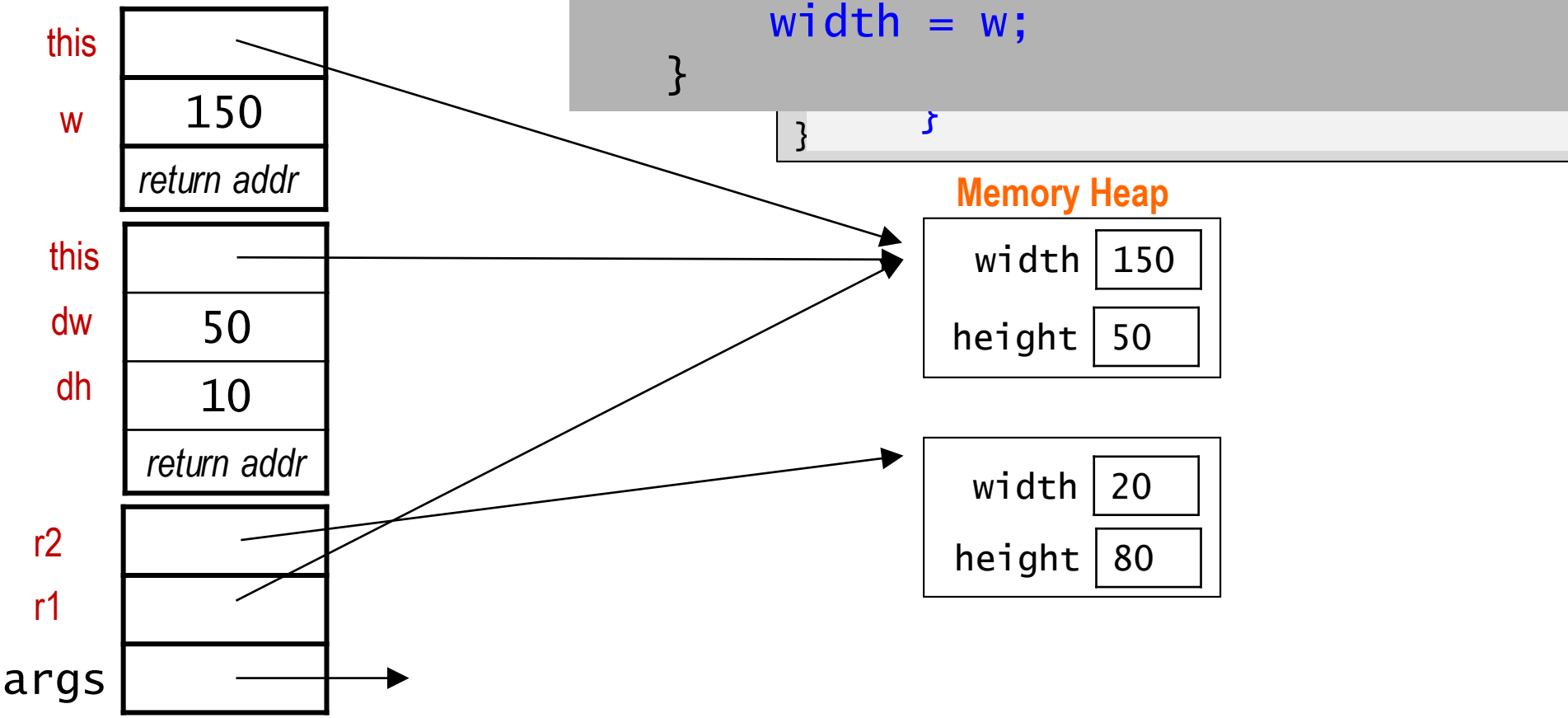
```
public class Rectangle {
    public void setWidth(int w) {
        if (w <= 0) {
            throw new IllegalArgumentExc
        }

        this.width = w;
    }
}
```



A sample Client Program:

memory trace

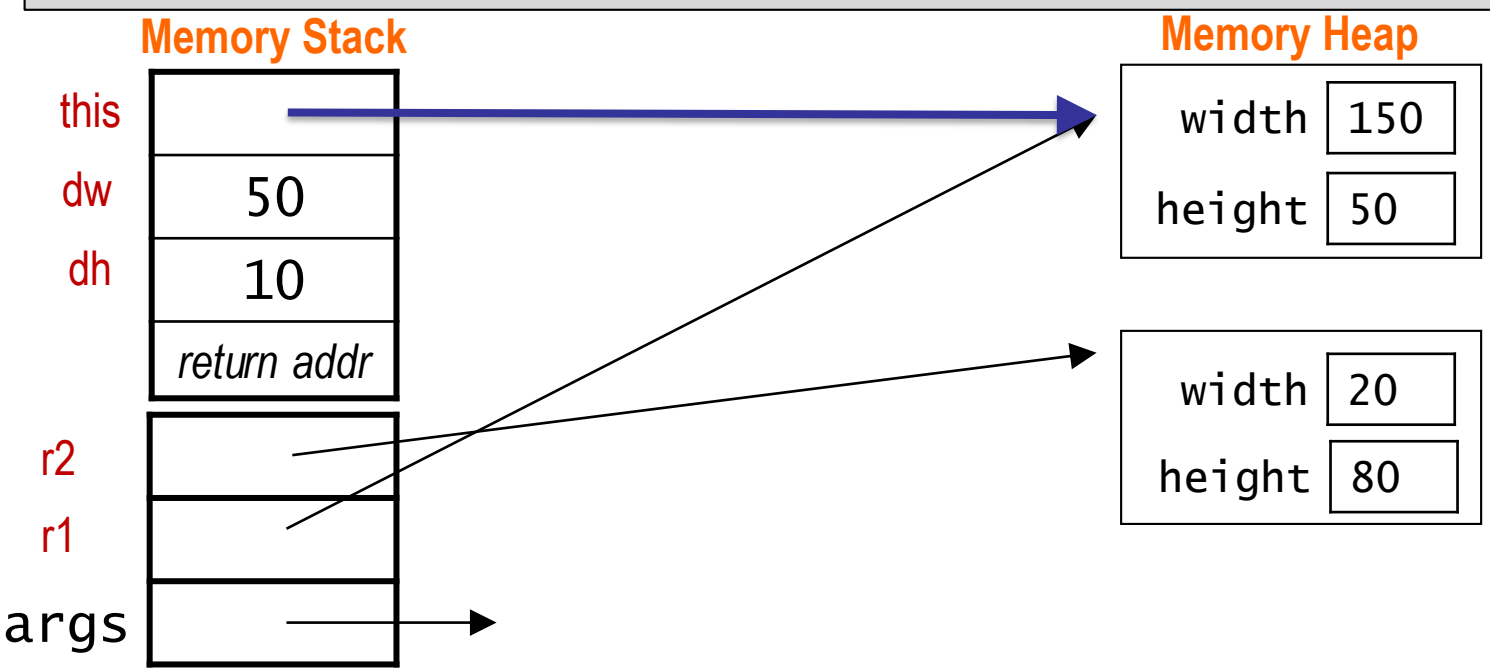
```
public class Rectangle {
    public void setWidth(int w) {
        if (w <= 0) {
            throw new IllegalArgumentExc
        }

        width = w;
    }
}
```



A sample Client Program:

memory trace

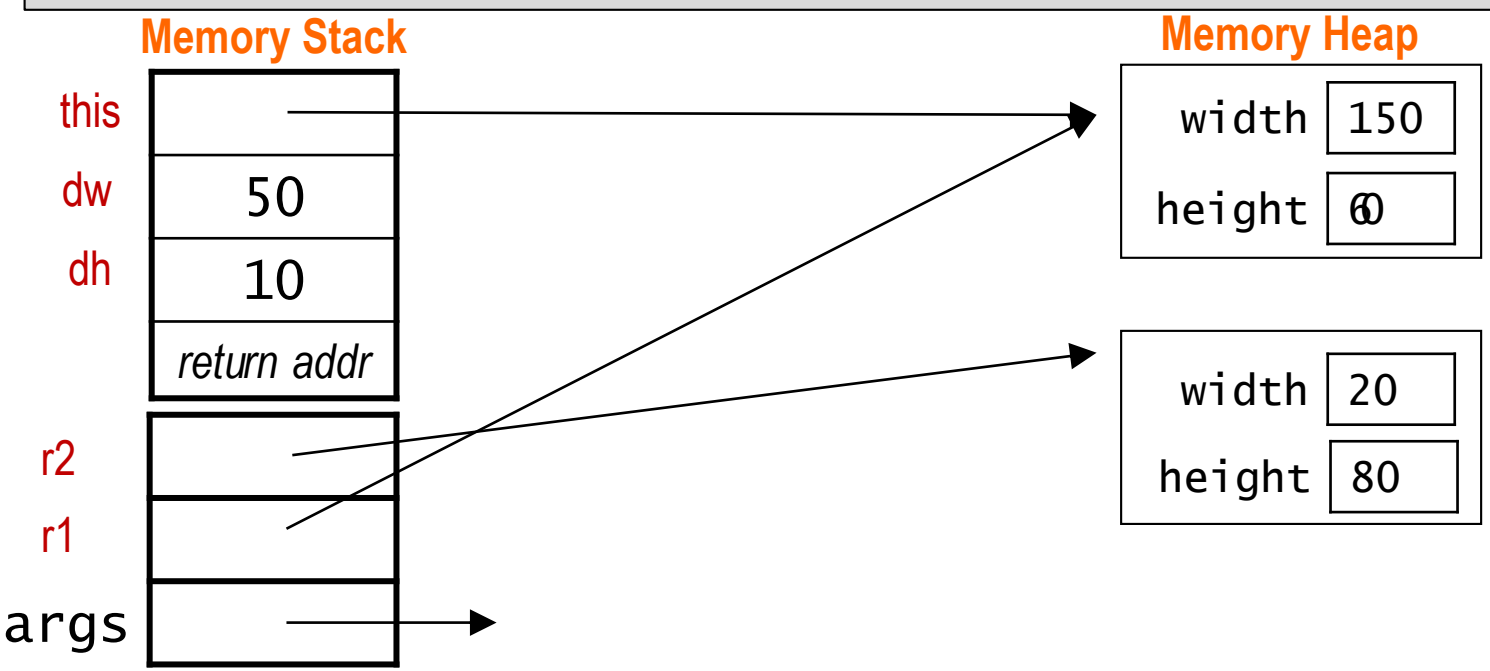
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public double grow( int dw, int dh ) {
        setWidth( width+dw );
        setHeight( height+dh );
    }
}
```



A sample Client Program:

memory trace

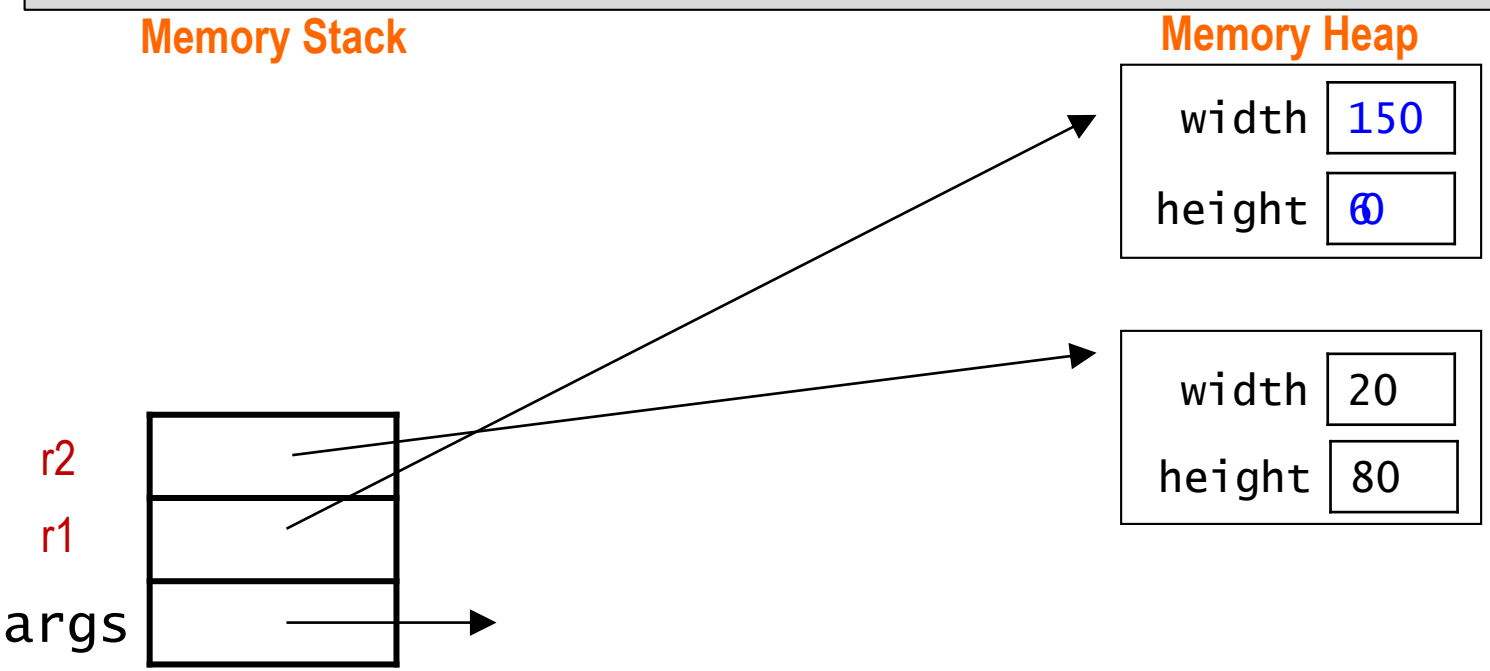
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public double grow( int dw, int dh ) {
        setWidth( width+dw );
        setHeight( height+dh );
    }
}
```



A sample Client Program:

memory trace

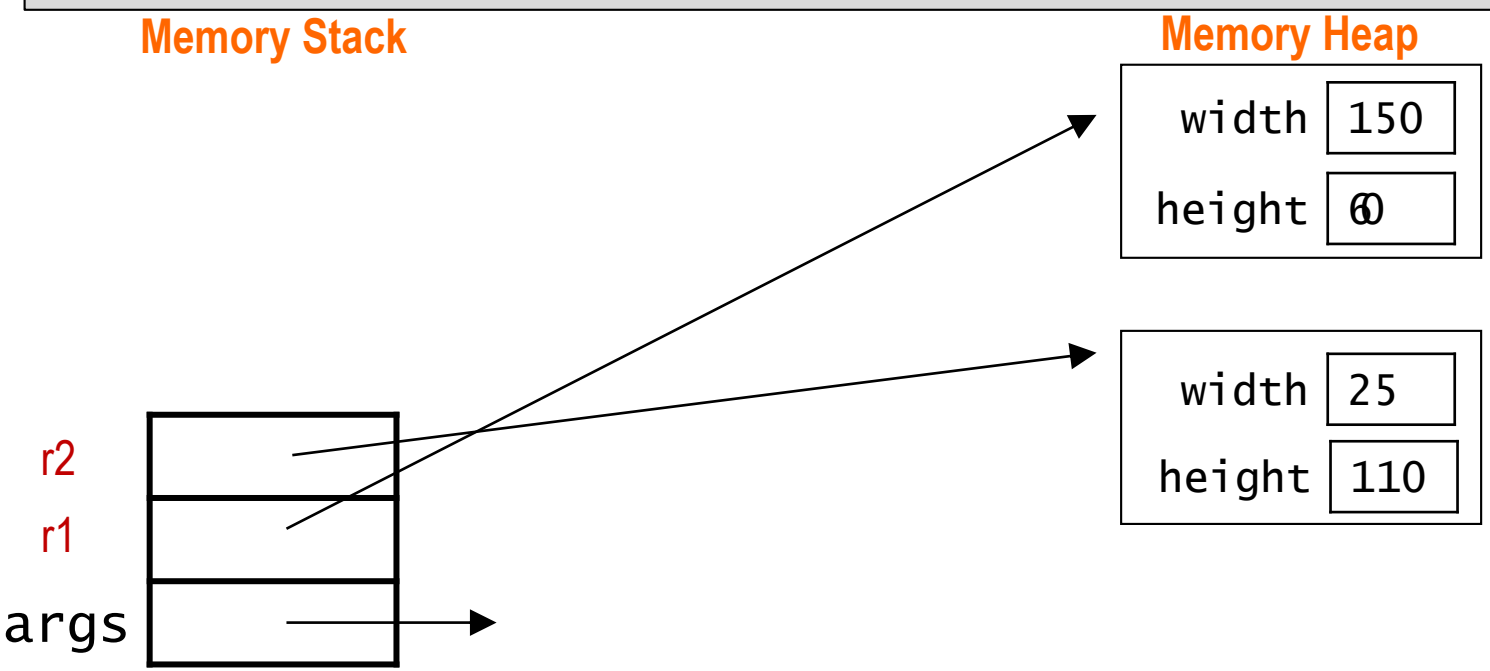
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600
      // grow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



A sample Client Program:

memory trace

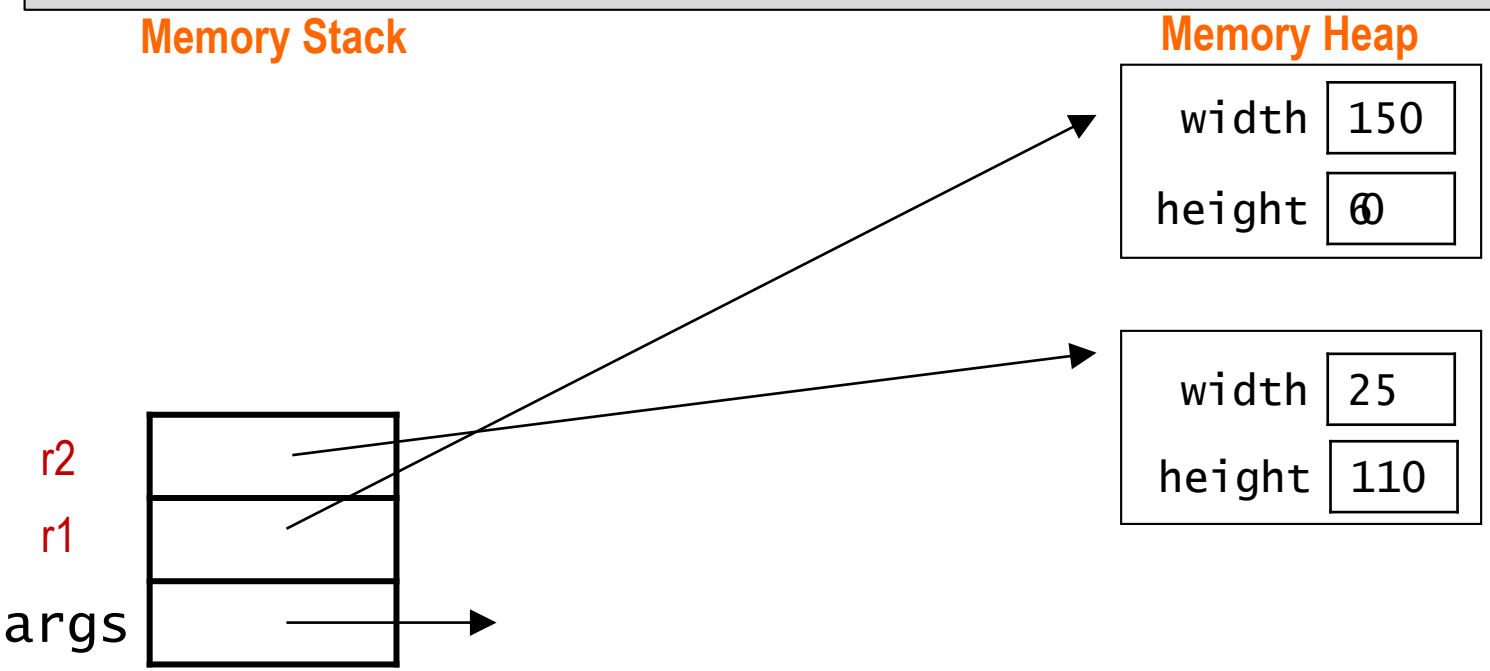
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600
      // grow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

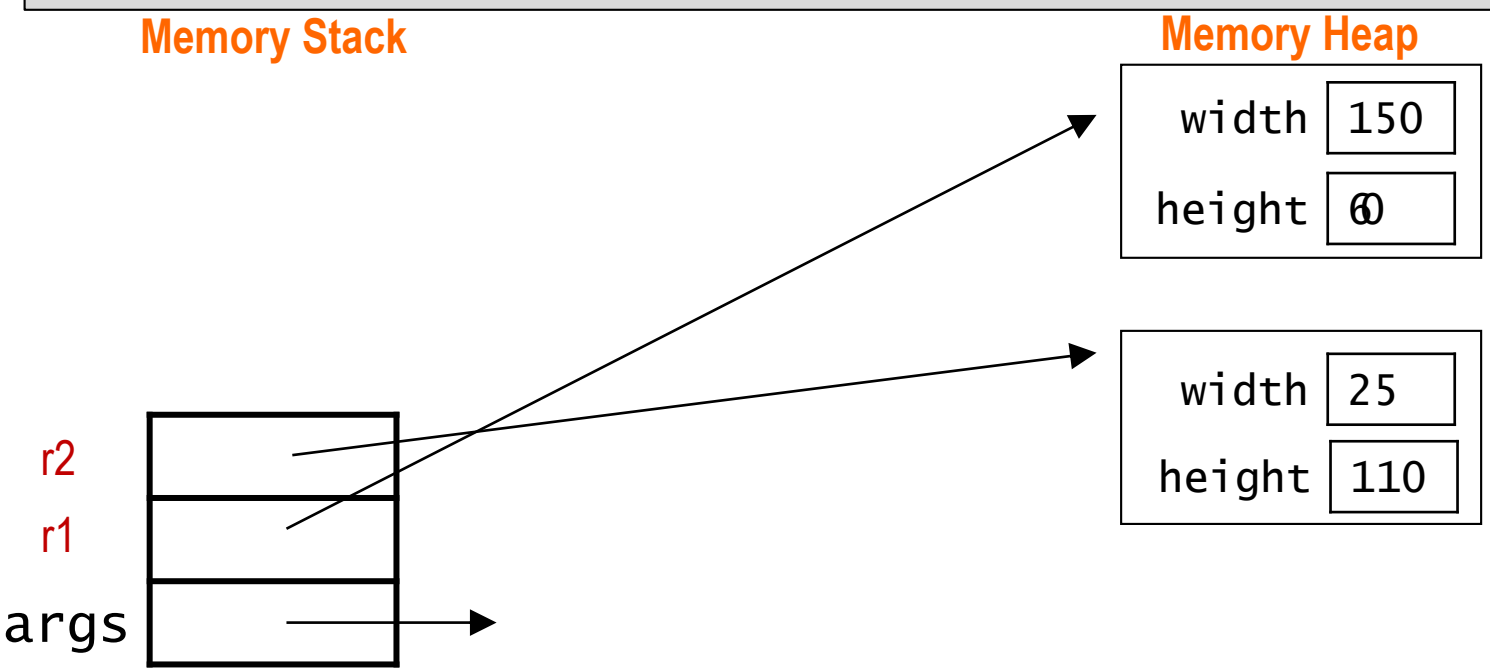
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600
      // grow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

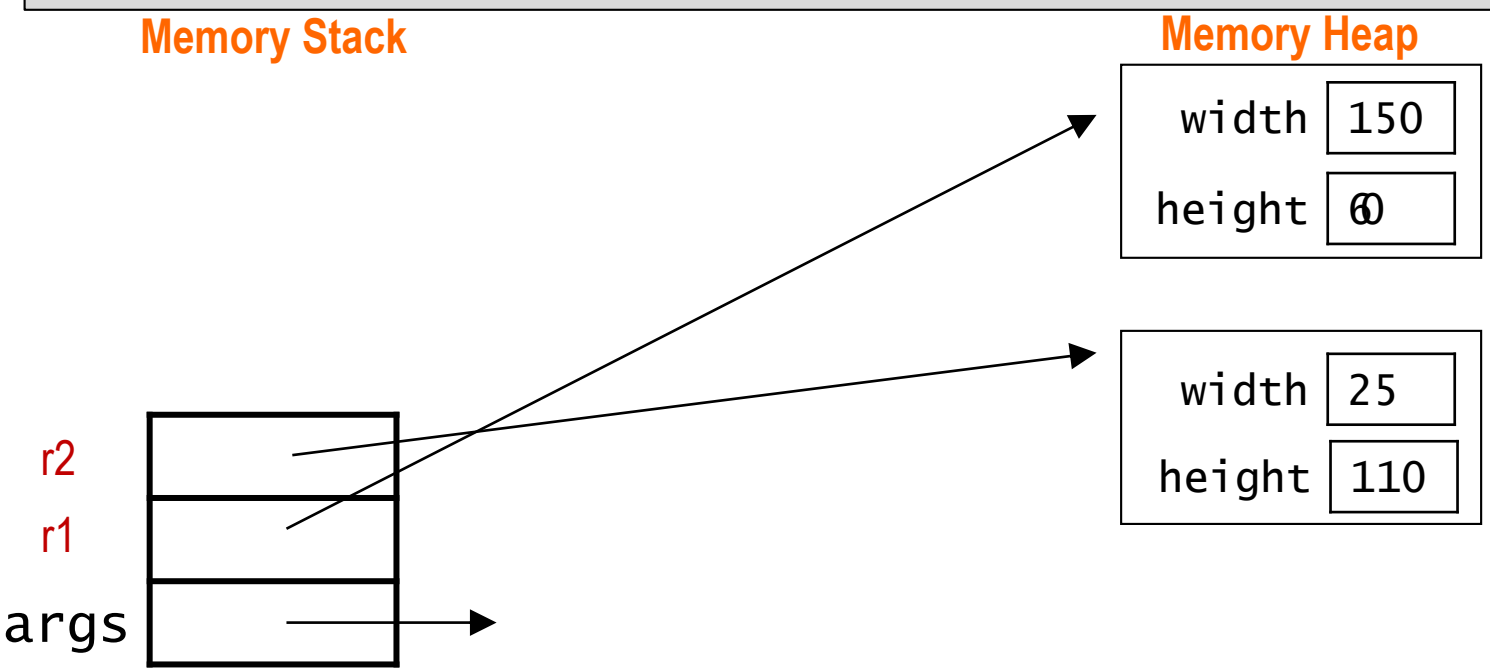
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600
      // grow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

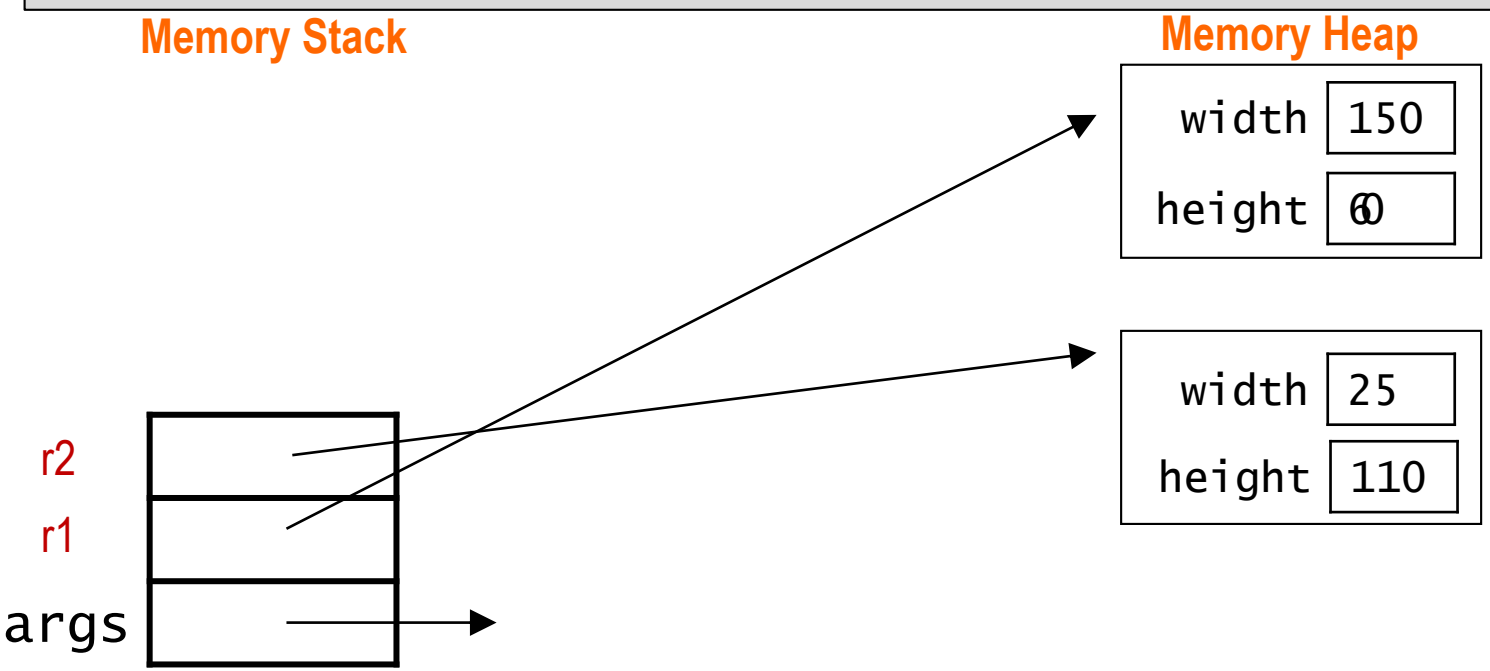
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600
      // grow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

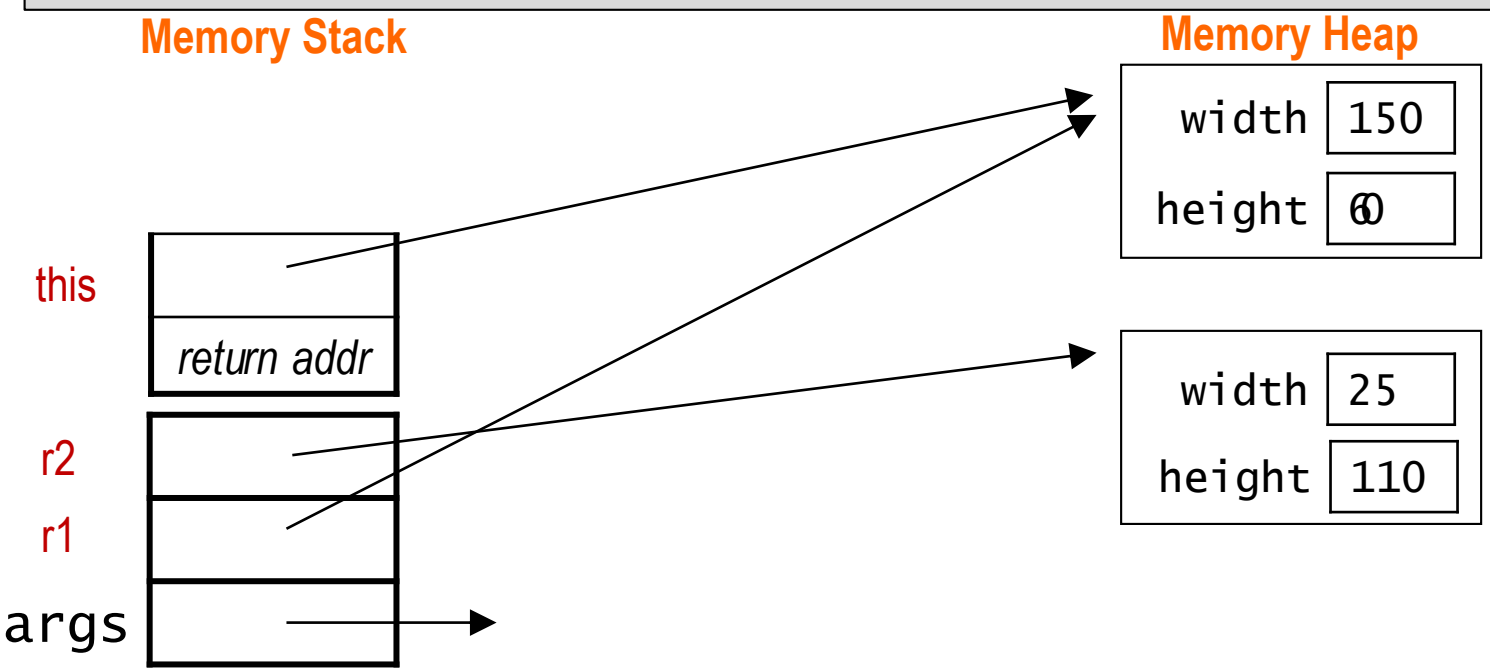
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public String toString() {
        return width + " x " + height;
    }
  }
}
```



A sample Client Program:

memory trace

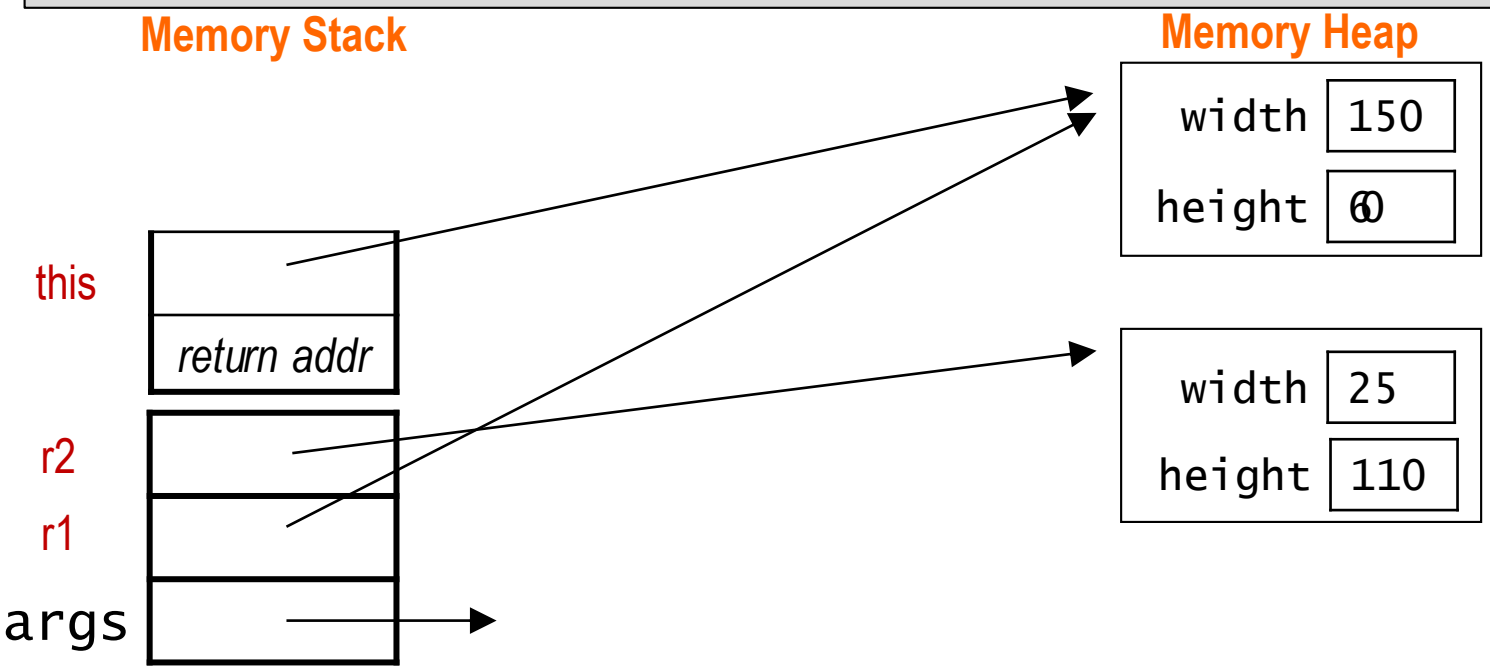
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public String toString() {
        return width + " x " + height;
    }
  }
}
```



A sample Client Program:

memory trace

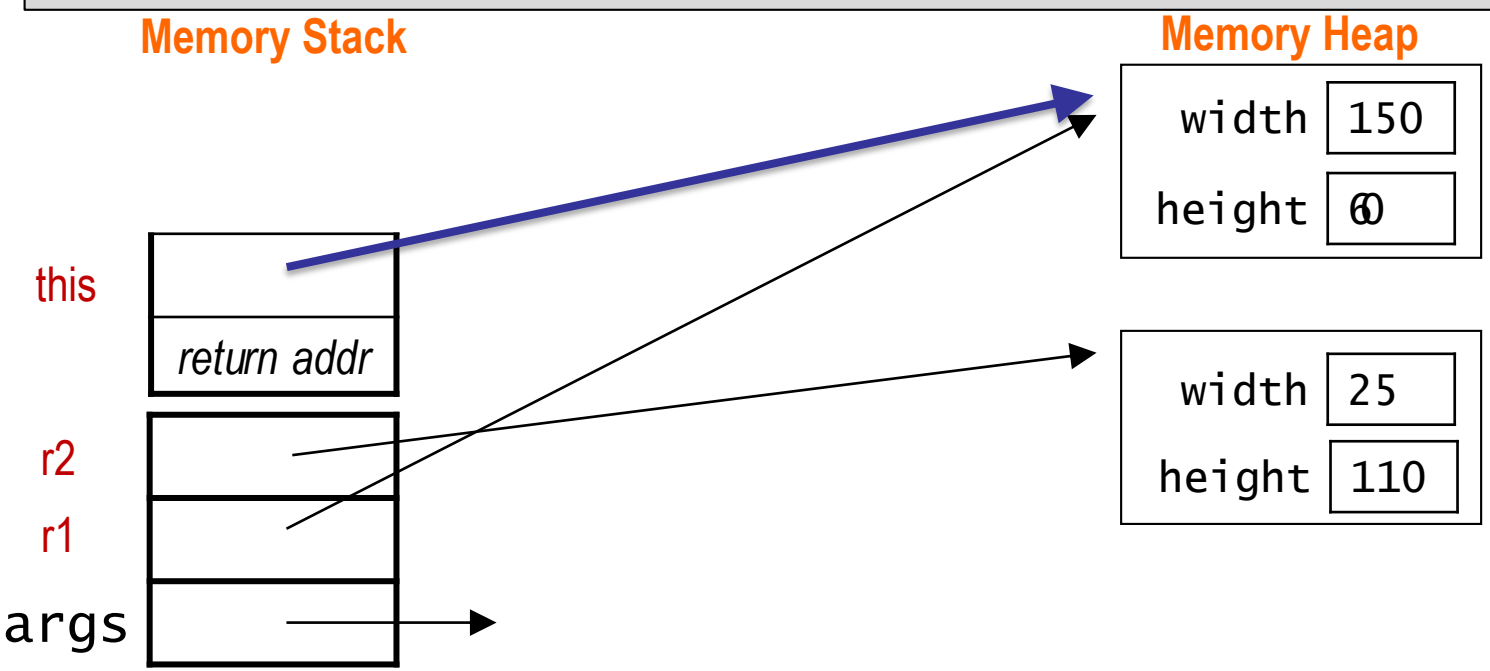
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public String toString() {
        return width + " x " + height;
    }
  }
}
```



A sample Client Program:

memory trace

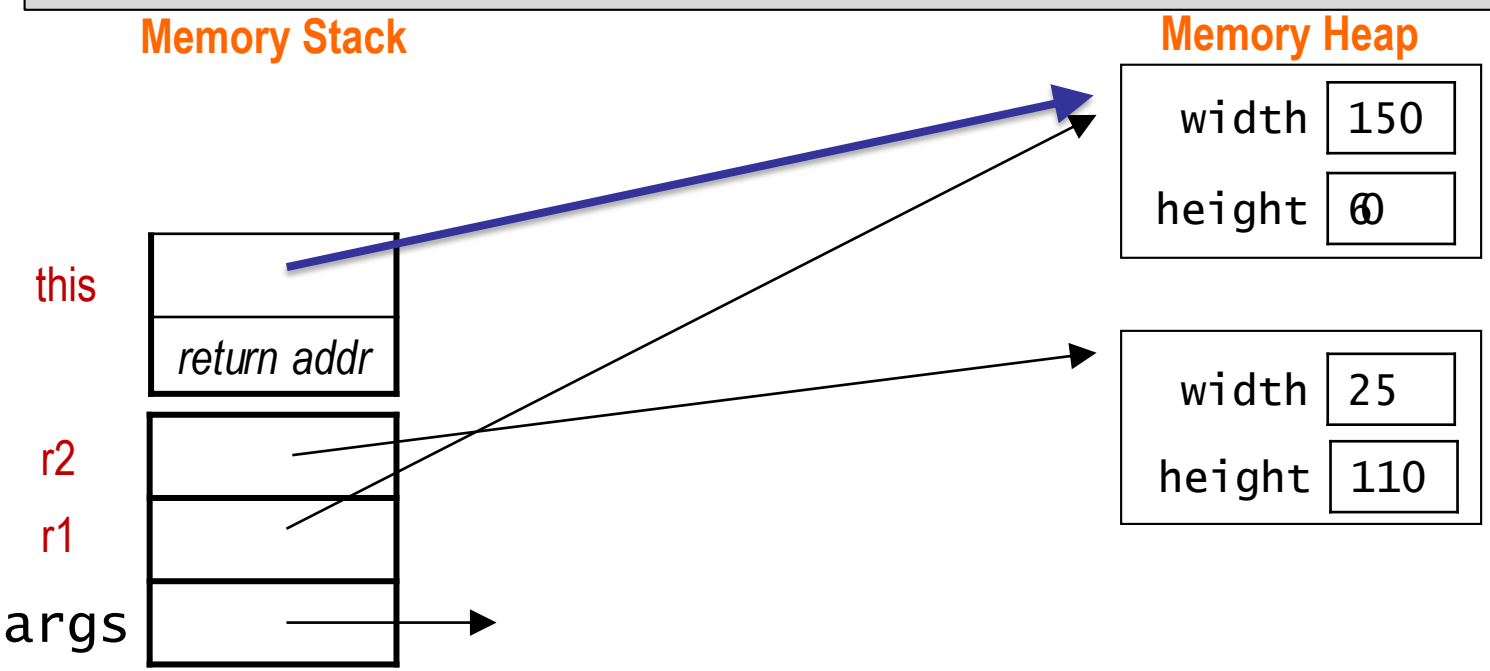
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public String toString() {
        return this.width + " x " + this.height;
    }
  }
}
```



A sample Client Program:

memory trace

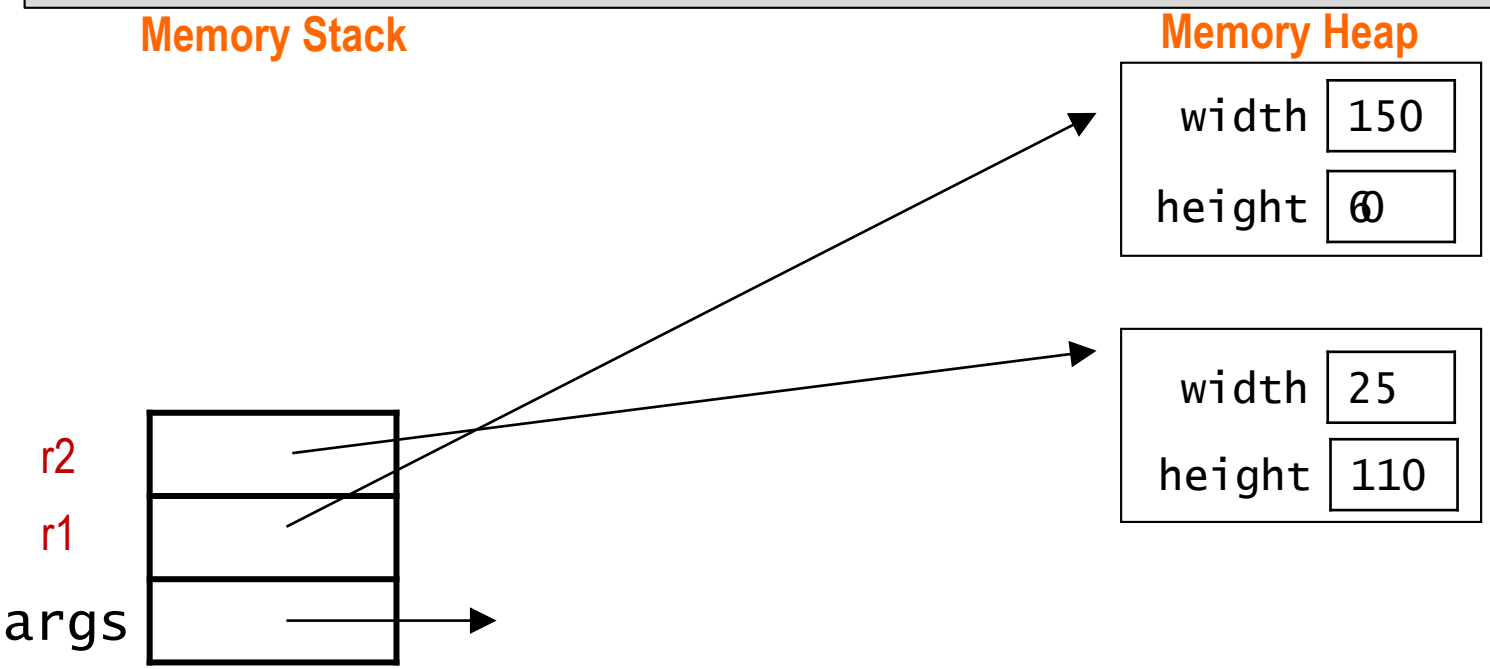
```
public class Rectangle {
    private int width;
    private int height;
    .
    .
    public String toString() {
        return width + " x " + height; // returns "150 x 60"
    }
  }
}
```



A sample Client Program:

memory trace

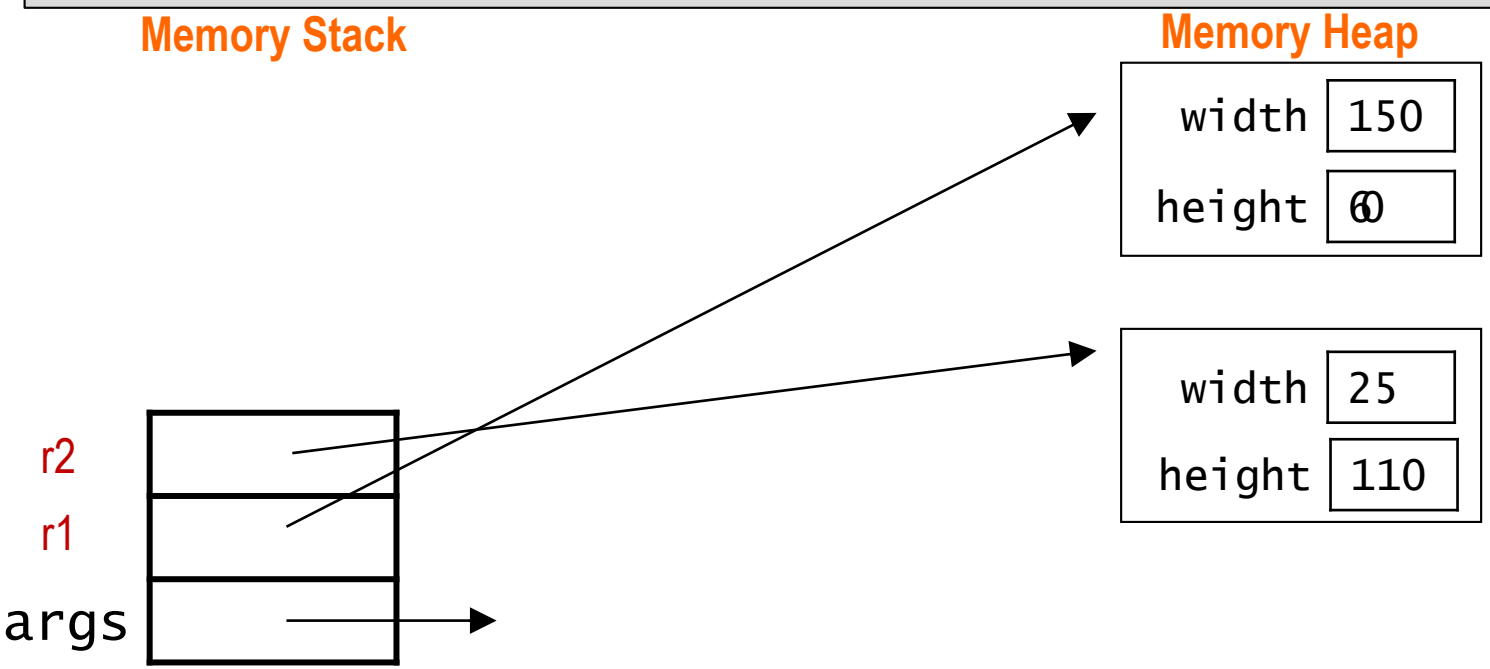
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600
      // grow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



A sample Client Program:

memory trace

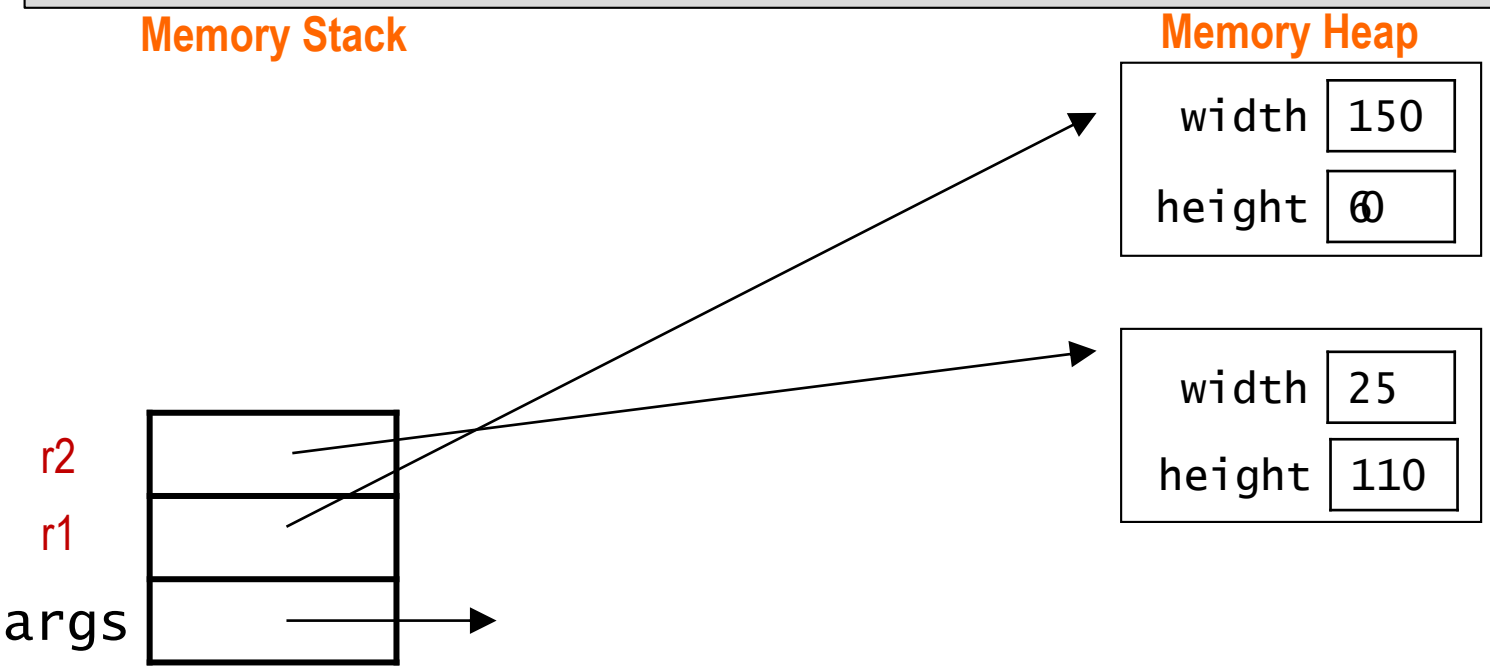
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600
      // grow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



A sample Client Program:

memory trace

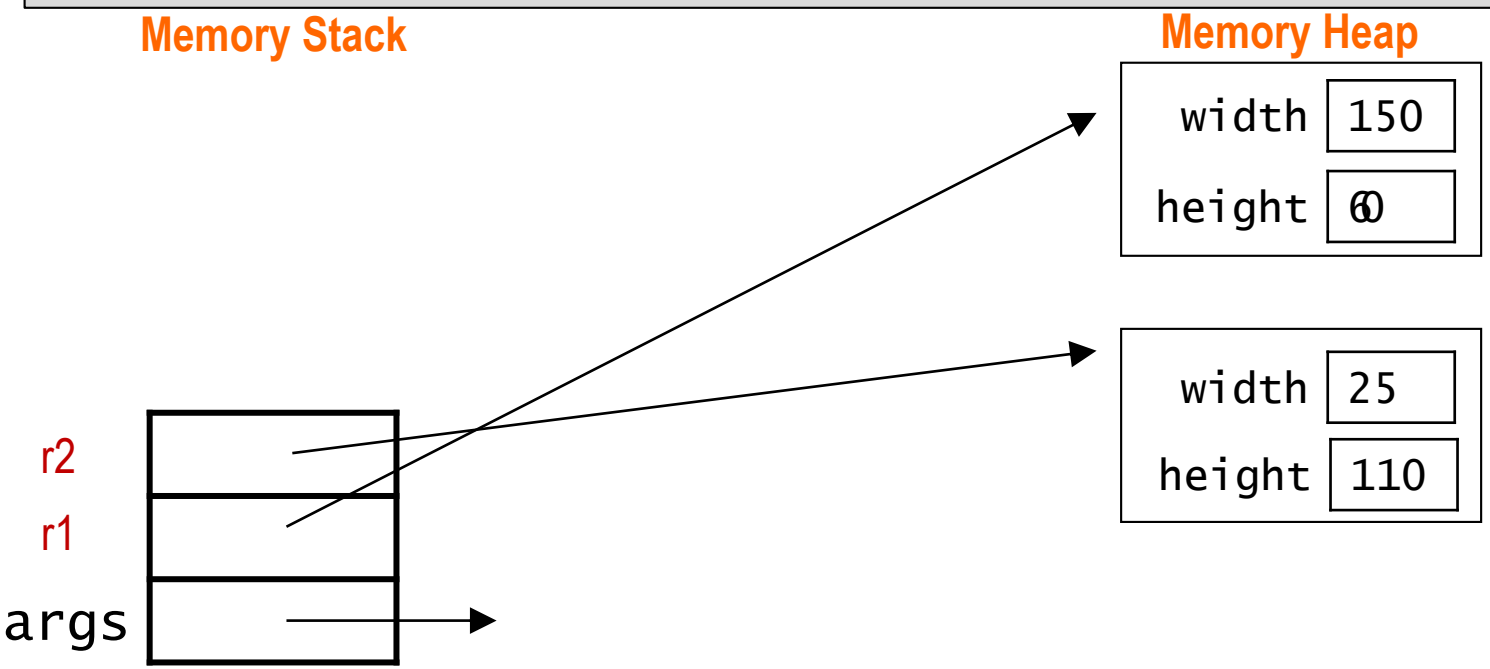
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600
      // grow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



A sample Client Program:

memory trace

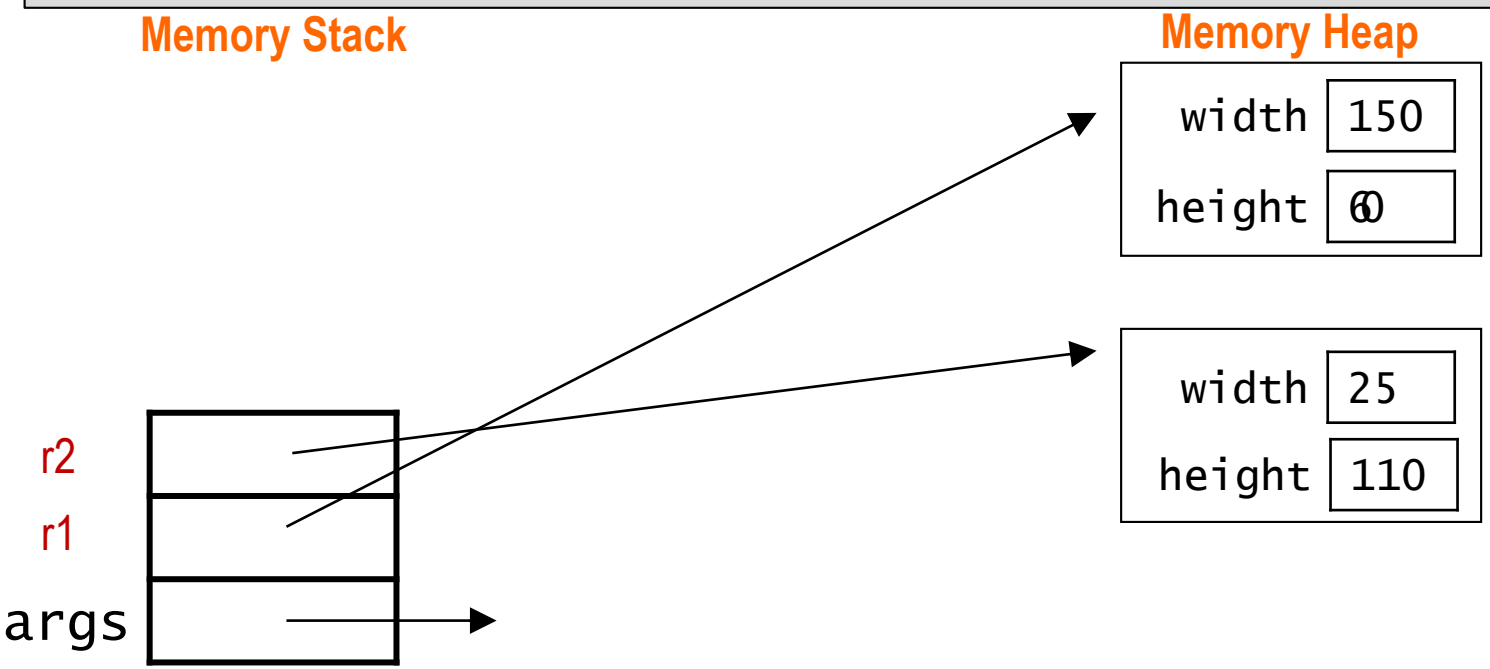
```
public class RectangleClient {
   public static void main(String[] args) {
      Rectangle r1 = new Rectangle(100, 50);
      Rectangle r2 = new Rectangle(20, 80);
      System.out.println("r1's area = " + r1.area() );
      System.out.println("r2's area = " + 1600 ); // outputs 1600
      // grow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



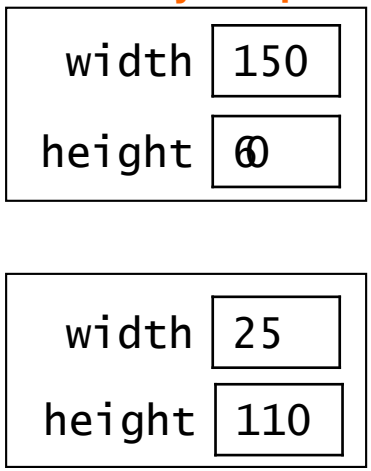
A sample Client Program:

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
      // grow both rectangles
      r1.grow(50, 10);
      r2.grow(5, 30);
      System.out.println("r1: " + r1);
      System.out.println("r2: " + r2);
   }  // end of 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
      // grow both rectangles
      r1.grow(50, 10);
      r2.grow(5, 30);
      System.out.println("r1: " + r1);
      System.out.println("r2: " + r2);
   }  // end of method
}
```

Memory Stack

Memory Heap

Classes as **Custom** Data Type:

a summary

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

      // primitive variables  // object equivalents
      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) {

      // primitive variables  // object equivalents
      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
```

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();
    }

}
```

Class Definition:

*the need for **static** data members*

```
public class Date {

    private int month;
    private int day;
    private int year;

    public String getMonth() {
       String monthOfYear; // variable for month as string
       if ( month == 1 )
            monthOfYear = "January";
       else
       if ( monthOfYear == 2 )
            monthOfYear = "February";
       else
       ...

       return( monthOfYear );
    }
}
```

Class Definition:

*the need for **static** data members*

```
public class Date {

    private int month;
    private int day;
    private int year;

    public String getMonth() {
       String[] months = { "January", "February" ... };

       return( months[month-1] );





    }
}
```

Class Definition:

*the need for **static** data members*

```
public class Date {

    private int month;
    private int day;
    private int year;

    public String getMonth() {
       String[] months = { "January", "February" ... };

       return( months[month-1] );
    }

}
```

There may be other methods of the `Date` class that would need access to this array!

Class Definition:

*the need for **static** data members*

```
public class Date {

    private int month;
    private int day;
    private int year;

    public String formatDate() {



        return( ???? );   // “February 10, 2020"
    }

    public String getMonth() {
        String[] months = { "January", "February" ... };
        return( months[month-1] );
    }
}
```

Class Definition:

*the need for **static** data members*

```
public class Date {

    private static final String[] months =
       {"January", "February", ..., "November", "December"};

    private int month;
    private int day;
    private int year;

    public String formatDate() {
        String formattedDate =
          months[month-1] + " " + day + "," + " " + year;

         return( formattedDate );
    }

    public String getMonth() {

        return( months[month-1] );
    }
}
```

Sources

Thanks to contributing slides from:

- David Sullivan, PhD