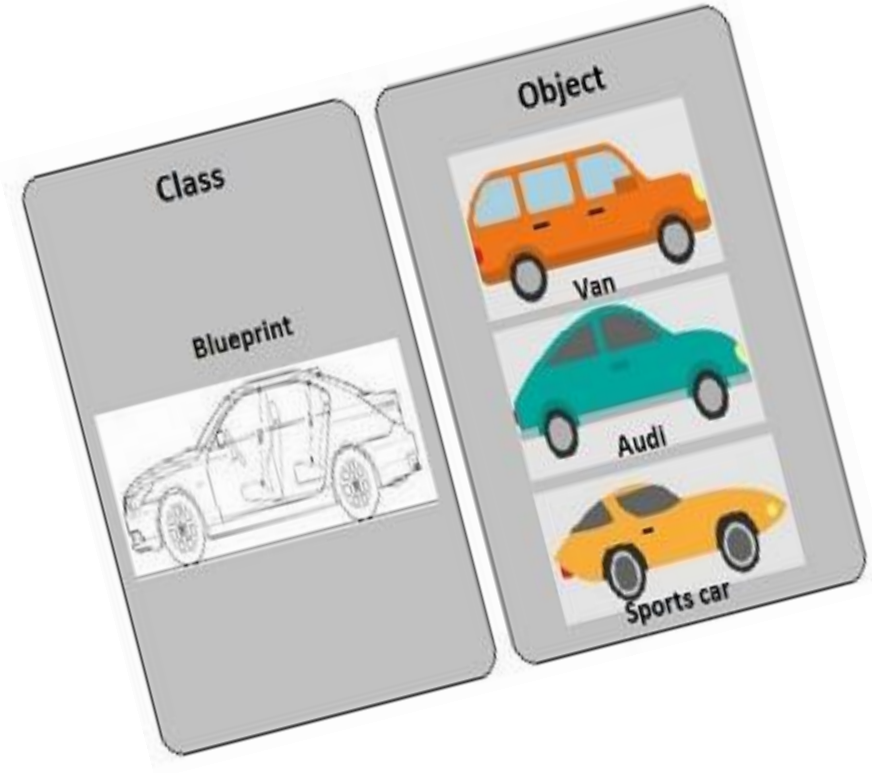
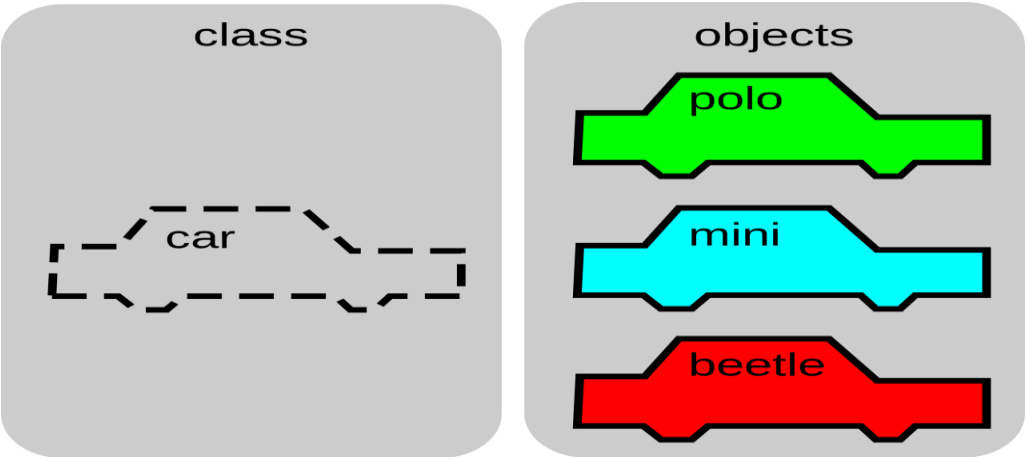


Writing our own classes to build *custom* data types

Computer Science OOD
Boston University

Christine Papadakis-Kanaris

Why classes?



Classes allow us to specify a blueprint that can be used to create a *physical structure of data* that *models* the logical entity it is representing!

Each physical structure (or object) created is an ***instance*** of the class!

Why methods?

The purpose of adding
functions or *methods*
to a class is to provide a class
with all the functionality that its'
objects need to perform.

Why classes?

There are two types of methods
we can define for a class:

Static

non Static

Why classes?

There are two types of methods
we can define for a class:

A **static** method can be called on the class, but **NOT** on an instance of the class.

```
className.method()
```

non Static

Why classes?

There are two types of methods
we can define for a class:

A **static** method can be called on the class, but **NOT** on an instance of the class.

```
Math.pow(8, 3)
```

non Static

Why classes?

There are two types of methods
we can define for a class:

A **static** method can be called on the class, but **NOT** on an instance of the class.

```
className.method(object)
```

If the static method needs access to a specific object, then the object must be explicitly passed to the method as an input parameter.

non Static

Why classes?

There are two types of methods
we can define for a class:

A **static** method can be called on the class, but **NOT** on an instance of the class.

```
int[] arr = {1,2,3};
Arrays.toString(arr)
```

If the static method needs access to a specific object, then the object must be explicitly passed to the method as an input parameter.

non Static

Why classes?

There are two types of methods
we can define for a class:

A static method can be called on the class, but NOT on an instance of the class.

```
className.method(object)
```

If the static method needs access to a specific object, then the object must be explicitly passed to the method as an input parameter.

A ***non* static** method must be called on an *instance* of the class.

```
object.method()
```

As the method is being called on an object, the method has direct access to all the data and methods of the class.

Why classes?

There are two types of methods
we can define for a class:

A static method can be called on the class, but NOT on an instance of the class.

```
className.method(object)
```

If the static method needs access to a specific object, then the object must be explicitly passed to the method as an input parameter.

A ***non* static** method must be called on an *instance* of the class.

```
String str = "Hello!";
int len = str.length()
```

As the method is being called on an object, the method has direct access to all the data and methods of the class.

Why classes?

There are two types of methods
we can define for a class:

```
className.method(object)
```

```
object.method()
```

Designing a Custom Class

- What's in a Name?

- Example:
  - Ms. Gremelda Lyons
  - Mr. John Reynolds III
  - Dr. Michael Carepenter

- **First Name** // String
- **Last Name** // String
- **Middle Initial** // String ***or char***
- **Prefix** // String
- **Suffix** // String
- **Nick Name** // String

Attributes of a *name*

Designing a Custom Class

- What's in a Date?

- Example:
  - 02/25/1962
  - 03/02/1996
  - 10/04/1999

- **month** // int
- **day** // int
- **year** // int

Attributes of a *date*

Class Definition:

a *user defined* custom datatype

```
public class Date {

    int month;
    int day;
    int year;

}
```

Attributes
of the class

Class Definition:

a *user defined* custom datatype

```
public class Date {

    int month;
    int day;
    int year;

    // determine if date is a holiday

    // calculate an age

    // determine number of days remaining

    // print the date as m/d/y or as . . .

    // determine if two days are the same

}
```

Attributes of the class

Behaviors of the class

Functions that each object can perform!

Class Definition:

a *user defined* custom datatype

```
public class Date {

    int month;
    int day;
    int year;

    boolean isHoliday() { ... }

    int calculateAge() { ... }

    int daysUntil( Date someDate ) { ... }

    String formatDate() { ... }

    boolean equals( Date someDate ) { ... }

}
```

Attributes of the class

Behaviors of the class

Functions that each object can perform!

Class Definition:

a *user defined* custom datatype

```
public class Date {

    int month;
    int day;
    int year;

    boolean isHoliday() { ... }

    int calculateAge() { ... }

    int daysUntil( Date someDate ) { ... }

    String formatDate() { ... }

    boolean equals( Date someDate ) { ... }

}
```

Who should have access to the *data attributes* and who should have access to the *methods*?

Behaviors of the class

Functions that each object can perform!

Class Definition:

a *user defined* custom datatype

```
public class Date {

    int month;
    int day;
    int year;

    boolean isHoliday() { ... }

    ...
}
```

```
public class testDate {

    public static void main( String[] args ) {
        Date bday = new Date();



    }
}
```

Class Definition:

a *user defined* custom datatype

```
public class Date {

    int month;
    int day;
    int year;

    boolean isHoliday() { ... }

    ...
}
```

```
public class testDate {

    public static void main( String[] args ) {
        Date bday = new Date();
        bday.month = 12;
        bday.day = 25;
        bday.year = 1962;
    }
}
```

Class Definition:

a *user defined* custom datatype

```
public class Date {

    int month;
    int day;
    int year;

    boolean isHoliday() { ... }

    ...
}
```

```
public class testDate {

    public static void main( String[] args ) {
        Date bday = new Date();
        bday.month = 13;        // Invalid Data
        bday.day = 25;
        bday.year = 1962;
    }
}
```

Class Definition:

a *user defined* custom datatype

```
public class Date {

    int month;
    int day;
    int year;

    boolean isHoliday() {

    ...
}
```

How can we ensure that *applications* or *clients* who create instances of our class use them correctly?

```
public class testDate {

    public static void main( String[] args ) {
        Date bday = new Date();
        bday.month = 13;        // Invalid Data
        bday.day = 25;
        bday.year = 1962;
    }
}
```

Class Definition:

a *user defined* custom datatype

```
public class Date {

    int month;
    int day;
    int year;

    boolean isHoliday() { ... }

    ...
}
```

```
public class testDate {

    public static void mai
        Date bday = new Da
        bday.month = 13;
        bday.day = 25;
        bday.year = 1962;
    }
}
```

Have the class determine exactly what information client programs can access!

Class Definition:

a *user defined* custom datatype

```
public class Date {

    int month;
    int day;
    int year;

    boolean isHoliday() { ... }

    int calculateAge() { ... }

    int daysUntil( Date someDate ) { ... }

    String formatDate() { ... }

    boolean equals( Date someDate ) { ... }

}
```

Attributes of the class

Behaviors of the class

Functions that each object can perform!

Class Definition:

a *user defined* custom datatype

```
public class Date {

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

    boolean isHoliday() { ... }

    int calculateAge() { ... }

    int daysUntil( Date someDate ) { ... }

    String formatDate() { ... }

    boolean equals( Date someDate ) { ... }

}
```

Attributes of the class

Behaviors of the class

Functions that each object can perform!

Class Definition:

a *user defined* custom datatype

```
public class Date {

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

    boolean isHoliday() { ... }

    int calculateAge() { ... }

    int daysUntil( Date someDate ) { ... }

    String formatDate() { ... }

    boolean equals( Date someDate ) { ... }

}
```

Attributes of the class

Behaviors of the class

Functions that each object can perform!

Class Definition:

a *user defined* custom datatype

```
public class Date {

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

    public boolean isHoliday() { ... }

    public int calculateAge() { ... }

    public int daysUntil( Date someDate ) { ... }

    public String formatDate() { ... }

    public boolean equals( Date someDate ) { .. }

}
```

Attributes of the class

Behaviors of the class

Functions that each object can perform!

Class Definition:

a *user defined* custom datatype

```
public class Date {

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

    public boolean isHoliday() { ... }

    public int calculateAge() { ... }

    public int daysUntil( Date someDate ) { ... }

    public String formatDate() { ... }

    public boolean equals( Date someDate ) { .. }

}
```

The class defines the public interface!

Behaviors of the class

Functions that each object can perform!

Class Definition:

a *user defined* custom datatype

```
public class Date {

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

    public boolean isHoliday() { ... }

    ...
}
```

```
public class testDate {

    public static void main( String[] args ) {
        Date bday = new Date();
        bday.month = 12;
        bday.day = 25;
        bday.year = 1962;
    }
}
```

Class Definition:

a *user defined* custom datatype

```
public class Date {

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

    public boolean isHoliday() { ... }

    ...
}
```

```
public class testDate {

    public static void main( String[] args ) {
        Date bday = new Date();
        bday.month = 12;
        bday.day = 25;
        bday.year = 1962;
    }
}
```

Class Definition:

a *user defined* custom datatype

```
public class Date {

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

    public boolean isHoliday() { ... }

    ...
}
```

```
public class testDate {

    public static void main( String[] args ) {
        Date bday = new Date();
        // can only access public items
        // need to assign the date by calling
        // a public method of the class!
    }
}
```

Class Definition:

a *user defined* custom datatype

```
public class Date {

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

    public boolean i   liday() {

    ...
}
```

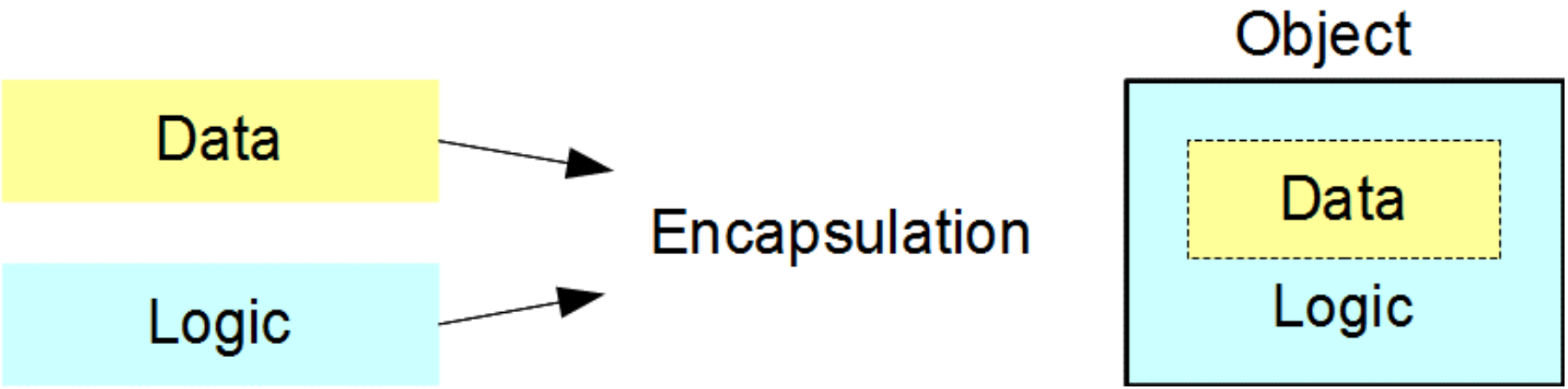
```
public class testD

    public static v
        Date bday = n
        // can only acces
        // need to assign the date by   ng
        // a public method of the class!
    }
}
```

Making the data attributes private ***encapsulates*** *the data within the object and we can prevent the data from being accessed by any external code.*

Encapsulation:

first principle of OO Design



Encapsulation

- *Encapsulation* is one of the key principles of object-oriented programming.
  - another name for it is *information hiding*
- It refers to the practice of “hiding” the implementation of a class from users of the class.
  - prevent *direct* access to the internals of an object
    - making the fields private
  - provide *controlled*, *indirect* access through a set of methods
    - creating the public interface
- In addition to preventing inappropriate changes, encapsulation allows us to change the implementation of a class without breaking the client code that uses it.

Encapsulation

- *Encapsulation* is one of the key principles of object-oriented programming.
  - another name for it is *information hiding*
- It refers to the practice of “hiding” the implementation of a class from users of the class.
  - prevent *direct* access to the internals of an object
    - making the fields private
  - provide *controlled*, *indirect* access through a set of methods
    - creating a ***public interface***
- In addition to preventing inappropriate changes, encapsulation allows us to change the implementation of a class without breaking the client code that uses it.

Encapsulation

- *Encapsulation* is one of the key principles of object-oriented programming.
  - another name for it is *information hiding*

- It refers to the practice of “hiding” the implementation of a class from users of the class.
  - prevent *direct* access to the internals of an object
    - making the fields private
  - provide *controlled*, *indirect* access through a set of methods
    - creating a public interface

- In addition to preventing inappropriate changes, encapsulation allows us to change the implementation of a class without breaking the client code that uses it.

Access Modifiers

- `public` and `private` are known as *access modifiers*.
  - they specify where a class, field, or method can be used

- A class is usually declared to be `public`:

```
public class Rectangle {
```

  - indicates that objects of the class can be used anywhere, including in other classes

- Fields (*members*) are usually declared to be `private`.

- Methods are usually declared to be `public`.

- Often times we need to define *`private`* methods:
  - serve as *helper methods* for the `public` methods
  - can only be invoked by other methods of the class
  - **cannot** be invoked by code that is outside the class

Class Definition:

another look

```
public class className {

    static variables    // class scope

    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
}
```

Attributes of the class

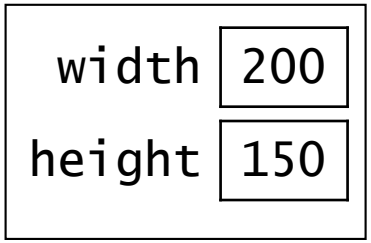
Behaviors of the class

Functions that each object can perform!

Example: 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;
    }
    .
    .
    .
    .
}
```

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 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;
    }

    .
    .
    .
    .

}
```

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 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;
    }

    .
    .
    .
    .

}
```

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 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;
    }

    .
    .
    .
    .

}
```

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;
    }


    .
    .
    .
    .



}
```

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);
    }


}
```

r1

r2

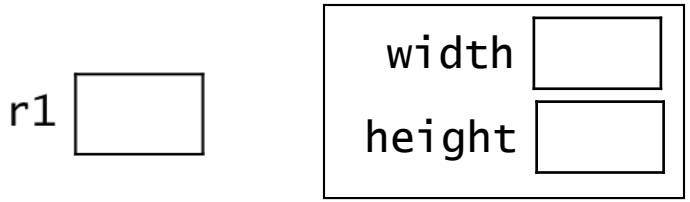
Sample Rectangle Class

```
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);
    }

}
```



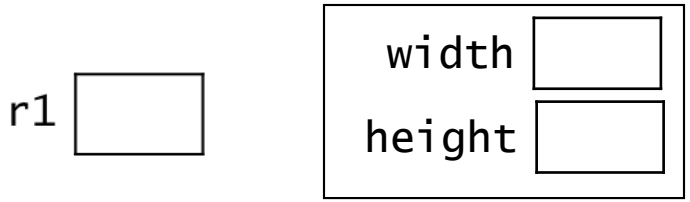
Sample Rectangle Class

```
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);
    }

}
```



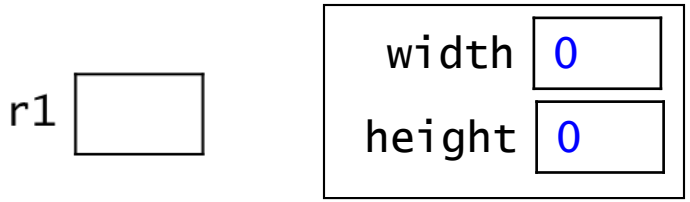
Sample Rectangle Class

```
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);
    }

}
```



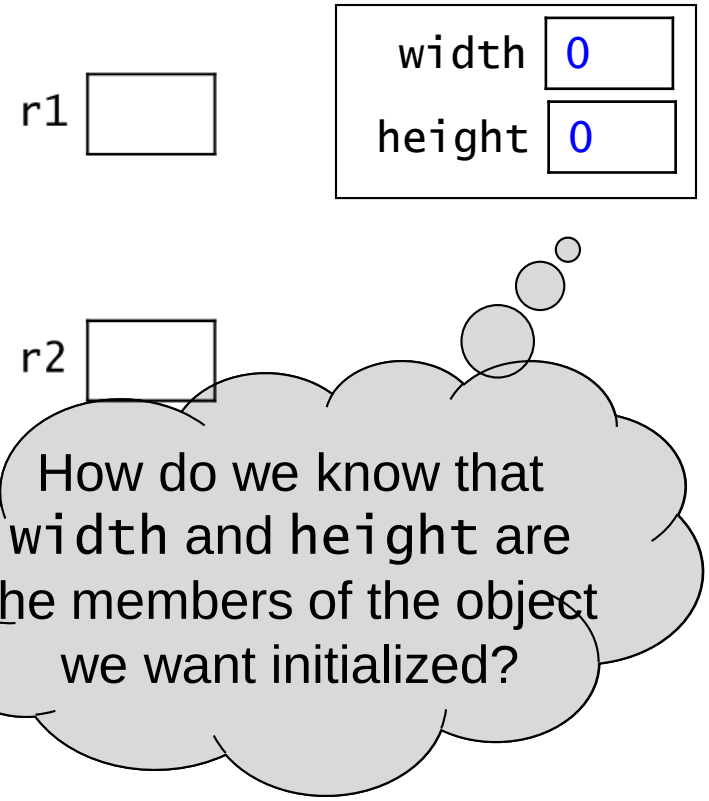
Sample Rectangle Class

```
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);
    }

}
```



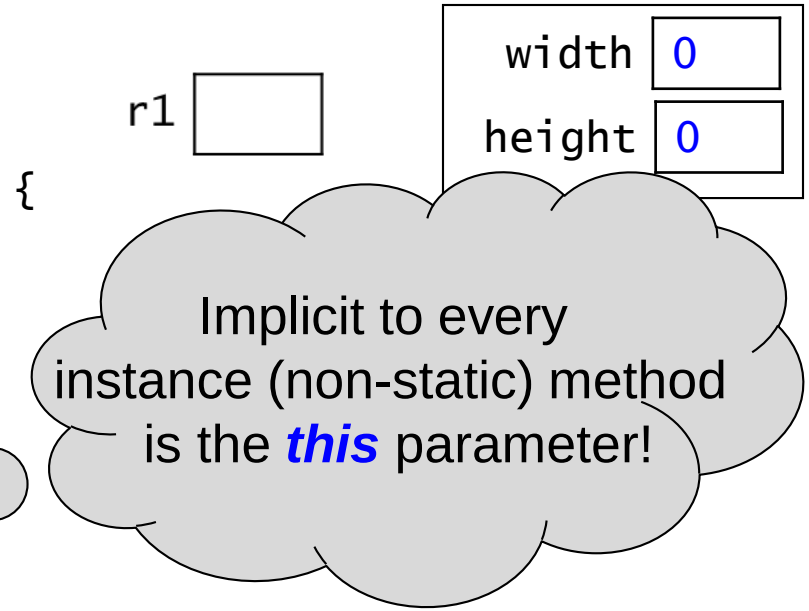
Sample Rectangle Class

```
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);
    }

}
```



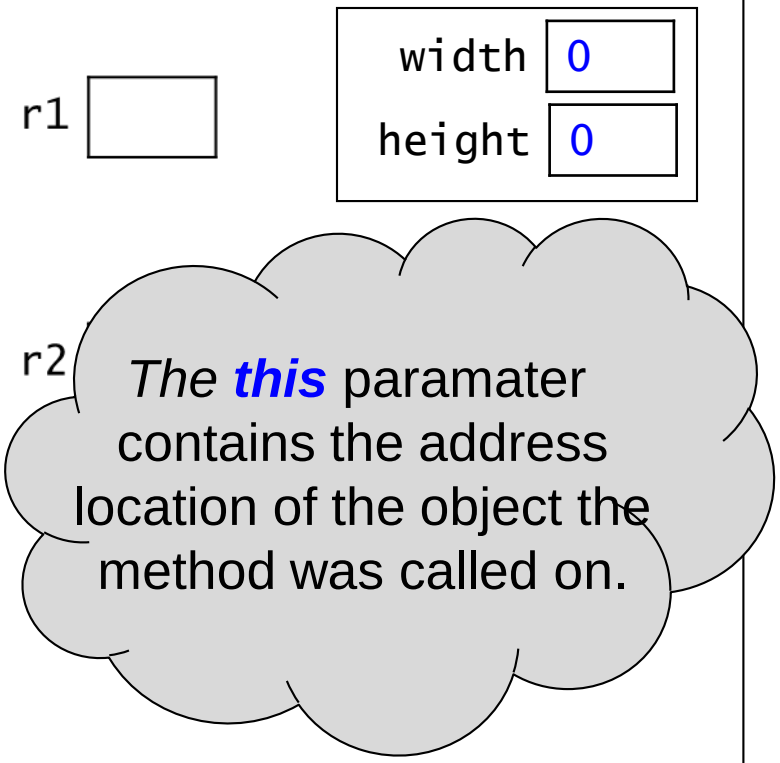
Sample Rectangle Class

```
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);
    }

}
```



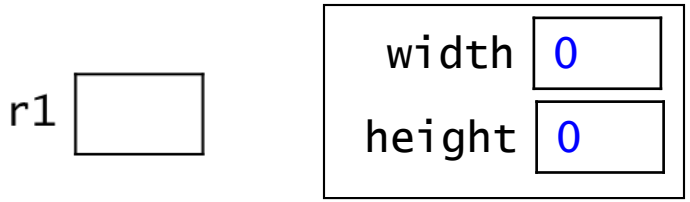
Sample Rectangle Class

```
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);
    }

}
```

Note that this call is part of an assignment statement.

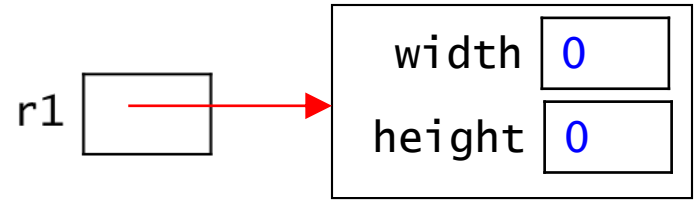
Sample Rectangle Class

```
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!

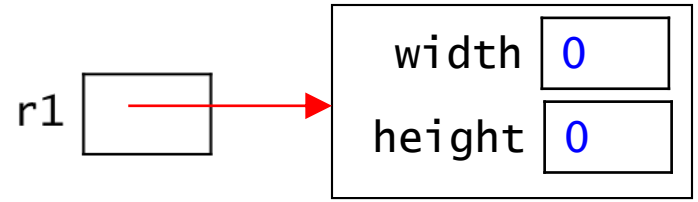
Sample Rectangle Class

```
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!

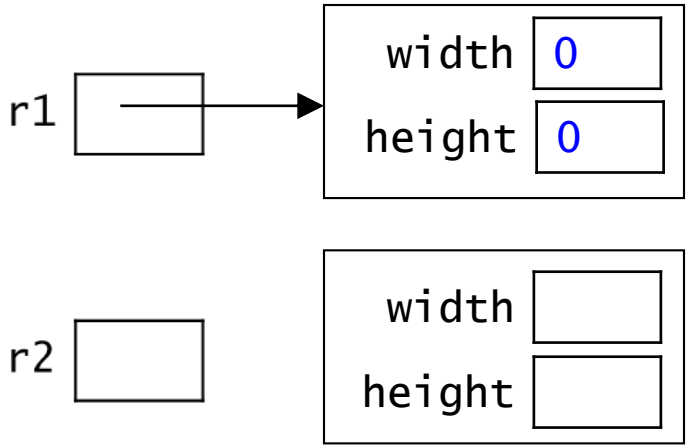
Sample Rectangle Class

```
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);
    }


}
```



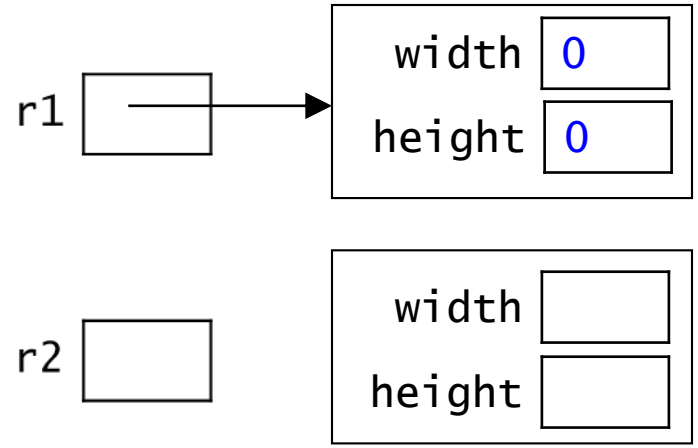
Sample Rectangle Class

```
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);
    }

}
```



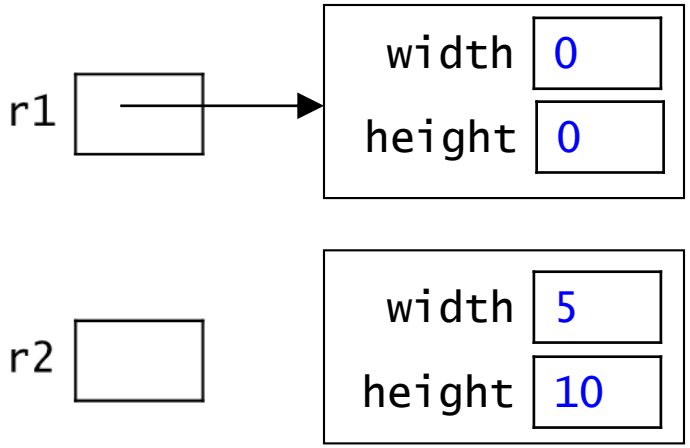
Sample Rectangle Class

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

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


}
```



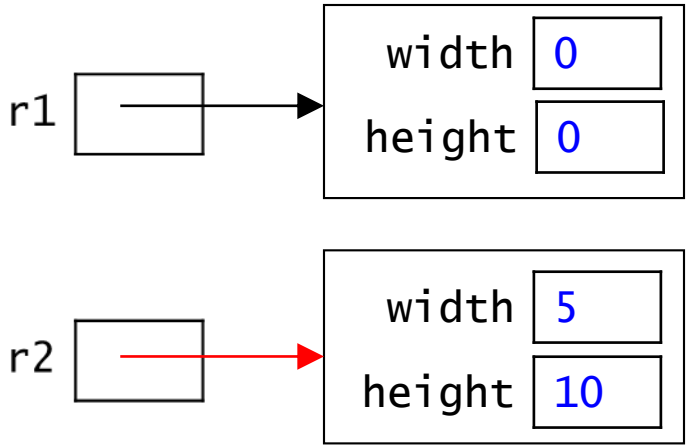
Sample Rectangle Class

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

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


}
```



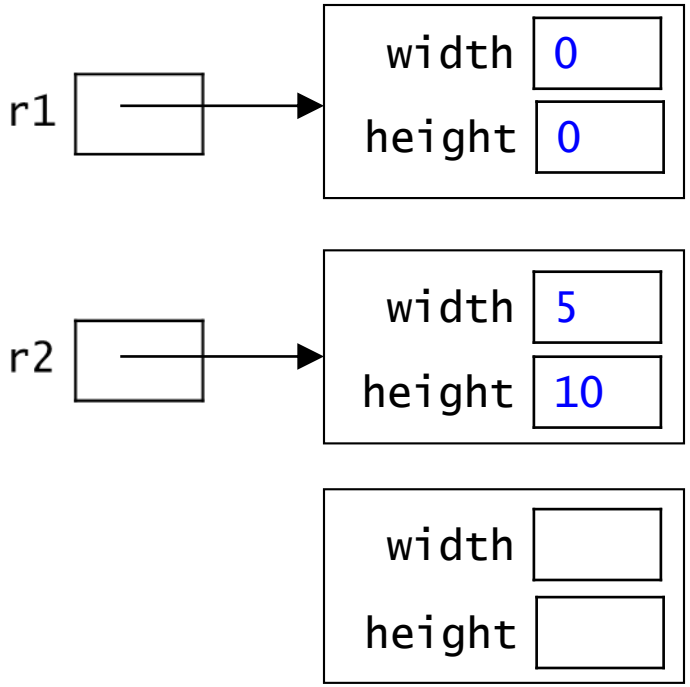
Sample Rectangle Class

```
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);
    }

}
```



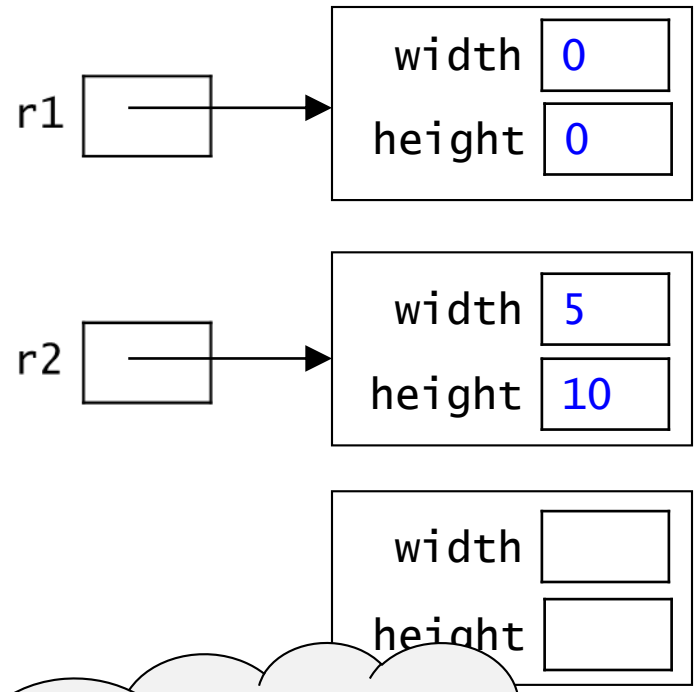
Sample Rectangle Class

```
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,
        Rectangle r3 = new Rectangle(7);
    }

}
```



Note that both constructors are doing the same thing.

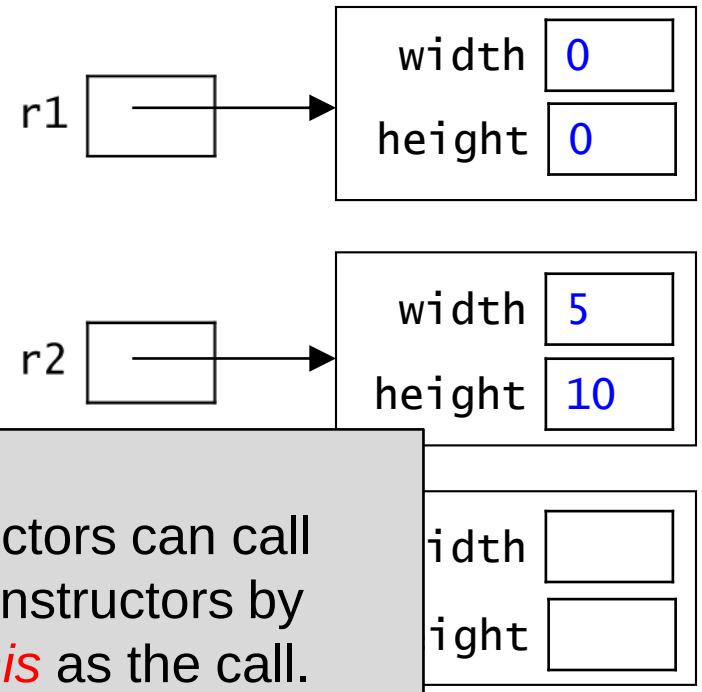
Sample Rectangle Class

```
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.

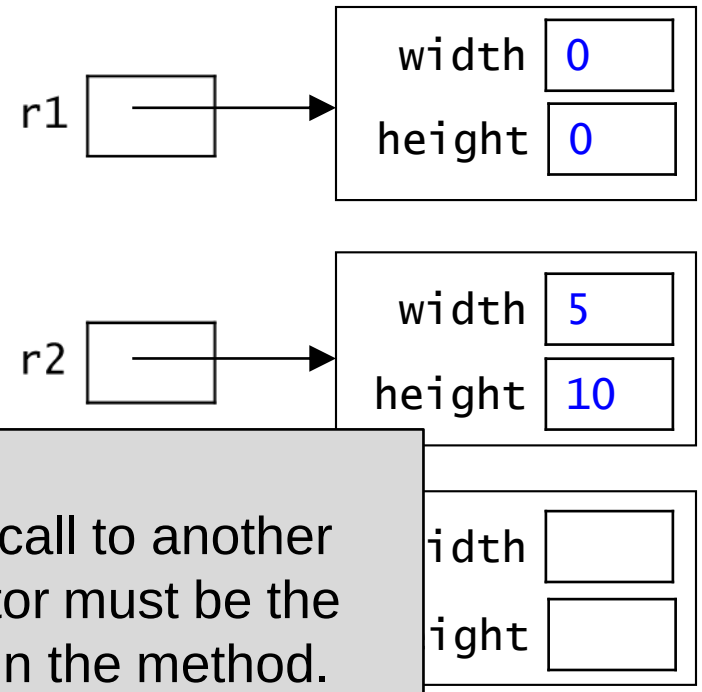
Sample Rectangle Class

```
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.

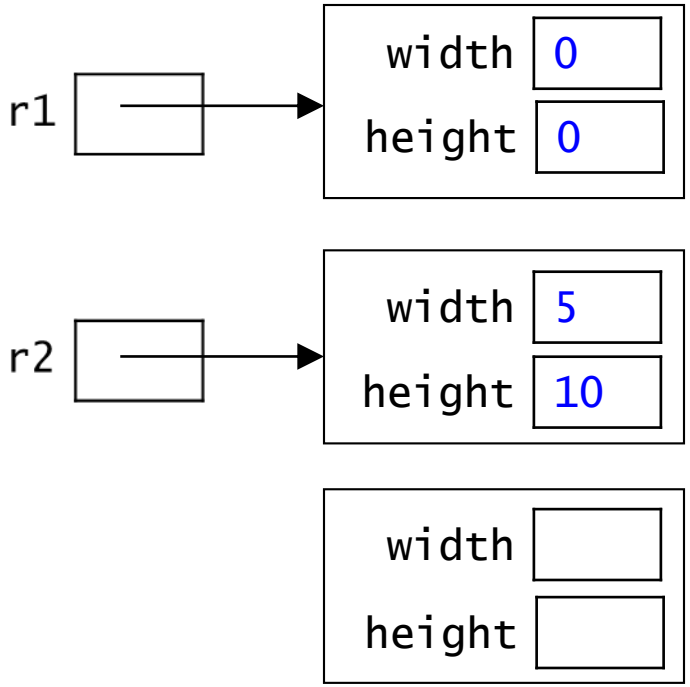
Sample Rectangle Class

```
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( String [] args ) {

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

}
```



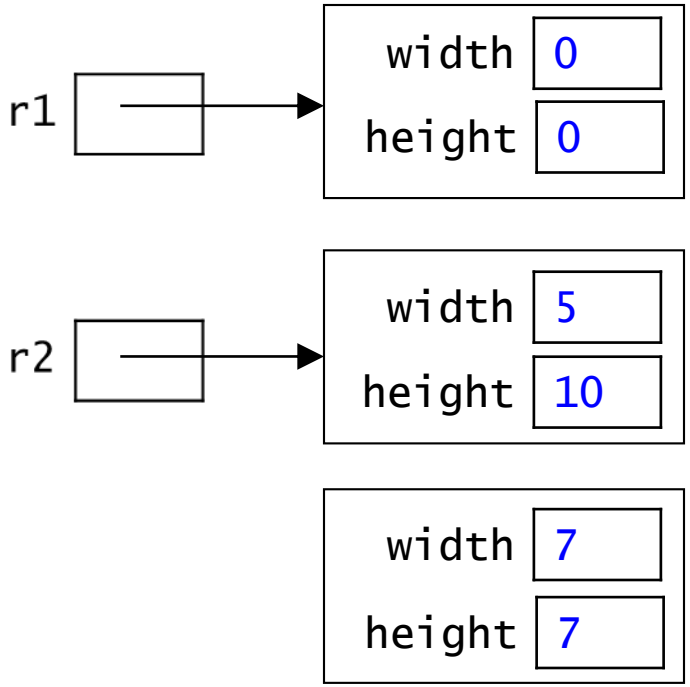
Sample Rectangle Class

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

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

}
```



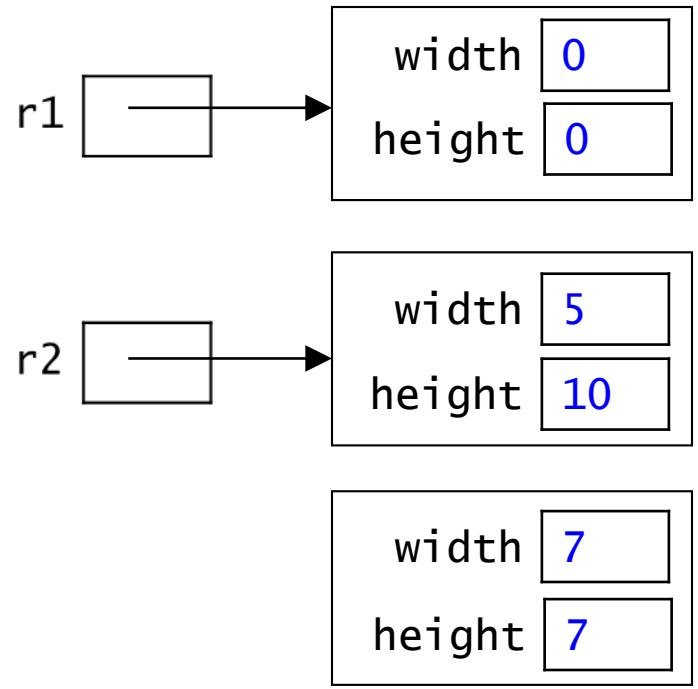
Sample Rectangle Class

```
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 mair

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

}
```



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

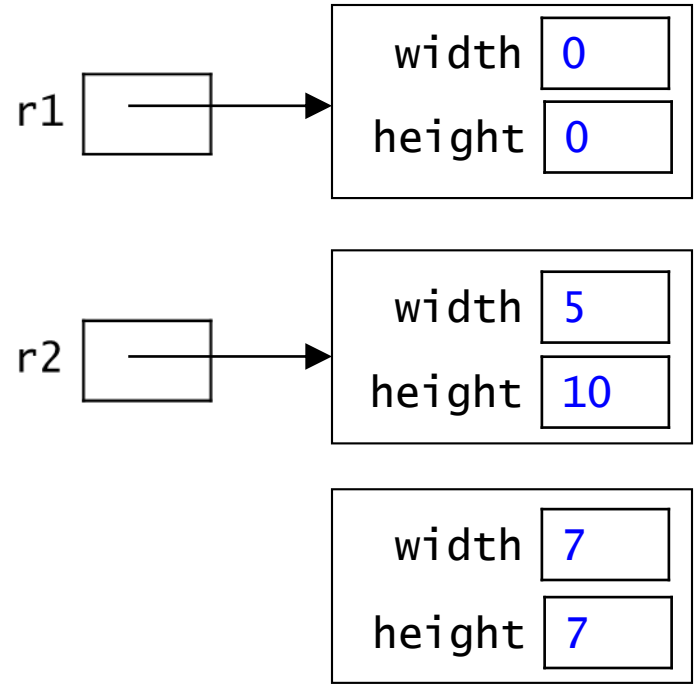
Sample Rectangle Class

```
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 Rectangle(
    }

}
```



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

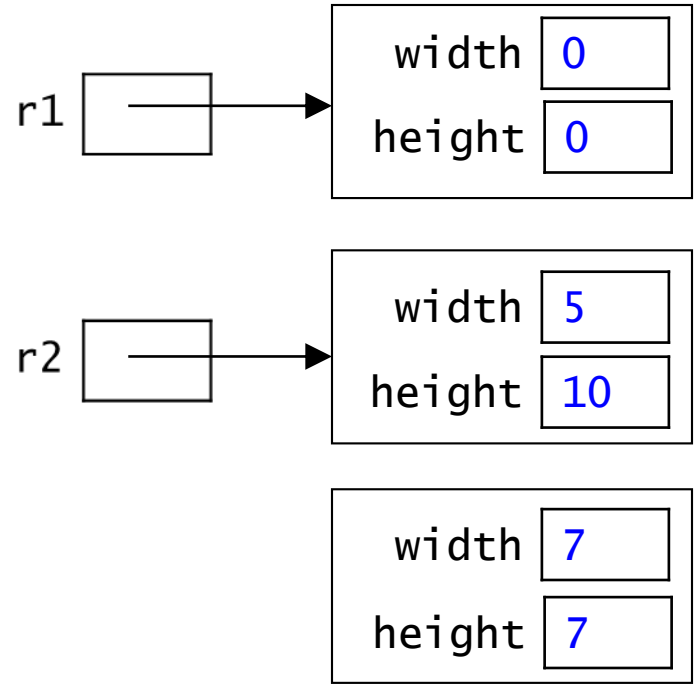
Sample Rectangle Class

```
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 Rectangle(
    }

}
```



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 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.