- Previous Lecture:
  - Intro to objects and classes
  - Creating objects and calling their methods
- Today's Lecture:
  - OO thinking
  - Defining a class:
    - Instance variables
    - Instance methods
    - Constructors
- Reading: Sec 4.1, 4.2

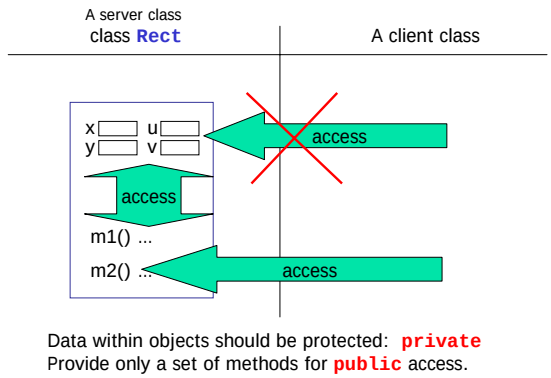
Multiple references can point to the same object

```
JFrame f1= new JFrame();
JFrame f3= f1;
f1.setTitle("X");
System.out.println(f3.getTitle());
```

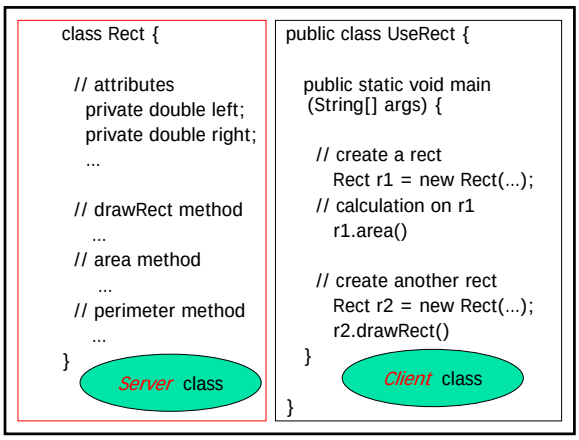
f3==f1 gives

OOP ideas

- Aggregate variables/methods into an abstraction (a class) that makes their relationship to one another explicit
- Objects (instances of a class) are self-governing (protect and manage themselves)
- Hide details from client, and restrict client's use of the services
- Allow clients to create/get as many objects as they want



Data within objects should be protected: **`private`**
Provide only a set of methods for **`public`** access.



- We have used different classes already:
  - **`System`**, **`Math`**
  - **`Keyboard`**, **`JFrame`**
- Above classes provide various *services* (related services are grouped in same class)
- Implementation details of the class are hidden from the *client* (user)

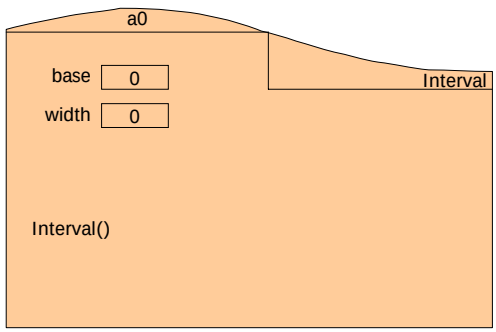
Class Definition

```
public class class-name {

   declaration (and initialization)

   constructor

   methods

}
```


Class definition: declarations

```
class Interval {
  private double base;  // low end
  private double width; // interval width
}
```

- *Declarations* in a class define ***fields*** (***instance variables***) of the class
- Each class is a *type*. Classes are *not* primitive types.



Declarations Revisited

- Syntax: *type name*;
- Examples:
```
int count;
Interval in1;
Interval in2;
```

- Instance variables have default initial values
  - `int` variables: `0`
  - Non-primitive (reference) variables: `null`
    Value `null` signifies that no object is referenced

Object instantiation

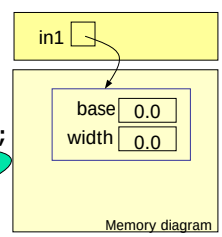
- An expression of the form
  `new` *class-name*`( )`
  computes a reference to a newly created object of the given class
- Examples:

```
Interval in1;           //declaration
in1 = new Interval();   //instantiation

//Combined declaration & instantiation
Interval in2 = new Interval();
```


Do not access fields directly

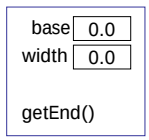
```
public class Client {
  public static void main(String[] args){

    Interval in1;
    in1= new Interval();
    System.out.println(
     in1.base+in1.width);
  }
}
```

base, width are private

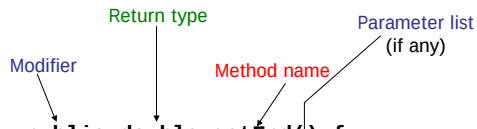


Class definition

```
class Interval {
  private double base;  // low end
  private double width; // interval width

  /* =Get right end of interval */
  public double getEnd() {
    return base + width;
  }

}
```



Instance method




```
public double getEnd() {
  return base + width;
}
```

The absence of the keyword `static` → an instance method
(There isn't a keyword "instance")

Methods

A method is a named, parameterized group of statements

```
modifier return-type method-name ( parameter-list ) {
    statement-list
}
```

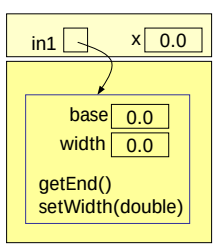
return-type **`void`** means nothing is returned from the method

There must be a **`return`** statement, unless return-type is **`void`**

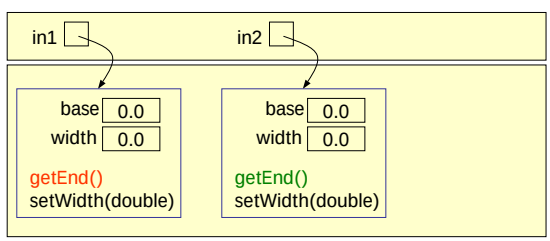

```
/* An Interval is [base, base+width] */
class Interval {
   private double base;  // low end
   private double width; // interval width

   /* =Get right end of interval */
   public double getEnd() {
     return base + width;
   }
   /* set width to 4 */
   public void setWidth(double w) {
     width= w;
   }
}
```


Calling an instance method

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

    Interval in1;
    in1= new Interval();
    double x;
    x= in1.getEnd();
    in1.setWidth(4);
  }
}
```




```
Interval in1= new Interval();
Interval in2= new Interval();
if ( in1.getEnd() > in2.getEnd() )
  System.out.println("blah...");
```



Constructor

- A *constructor* is used to create objects
- Each class has a default constructor
- You can define your own constructor:

modifier class-name **(** *parameter-list* **) {**
statements-list
}

- Use **public** as the modifier for now
- an instance method that has *no* return type


```
class Interval {
  private double base;  // low end
  private double width; // interval width

  /* An Interval with base b, width w */
  public Interval(double b, double w) {
    base= b;
    width= w;
  }

  public double getEnd() {
    return base + width;
  }
}
```



```
class Interval {
  private double base;  // low end
  private double width; // interval width

  /* Default constructor */
  public Interval() {}



  public double getEnd() {
    return base + width;
  }
}
```

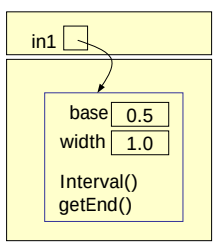

Constructor invocation

new *class-name* **(** *expression-list* **)**

- The value of above expression is a reference to a *new* object of the given *class-name*
- The defined (or default) constructor is invoked on the new object created by **new**

Creating an object

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

    Interval in1;
    in1= new Interval(
                0.5,1);


  }
}
```




```
public Interval(double b, double w) {
    base= b;
    width= w;
}
```