- Previous Lecture:
  - Review methods (functions)
  - Iteration with **`for`** loop
  - Intro to objects and classes
- Today's Lecture:
  - Intro to objects and classes
  - Creating objects and calling their methods
  - OO thinking
- Reading: start reading Sec 4.1
- Announcement: Project 4 due today at 6pm

Primitive vs non-primitive values

```
int x= 2;
int y= 2;
JFrame f1= new JFrame();
JFrame f2= new JFrame();
JFrame f3= f1;
```

alias

x==y gives ______

f1==f2 gives ______

f1==f3 gives ______

Class *definition* vs. object *instantiation*

If you want make a whole lot of cookies, you may want to

- Make a cookie cutter—*define the class*
- Stamp out the cookie—*instantiate an object*

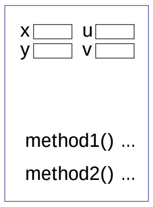
Making a cookie cutter

≠

Getting a cookie

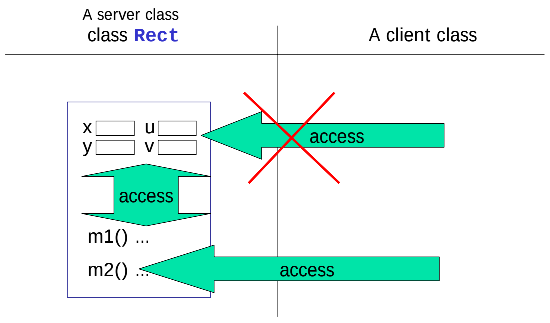
```
class Rect {

  // attributes
    private double left;
    private double right;
    ...

  // drawRect method
    ...
  // area method
    ...
  // perimeter method
    ...
}
```

Object from class **`Rect`**



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.

```
class Rect {

  // attributes
    private double left;
    private double right;
    ...

  // drawRect method
     ...
  // area method
      ...
  // perimeter method
     ...
}
```

Server class

```
public class UseRect {

   public static void main
    (String[] args) {

     // create a rect
        Rect r1 = new Rect(...);
     // calculation on r1
        r1.area()

     // create another rect
        Rect r2 = new Rect(...);
        r2.drawRect()
   }
}
```

Client class

- We have used different classes already:
  - `System`, `Math`, `Scanner`
  - `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

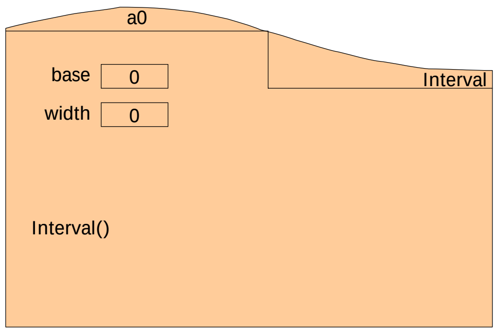
methods

`}`

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