Announcements

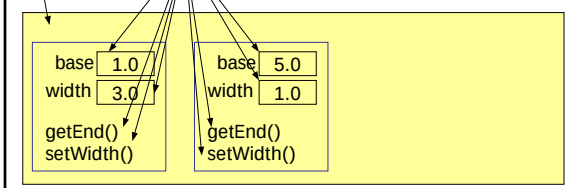
- P5 due Thurs, 4/13, at 6pm
- Prelim 3 on Tues, 4/18. Tell us now if you have an official university conflict
- Review session Sunday 1-2:30

- Previous Lecture:
  - Defining a class:
    - Constructors
    - Keyword this
    - Method toString
- Today's Lecture:
  - Defining a class:
    - Non-primitive type parameters
    - Static variables and methods
    - Method overloading
- Reading: Sec 4.3, Sec 5.1
- Optional reading: Sec 5.2


```
public class Client {
  public static void main(String[] args) {
    Interval i1= new Interval(1,3);
    Interval i2= new Interval(5,1);
  }
}
```

Instance var., methods live inside the object

The world of class `Interval`



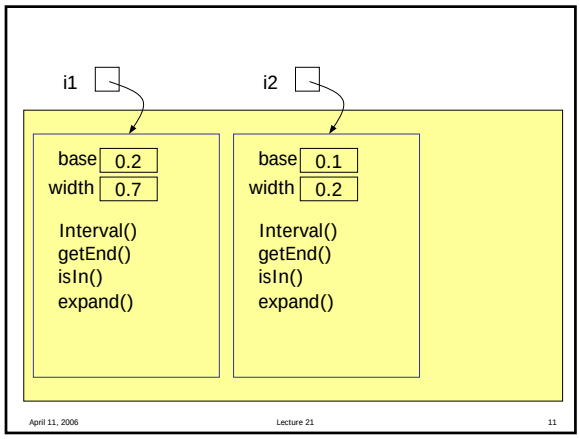
Non-primitive input parameter

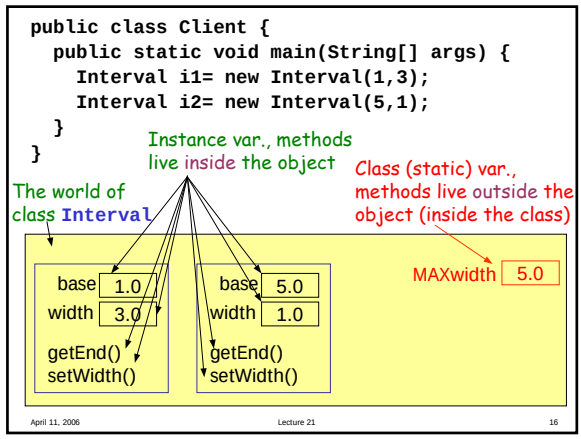
- Write an instance method
  `isIn(Interval i)`
- that returns the `boolean` value `true` if the instance is in `Interval i`. Return `false` otherwise.
- Parameter of non-primitive type: pass-by-reference
  I.e., Reference is copied; object itself is not copied


```
/** ={this Interval is in Interval i} */
public boolean isIn(Interval i) {
  return (     getBase()>=i.getBase() &&
               getEnd()<=i.getEnd() );
}
```



```
public class Client {
  public static void main(String[] args){
    Interval i1= new Interval(0.2,0.7);
    Interval i2= new Interval(
                      Math.random(),0.2);
    if (i2.isIn(i1))
      System.out.println("Interval i2 "
                 + "is in Interval i1.");
    else
      System.out.println("Interval i2 "
             + "is not in Interval i1.");
  }
}
```






```
class Interval {
  private double base;  // low end
  private double width; // interval width
  public static final double MAXwidth= 5; //...
  public Interval(double b, double w) {
    setBase(b);
    setWidth(w);
  }
  public double getEnd() {
    return base+width;
  }
  /* Set width to w, w<=MAXwidth */
  public void setWidth(double w) {
    width= Math.min(w,MAXwidth);
  }
}
```


Static Variables & Methods

- *Shared* by all instances of a class
- Only one copy no matter how many objects have been instantiated
- Keyword: **`static`**
- Examples:
  - A constant used by the whole class
  - A variable to keep track of how many Intervals have been created
  - A method that doesn't need to reference the fields in objects

Class (`static`) method

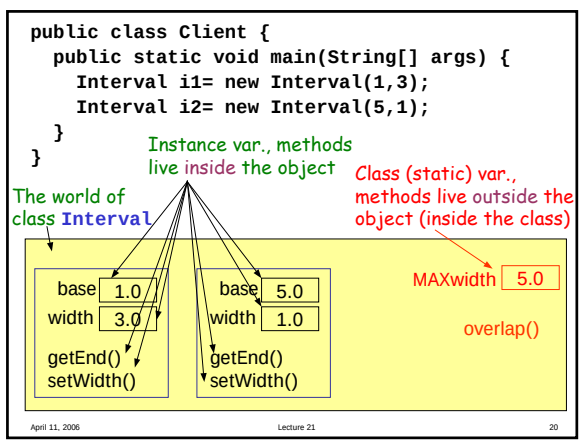
Write a class method

`overlap(Interval a, Interval b)`

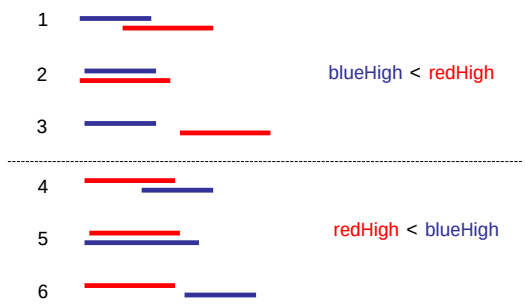
that returns a new **`Interval`** representing the overlap between **`Interval`**s **`a`** and **`b`**. (Return **`null`** if there's no overlap)

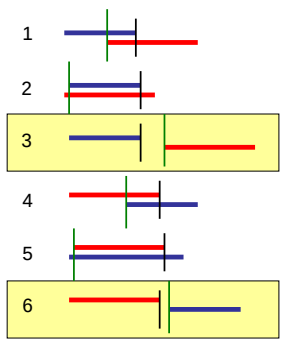
Where will the method live?

What is the method header?



```
/* =the overlapped Interval between
   Intervals a and b */
public static Interval overlap(
                Interval a, Interval b) {




}
```



The overlap's left is the rightmost of the two original lefts

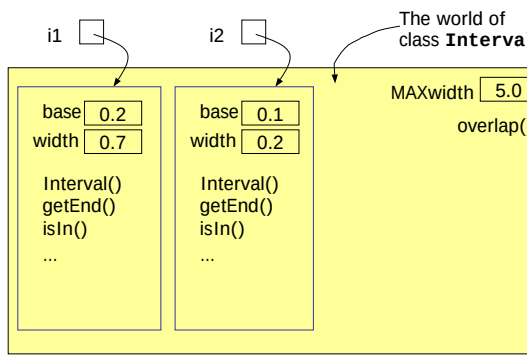
The overlap's right is the leftmost of the two original rights

No overlap if OLeft>ORight


```
/* =the overlapped Interval between
   Intervals a and b */
public static Interval overlap(Interval a,
                               Interval b) {
  Interval olap;      // overlapped interval
  double left, right; // olap's left & right

  left = Math.max(a.getBase(),b.getBase());
  right = Math.min(a.getEnd(),b.getEnd());
  if ( (right-left) <= 0 )
      olap= null;
  else
      olap= new Interval(left, right-left);
  return olap;
}
```



```
public class Client {
  public static void main(String[] args){
    Interval i1= new Interval(0.2,0.7);
    Interval i2= new Interval(
                    Math.random(),0.2);

    Interval o= Interval.overlap(i1,i2);

  }
}
```

An instance `overlap` method

- Write an instance method
  `overlap(...)`
  that returns a new `Interval` if two `Interval`s overlap. Return `null` otherwise.
- What is the method header? What should be the parameters, if any?
- Are the static and instance versions very different?

Method overloading

- Different methods can have the same name
- A method has a *signature*: method name and the parameter types (including the order)
- In a class, all methods must have different signatures
- E.g., the `abs` method in the `Math` class


```
class Interval {
  private double base;  // low end
  private double width; // interval width
  public static final double maxWidth=5;

  public Interval(double b, double w) {
    setBase(b);
    setWidth(w);
  }

  public Interval() {}

  /* An Interval with base b and maxWdith */
  public Interval(double b) {
    setBase(b);
    setWidth(maxWidth);
  }

  // other methods below
```


Chain invocation of methods

- Suppose there are 3 intervals: `i1`, `i2`, `i3`
- You know that `i1` and `i2` overlap
- Write code to find if the overlapped interval of `i1` and `i2` *is in* interval `i3`

```
Interval i1 = new Interval(…);
Interval i2 = new Interval(…);
Interval i3 = new Interval(…);
// Assume i1 and i2 overlap
if (                                    )
   System.out.println("in i3");
else
   System.out.println("not in i3");
```



```
Interval i1 = new Interval(…);
Interval i2 = new Interval(…);
Interval i3 = new Interval(…);
/* Without assuming that i1 and i2
   overlap */
```