Previous Lecture:
- Defining a class
  - Constructors
  - Keyword this
  - Method toString

Today's Lecture:
- Defining a class
  - Static variables and methods
  - Method overloading

Reading: Sec 4.3, Sec 5.1
Optional reading: Sec 5.2

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

### 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 fields

### Class (static) method
Write a class method
```
overlap(Interval a, Interval b)
```
that returns a new `Interval` representing the overlap between `Intervals a` and `b`.
(Return `null` if there's no overlap)

Where will the method live?
What is the method header?
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.

```java
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;
}
```

```java
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 head? 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");

A different example
- Create a Person class to organize data about a Person:
  - Name
  - Age
  - ...

public class Person {
    private String name;
    private int age;
    public static final int LEGALage=18;
    /** Constructor */
    public Person(String name, int age) {
        this.name= name; this.age= age; }
    /** =This Person is an adult */
    public boolean isAdult() {
        return age >= LEGALage; }
    /** =String description of this Person */
    public String toString() {
        return name + " is " + age; }
} // class Person

Modify Person class
- Modify Person class to store data about a Person’s best friend: add another instance variable friend
- What should be the type of the field friend?
- Add two more methods to the class definition: makeFriend, beFriendOf