

- Previous Lecture:
  - OO thinking
  - Defining a class:
    - Instance variables
    - Instance methods, getters and setters
  
- Today's Lecture:
  - Defining a class:
    - Constructors
    - Keyword `this`
    - Method `toString`
  
- Reading: Sec 4.4
- Announcement: P5 on its way...

April 6, 2006

Lecture 20

1

```
/* 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 w */
    public void setWidth(double w) {
        width= w;
    }
}
```

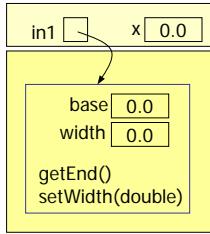
April 6, 2006

Lecture 20

5

## Calling an instance method

```
public class Client {
    public static void main(String[] args){
        Interval in1;
        in1= new Interval();
        double x;
        x= in1.getEnd();
    }
}
```



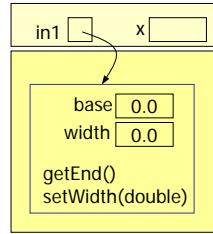
April 6, 2006

Lecture 20

7

## Calling an instance method

```
public class Client {
    public static void main(String[] args){
        Interval in1;
        in1= new Interval();
        double x;
        x= in1.base +
            in1.width;
    }
}
```



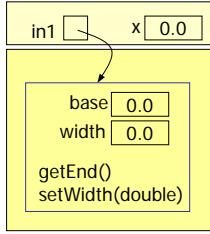
April 6, 2006

Lecture 20

8

## 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);
    }
}
```

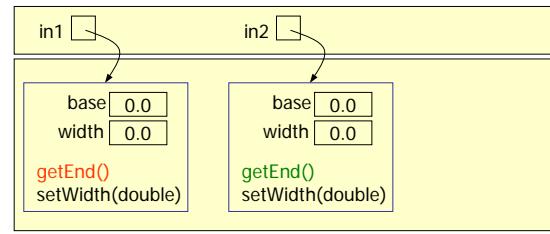


April 6, 2006

Lecture 20

10

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



## Class Definition

```
public class class-name {  
  
    declaration (and initialization)  
  
    constructor  
  
    methods  
  
}
```

April 6, 2006

Lecture 20

14

## 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

April 6, 2006

Lecture 20

15

```
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;  
    }  
}
```

April 6, 2006

Lecture 20

16

```
class Interval {  
    private double base; // low end  
    private double width; // interval width  
  
    /* Default constructor */  
    public Interval() {}  
  
  
    public double getEnd() {  
        return base + width;  
    }  
}
```

April 6, 2006

Lecture 20

17

## 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**

April 6, 2006

Lecture 20

18

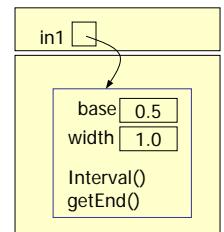
## Creating an object

```
public class Client {  
    public static void main(String[] args){  
  
        Interval in1;  
        in1= new Interval(  
            0.5,1);  
  
    }  
}
```

April 6, 2006

Lecture 20

19



```
public class IntervalClient {
    public static void main(String[] args) {
        Interval in1= new Interval(3,1);
    }
}
class Interval {
    ...
    public Interval(double b, double w) {
        base= b;
        width= w;
    }
    ...
}
```

The diagram illustrates the creation of an `Interval` object named `in1`. A yellow box labeled `in1` has an arrow pointing to a larger yellow box representing the constructor call `new Interval(3,1)`. This larger box contains fields `base` and `width`, both represented by small rectangular boxes, and methods `Interval()` and `getEnd()`, both represented by green rectangular boxes.

April 6, 2006

Lecture 20

20

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

- Keyword `this` returns a reference to the object itself, so `this.base` is the field `base` inside "this" object
- Use keyword `this` only when it is necessary. (It is not necessary in the example above.)

April 6, 2006

Lecture 20

22

```
public class IntervalClient {
    public static void main(String[] args) {
        Interval in1= new Interval(3,1);
    }
}
class Interval {
    ...
    public Interval(double base, double w) {
        this.base= base;
        width= w;
    }
    ...
}
```

This diagram is similar to the one above, but it includes the `this` keyword in the constructor assignment. The `in1` object points to a constructor call `new Interval(3,1)`, which then creates an `Interval` object with fields `base` and `width` and methods `Interval()` and `getEnd()`.

April 6, 2006

Lecture 20

23

## More instance methods with input parameters

- Write an instance method `expand(double f)` that expands the `Interval` by a factor of `f`.
- What should be the method header?
- Parameter of `primitive` type

April 6, 2006

Lecture 20

25

```
/** Expand this Interval by a
 *  factor of f
 */
public void expand(double f) {
    width *= f;
}
```

April 6, 2006

Lecture 20

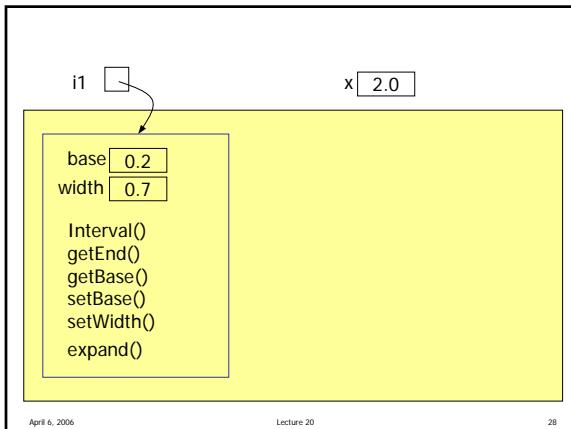
26

```
public class Client {
    public static void main(String[] args) {
        Interval il= new Interval(0.2,0.7);
        double x= 2;
        il.expand(x);
        System.out.println(il.getEnd());
    }
}
```

April 6, 2006

Lecture 20

27



April 6, 2006

Lecture 20

28

```
/** Expand this Interval by a
 * factor of f
 */
public void expand(double f) {
    setWidth(width*f);
}
```

Use available methods when possible!

April 6, 2006

Lecture 20

31

### 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

April 6, 2006

Lecture 20

32

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

April 6, 2006

Lecture 20

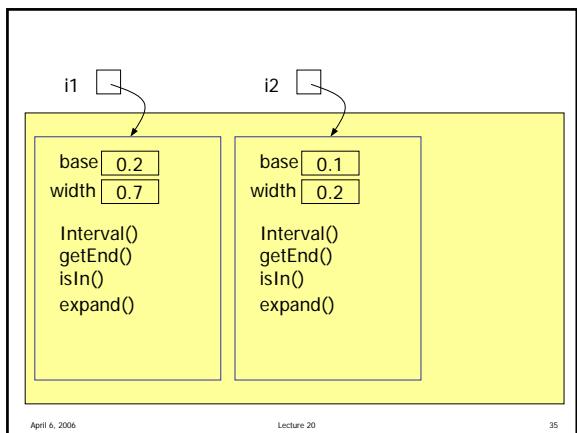
33

```
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.");
    }
}
```

April 6, 2006

Lecture 20

34



April 6, 2006

Lecture 20

35

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

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

Not concise!!

April 6, 2006

Lecture 20

36

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

public boolean isIn(Interval i) {
    if ( getBase()>=i.getBase() &&
        getEnd()<=i.getEnd()
        == true )
        return true;
    else
        return false;
}
```

Not concise!!

April 6, 2006

Lecture 20

37

### Method `toString()`

- Every object has default method `toString`
  - Automatically* invoked by `print`, `println`
- ```
Interval a = new Interval(1,2);
System.out.println(a);
```
- Some default text will be printed unless you define a `toString` method

April 6, 2006

Lecture 20

38

### Method `toString()`

- Usually defined to give a *useful* description of an instance of a class
- E.g., useful description of an instance of `Interval` would be the mathematical notation for an Interval, e.g.,

[3, 7.5]

for an `Interval` object with `base` 3 and `width` 4.5.

April 6, 2006

Lecture 20

39

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

    public Interval(double base, double w){
        this.base= base;
        width= w;
    }

    /** =String description of Interval */
    public String toString() {
        return "[" + getBase() + "," + getEnd() +
               "]";
    }
}
```

April 6, 2006

Lecture 20

40

```
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(i2 + "is in" +
                               i1);
        else
            System.out.println(i2 + "is not in" +
                               + i1);
    }
}
```

April 6, 2006

Lecture 20

42

```

/** Numeric interval -- closed intervals
 */
class Interval {

    private double base; // low end
    private double width; // interval width

    /** Constructor: An Interval has a specified base and width w */
    public Interval(double base, double w) {
        this.base= base;
        setWidth(w);
    }

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

    /** =Get base of this Interval */
    public double getBase() { return base; }

    /** Set width of this Interval to w */
    public void setWidth(double w) { width= w; }

    /** Expand this Interval by a factor of f (expand to the right) */
    public void expand(double f) {
        setWidth(width*f);
    }

    /** ={This Interval is in Interval i}
     * If the ends of this Interval and i are exactly equal, consider
     * this Interval to be in i.
     */
    public boolean isIn(Interval i) {
        return ( getBase()>=i.getBase() && getEnd()<=i.getEnd() );
    }

    /** =String description of this Interval */
    public String toString(){
        return "[" + getBase() + "," + getEnd() + "]";
    }
} //class Interval

```