Announcements:
- Project 5 due Thursday 4/14 at 6pm
- Section in computer labs this week
- Prelim 3 on Tues, 4/19

Previous Lecture:
- Defining a class:
  - Instance variables
  - Instance methods
  - Constructors

Today’s Lecture:
- Defining a class:
  - Constructors

Method toString

static variables and methods

Reading:
- JV: Sec 6.1-6.4, Sec 6.7
  (6.3, 6.4, 6.7 are very important)

---

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

---

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;
  }
}

---

Constructor
- A constructor is used to create objects
- Each class has a default constructor
- 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

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

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

- The parameter list of a method is enclosed in parentheses ( )
- The list is made of type-name pairs separated by commas
- E.g., (double b, double w)

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

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

```java
public class Interval {
    
    public Interval(double b, double w) {
        base = b;
        width = w;
    }
    
    public Interval() {
        this.base = 0.5;
        this.width = 1.0;
    }
}
```

```java
public class IntervalClient {
    public static void main(String[] args) {
        Interval in1 = new Interval(3,1);
    }
}
```

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

```java
public class Interval {
    
    public Interval(double b, double w) {
        this.base = b;
        this.width = w;
    }
    
    public Interval() {
        this.base = 0.5;
        this.width = 1.0;
    }
}
```

```java
public class IntervalClient {
    
    public static void main(String[] args) {
        Interval in1 = new Interval(3,1);
    }
}
```

```java
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.)
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;
    }
    ...
}

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

public class Client {
    public static void main(String[] args){
        Interval i1= new Interval(0.2,0.7);
        double x= 2;
        i1.expand(x);
        System.out.println(i1.getEnd());
    }
}

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;
    }
    ...
}

/** Set this Interval’s width to w */
public void setWidth(double w) {
    width = w;
}
/** Expand this Interval by a factor of f */
public void expand(double f) {
    setWidth(width*f);
}

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) {
        setBase(b);
        setWidth(w);
    }
    ...
}

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

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

Not concise!
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.");
    }
}

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

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.

```java
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() + "]";
    }
}
```
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 void setBase(double base) {
        this.base= base;
    }

    /* 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 fields