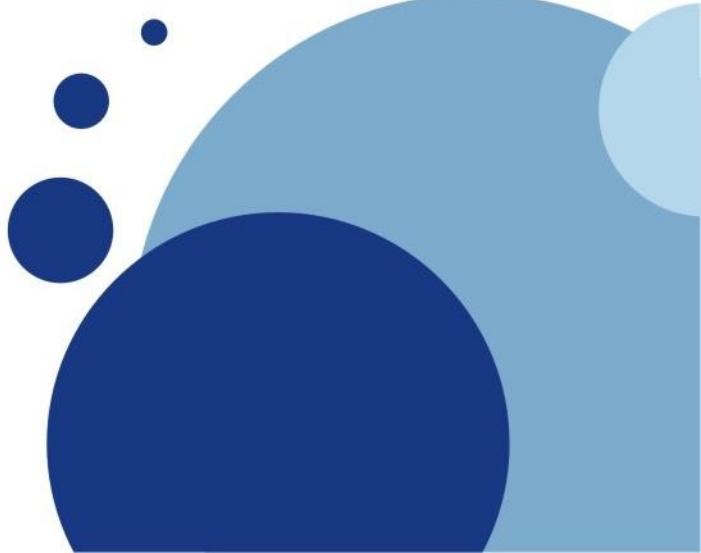


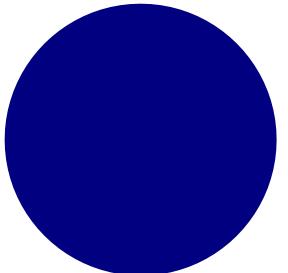
GEOG 178/258

Week 5:

Inheritance and Delegation

mike johnson





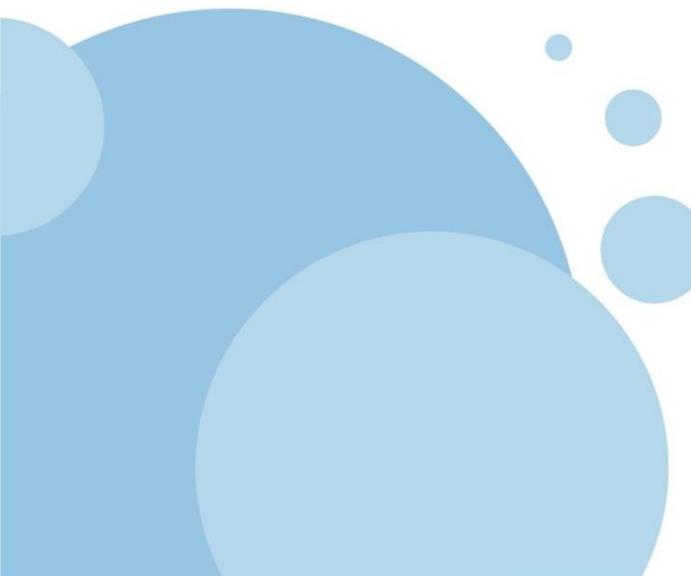
Inheritance

Week

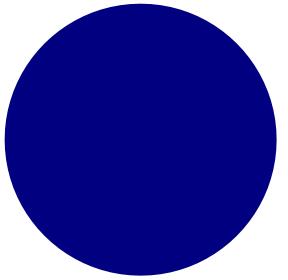
5



Inheritance



1. Inheritance is one of the key features of OOP and allows a **class** to use the **PROPERTIES** and **METHODS** of another **class**!
2. To do this, subclasses can inherit the **STATES** and **BEHAVIORS** of a super class through **EXTENSION**...
3. Subclasses can also add variables and methods to differentiate it from the superclass.
4. Note that a super class can have multiple subclasses, however a subclass can only have one superclass



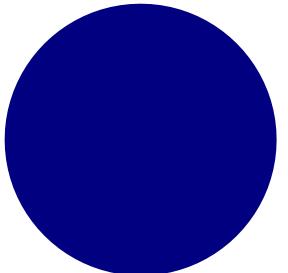
Delegation

Week

5

Delegation

1. In Java, when an object receives a request it can either perform the operation itself OR pass it to another object
2. Thus, delegation refers to a way of applying a method to an object outside of its direct class.
3. In a way, delegation can be seen as a “pointer” or a reference to a method
4. Example:
 - Say a request is sent to object 1. Object 1 chooses NOT to execute the responsibility and passes the responsibility (and itself) to object 2.
 - Object 2 is now the delegate and processes the request!



So what's the Difference?

Week

5



Inheritance v. Delegation



1. By using *Inheritance*, a subclass inherits all of the states and behaviors defined in the superclass.
2. By *delegation* you write another class with additional functionality that uses instances of the original class to provide the original functionality.
3. In other words:
 - When using inheritance, the subclass is simply a version of the superclass with some additional (or specific) functionality.
 - When using delegation, your delegating class contains a reference to an instance of the superclass and delegates the method calls to the superclass.

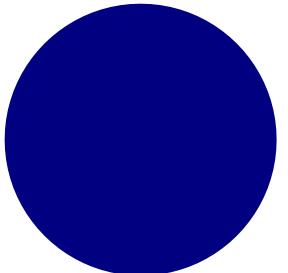
Things to Note about Private v. Public

Week

5

Example 2

1. The derived class inherits all the members and methods that are declared as public or protected.
2. If declared as private it can not be inherited by the derived classes.
3. The private members can be accessed only in its own class.
4. The private members can be accessed through assessor methods as shown in the example below. The derived class cannot inherit a member of the base class if the derived class declares another member with the same name.



But what about constructors?

Week

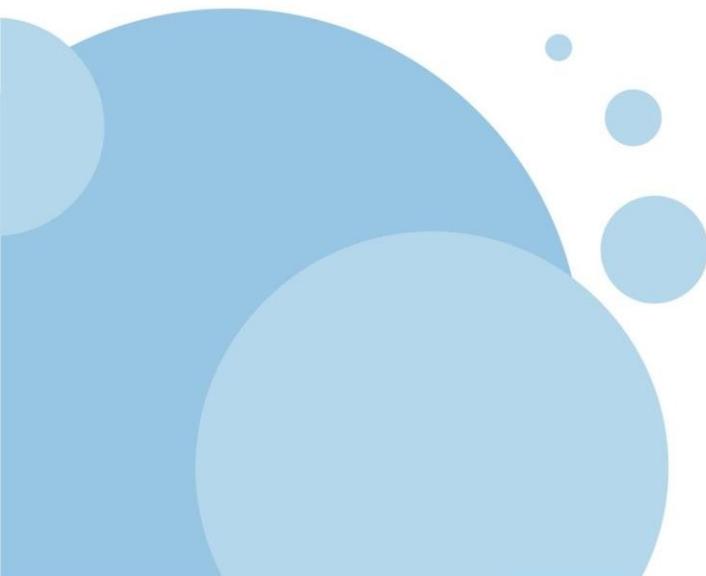
5

Example 2

1. The constructor in the superclass is responsible for building the object of the superclass.
2. The constructor of the subclass builds the object of subclass.
3. However! When the subclass constructor is called, it by default invokes the default constructor of super-class.
4. Hence, in inheritance the objects are constructed top-down.

EXAMPLE # 1

POINTS, POLYLINES, POLYGONS



Lets look at our point class....

Week

5

Example 1

```
1 package week5_examples;
2
3 public class Point {
4
5     //Member variables
6     private double x, y;
7
8     // Constructors
9     public Point(double x, double y) {
10         this.x = x;
11         this.y = y;
12     }
13
14     public Point() {
15         this(0,0);
16     }
17
18     //Getters & Setters
19     public double getX() { return x; }
20
21     public void setX(double x) { this.x = x; }
22
23     public double getY() { return y; }
24
25     public void setY(double y) { this.y = y; }
26
27     public String getType(){ return "POINT"; }
28
29     // Overrides
30     public String toString() {
31         return this.getType() + " ("+x+", "+y+")";
32     }
33
34     // Methods
35     public double getLength() { return 0.0; }
36
37     // Distance by double coordinate
38     public double distance(double x, double y) {
39         return Math.sqrt(Math.pow(this.x-x, 2) + Math.pow(this.y-y, 2));
40     }
41
42     // Distance by point
43     public double distance(Point p) {
44         return Math.sqrt(Math.pow(this.x-p.x, 2) + Math.pow(this.y-p.y, 2));
45     }
46 }
```

Tests!

Week

5

Example 1

```
1 package week5_examples;
2
3 public class Test {
4
5     public static void main(String[] args) {
6
7         Point p1 = new Point (2,2);
8         System.out.println(p1);
9         System.out.println(p1.getLength());
10        System.out.println(p1.getType());
11        System.out.println();
12
13        Point p2 = new Point (1,3);
14        System.out.println(p2);
15        System.out.println(p2.distance(p1));
16    }
17
18 }
19
```



The screenshot shows a Java application running in an IDE. The code in the editor is identical to the one above. In the bottom right corner, there is a terminal window titled "Console" showing the following output:

```
<terminated> Test (7) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_111
POINT (2.0, 2.0)
0.0
POINT

POINT (1.0, 3.0)
1.4142135623730951
```

Creating a class of POLYLINE

Week

5

Example 1

What do we know about Polylines?

- They are made up of points
- They must have at least 2 points

```
1 package week5_examples;
2
3 // Imports
4 import java.util.ArrayList;
5 import java.util.Iterator;
6 import java.util.Arrays;
7
8 public class Polyline {
9
10    //Member Variables
11
12    private ArrayList <Point> points;
13
14    //Constructors
15    public Polyline() {
16        points = new ArrayList<Point>();
17    }
18
19    public Polyline(ArrayList<Point> points) {
20        this.points = points;
21    }
}
```

Delegating ARRAYLIST methods. Getter and Setters

Week

5

Example 1

```
23 // Delegates
24 public boolean add(Point e) {
25     return getPoints().add(e);
26 }
27
28 public int size() {
29     return getPoints().size();
30 }
31
32 // Getters
33
34 public String getType() {
35     return "POLYLINE";
36 }
37
38 public ArrayList<Point> getPoints(){
39     return points;
40 }
41
42 //Setters
43 public void setPoints(ArrayList <Point> points) {
44     this.points = points;
45 }
```

‘Add’ and ‘size’ are methods of ARRAYLISTS. When it comes to POLYLINES we want them to behave in a certain way. Therefore we must DELEGATE the method.

Overrides and Methods

Week

5

Example 1

```
47 //Overrides
48
49 public String toString() {
50     return this.getType() + " " + Arrays.toString(getPoints().toArray());
51 }
52
53 //Methods
54 public boolean checkValid() {
55     return getPoints().size() >= 2;
56 }
57
58 public double getLength() {
59     Iterator<Point> pointIterator = getPoints().iterator();
60     Point lastP = pointIterator.next();
61     Double distance = 0.0;
62     while (pointIterator.hasNext()) {
63         Point p = pointIterator.next();
64         distance += lastP.distance(p);
65         lastP = p;
66     }
67     return distance;
68 }
```

Full Polyline

Week

5

Example 1

```
1 package week5_examples;
2
3 // Imports
4 import java.util.ArrayList;
5 import java.util.Iterator;
6 import java.util.Arrays;
7
8 public class Polyline {
9
10    //Member Variables
11
12    private ArrayList <Point> points;
13
14    //Constructors
15    public Polyline() {
16        setPoints(new ArrayList<Point>());
17    }
18
19    public Polyline(ArrayList<Point> points) {
20        this.setPoints(points);
21    }
22
23    // Delegates
24    public boolean add(Point e) {
25        return getPoints().add(e);
26    }
27
28    public int size() {
29        return getPoints().size();
30    }
31
32    // Getters
33
34    public String getType() {
35        return "POLYLINE";
36    }
37
38    public ArrayList<Point> getPoints(){
39        return points;
40    }
41
42    //Setters
43    public void setPoints(ArrayList <Point> points) {
44        this.points = points;
45    }
46
47    //Overrides
48
49    public String toString() {
50        return this.getType() + " " + Arrays.toString(getPoints().toArray());
51    }
52
53    //Methods
54    public boolean checkValid() {
55        return getPoints().size() >= 2;
56    }
57
58    public double getLength() {
59        Iterator<Point> pointIterator = getPoints().iterator();
60        Point lastP = pointIterator.next();
61        Double distance = 0.0;
62        while (pointIterator.hasNext()) {
63            Point p = pointIterator.next();
64            distance += lastP.distance(p);
65            lastP = p;
66        }
67        return distance;
68    }
69 }
```

Tests!

Week

5

Example 1

```
1 package week5_examples;
2
3 import java.util.ArrayList;
4
5 public class Test {
6
7     public static void main(String[] args) {
8
9         Point p1 = new Point (2,2);
10        Point p2 = new Point (1,3);
11
12        Polyline pl = new Polyline();
13        pl.add(p1);
14        pl.add(p2);
15        System.out.println(pl);
16        System.out.println("Distance: " + pl.getLength());
17        System.out.println("Valid GEOM: " + pl.checkValid());
18        System.out.println();
19
20        ArrayList<Point> pts = new ArrayList<Point>();
21        pts.add(p1);
22        pts.add(p2);
23        Polyline pl2 = new Polyline(pts);
24        System.out.println(pl2);
25
26    }
27
28 }
```

```
Problems @ Javadoc Declaration Console
<terminated> Test (7) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_161.jdk/Contents/Home
POLYLINE [POINT (2.0, 2.0), POINT (1.0, 3.0)]
Distance: 1.4142135623730951
Valid GEOM: true

POLYLINE [POINT (2.0, 2.0), POINT (1.0, 3.0)]
```

POLYGON

Week

5

What do we know about Polygons?

They are closed polylines

They require 3 or more unique points

Almost all polyline methods apply to polygon

Example 1

```
1 package week5_examples;  
2  
3 import java.util.ArrayList;  
4  
5 // Polygon extends Polyline  
6 public class Polygon extends Polyline {  
7
```

POLYLINE → POLYGON

Week

5

Example 1

```
1 package week5_examples;
2
3 import java.util.ArrayList;
4
5 // Polygon extends Polyline
6 public class Polygon extends Polyline {
7
8    public Polygon() {
9        setPoints(new ArrayList<Point>());
10    }
11
12    public Polygon(ArrayList<Point> points) {
13        super(points);
14    }
}
```

Will this work??

POLYLINE → POLYGON

Week

5

Example 1

```
1 package week5_examples;
2
3 import java.util.ArrayList;
4
5 // Polygon extends Polyline
6 public class Polygon extends Polyline {
7
8    public Polygon() {
9        setPoints(new ArrayList<Point>());
10    }
11
12    public Polygon(ArrayList<Point> points) {
13        super(points);
14    }
}
```

```
1 package week5_examples;
2
3 // Imports
4 import java.util.ArrayList;
5 import java.util.Iterator;
6 import java.util.Arrays;
7
8 public class Polyline {
9
10    //Member Variables
11    private ArrayList <Point> points;
12
13    //Constructors
14    public Polyline() {
15        setPoints(new ArrayList<Point>());
16    }
17
18    public Polyline(ArrayList<Point> points) {
19        this.setPoints(points);
20    }
21
22}
```

'points' is PRIVATE!

Encapsulation

POLYLINE → POLYGON

Week

5

Example 1

```
16     // Override check valid with new values
17     public boolean checkValid() {
18         return getPoints().size() >= 3;
19     }
20
21     public String getType() {
22         return "POLYGON";
23     }
24
25     // Make valid geometries
26
27     public boolean makeValid(){
28
29         if(!checkValid()) return false;
30
31         Point first = getPoints().get(0);
32         Point last = getPoints().get(getPoints().size() - 1);
33
34         if(first.getX() != last.getX() || first.getY() != last.getY()) {
35             return getPoints().add(first);
36         }
37
38         return true;
39     }
```

Full Polygon

Week

5

Example 1

```
1 package week5_examples;
2
3 import java.util.ArrayList;
4
5 // Polygon extends Polyline
6 public class Polygon extends Polyline {
7
8     public Polygon() {
9         setPoints(new ArrayList<Point>());
10    }
11
12    public Polygon(ArrayList<Point> points) {
13        super(points);
14    }
15
16    // Override check valid with new values
17    public boolean checkValid() {
18        return getPoints().size() >= 3;
19    }
20
21    public String getType() {
22        return "POLYGON";
23    }
24
25    // Make valid geometries
26
27    public boolean makeValid(){
28
29        if(!checkValid()) return false;
30
31        Point first = getPoints().get(0);
32        Point last = getPoints().get(getPoints().size() - 1);
33
34        if(first.getX() != last.getX() || first.getY() != last.getY()) {
35            getPoints().add(first);
36        }
37
38        return true;
39    }
40
41 }
```

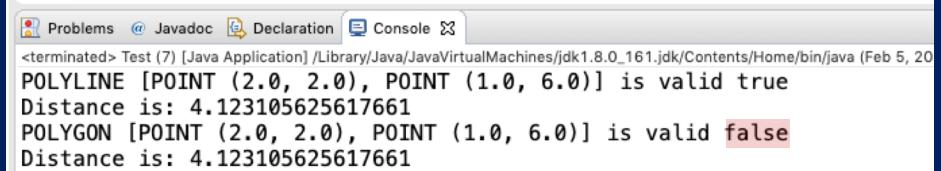
Tests!

Week

5

Example 1

```
1 package week5_examples;
2
3 import java.util.ArrayList;
4
5 public class Test {
6
7     public static void main(String[] args) {
8
9         ArrayList<Point> pts = new ArrayList<Point>();
10        pts.add(new Point (2,2));
11        pts.add(new Point (1,6));
12
13        Polyline pl = new Polyline(pts);
14        Polygon pg = new Polygon(pts);
15        System.out.println(pl + " is valid " + pl.checkValid());
16        System.out.println("Distance is: " + pl.getLength());
17        System.out.println(pg + " is valid " + pg.checkValid());
18        System.out.println("Distance is: " + pg.getLength());
19    }
20
21 }
```



The screenshot shows the Java application's console output. It displays the creation of a Polyline object with two points and its validation status. It then creates a Polygon object with the same points and checks its validity. The output indicates that the Polyline is valid and has a distance of approximately 4.123, while the Polygon is invalid (false).

```
Problems @ Javadoc Declaration Console
<terminated> Test (7) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_161.jdk/Contents/Home/bin/java (Feb 5, 20
POLYLINE [POINT (2.0, 2.0), POINT (1.0, 6.0)] is valid true
Distance is: 4.123105625617661
POLYGON [POINT (2.0, 2.0), POINT (1.0, 6.0)] is valid false
Distance is: 4.123105625617661
```

Tests!

Week

5

Example 1

```
1 package week5_examples;
2
3 import java.util.ArrayList;
4
5 public class Test {
6
7     public static void main(String[] args) {
8
9         ArrayList<Point> pts = new ArrayList<Point>();
10        pts.add(new Point(2,2));
11        pts.add(new Point(1,6));
12
13        Polyline pl = new Polyline(pts);
14        Polygon pg = new Polygon(pts);
15        System.out.println(pl + " is valid: " + pl.checkValid());
16        System.out.println("Distance is: " + pl.getLength());
17        System.out.println(pg + " is valid: " + pg.checkValid());
18        System.out.println("Distance is: " + pg.getLength());
19        System.out.println();
20
21        pg.add(new Point(4,12));
22        System.out.println(pg + " is valid: " + pg.checkValid());
23        System.out.println("Distance is: " + pg.getLength());
24        System.out.println(pg + " has " + pg.size() + " points.");
25        System.out.println();
26
27        pg.makeValid();
28        System.out.println(pg + " is valid: " + pg.checkValid());
29        System.out.println("Distance is: " + pg.getLength());
30        System.out.println(pg + " has " + pg.size() + " points.");
31    }
32
33 }
34
```

Problems @ Javadoc Declaration Console

<terminated> Test (7) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_161.jdk/Contents/Home/bin/java (Feb 5, 2019, 10:09:02 AM)

POLYLINE [POINT (2.0, 2.0), POINT (1.0, 6.0)] is valid: true
Distance is: 4.123105625617661
POLYGON [POINT (2.0, 2.0), POINT (1.0, 6.0)] is valid: false
Distance is: 4.123105625617661

POLYGON [POINT (2.0, 2.0), POINT (1.0, 6.0), POINT (4.0, 12.0)] is valid: true
Distance is: 10.831309558117031
POLYGON [POINT (2.0, 2.0), POINT (1.0, 6.0), POINT (4.0, 12.0)] has 3 points.

POLYGON [POINT (2.0, 2.0), POINT (1.0, 6.0), POINT (4.0, 12.0), POINT (2.0, 2.0)] is valid: true
Distance is: 21.0293485853026
POLYGON [POINT (2.0, 2.0), POINT (1.0, 6.0), POINT (4.0, 12.0), POINT (2.0, 2.0)] has 4 points.

EXAMPLE # 2

BOUNDING BOXES, DISASTERS, FLOODS



Bounding Boxes

Week

5

Example 2

```
1 package week5_examples;
2
3 public class BoundingBox {
4
5     // Member Variables
6     private Point LR; //xmax, ymin
7     private Point UL; //xmin, ymax
8
9     // Constructors
10    public BoundingBox(double x1, double y1, double x2, double y2) {
11        this.LR = new Point(Math.max(x1, x2), Math.min(y1, y2));
12        this.UL = new Point(Math.min(x1, x2), Math.max(y1, y2));
13    }
14
15    public BoundingBox(Point p1, Point p2) {
16        if(p1.getX() > p2.getX()) {
17            this.LR = p1;
18            this.UL = p2;
19        } else {
20            this.LR = p2;
21            this.UL = p1;
22        }
23
24        // Getters & setters
25
26        public Point getUL() { return UL; }
27
28        public void setUL(Point p1) { this.UL = p1; }
29
30        public Point getLR() { return LR; }
31
32        public void setLR(Point p2) { this.LR = p2; }
33
34        public String getType(){ return "BOUNDINGBOX"; }
35
36        // String conversion
37        public String toString() { return this.getType()+" ("+UL+", "+LR+"); }
38
39
40        public boolean isInside(Point p) {
41            if (UL.getX() > p.getX() || LR.getX() < p.getX()) return false;
42            if (UL.getY() < p.getY() || LR.getY() > p.getY()) return false;
43            return true;
44        }
45
46        public double area() {
47            return Math.abs(UL.getX() - LR.getX()) * Math.abs(UL.getY() - LR.getY());
48        }
49    }
```

Tests

Week

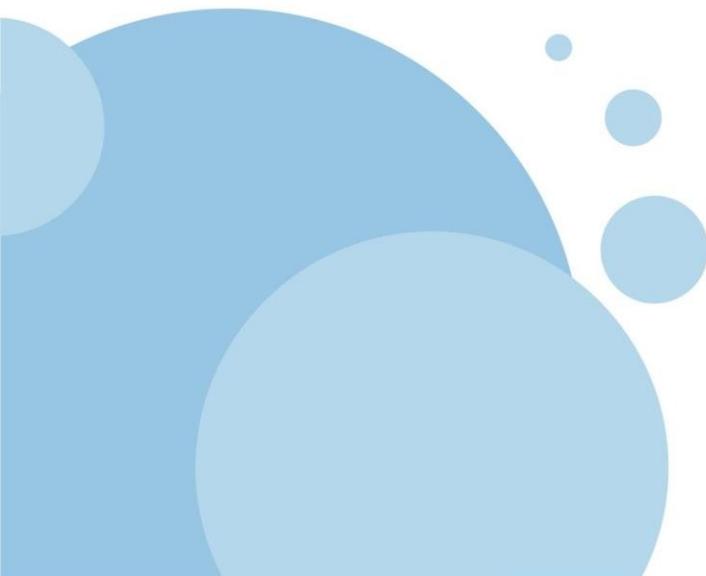
5

Example 2

```
1 package week5_examples;
2
3 public class Test {
4
5     public static void main(String[] args) {
6
7         BoundingBox bb = new BoundingBox(0,10, 10, 0);
8         System.out.println(bb + " has an area of " + bb.area());
9         BoundingBox bb_flip = new BoundingBox(10,0, 0, 10);
10        System.out.println(bb_flip + " has an area of " + bb_flip.area());
11
12        System.out.println(bb.isInside(new Point (5,5)));
13        System.out.println(bb_flip.isInside(new Point (5,5)));
14        System.out.println(bb.isInside(new Point (50,50)));
15        System.out.println(bb_flip.isInside(new Point (50,50)));
16    }
17
18 }
19 }
```

```
Problems @ Javadoc Declaration Console ✎
<terminated> Test (7) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_161.jdk/Contents/Home/bin/java (Feb 5, 2019, 10:20:10 AM)
BOUNDINGBOX (POINT (0.0, 10.0), POINT (10.0, 0.0)) has an area of 100.0
BOUNDINGBOX (POINT (0.0, 10.0), POINT (10.0, 0.0)) has an area of 100.0
true
true
false
false
```

Work through together in class



Disasters are temporal BBOXs!

Week

5

Example 2

```
1 package week5_examples;  
2  
3 public class Disaster extends BoundingBox {  
4  
5     private int duration; // the number of weeks a disaster lasts  
6
```

Inherit the constructors and member variables from BBOX

Week

5

Example 2

```
1 package week5_examples;
2
3 public class Disaster extends BoundingBox {
4
5     private int duration; // the number of weeks a disaster lasts
6
7     public Disaster(double x1, double y1, double x2, double y2) {
8         super(x1, y1, x2, y2);
9         this.duration = 0;
10    }
11
12    public Disaster(Point UL, Point LR) {
13        super(UL, LR);
14        this.duration = 0;
15    }
}
```

Note that $(x1, y1, x2, y2, UL, LR)$ are all inherited from the super class **BoundingBox** while **duration** is a member variable of the class **disaster**.

Add getters and setters

Week

5

Example 2

```
16  
17  public String getType() {  
18      return "Disaster";  
19  }  
20  
21  public int getDuration() {  
22      return duration;  
23  }  
24  
25  public void setDuration(int d) {  
26      this.duration = d;  
27  }  
28  
29
```

Full Disaster Class

Week

5

Example 2

```
1 package week5_examples;
2
3 public class Disaster extends BoundingBox {
4
5     private int duration; // the number of weeks a disaster lasts
6
7     public Disaster(double x1, double y1, double x2, double y2) {
8         super(x1, y1, x2, y2);
9         this.duration = 0;
10    }
11
12    public Disaster(Point UL, Point LR) {
13        super(UL, LR);
14        this.duration = 0;
15    }
16
17    public String getType() {
18        return "Disaster";
19    }
20
21    public int getDuration() {
22        return duration;
23    }
24
25    public void setDuration(int d) {
26        this.duration = d;
27    }
28 }
29
```

Disaster to Flood

Week

5

Example 2

```
1 package week5_examples;
2
3 public class Flood extends Disaster {
4
5     public Flood(double x1, double y1, double x2, double y2) {
6         super(x1, y1, x2, y2);
7         setDuration(3);
8     }
9
10    public Flood(Point UL, Point LR) {
11        super(UL, LR);
12        setDuration(3);
13    }
14
15    public String getType() {
16        return "Flood";
17    }
18
19}
20
```

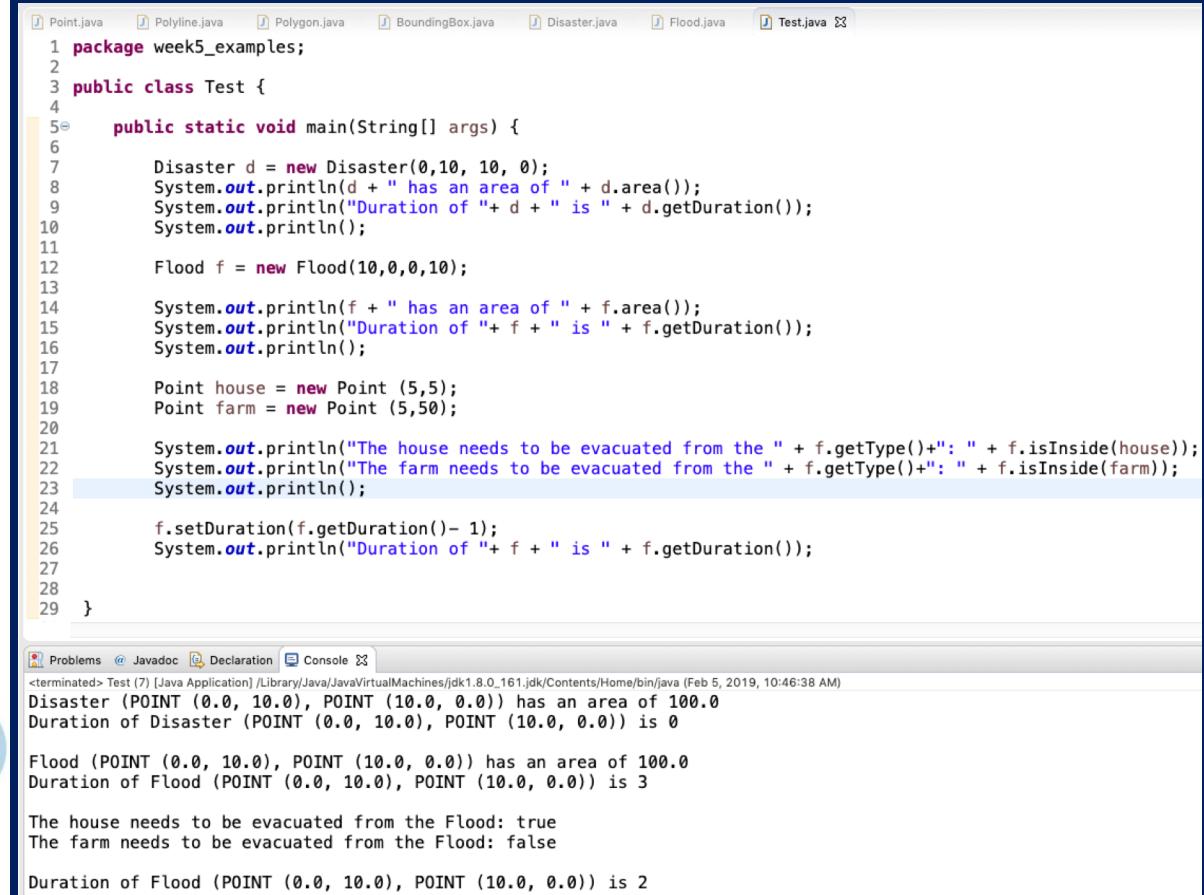
The `setDuration()` method allows us access to the private duration member variable of DIASASTER (encapsulation)

Tests

Week

5

Example 2



The screenshot shows an IDE interface with a code editor and a console window.

Code Editor:

```
1 package week5_examples;
2
3 public class Test {
4
5     public static void main(String[] args) {
6
7         Disaster d = new Disaster(0,10, 10, 0);
8         System.out.println(d + " has an area of " + d.area());
9         System.out.println("Duration of " + d + " is " + d.getDuration());
10        System.out.println();
11
12        Flood f = new Flood(10,0,0,10);
13
14        System.out.println(f + " has an area of " + f.area());
15        System.out.println("Duration of " + f + " is " + f.getDuration());
16        System.out.println();
17
18        Point house = new Point (5,5);
19        Point farm = new Point (5,50);
20
21        System.out.println("The house needs to be evacuated from the " + f.getType() + ": " + f.isInside(house));
22        System.out.println("The farm needs to be evacuated from the " + f.getType() + ": " + f.isInside(farm));
23        System.out.println();
24
25        f.setDuration(f.getDuration()- 1);
26        System.out.println("Duration of " + f + " is " + f.getDuration());
27
28    }
29 }
```

Console Window:

```
<terminated> Test (7) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_161.jdk/Contents/Home/bin/java (Feb 5, 2019, 10:46:38 AM)
Disaster (POINT (0.0, 10.0), POINT (10.0, 0.0)) has an area of 100.0
Duration of Disaster (POINT (0.0, 10.0), POINT (10.0, 0.0)) is 0

Flood (POINT (0.0, 10.0), POINT (10.0, 0.0)) has an area of 100.0
Duration of Flood (POINT (0.0, 10.0), POINT (10.0, 0.0)) is 3

The house needs to be evacuated from the Flood: true
The farm needs to be evacuated from the Flood: false

Duration of Flood (POINT (0.0, 10.0), POINT (10.0, 0.0)) is 2
```

Homework Hints!

Week

5

Homework Hints

Print the following three statements with the information filled in correctly:

- 1) There are <#> farms, <#> stores and <#> hospitals within the <disaster type> bounding box.
- 2) <#> structures are unaffected.
- 3) The <#> is defined by the points <P1>, <P2>.

Inheritance		
Superclass	Point	Bounding Box
Sub-class	Building	Disaster
Sub-class	Hospital; Farm; Store	Flood; Fire