

Object Oriented

January 29, 2019

1 Covered writing the program

1.1 Files Used:

1. Day4Code & Lab

1.2 Notes

I never downloaded the files for this, so I can't compile yet.
%5d is the format specifier that should make things align

```
public class CheckArrays {
public static boolean sameDimensions(int[][] theA, int[][] theB) {
    return theA.length == theB.length && theA[0].length ==
        theB[0].length;
}
public static boolean areEqual(int[][] theA, int[][] theB) {
    boolean result = sameDimensions(theA, theB);
    for (int row = 0; row < theA.length && result; row++) {
        for (int col = 0; col < theA[row] && result; col++) {
            result = theA[row][col] == theB[row][col];
        }
    }
    return result;
}
}
public static int sum(int[][] theA, int[][] theB) {
int result = 0;
for (int row = 0; row < theA.length; row++) {
    for (int col = 0; col < theA[row]; col++) {
        result += theA[row][col];
    }
}
return result;
}
}
```

2 Day 5 Code and notes

2.0.1 Misc Notes

This is the code for the day explaining a bomb program that uses point classes and final parameters. These points make use of implied parameters, and there are **this** code references??

if throwing exceptions, do that first in a method.

1. Protecting methods is one of the key points in this program
2. reuseability is also key as demonstrated in this program

```
package app;

public final class Point {
    public static void main(String[] args) throws Exception {
        public static final int DEFAULT_X = 0;
        public static final int DEFAULT_Y = 0;
        private int myX;
        private int myY;
        public Point(final int theX, final int theY) {
            if (theX < 0 || theY < 0) {
                throw new IllegalArgumentException("Coordinates cannot "
                    + "be negative.");
            }
            myX = theX;
            myY = theY;
        }
        public Point() {
            this(DEFAULT_X, DEFAULT_Y);
        }
        public Point(Point theP) {
            this(theP.myX, theP.myY);
        }
        public int getX() {
            return myX;
        }
        public int getY() {
            return myY;
        }

        public double calculateDistance(final Point theOtherPoint) {
            if (theOtherPoint == null) {
                throw new NullPointerException ("Cannot use a point of
                    null" + "to calculate a distance");
            }
        }
    }
}
```

```

        final double dx = myX - theOtherPoint.myX;
        final double dy = myY - theOtherPoint.myY;
        return Math.sqrt(dx * dx + dy * dy);
    }

    public void setX(final int theX) {
        if (theX < 0) {
            throw new IllegalArgumentException("Coordinates cannot "
                + "be negative.");
        }
        // This is called a mutator method
        myX = theX;
    }

    public void setY(final int theY) {
        if (theY < 0) {
            throw new IllegalArgumentException("Coordinates cannot "
                + "be negative.");
        }
        // This is called a mutator method
        myY = theY;
    }
}

}

public final class Point {
    public static void main(String[] args) throws Exception {
        public static final int DEFAULT_X = 0;
        public static final int DEFAULT_Y = 0;
        private int myX;
        private int myY;
        public Point(final int theX, final int theY) {
            if (theX < 0 || theY < 0) {
                throw new IllegalArgumentException("Coordinates cannot "
                    + "be negative.");
            }
            myX = theX;
            myY = theY;
        }
        public Point() {
            this(DEFAULT_X, DEFAULT_Y);
        }
        public Point(Point theP) {
            this(theP.myX, theP.myY);
        }
        public int getX() {
            return myX;
        }
        public int getY() {
            return myY;
        }
    }
}

```

```

public double calculateDistance(final Point theOtherPoint) {
    if (theOtherPoint == null) {
        throw new NullPointerException ("Cannot use a point of
            null" + "to calculate a distance");
    }
    final double dx = myX - theOtherPoint.myX;
    final double dy = myY - theOtherPoint.myY;
    return Math.sqrt(dx * dx + dy * dy);
}

public void setX(final int theX) {
    if (theX < 0) {
        throw new IllegalArgumentException("Coordinates cannot "
            + "be negative.");
    }
    // This is called a mutator method which change state of
    // object
    myX = theX;
}

public void setLocation(int theX, int theY) {
    if (theX < 0 || theY < 0) {
        throw new IllegalArgumentException("Coordinates cannot "
            + "be negative.");
    }
    // This is called a mutator method which change state of
    // object
    myX = theX;
    myY = theY;
}

public void translate(int theX, int theY) {
    if (theX < 0 || theY < 0) {
        throw new IllegalArgumentException("Coordinates cannot "
            + "be negative.");
    }
    // This is called a mutator method which change state of
    // object
    setLocation(myX + theX, myY + theY);
}

public String toString(){
    String result = "";
    result += "Point";
    result += "(";
    result += myX;
    result += ", " + myY + ")";
    return result;
}
}

```

```
}
```

This is another class held in a different file that deals with the checkpoints.

```
// This is a a different class that we made in a differnet file that we
    used to execute the point class and make the points
public class Checkpoint {
    public static void main(String[] theArgs) {
        Point p1 = new Point(s, 9);
        Point p2 = new Point();
        Point p3 = new Point(p1);
        // Note that here you don't have to do p1.toString()
        // Becasue it is implicitly understood as that
        System.out.println(p1 + "\n" + p2 + "\n" + p3);
        p1.setLocation()
    }
}
```

3 1/29/2019

3.0.1 Misc

Files used:

1. codeFromClass/MyClass.java

3.1 Inheritance

We started an employee program that demonstrated common characteristics among them.

when you Inherite, you do:

```
public class secretary extends Employee {
    // This is where the method goes
}
```

You can override to a new method in the subclass
When you need to access base pay for example, you should call the super class:

```
public class Legalsecretary extends Secretary {  
    public double Salary() {  
        // The keyword super grabs from the super class the base pay for  
        example  
        super.getSalary();  
    }  
}
```

You can call the superclasses constructors by doing this:

```
super(years) // Calls the Employee constructor
```

3.1.1 Next programming project

Nevermind the crazy driver it just tests the methods

Do not write it based on the output, or you'll be jacked up. Its meant to be confusing

Makes use of interfacing which describe the methods that you have to include in your class

Needs javadoc for every class and even the constructor

3.2 Polymorphism

Every class is an objectm