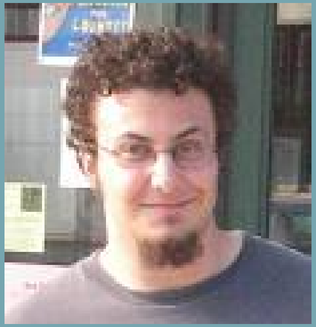
Observer Design Pattern in Java

Speaker: Joseph Paul Cohen
Knowledge Discovery Lab
University of Massachusetts Boston

Who am I?



Research Assistant at the Knowledge Discovery Lab

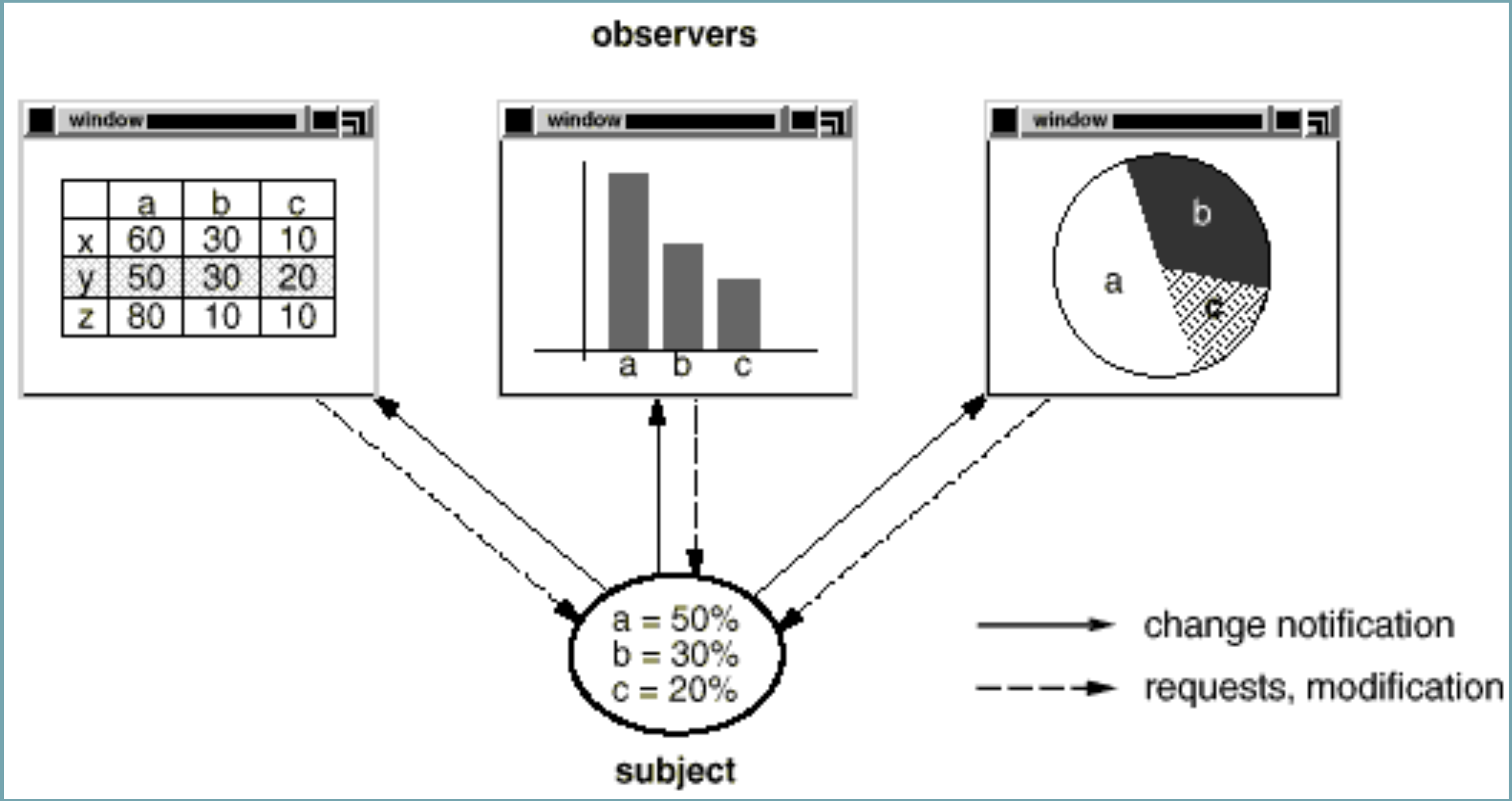
Software Engineer at Viridity Software (Startup Company)

System Engineer/Security Consultant Managing over 700 HIPAA regulated computers.

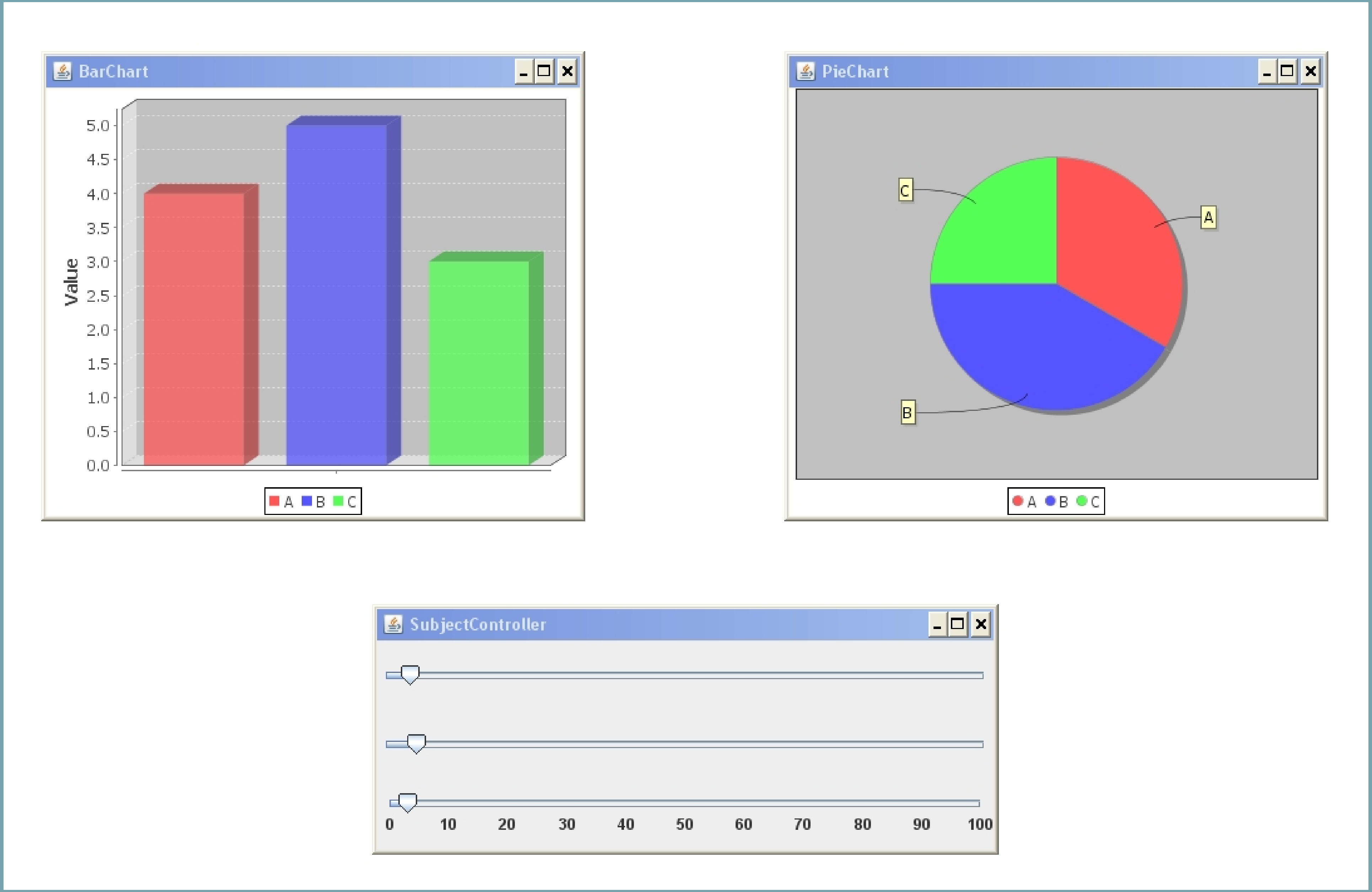
Writing Java for >3 years
Current Project has

Language	files	blank	comment	code
XML	315	2,739	146,881	1,112,382
Java	842	21,168	35,218	94,707
Perl	268	8,935	20,549	38,839

(cloc.sourceforge.net)



From Design Patterns: Elements of Reusable Object-Oriented Software
Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides



http://www.cs.umb.edu/~joecohen/observerdesignpattern/ObserverDesignPattern.jnlp

Observer Design Pattern is

Is the inferior of **Publish/Subscribe Pattern**
-Subject is publisher
-Observers subscribe

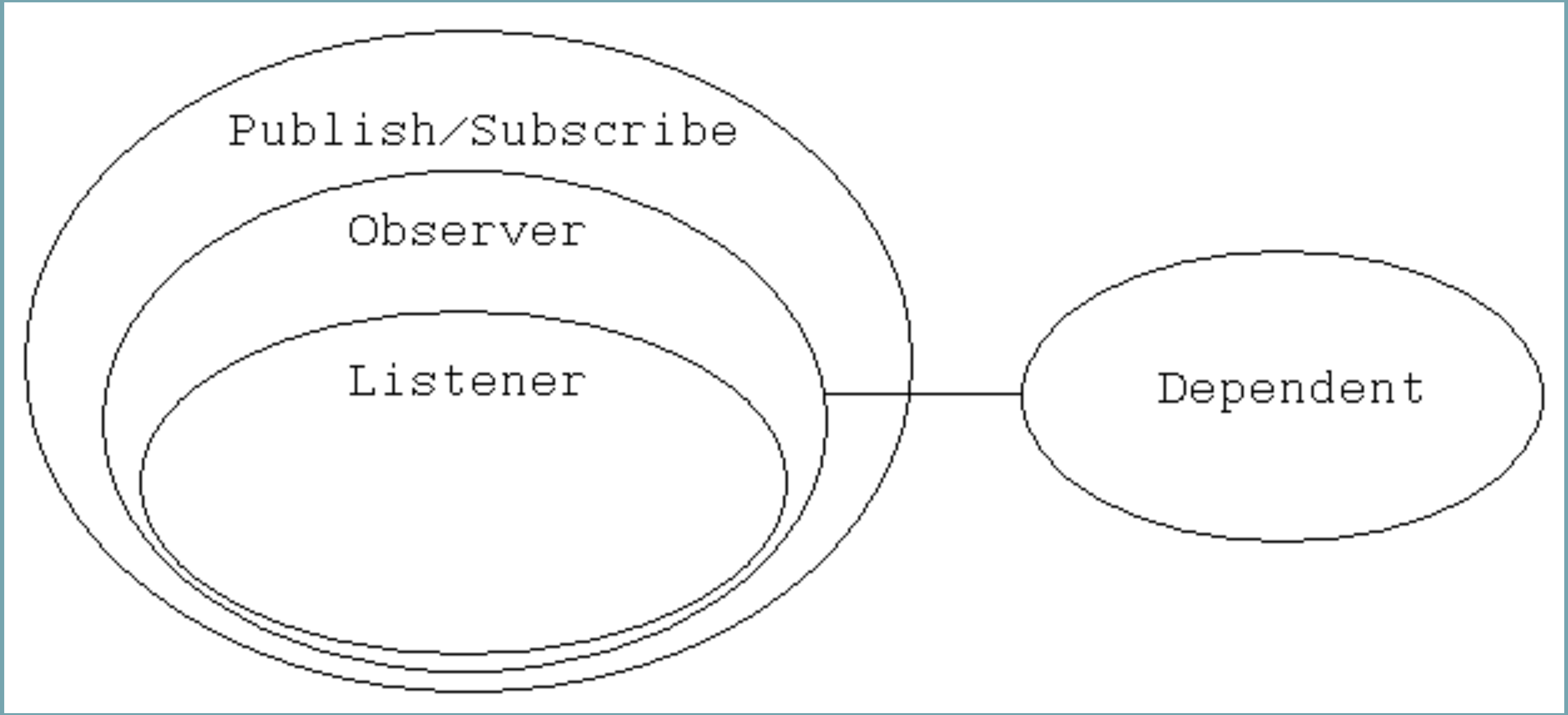
Is the superior of the **Listener Pattern**
-Subject is being listened to
-Observers are listeners

Is an implementation of the **Dependent Pattern**
-Subject has things that depend on it
-Observers can be dependent on the Subject

Participants

Subject
-Knows it's observers.
-No limit to observers
-Provides an interface to attach/detach observer objects
-Defines what the observers can listen to.

Observer
-Implements the interface that can Observe the Subject


Publish/Subscribe
Observer
Listener
Dependent

Who is in charge?

Who's code gets run?

Who triggers the code to run?

Who is in charge?
Subject

Who's code gets run?

Who triggers the code to run?

Who is in charge?
Subject

Who's code gets run?
Observers

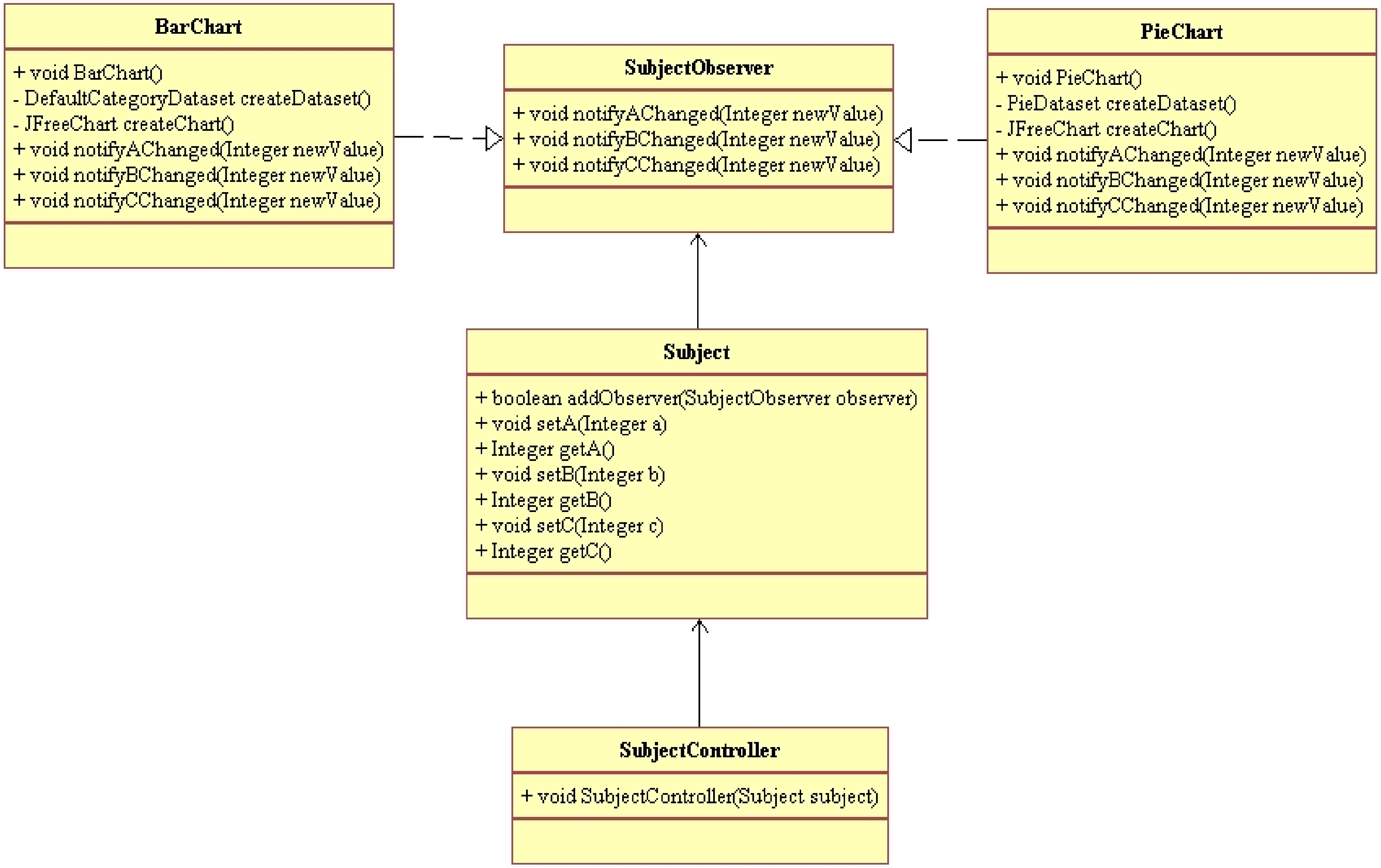
Who triggers the code to run?

Who is in charge?
Subject

Who's code gets run?
Observers

Who triggers the code to run?
Subject

```
public static void main(String[] args) {
        /*
        * We create a subject and something to control it
        */
        Subject subject = new Subject();
        SubjectController controller = new SubjectController(subject);
        /*
        * We create an Observer that displays a bar graph
        */
        BarChart barGraph = new BarChart();
        subject.addObserver(barGraph);
        /*
        * We create an Observer that displays a pie graph
        */
        PieChart pieGraph = new PieChart();
        subject.addObserver(pieGraph);
  }
```


BarChart
+ void BarChart()
- DefaultCategoryDataset createDataset()
- JFreeChart createChart()
+ void notifyAChanged(Integer newValue)
+ void notifyBChanged(Integer newValue)
+ void notifyCChanged(Integer newValue)
SubjectObserver
+ void notifyAChanged(Integer newValue)
+ void notifyBChanged(Integer newValue)
+ void notifyCChanged(Integer newValue)
PieChart
+ void PieChart()
- PieDataset createDataset()
- JFreeChart createChart()
+ void notifyAChanged(Integer newValue)
+ void notifyBChanged(Integer newValue)
+ void notifyCChanged(Integer newValue)
Subject
+ boolean addObserver(SubjectObserver observer)
+ void setA(Integer a)
+ Integer getA()
+ void setB(Integer b)
+ Integer getB()
+ void setC(Integer c)
+ Integer getC()
SubjectController
+ void SubjectController(Subject subject)

```
public class Subject {

	private List<SubjectObserver> observers =
							new ArrayList<SubjectObserver>();
	/**
	* This will add an observer to this Subject
	*
	* @param observer
	* @return
	*/
	public boolean addObserver( SubjectObserver observer){
		return observers.add(observer);
	}
```

```
public class Subject {

        private Integer A = 4;
        public void setA(Integer a) {
                A = a;
                for (SubjectObserver observer : observers)
                    observer.notifyAChanged(a);
        }
        public Integer getA() {
                return A;
        }
```

```
public interface SubjectObserver {

        /**
        * This method is called when the value of A changes
        *
        * @param newValue
        */
        public void notifyAChanged(Integer newValue);
public class BarChart  implements SubjectObserver{

        ...

        @Override
        public void notifyAChanged(Integer newValue) {

                dataset.setValue(newValue, rowA, "");
        }
```

In class exercise

Think of 1 application of the Observer design pattern that we have not talked about.

- Identify the subject being observed
- Identify the observers
- Describe some notifications that would be sent