CMPE 202

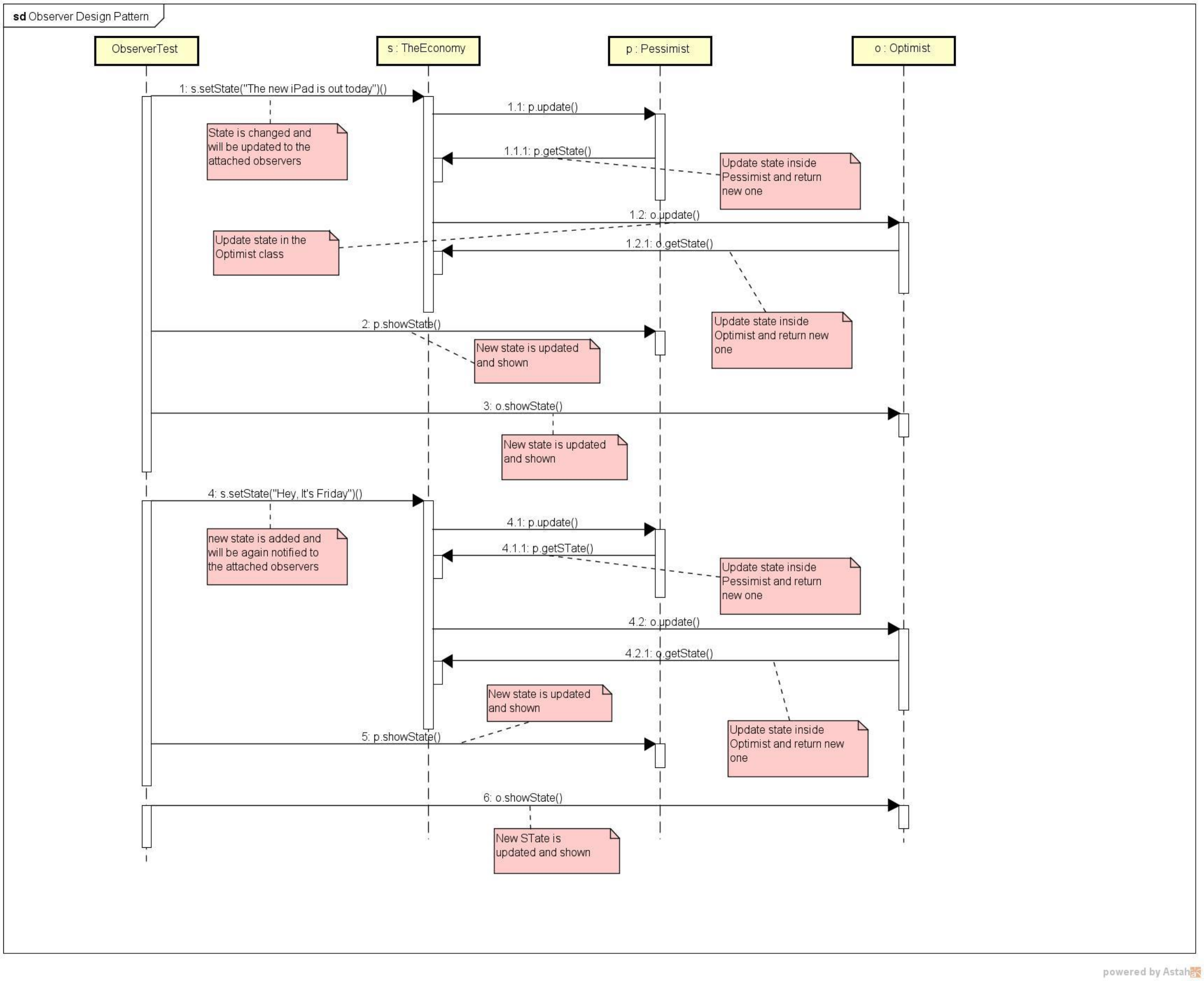
LAB 7

Observer Pattern Sequence Diagram

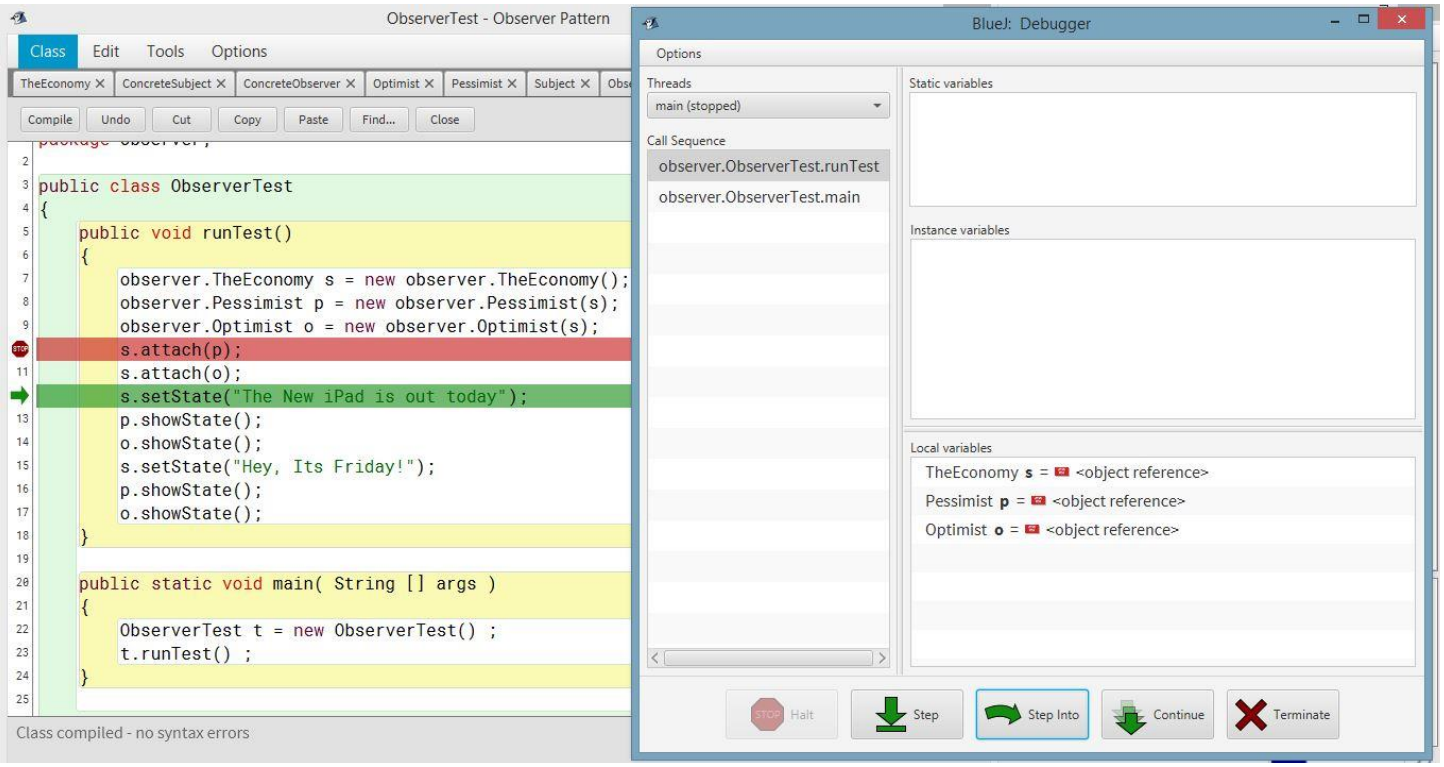
Name: Pavan Kothawade

SJSU ID: 012169690

Sequence Diagram


sd Observer Design Pattern
ObserverTest
s : TheEconomy
p : Pessimist
o : Optimist
1: s.setState("The new iPad is out today")()
State is changed and will be updated to the attached observers
1.1: p.update()
1.1.1: p.getState()
Update state inside Pessimist and return new one
1.2: o.update()
Update state in the Optimist class
1.2.1: o.getState()
Update state inside Optimist and return new one
2: p.showState()
New state is updated and shown
3: o.showState()
New state is updated and shown
4: s.setState("Hey, It's Friday")()
new state is added and will be again notified to the attached observers
4.1: p.update()
4.1.1: p.getSTate()
Update state inside Pessimist and return new one
4.2: o.update()
4.2.1: o.getState()
New state is updated and shown
Update state inside Optimist and return new one
5: p.showState()
6: o.showState()
New STate is updated and shown
powered by Astah

1: s.setState(“The new iPad is out today”)- Set new state to TheEconomy class



```
package observer;

public class ObserverTest
{
    public void runTest()
    {
        observer.TheEconomy s = new observer.TheEconomy();
        observer.Pessimist p = new observer.Pessimist(s);
        observer.Optimist o = new observer.Optimist(s);
        s.attach(p);
        s.attach(o);
        s.setState("The New iPad is out today");
        p.showState();
        o.showState();
        s.setState("Hey, Its Friday!");
        p.showState();
        o.showState();
    }

    public static void main( String [] args )
    {
        ObserverTest t = new ObserverTest() ;
        t.runTest() ;
    }

```

Class compiled - no syntax errors

Threads: main (stopped)

Call Sequence
- observer.ObserverTest.runTest
- observer.ObserverTest.main

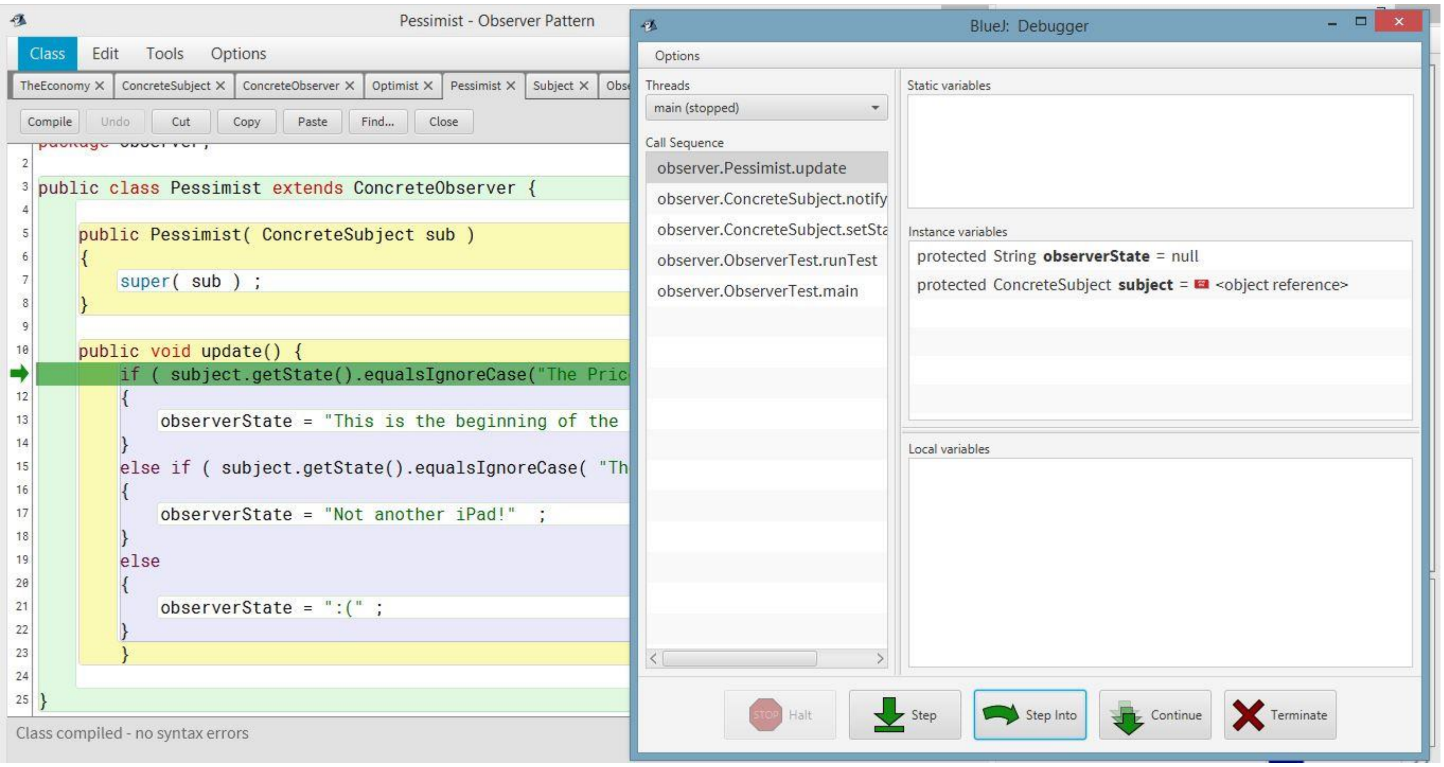
Static variables

Instance variables

Local variables

```
TheEconomy s = <object reference>
Pessimist p = <object reference>
Optimist o = <object reference>
```

1.1 p.update()- Update state in pessimist class



```
package observer;

public class Pessimist extends ConcreteObserver {

    public Pessimist( ConcreteSubject sub )
    {
        super( sub ) ;
    }

    public void update() {
        if ( subject.getState().equalsIgnoreCase("The Pric
        {
            observerState = "This is the beginning of the
        }
        else if ( subject.getState().equalsIgnoreCase( "Th
        {
            observerState = "Not another iPad!"  ;
        }
        else
        {
            observerState = ":(" ;
        }
        }

}
```

Class compiled - no syntax errors

Threads: main (stopped)

Call Sequence
- observer.Pessimist.update
- observer.ConcreteSubject.notify
- observer.ConcreteSubject.setSta
- observer.ObserverTest.runTest
- observer.ObserverTest.main

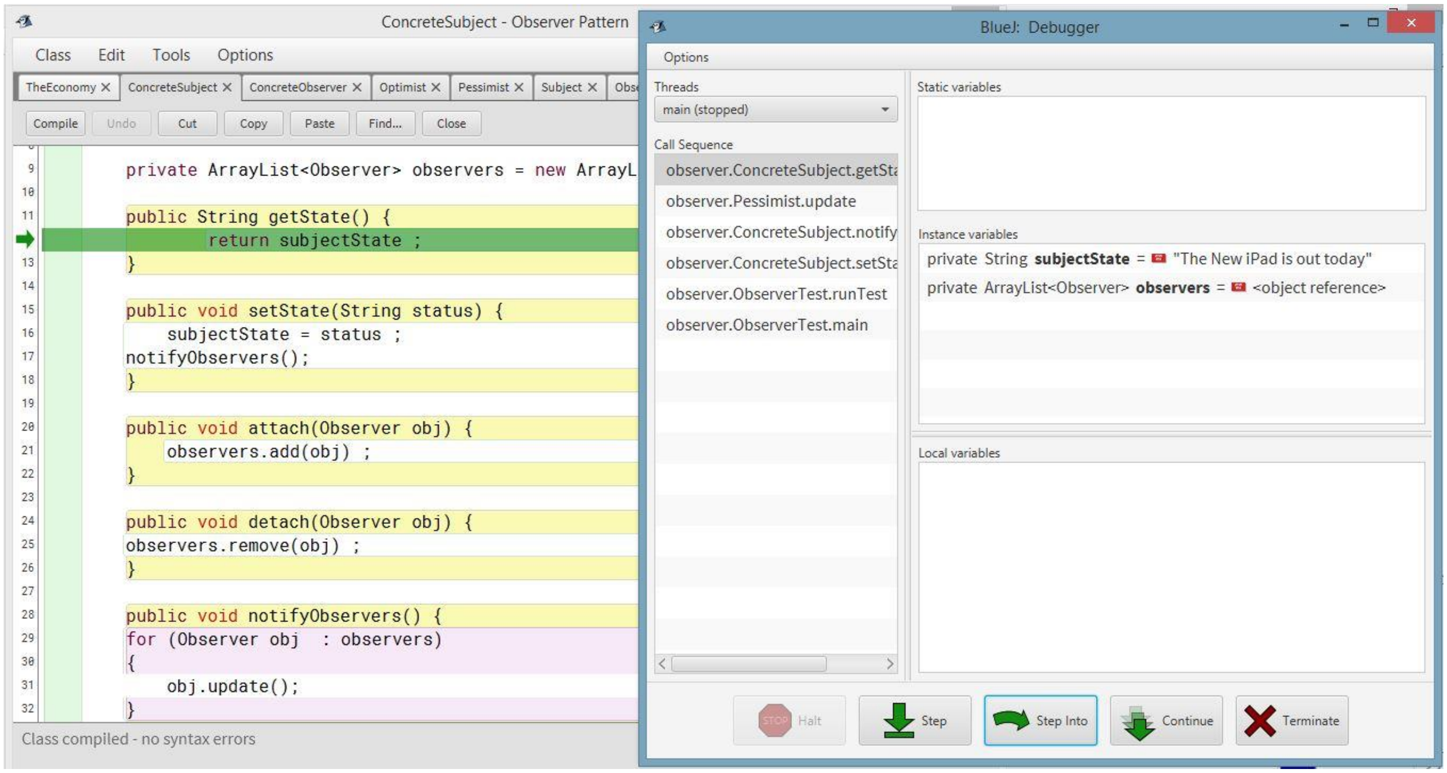
Static variables

Instance variables

```
protected String observerState = null
protected ConcreteSubject subject = <object reference>
```

Local variables

1.1.1 p.getState()-Retrieve new state of Pessimist class



```
    private ArrayList<Observer> observers = new ArrayL

    public String getState() {
        return subjectState ;
    }

    public void setState(String status) {
      subjectState = status ;
    notifyObservers();
    }

    public void attach(Observer obj) {
      observers.add(obj) ;
    }

    public void detach(Observer obj) {
    observers.remove(obj) ;
    }

    public void notifyObservers() {
    for (Observer obj  : observers)
    {
      obj.update();
    }
```

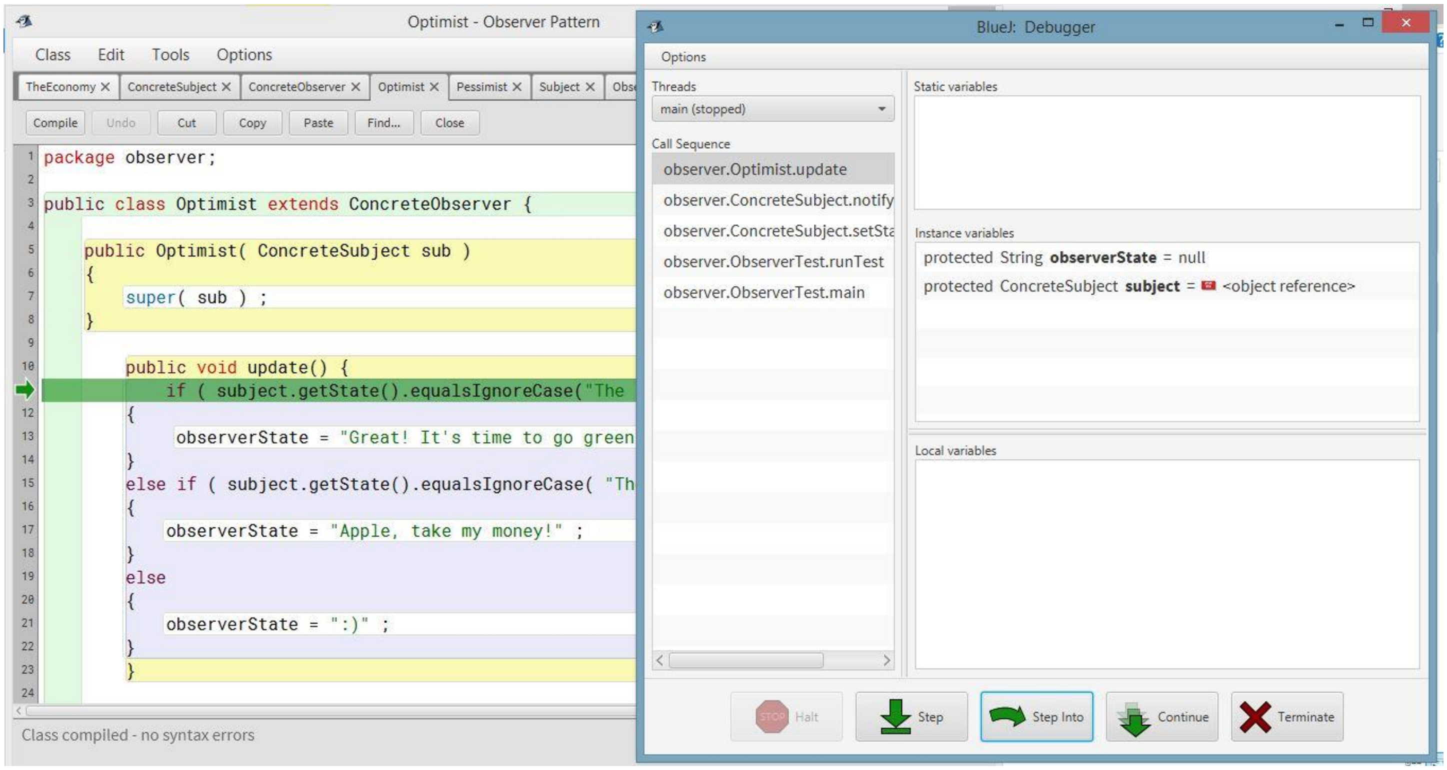
Class compiled - no syntax errors

main (stopped)

observer.ConcreteSubject.getSta
observer.Pessimist.update
observer.ConcreteSubject.notify
observer.ConcreteSubject.setSta
observer.ObserverTest.runTest
observer.ObserverTest.main

private String **subjectState** = "The New iPad is out today"
private ArrayList<Observer> **observers** = <object reference>

1.2 o.update()- Update state in the Optimist class



```
package observer;

public class Optimist extends ConcreteObserver {

  public Optimist( ConcreteSubject sub )
  {
    super( sub ) ;
  }

    public void update() {
      if ( subject.getState().equalsIgnoreCase("The
    {
      observerState = "Great! It's time to go green
    }
    else if ( subject.getState().equalsIgnoreCase( "Th
    {
      observerState = "Apple, take my money!" ;
    }
    else
    {
      observerState = ":)" ;
    }
    }
```

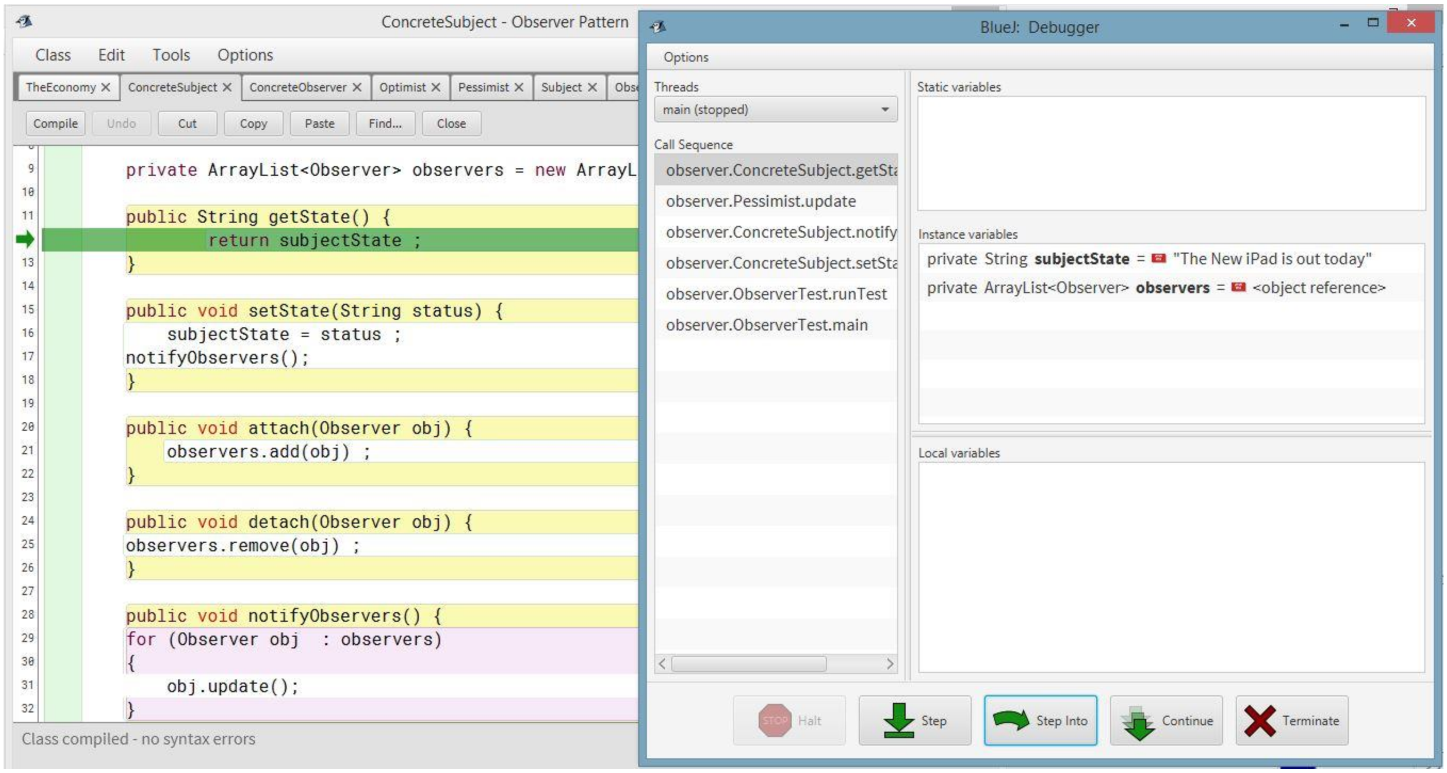
Class compiled - no syntax errors

main (stopped)

observer.Optimist.update
observer.ConcreteSubject.notify
observer.ConcreteSubject.setSta
observer.ObserverTest.runTest
observer.ObserverTest.main

protected String **observerState** = null
protected ConcreteSubject **subject** = <object reference>

1.2.1 o.getState()- Retrieve new state of Optimist class



```
    private ArrayList<Observer> observers = new ArrayL

    public String getState() {
            return subjectState ;
    }

    public void setState(String status) {
        subjectState = status ;
    notifyObservers();
    }

    public void attach(Observer obj) {
        observers.add(obj) ;
    }

    public void detach(Observer obj) {
    observers.remove(obj) ;
    }

    public void notifyObservers() {
    for (Observer obj  : observers)
    {
        obj.update();
    }
```

Class compiled - no syntax errors

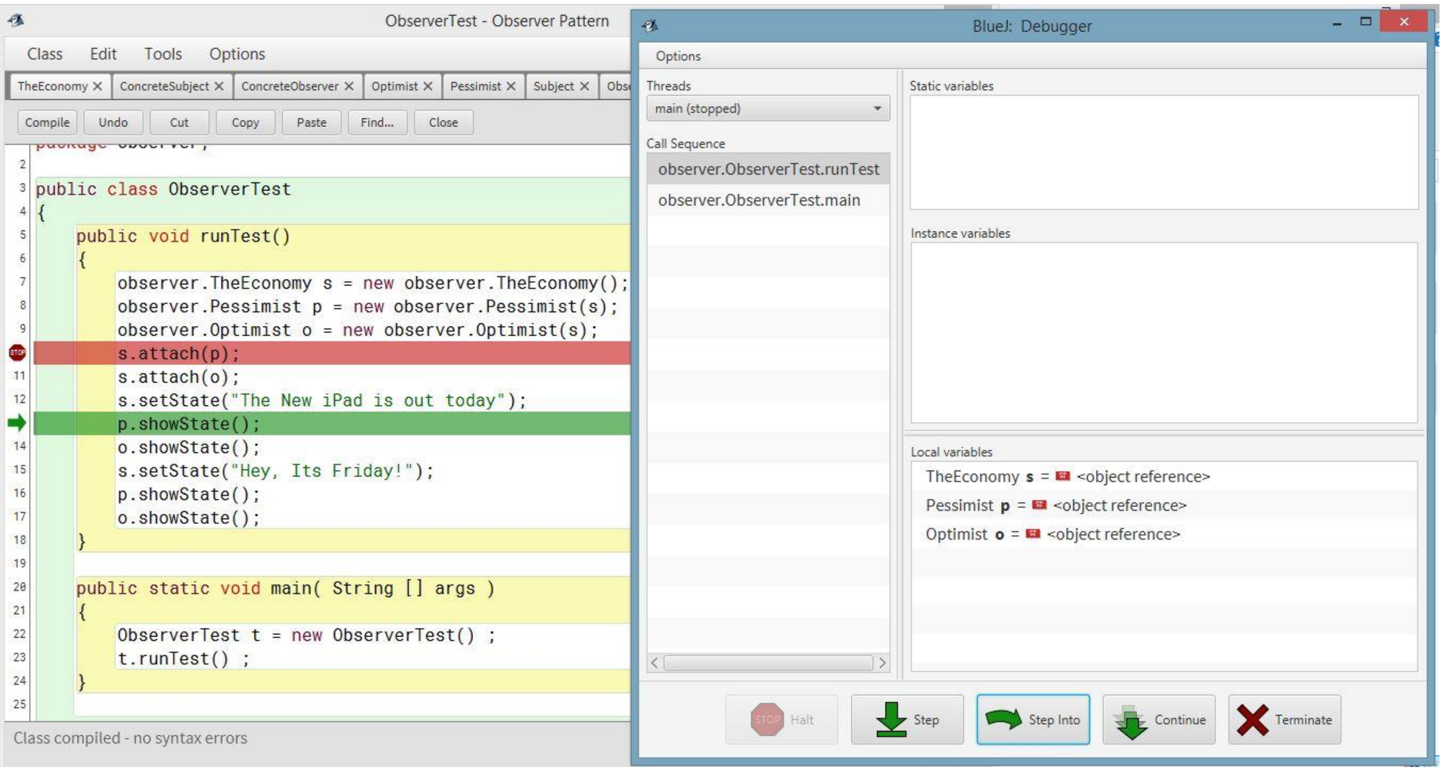
Call Sequence:

observer.ConcreteSubject.getSta
observer.Pessimist.update
observer.ConcreteSubject.notify
observer.ConcreteSubject.setSta
observer.ObserverTest.runTest
observer.ObserverTest.main

Instance variables:

private String **subjectState** = "The New iPad is out today"
private ArrayList<Observer> **observers** = <object reference>

2. p.showState()- Display the current state of Pessimist class



```
package observer;

public class ObserverTest
{
    public void runTest()
    {
        observer.TheEconomy s = new observer.TheEconomy();
        observer.Pessimist p = new observer.Pessimist(s);
        observer.Optimist o = new observer.Optimist(s);
        s.attach(p);
        s.attach(o);
        s.setState("The New iPad is out today");
        p.showState();
        o.showState();
        s.setState("Hey, Its Friday!");
        p.showState();
        o.showState();
    }

    public static void main( String [] args )
    {
        ObserverTest t = new ObserverTest() ;
        t.runTest() ;
    }
```

Class compiled - no syntax errors

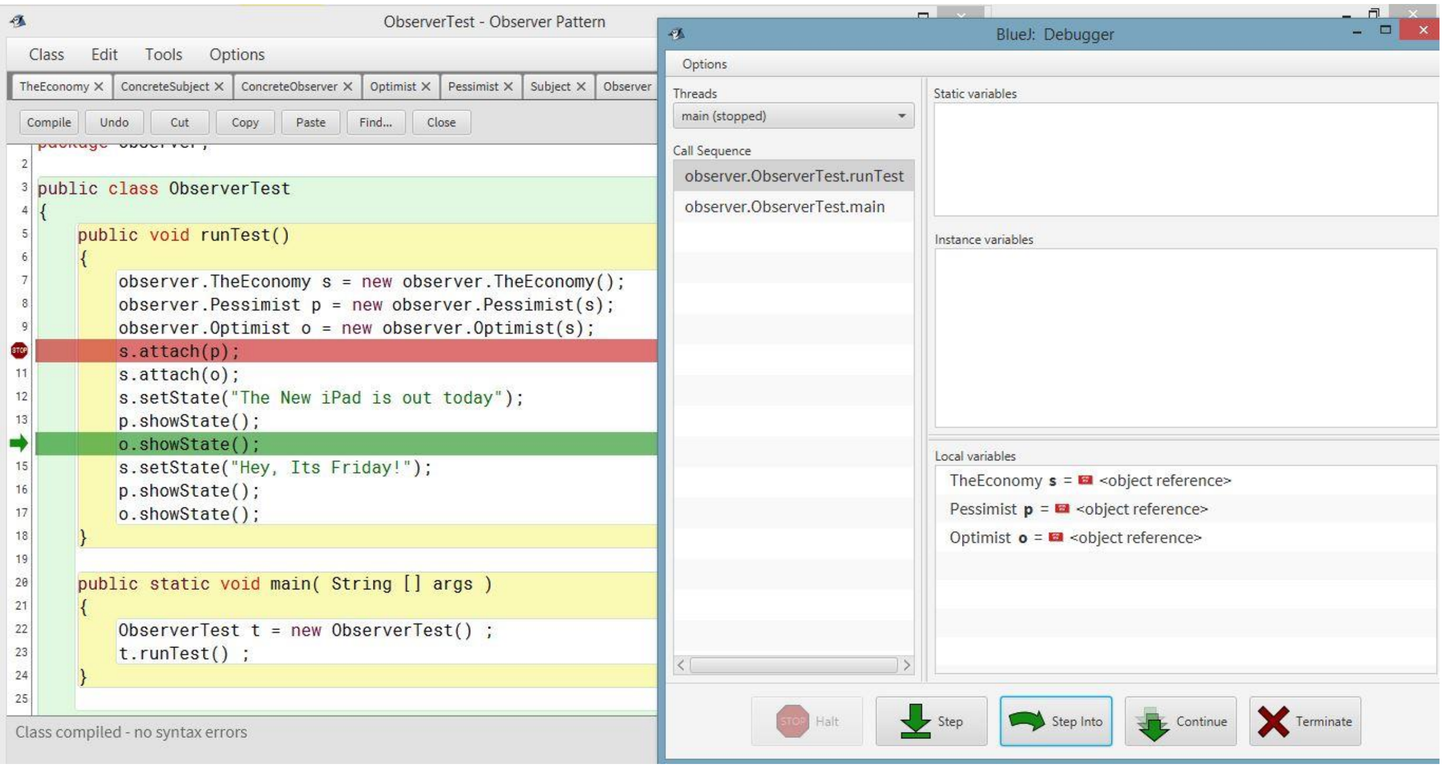
Call Sequence:

observer.ObserverTest.runTest
observer.ObserverTest.main

Local variables:

TheEconomy **s** = <object reference>
Pessimist **p** = <object reference>
Optimist **o** = <object reference>

3. o.showState()-Display the current state of Optimist class



```
package observer;

public class ObserverTest
{
    public void runTest()
    {
        observer.TheEconomy s = new observer.TheEconomy();
        observer.Pessimist p = new observer.Pessimist(s);
        observer.Optimist o = new observer.Optimist(s);
        s.attach(p);
        s.attach(o);
        s.setState("The New iPad is out today");
        p.showState();
        o.showState();
        s.setState("Hey, Its Friday!");
        p.showState();
        o.showState();
    }

    public static void main( String [] args )
    {
        ObserverTest t = new ObserverTest() ;
        t.runTest() ;
    }
```

Class compiled - no syntax errors

Threads: main (stopped)

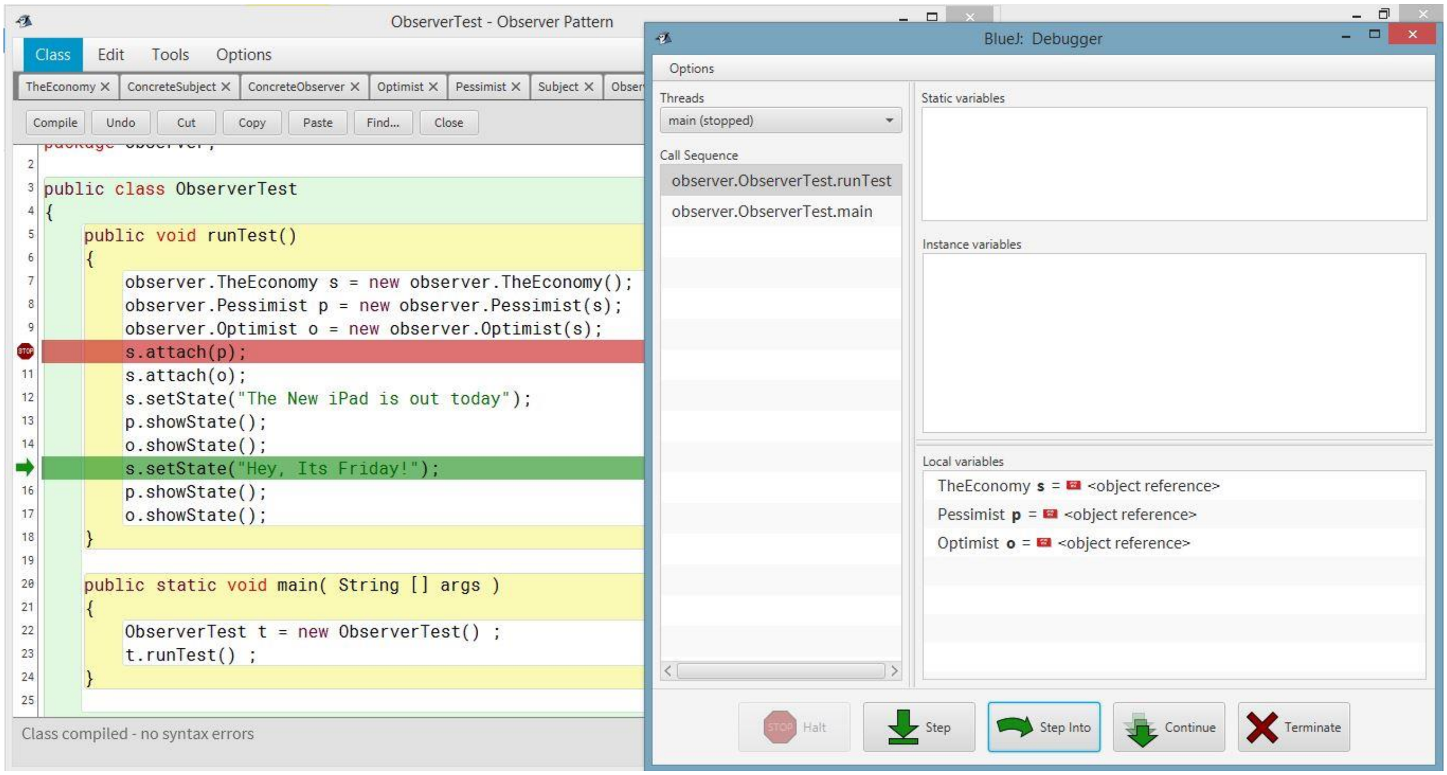
Call Sequence:
observer.ObserverTest.runTest
observer.ObserverTest.main

Static variables

Instance variables

Local variables:
TheEconomy **s** = <object reference>
Pessimist **p** = <object reference>
Optimist **o** = <object reference>

4. s.setState()- Set new state in TheEconomy class



```
package observer;

public class ObserverTest
{
    public void runTest()
    {
        observer.TheEconomy s = new observer.TheEconomy();
        observer.Pessimist p = new observer.Pessimist(s);
        observer.Optimist o = new observer.Optimist(s);
        s.attach(p);
        s.attach(o);
        s.setState("The New iPad is out today");
        p.showState();
        o.showState();
        s.setState("Hey, Its Friday!");
        p.showState();
        o.showState();
    }

    public static void main( String [] args )
    {
        ObserverTest t = new ObserverTest() ;
        t.runTest() ;
    }
```

Class compiled - no syntax errors

Threads: main (stopped)

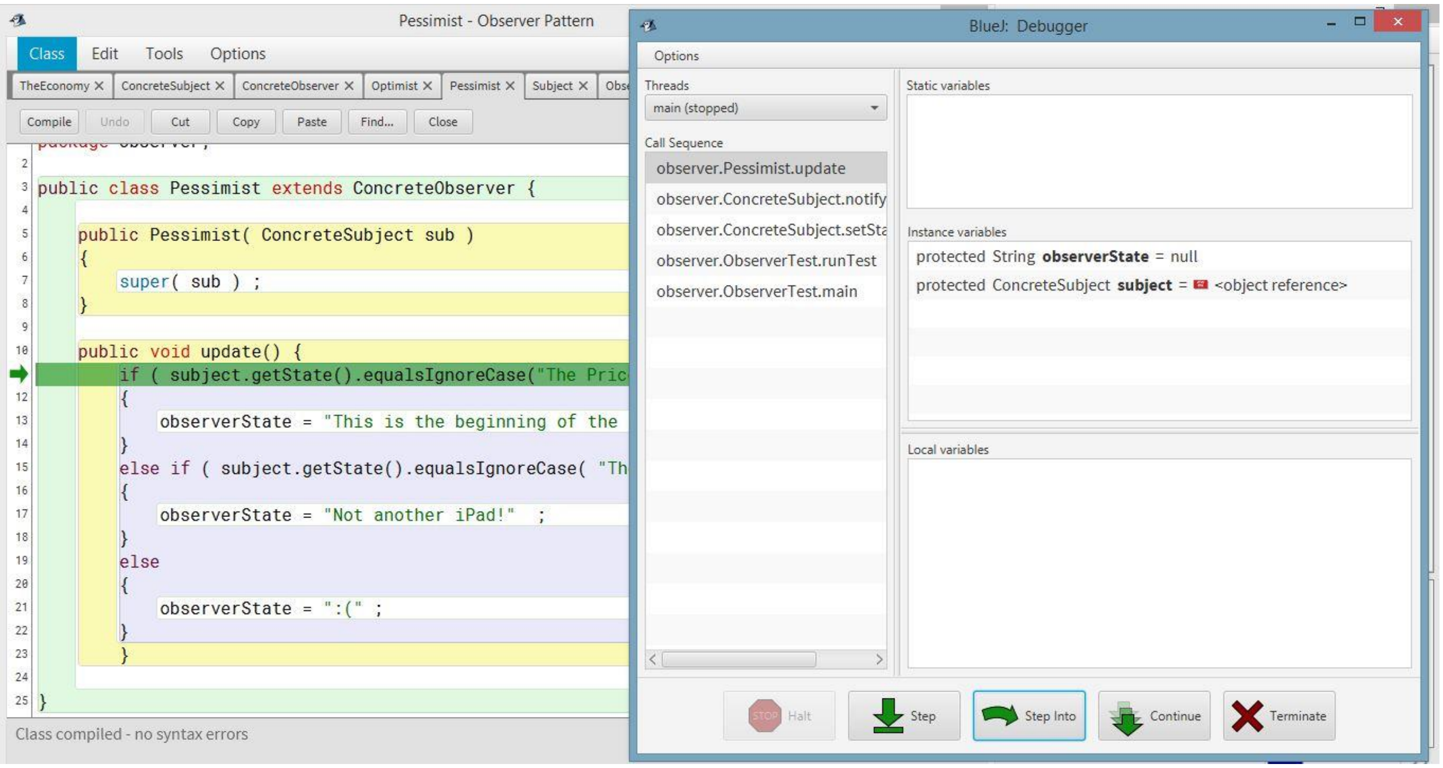
Call Sequence:
observer.ObserverTest.runTest
observer.ObserverTest.main

Static variables

Instance variables

Local variables:
TheEconomy **s** = <object reference>
Pessimist **p** = <object reference>
Optimist **o** = <object reference>

4.1 p.update()- Update state change in Pessimist class



```
package observer;

public class Pessimist extends ConcreteObserver {

    public Pessimist( ConcreteSubject sub )
    {
        super( sub ) ;
    }

    public void update() {
        if ( subject.getState().equalsIgnoreCase("The Pric
        {
            observerState = "This is the beginning of the
        }
        else if ( subject.getState().equalsIgnoreCase( "Th
        {
            observerState = "Not another iPad!"  ;
        }
        else
        {
            observerState = ":(" ;
        }
    }

}
```

Class compiled - no syntax errors

Threads: main (stopped)

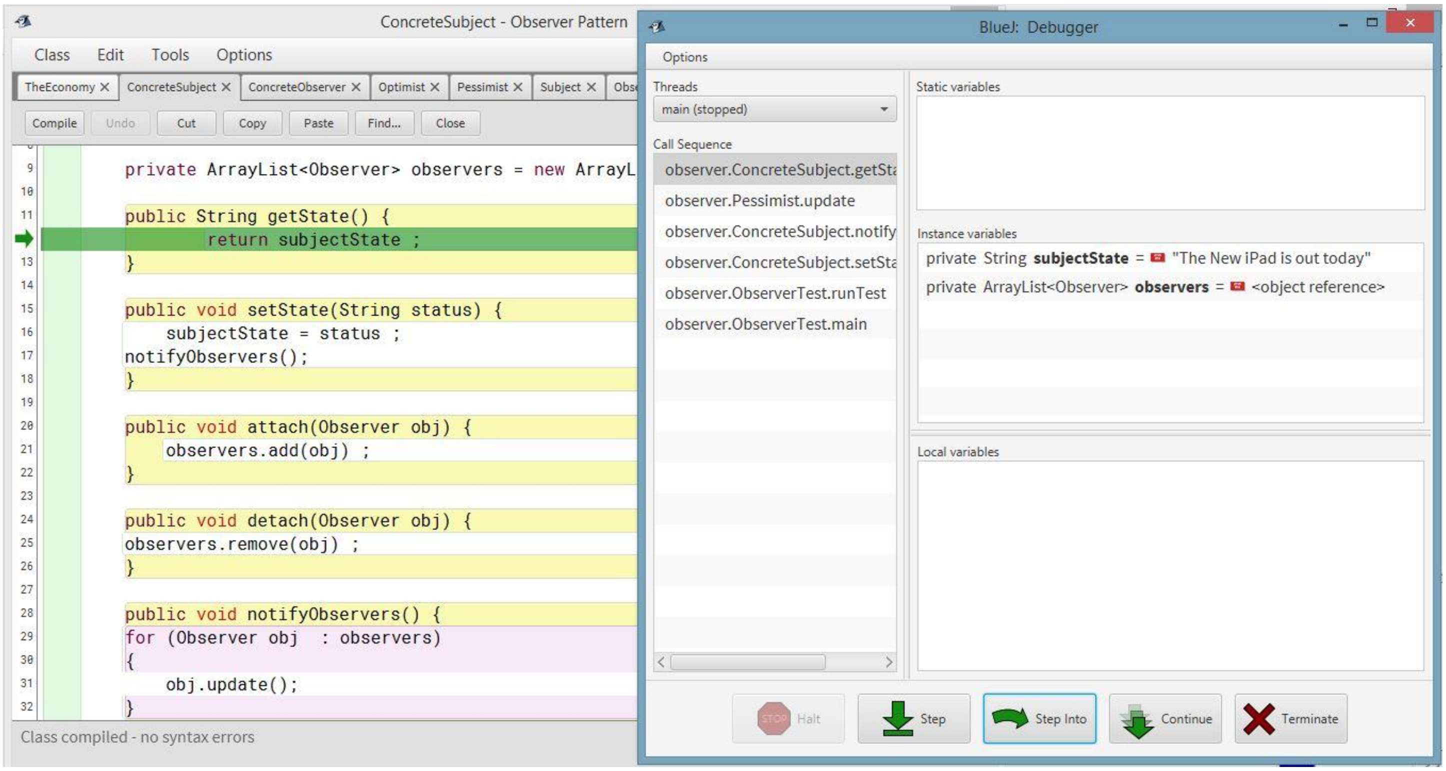
Call Sequence:

```
observer.Pessimist.update
observer.ConcreteSubject.notify
observer.ConcreteSubject.setSta
observer.ObserverTest.runTest
observer.ObserverTest.main
```

Instance variables:

```
protected String observerState = null
protected ConcreteSubject subject = <object reference>
```

4.1.1 p.getState()- Retrieving Pessimists class state



```
    private ArrayList<Observer> observers = new ArrayL

    public String getState() {
            return subjectState ;
    }

    public void setState(String status) {
        subjectState = status ;
    notifyObservers();
    }

    public void attach(Observer obj) {
        observers.add(obj) ;
    }

    public void detach(Observer obj) {
    observers.remove(obj) ;
    }

    public void notifyObservers() {
    for (Observer obj  : observers)
    {
        obj.update();
    }
```

Class compiled - no syntax errors

Threads: main (stopped)

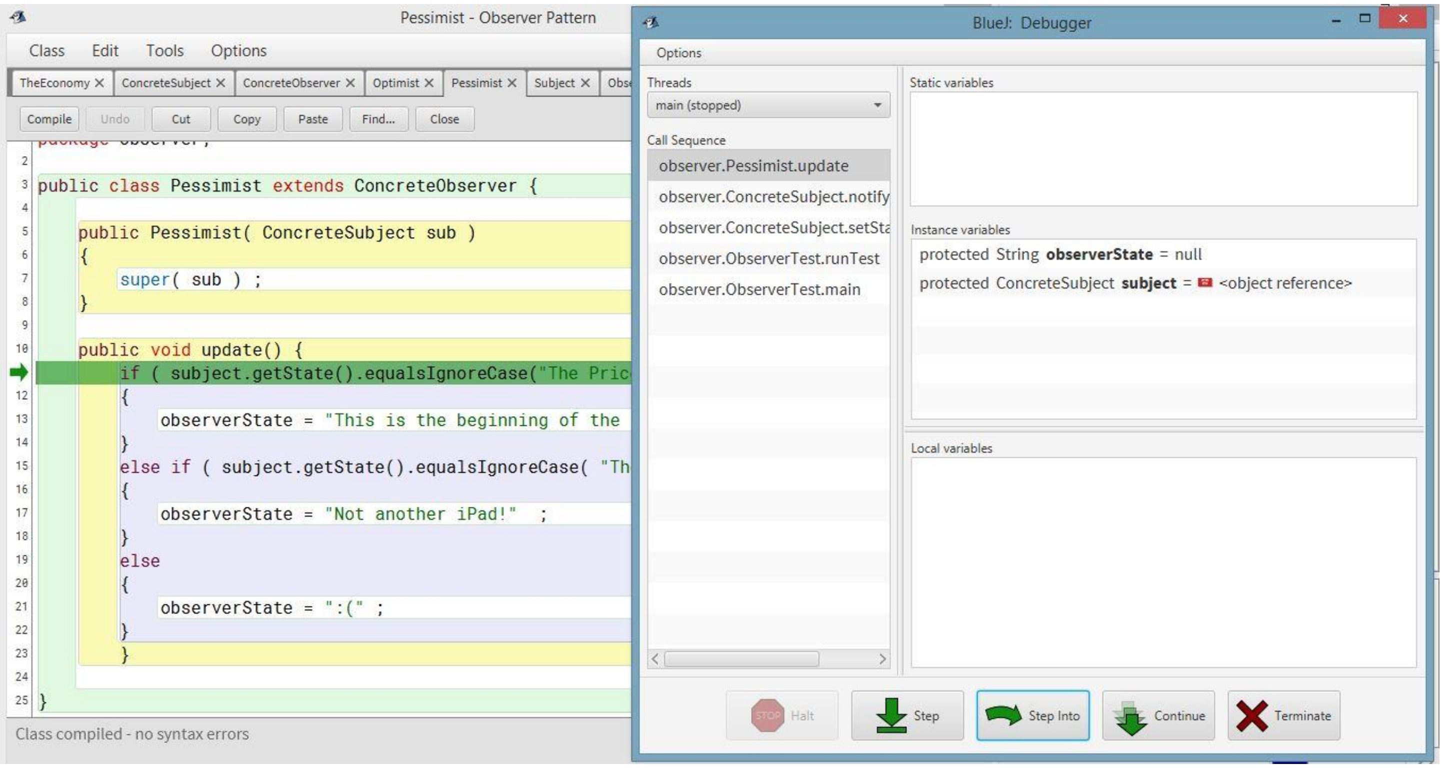
Call Sequence:

```
observer.ConcreteSubject.getSta
observer.Pessimist.update
observer.ConcreteSubject.notify
observer.ConcreteSubject.setSta
observer.ObserverTest.runTest
observer.ObserverTest.main
```

Instance variables:

```
private String subjectState = "The New iPad is out today"
private ArrayList<Observer> observers = <object reference>
```

4.2 o.update()- Updating state change in Optimist class



```
package observer;

public class Pessimist extends ConcreteObserver {

    public Pessimist( ConcreteSubject sub )
    {
        super( sub ) ;
    }

    public void update() {
        if ( subject.getState().equalsIgnoreCase("The Pric
        {
            observerState = "This is the beginning of the
        }
        else if ( subject.getState().equalsIgnoreCase( "Th
        {
            observerState = "Not another iPad!"  ;
        }
        else
        {
            observerState = ":(" ;
        }
        }

}
```

Class compiled - no syntax errors

Call Sequence:

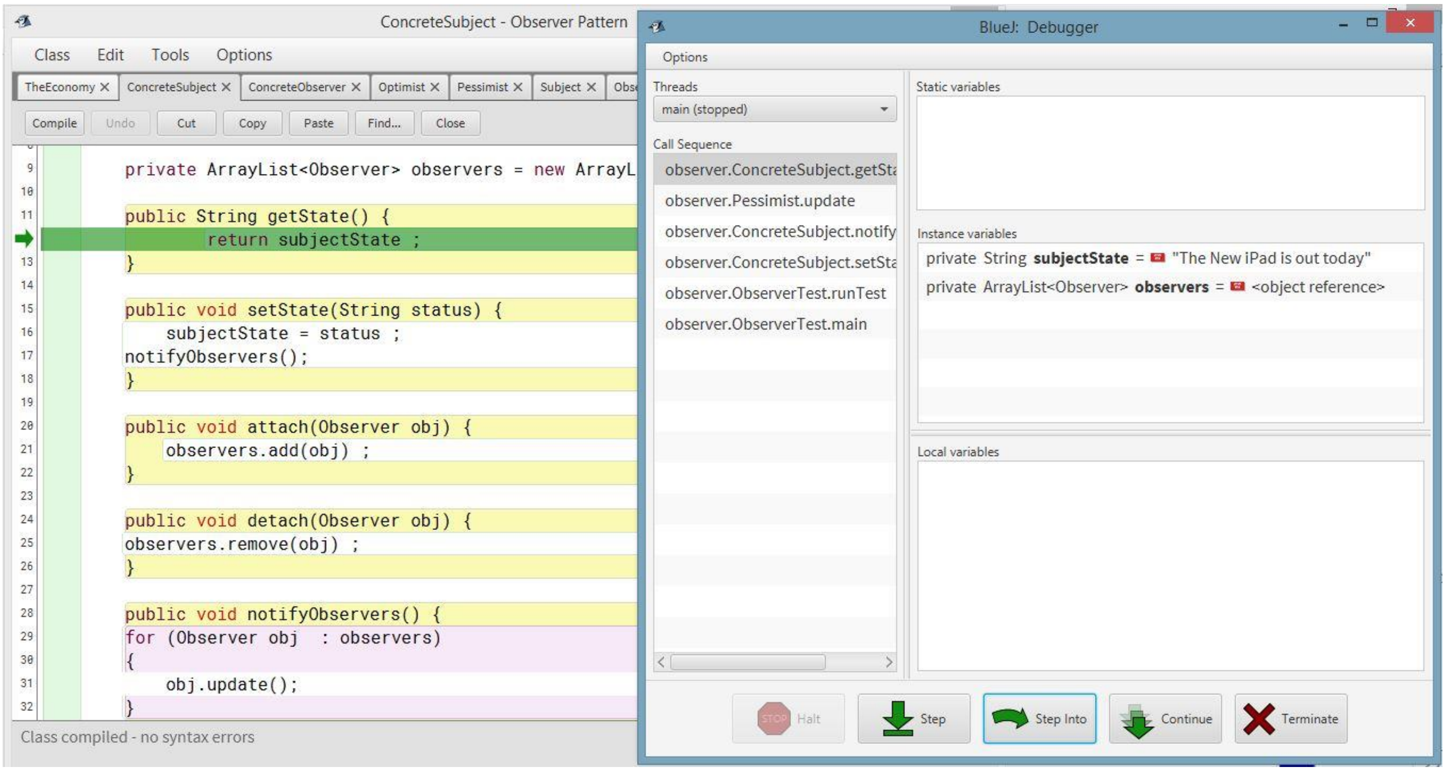
- observer.Pessimist.update
- observer.ConcreteSubject.notify
- observer.ConcreteSubject.setSta
- observer.ObserverTest.runTest
- observer.ObserverTest.main

Instance variables:

protected String **observerState** = null

protected ConcreteSubject **subject** = <object reference>

4.2.1 o.getState()- Retrieving Optimist class state



```
    private ArrayList<Observer> observers = new ArrayL

    public String getState() {
            return subjectState ;
    }

    public void setState(String status) {
        subjectState = status ;
    notifyObservers();
    }

    public void attach(Observer obj) {
        observers.add(obj) ;
    }

    public void detach(Observer obj) {
    observers.remove(obj) ;
    }

    public void notifyObservers() {
    for (Observer obj  : observers)
    {
        obj.update();
    }
```

Class compiled - no syntax errors

Call Sequence:

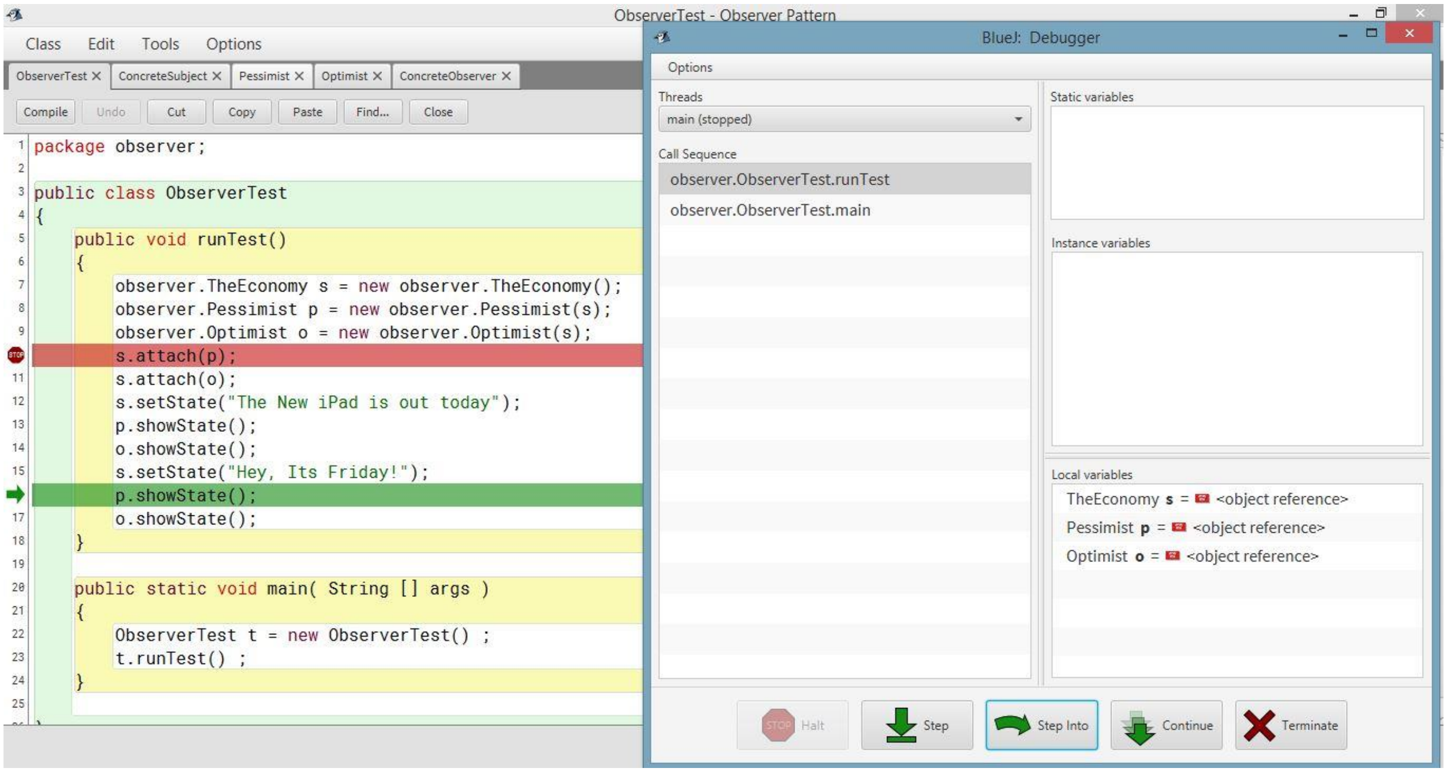
- observer.ConcreteSubject.getSta
- observer.Pessimist.update
- observer.ConcreteSubject.notify
- observer.ConcreteSubject.setSta
- observer.ObserverTest.runTest
- observer.ObserverTest.main

Instance variables:

private String **subjectState** = "The New iPad is out today"

private ArrayList<Observer> **observers** = <object reference>

5. p.showState()- Show current state of Pessimist class



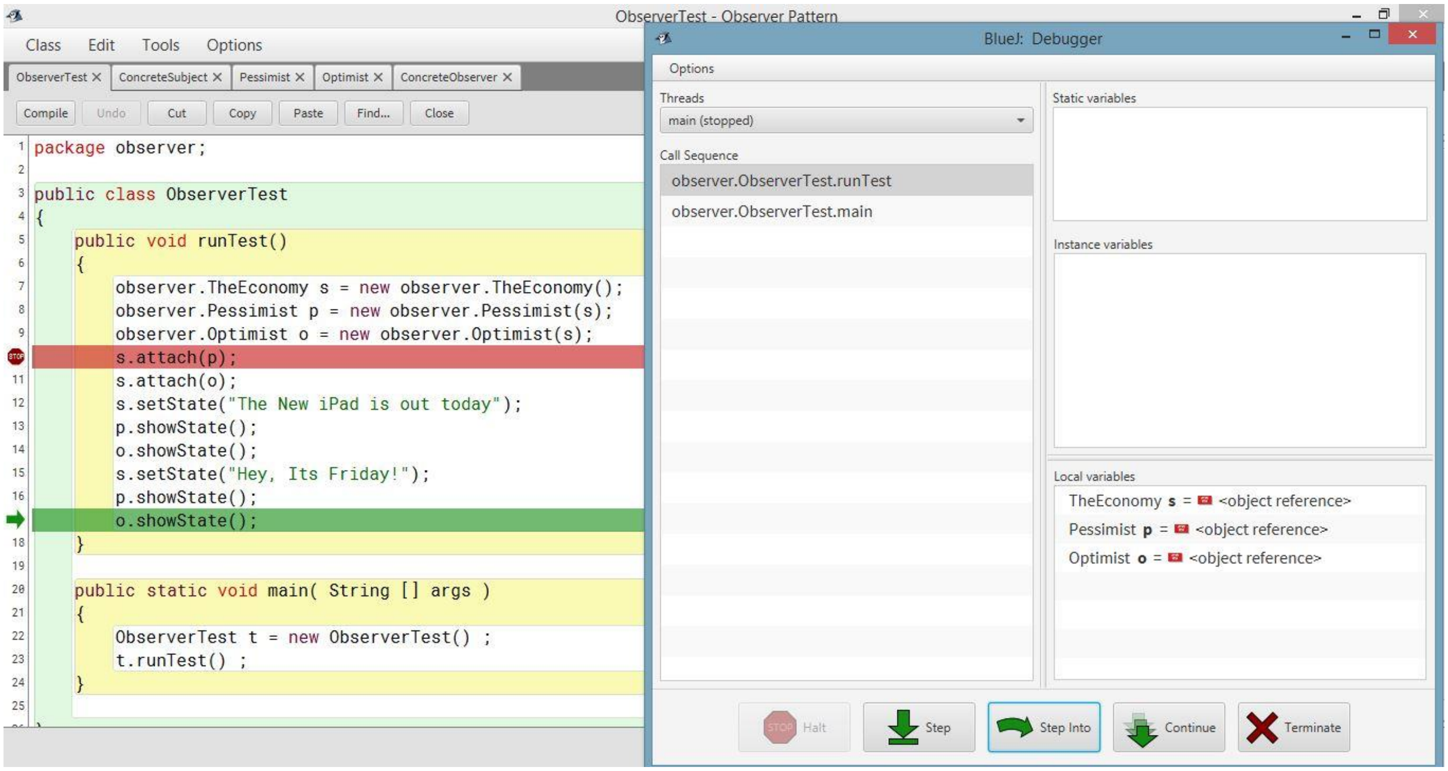
```
package observer;

public class ObserverTest
{
    public void runTest()
    {
        observer.TheEconomy s = new observer.TheEconomy();
        observer.Pessimist p = new observer.Pessimist(s);
        observer.Optimist o = new observer.Optimist(s);
        s.attach(p);
        s.attach(o);
        s.setState("The New iPad is out today");
        p.showState();
        o.showState();
        s.setState("Hey, Its Friday!");
        p.showState();
        o.showState();
    }

    public static void main( String [] args )
    {
        ObserverTest t = new ObserverTest() ;
        t.runTest() ;
    }

```

Threads: main (stopped)

Call Sequence:
observer.ObserverTest.runTest
observer.ObserverTest.main

Local variables:
TheEconomy **s** = <object reference>
Pessimist **p** = <object reference>
Optimist **o** = <object reference>

6. o.showState()- Show current state of Optimist class



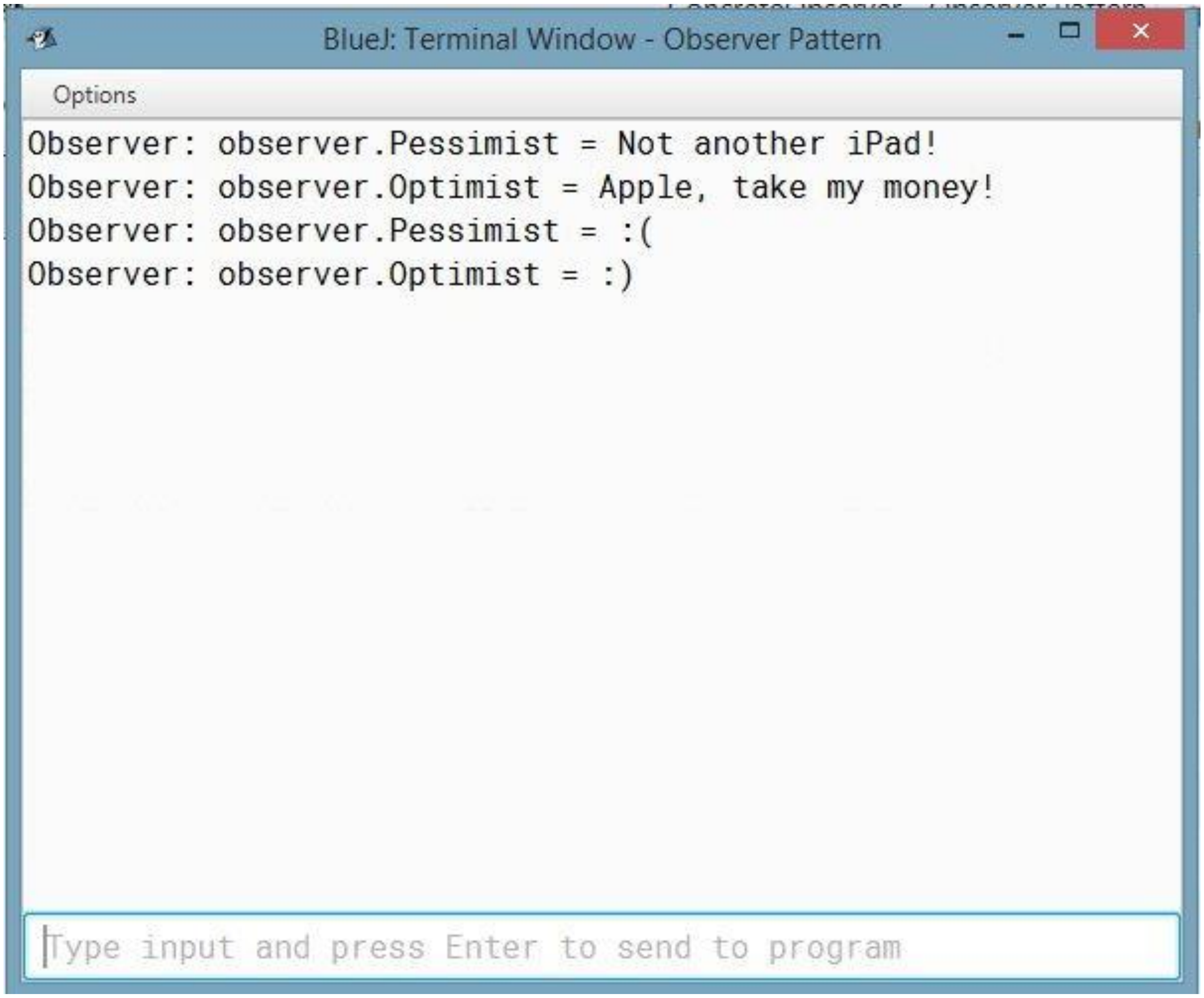
```
package observer;

public class ObserverTest
{
    public void runTest()
    {
        observer.TheEconomy s = new observer.TheEconomy();
        observer.Pessimist p = new observer.Pessimist(s);
        observer.Optimist o = new observer.Optimist(s);
        s.attach(p);
        s.attach(o);
        s.setState("The New iPad is out today");
        p.showState();
        o.showState();
        s.setState("Hey, Its Friday!");
        p.showState();
        o.showState();
    }

    public static void main( String [] args )
    {
        ObserverTest t = new ObserverTest() ;
        t.runTest() ;
    }

```

Threads: main (stopped)

Call Sequence:
observer.ObserverTest.runTest
observer.ObserverTest.main

Local variables:
TheEconomy **s** = <object reference>
Pessimist **p** = <object reference>
Optimist **o** = <object reference>

OUTPUT:


BlueJ: Terminal Window - Observer Pattern
Options
Type input and press Enter to send to program

```
Observer: observer.Pessimist = Not another iPad!
Observer: observer.Optimist = Apple, take my money!
Observer: observer.Pessimist = :(
Observer: observer.Optimist = :)
```