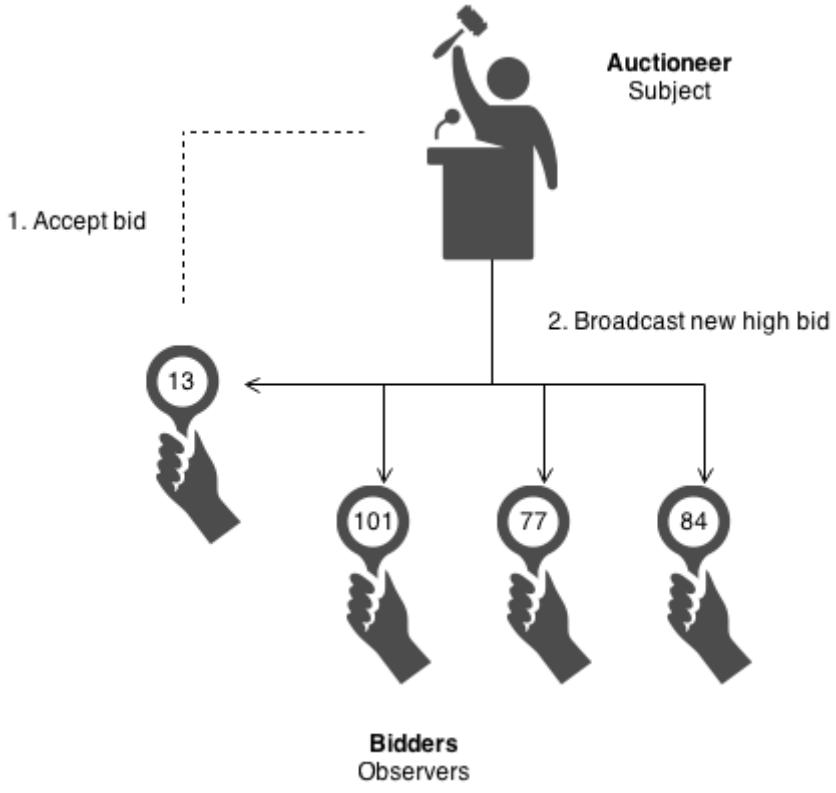
DESIGN PATTERNS: **OBSERVER**

Behavioral Pattern

1. Design Pattern Description

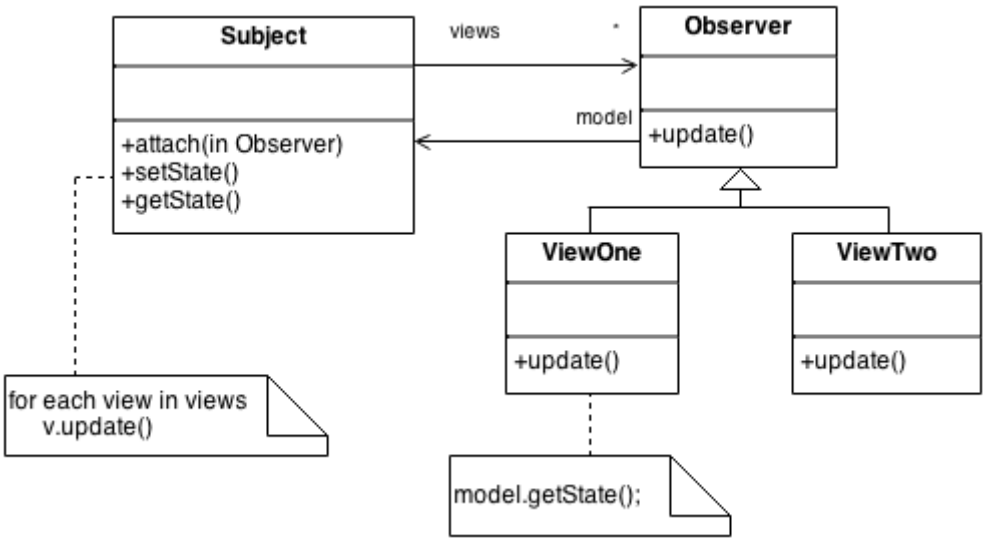
Intent

- Define a **one-to-many dependency** between objects so that when one object changes state, all its dependents are notified and updated automatically.
- Encapsulate the core (or common or engine) components in a Subject abstraction, and the variable (or optional or user interface) components in an Observer hierarchy.
- The "View" part of Model-**View**-Controller.



Problem

A large monolithic design does not scale well as new graphing or monitoring requirements are levied.



2. Design Pattern Example

```
abstract class Observer {
    protected Subject subject;
    public abstract void update();
}

class Subject {
    private List<Observer> observers = new ArrayList<>();
    private int state;

    public void add(Observer o) {
        observers.add(o);
    }

    public int getState() {
        return state;
    }

    public void setState(int value) {
        this.state = value;
        execute();
    }

    private void execute() {
        for (Observer observer : observers) {
            observer.print();
        }
    }
}

class HexObserver extends Observer {
    public HexObserver(Subject subject) {
        this.subject = subject;
        this.subject.add(this);
    }

    public void print() {
        System.out.print(" " + Integer.toHexString(subject.getState()));
```

```
    }
}

class OctObserver extends Observer {
    public OctObserver(Subject subject) {
        this.subject = subject;
        this.subject.add( this );
    }

    public void update() {
        System.out.print(" " + Integer.toOctalString(subject.getState()));
    }
}

class BinObserver extends Observer {
    public BinObserver(Subject subject) {
        this.subject = subject;
        this.subject.add(this);
    }

    public void update() {
        System.out.print(" " + Integer.toBinaryString(subject.getState()));
    }
}

public class ObserverDemo {
    public static void main( String[] args ) {
        Subject sub = new Subject();
        // Client configures the number and type of Observers
        new HexObserver(sub);
        new OctObserver(sub);
        new BinObserver(sub);
        Scanner scan = new Scanner(System.in);
        for (int i = 0; i < 5; i++) {
            System.out.print("\nEnter a number: ");
            sub.setState(scan.nextInt());
        }
    }
}
```

Output

```
Enter a number: 55
 37 67 110111
Enter a number: 12
 c 14 1100
Enter a number: -10
 fffffff6 37777777766 11111111111111111111111111110110
Enter a number: 112
 70 160 1110000
Enter a number: 5
 5 5 101
```

3. Existing Example in SwanLab

Nope.

4. Design Proposal in SwanLab

Example 1:

Monitoring. Instead having hard-coded concrete observers:

```
methods (Access = public)

    function solveProblem(obj)
        obj.cost.computeCostAndGradient();
        obj.constraint.computeCostAndGradient();
        obj.printOptimizerVariable();

        obj.hasFinished = false;

        while ~obj.hasFinished
            obj.increaseIter();
            obj.update();
            obj.updateStatus();
            obj.refreshMonitoring();
            obj.printOptimizerVariable();
            obj.printHistory();
        end
        obj.hasConverged = 0;
    end

end
```

They could be attached externally and simply add a method "notifyObservers()".

```
methods (Access = public)

    function solveProblem(obj)
        obj.cost.computeCostAndGradient();
        obj.constraint.computeCostAndGradient();

        obj.hasFinished = false;

        while ~obj.hasFinished
            obj.increaseIter();
            obj.update();
            obj.updateStatus();
            obj.notifyObservers();
        end
        obj.hasConverged = 0;
    end

    function attachObserver(obj,observer)
        obj.observers(end+1) = observer;
    end

end

methods (Access = private)

    function notifyObservsers(obj)
        for iObs = 1:nObservers
            obj.observers(iObs).update();
        end
    end

end
```

Example 2:

This design pattern could be also applied to print unfitted meshes for Paraview & GiD.