최종 결과

```
One
Two
Three
One is Changed
Two
"Three" is chaged
One is Changed
Two is Changed
""Three" is chaged" is chaged

Process finished with exit code 0
```

최종 코드 (four 객체 포함)

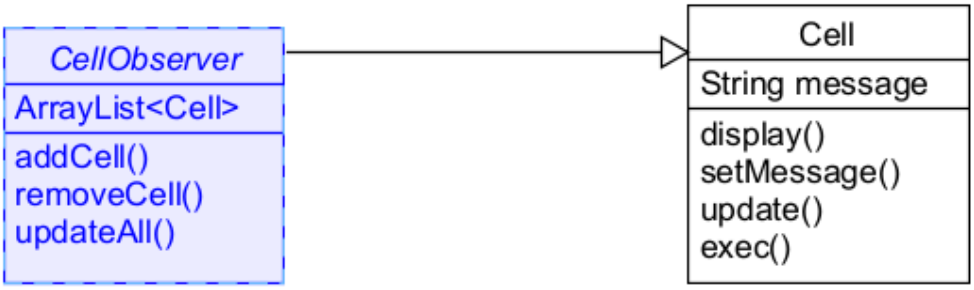
```
package Test_observer;

public class Main {
    public static void main(String[] args){

        Cell One = new Cell("One");
        Cell Two = new Cell("Two");
        Cell Three = new Cell("Three");
        Cell four = new Cell("four");
        One.addCell(Three);
        Two.addCell(Three);
        One.addCell(four);
        Two.addCell(four);

        One.display();
        Two.display();
        Three.display();
        System.out.println();

        One.setMessage("One is Changed");
        One.display();
        Two.display();
        Three.display();
        System.out.println();

        Two.setMessage("Two is Changed");
        One.display();
        Two.display();
        Three.display();
        System.out.println();

        //four.display();
```

```
    }
}
```

```
package Test_observer;

public class Cell extends CellObserver{
    String message;

    Cell(String message){
        this.message = message;
    }

    public String getMessage(){
        return this.message;
    }
    public void setMessage(String message){
        this.message = message;
        updateAll();

    }
    public void display(){
        System.out.println(message);
    }
    public void exec(){
        this.message = "\"" + this.message + "\" is chaged";
    }

    @Override
    public void update() {
        exec();
    }
}
```

```
package Test_observer;

import java.util.ArrayList;

public abstract class CellObserver {

    private ArrayList<Cell> observers = new ArrayList<Cell>();

    public abstract void update();
    public void updateAll(){
        for(Cell observer : observers){
            observer.update();
        }
    }
    public void addCell(Cell cellObserver){
        observers.add(cellObserver);
    }
    public void removeCell(Cell cellObserver){
        observers.remove(cellObserver);
    }
```

```
}
```



9.

Observer의 역할이 cell 클래스에 국한되지 않고 사용할 수 있기 때문에 상속관계로 만들어 필요한 클래스에서 상속받아 사용하는 것이 코드의 재사용성을 높인다고 생각한다. 즉, cell 클래스가 observer를 필요로 하는 다른 클래스가 있을 때 해당 observer을 그대로 사용할 수 있다. 하지만 그렇지 않다면 cell 클래스에 모든 기능을 다 넣어도 무방하다고 본다. 지금 이 모델의 구조는 cell 인스턴스가 생기면 해당 인스턴스에 대한 observer가 생성되고 cell 마다 다른 cell을 집합체에 넣어 관리/감독 할 수 있도록 설계되어 있다. 이를 하나의 cell 클래스에서 모두 구현해도 다른 cell 객체에 대한 참조 및 관리 감독이 가능하기 때문이다.