# Язык программирования Java

Дмитриев Андрей Владиславович
andrei-dmitriev@yandex.ru
2007

# Наблюдатель (Observer)

- Определяет между объектами зависимость типа один-ко-многим так, что при изменении состояния одного объекта, все зависящие от него объекты получают об этом оповещение.

# Наблюдатель

Класс LogConsole должен изменять цвет вывода сообщений в зависимости от их приоритета, определяемого и устанавливаемого классом MainWindow.

```
class LogConsole extends Terminal{
   private color = Color.BLACK;
   void setColor(Color c){
      this.color = c;
   }
}
```

# Наблюдатель (cont.)

Данный класс может напрямую изменять состояние подчиненного объекта при изменении ситуации:

```
class MainWindow{
   LogConsole logConsole = LogConsole.create();
   void userAction(Action a){
      if (a.getLevel() ==
          Level.isCritical())
      {
          logConsole.setColor(Color.RED);
      }
   }
}
```

# Наблюдатель (cont.)

У нас появился еще один класс, заинтересованный в типе действия пользователя – IntrusionDetector.

```
class MainWindow{
  … //все поля и методы остаются
  IntrusionDetector id = IntrusionDetector.get();
  void userAction(Action a){
      … //все прошлые действия остаются
      if (a.getLevel() ==
          Level.isIntrusion()) {
          id.alarm();
      }
   }
}
```

# Наблюдатель (cont.)

Для большей структурированности имеет смысл выделить круг классов, заинтересованных в данном типе событий:

```
public interface ActionObserver{
  void actionHappen(Action a);
}
```

# Наблюдатель (cont.)

Теперь класс LogConsole и IntrusionDetector можно причислить к кругу заинтересованных:

```
class LogConsole extends Terminal implements
  ActionObserver {
  void actionHappen(Action a){
      if (a.getLevel() ==
          Level.isCritical())
      {
          setColor(Color.RED);
      }
  }
}
```

# Наблюдатель (cont.)

Аналогично с классом IntrusionDetector:

```
class IntrusionDetector implements
  ActionObserver{
     public void actionHappen(Action a){
          if (a.getLevel() ==
             Level.isIntrusion())
          {
              alarm();
          }
```

# Наблюдатель (cont.)

Класс MainWindow должен хранить ссылки на все классы, заинтересованные в событиях:

```
class MainWindow{
  ActionObserver []observers = new ActionObserver
   [10];
  void userAction(Action a){
      //обход массива
      for (…) {
         observers[i].actionHappen(a);
      }
     //заметим, что вся логика по обработке
     //события переместилась в конкретные классы.
   }
}
```

# Наблюдатель (cont.)

Класс MainWindow должен предоставлять интерфейс для пополнения списка объектов, заинтересованных в данных сообщениях:

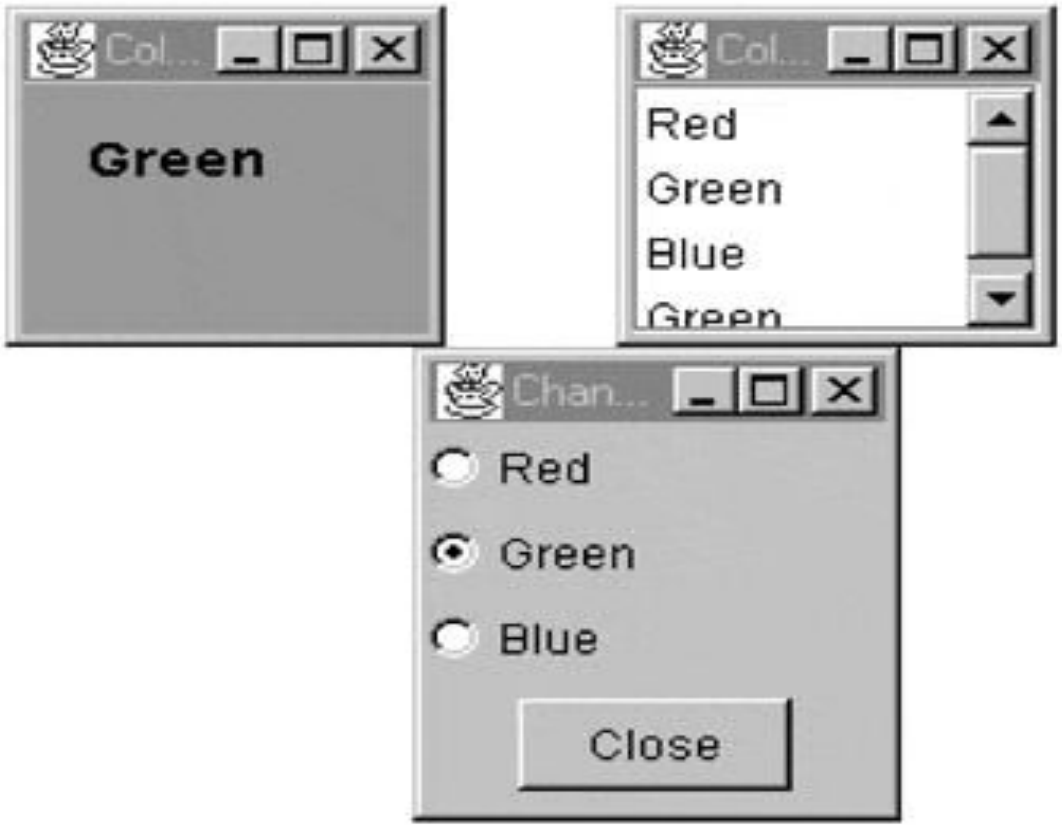
```
class MainWindow{
 ActionObserver []observers = new ActionObserver [10];
   …
   public void  addActionObserver(ActionObserver aa){
      observers[last++] = aa;
   }
   public void  removeActionObserver(ActionObserver aa){
      //удаление объекта из массива
   }
}
```

# Пример реализации наблюдателя в JDK

Наблюдатель – любой класс, реализующий один или несколько слушателей: MouseListener, ItemListener, FocusListener, и т.д.

```
import java.awt.event.*
…
List list = new List();
il = new ItemListener(){
  //метод будет вызываться каждый раз при
  //изменении состояния списка
  public void itemStateChanged(ItemEvent e){
     System.out.println("event " + e);
  }
};
list.addItemListener(il);
```

# Наблюдатель (иллюстрация)



# Наблюдатель (выводы)

- Позволяет избежать циклических опросов состояния.
- Возможность реализовать неограниченное число наблюдателей.