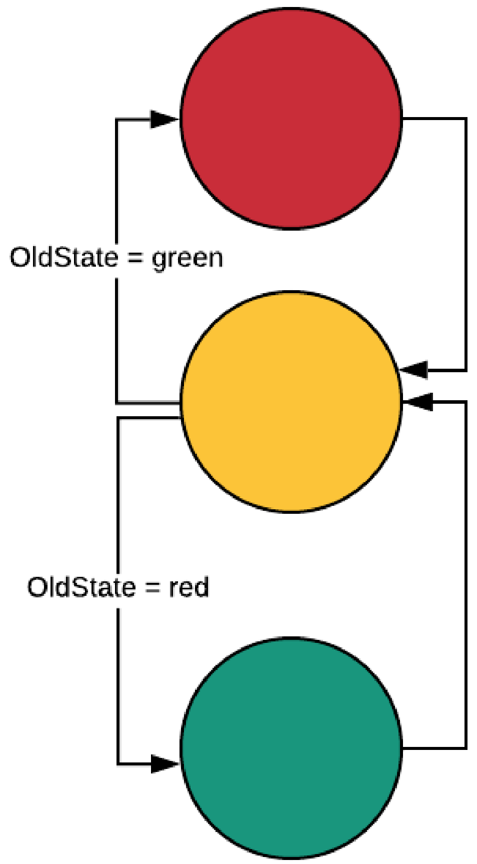
Паттерн состояние

Описание

- Управляет поведением путем изменения своего состояния
- Состояния можно описать конечным автоматом

Пример



Простое решение

```
public class SimpleSemaphorBlink {
   final static int RED = 0;
   final static int GREEN = 1;
   final static int YELLOW = 2;
   final static int BLINK_GREEN = 3;
   int state;
   int stateOld;
   public SimpleSemaphorBlink() {
     state = RED;
     stateOld = RED;
   }
   void changeState(){

      if(state==RED){
         stateOld = state;
         state = YELLOW;
         System.out.println("YELLOW");
      }
      else if(state==BLINK_GREEN){
         stateOld = state;
         state = YELLOW;
         System.out.println("YELLOW");
      }
      else if(state==YELLOW && stateOld==RED){
         stateOld = state;
         state = GREEN;
         System.out.println("GREEN");
      }
      else if(state==YELLOW && stateOld==BLINK_GREEN){
         stateOld = state;
         state = RED;
         System.out.println("RED");
      }
      else if(state == GREEN){
         stateOld = state;
         state = BLINK_GREEN;
         System.out.println("BLINK_GREEN");
      }
   }
    public static void main(String[] args) {
     SimpleSemaphorBlink semaphor = new
       SimpleSemaphorBlink();
     for(int i=0;i<10;i++)
     {
     semaphor.changeState();
     }
   }
}
```

Потоки

```
public ColorEnum print() {
      return colorEnum;
   }

   public void changeState() {
      state.changeColor();
      gm.setColor(colorEnum);
   }

   @Override
   public void run() {
      for (int i = 0; i < 200; i++) {
         changeState();
         stop();
      }

   }

   private void stop() {
      try {
         Thread.sleep(200);
         synchronized (this) {
            while (suspendFlag) {
               wait();
            }
         }
      } catch (InterruptedException ex) {

       Logger.getLogger(StateSemaphor.class.getName()).log(
       Level.SEVERE, null, ex);
      }
   }

   public synchronized void mysuspend() {
      suspendFlag = true;
   }

   public synchronized void myresume() {
      suspendFlag = false;
      notify();
   }
```

Модель

```
public class StateSemaphor
    implements Runnable {

   ChangeColor green;
   ChangeColor red;
   ChangeColor yellow;
   ChangeColor state;
   ChangeColor oldState;
   GraphicsModel gm;
   ColorEnum colorEnum;
   boolean suspendFlag = false;
   int time;

   public
StateSemaphor(GraphicsModel
model) {
   green = new Green();
   red = new Red();
   yellow = new Yellow();
   state = green;
   oldState = green;
   time = 10;
   gm = model;
   colorEnum =
ColorEnum.TGreenYellowRed;
   suspendFlag = false;
}
```

Зеленый

```
public class Green implements ChangeColor {

    @Override
    public void changeColor() {
        oldState = green;
        state = yellow;
        colorEnum = GreenTYellowRed;
        try {
            Thread.sleep(100);
        } catch (InterruptedException ex) {
            Logger.getLogger(Green.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
  }
```

Желтый

```
public class Yellow implements ChangeColor {

    @Override
    public void changeColor() {
      if (oldState == red) {
        state = green;
        oldState = yellow;
        colorEnum = TGreenYellowRed;
      } else {
        state = red;
        oldState = yellow;
        colorEnum = GreenYellowTRed;
      }
      try {
        Thread.sleep(100);
      } catch (InterruptedException ex) {
        Logger.getLogger(Yellow.class.getName()).log(Level.SEVERE, null, ex);
      }
    }
  }
```

Графическая модель

```
public class GraphicsModel extends Observable {

   Color green = Color.green;
   Color red = Color.red;
   Color yellow = Color.yellow;
   ColorEnum colorEnum;
   RectangularShape oneShape = new Ellipse2D.Double();
   RectangularShape shape[] = new RectangularShape[3];

   public GraphicsModel() {
      Point2D loc = new Point2D.Double();
      colorEnum = TGreenYellowRed;
      //oneShape.setFrame(loc, size);
   }

   public void paint(Graphics g) {
      //   this.setBackground(Color.black );
      //  super.paintComponent(g);
      if (colorEnum != null) {
         g.setColor(Color.red);
         int x = -88, y = -88;
         if (colorEnum.green) {
            g.fillOval(165 + x, 100 + y, -2 * x, -2 * y);
         } else {
            g.drawOval(165 + x, 100 + y, -2 * x, -2 * y);
         }
         g.setColor(Color.yellow);
         if (colorEnum.yellow) {
            g.fillOval(165 + x, 285 + y, -2 * x, -2 * y);
         } else {
            g.drawOval(165 + x, 285 + y, -2 * x, -2 * y);
         }
         g.setColor(Color.green);
         if (colorEnum.red) {
            g.fillOval(165 + x, 470 + y, -2 * x, -2 * y);
         } else {
            g.drawOval(165 + x, 470 + y, -2 * x, -2 * y);
         }
      }
   }

   public void setColor(ColorEnum c) {
      colorEnum = c;
      setChanged();
      notifyObservers();
   }
}
```

Controller

```
public class Controller {

    GraphicsModel model;
    StateSemaphor semaphor;
    MyPanel panel;
    MyFrame frame;
    private static Controller controller = null;

    public void draw(Graphics g) {
        model.paint(g);
    }

    private Controller() {
        panel = new MyPanel(this);
        model = new GraphicsModel();
        model.addObserver(panel);
        semaphor = new StateSemaphor(model);
        java.awt.EventQueue.invokeLater(new
        Runnable() {
            public void run() {
                new MyFrame(panel,
            semaphor).setVisible(true);
            }
        });
    }

    public static Controller getIntance() {
        if (controller == null) {
            controller = new Controller();
        }
        return controller;
    }
}
```

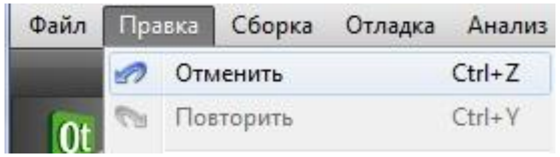
Frame

```
public class MyFrame extends JFrame {

   MyPanel myPanel;
   StateSemaphor ss;
//    Controller controller;

   public MyFrame(MyPanel myPanel, StateSemaphor s) {
      this.myPanel = myPanel;
      ss = s;
      JToolBar bar = new JToolBar();
      add(bar, BorderLayout.NORTH);
      JMenuItem start = new JMenuItem(new ImageIcon("start.png"));
      JMenuItem stop = new JMenuItem(new ImageIcon("stop.png"));
      JMenuItem continue1 = new JMenuItem(new
         ImageIcon("continue.png"));
      stop.setEnabled(false);
      continue1.setEnabled(false);
      start.setEnabled(true);
      bar.add(start);
      start.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            new Thread(ss).start();
            stop.setEnabled(true);
            continue1.setEnabled(false);
            start.setEnabled(false);
         }
      });
      bar.add(continue1);
      continue1.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            ss.myresume();
            start.setEnabled(false);
            stop.setEnabled(true);
            continue1.setEnabled(false);
         }
      });

      bar.add(stop);
      stop.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            ss.mysuspend();
            start.setEnabled(false);
            stop.setEnabled(false);
            continue1.setEnabled(true);
         }
      });
      add(this.myPanel);
      this.setSize(350, 780);
      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }

}
```

Задание

Измените светофор – добавьте мигающий зеленый

UndoMachine



Undo

Redo

Activity

}

```
public interface Activity {
        void getPointOne(Point2D p1);
        void getPointTwo(Point2D p1);
        void setModel(Model m);
        void execute();
        void unexecute();
        Activity clone();
}
```

Activity Draw

```
public class Draw implements Activity{
   Model model;
   Point2D[] p;
   MyShape myShape;

…
   @Override
   public void getPointOne(Point2D p1){
       p[0] = p1;
       myShape =model.inintCurrentShape();
   }

   public void getPointTwo(Point2D p1){
      p[1] = p1;
      model.changeShape(p);
   }

   @Override
   public void execute() {
      model.setActiveShape(myShape);
   }

   @Override
   public void unexecute() {
      model.ctrlZ_Shape();
   }

   @Override
   public Activity clone() {
     Draw d = new Draw(model);
      d.myShape = myShape;
      d.p = p;
      return d;
   }

   @Override
   public void setModel(Model m) {
      model = m;
   }

}
```

Автомат

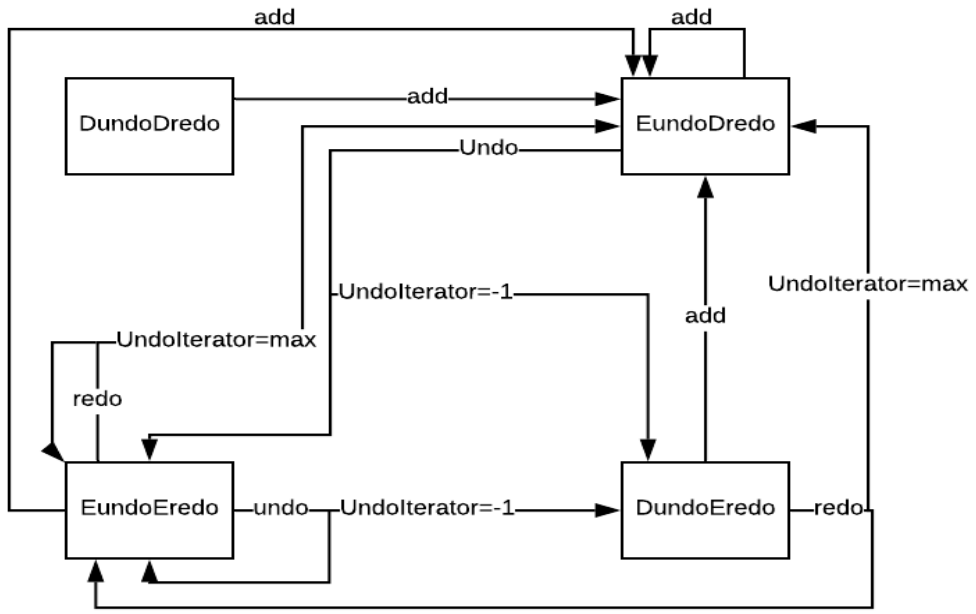
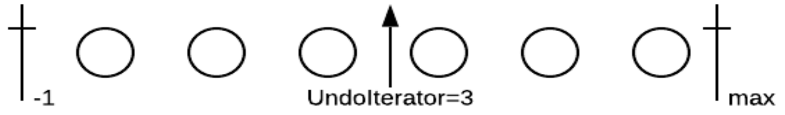


Диаграмма состояний
автомата UndoRedo



class UndoMachine

```
ArrayList<Activity> activityList;

    UndoRedoState stateDUndoDRedo;
    UndoRedoState stateEUndoERedo;
    UndoRedoState stateDUndoERedo;
    UndoRedoState stateEUndoDRedo;
    UndoRedoState state;
    int undoIterator;

    public UndoMachine() {

        activityList = new ArrayList<Activity>();
        stateDUndoDRedo = new StateDUndoDRedo();
        stateEUndoERedo = new StateEUndoERedo();
        stateDUndoERedo = new StateDUndoERedo();
        stateEUndoDRedo = new StateEUndoDRedo();
        state = stateDUndoDRedo;
        undoIterator = -1;
    }

    public void add(Activity action) {
        state.add(action);
    }

    public void execute() {
        state.redo();
    }

    public void unexecute() {
        state.undo();
    }

    public int getUndoIterator() {
        return undoIterator;
    }

    public void notifyMenu() {
        setChanged();
        notifyObservers(state.getButtonState());
    }
}
```

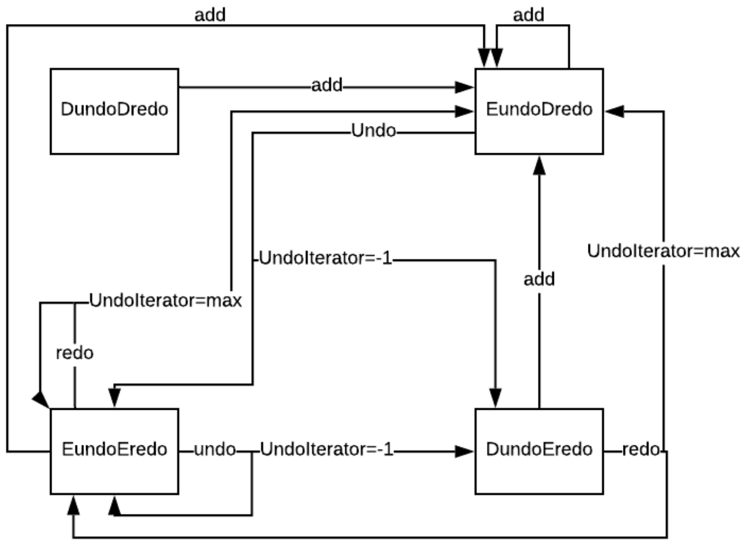
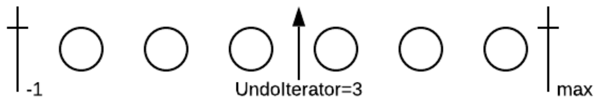


Диаграмма состояний
автомата UndoRedo



enum UndoRedoButtonState

Undo	Redo
0	0
1	1
1	0
0	1

```
public enum UndoRedoButtonState {
      DUndoDRedo(false, false)
      EUndoERedo(true, true),,
      EUndoDRedo(true, false),
      DUndoERedo(false, true);
    public boolean undo;
    public boolean redo;

    UndoRedoButtonState(boolean u,
      boolean r) {
        undo = u;
        redo = r;
    }
}
```

UndoRedoState

```
private class UndoRedoState {
    UndoRedoButtonState buttonState;
    public UndoRedoState(UndoRedoButtonState
      buttonState) {
      this.buttonState = buttonState;
    }
    public UndoRedoButtonState getButtonState() {
      return buttonState;
    }

    void undo() {
      activityList.get(undoIterator).unexecute();
      undoIterator--;
      if (undoIterator == -1) {
        state = stateDUndoERedo;
        notifyMenu();
      } else {
        goToEUndoERedo();
      }
    }

    void redo() {
      undoIterator++;
      activityList.get(undoIterator).execute();
      if (undoIterator == activityList.size() - 1) {
        state = stateEUndoDRedo;
        notifyMenu();
      } else {
        goToEUndoERedo();
      }
    }
  }
```

```
final void add(Activity action) {
        deleteHistory();
        activityList.add(action);
        undoIterator++;
        state =
stateEUndoDRedo;
        notifyMenu();
    }

    void goToEUndoERedo() {
        state =
stateEUndoERedo;
        notifyMenu();
    }
```

```
void deleteHistory() {
        if (!activityList.isEmpty()) {
          for (int i = undoIterator; i <
activityList.size(); i++) {
              activityList.remove(i);
          }
        }
    }
```

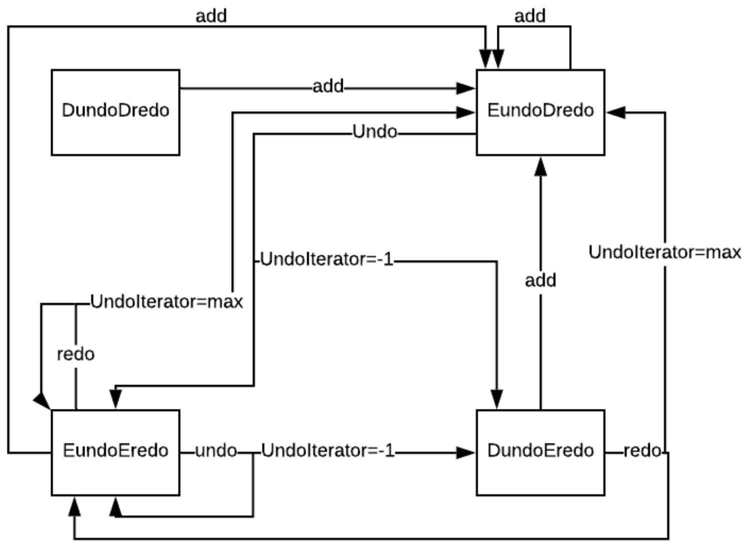
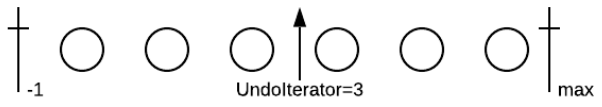


Диаграмма состояний
автомата UndoRedo



UndoMachine inner classes

```
private class StateDUndoDRedo extends UndoRedoState {

      public StateDUndoDRedo() {
         super(UndoRedoButtonState.DUndoDRedo);
      }

      @Override
      public void undo() {
      }

      @Override
      public void redo() {
      }

      @Override
      void goToEUndoERedo() {
      }

      @Override
      void deleteHistory() {
      }
   }

private class StateDUndoERedo extends UndoRedoState {

      public StateDUndoERedo() {
         super(UndoRedoButtonState.DUndoERedo);
      }

      @Override
      public void undo() {
      }
   }
```

```
private class StateEUndoERedo extends UndoRedoState {

      public StateEUndoERedo() {
         super(UndoRedoButtonState.EUndoERedo);
      }

      @Override
      void goToEUndoERedo() {
      }
   }

private class StateEUndoDRedo extends UndoRedoState {

      public StateEUndoDRedo() {
         super(UndoRedoButtonState.EUndoDRedo);
      }

      @Override
      public void redo() {
      }

      @Override
      void deleteHistory() {
      }
   }
```

Menu and Frame

```
menuItems.add(new SwitchUndo("undo",new
    ImageIcon("undo.gif"),undoMachine));
    menuItems.add(new SwitchRedo("redo",new
    ImageIcon("redo.gif"),undoMachine));
    menuItems.add(new SwitchState("выбор
    цвета", new ImageIcon("colors.gif"),
        new SwitchColor(state)));

    undoMachine.addObserver((SwitchUndo)me
    nuItems.get(menuItems.size()-3));

    undoMachine.addObserver((SwitchRedo)me
    nuItems.get(menuItems.size()-2));
    undoMachine.notifyMenu();
```

```
public class SwitchRedo extends AbstractAction implements Observer{

  public SwitchRedo(String name, Icon icon, UndoMachine machine) {
    super(name, icon);
    putValue("machine", machine);
  }

  @Override
  public void actionPerformed(ActionEvent e) {
    UndoMachine m = (UndoMachine)getValue("machine");
    if (this.isEnabled()) m.execute();
  }

  @Override
  public void update(Observable o, Object arg) {
    UndoMachine.UndoRedoButtonState buttonState =
      (UndoMachine.UndoRedoButtonState) arg;
    this.setEnabled(buttonState.redo);
  }
}
public class SwitchUndo extends AbstractAction implements Observer{
…..
  @Override
  public void update(Observable o, Object arg) {
    UndoMachine.UndoRedoButtonState buttonState =
      (UndoMachine.UndoRedoButtonState) arg;
    this.setEnabled(buttonState.undo);
  }
}
```

Изменения

- Controller
- MyFrame
- Activity
- Model