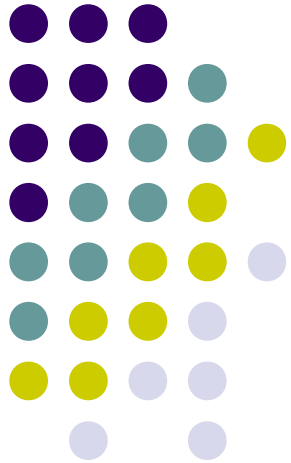
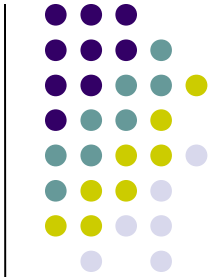
CO225: Software construction

Event handling

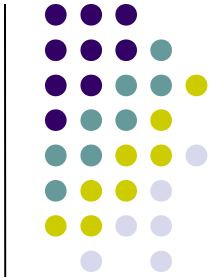


Plan



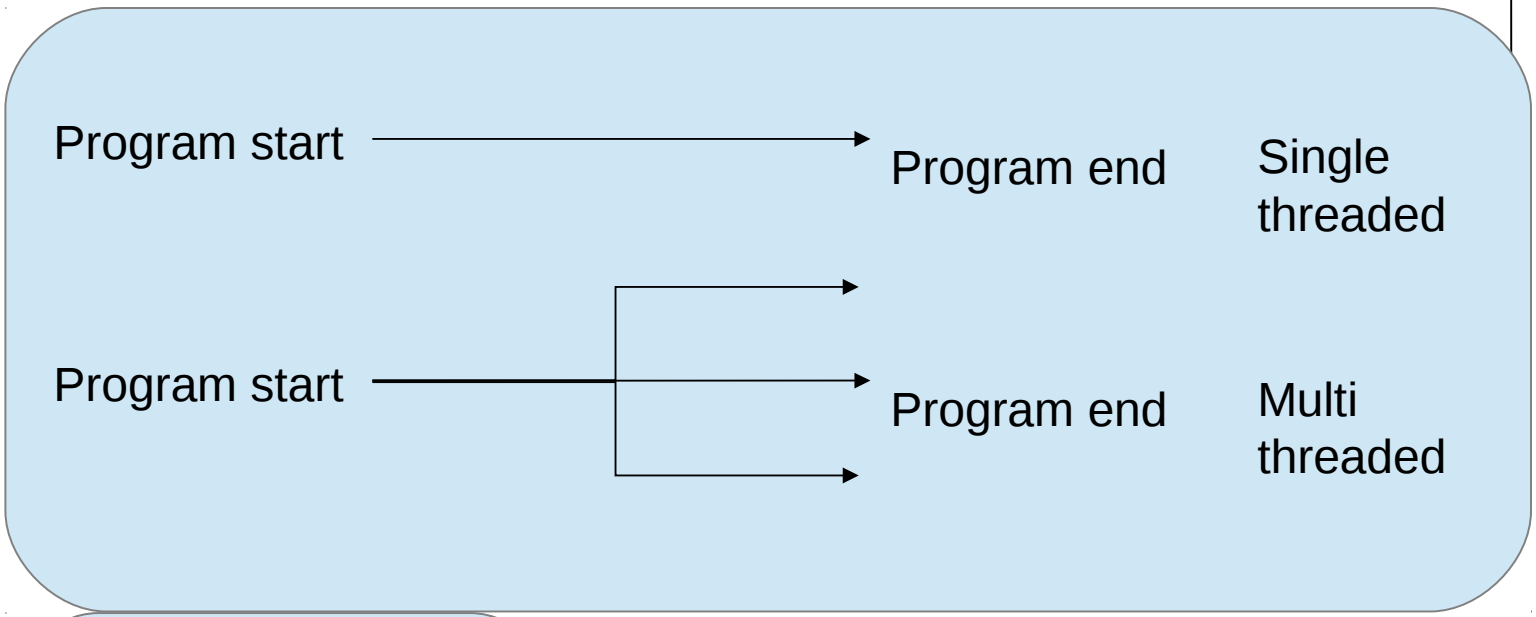
- Basics of event handling
  - Concept of events vs threads
- How it is done in Java (GUI)
  - You have already used this
- Examples

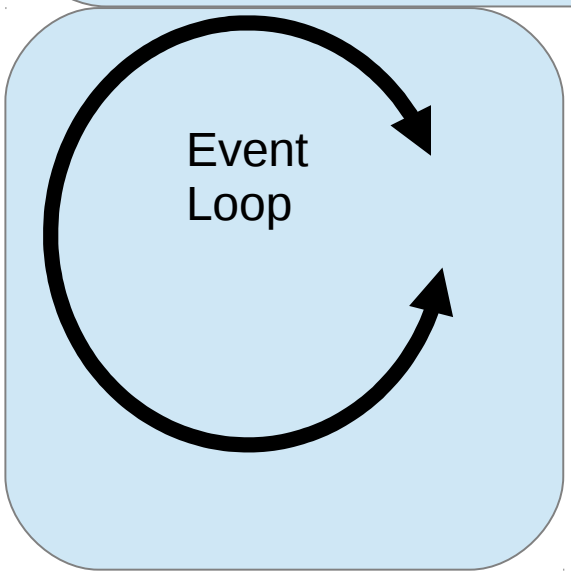
Plan



- Basics of event handling
  - Concept of events vs threads
- How it is done in Java (GUI)
  - You have already used this
- Examples

Threads vs. events





Events

First example
(see Flip.java)

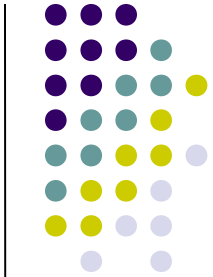
```
public class Flip extends JPanel
    implements ActionListener {

    JButton button;

    public Flip() {
        super(new BorderLayout());
        button = new JButton("Click Me");
        button.setPreferredSize(new Dimension(200, 80));
        add(button, BorderLayout.CENTER);
        button.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        button.setText("On");
    }
```

More on this later

Set a listener for the button

Invoked when the button is clicked

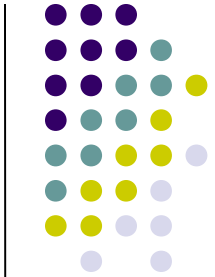
Back to the sample code (given for GUI)



```
JButton loginbutton=new JButton("Login");
loginbutton.setBounds(10, 80, 80, 25);
loginbutton.addActionListener(new Action());
panel.add(loginbutton);
```

```
static class Action implements ActionListener{



        public void actionPerformed(ActionEvent arg0) {
            JFrame frame2=new JFrame("Login ");
            frame2.setVisible(true);
            frame2.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame2.setSize(200,200);
```

Exercise



- Change the above implementation so that when you click the label will change from on to off and off to on ..

Model View Control (MVC)

Flip switch

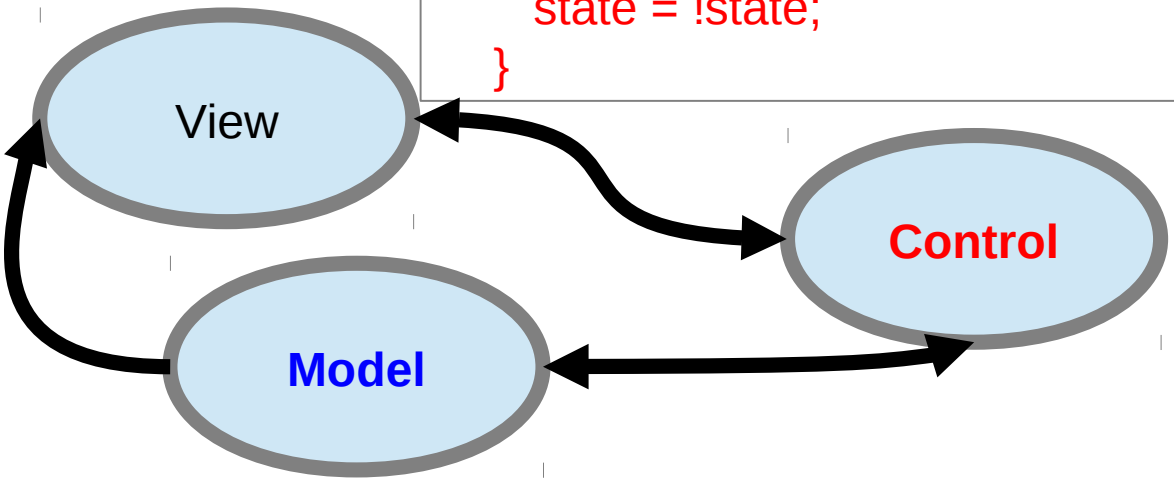
You have a

→ State (Model)

→ Controller

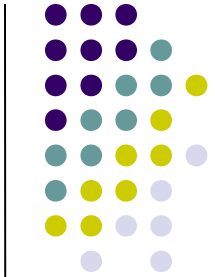
→ View

```
.....
 Boolean state = false;


    public void actionPerformed(ActionEvent e) {
      if(!state)
          button.setText("On");
      else
          button.setText("Off");

      state = !state;
    }
```



Listing to other events (timer)

See StopWatch.java



```
public StopWatch() {
      super(new BorderLayout());
      button = new JButton(min + ":" + sec);
      button.setPreferredSize(new Dimension(200, 80));
      add(button, BorderLayout.CENTER);
     Timer timer = new Timer(1000, this);
     timer.start();
  }


  public void actionPerformed(ActionEvent e) {
     sec += 1;
     if(sec == 60) {
         min ++;
         sec = 0;
     }

     button.setText(min + ":" + sec);
  }
```

Note:

→ MVC

→ Listen to timer events

Listing to other events (timer)

See StopWatch2.java

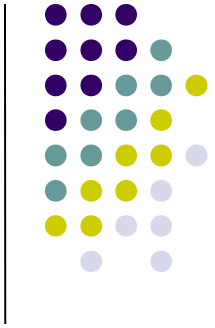
```
Timer timer;
Int sec, min;
….
public StopWatch2() {
...
      button.addActionListener(this);
      timer = new Timer(1000, this);
...
   }

   public void actionPerformed(ActionEvent e) {
      System.out.println(e.getSource()); // registered for different sources
      if(e.getSource() == timer) {// caused by timer
      ….
      }
      else {
      ….
      }
      button.setText(min + ":" + sec);
   }
```

Note:

→ MVC

→ Listen to timer events and clicks

Listing to events
(multiple listeners for same event)



```
public Multi() {
      super(new BorderLayout());
      button = new JButton("Click me");
      button.setPreferredSize(new Dimension(200, 80));
      add(button, BorderLayout.CENTER);
      button.addActionListener(new MyEvent(1));
      button.addActionListener(new MyEvent(2));
   }
```

See Multi.java

```
class MyEvent implements ActionListener {
    int eventId;
    public MyEvent(int eventId) {
       this.eventId = eventId;
    }

    public void actionPerformed(ActionEvent e) {
       System.out.println(eventId + ": Got an event");
    }
}
```

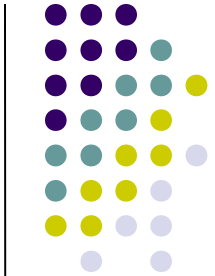
Treadmill display

→ Display Time, speed, distance, calories
→ We will leave Calories out for now. (see the additional documentation later to see how this should be calculated)

→ You should be able to change the speed in steps of 0.1KMPH
→ Accuracy?????

Design a View and Model for this.

Treadmill display (OOP concepts via example)



Major concepts of OOP:

→ Encapsulation (data and code bundled together)

→ Abstraction (internal representation is hidden)

→ Polymorphism (more than one existence)

→ Inheritance (get functionality and state from parent)

Program architecture

Model View Control

Controller

Update time, update distance..

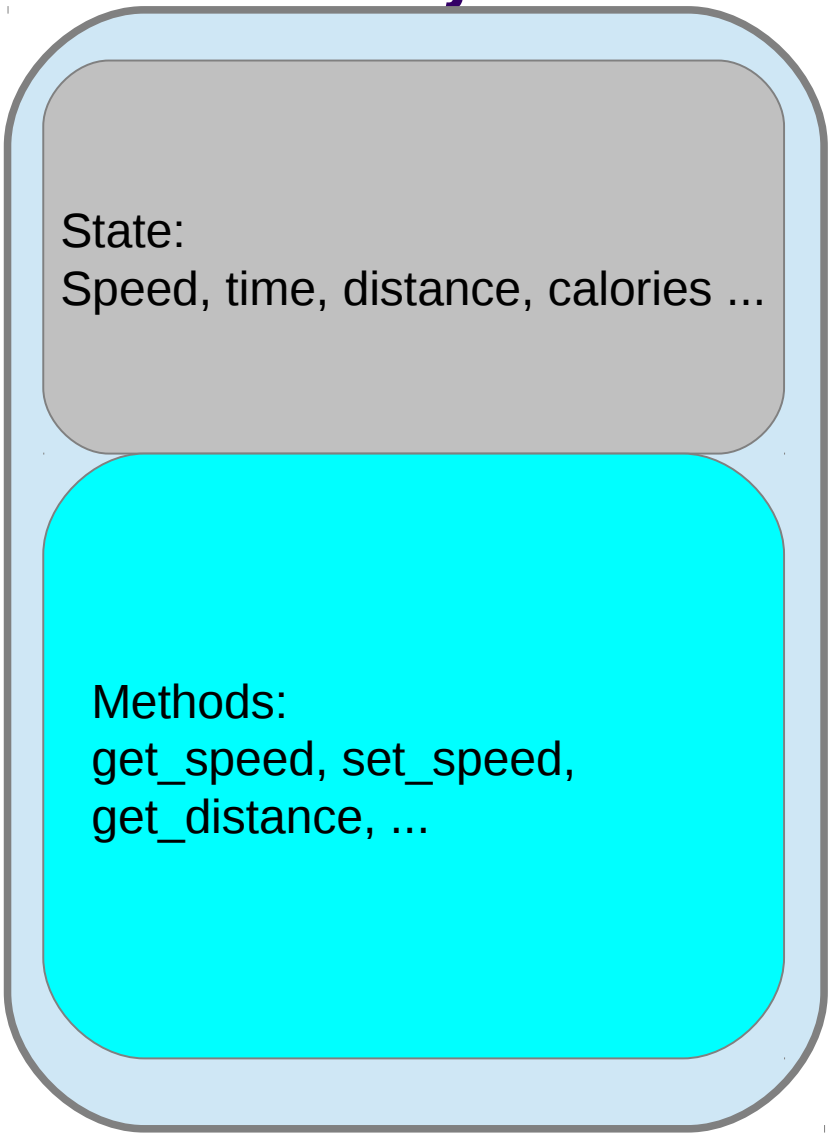
Model

Speed,time, distance, calories bunt …

Some sate not visible as well

Model

see Model.java



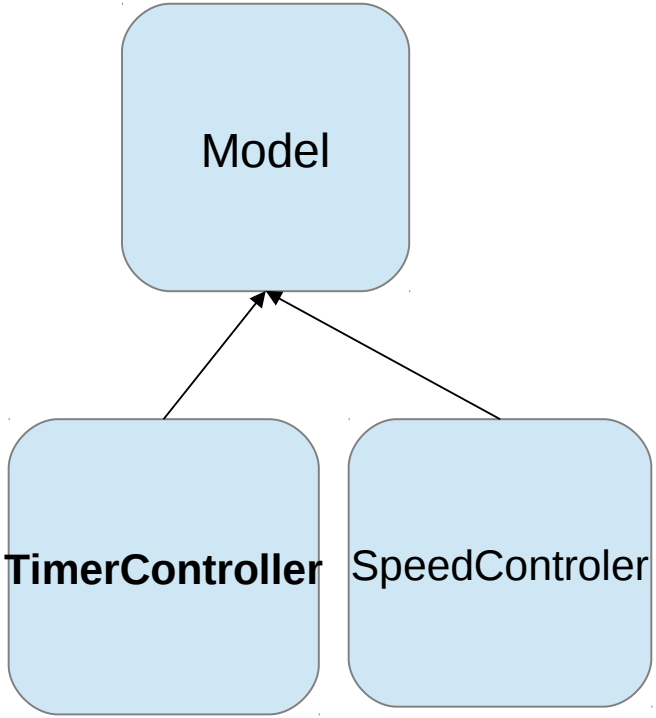
What should go into the model?

What should be the interface?

(encapsulation, abstraction)

Model

see TimerControler.java



All controllers will inherit from Model

→ So we can have a shared, mutable state

→ Some functions can be re-used

→ Access shared state via functions in Model

Model

see TimerControler.java

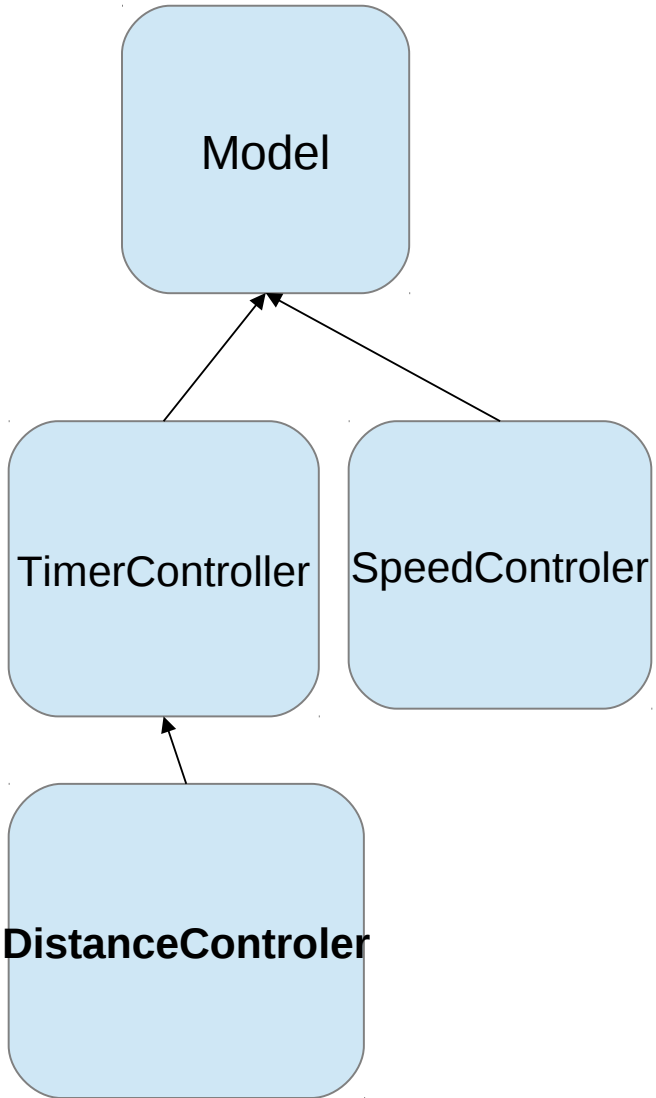
```
protected JButton my_button; //accessible from subclasses
   private Timer timer;

   static int time_step; // in ms

   public TimerControler(JButton button, int time) {
      super();
      my_button = button;
      time_step = time;

      // start a timer
      timer = new Timer(time, this);
      timer.start();
   }
```

Timer controller starts a new timer and on each timer event updates the time.

Receives a reference to the view's label

Model

see DistanceControler.java



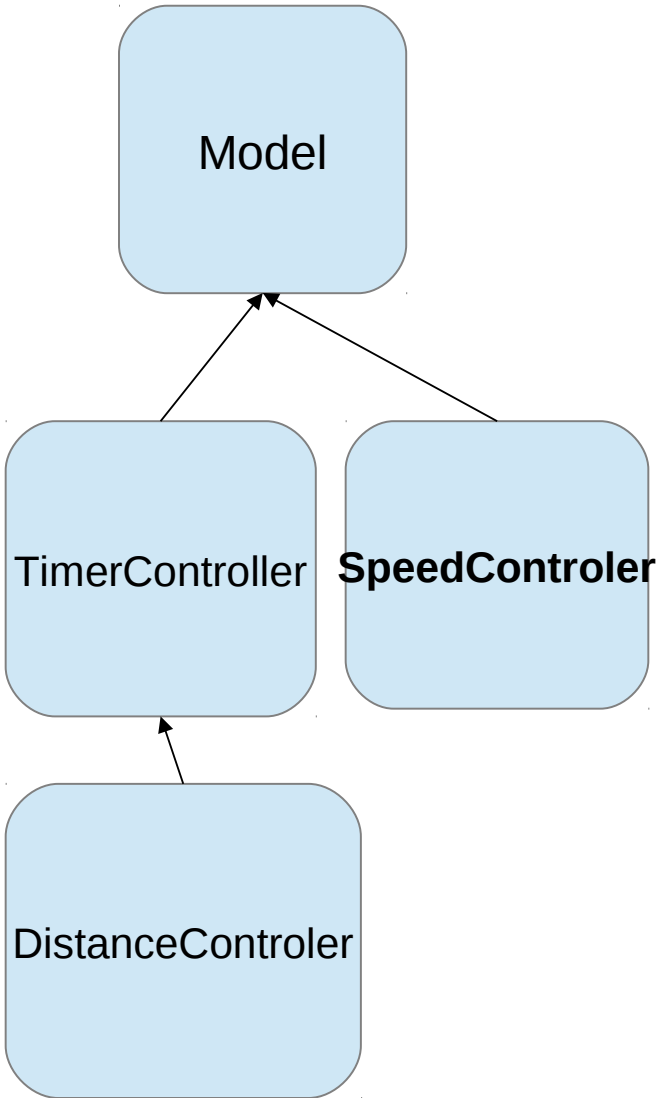
DistanceContoller inherit the behaviour from timer.

After each tick it updates the distance

(inheritance, polymorphism)

Model

see SpeedControler.java



We need two speed controllers to work on + and – buttons.

Use the same class and have a instance variable to say add or subtract