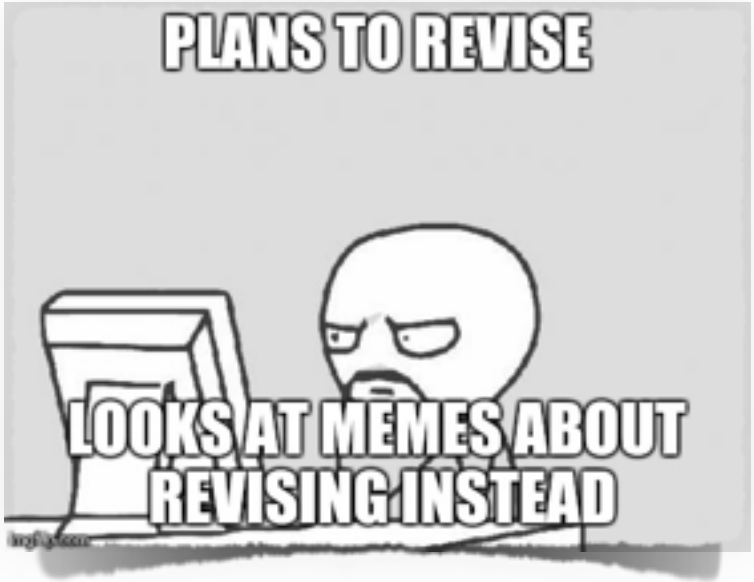
Software Development 2

State Diagram
& GUI Example
F27SB

Labs

- Only 2 labs left
  - Lab 7 - Due week 11
  - Lab 8 - Due week 12
- Lab 8 requires time! Might need more than 2 hours to complete.
- Late submission of Lab 8, come and see me in EM1.52
  - Send me an email first to make sure I'm there



Have your say

OOP

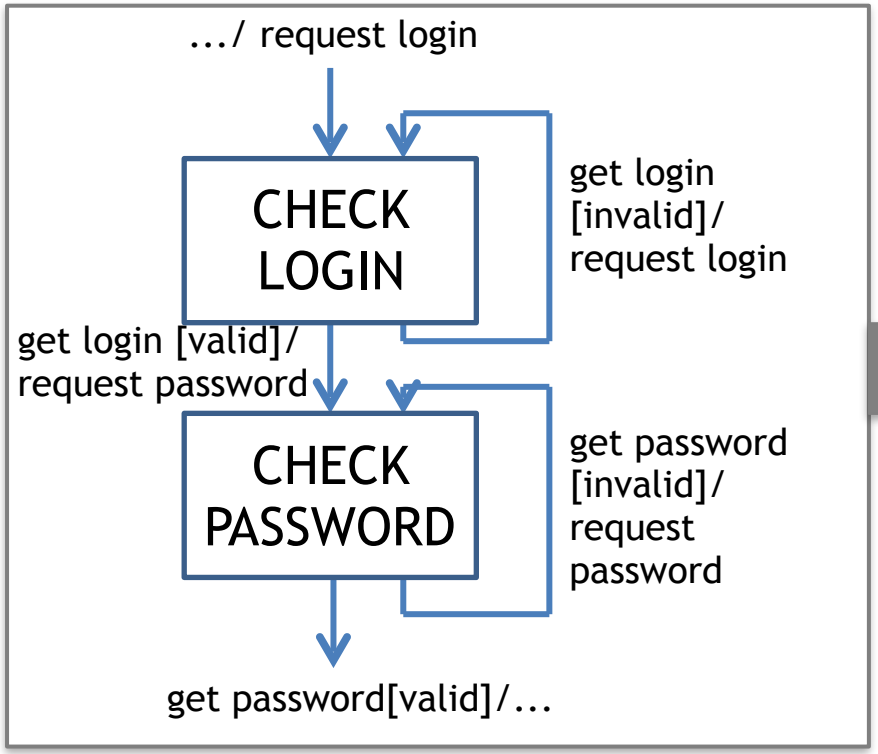
https://doodle.com/poll/pkv4w5kxwya9d89r

GUI

https://doodle.com/poll/tfxbv4sx6yvh858k

Previous Lecture

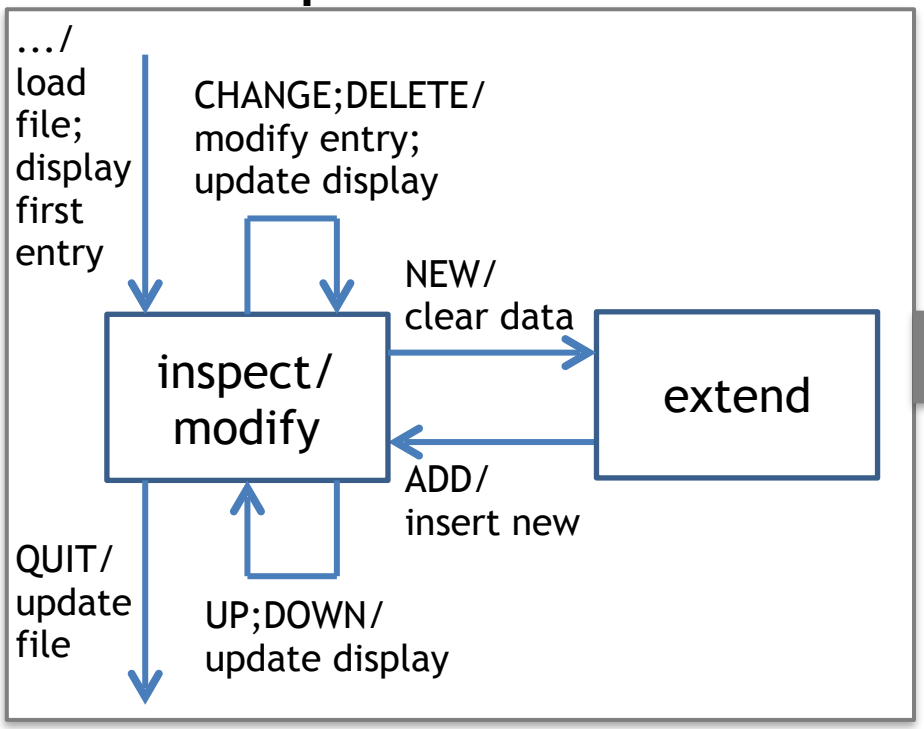
- State diagrams and interactive system design



TODAY’S LECTURE

Today's Lecture

- Example of designing a GUI
  - State diagram, interface design, implementation



Overview

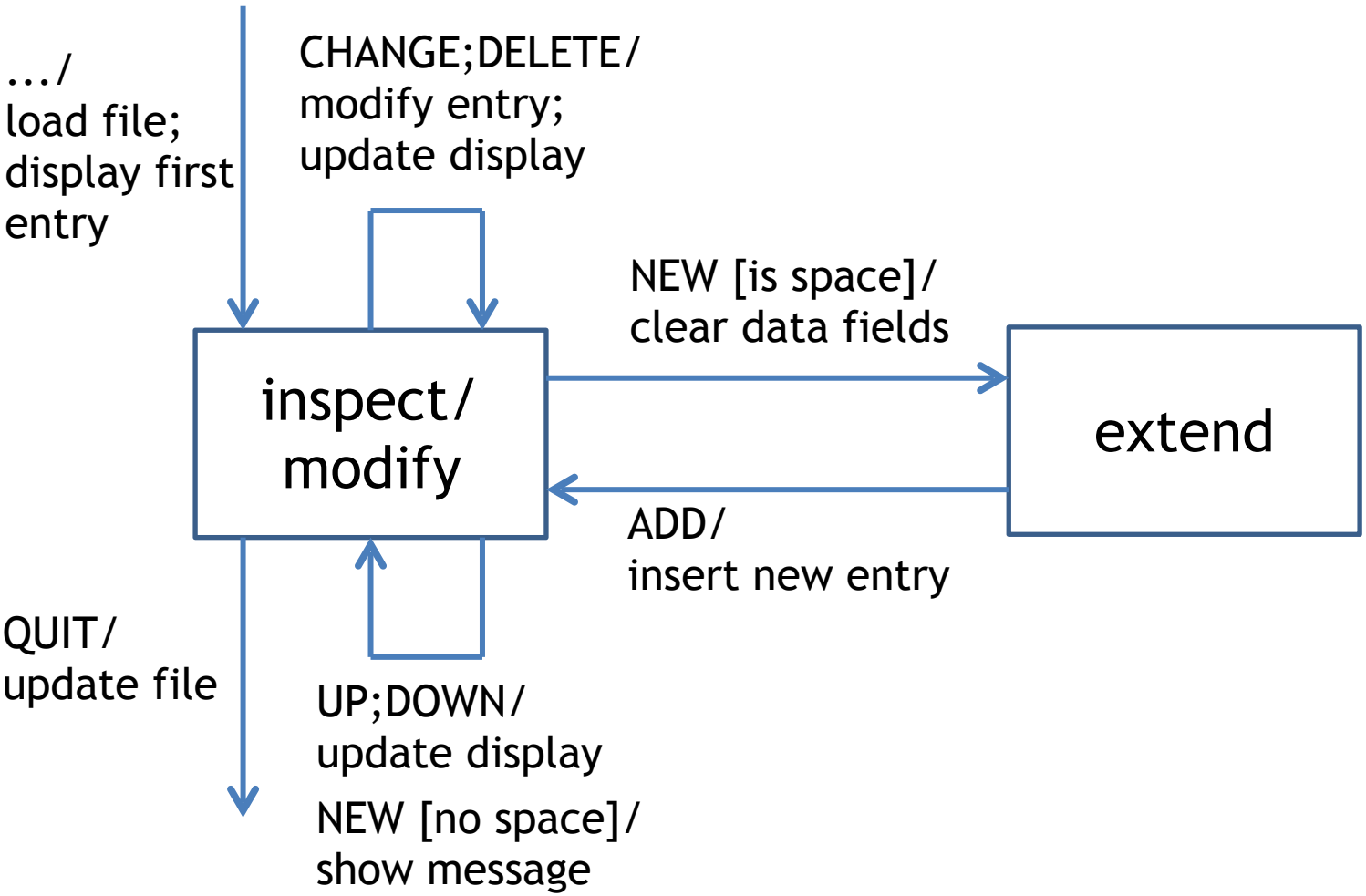
We will develop a simple software address book

- each entry consists of:
  - name
  - street
  - town
  - postcode
  - telephone number
  - email address

Overview

- it loads existing entries from a file
- and saves new entries to the same file
- has controls to:
  - move forwards/backwards through entries
  - add new entry
  - delete entry
  - change entry
  - quit

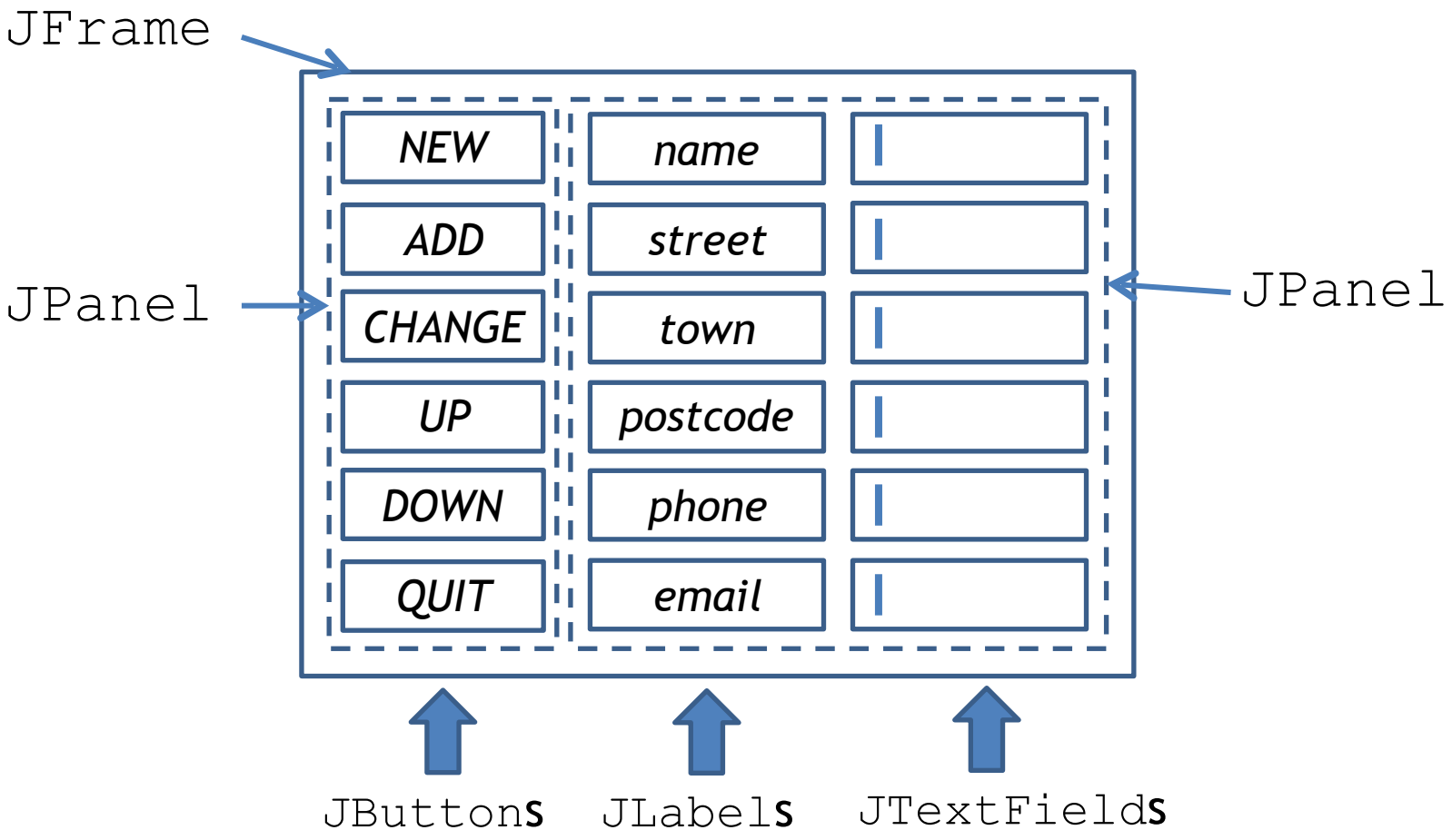
Overview


.../
load file;
display first
entry
CHANGE;DELETE/
modify entry;
update display
NEW [is space]/
clear data fields
inspect/
modify
extend
ADD/
insert new entry
QUIT/
update file
UP;DOWN/
update display
NEW [no space]/
show message

Design

- components:
  - `JButton`s for controls
  - `JLabel`s for headings/prompts
  - `JTextField`s for changeable details
- separate `JPanel`s for controls & details
  - may want to vary number of controls or details

Design


JFrame
JPanel
JPanel
NEW
ADD
CHANGE
UP
DOWN
QUIT
name
street
town
postcode
phone
email
JButtons
JLabels
JTextFields

Entry representation

Each entry within program represented as a class:

- which contains 6 `String` fields

Use individual variables?

- advantage
  - can see which field is which
- disadvantage
  - need lots of individual assignments/accesses
  - might want to add new fields, e.g. country

Entry representation

Use array of strings?

- advantage
  - can manipulate with for loops
- disadvantage
  - may forget which array element corresponds to which field
  - does this matter?

Entry representation

```
// Represents an address book entry
class Entry {
   String [] details; // contents of each field
   static final int MAXDETAILS = 6; // num. fields

   public Entry(String [] newdetails) {
      details = new String[MAXDETAILS];
      for(int i=0;i<MAXDETAILS;i++)
        details[i]=newdetails[i];
   }
```

Entry representation

Keep `Entry`s in a text file when not running

- save each `Entry` to file when the program ends

```
// Entry method: write this entry to a file
  public void writeEntry(PrintWriter file) {
     for(int i=0;i<MAXDETAILS;i++)
        file.println(details[i]);
  }
}
```

Entry representation

```
class Address extends JFrame
              implements ActionListener {
   ...
   Entry [] entries; // address book entries
   int entryno; // number of entries
   int current; // entry currently displayed
   final int MAXENTRIES = 100;


   // file containing saved entries
   final String addressbook = "addressbook.txt";
```

Entry representation

Load array from file when program starts

```
void readEntries() throws IOException {
      BufferedReader file;
      entries = new Entry[MAXENTRIES];
      String [] details =
         new String[Entry.MAXDETAILS];
      entryno = 0; // initialise count

      // first try and open the file
      try {
         file = new BufferedReader
                      (new FileReader(addressbook));
      } catch(FileNotFoundException e){
         return;
      }
```

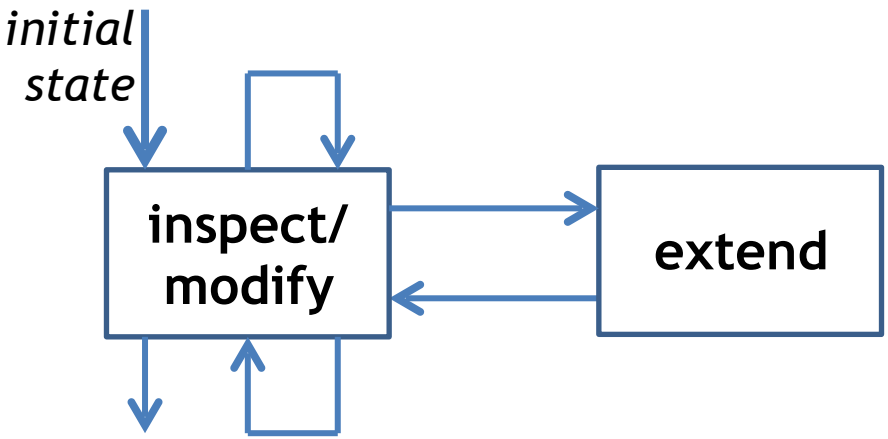
.../
load file;
display
first entry

inspect/
modify

Entry representation

```
String line = file.readLine(); // read file one line at a time
while (line != null) { // until the end is reached
        if (entryno == MAXENTRIES) { // check there's space
            System.out.println("More than " + MAXENTRIES);
            break;
        }
        // read in all the (6) lines for an entry
        // and save them temporarily in an array
        details[0] = line;
        for (int i = 1; i < Entry.MAXDETAILS; i++)
            details[i] = file.readLine();
        // create and save a new entry from this array
        entries[entryno] = new Entry(details);
        entryno++; // increment count
        // start to read next entry ...
        line = file.readLine();
}
```

States

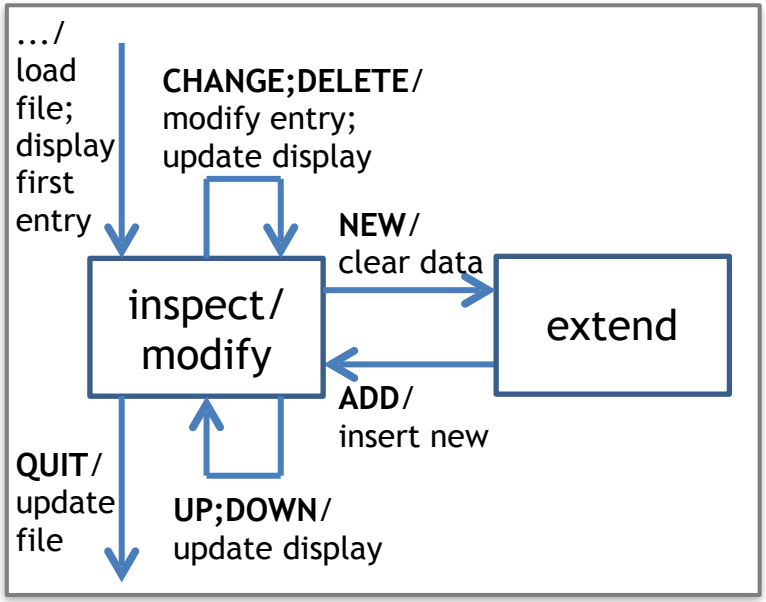



```
class Address extends JFrame
      implements ActionListener {
   final int INSPECT = 0; // inspect/modify state
   final int EXTEND = 1; // extend state
   int state = INSPECT; // set initial state
```

Events

Event sources:

- `JButton` for each control action
- A `string` array for button labels
  - Makes it easier to initialise and add new buttons in the future




```
// events
   JButton [] actions; // event sources
   String [] actionText = {"New","Add","Change",
   "Delete","Forward","Back","Quit"}; // event names
   final int MAXEVENTS = actionText.length;
```

Event sources

- Need to remember `JButton` map
  - actions[0] => *New*
  - actions[1] => *Add*
  - actions[2] => *Change*
  - actions[3] => *Delete*
  - actions[4] => *Forward*
  - actions[5] => *Back*
  - actions[6] => *Quit*

Interface & Initialisation

- `JLabel`s for field titles
  - again, we use an array of `String` for their text
- `JTextField`s for modifiable field details
- `JPanel`s for `Entry` on interface and controls

```
JLabel [] headings; // field labels
String [] text ={"Name","Street","Town","Postcode",
          "Telephone","Email"}; // field text

JTextField [] details; // modifiable text fields

JPanel entry, controls;
```

Interface & Initialisation

Methods to initialise `JLabel`s, `JButton`s and `JTextField`s:

```
// create text field and add to container
JTextField setupTextField(String s,Container c) {
   JTextField t = new JTextField(s);
   t.setFont(new Font("Sansserif",Font.PLAIN,18));
   t.setBackground(Color.white);
   c.add(t);
   return t;
}
```

Interface & Initialisation

```
// create button, add to container, attach listener
JButton setupButton(String s, Container c) {
   JButton b = new JButton(s);
   b.setFont(new Font("Sansserif",Font.PLAIN,18));
   b.setBackground(Color.white);
   c.add(b);
   b.addActionListener(this);
   return b;
}
```

```
// create label and add to container
JLabel setupLabel(String s, Container c)
{  JLabel l = new JLabel(s,JLabel.CENTER);
   l.setFont(new Font("Sansserif",Font.PLAIN,18));
   l.setBackground(Color.white);
   c.add(l);
   return l;
}
```

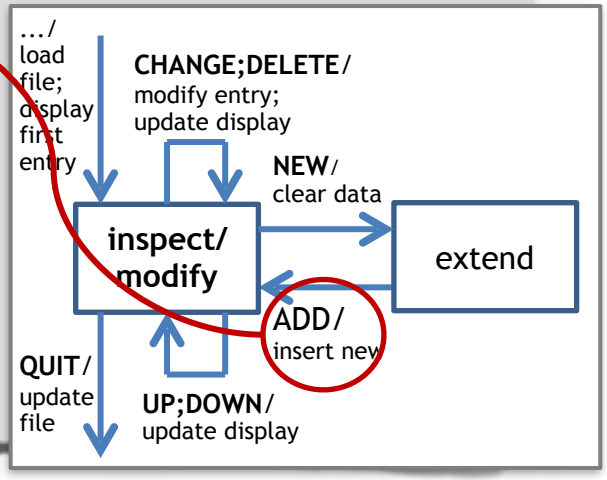
Interface & Initialisation

Constructor initialises interface with empty details

```
public Address() {
   // set up form
   entry=new JPanel(new GridLayout(Entry.MAXDETAILS,2));
   headings=new JLabel[Entry.MAXDETAILS];
   details=new JTextField[Entry.MAXDETAILS];
   for(int i=0;i<Entry.MAXDETAILS;i++) {
      headings[i]=setupLabel(text[i],entry);
      details[i]=setupTextField("",entry);
   }
   add(entry, BorderLayout.CENTER);
```

Interface & Initialisation

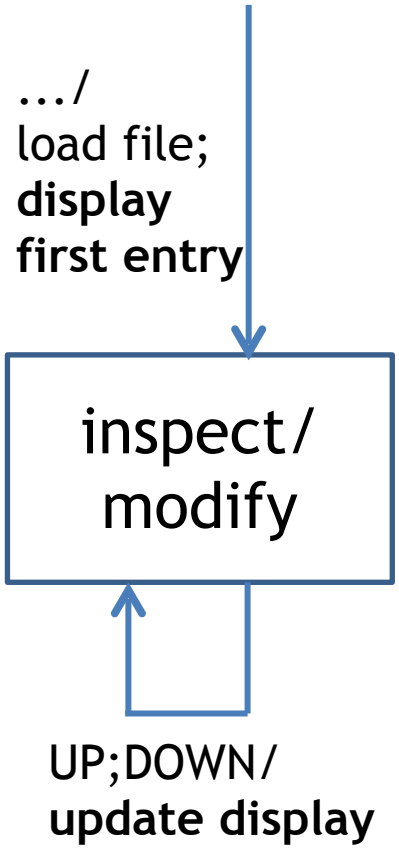
```
    // create buttons in control panel
    controls = new JPanel(new GridLayout(MAXEVENTS,1));
    actions = new JButton[MAXEVENTS];
    for(int i=0;i<MAXEVENTS;i++)
       actions[i]=
           setupButton(actionText[i],controls);
    // "Add" not valid in initial state, so disable it
    actions[1].setEnabled(false);
    add(controls, BorderLayout.WEST);
}
```



Interface & Initialisation

Method to display Entry from array in interface

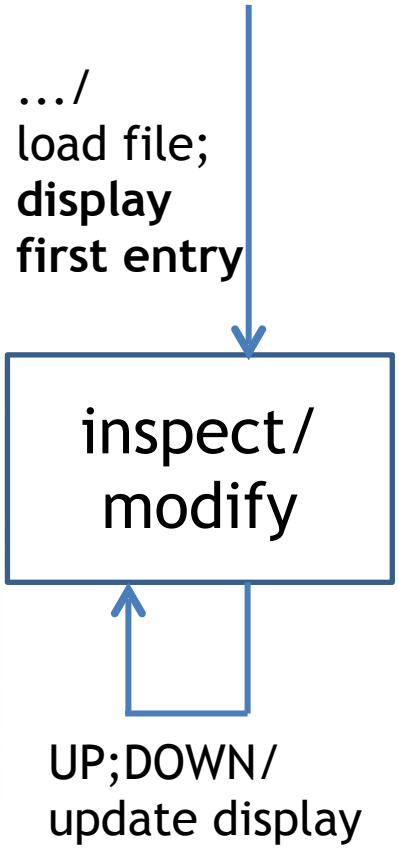
```
// display given entry in the GUI
void showEntry(Entry e) {
  for(int i=0;i<Entry.MAXDETAILS;i++)
    details[i].setText(e.details[i]);
}
```



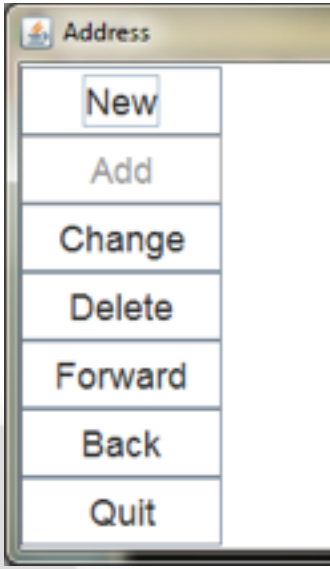
Interface & Initialisation

- at end of `readEntries`, need to set current `Entry` to first in array and display it

```
    ...
    file.close();
    if(entryno!=0) {
        current=0;
        showEntry(entries[0]);
    }
}
```



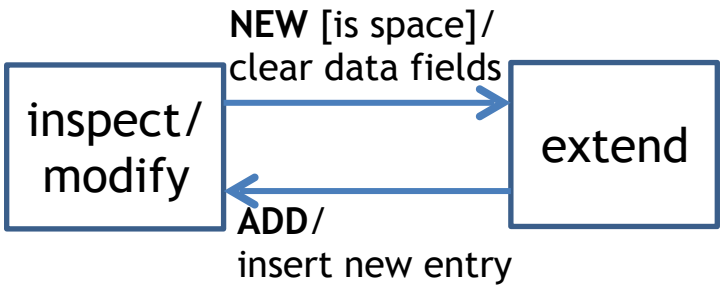
Quit control

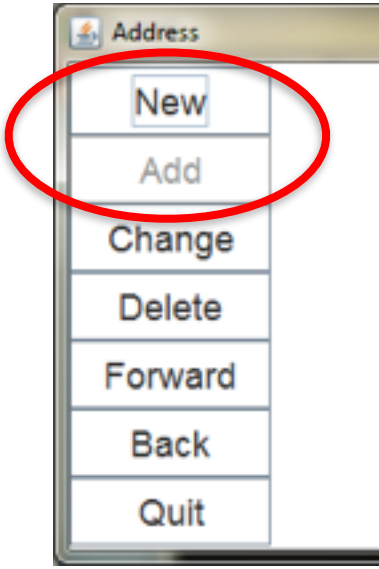


Write array back to file when exiting

```
void doQuit() {
   if(entryno==0) // if no entries
      System.exit(0);
   try {
      PrintWriter file =
         new PrintWriter(new FileWriter(addressbook));
      for(int i=0;i<entryno;i++) // save all entries
         entries[i].writeEntry(file);
      file.close();
      System.exit(0); // quit program
   }
   catch(IOException e){};
}
```

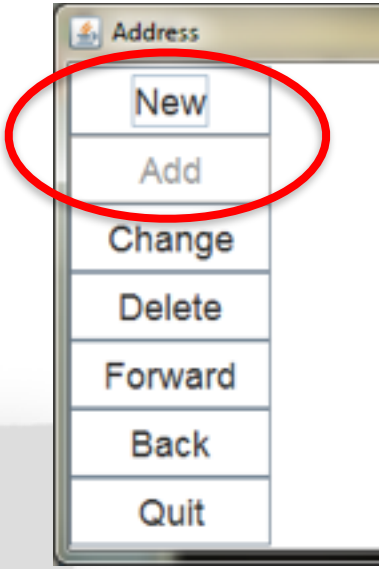
Add controls





- Two `JButton`s:
- *New*
  - clear all `JTextField`s
  - disable all `JButton`s apart from "Add"
  - user adds new details to empty `JTextField`s
- *Add*
  - copy details from `JTextField`s to new `Entry`
  - insert `Entry` into array in ascending name order
  - enable all `JButton`s and disable "Add"

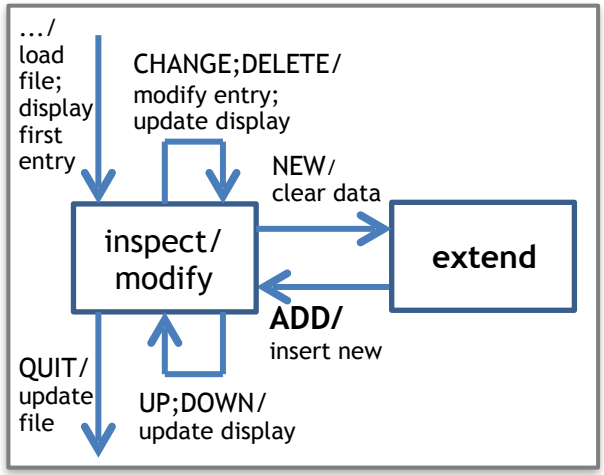
Add controls



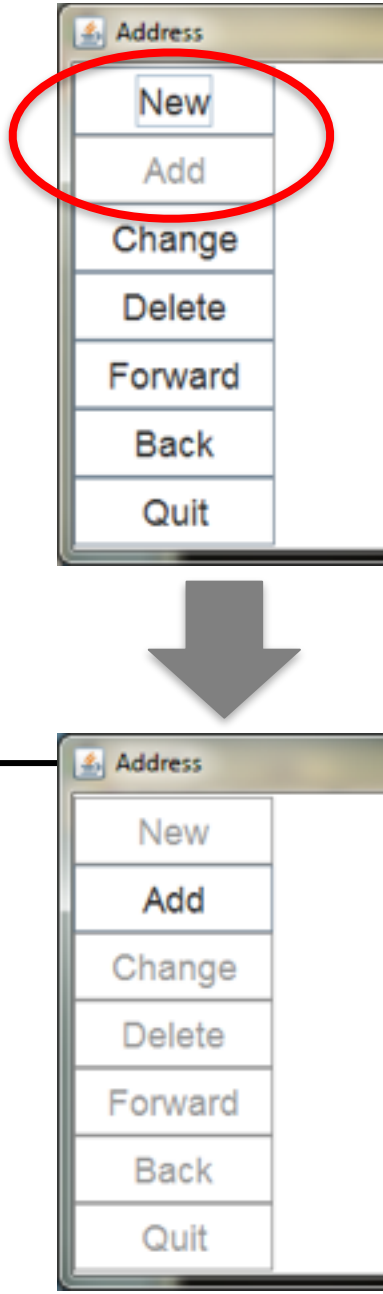
For *New*…

```
// prepare GUI for user to enter new entry
void doNew() {
    int i;
    // first check if space in array
    if(entryno==MAXENTRIES) {
        System.out.println("More than "+
                           MAXENTRIES+" entries.");
        return;
    }
      // set all input fields to be empty
      for(i=0;i<Entry.MAXDETAILS;i++)
         details[i].setText("");
```

Add controls

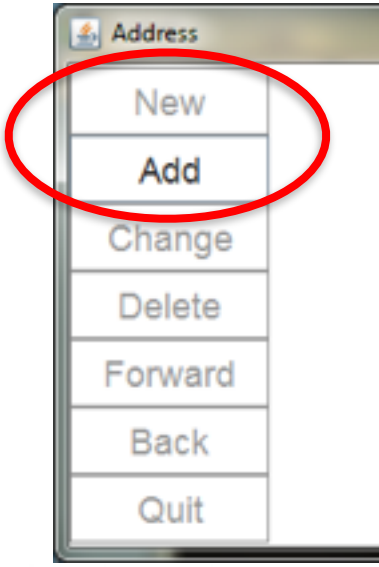


Moving to extend state, so enable *Add* and disable all other event sources




```
  actions[0].setEnabled(false);// disable new
  actions[1].setEnabled(true); // enable add
  for(i=2;i<MAXACTIONS;i++) // disable others
     actions[i].setEnabled(false);
}
```

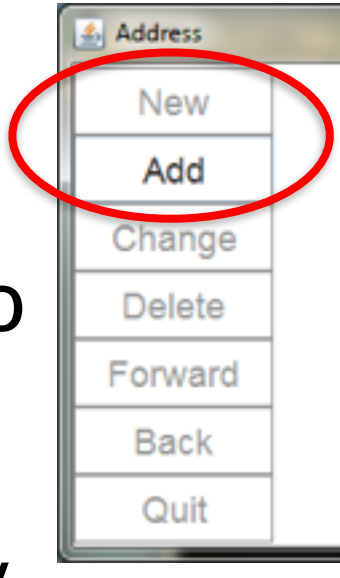
Add controls



- for *Add*...
  - get details from `JTextFields` and make new `Entry`

```
// create new entry using the details entered by user
void doAdd() {
    int i,j;
    String [] newdetails = new String[Entry.MAXDETAILS];
    // read from text fields
    for(i=0;i<Entry.MAXDETAILS;i++)
        newdetails[i]=details[i].getText();
     // create entry
    Entry e = new Entry(newdetails);
```

Add controls



We want to maintain ordered entries, so

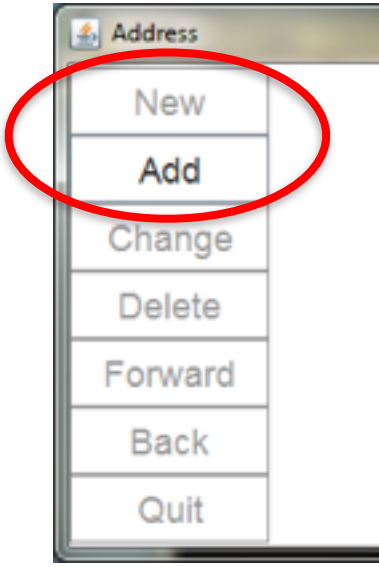
- search array to find first with name > new `Entry`'s name
- move all `Entry`s down one place in array

```
  // find the right alphabetical location in the array
  for(i=0;i<entryno;i++)
     if(e.details[0].compareTo(entries[i].details[0])<0)
         break;

// make space by moving remaining entries one place along
  for(j=entryno;j>i;j--)
    entries[j]=entries[j-1];
```

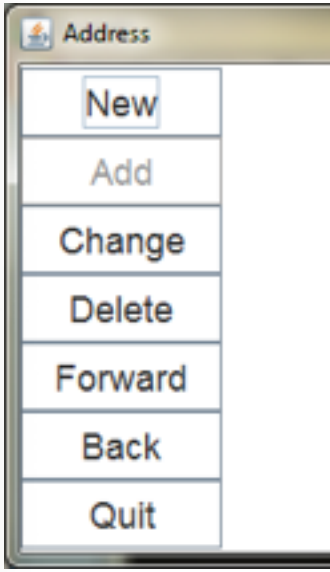
Add controls

- put new `Entry` in place
- update number of entries and current
- enable all `JButton`s and disable *Add*




```
// insert new entry into the resulting space
entries[i] = e;
current = i;
entryno++;

// disable Add and enable all other events
actions[0].setEnabled(true);
actions[1].setEnabled(false);
for (i = 2; i < MAXACTIONS; i++)
        actions[i].setEnabled(true);
```



Move controls

- for *Forward*...
  - check if at end of array
  - increment current and display `Entry`

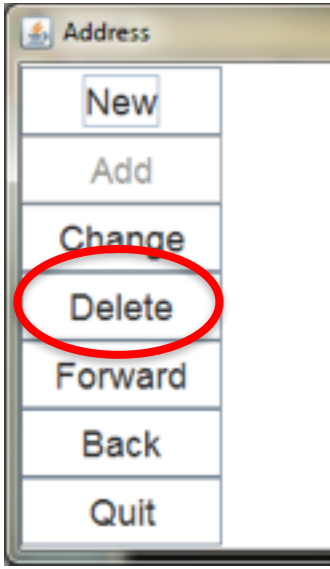
```
// move to next entry in address book
void doForward() {
   if(entryno==0 || current+1==entryno) // end of array?
      return;
   current++;
   showEntry(entries[current]);
}
```

Move controls

- for *Back*…
  - check if at start of array
  - decrement current and display `Entry`

```
// move to previous entry in address book
void doBack() {
   if(entryno==0 || current==0) // at beginning?
      return;
   current--;
   showEntry(entries[current]);
}
```

Delete control

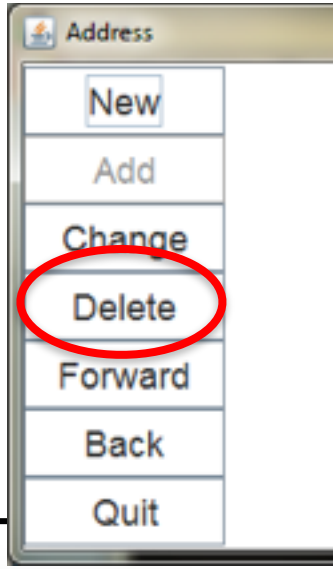


- for *Delete*…
  - move all `Entry`s after current back one place in array and decrement count

```
// remove current entry from address book
void doDelete() {
  if (entryno == 0) // ignore if no entries
        return;

  // move following entries back one place
  for (int i = current; i < entryno - 1; i++)
        entries[i] = entries[i + 1];
  entryno--; // decrement count
```

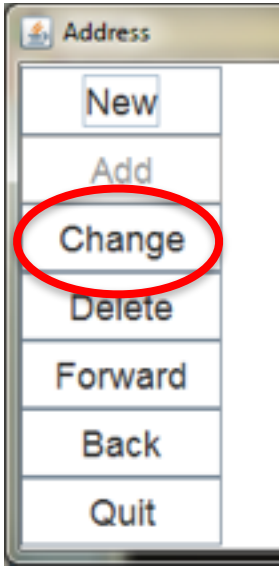
Delete control



- for *Delete*…
  - then update or clear display

```
// if address book now empty, reset display
  if (entryno == 0) {
        for (int i = 0; i < Entry.MAXDETAILS; i++)
              details[i].setText("");
        return;
  }
  if (current == entryno) // if last entry removed
        current--;
  showEntry(entries[current]); // update display
}
```

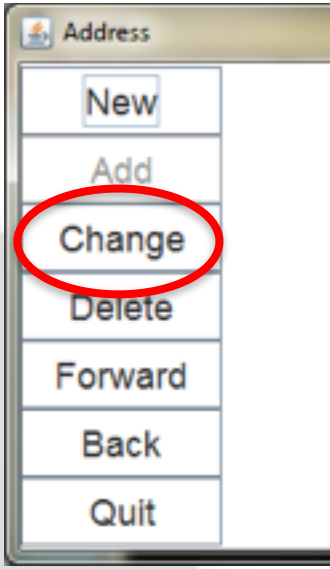
Change control



for *Change*…

- user can alter any text field at any time
  - only takes effect if *Change* selected
- can't just copy details back to current entry
  - if name changes, then entry order changes
    - delete current entry
    - can't use `doDelete` - updates text fields
    - add details from text fields as new entry - use `doAdd`

Change control




```
// update current entry
void doChange() {
    if (entryno == 0)  return;

    // delete current entry without displaying
    for (int i = current; i < entryno - 1; i++)
        entries[i] = entries[i + 1];
    entryno--;
    // update current if last entry "deleted"
    if (current == entryno)
        current--;

    // add as new entry in alphabetical position
    doAdd();
}
```

actionPerformed

```
public void actionPerformed(ActionEvent e)
{  switch(state)
   {  case INSPECT:
       if(e.getSource()==actions[0]) // NEW
       {  doNew();
          state = EXTEND; return;  }
       if(e.getSource()==actions[2]) // CHANGE
       {  doChange(); return;  }
       if(e.getSource()==actions[3]) // DELETE
       {  doDelete(); return;  }
```

actionPerformed

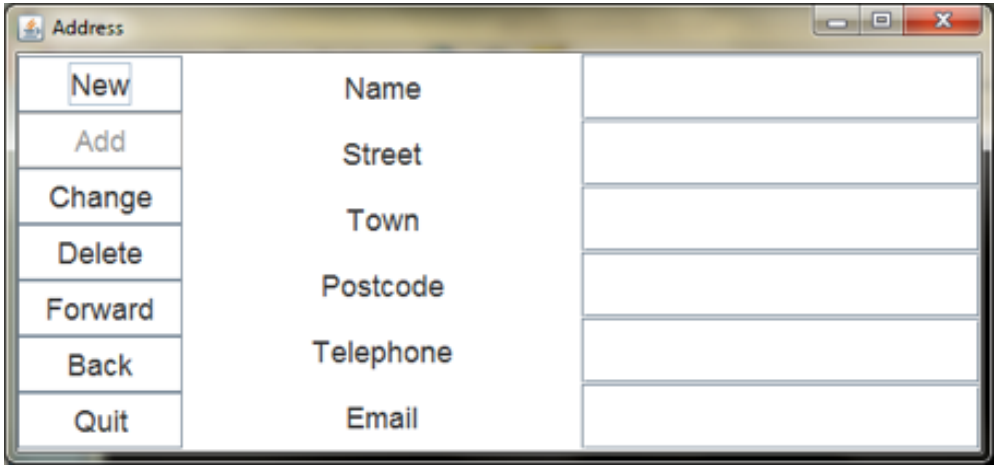
```
        if(e.getSource()==actions[4]) // UP
        {  doForward(); return;  }
        if(e.getSource()==actions[5]) // DOWN
        {  doBack(); return;  }
        if(e.getSource()==actions[6]) // QUIT
        {  doQuit();  return;  }

       case EXTEND:
        if(e.getSource()==actions[1]) // ADD
        {  doAdd();
           state = INSPECT; return;  }
      }
   }
}
```

main

```
class TestAddress
{  public static void main(String [] args) throws IOException
   {  Address a;
      a = new Address();
      a.setSize(600,280);
      a.setTitle("Address");
      a.setVisible(true);
      a.addWindowListener
        (new WindowAdapter()
         {  public void windowClosing(WindowEvent e)
            {  System.exit(0);  } });
      a.readEntries(); }
}
```

Program


Address
New
Add
Change
Delete
Forward
Back
Quit
Name
Street
Town
Postcode
Telephone
Email


Address
New
Add
Change
Delete
Forward
Back
Quit
Name
Allan
Street
Allans
Town
Alloa
Postcode
AA1 1AA
Telephone
01234567890
Email

Improvements/extensions

You might try improving this program, e.g.

- add a `JLabel` to display system messages
- after *NEW*, check that all requisite fields have been filled in before *ADD* can succeed
- add a search facility
- check for duplicate entries before *CHANGE* or *ADD* can succeed

THAT’S IT!

Next Week

- Exploring other Swing components
  - Text panes
  - Scroll bars
  - Menus
  - Dialogues