Understanding variables and Fields

- Places: variables vs fields
- Scope and Extent
- Visibility
- Encapsulation
- **final**
- Constants vs fields

Places: variables *vs* fields

- Two kinds of places to store information:
- Variables (including parameters)
  - defined inside a method
  - specify places on a worksheet
  - temporary – information is lost when worksheet is finished
  - new place created every time method is called (each worksheet)
  - only accessible from inside the method.

- Fields
  - defined inside a class, but not inside a method
  - specify places in an object
  - long term – information lasts as long as the object
  - new place created for each object
  - accessible from all methods in the class, and from constructor.

Extent and scope

- A place with a value must be accessible to some code at some time.
- **Extent**: how long it will be accessible
  - local variables (and parameters) in methods have a limited extent
    ⇒ only until the end of the current invocation of the method
  - fields have indefinite extent
    ⇒ as long as the object exists
- **Scope**: what parts of the code can access it
  - Full scope rules are complicated!!!
  - local variables: accessible only to statements
    - inside the block { … } containing the declaration
    - after the declaration
  - fields: at least visible to the containing class; maybe further.

Scope of variables

```
//read info from file and display
while (scan.hasNext() ){
      String ans = scan.next();
      if ( ans.equals("flower") ) {
            Color  center = Color.red;
            int diam = 30;
      }
      else if (ans.equals("bud") ) {
            Color  center = Color.green;
            int diam = 15;
      }
        :
      UI.setColor(center);
      UI.fillOval(x, y, diam, diam);
        :
}
```

different variables!

Out of scope

```
while (scan.hasNext() ){
      String ans = scan.next();
      Color center;
      int diam;
      if ( ans.equals("flower") ) {
            center = Color.red;
            diam = 15;
      }
      else if (ans.equals("bud") ) {
            center = Color.blue;
            diam = 30;
      }
        :
      UI.setColor(center);
      UI.fillOval(x, y, diam, diam);
        :
}
```

may not be intialised

How do you fix it?

Fields: scope, visibility, encapsulation

- Fields are accessible to all code in all the (ordinary) methods in the class.
- Should they be accessible to methods in other classes?
  - ⇒ **visibility**: **public** or **private**
  - **public** means that methods in other classes can access the fields

    cfg1.figX = 30 in the CartoonStory class would be OK
  - **private** means that methods in other classes **cannot** access the fields

    cfg1.figX = 30 in the CartoonStory class would be an error.

The principle of encapsulation says

- Keep fields private.
- Provide methods to access and modify the fields, if necessary

⇒ LDC 5.3

Final: fields that don't vary

- If a place will hold a value that should not change (a "constant"):
  - signal it to reader
  - ensure that no code changes it by mistake
- **final** is a modifier on field or variable declarations
  - means that it can only be assigned to once.

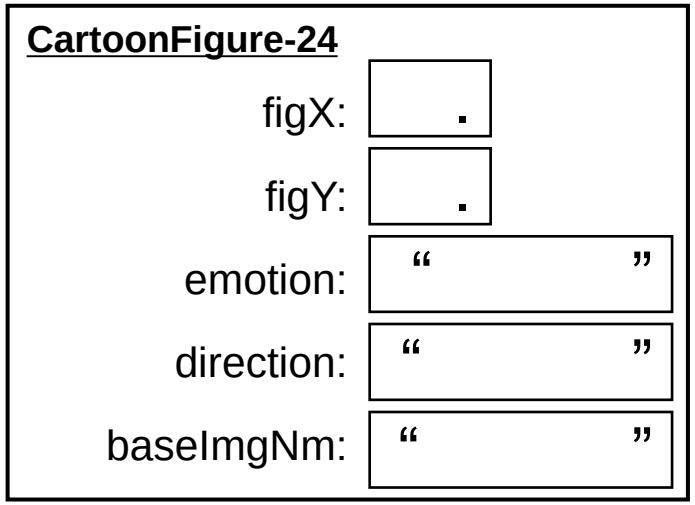
```
public class CartoonFigure {
    private double figX,  figY;
    private String direction = "right";
    private String emotion = "smiling";
    private final String imagePrefix;
    private final double wd = 40
    private final double  ht = 80;

    public CartoonCharacter(double x, double y, String folder ){
        this.imagePrefix = img   // fine – this is the first assignment
        this.wd = 50;            // NO!!! Can't change the previous value
```

public static final: class wide constants

- Constants: **public static final** fields
  - **public** – can be accessed by code outside this class
  - **static** – single place belonging to the class, not a separate place for each object
  - **final** – value can't be changed once assigned

```
public class CartoonFigure {
    private double figX;
    private double figY;
    private String direction = "right";
    private String emotion = "smiling";
    private final String baseImgNm;
    public static final double WD = 40
    public static final double HT=80;
```

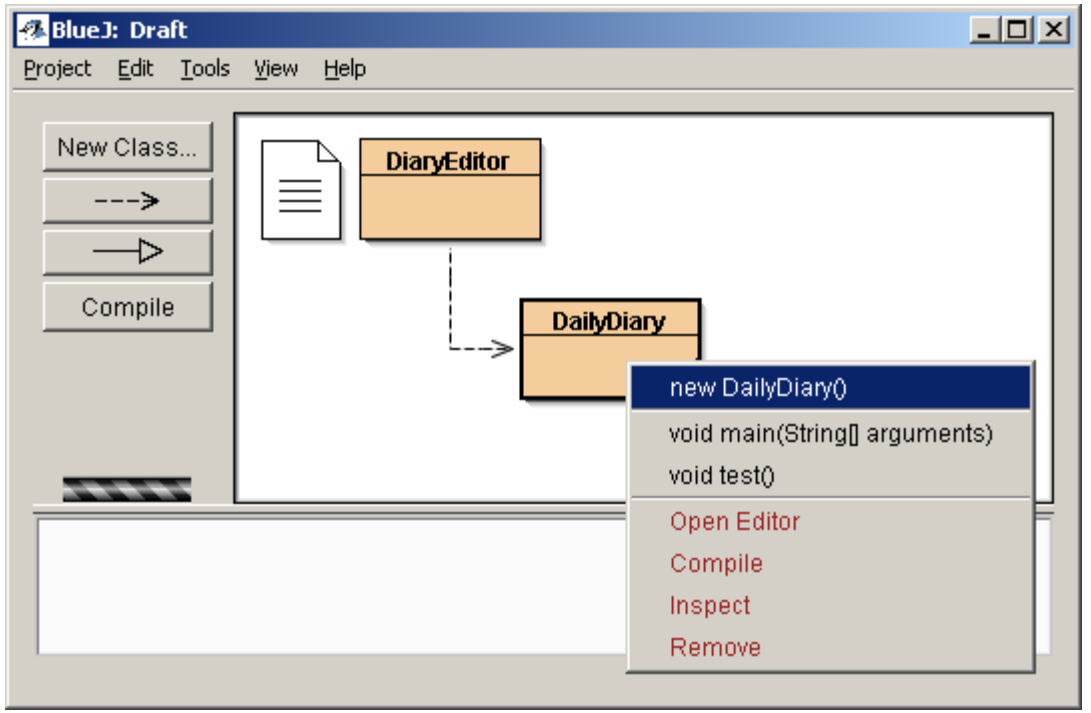


GUI's and Event driven input

- In a GUI, the interaction is controlled by the user, not by the program

- User initiates "events"
  - buttons
  - menus
  - mouse press/release/drag
  - text fields
  - sliders
  - keys

- Program responds



Buttons using the ecs100 library

```
public class MyClass {
    public void setupGUI(){
        UI.addButton("Clear", UI::clearGraphics);
        UI.addButton("Go", this::runFireworks);
        UI.addButton("Quit", UI::quit);
    }

    public void runFireworks(){
        …….
    }

    public static void main(String[ ] args){
        MyClass mc = new MyClass();
        mc.setupGUI();
    }
}
```

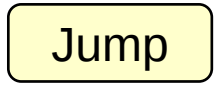
More kinds of events.

- Buttons
- Text fields
- Menus
- Mouse press/release/drag
- Sliders
- Keys
- …..
- How does Java respond to events etc?
  - When event occurs (button pressed / text entered in box / slider changed / mouse clicked/…)
    - Java looks up the object & method attached to the event (the "listener")
    - Calls the method on the object
      - passing any information involved in the event as arguments.

Event driven input:

Simplest event: "do it"

- Buttons:
  - must specify what method to call on what object
  - no further information available



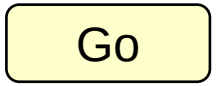
Events with information attached

- TextFields:
  - user enters a text value
  - must specify the method to call, and
  - ensure that the text value gets passed to the method

- Mouse events:
  - presses, releases, clicks, drags, moves
  - must specify what method to call
  - ensure the kind of action and the position of the mouse gets passed to the method.

Setting up event-driven input

- Setting up the GUI:
  - To add a button to the UI:

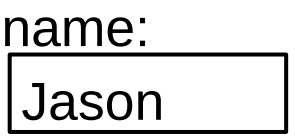


    - specify name of button and method to call (*object*::*method* or *class*::*method*) (must be a method with no parameters)

```
eg:  UI.addButton("Go", this::startGame);
     UI.addButton("End", UI::quit);
```

```
public void startGame(){…..
```

  - To add a textfield to the UI:
    - Specify name of textfield and method to call (must be a method with one String parameter)




```
eg   UI.addTextField("name", this::setName);
```

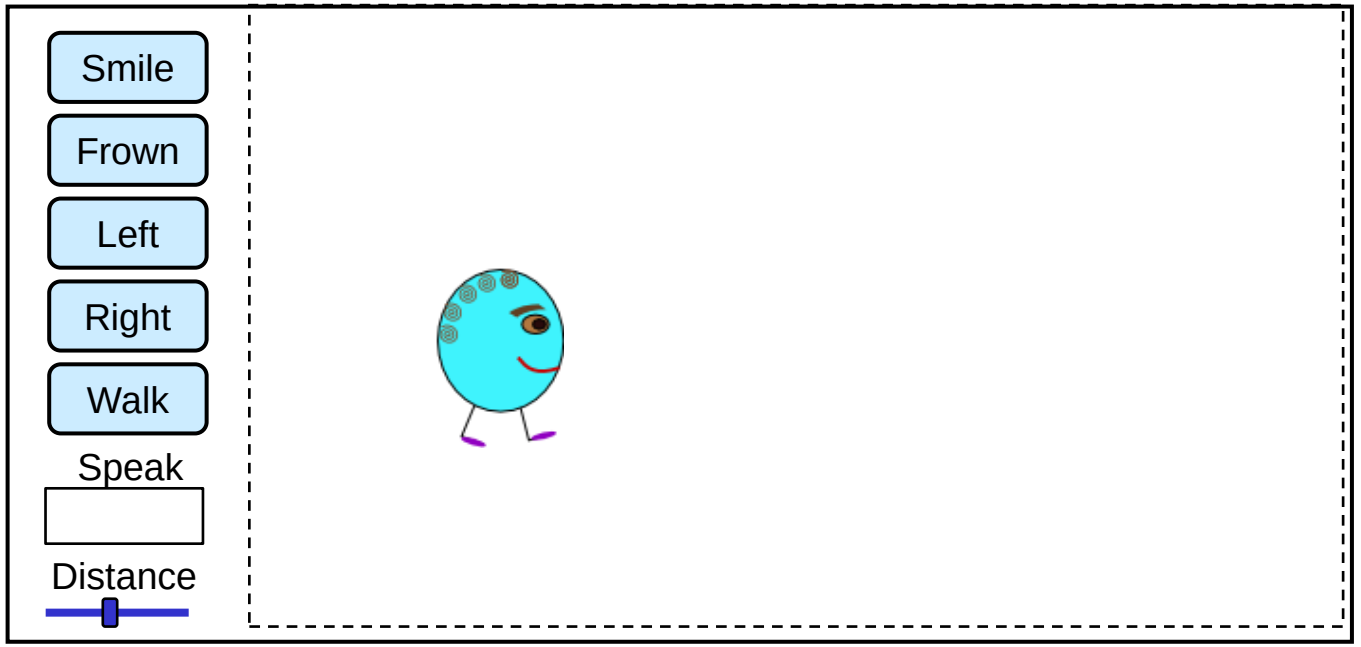
```
public void setName(String n){…..
```

  - To add a slider to the UI:
    - Specify name of slider, min, max, initial values, and method to call (must be a method with one double parameter)


```
eg   UI.addSlider("speed", 10, 50, 20, this::setSpeed); );
```

```
public void setSpeed(double v){…..
```

PuppetMaster

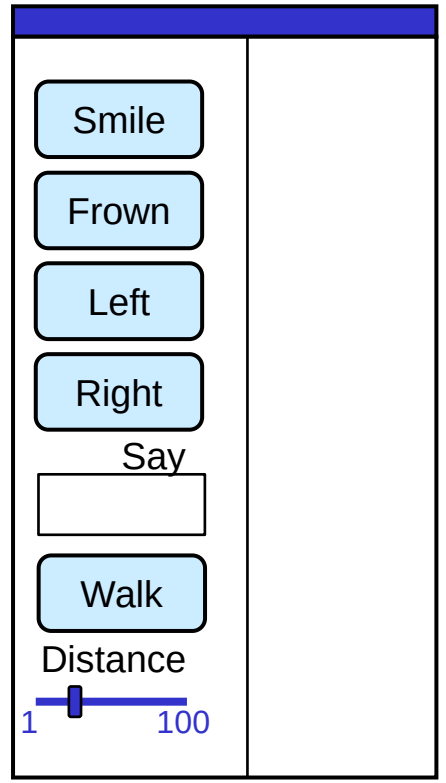


PuppetMaster: setting up Buttons etc

```
public class PuppetMaster   …   {
   // fields

   /** set up the GUI */
   public void setupGUI (){
      UI.addButton( "Smile", this::doSmile);
      UI.addButton( "Frown", this::doFrown);
      UI.addButton( "Left", this::doLeft);
      UI.addButton( "Right", this::doRight);
      UI.addTextField( "Say", this::doSpeak);
      UI.addButton( "Walk", this::doWalk);
      UI.addSlider( "Distance", 1, 100, 20, this::setDist);
            …
   }
   // methods to respond

   public static void main (String[ ] args){
      new PuppetMaster().setupGUI();
}
```



Responding to buttons and textFields

```
public class PuppetMaster {
    public void doSmile(){
        // tell the CartoonCharacter to smile
    }
    public void doFrown(){
        // tell the CartoonCharacter to frown
    }
    public void doSpeak(String words){
        // tell the CartoonCharacter to say the words
    }
    public void setDist(double value){
        // remember the value
    }
    public void setupGUI(){
        UI.addButton("Smile", this::doSmile);
        UI.addButton("Frown", this::doFrown);  …..
        UI.addTextField("Say", this::doSpeak);
        UI.addSlider( "Distance", 1, 100, 20, this::setDist);
    }
```

Methods called by buttons must have no parameters

Methods called by a textField must have one String parameter

Methods called by a slider must have one double parameter

Event driven input and fields

- Each event will make a new method call.
- $\Rightarrow$ can't remember anything between events in local variables in the methods.

- Typically, need fields in the object to remember information between events.

  - eg: PuppetMaster has to remember the CartoonCharacter object in a field

PuppetMaster: Design

Structure of the PuppetMaster class:

```
public class PuppetMaster  {

    // fields to store values between events/method calls
    private ….

    // set up GUI
    public void setupGUI() {
        // set up the buttons, slider, textField, to call methods on the object
    }

    // methods to respond to the buttons,  slider, textField
    public  void …


    public static void main (String[] args){
        // make a PuppetMaster object and call setupGUI
    }
```

PuppetMaster: Using Fields

Actions on the CartoonCharacter happen in response to different events

⇒ will be in different method calls

⇒ need to store character in a field, not a local variable.

```
public class PuppetMaster{
    // fields
    private CartoonCharacter cc = new CartoonCharacter(200, 100, "blueguy");

    public void doSmile(){
        this.cc.smile();
    }
    public void doFrown(){
        this.cc.frown();
    }
    public void setupGUI(){
        UI.addButton("Smile", this::doSmile);
        UI.addButton("Frown", this::doFrown);
          ⋮
    }
```

PuppetMaster: TextFields (boxes)

```
public  class  PuppetMaster{
    private CartoonCharacter cc = new CartoonCharacter(200, 100, "blueguy");

    public void doSmile(){
        this.cc.smile();
    }
            :
    public void doSpeak(String words){
        this.cc.speak(words);
    }

    public void setupGUI(){
        UI.addButton("Smile", this::doSmile);
        UI.addButton("Frown", this::doFrown);

        UI.addTextField("Say", this::doSpeak);
    }
```

PuppetMaster: Sliders

```
public  class  PuppetMaster {
    private CartoonCharacter cc = new CartoonCharacter(200, 100, "blueguy");
    private double walkDist = 20 ;

    public void doWalk() {
        this.cc.walk(this.walkDist);
    }
    public void setDist(double value){
        this.walkDist = value;
    }
    public void setupGUI(){
        UI.addButton("Smile", this::doSmile);
        UI.addButton("Frown", this::doFrown);

              :
        UI.addButton("Walk", this::doWalk);
        UI.addSlider( "Distance", 1, 100, 20, this::setDist);
    }
              .
```

Typical design:
field to store value
from one event,
for use by another event

A method called by
a slider must have
one double parameter

PuppetMaster: Using Fields

Listeners in the buttons etc don't *have* to call methods on this or UI:

```
public  class  PuppetMaster{
    // fields
    private CartoonCharacter cc = new CartoonCharacter(200, 100, "blue");
    // constructor
    public void setupGUI(){
        UI.addButton("Smile",  this::doSmile);
        UI.addButton("Frown", this::doFrown);
            :
    }
    public void doSmile(){
        this.cc.smile();
    }
    public void doFrown(){
        this.cc.frown();
    }
```

GUI: Mouse input

- Just like buttons, except don't have to put anything on screen
  - Each press / release / click on the graphics pane will be an event
  - Must tell UI the listener: the object::method to call when a mouse event occurs

```
UI.setMouseListener(game::doMouse);
```

  - Must define method to say how to respond to the mouse.
    parameters: kind of mouse event and position of mouse event

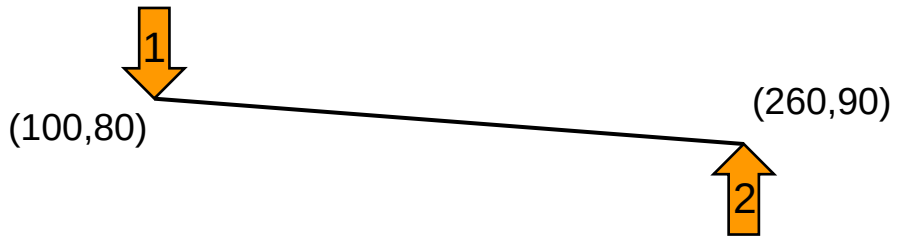
```
public void doMouse(String action, double x, double y) {
    if (action.equals("pressed") ) {
        // what to do if mouse button is pressed
    }
    else if (action.equals("released") ) {
        // what to do if mouse button is released
    }
    else if (action.equals("clicked") ) {
        // what to do if mouse button is clicked
    }
}
```

where action occurred

press-release in same place

Using the mouse.

- Want to let user specify input with the mouse,
  - eg: drawing lines



- Typical pattern:
  - On "pressed",
    - just remember the position
  - On "released",
    - do something with remembered position and new position

Mouse Input

```
public class LineDrawer {      /**Let user draw lines on graphics pane with the mouse. */
    private double startX, startY;  // fields to remember "pressed" position
    public void setupGUI(){
        UI.setLineWidth(10);
        UI.setMouseListener(this::doMouse);
        UI.setDivider(0.0);
    }
    public void doMouse(String action, double x, double y) {
        if (action.equals("pressed") ) {
            this.startX = x;
            this.startY = y;
        }
        else if (action.equals("released") ) {
            UI.drawLine(this.startX, this.startY,  x,  y);
        }
    }
```

Mouse Input

Simple mouse events: UI.setMouseListener(this::doMouse);

- pressed
- released
- clicked

Mouse movement: UI.setMouseMotionListener(this::doMouse);

- pressed, released, clicked
- dragged
- moved

Selecting Colors: JColorChooser

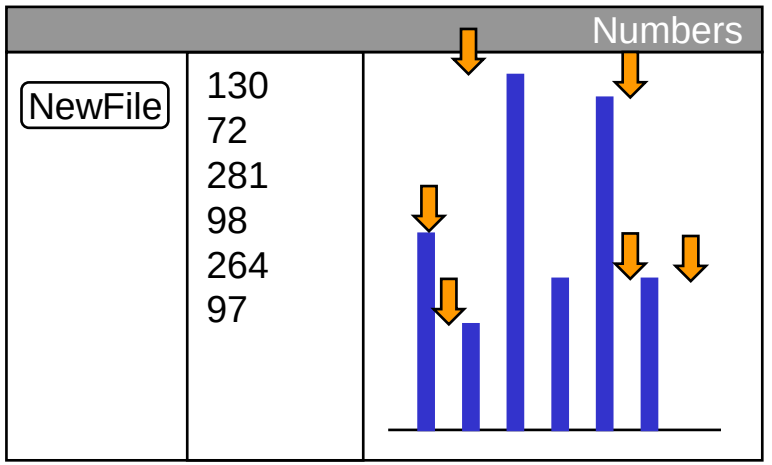
```
public class LineDrawer {

    private double startX, startY;  // fields to remember "pressed" position
    private Color currentColor = Color.black;

    public void doMouse(String action, double x, double y) {
        if (action.equals("pressed") )          {   this.startX = x;  this.startY = y;  }
        else if (action.equals("released") ) {   UI.drawLine(this.startX, this.startY,  x,  y);  }
    }
    public void doChooseColour(){
        this.currentColor = JColorChooser.showDialog(null, "Choose Color", this.currentColor);
        UI.setColor(this.currentColor);
    }
    public static  void main(String[] args){
        UI.setLineWidth(10);
        LineDrawer drawer = new LineDrawer();
        UI.setMouseListener(drawer::doMouse);
        UI.addButton("Color", drawer::doChooseColour);
    }
```

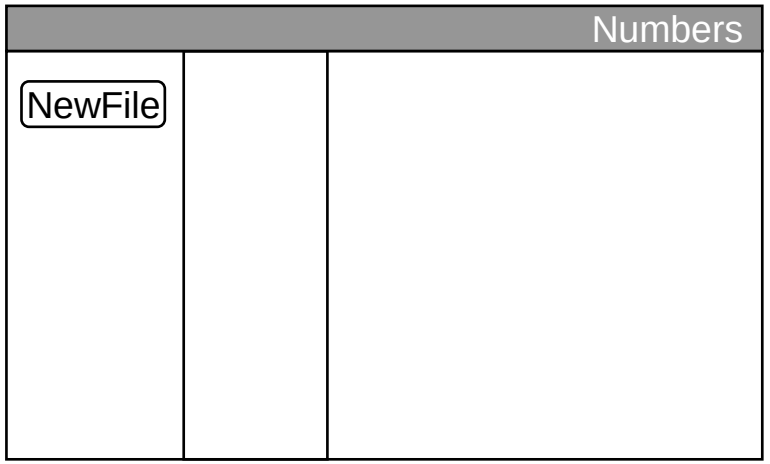
Numbers program

- Program for constructing files of numbers:
  - Allow user to select a new file
  - Allow user to enter a set of numbers with the mouse (height of mouse click is the number)
  - Display numbers as bar chart and list in text pane
  - Save numbers to the file as they are entered
- User Interface:
  - Button to clear screen and select new file.
  - Graphics pane to select (with mouse) and display the numbers
  - Text pane to display list of numbers



Numbers: Design

- Design:
  - When does something happen?
    - button presses
    - mouse clicks
  - Fields
    - to store the file (PrintStream) that the numbers are being saved to
    - to remember the horizontal position of the next bar.
  - Methods to respond to mouse
    - record a new number
  - Method to respond to button
    - clear and start a new file
  - main method
    - create object
    - set up the interface



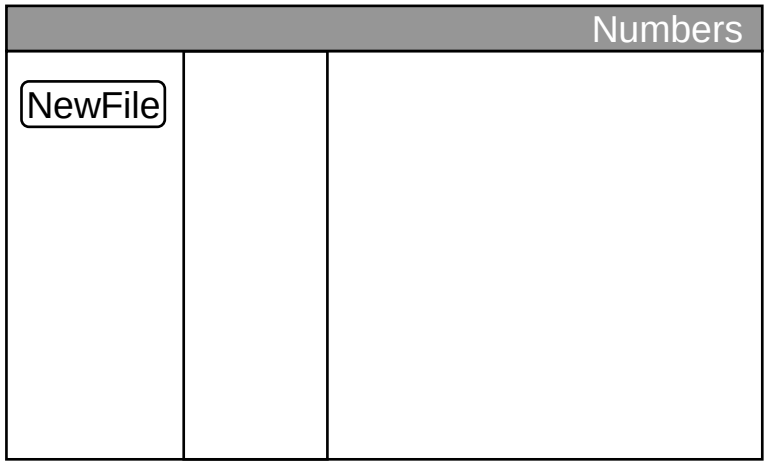
Numbers: Design

```
public class Numbers {
    private PrintStream output;
    private double barX = 0;
    private static final double BASE= 450;

    public void doNew() {…


    public void doMouse( …

    public static void main(String[ ] args){
        Numbers num = new Numbers();
        UI.setMouseListener(num::doMouse);
        UI.addButton("NewFile", num::doNewFile);
        UI.drawLine(0, BASE, 600, BASE);
    }
}
```

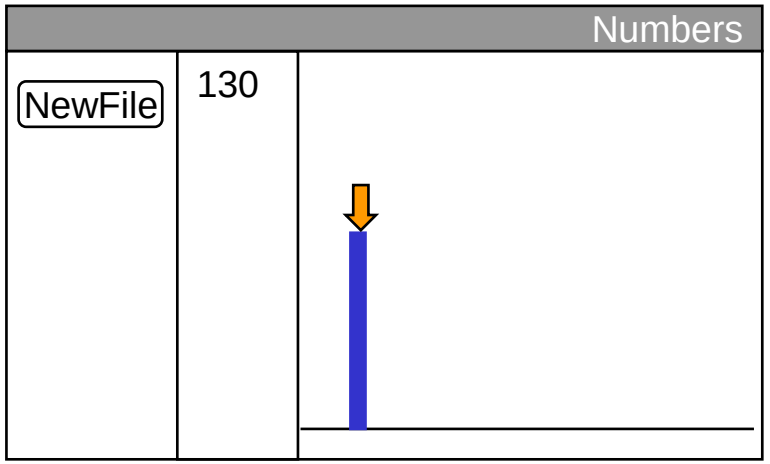


Respond to Mouse:

- When user clicks/releases:
  - work out the number they meant
  - draw a bar on the graphics pane
  - display it in the text pane
  - print it to the file

```
public void doMouse(String action, double x, double y) {
    if (action.equals("released")) {
        double number = BASE - y;
        this.barX = this.barX + 10;
        UI.fillRect(this.barX, y, 5, number);
        UI.println(number);
        this.output.println(number);
    }
}
```

What's the problem?



Respond to "NewFile" button

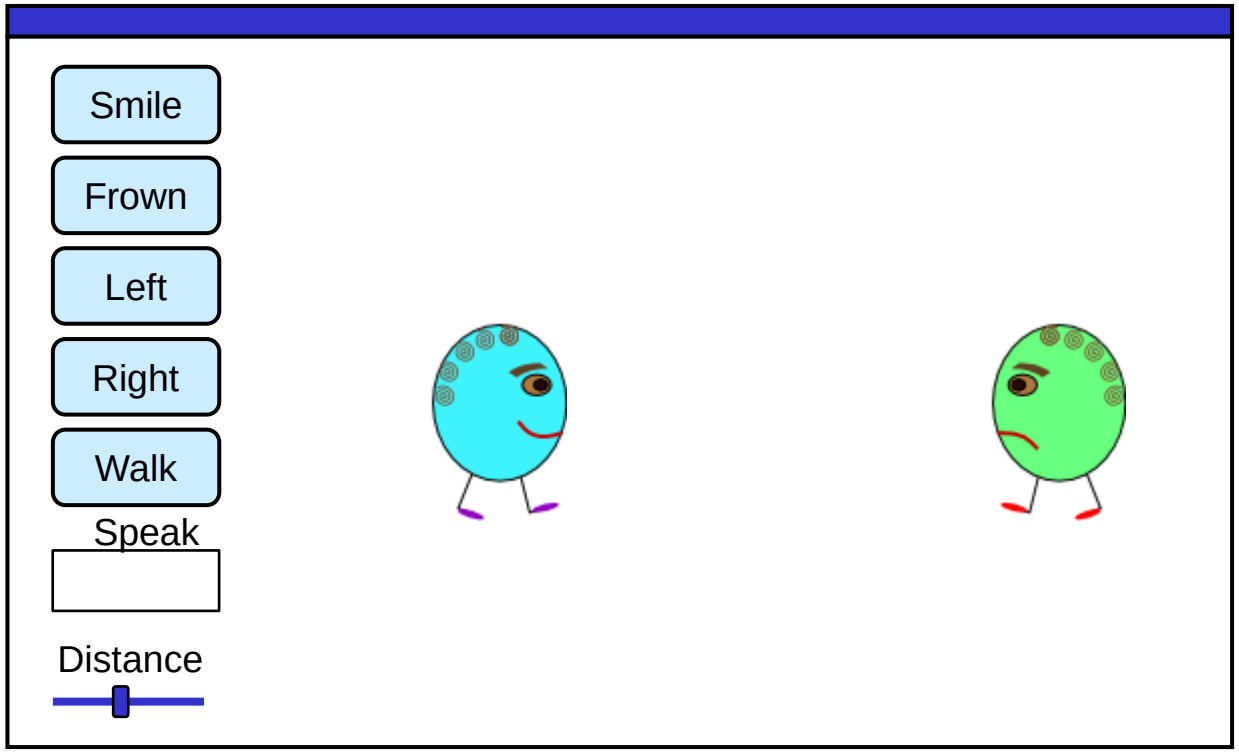
```
public void doNewFile(){
    UI.clearPanes();
    UI.drawLine(0, BASE, 600, BASE);
    this.barX = 0;
    this.output.close();
    try{
        this.output = new PrintStream(UIFileChooser.save());
    } catch(IOException e) { UI.println("File error: "+e); }
}
```

Still a problem!

```
if (this.output != null) {
    this.output.close();
}
```

GUI design: choosing object to act on

Suppose we have two characters!



Problem:

- Which character should smile/turn/walk/speak?

- Event-driven input can be tricky!

GUI design: choosing object to act on

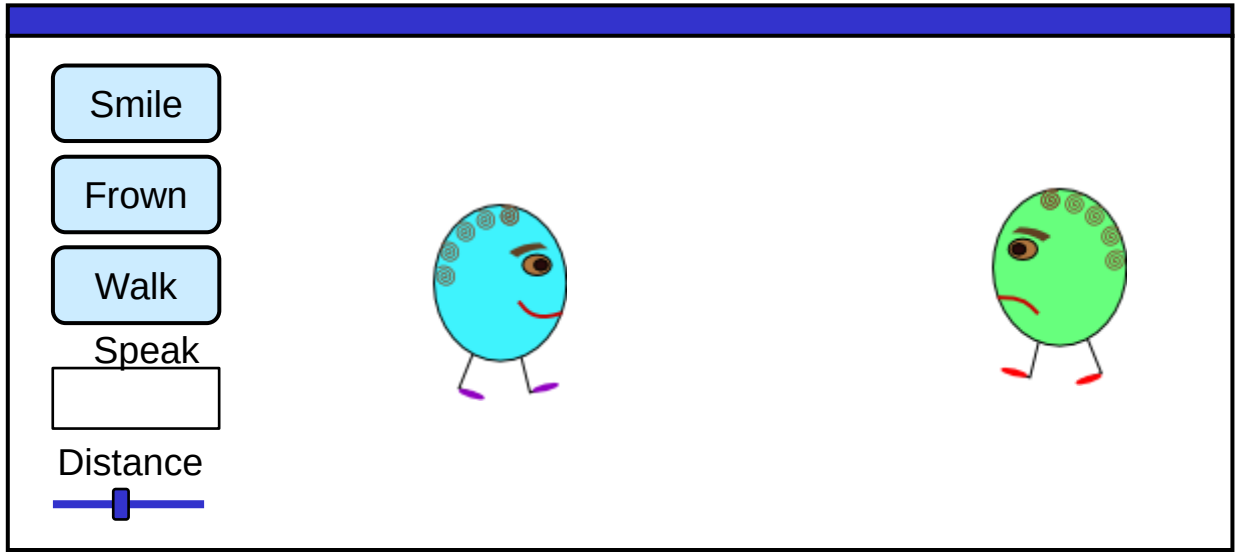
- One typical simple GUI interaction mechanism
  1. Select object you want to act on
  2. Choose action.

- Must remember the currently selected object:
  - in a field, because the action will be performed in a later method

```
this.selectedCC = cc1;
```

- Typically, the “selected object” doesn’t change until user selects another object.

PuppetMaster: two characters



PuppetMaster-3

cc1: CartoonCharacter-11

cc2: CartoonCharacter-12

selectedCC: CartoonCharacter-12

walkDistance: 20

CartoonCharacter-11

figX: 110 emotion: "smile"

figY: 200 direction: "right"

imgBaseName: "blue"

CartoonCharacter-12

figX: 350 emotion: "frown"

figY: 200 direction: "left"

imgBaseName: "green"

PuppetMaster: selecting a character.

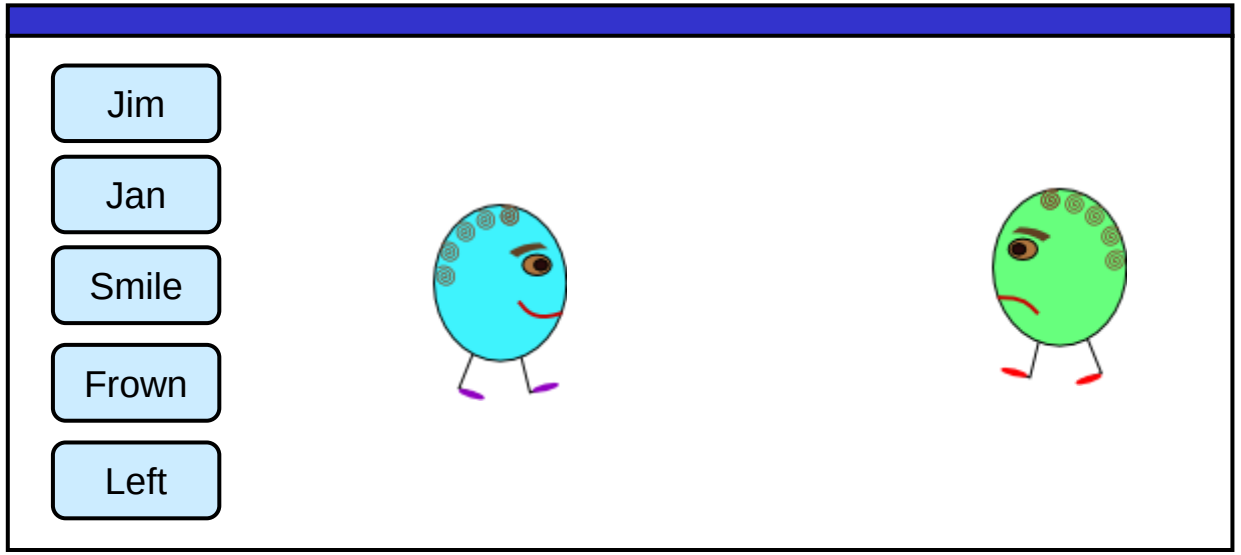
```
public class PuppetMaster{
    private CartoonCharacter  cc1= new CartoonCharacter(100, 100, "blueguy");
    private CartoonCharacter  cc2= new CartoonCharacter(500, 100, "greenguy" );
    private CartoonCharacter  selectedCC = cc1;   // the selected one
    private double walkDistance = 20;
    public void setupGUI(){
        UI.addButton( "Smile", this::doSmile);
            ⋮
    }
    public void doSmile(){
        this.selectedCC.smile();
    }
    public void doFrown(){
        this.selectedCC.frown();
    }
    public static void main (String[] args){
        new PuppetMaster().setupGUI();
    }
```

How do we change the selected character?

PuppetMaster: buttons for selecting

```
public void doJim() {
    this.selectedCC = this.cc1;
}
public void doJan() {
    this.selectedCC = this.cc2;
}
public void doSmile(){
    this.selectedCC.smile();
}
public void doWalk() {
    this.selectedCC.walk(this.walkDistance );
}
public static void main (String[] args){
    PuppetMaster pm = new PuppetMaster();
    UI.addButton( "Jim", this::doJim);
    UI.addButton( "Jan", this::doJan);
    UI.addButton( "Smile", this::doSmile);
        :
}
```

PuppetMaster: two characters



PuppetMaster-3

cc1: CartoonCharacter-11

cc2: CartoonCharacter-12

selectedCC: CartoonCharacter-12

CartoonCharacter-11

figX: 110 emotion: "smile"

figY: 200 direction: "right"

imgBaseName: "blue"

CartoonCharacter-12

figX: 350 emotion: "frown"

figY: 200 direction: "left"

imgBaseName: "green"

Which objects can be in the "listeners"

```
public class PuppetMaster{
    private CartoonCharacter cc1 = new CartoonCharacter(200, 100, "blue");

    public PuppetMaster(){
        UI.addButton("Smile", this::doSmile);
        UI.addButton("Frown", this::doFrown);
        UI.addTextField("Say", this::doSpeak);
            :
    }
    public void doSmile(){
        this.cc1.smile();
    }
    public void doFrown(){
        this.cc1.frown();
    }
    public void doSpeak(String words){
        this.cc1.speak(words);
    }
```

Lots of typing for just one line

Saving unnecessary methods:

```
public class PuppetMaster{
    private CartoonCharacter cc1 = new CartoonCharacter(200, 100, "blue");

    public PuppetMaster(){
        UI.addButton("Smile", this.cc1::smile );
        UI.addButton("Frown", this::doFrown);
        UI.addTextField("Say", this::doSpeak);
            :
    }
    public void doSmile(){
        this.cc1.smile();
    }
    public void doFrown(){
        this.cc1.frown();
    }
    public void doSpeak(String words){
        this.cc1.speak(words);
    }
```

Problem: Button remembers the object!!

```
public class PuppetMaster{
    private CartoonCharacter cc1 = new CartoonCharacter(200, 100, "blueguy");
    private CartoonCharacter cc2 = new CartoonCharacter(500, 100, "greenguy");
    private CartoonCharacter selectedCC = cc1;

    public PuppetMaster(){
        UI.addButton("Jan", this::doJan);
        UI.addButton("Smile", this.selectedCC::smile );
        UI.addButton("Frown", this::doFrown);
            :
    }
    public void doJan(){
        this.selectedCC = this.cc2;
    }
    public void doSmile(){
        this.selectedCC.smile();
    }
    public void doFrown(){
```

Doesn't work!!!

The button remembers the object in this.cc1 at the time the button was created!!!!

Shorthand: "Lambda expressions"

```
public class PuppetMaster{
    private CartoonCharacter cc1 = new CartoonCharacter(200, 100, "blueguy");
    private CartoonCharacter cc2 = new CartoonCharacter(500, 100, "greenguy");
    private CartoonCharacter selectedCC = cc1;

    public PuppetMaster(){
        UI.addButton("Jan", this::doJan);
        UI.addButton("Smile",  () -> { this.cc1.smile(); } );
        UI.addButton("Frown", this::doFrown);
            :
    }
    public void doJan(){
        this.selectedCC = this.cc2;
    }
    public void doSmile(){
        this.selectedCC.smile();
    }
    public void doFrown(){
```

Lambda Expression:
Anonymous methods!!
- has parameters
- has body
- but no name

It is a value!!

Shorthand: "Lambda expressions"

```
public class PuppetMaster{
    private CartoonCharacter cc1 = new CartoonCharacter(200, 100, "greenguy");
    private CartoonCharacter cc2 = new CartoonCharacter(500, 100, "blueguy");
    private CartoonCharacter selectedCC = cc1;
    private double walkDist = 20;

    public PuppetMaster(){
        UI.addButton("Jim",      () -> {  this.selectedCC = this.cc1;  }  );
        UI.addButton("Jan",      () -> {  this.selectedCC = this.cc2;  }  );
        UI.addButton("Smile",    () -> {  this.selectedCC.smile();  }  );
        UI.addButton("Frown",    () -> {  this.selectedCC.frown();  }  );
        UI.addButton( "Left",    () -> {  this.selectedCC.lookLeft();  }  );
        UI.addButton( "Right",   () -> {  this.selectedCC.lookRight();  }  );
        UI.addTextField("Say", (String wds) -> {  this.selectedCC.speak(wds);  }  );
        UI.addButton("Walk",     () -> {  this.selectedCC.walk(this.walkDist);  }  );
        UI.addSlider( "Distance", 1, 100, this.walkDist,  (double val) -> {  this.walkDist = val;  }    );
    }
```

You do NOT HAVE TO USE THESE!!
It is always safe to have an explicit, named method.