CS1013 – Programming Project

Dr. Gavin Doherty
ORI LG.19
Gavin.Doherty@cs.tcd.ie

Trinity College Dublin

DSG
Distributed Systems Group

Button Widget class

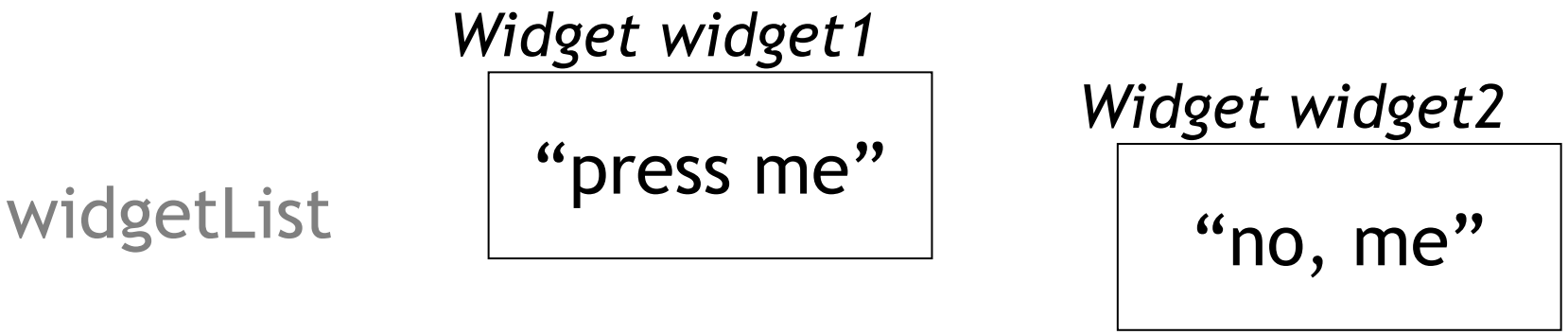
```
class Widget {
  int x, y, width, height;
  String label; int event;
  color widgetColor, labelColor;
  PFont widgetFont;

  Widget(int x,int y, int width, int height, String label,
  color widgetColor, PFont widgetFont, int event){
    this.x=x; this.y=y; this.width = width; this.height= height;
    this.label=label; this.event=event;
    this.widgetColor=widgetColor; this.widgetFont=widgetFont;
    labelColor= color(0);
   }
  void draw(){
    fill(widgetColor);
    rect(x,y,width,height);
    fill(labelColor);
    text(label, x+10, y+height-10);
  }
  int getEvent(int mX, int mY){
     if(mX>x && mX < x+width && mY >y && mY <y+height){
        return event;
     }
     return EVENT_NULL;
  }
}
```

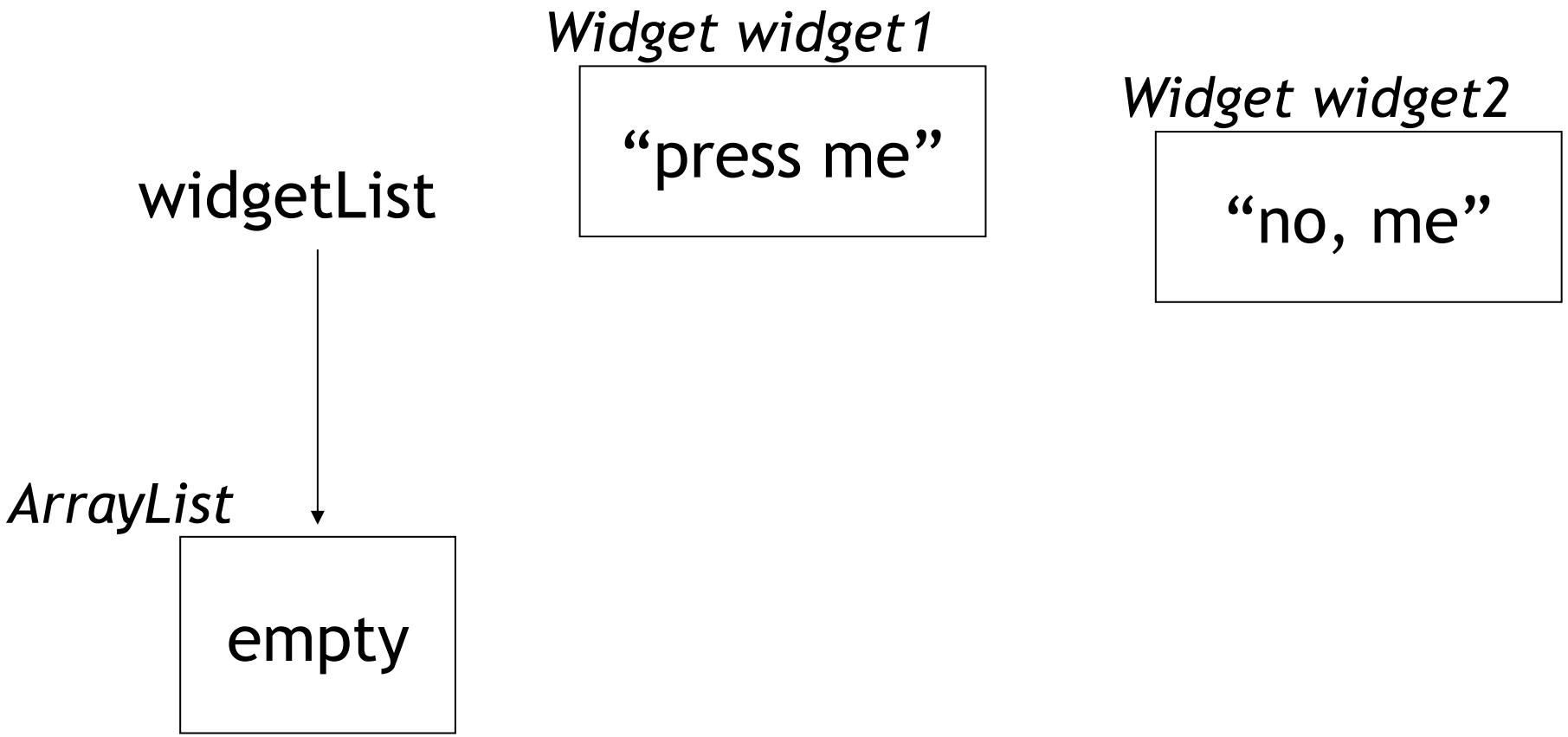
ArrayList of Widgets

```
ArrayList widgetList;
PFont stdFont;
final int EVENT_BUTTON1=1;
final int EVENT_BUTTON2=2;
final int EVENT_NULL=0;
void setup(){
  Widget widget1, widget2;
  size(400, 400);
  stdFont=loadFont("Chiller-Regular-36.vlw"); textFont(stdFont);
  widget1=new Widget(100, 100, 100, 40,
                     "press me!", color(100), stdFont, EVENT_BUTTON1);
  widget2=new Widget(100, 200, 100, 40,
                     "no, me!", color(150), stdFont, EVENT_BUTTON2);
  widgetList = new ArrayList();
  widgetList.add(widget1); widgetList.add(widget2);
}
void draw(){
  for(int i = 0; i<widgetList.size(); i++){
     Widget aWidget = (Widget) widgetList.get(i);
     aWidget.draw();
 }
}
```

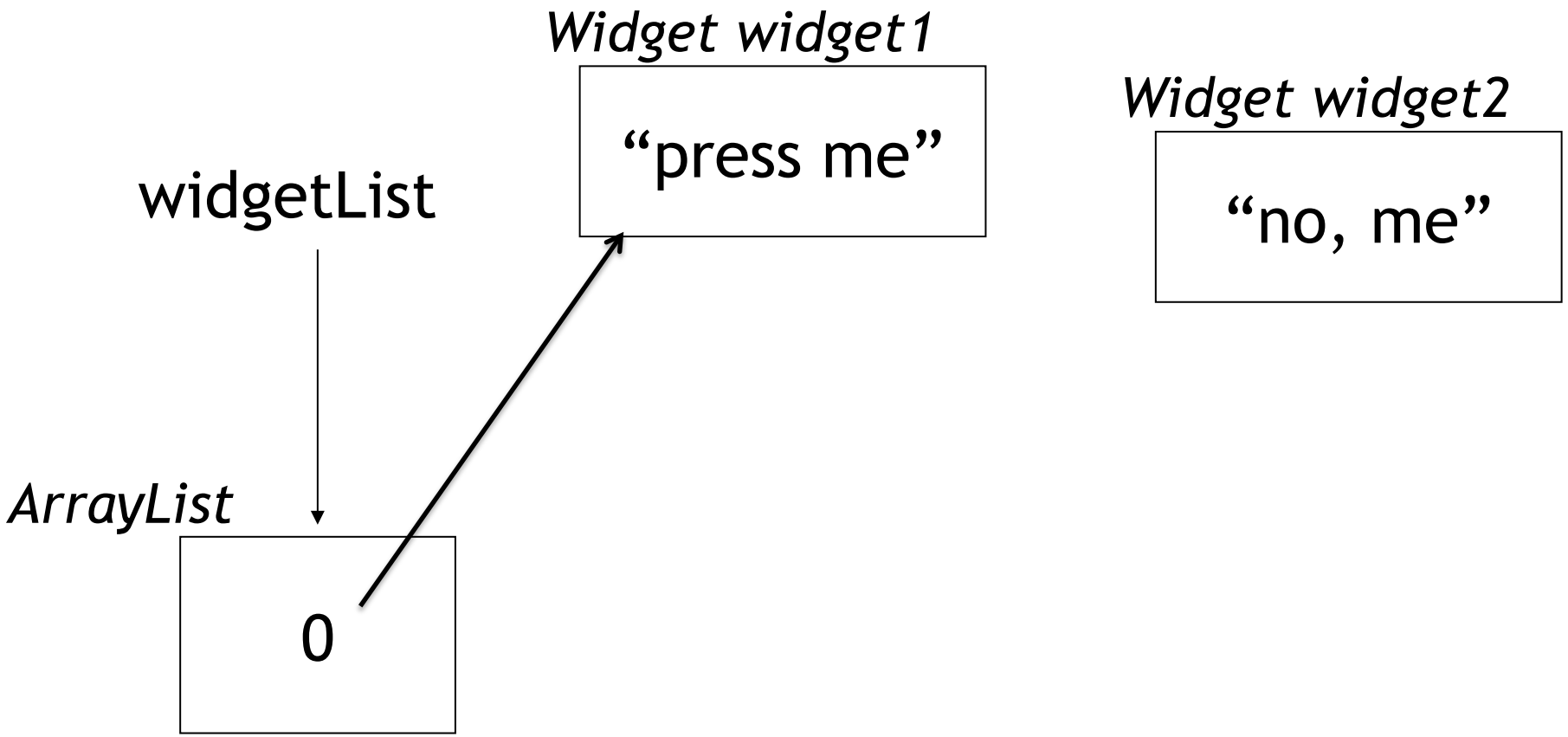
Constructing the list



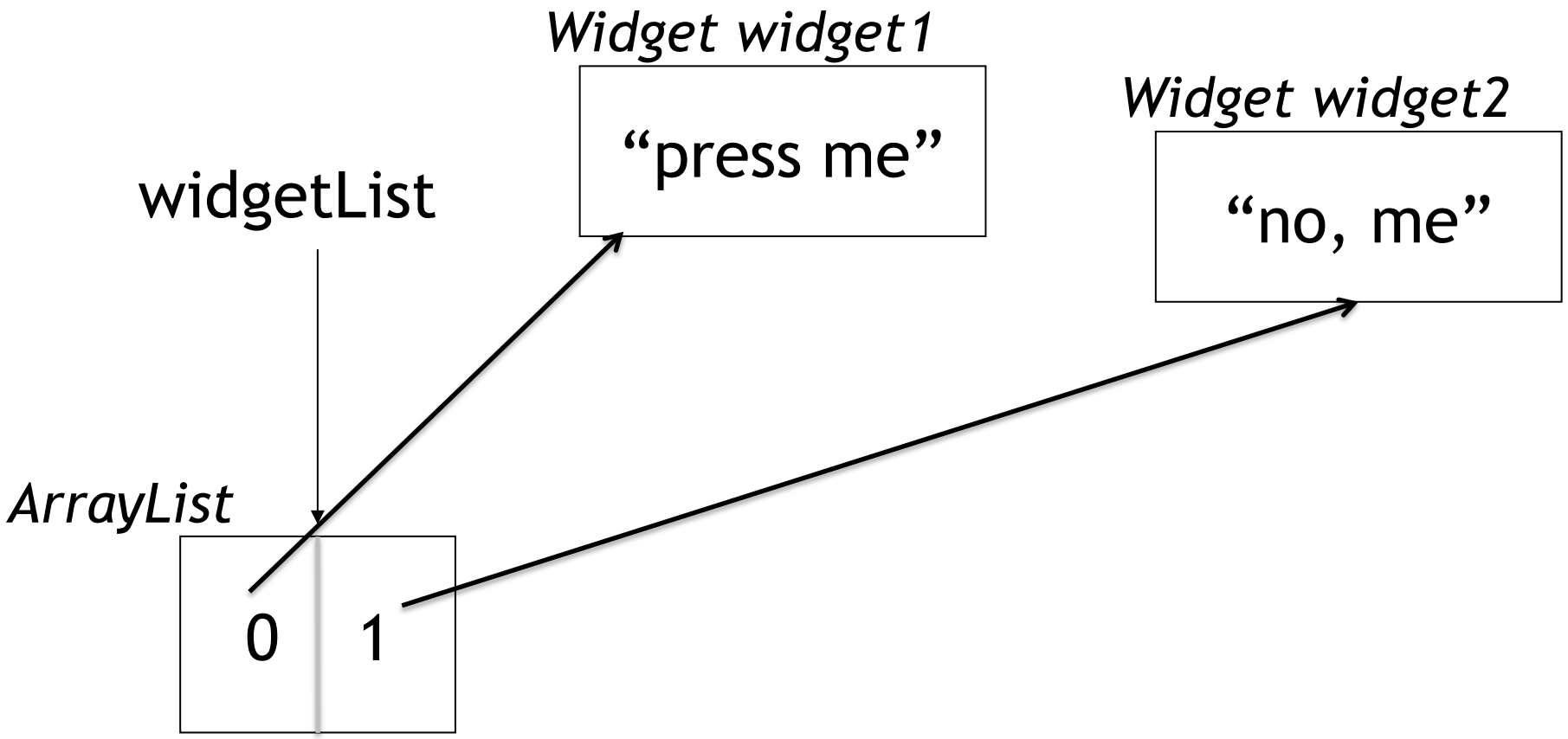
Constructing the list



Constructing the list



Constructing the list



Input handling

```
void mousePressed(){
  int event;

  for(int i = 0; i<widgetList.size(); i++){
    Widget aWidget = (Widget) widgetList.get(i);
      event = aWidget.getEvent(mouseX,mouseY);
      switch(event) {
        case EVENT_BUTTON1:
          println("button 1!");
          break;
        case EVENT_BUTTON2:
          println("button 2!");
          break;
      }
  }
}
```

Reacting to movement

```
void mouseMoved(){
  int event;

  ArrayList widgetList = currentScreen.getWidgets();
  for(int i = 0; i<widgetList.size(); i++){
    Widget aWidget = (Widget) widgetList.get(i);
    event = aWidget.getEvent(mouseX,mouseY);
    if(event != EVENT_NULL){
      aWidget.mouseOver();
    }
    else
      aWidget.mouseNotOver();
  }
}
```

Program with two screens

- Pseudo-code:
- setup:
  - create screen 1.
  - add widgets to screen 1.
  - create screen 2
  - add widgets to screen 2
  - set current screen to screen 1
- draw:
  - ask current screen to draw itself
- mousePressed:
  - ask current screen if any of it's widgets have been pressed.
  - if so, take appropriate action for that event

Screen class

```
class Screen {

  ArrayList screenWidgets;
  color screenColor;

  Screen(color in_color){
    screenWidgets=new ArrayList();
    screenColor=in_color;
  }
  void add(Widget w){
    screenWidgets.add(w);
  }
```

```
void draw(){
    background(screenColor);
    for(int i = 0; i<screenWidgets.size(); i++){
      Widget aWidget = (Widget)screenWidgets.get(i);
      aWidget.draw();
    }
 }
 int getEvent(int mx, int my){
   for(int i = 0; i<screenWidgets.size(); i++){
      Widget aWidget = (Widget) screenWidgets.get(i);
      int event = aWidget.getEvent(mouseX,mouseY);
      if(event != EVENT_NULL){
        return event;
      }
   }
   return EVENT_NULL;
 }
```

Main program

```
PFont stdFont;
final int EVENT_BUTTON1=1; final int EVENT_FORWARD=2;
final int EVENT_BUTTON2=3; final int EVENT_BACKWARD=4;
final int EVENT_NULL=0;
Widget widget1, widget2, widget3, widget4;
Screen currentScreen, screen1, screen2;
void setup(){
  stdFont=loadFont("Chalkboard-30.vlw");
  textFont(stdFont);
  widget1=new Widget(100, 100, 180, 40,
    "Button 1”, color(200, 0, 0), stdFont, EVENT_BUTTON1);
  widget2=new Widget(100, 200, 180, 40,
    "Forward", color(0, 200, 0), stdFont, EVENT_FORWARD);
  widget3=new Widget(100, 100, 180, 40,
    "Button 2", color(0,0,200), stdFont, EVENT_BUTTON2);
```

```
    widget4=new Widget(100, 200, 180, 40,
     "Backward", color(0,200,200), stdFont, EVENT_BACKWARD);
    size(400, 400);
    screen1 = new Screen(color(255));
    screen2 = new Screen(color(150));
    screen1.add(widget1);
    screen1.add(widget2);
    screen2.add(widget3);
    screen2.add(widget4);
    currentScreen = screen1;
}
void draw(){
    currentScreen.draw();
}
```

```
void mousePressed(){
  switch(currentScreen.getEvent(mouseX, mouseY)) {
  case EVENT_BUTTON1:
    println("button 1!");
    break;
  case EVENT_BUTTON2:
    println("button 2!");
    break;
  case EVENT_FORWARD:
    println("forward"); currentScreen = screen2;
    break;
  case EVENT_BACKWARD:
    println("backward"); currentScreen = screen1;
    break;
  }
}
```

Structure

- To start with, we want to read in the data from file.
- We'll need to put the data somewhere:
  - what should you use to store the data?
- Do we need 1 class or many classes?
- Where should this code go?
  - Setup would be a reasonable place.

Extracting data

- We typically only want to look at a small portion of the data at once.
- So we want to define a number of queries on the data.
- Some queries will just select a smaller part of the dataset.
- Other queries will generate new information – eg. Totals or averages for a subset of the data.
  - You will have to decide how to pass this information around within your program.
- Where will the queries be used?
  - Generally in response to user input, but may have initial (default) displayed.

Rendering – this week's task

- The data has been loaded in, the information we want has been extracted from it, now we want to draw it to the screen.
- We've already looked at drawing images, text, shapes.
- Where does this code go?
- Lots of different ways of displaying the information.

User Input

- In the outline program, you could just use `keyPressed()` and use the keyboard to invoke the different queries.
- After this could use something like Widget and widgetList classes from Week 6 to handle input.
- Could interact with the rendering itself, but this is more challenging.
- Various GUI libraries for processing are available.
  - Might not be worth the effort for just a few buttons.

Main Program (Pseudo-code)

- Setup

```
read_in_the_file(); // done, week 1
result = default_query();
current_query = query3;// whatever type of query is default
```

- Draw

```
switch(current_query){
    case query1:
        render_query1(results); /week 2
        break;
    case query2:
        Etc…..
    }
render_controls();
```

Main Program (Pseudo-code)

- mousePressed()
  - Work out which button pressed

```
switch(event)
    case button 1:
        current_query = query1;
        results=query1();
        break;
    case button 2:
        current_query = query2;
        results=query2();
    Etc.
```

  - You may need several “results” variables for different types of data returned by different queries.

Rendering

- The way you draw results for the query depends on the type of query:
- Data for a particular attribute over different subsets of the data could be displayed as a line or bar chart.
- Subsets with the highest or lowest values for an attribute could be displayed using combination of text (for the name of the subset) and horizontal bar representing the value.

Classes

- You will have some classes representing the user interface widgets.
- You will have one or more classes for storing the data.
- You may also have classes representing the results of a query. Taking this approach, the class representing the query result might have a method to draw itself.

Outline of minimal code for this week

- read in the csv file and store the data (in setup), as for last week.
- create a (simple) query result and set this to be the current query (in setup)
- write a method which will draw this query result as a chart (e.g. bar chart).
- draw the results of this query as a bar chart (in draw), using the method you have defined.