CS1013 – Programming Project

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

Trinity College Dublin

DSG Distributed Systems Group

The project

- Begins after reading week.
- Groups to be allocated later this week.
- The project is to construct an application to explore a large data set using Processing.
- Previous years included tasks like visualising music play data, property prices, weather data etc. given in a file of comma separated values.
- Three major components – handling data, visualization of data, and user interface.
- Full specification to be released later this week.

The project

- There will be a fixed demonstration schedule, with specific features to be demonstrated every week.
- Code will be submitted every week.
- Keeping track of who has written and modified the code is part of the submission.
- Half of the project marks are for the continuous submission, half for the final result.

The project

- DO NOT MISS THE LABS.
- The labs are not just about demonstration/marking.
- You are expected to be working with your project group for the duration of the labs.
- Agree when and where outside of the labs your group will meet, and how you will communicate.
- Discuss issues with your demonstrator.

Program structure

setup()

draw()

thealiens

theplayer

thebullet

Program structure

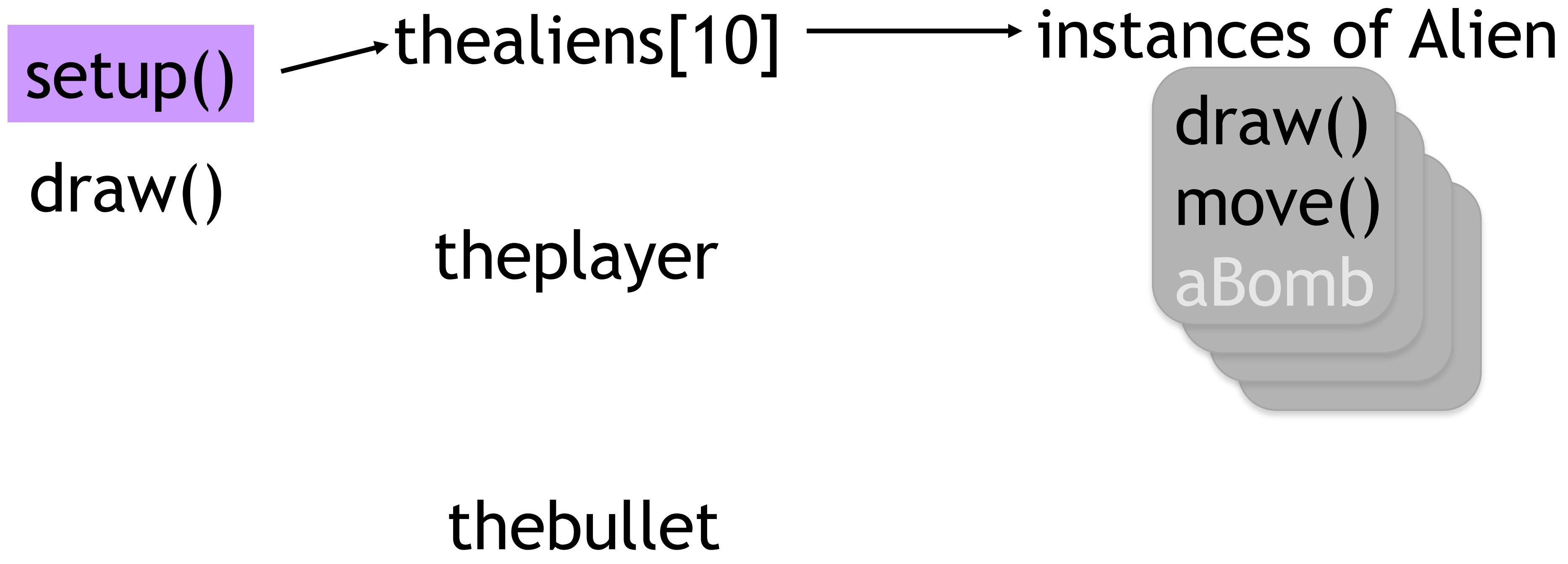
$$\begin{tikzcd} \text{setup()} \arrow[r] & \text{thealiens[10]} \arrow[r] & {} \end{tikzcd}$$

draw()

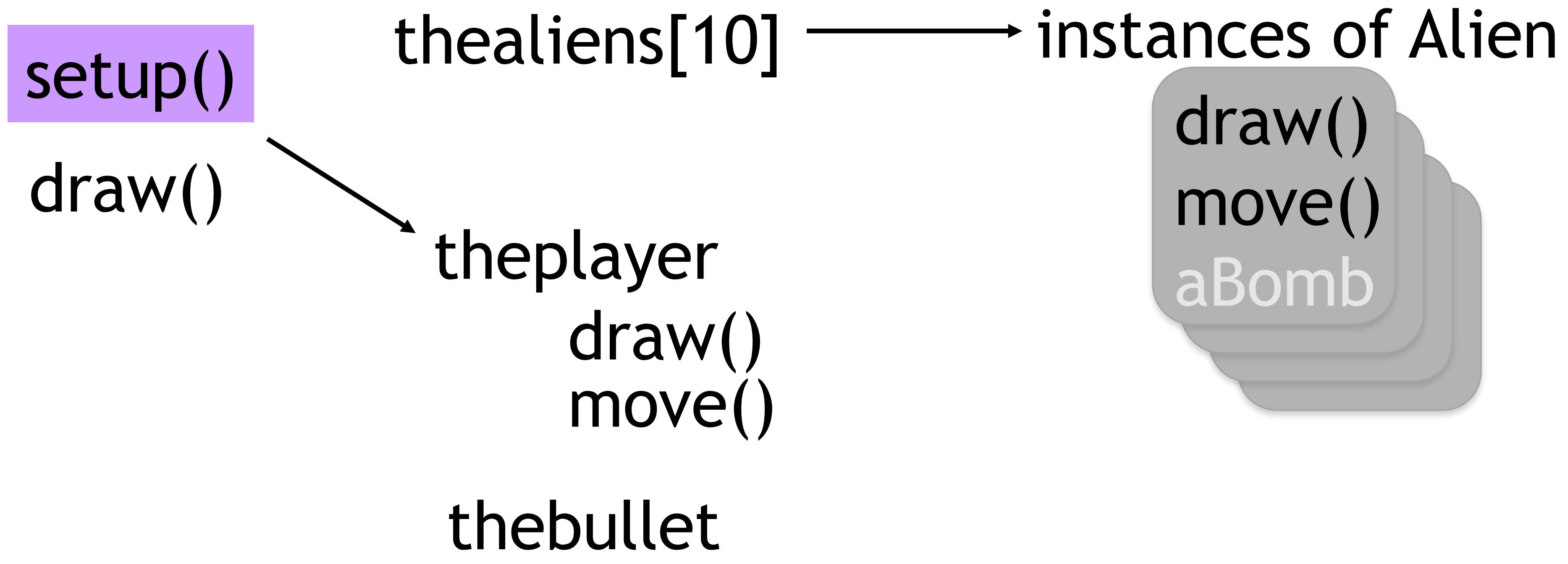
theplayer

thebullet

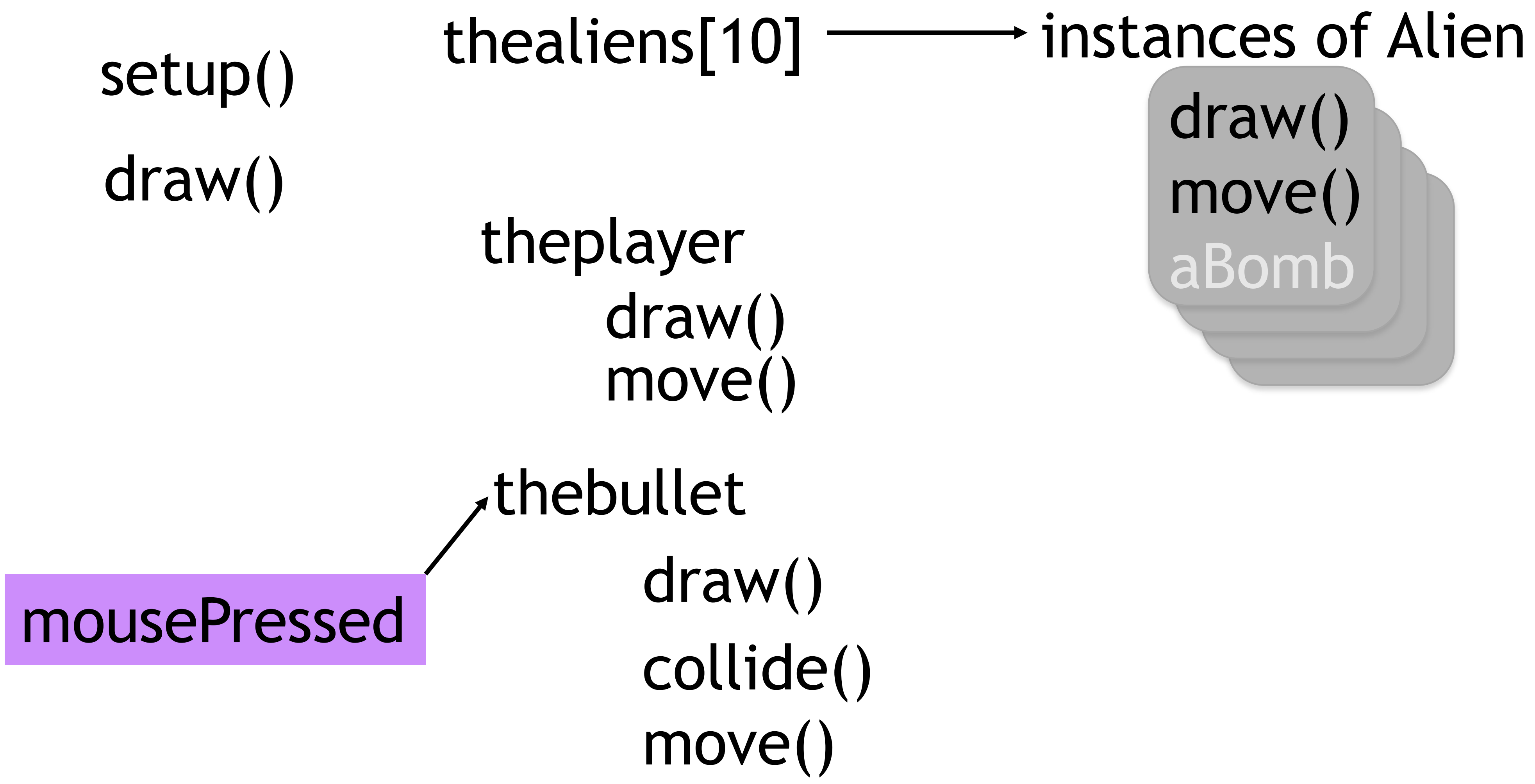
Program structure



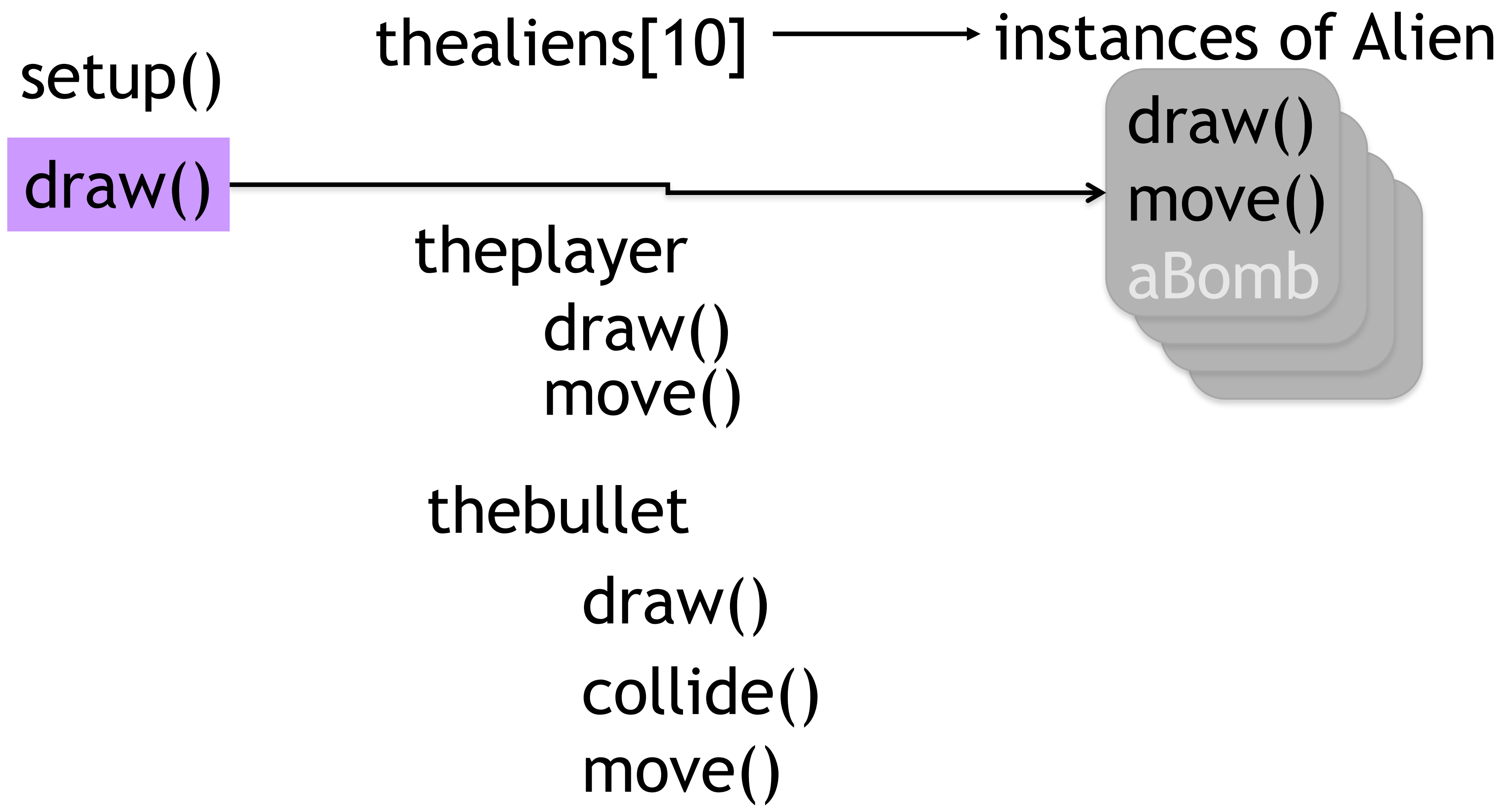
Program structure



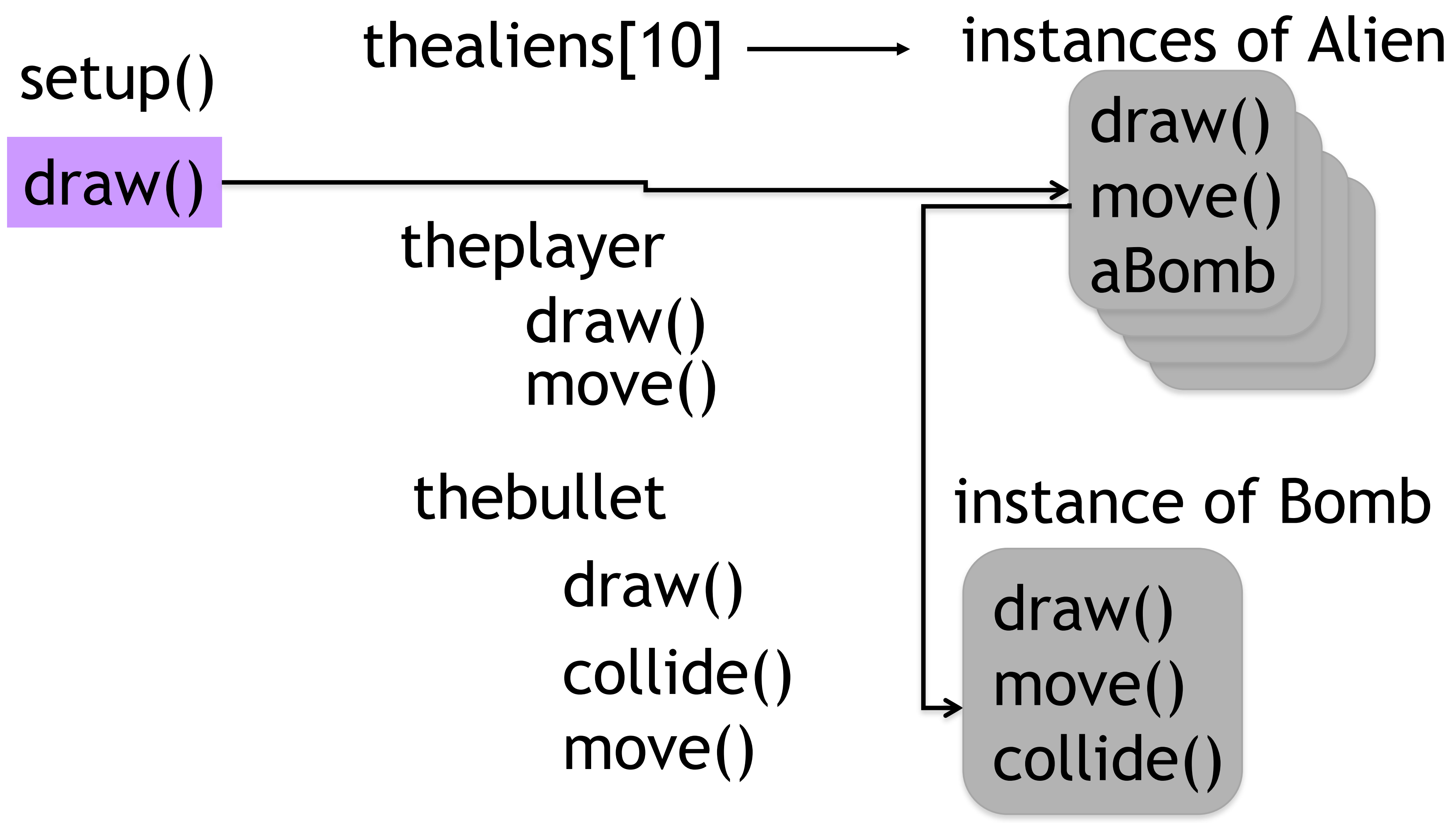
Program structure



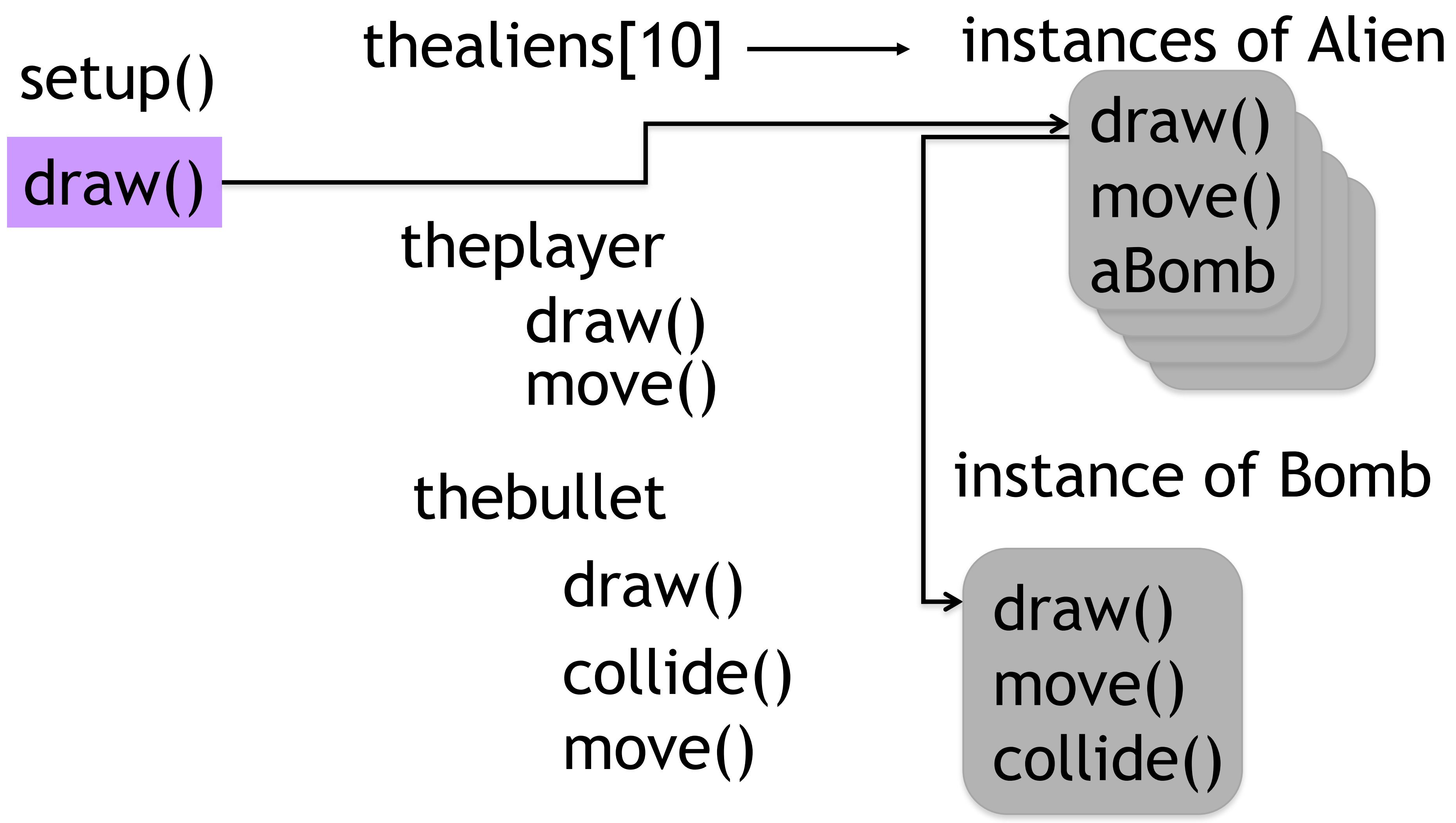
Program structure



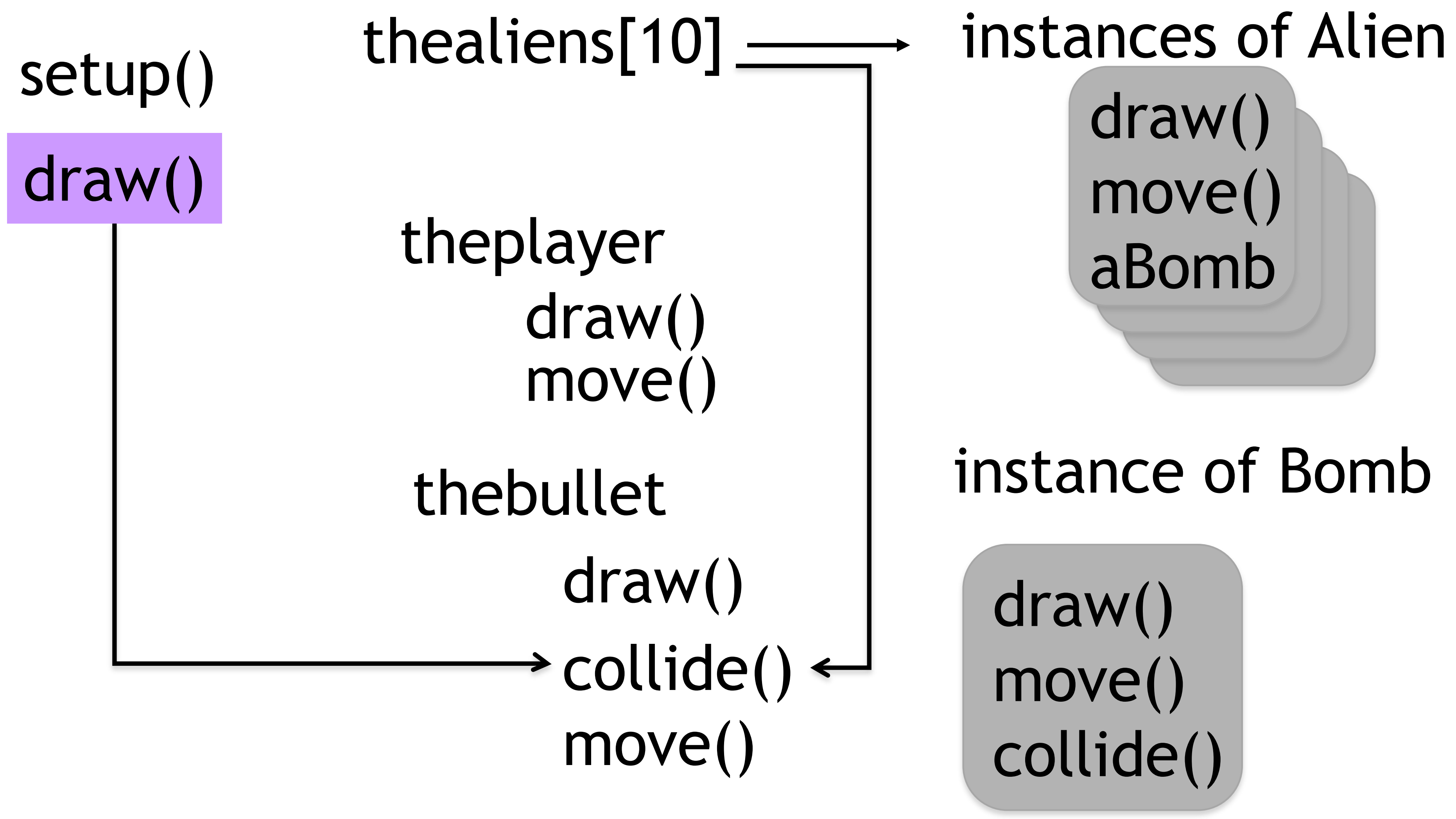
Program structure



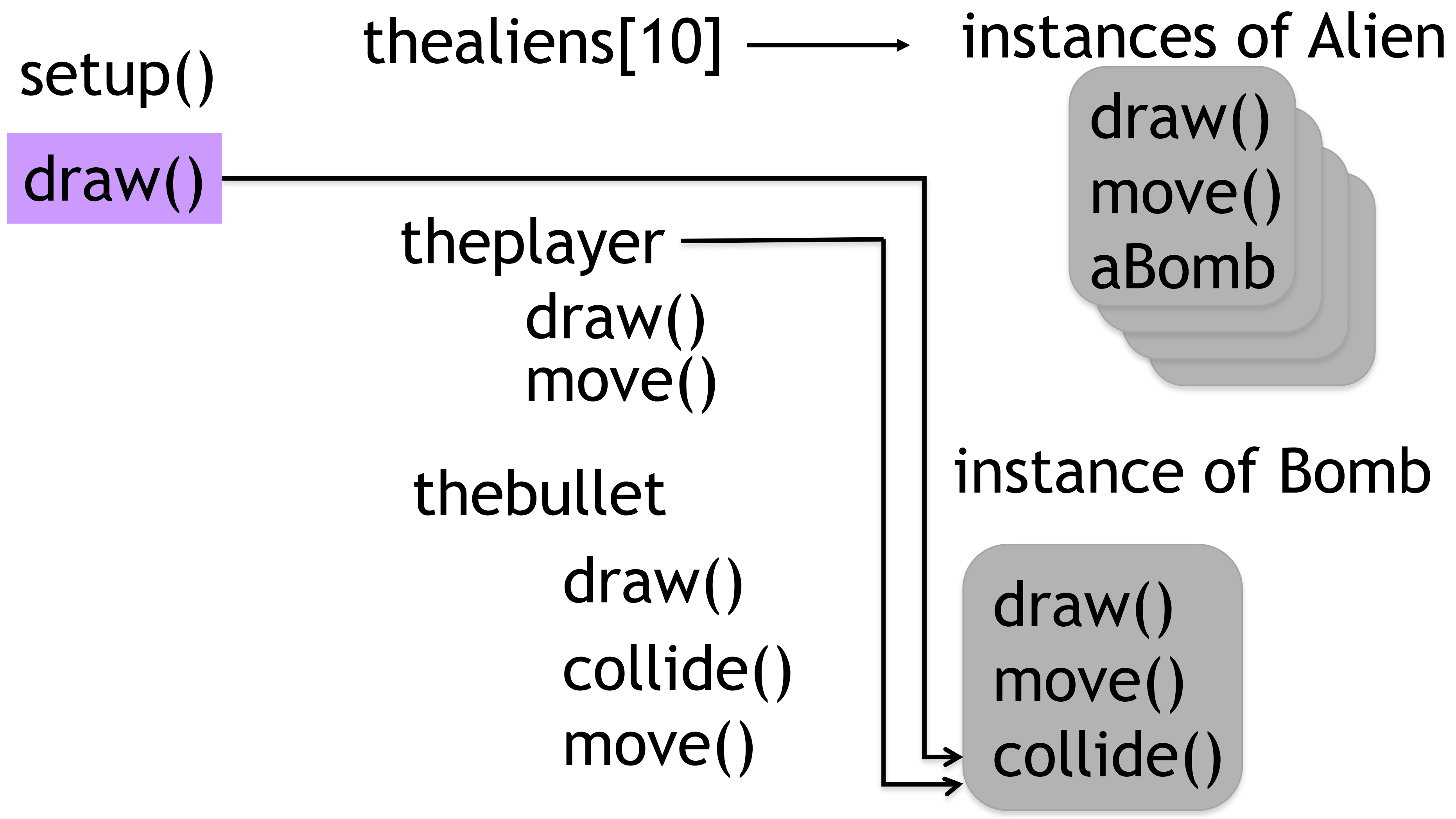
Program structure



Program structure



Program structure



Bomb class

```
class Bomb{
  int x, y, width, height;
  color bombColor= color(255, 0, 0);

  Bomb(int xpos, int ypos, int bwidth, int bheight){
    x=xpos;
    y=ypos;
    width=bwidth;
    height=bheight;
  }
  void move(){
    y++;
  }
  void draw(){
    fill(bombColor);
    rect(x,y,width,height);
  }
  boolean offScreen(){
    return y>=SCREENY;
  }
  boolean collide(Player thePlayer){
    return (x+width > thePlayer.x() && x < thePlayer.x()+thePlayer.width()
                 && y+height > thePlayer.y() && y <
   thePlayer.y()+thePlayer.height());
  }
}
```

Alien constructor

```
Bomb myBomb;
Alien (int xpos, int ypos, PImage okImg,
  PImage exImg){
    x = xpos;
    y = ypos;
    status = ALIEN_ALIVE;
    normalImg=okImg;
    explodeImg=exImg;
    direction=FORWARD;
    myBomb=null;
  }
```

Changes to Alien.move()

```
void move(){
    if(myBomb!= null){
      myBomb.move();
      if(myBomb.offScreen())
         myBomb=null;
    }
  else if(status==ALIEN_ALIVE)
     if(random(0, 100)<1) myBomb=new
     Bomb(x, y);
```

Changes to Alien.draw()

```
void draw(){
    if(myBomb!=null)
      myBomb.draw();
    if(status==ALIEN_ALIVE)
      image(normalImg, x, y);
    else if(status!=ALIEN_DEAD){
      image(explodImg, x, y);
      status++;
    }
 Bomb getBomb(){
    return myBomb;
  }
```

Changes to main draw method

```
for(int i=0; i<theAliens.length; i++){
    theAliens[i].move();
    theAliens[i].draw();
    Bomb myBomb=theAliens[i].getBomb();
    if(myBomb!= null){
      if(myBomb.collide(thePlayer))
         game_over();
     }
}
```

Input handling

- Let’s say I want to have various buttons etc. that I want to click on.
- Could have a really long if-statement checking the mouse X and Y position against all the different buttons.
- It would also have to write all the button labels individually, change the colors etc.
- Would be really messy and difficult to modify the program.
- Better to define a class which takes care of everything to do with the button.

Widgets

Pronounced wih-jit. (1) A generic term for the part of a GUI that allows the user to interface with the application and operating system. Widgets display information and invite the user to act in a number of ways. Typical widgets include buttons, dialog boxes, pop-up windows, pull-down menus, icons, scroll bars, resizable window edges, progress indicators, selection boxes, windows, tear-off menus, menu bars, toggle switches and forms.

Events

- Events are things that happen that we are interested in.
- If someone presses a button on the screen, we are interested in this “event”.
- We might ask a button “have you been pressed?”. If it has, we have an event.
- We might ask a whole list of buttons “have any of you been pressed?”. If we have an event, we will want to know which event it is, i.e. which button has been pressed.

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;
  }
}
```

Main program

```
PFont stdFont;
final int EVENT_BUTTON1=1;
final int EVENT_BUTTON2=2;
final int EVENT_NULL=0;
Widget widget1, widget2;

void setup(){
  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(100),
                         stdFont, EVENT_BUTTON2);
  size(400, 400);
}

void draw(){
   widget1.draw();
   widget2.draw();
}
```

Input handling

```
void mousePressed(){
  int event;
  event = widget1.getEvent(mouseX,mouseY);
  if(event== EVENT_BUTTON1)
    println(“button 1 pressed”);
  else {
      event =
  widget2.getEvent(mouseX,mouseY);
      if(event==EVENT_BUTTON2)
         println(“button 2 pressed”);
  }
}
```

Lots of widgets

- This would be very messy if we had lots of widgets, but there is clearly a pattern.
- We could have an array of widgets, but this might be awkward if we want to add and remove buttons etc.
- Try using a list.
- No reason for this to be complicated.

Lists

- Lists allow us to store arbitrary numbers of objects – so don't have to decide exactly how many items we will have.
- Java (and processing) provide an `ArrayList` class which is a type of list.
- Can store any type of object.
- Use `add()` to add an item to the `ArrayList`.
- Use `get(index)` to retrieve an item from the list.
- Need to cast the object returned to the appropriate type before you use it.
  - `myVariable = (myType)myArrayList.get(i);`

ArrayList of Strings

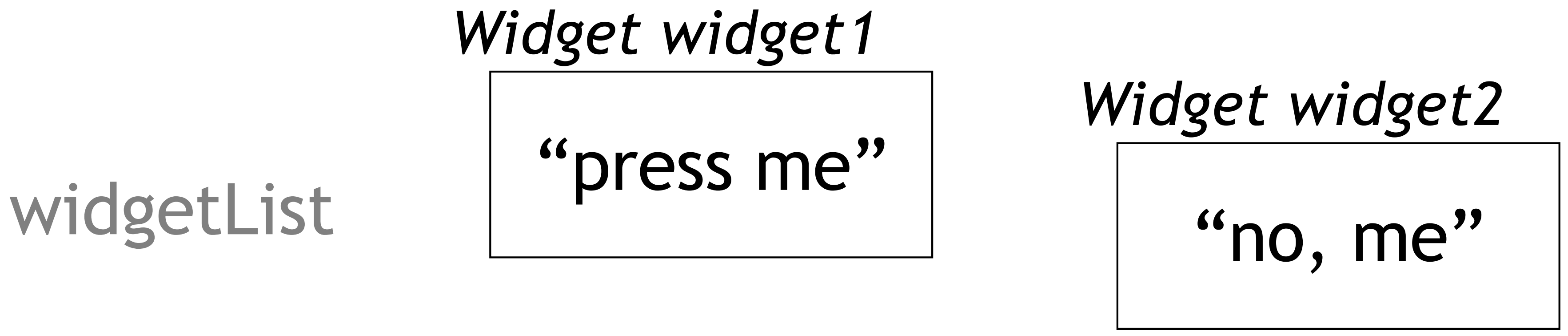
```
ArrayList myStrings;
void setup(){
  myStrings = new ArrayList();
  myStrings.add("the first string");
  myStrings.add("the second string");
  myStrings.add("the third string");

  for(int i= 0 ; i<myStrings.size(); i++){
    String theString;
    theString = (String) myStrings.get(i);
    println("String number " + i + " is: "+ theString);
  }
}
```

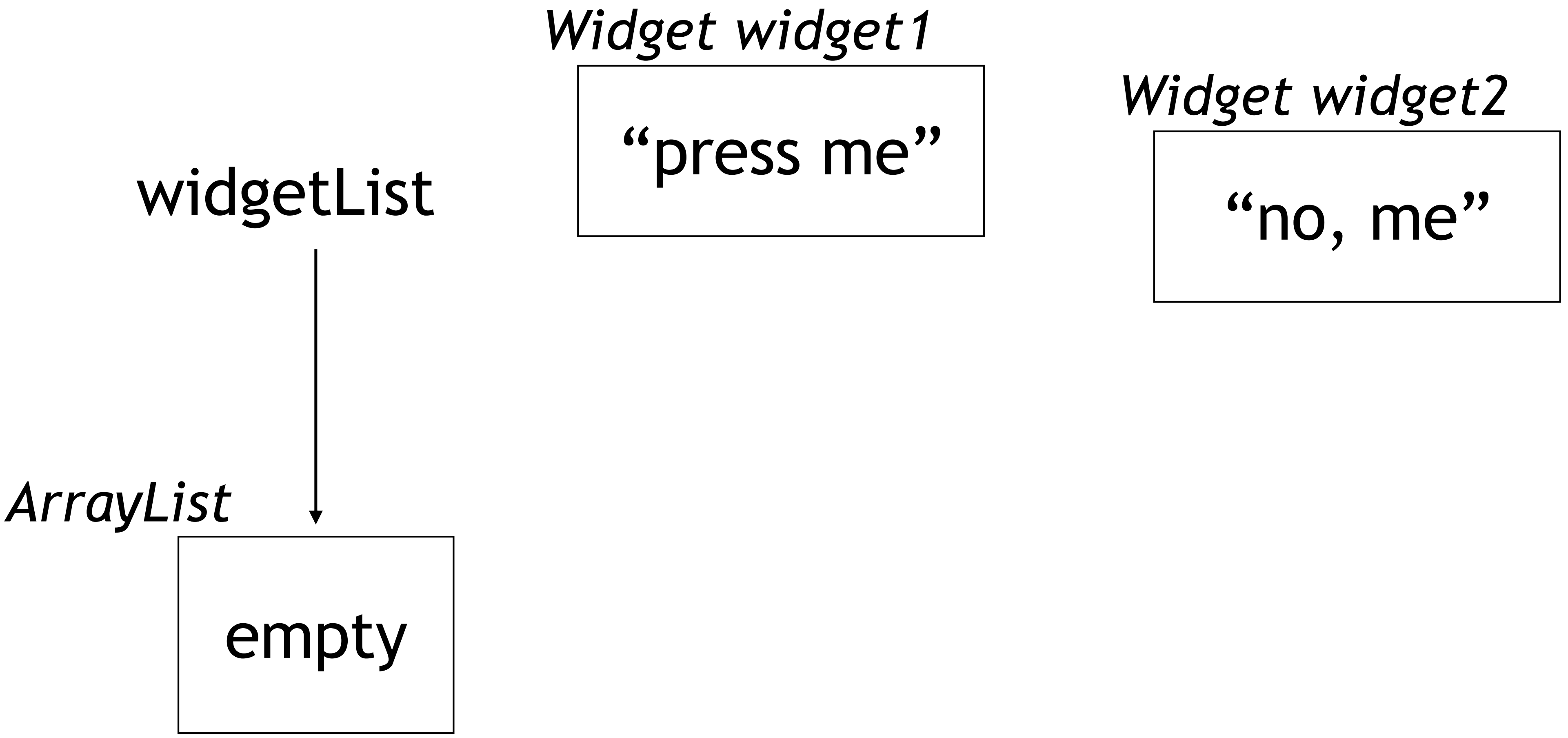
ArrayList of Widgets

```
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();
 }
}
```

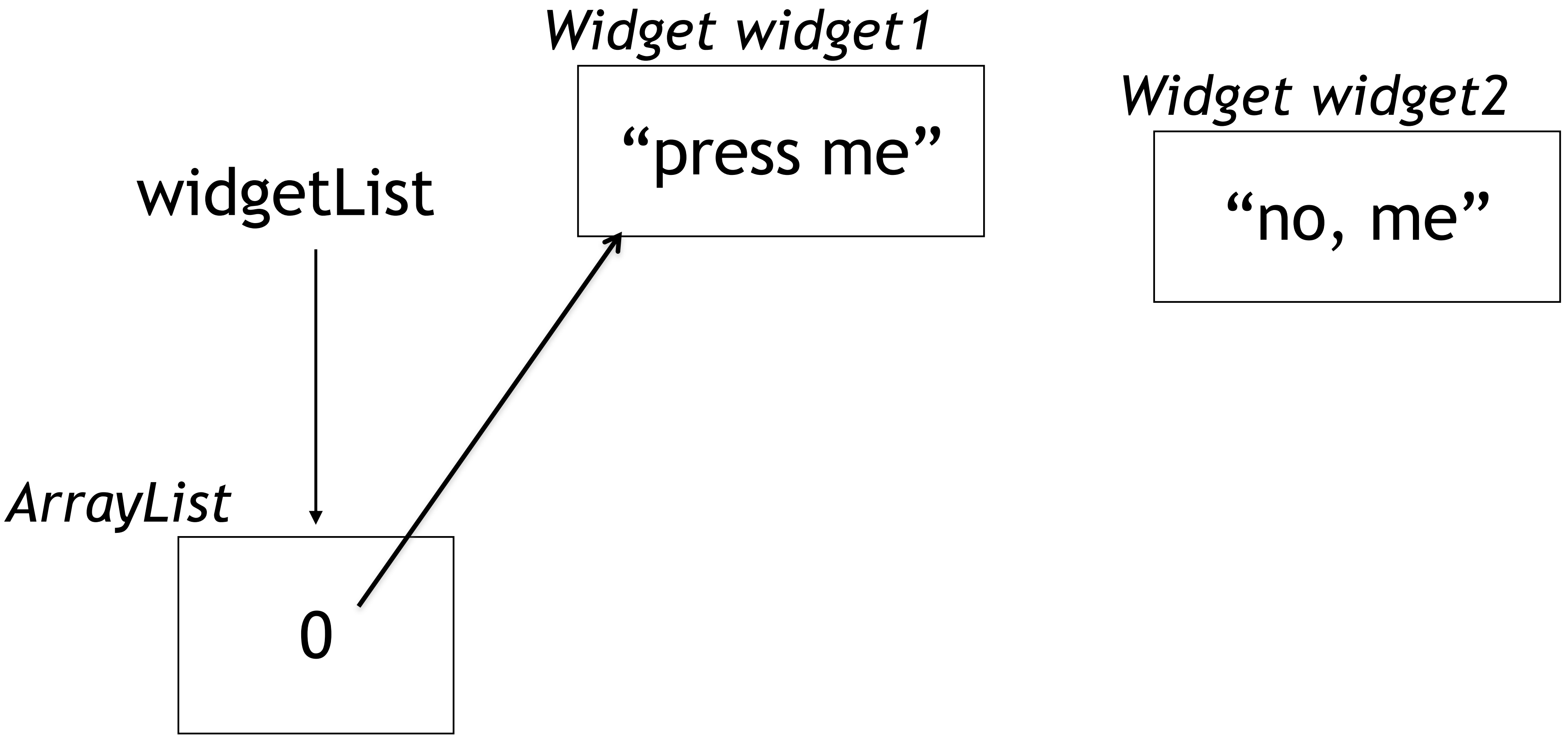
Constructing the list



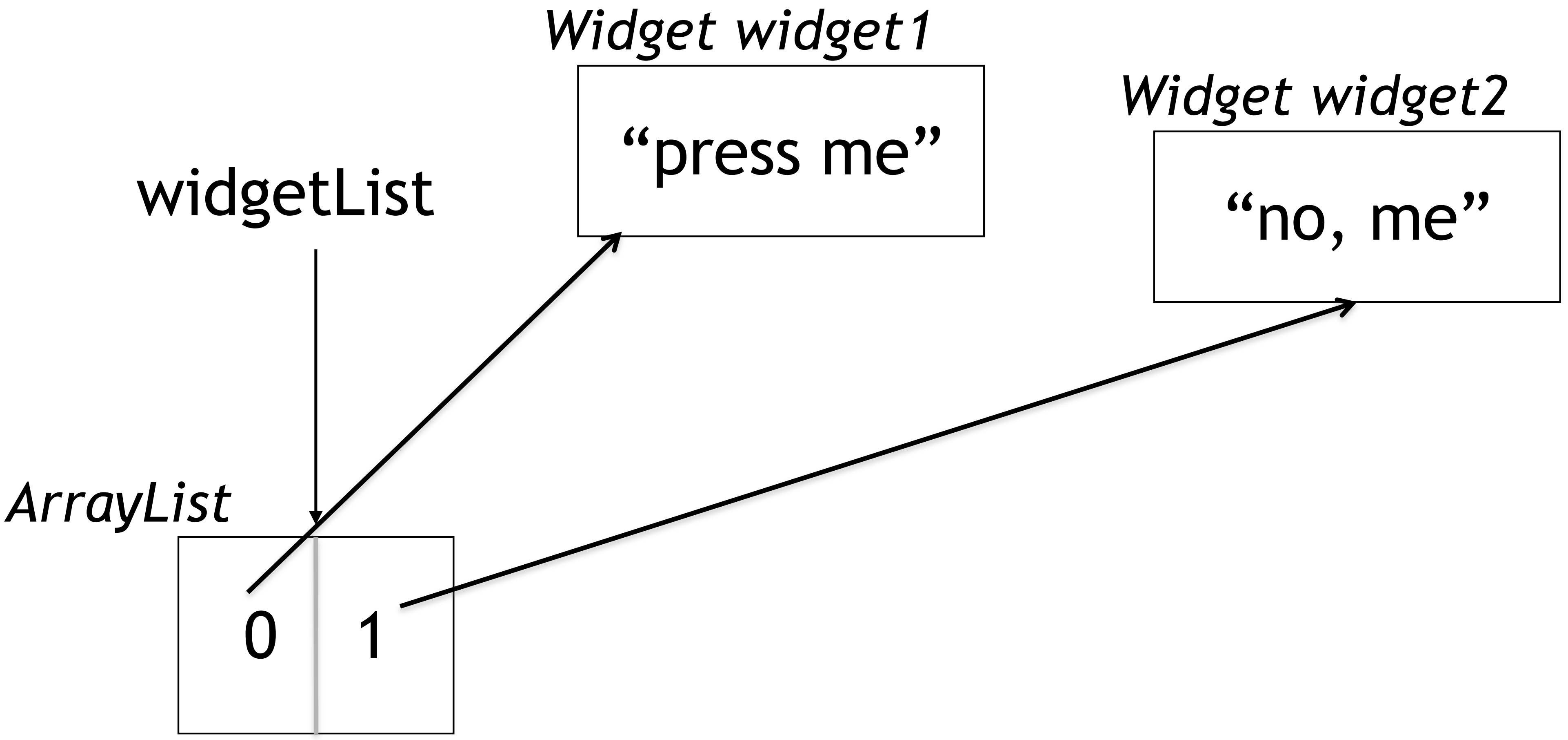
Constructing the list



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;
       }
  }
}
```

Event handlers in Processing

- mousePressed() – button pressed – we have been using this.
- mouseReleased() – button released.
- mouseMoved() – mouse moved without mouse button pressed. Used in this weeks exercise.
- mouseDragged() – mouse moved with mouse button pressed.

Exercise – Week 6

1. Using the **widgetList** program and **Widget** class covered at the lecture, write a program containing three widgets/buttons labelled "red", "green" and "blue". Alter the ***mousePressed*** method so that a square displayed on the screen changes color as each button is pressed. (3 marks)

Exercise – Week 6

2. Alter your program so that the border color of a widget changes (to white) when the mouse pointer is over it but reverts to normal (black) when the mouse pointer is not over it. Processing will call the ***mouseMoved*** method (if you define it) whenever the mouse is moved. The ***stroke()*** method allows you to define the border color. (3 marks)

Exercise – Week 6

- Define a **Screen** class which contains an **ArrayList** of Widgets. The screen should have it's own background colour. Give **Screen** a ***getEvent*** method which returns an event (the Widget pressed), and define a ***draw*** method in **Screen** which draws the screen's widgets. Each screen has it's own background color.
- Create an ***addWidget*** method to add a widget to the ArrayList of the screen. Use your **Screen** class to write a program where there are two screens, and two buttons on each screen, one of the buttons on each screen should move you forward and back through the different screens as illustrated below. You can do this by creating two instances of Screen. Pressing the other button should result in some text output from `println` that that button has been pressed. *Hint: you can create an extra Screen variable currentScreen which keeps track of the current screen – i.e. one of the existing instances.*
- Your mousePressed method in the main program will no longer require a loop (Screen should provide a `getEvent()` method), but should still deal with the different events.