CS1013 – Programming Project

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

Trinity College Dublin

DSG
Distributed Systems Group

Program structure

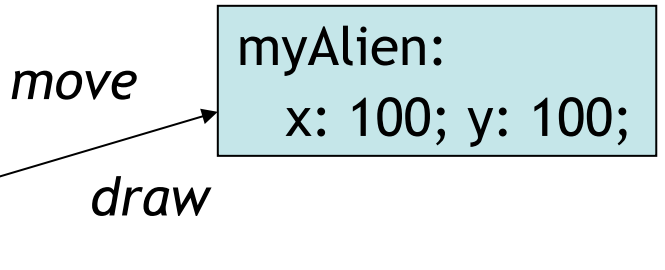
- Get the structure of your program right and putting it together will be a lot easier.
- Classes are good for providing a clear structure, and help to keep your code clean, so that the code is easier to maintain and improve.
- Why write hideously complex, unmaintainable code to avoid using classes?

The alien example

```
class Alien
    move()
    draw()
```

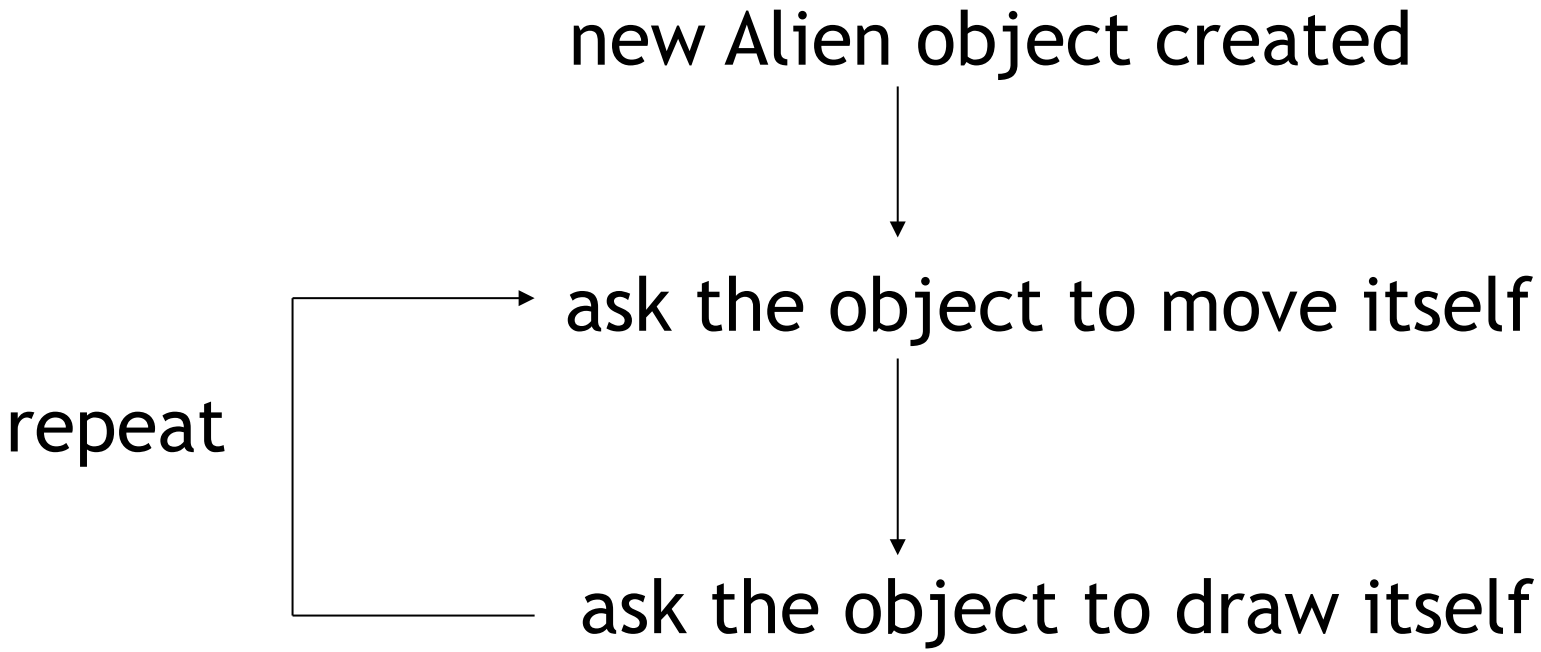
Main program:

```
setup()
  - create a new Alien object
draw()
   - ask alien object to move itself
   - ask alien object to draw itself
```



One Alien

- Start out with one alien object.



Two Aliens

- Using a class already starts to pay off when we have more than one alien:

```
firstAlien = new Alien(100, 100);
secondAlien = new Alien(100+ALIENWIDTH, 100);
.
.
.
.
firstAlien.move(); secondAlien.move();
firstAlien.draw(); secondAlien.draw();
```

Many Aliens

- Pretty tedious and inflexible to code all the aliens by hand. Just use an array.

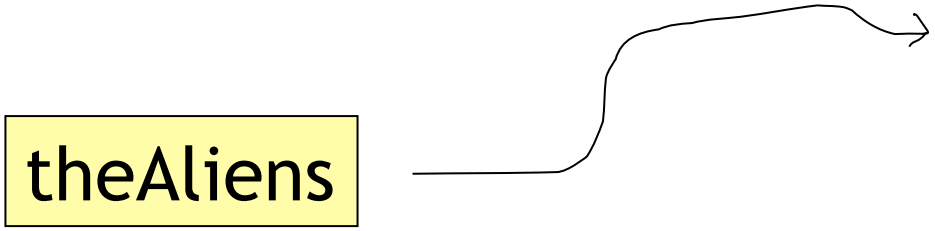
```
/* This creates something we can
  point to an array of Aliens with */
Alien theAliens[];


void setup(){
// Now it points at an empty array
    theAliens = new Alien[10];
}
```

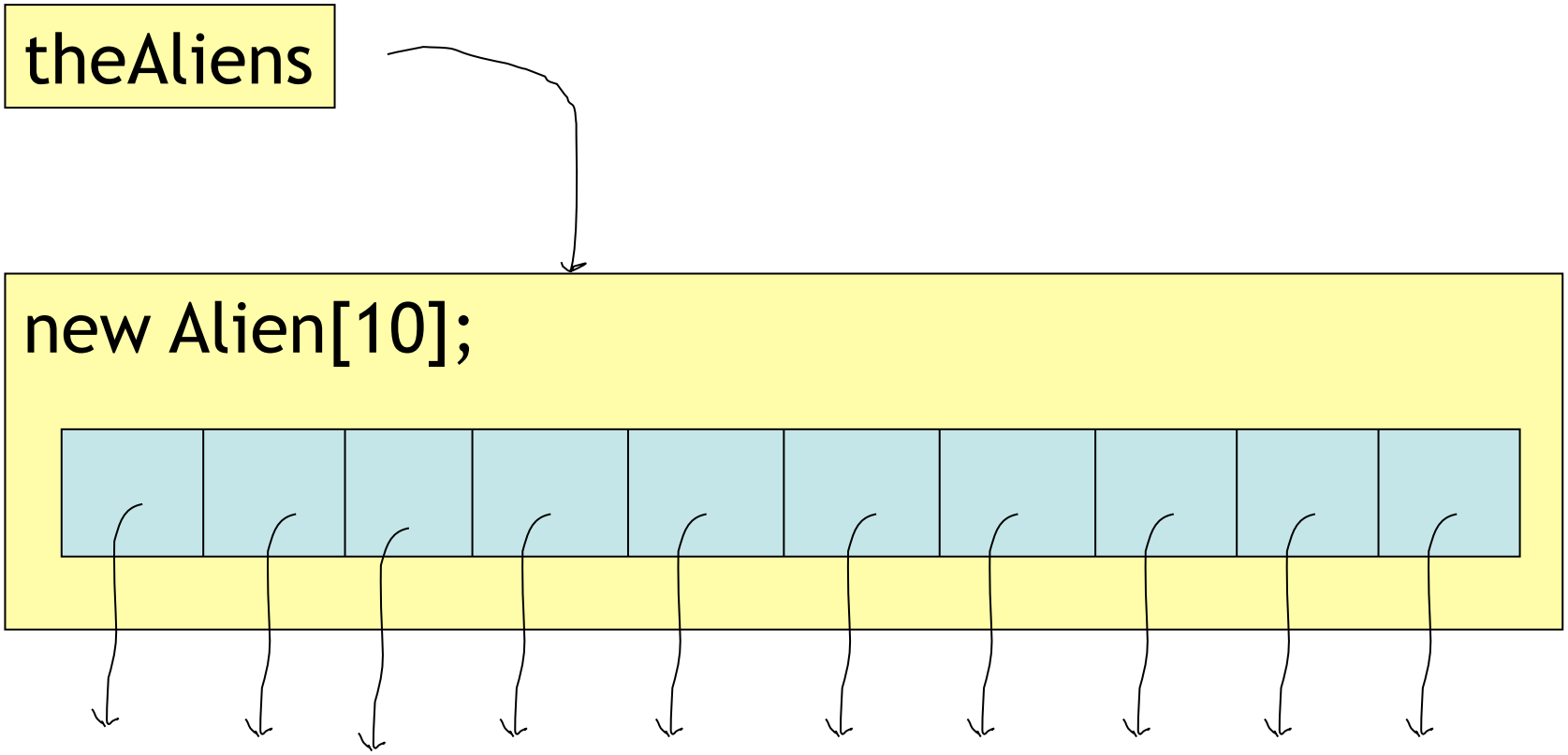
Many aliens

```
init_array(Alien theAliens[])
{
   for(int i=0; i<theAliens.length; i++){
      theAliens[i] = new Alien(100 +
                     i*ALIENWIDTH, 100);
}
```

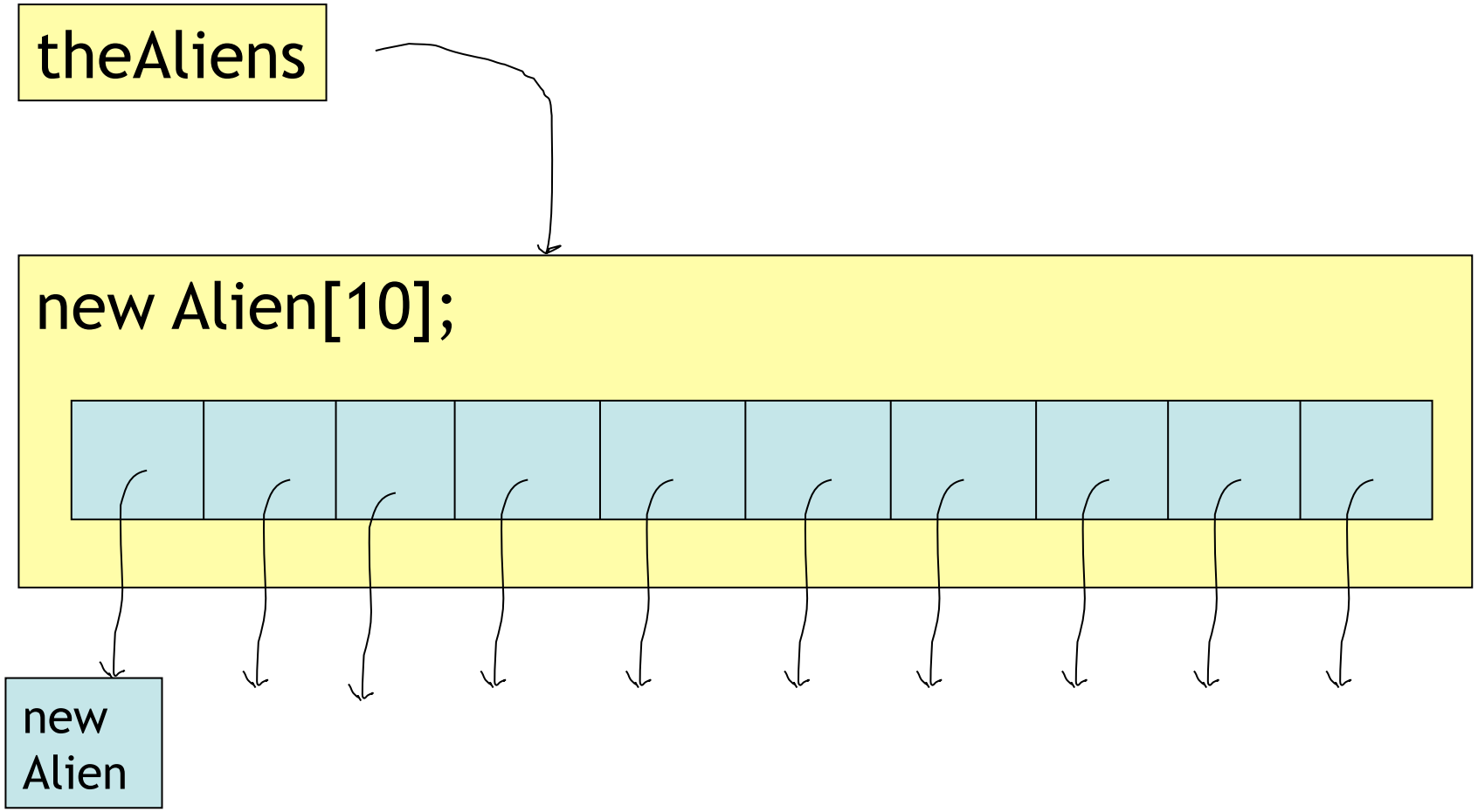
What is happening?



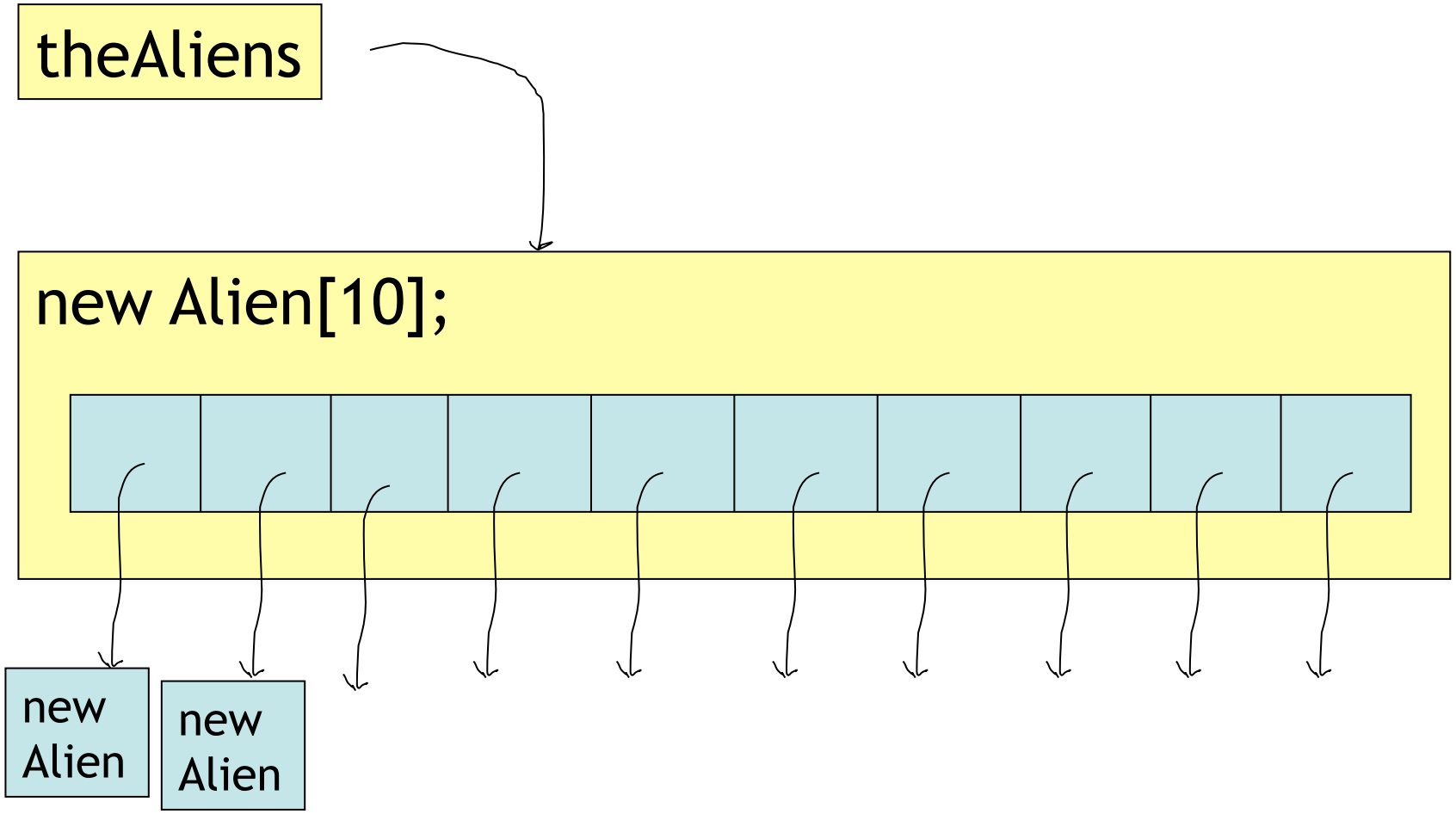
What is happening?



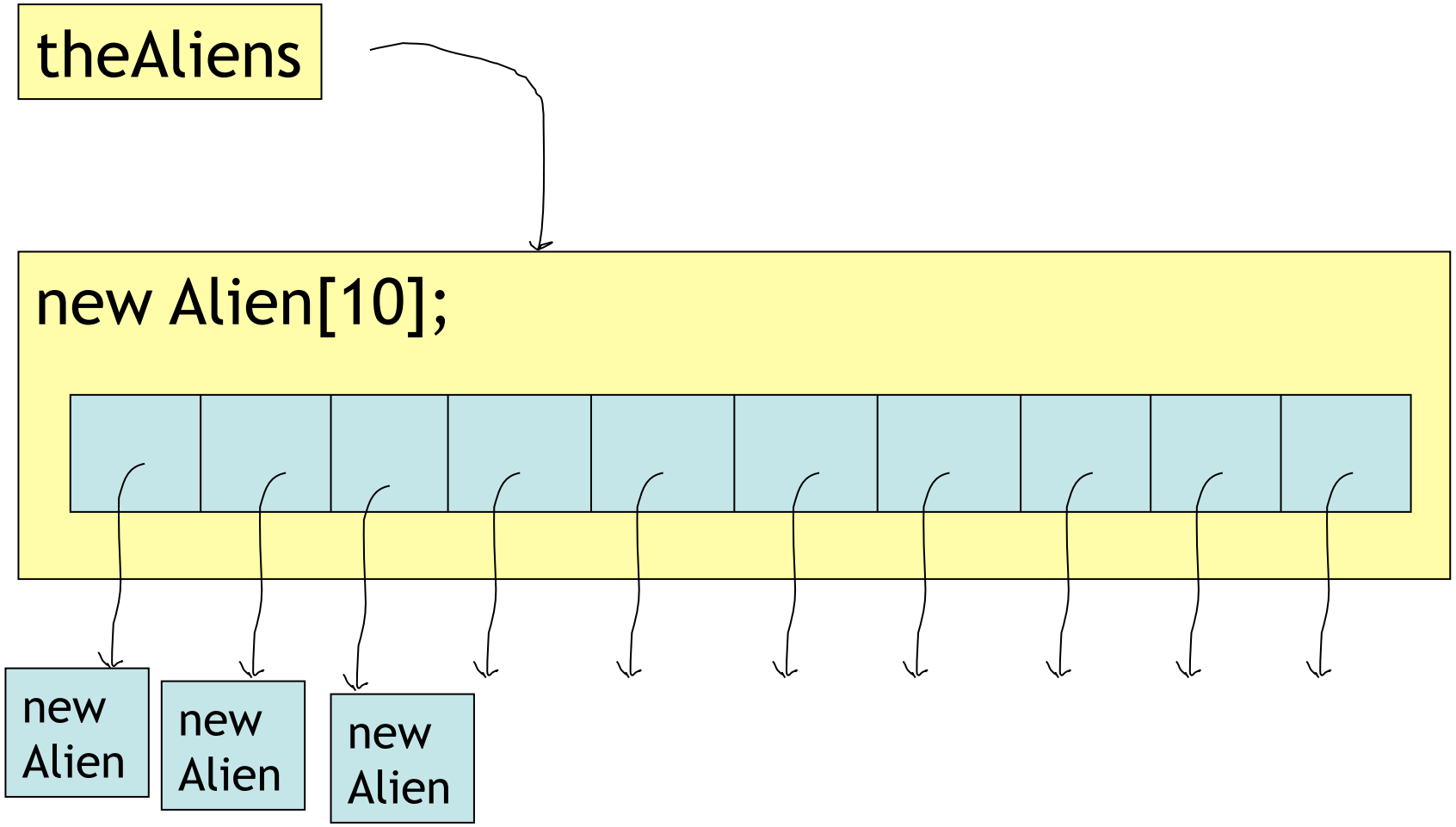
What is happening?



What is happening?



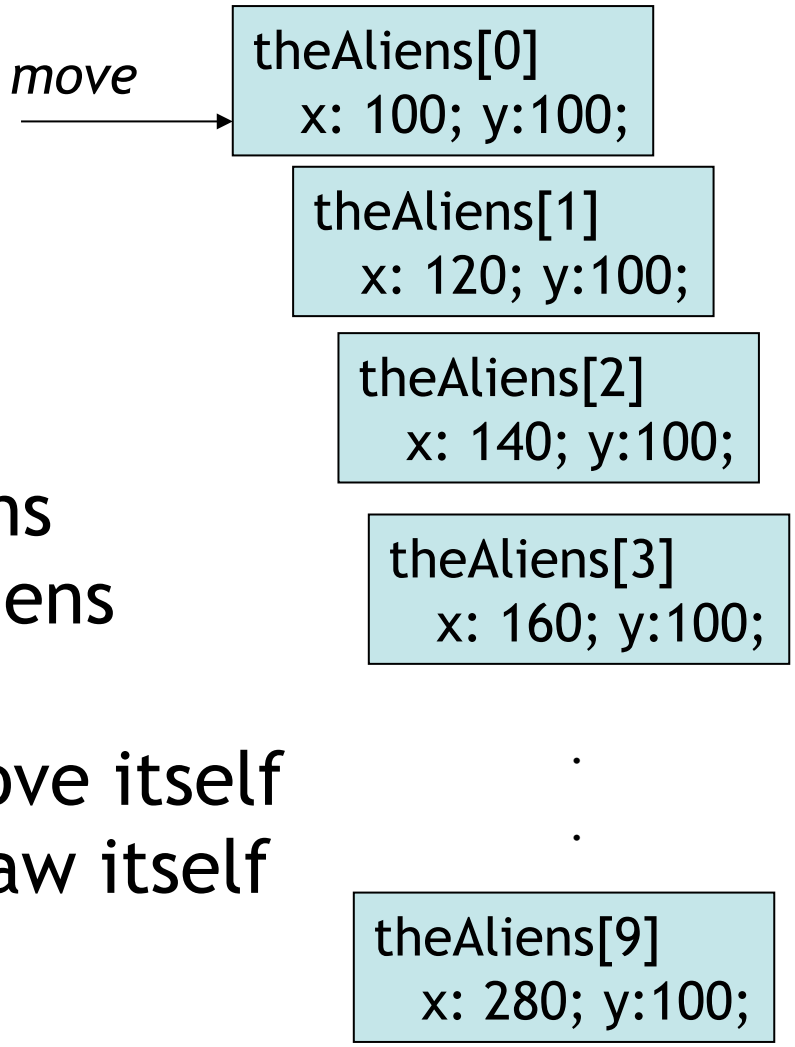
What is happening?



Many aliens

- When we run the program:
  - Create an empty array of Aliens with 10 positions in it.
  - Assign this new array to theAliens so we can use it.
  - Go through the array and create a new Alien object to go in to each position in the array.

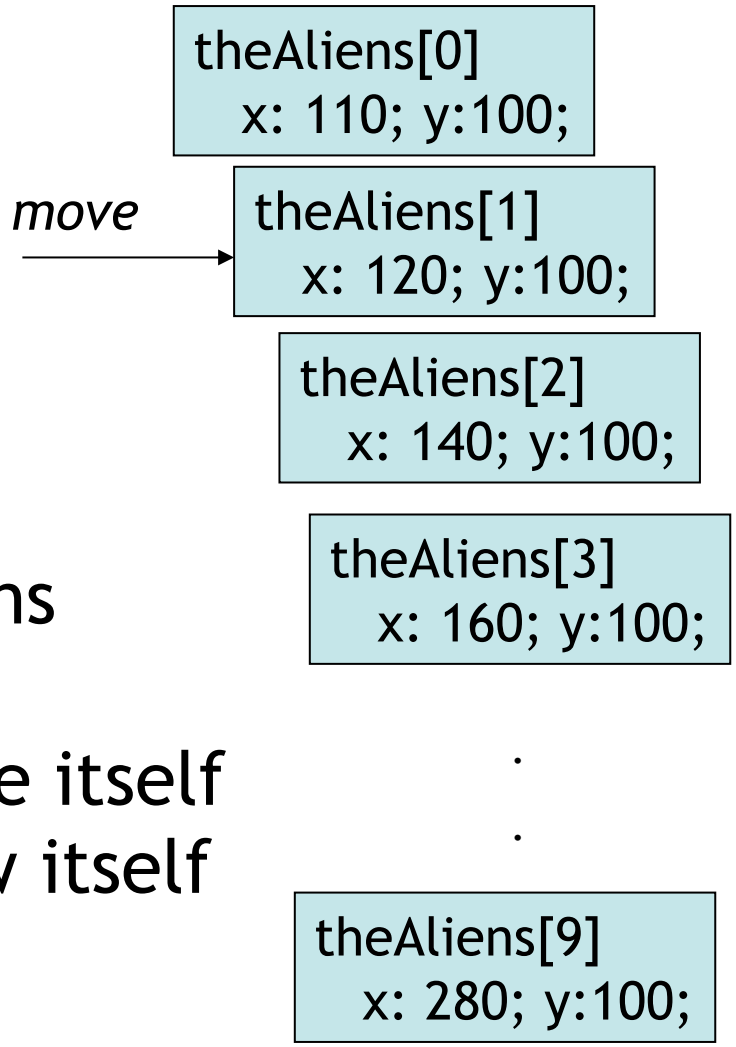
The alien example



Main program
setup()
- create an array to hold Aliens
- fill in the array with new Aliens

draw()
- ask each alien object to move itself
- ask each alien object to draw itself

The alien example



Main program

setup()

- create an array to hold Aliens
- fill in the array with new Aliens

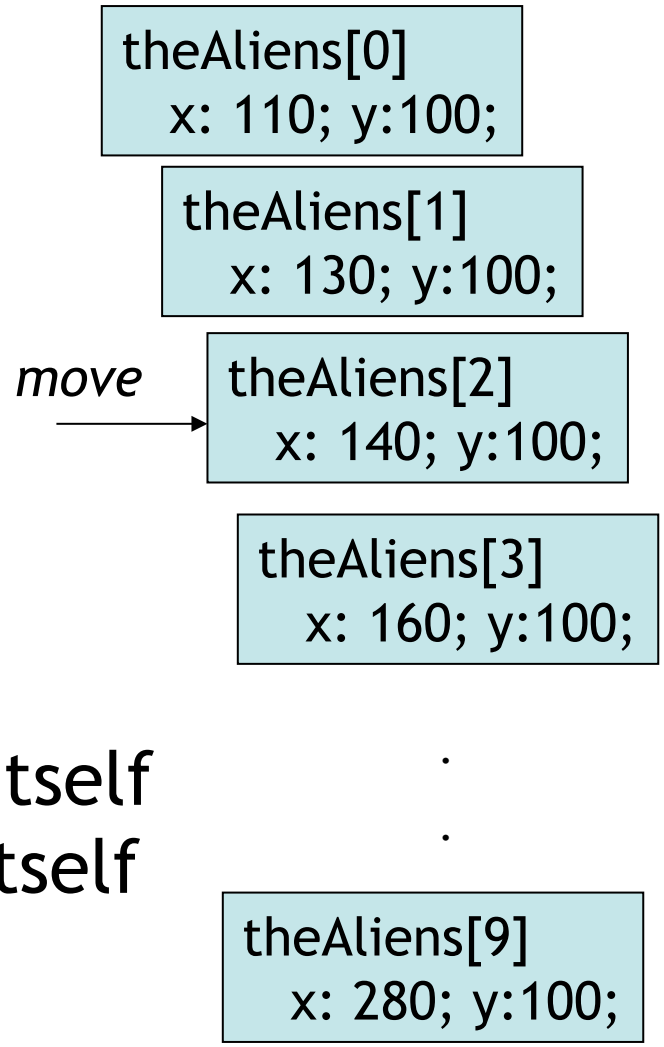
draw()

- ask each alien object to move itself
- ask each alien object to draw itself

The alien example

Main program
setup()
- create an array to hold Aliens
- fill in the array with new Aliens

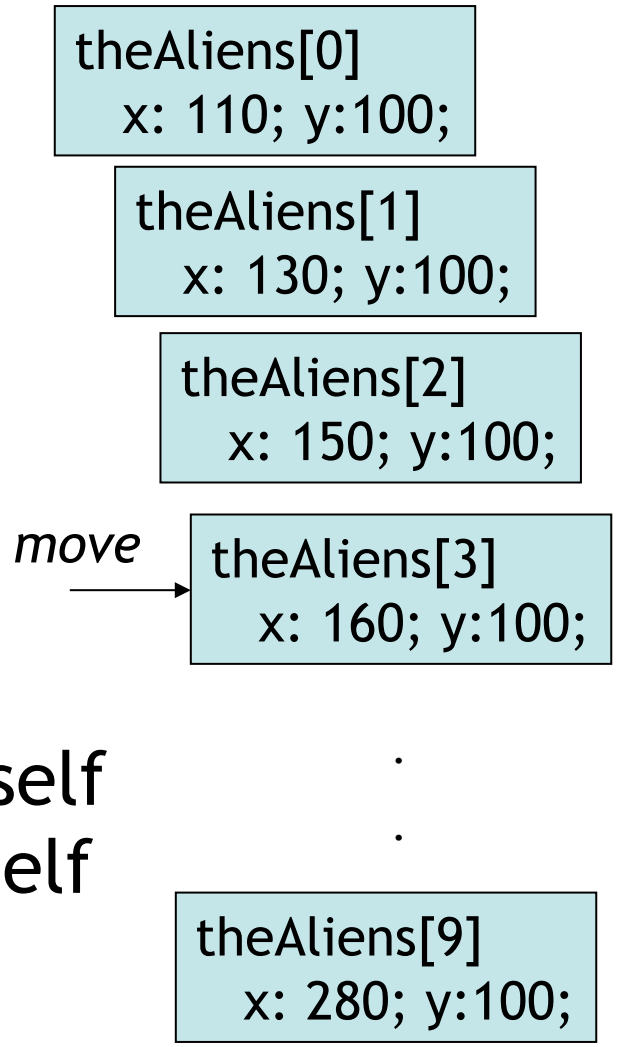
draw()
- ask each alien object to move itself
- ask each alien object to draw itself



The alien example

Main program
setup()
- create an array to hold Aliens
- fill in the array with new Aliens

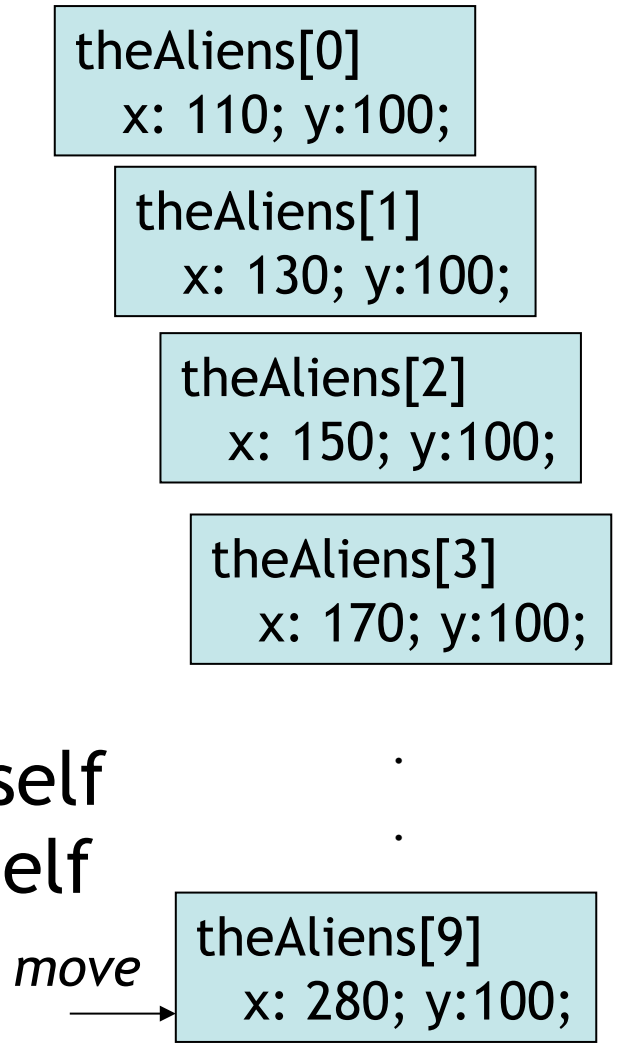
draw()
- ask each alien object to move itself
- ask each alien object to draw itself



The alien example

Main program
setup()
- create an array to hold Aliens
- fill in the array with new Aliens

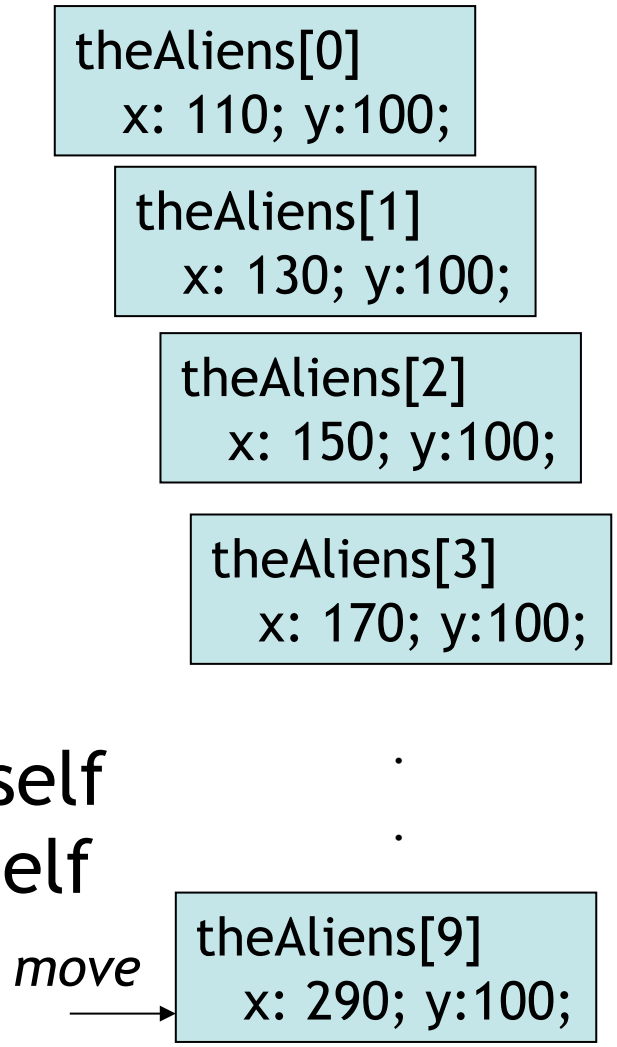
draw()
- ask each alien object to move itself
- ask each alien object to draw itself



The alien example

Main program
setup()
- create an array to hold Aliens
- fill in the array with new Aliens

draw()
- ask each alien object to move itself
- ask each alien object to draw itself



Arrays of objects

- We see a similar pattern to a lot of our code:

Pseudo-code:

```
for (each position in the array) {
   tell the object at that position to do
  something.
}
```

Arrays of objects

- Overall pattern will be:
  - Declare the array – this creates a placeholder for it.
  - Create the empty array with a particular number of positions.
  - Fill in the array
    - For each position in the array, create a new object.
    - What happens if we don't do this?
- To draw the array
  - For each position in the array, call the draw method for the object in that position.

Initialisation

```
Alien theAliens[];

void setup(){
  PImage normalImg, explodeImg;

normalImg= loadImage("invader.GIF");
  explodeImg = loadImage("exploding.GIF");
  theAliens = new Alien[10];
  init_aliens(theAliens,normalImg, explodeImg);
}
void init_aliens(Alien baddies[], PImage okImg, PImage
   exImg){
  for(int i=0; i<baddies.length; i++){
// This is buggy, what is the problem?
    baddies[i] = new Alien(i*(okImg.width+GAP), 0, okImg,
   exImg);
  }
}
```

class Alien

```
class Alien {
  int x, y, direction;
  int status;
  PImage normalImg, explodeImg;

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

Alien.move

```
void move(){
    if(direction==FORWARD){
      if(x+normalImg.width<SCREENX-1)
        x++;
      else{
        direction=BACKWARD;
        y+=normalImg.height+GAP;
      }
    }
    else if(x>0) x--;
    else {
      direction=FORWARD;
      y+=normalImg.height+GAP;
    }
}
```

Alien.draw & Alien.die

```
void draw(){
    if(status==ALIEN_ALIVE)
      image(normalImg, x, y);
    else if(status!=ALIEN_DEAD){
      image(explodeImg, x, y);
      status++;
    }
    // otherwise dead, don't draw anything
  }

 void die(){
   if(status==ALIEN_ALIVE)
     status++;
 }
```

Constants

```
final int SCREENX=400;
final int SCREENY=400;
final int GAP=10;
final int ALIEN_ALIVE=0;
final int ALIEN_DEAD=6;
final int FORWARD=0;
final int BACKWARD=1;
```

Main draw method

```
void draw(){
  background(0);
  for(int i=0; i<theAliens.length; i++){
    theAliens[i].move();
    theAliens[i].draw();
    if(random(0, 500)<1)theAliens[i].die();
  }
}
```

Scope

- In processing we use a fair number of global variables.
- This is fine, but we should consider carefully what needs to be global and what does not.
- If something should be global and isn't, it will disappear as soon as the enclosing method finishes.
- If something should be local and isn't, it makes our program messier and more difficult to understand, change and maintain.

So far:

- A Player which follows the users movement across the screen.
- A Ball which moves about the screen which we can ask whether it has hit anything.
- Aliens which flow down the screen, randomly exploding.

Exercise 4

1. Building on the ***Alien*** class from last week (or the sample code given in the lecture), and the ***Player*** class from week 2, add a ***Player*** object to your program so that the ***Player*** can be moved around at the bottom of the screen while the ***Aliens*** move across and down the screen. (4 marks).

Exercise 4

2. Add a ***Bullet*** class with a **move** method which moves the ***Bullet*** up the screen, and a **draw** method which draws the bullet. An instance of the ***Bullet*** class should be created at the ***Player's*** position when the mouse button is pressed. (3 marks)

Exercise 4

3. Implement a **collide** method in the ***Bullet*** class to check whether the bullet has collided with an ***Alien***. The array of ***Aliens*** should be passed as an argument to this method. Note - A single bullet can hit multiple aliens, these are futuristic bullets. (3 marks)