Week 7 Lab – Collections

# Lab Intro

Although the lab this week is dedicated to the test, you can attempt these exercises after the lecture as a way of practising managing a collection of objects of the same type. We will be looking at collections in the two webinars that follow the week 7 lecture, and next week (week 8) we will continue to use them in the lab when we move over to console applications.

## Learning Outcomes

* **Manage** multiple objects of the same type using a **static** collection (array)
* **Manage** multiple objects of the same type using a **dynamic** collection (ArrayList)

## Resources

* Week 7 slides

# Exercise 1 – Array of Circles

This exercise will involve creating an **array** of circle objects and rendering all of them to the canvas (see Figure 1).

**Key Term – Array**An **array** is a fixed-size data structure that can store data of the same type.



1. Create a **void setup** procedure that sets the size of the canvas to 500x500
2. In a new tab, create a class named **Circle** and give it variables for **x, y, size,** and **colour**.
   * The Circle class should have a **constructor** that accepts an **x** and **y** value to be passed to it. The constructor should also initialise the size to be a random value from 20 to 50 and set the colour variable to a random colour

**Key Term – Iterate**Iterate means to ‘travel’ through a collection (e.g. array) and visit each of the elements.



* + Add a **void render()** method to this class that will simply render an ellipse at the circle object’s x and y. Before you render this ellipse, you will want to change the fill colour to match the colour variable of the circle

1. Outside of setup/draw, declare an array to hold 100 **circles**
   * Remember that you will need to declare an array variable using the following syntax: **DataType[] arrayName;**
2. Inside **void setup**, create the array (**remember:** arrays themselves are also objects, so they need to be created). **Syntax: arrayName = new DataType[size];**
3. Inside **void setup**, **iterate** through the array, calling each of the object’s **render** method

Chart, background pattern, bubble chart

Description automatically generated

Figure Exercise 1 Output

# Exercise 2 – Array of Motorbike Objects

In week 5, you created a program that involved creating a few motorbike objects which would race each other across the canvas. This exercise will involve creating an array of 10 motorbike objects and getting them to race across the canvas (see Figure 2).

1. Create a copy of this program and remove all of the individual motorbike objects from the code (remove the declarations at the top and any code that makes use of the motorbikes)
2. Create an array of Motorbikes (you can declare the array outside of **setup**/**draw**, and then create the array inside **setup**)
3. Iterate through the motorbikes array and call each other the motorbike object’s update method

**Optional:** Implement the scoring functionality that was also in the week 5 lab to record how many times each bike has won.

Diagram

Description automatically generated with medium confidence

Figure Array of Motorbikes racing

# Exercise 3. Using an ArrayList of Objects

In a new sketch add the code below

ArrayList<IntObject> intArrayList; //declare an ArrayList of class IntObject

void setup()

{

intArrayList; = new ArrayList<IntObject>(); //create an empty list

//add numbers 1 to 10

IntObject numberObject;

for (int number=1; number<=10; number++)

{

numberObject = new IntObject(number);

intArrayList.add(numberObject);

}

}

And add the class below to a new tab

class IntObject

{

int value;

IntObject(int number)

{

value = number;

}

}

What does it do?

Comment your code appropriately.

Ex 3.A. How would we print to console all the numbers in the ArrayList? Add a suitable **procedure** and test it.

Alter the code so that the arrayList contains the numbers 16,14,12,10,8 in that order.

Ex 3.B How would we sum all the numbers in the ArrayList and print to console? Add a suitable **function** that will **return** the total and test it. We covered functions in Week 2 on moodle.

Ex 3.C Write a **Boolean function** that will search for a specified value in the ArrayList and return true if it’s there, false otherwise. Test it.

# Exercise 4 – ArrayList of ‘Bugs’



This exercise will involve using both an array of colours and an **ArrayList** of ‘bugs’ (Figure 3).

* The ArrayList will store ‘bug’ objects (it will be empty initially, and a new bug will be added every time the mouse is clicked in the canvas)
* The **array** will store **5** different colour values that a bug can take:

**Key Term – ArrayList**An **ArrayList** is a **dynamically sized** data structure that can store data of the same type and can grow/shrink as things are added/removed.



* + **red** – fill(255, 0, 0);
  + **green** – fill(0, 255, 0);
  + **blue** – fill(0, 0, 255);
  + **teal** – fill(0, 255, 255);
  + **yellow** – fill(255, 255, 0);

Copy over the starter code found in Table 1 into a new sketch and use the comments to help you finish this exercise.

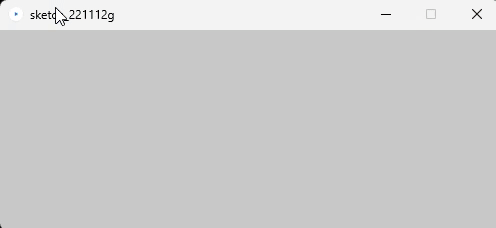


Figure Exercise 3 Desired Output

After doing this, refactor the code so that instead of generating a random colour every time a bug object is created, it cycles through the colours (e.g. the first bug added will always be red, followed by green, etc.).

Table Exercise 4 Starter Code

|  |  |
| --- | --- |
| **Main Tab** | **Bug Tab** |
| /\* TODO  Create an array of the color data type (size of 5)  Create an ArrayList of Bug objects  \*/  color[] colours = new color[5];  ArrayList<Bug> bugs;  void setup() {  size(500, 200);  bugs = new ArrayList<>();  /\* go inside the colours array and set each index's value  e.g. 1st element = color(255, 0, 0); \*/  }  void draw() {  background(200);  /\* loop through the bugs ArrayList and call  each of the bug's update method \*/  }  void mousePressed() {  /\* add a new Bug to the bugs ArrayList at  the mouseX and mouseY location \*/  } | class Bug {  float x;  float y;  float speed = 1.5;  float size = 30;  color bodyColour;  Bug(float x, float y) {  this.x = x;  this.y = y;  /\* TODO 1: generate a random number from 0  to the length of the colours array and use  that value for the bug's body colour\*/  }  void move() {  x += random(-speed, speed);  y += random(-speed, speed);  }  void render() {  fill(bodyColour);  ellipse(x, y, size, size);  fill(0);  ellipse(x - size/4, y, size/3, size/2);  ellipse(x + size/4, y, size/3, size/2);  }  void update() {  move();  render();  }  } |

Extension exercise: **Simple collecting game**

Write a simple game where we can move a green circle around the screen (use the mouse) to collect (remove) multiple red squares.

Top down design what do we need? What Classes? What Events?