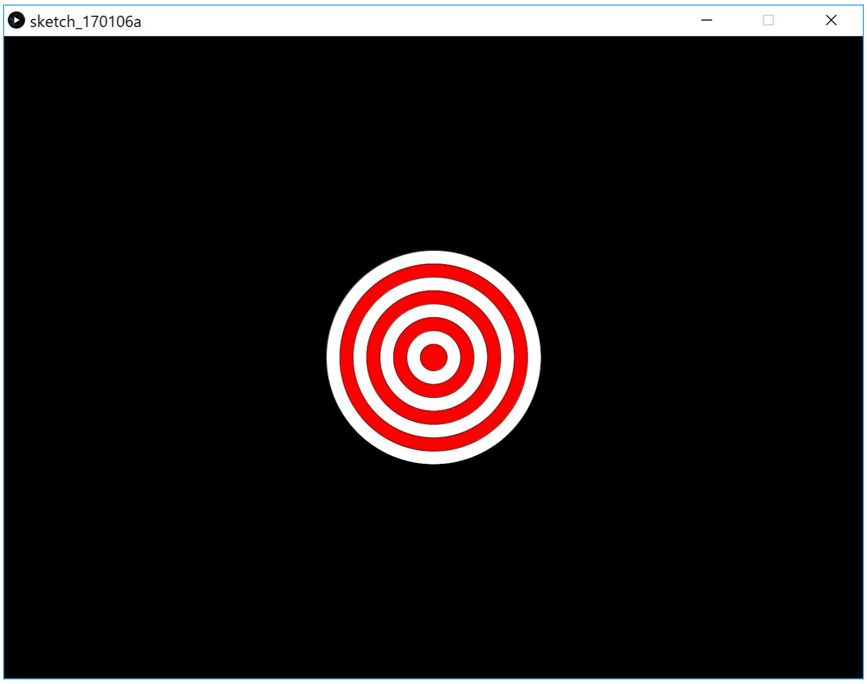
Targets

Start with the Grade 0 instructions.

When you complete a grade get a teacher to check your progress before moving on to the next grade. Make sure a teacher sees your progress before the end of class.

Grade 0 - Start Here

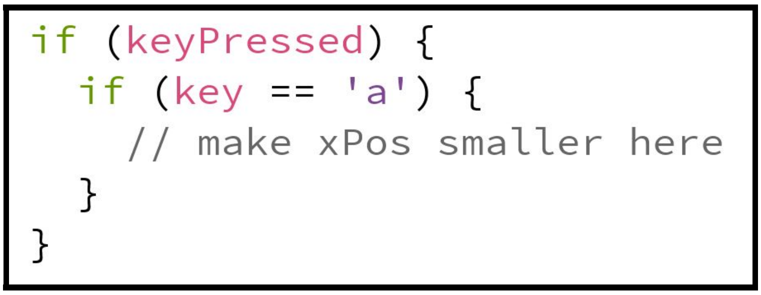
1. Use ellipse and fill to create a red and white target on a black background, like this:



2. Make it follow the mouse (all circles positioned at mouseX, mouseY). Ensure there are no trails. REMEMBER: You will need to use void setup() and void draw() here.

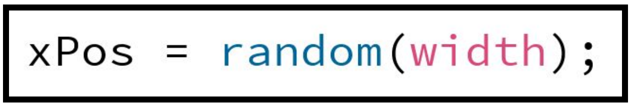
Grade I

1. Use variables (call them xPos and yPos) to control the position of the target.
    a. HINT: declare them as type *float* instead of *int* - this will let us assign a random value much more easily later on.
    b. HINT 2: Use "Find and Replace" to change mouseX to xPos and mouseY to yPos - this saves you a lot of typing.
2. Add keyboard controls to move the target left, right, up and down.
    a. HINT: be *very* careful with brackets! This should get you started:



```
if (keyPressed) {
  if (key == 'a') {
    // make xPos smaller here
  }
}
```

3. Add a key that gives the target a random position.
    a. HINT: You'll need to give xPos and yPos random values. This code gives xPos a random value between 0 and *width* (can you guess what *width* is?):



```
xPos = random(width);
```

Grade 2

Task: Create a class "Target", which will allow us to create objects of type “Target”. Create a new object every time you click (overwriting previous object).

Classes allow us to design objects, which are like more complex variables. If you declare an *int*, that is just a number. When we create a *Target*, that can contain a number of different numbers - for example, xPos, yPos, diameter. We put this code in a new tab, which we will call Target. Let’s start by declaring the class like this (we’ll add more bits later):

```
class Target {
  float xPos;
  float yPos;
  float diameter;
}
```

Note the capital letters here.

- A class name should *always* be in **U**pper**C**amel**C**ase.
- A variable or object name should *always* be in **l**ower**C**amel**C**ase.
- In this case the class will be Target, and if we only have one object of type *Target*, we will call it *target*.

To create an object of type *Target* there are two steps other steps after copying the code above:

1. *Declare* the new object.
2. *Initialise* the new object.

```
Target target; // Step 1 - declare

void setup() {
  size(1600, 1200);
  target = new Target(); // Step 2 - initialise
}

void draw() {
  target.xPos = 500;
  target.yPos = 400;
  target.diameter = 300;

  ellipse(target.xPos, target.yPos, target.diameter, target.diameter);
}
```

As you can see, we access different bits of the object using “dot notation”. To get to the xPos of *target*, we write target.xPos.

The code above only draws a circle. This is not really what we’re after! Storing variables like this can be useful, but classes really become important when we add *methods*. These are bits of code stored inside the class, which do something related to the class. In this case, it would be really useful to have a method target.drawTarget(), and have that code draw a target of whatever size and position we have stored in the target object. Our class will now look like this:

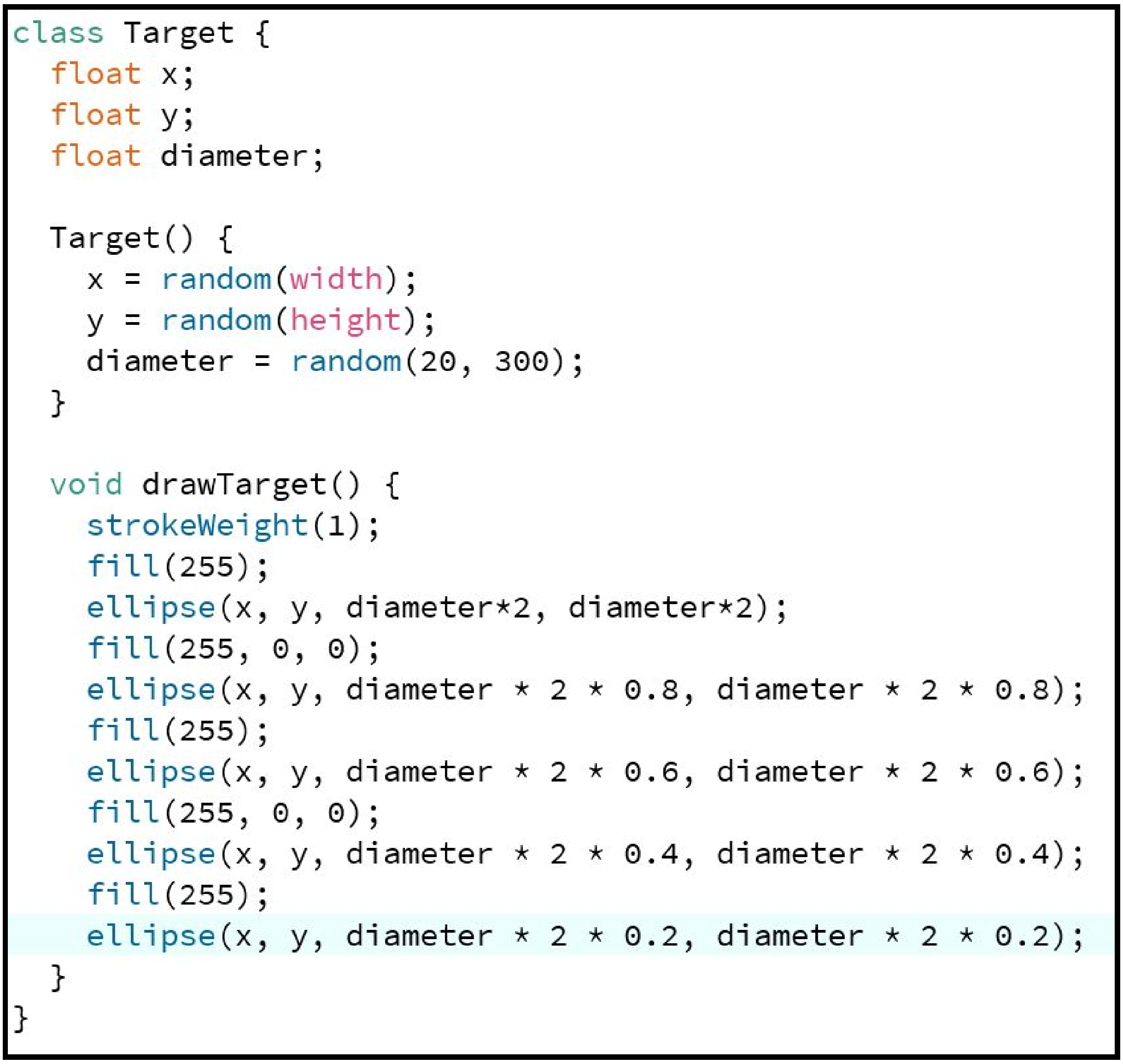
```
class Target {
  float xPos;
  float yPos;
  float diameter;

  void drawTarget() {
    background(0);
    fill(255);
    ellipse(xPos, yPos, diameter, diameter);
    fill(255, 0, 0);
    ellipse(xPos, yPos, diameter*0.8, diameter*0.8);
    fill(255);
    ellipse(xPos, yPos, diameter*0.6, diameter*0.6);
    fill(255, 0, 0);
    ellipse(xPos, yPos, diameter*0.4, diameter*0.4);
    fill(255);
    ellipse(xPos, yPos, diameter*0.2, diameter*0.2);
  }
}
```

Now instead of having all those fills and ellipses cluttering up our main code, they're stored in the Target class. All we have to do is set the position of the target and then call "target.drawTarget()".

Once you have this working, declare a second object of type *Target*. Call it *target2*. Rename the first one *target1*.

Constructor

The final part we need for our objects is a *constructor*. When we write "target = new Target()" the *new* keyword triggers the *default constructor*. This creates a new object of type target, and sets the variables to their default values (in our case, xPos, yPos and diameter will all be zero). Wouldn't it be convenient to declare our own constructor? Perhaps we could set all of those variables to be random instead? We can, and it looks like this:



```
class Target {
  float x;
  float y;
  float diameter;

  Target() {
    x = random(width);
    y = random(height);
    diameter = random(20, 300);
  }

  void drawTarget() {
    strokeWeight(1);
    fill(255);
    ellipse(x, y, diameter*2, diameter*2);
    fill(255, 0, 0);
    ellipse(x, y, diameter * 2 * 0.8, diameter * 2 * 0.8);
    fill(255);
    ellipse(x, y, diameter * 2 * 0.6, diameter * 2 * 0.6);
    fill(255, 0, 0);
    ellipse(x, y, diameter * 2 * 0.4, diameter * 2 * 0.4);
    fill(255);
    ellipse(x, y, diameter * 2 * 0.2, diameter * 2 * 0.2);
  }
}
```

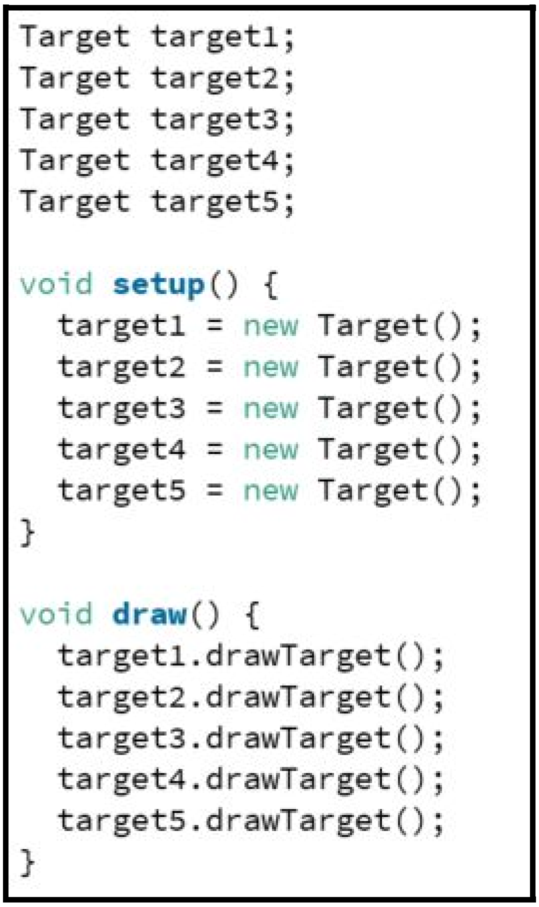
Note that we've added the Target() function, **without the word void in front**. This overrides the default constructor.

There are other things we can do with the constructor to make this more useful (for example, setting the xPos, yPos and radius to a specific value when we call "new Target()"), but we'll leave that as it is for now.

Grade 3

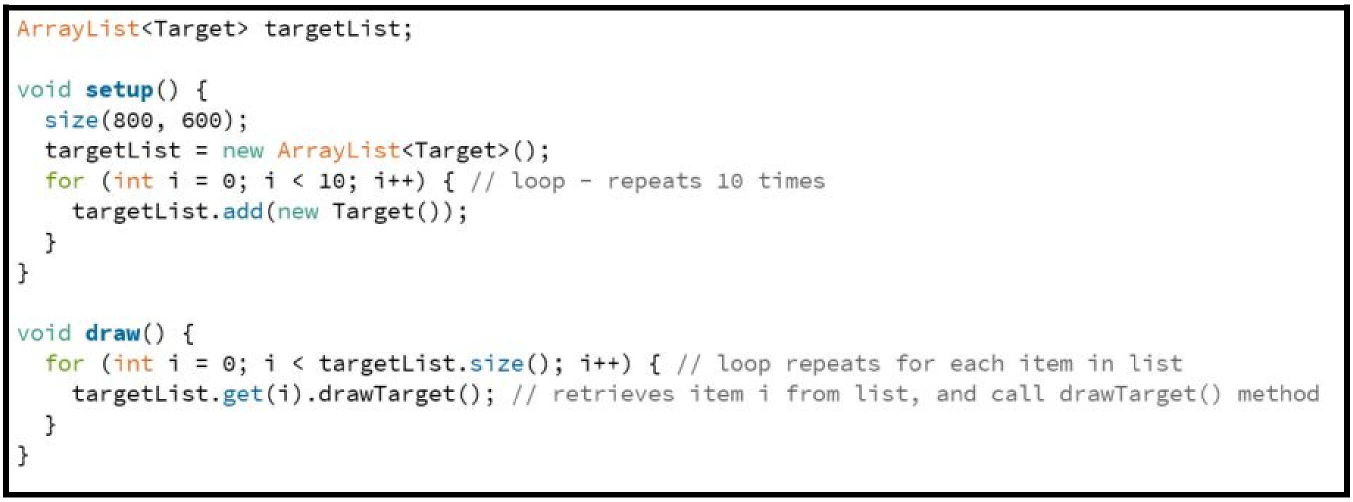
Task: Add an ArrayList to store targets. Provide multiple ways to add new targets (eg click, automatically every second).

ArrayLists provide a really neat way of storing lots of objects of the same type. Let's imagine we have multiple targets we want to create and store. We could do it one at a time (imagine how this would scale for 10, 100, 1000…):

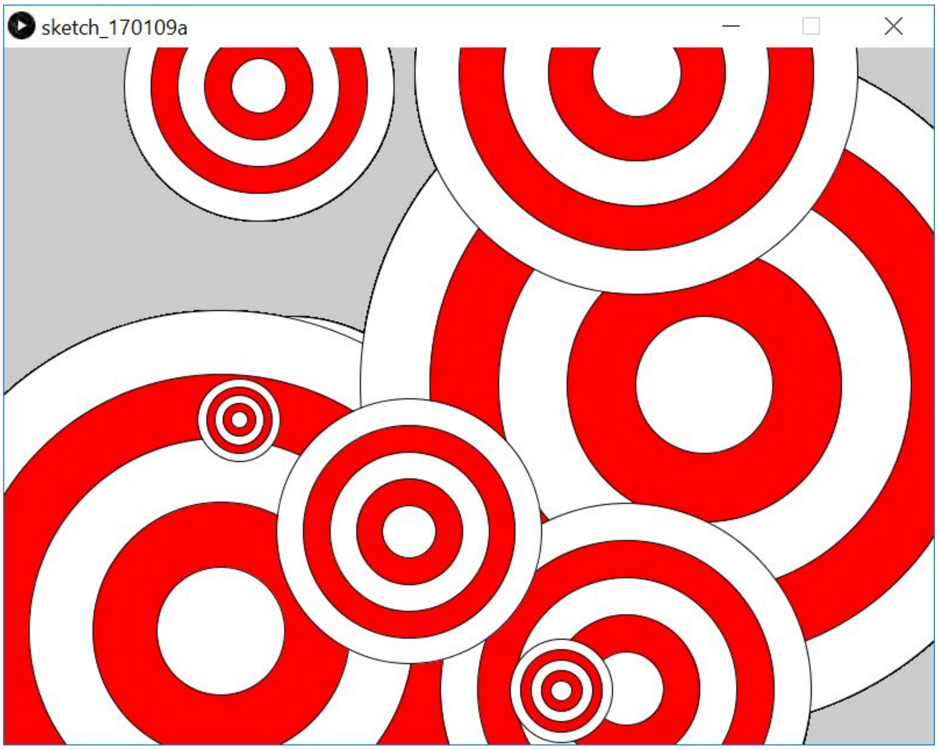


```
Target target1;
Target target2;
Target target3;
Target target4;
Target target5;

void setup() {
  target1 = new Target();
  target2 = new Target();
  target3 = new Target();
  target4 = new Target();
  target5 = new Target();
}

void draw() {
  target1.drawTarget();
  target2.drawTarget();
  target3.drawTarget();
  target4.drawTarget();
  target5.drawTarget();
}
```

Alternatively we could use an ArrayList. This is a little trickier up front, but gives us the structure we can use to scale up to tens, hundreds or thousands of objects. Change the **10** in the first for loop and this will scale to almost as many objects as you like.



```
ArrayList<Target> targetList;

void setup() {
  size(800, 600);
  targetList = new ArrayList<Target>();
  for (int i = 0; i < 10; i++) { // loop - repeats 10 times
    targetList.add(new Target());
  }
}

void draw() {
  for (int i = 0; i < targetList.size(); i++) { // loop repeats for each item in list
    targetList.get(i).drawTarget(); // retrieves item i from list, and call drawTarget() method
  }
}
```



The key part of this to understand is that to add a new target to targetList we use the line “targetList.add(new Target());”, and to retrieve it we use “targetList.get(i)”, where i is a number. If i is 0 we are retrieving item 0 from the list, and so on. We often use a for loop (as in the example) to retrieve every item in the list, one at a time.

Grade 4+

Add a user controlled cannon in the corner which can shoot shells at the targets. You can do the shells much like you did the bouncing balls in the relevant lesson. You will now have classes as follows:

- Target
- Cannon
- Shell
- Game

The Game class will contain the targets, cannon and shells, as well as a lot of the methods we will need (eg for detecting collisions between shell and target). This allows us to keep the main setup and draw blocks quite neat (they should only contain user input code, and perhaps some background and text functions.