# Lab Week 5. Classes and Objects

### Topic so far

* Variables, local and global
* Loops
* Procedures (void) and parameters
* Functions – return a value
* Animation & Events – setup, draw, keyPressed etc

## Learning Objectives

* Understanding and using a Class
* Creating objects (instances of a Class)
* Manipulating object behaviour using its methods
* Using a constructor

## Resources

* Lecture Notes
* Extra tutorials – see moodle
* Example Ball class – download from moodle and take a look. Note the members, constructor, methods and how it’s used from the setup and draw events
* Processing website – reference
* Objects tutorials on moodle – read for a different explanation and examples

Save your code after each exercise

***Using a class***. In webinarA we looked at creating two instances of a class Ball and using its methods to manipulate the instances. We’ll start with a simple class and add methods to improve our program over the next few exercises.

**Ex1.** What variables (members) would a class to represent each of the following require?

1. pong bat
2. space invader (move left, right and down)
3. person
4. student

**Ex2.** Copy and paste the class code below and create **a single instance** of a **red motorbike** that moves from left to right across the screen. Add a **2nd blue motorbike** moving in the same fashion. Lastly make the red bike win the race.

final color RED = color(255,0,0); //constant

final color BLUE = color(0,0,255);

class Motorbike

{

int x = 5; //members

int y;

int speed=2;

int size=30;

color colour;

//methods, procedures an object of this type can perform

void render()

{

float wheelHeight = size/3;

fill(colour);

triangle(x,y,x+size,y,x+size/2,y-size/2); //built-in triangle routine

drawWheel(x,y,wheelHeight);

drawWheel(x+size,y,wheelHeight);

}

void drawWheel(int x,int y,float size)

{

float inner = size\*2/3;

fill(0);

ellipse(x,y,size,size);

fill(255);

ellipse(x,y,inner,inner);

}

} //end of class description

void setup()

{

size(500,100);

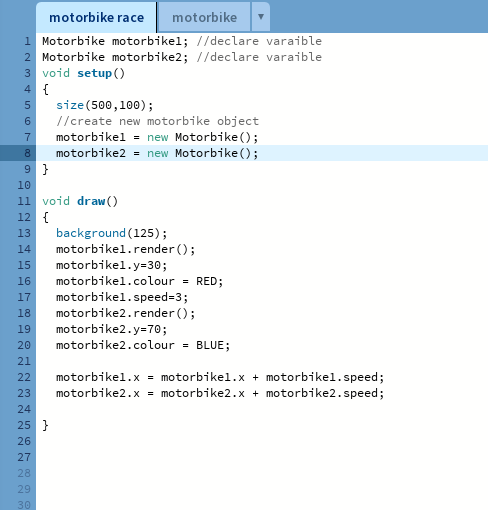
}

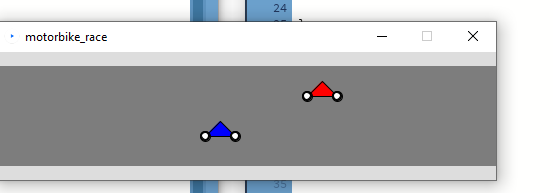
void draw()

{

background(125);

}





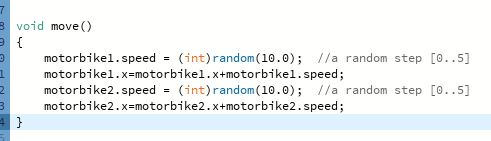
**Ex3.** Adding more methods – improving our code. Let’s add a **move** method below to the Class and alter our calling code to make use of it.

void move() {

speed = (int)random(5.0); //a random step [0..5]

x=x+speed;

}



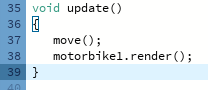
**Ex4.** Let’s add the **update** method below and alter our calling code to make use of it

void update() {

move();

render();

}



**Ex5.** Add the constructor below and alter your code accordingly

Motorbike(int y,color col){ //constructor

this.y = y;

this.speed = (int)random(5.0);

this.colour = col;

}

**Ex6**. Let’s add the **finished** function method below and alter our **draw event** so the bike only moves if it hasn’t finished.

boolean finished()

{

return x>(width-10); //screen width

}

**Ex7. Game modes**, we can set up different behaviours within the draw event dependent on a ***gameMode*** variable. If this variable has one value then one set of commands are used, each value can allow a specific set of commands to be used (which may or may not involve animation or redrawing the screen). Add this to your code and modify the program to allow the winner to be displayed on the screen, and pressing the space bar allows the race to restart.

We need

* If statement(s) for ***gameMode*** in the draw event, draw shouldn’t do anything if ***gameMode*** is FINISH
* KeyPressed event to change gameMode, reset the bikes

//constants and variables : At top of code

final int RACING=0;

final int FINISH=1;

int gameMode = RACING;

**Ex8.**  Amend your code so that the race involves 3 bikes and the number of wins for each bike is displayed on the screen. Hint, the code below would display “blue:0” at position 10,10.

int winBlue = 0;

text(“blue:”+winBlue,10,10);

**Ex9. Directional bouncing objects.** On moodle you will find a ZIP file called **direction Images**. Create a new empty program, save the program as “directionalObjects” and unzip the image files into the same directory as your code.

The zip has one image for when the Object is heading each of:

* up and right
* up and left
* down and right
* down and left

1. Create a class called **Bouncer** that allows multiple objects to move in **all 4** directions around the screen whilst using your 4 images. The class should contain a **constructor**, a **render** and a **move** method (perhaps more).
2. Create a program that allows 3 instance objects of your class to move around the screen, bouncing off each other. and off the edges. You will need a collision *function method* in your class. We can pass a parameter of any type including an instance of a class.

boolean crash( **Bouncer** other )

{

return dist(this.x, this.y, other.x, other.y) < 20; //true if distance less than 20

}

When detecting collisions between objects, it’s easier if the **x,y** location of each object is in the **centre** rather than the top left corner. The command “ **imageMode(CENTER);**” can be added to **setup()** and will cause every PImage drawn to the screen to be drawn with x,y at the centre, rather than top left.

[note : if your objects seem to collide and stick together – it’s likely that you are detecting a collision but when they move apart (next move()) a collision is detected again which repeats – objects keep changing direction. Try reducing the value you are using for the distance]

**Ex10.** Similar to the motorbike race above, you should write a program to allow three cars to race from the **bottom** of the screen to the **top**. This will need a class Car and images can be found on moodle which can be stored in a PImage (covered in Monday’s webinar). There is no animation sequence necessary.

Your Car class should have

* suitable member variables – what do we need to know to draw the car in its current position and move it up the screen?
* Constructor – allow position and an image file to be passed in
* Render or display method – draw the car in its current position
* Move method – change the position variable values – use the random function to provide a random value [1..3], random function, e.g.
  + x= x + (int) random(1,3); //random value between 1 & 3.
* Update method – call the move and render methods

Add in game modes to control the draw event and stop them moving once they have finished (reached the top of the screen). Use the text command to state which car won the race (may be a delay with this command).

//constants and variables : At top of code

final int RACING=0;

final int FINISH=1;

int gameMode = RACING;

**Extension exercises**, modify your code to allow one of the cars to be controlled by the user – moving that particular car by pressing a specific key (or a mouse click). One key press should move the bike a set value (try 15 pixels). You can also add a start screen.

Look at and experiment with the example at <https://processing.org/examples/objects.html> add another instance of the MRect class.