Lab Week 6. Objects

Last Week

- Using functions
- Implementing functions
- Text command

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 {take a quick look at this}
- Processing website – reference
- Objects tutorials on moodle {take a quick look at this}

Save your code after each exercise

Using a class. In the lecture we looked at creating two instances of a class 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. Download from moodle (or 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);
final color BLUE = color(0,0,255);

class motorbike
{
 int x = 5;  //members
 int y;
 int speed=2;
 int size=30;
 color colour;

 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);
}
```

Ex2. 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;
 }
```

Ex3. Let's add the **update** method below and alter our calling code to make use of it

```
void update()  {
   move();
   render();
 }
```

Ex4. 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;
 }
```

Ex5. 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
 }
```

Ex6. 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;
```

Ex7 (portfolio). 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);

Extension exercises, modify your code to allow one of the bikes to be controlled by the user – moving that particular bike 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.