Processing Notes

Chapter 13: Classes/Objects

We now have enough programming power to do all sorts of interesting things. But the tools we have available are sort of awkward. Just as a early craftsmen were able to fabricate amazing furniture and art with simple tools, modern tools make manufacture faster and easier. Computer Science is about making better (virtual) tools and finding new applications. Because it is a virtual world, we can make our own tools. One big advance in tool building is called **Object Oriented Programming.** You may not realize it, but you have implicitly been using an OO approach with some of your programming already. We have created PImage **objects** and used **methods** on those objects. A method is just a function, but, it is a function tied to a particular **class.** Lets develop an working example of a better virtual tool: a class to manipulate moving rectangles more easily.

```
class MovingRectangle
{
  // member data variables
  float rx, ry, rwidth, rheight ;   // the x, y, width, and heigh of this rect
  float velX, velY ;   // the x and y-velocties
  color currentColor ;

  MovingRectangle(float inRx, float inRy, float inRwidth, float inRheight,
              float inVelx, float inVely, float inR, float inG, float inB)
  // constructor number 1
  {
    rx = inRx ;
    ry = inRy ;
    rwidth = inRwidth ;
    rheight = inRheight ;
    velX = inVelx ;
    velY = inVely ;
    currentColor = color(inR,inG,inB) ;
  }


  // setter methods
  void setX(float inRx) { rx = inRx;}
  void setY(float inRy) { ry = inRy;}
  void setWidth(float inW) { rwidth = inW;}
  void setHeight(float inH) { rheight = inH;}
```

```
  void setVelX(float inVx) { velX = inVx;}
  void setVelY(float inVy) { velY = inVy;}
  void setColor(float r, float g, float b) {  currentColor = color(r,g,b) ; }

  // getter methods
  float getX() { return rx;}
  float getY() { return ry;}
  float getWidth() { return rwidth;}
  float getHeight() { return rheight;}
  float getVelX() { return velX;}
  float getVelY() { return velY;}
  color getColor() {return currentColor;}


  void updateLocation()
  // change the x,y location to be current plus velocities
  {
    rx += velX ;
    ry += velY ;
    if ( (rx < 0) || ((rx+rwidth) > width) )
      velX *= -1 ;
    if ( (ry < 0) || ((ry+rheight) > height) )
      velY *= -1 ;

  }

  void drawRect()
  // set the fill to current color and call Procesing rect()
  // command to draw the rectangle
  {
    fill(currentColor) ;
    rect(rx,ry,rwidth,rheight) ;
  }


}    // close of the class definition
```

Above is the class definition for a MovingRectangle class. The general format is:

* class className
* member variables
* constructor
* getter methods
* setter methods
* additional methods

The member variables are the holder of data necessary for the MovingRectangle objects. In this case we need to keep track of the rectangle x-coordinate, y- coordinate, width, height, x-velocity, y-velocity, and color. Hence, each of these is a member variable. Next is the **constructor**. The constructor is a special method that is called when an object is created. It contains the code to initialize the object. Next are the "getter" methods. A getter method is a method (i.e. function belonging to this class) that returns the current value of the member variables. Next are the setter methods. These methods allow the user of the class, i.e. the person writing code to manipulate MovingRectangle objects, to change member variable values. Finally, come the additional methods. In this case we have included two additional methods: updateLocation( ) and drawRect( ). The first is called when we wan the object to move. It does so by changing its x and y coordinate by the amount specified in velX and velY. The drawRect() method is called when we want to draw the rectangle.

The following code shows how to create and use MovingRectangle objects:

```
MovingRectangle mr1, mr2, mr3, mr4, mr5 ;

void setup()
{
  size(400,400) ;
  fill(0) ;

  mr1 = new MovingRectangle(10,10,20,20,2,3,255,0,255) ;
  mr2 = new MovingRectangle(10,100,20,20,2,3,255,0,255) ;
  mr3 = new MovingRectangle(100,10,20,20,2,3,255,0,255) ;
  mr4 = new MovingRectangle(300,10,20,20,2,3,255,0,255) ;
  mr5 = new MovingRectangle(10,300,20,20,2,3,255,0,255) ;
}

void draw()
{
  background(100) ;

  mr1.updateLocation() ;
  mr1.drawRect() ;
  mr2.updateLocation() ;
  mr2.drawRect() ;
  mr3.updateLocation() ;
  mr3.drawRect() ;
  mr4.updateLocation() ;
  mr4.drawRect() ;
  mr5.updateLocation() ;
  mr5.drawRect() ;

  if (mr5.getX() > width/2)
    mr5.setColor(255,0,0) ;
  else
    mr5.setColor(255,0,255) ;

  if (mr4.getX() > width/2)
    mr4.setWidth(50) ;
  else
    mr4.setWidth(20) ;
}
```

To use this code the class definition code needs to be in the same Processing sketch file (actually, there is a way around this, but this is the easiest way for now). The class

definition code is not repeated in the example above, but, realize it must be there else Processing will inform you it does not know what a MovingRectangle is.

The first line:

```
MovingRectangle mr1, mr2, mr3, mr4, mr5 ;
```

defines 5 variables that can hold objects of type MovingRectangle. As shown in the setup( ) function we need to CREATE the objects by calling **new( )** as follows:

```
mr1 = new MovingRectangle(10,10,20,20,2,3,255,0,255) ;
```

What happens is the constructor is called with that big list of parameters. If you look closely at the constructor you will see it initializes all the member variables with the values passed in as parameters. Thus, for the line above, a new MovingRectangle object is created with x and y set to 10, width and height set to 20, velX and velY set to 2 and 3 respectively, and the color set to (255,0,255). The code above creates 5 new objects and puts them in variables mr1, mr2 , mr3, mr4, and mr5. In the draw( ) function the updateLocation( ) and draw( ) methods are called for each of these 5 objects. Then, to demonstrate using getters and setters, we make it so the object in mr4 becomes wider when on the right hand side of the screen and the object in mr5 turns red when on the right hand side of the screen.

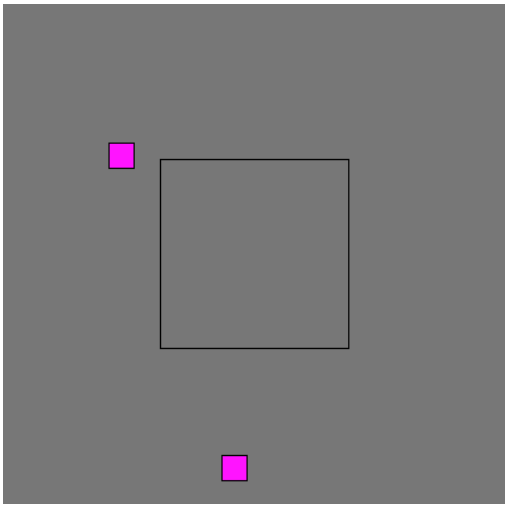
Class definitions can be modified to accomodate changes such as new fuctionality. For example, an obvious thing to add to the MovingRectangle class is a method that determines if the rectangle is intersecting another rectangle. Consider the following method:

```
boolean intersectRectangle(float inOtherX, float inOtherY,
    float inOtherWidth, float inOtherHeight)
 {
   if (
    ( ( (rx < inOtherX) && (rx+rwidth > inOtherX) )  ||
    ( (rx >  inOtherX) && (rx < inOtherX+inOtherWidth) ) )
    &&
    ( ( (ry < inOtherY) && (ry+rheight > inOtherY) )  ||
    ( (ry >  inOtherY) && (ry < inOtherY+inOtherHeight) ) )
    )
    return (true) ;
  else
    return(false) ;
  }
```

If we add this method inside the class definition, we can not make calls to it just as we do to the update( ), draw( ), and getter/setter methods. Consider the following code:

```
void draw()
{
  background(100) ;

  noFill() ;
  rect(width/2-75,height/2-75,150,150) ;

  mr1.updateLocation() ;
  if ( !mr1.intersectRectangle(width/2-75,height/2-75,150,150) )
    mr1.drawRect() ;

  mr2.updateLocation() ;
  if ( !mr2.intersectRectangle(width/2-75,height/2-75,150,150) )
    mr2.drawRect() ;

  mr3.updateLocation() ;
  if ( !mr3.intersectRectangle(width/2-75,height/2-75,150,150) )
    mr3.drawRect() ;

  mr4.updateLocation() ;
  if ( !mr4.intersectRectangle(width/2-75,height/2-75,150,150) )
    mr4.drawRect() ;

  mr5.updateLocation() ;
  if ( !mr5.intersectRectangle(width/2-75,height/2-75,150,150) )
    mr5.drawRect() ;
}
```

This code assumes we have first created 5 MovingRectangle objects and put them in mr1 .. mr5 as above. In this case every time we first draw a non-filled static rectangle in the center of the screen. Then we update the location of each MovingRectangle object and check to see if it intersects the center rectangle. We do this check by calling the MovingRectangle intersectRectangle( ) method. If the moving rectangle object does not intersect the center rectangle we draw it. A static screen shot of this might look like the following, where at this instant of time the screen shot was grabbed three of the 5 moving rectangles were located inside the center rectangle and hence not drawn:



The intersectRectangle test will test if a given object intersects with a specified rectangle. You may want to see if it intersects with another MovingRectangle. A method to do that is:

```
boolean intersectRectangle(MovingRectangle inOtherRect)
{
  float otherX = inOtherRect.getX() ;
  float otherY = inOtherRect.getY() ;
  float otherWidth = inOtherRect.getWidth() ;
  float otherHeight = inOtherRect.getHeight() ;

  if (
   ( ( (rx <= otherX) && (rx+rwidth >= otherX) )  ||
   ( (rx >=  otherX) && (rx <= otherX+otherWidth) ) )
   &&
   ( ( (ry <= otherY) && (ry+rheight >= otherY) )  ||
   ( (ry >=  otherY) && (ry <= otherY+otherHeight) ) )
   )
   return (true) ;
 else
   return(false) ;
}
```

If you insert this into your class you can then make calls to it. Note, this method has the exact same name as the first method. This is okay as long as the signatures (i.e. the number and types of parameters) are different. This is called **overloading.** We are overloading the name “intersectRectangle” to have two different meanings, and the language can figure out which one you mean by the signature of the method call. Consider the following code that uses this method:

```
void setup()
{
  size(400,400) ;
  mr1 = new MovingRectangle(10,10,60,60,2,3,0,0,0) ;
  mr2 = new MovingRectangle(10,100,60,60,2,3,0,0,0) ;
  mr3 = new MovingRectangle(100,10,60,60,2,3,0,0,0) ;
}

void draw()
{
  background(100) ;

  mr1.updateLocation() ;
  mr2.updateLocation() ;
  mr3.updateLocation() ;

  if (mr1.intersectRectangle(mr2) || mr1.intersectRectangle(mr3) )
    mr1.setColor(255,0,0) ;
  else
    mr1.setColor(0,0,0) ;

  mr1.drawRect() ;
  mr2.drawRect() ;
  mr3.drawRect() ;

}
```

In this example we create three moving rectangle objects. In draw( ) we update their locations. If the object in mr1 intersects either of the other two moving rectangles we set the color of the object in mr1 to be red.

Note, we could have made use of our old intersectRectangle method also, but it would have been less elegant:

```
 if (mr1.intersectRectangle(mr2.getX(), mr2.getY(),
     mr2.getWidth(), mr2.getHeight() ) ||
      mr1.intersectRectangle(mr3.getX(), mr3.getY(),
     mr3.getWidth(), mr3.getHeight() )       )
    mr1.setColor(255,0,0) ;
  else
    mr1.setColor(0,0,0) ;
```

Remember the hassle it was to write the code for Exercise 12A? This was the one where you had one big rectangle that moved with the mouse and 5 little rectangles that moved on their own. Do do that assignment without classes/object you had to keep track of 30 variables: x, y, velX, velY, width, and height for each of the 5 rectangles. It was easy to make a type that broke the code. Frankly, it was just a hassle (that was the intention of that assignment). Using our MovingRectangle class it is now much easier. The following is a solution to the same assignment using the MovingRectangle class:

```
void setup()
{
  size(400,400) ;
  mr1 = new MovingRectangle(10,10,20,20,2,3,0,0,0) ;
  mr2 = new MovingRectangle(10,100,20,20,2,3,0,0,0) ;
  mr3 = new MovingRectangle(100,10,20,20,2,3,0,0,0) ;
  mr4 = new MovingRectangle(290,10,20,20,2,3,0,0,0) ;
  mr5 = new MovingRectangle(100,300,20,20,2,3,0,0,0) ;
}

void draw()
{
  background(100) ;
  int bigRectWidth = 80 ;
  int bigRectHeight = 80 ;

  // draw a rectangle that moves with the mouse
  fill(0) ;
  rect(mouseX,mouseY,bigRectWidth,bigRectHeight) ;

  mr1.updateLocation() ;
  mr2.updateLocation() ;
  mr3.updateLocation() ;
  mr4.updateLocation() ;
  mr5.updateLocation() ;

  if (mr1.intersectRectangle(mouseX,mouseY,bigRectWidth,bigRectHeight) )
    mr1.setColor(255,0,0) ;
  else
    mr1.setColor(0,0,0) ;
  if (mr2.intersectRectangle(mouseX,mouseY,bigRectWidth,bigRectHeight) )
    mr2.setColor(255,0,0) ;
  else
    mr2.setColor(0,0,0) ;
  if (mr3.intersectRectangle(mouseX,mouseY,bigRectWidth,bigRectHeight) )
    mr3.setColor(255,0,0) ;
  else
```

```
    mr3.setColor(0,0,0) ;
  if (mr4.intersectRectangle(mouseX,mouseY,bigRectWidth,bigRectHeight) )
    mr4.setColor(255,0,0) ;
  else
    mr4.setColor(0,0,0) ;
  if (mr5.intersectRectangle(mouseX,mouseY,bigRectWidth,bigRectHeight) )
    mr5.setColor(255,0,0) ;
  else
    mr5.setColor(0,0,0) ;

  mr1.drawRect() ;
  mr2.drawRect() ;
  mr3.drawRect() ;
  mr4.drawRect() ;
  mr5.drawRect() ;
}
```

The code is easier to understand, easier to write, and easier to modify. By way of comparison, here is a solution without using classes. Note the function definitions are not included, they were shown in the previous chapter:

```
float rX1, rY1, rW1, rH1, velX1, velY1 ;   // current x, y, width, height, velocity-X
and velocity-Y of rectangle1
float rX2, rY2, rW2, rH2, velX2, velY2 ;   // current x, y, width, height, velocity-X
and velocity-Y  of rectangle2
float rX3, rY3, rW3, rH3, velX3, velY3 ;   // current x, y, width, height, velocity-X
and velocity-Y  of rectangle3
float rX4, rY4, rW4, rH4, velX4, velY4 ;   // current x, y, width, height, velocity-X
and velocity-Y  of rectangle4
float rX5, rY5, rW5, rH5, velX5, velY5 ;   // current x, y, width, height, velocity-X
and velocity-Y  of rectangle5


float mouseRectW = 100 ;  // the width of the rectangle that follows the mouse
float mouseRectH = 100 ;  // the height of the same rectangle


void setup()
{
  size(400,400) ;
  fill(0) ;

  rX1 = 10 ;
```

```
  rY1 = 10 ;
  rW1 = 20 ;
  rH1 = 20 ;
  velX1 = 2 ;
  velY1 = 3 ;

  rX2 = 50 ;
  rY2 = 100 ;
  rW2 = 20 ;
  rH2 = 20 ;
  velX2 = 2 ;
  velY2 = 3 ;

  rX3 = 50 ;
  rY3 = 300 ;
  rW3 = 20 ;
  rH3 = 20 ;
  velX3 = 2 ;
  velY3 = 3 ;

  rX4 = 300 ;
  rY4 = 60 ;
  rW4 = 20 ;
  rH4 = 20 ;
  velX4 = 2 ;
  velY4 = 3 ;

  rX5 = 200 ;
  rY5 = 80 ;
  rW5 = 20 ;
  rH5 = 20 ;
  velX5 = 2 ;
  velY5 = 3 ;

}

void draw()
{
  background(100) ;

  // update all locations
  rX1 += velX1 ;
  rY1 += velY1 ;
  if ( (rX1 < 0) || ((rX1+rW1) > width) )
    velX1 *= -1 ;
  if ( (rY1 < 0) || ((rY1+rH1) > height) )
```

```
    velY1 *= -1 ;
  rX2 += velX2 ;
  rY2 += velY2 ;
  if ( (rX2 < 0) || ((rX2+rW2) > width) )
    velX2 *= -1 ;
  if ( (rY2 < 0) || ((rY2+rH2) > height) )
    velY2 *= -1 ;
  rX3 += velX3 ;
  rY3 += velY3 ;
  if ( (rX3 < 0) || ((rX3+rW3) > width) )
    velX3 *= -1 ;
  if ( (rY3 < 0) || ((rY3+rH3) > height) )
    velY3 *= -1 ;
  rX4 += velX4 ;
  rY4 += velY4 ;
  if ( (rX4 < 0) || ((rX4+rW4) > width) )
    velX4 *= -1 ;
  if ( (rY4 < 0) || ((rY4+rH4) > height) )
    velY4 *= -1 ;
  rX5 += velX5 ;
  rY5 += velY5 ;
  if ( (rX5 < 0) || ((rX5+rW5) > width) )
    velX5 *= -1 ;
  if ( (rY5 < 0) || ((rY5+rH5) > height) )
    velY5 *= -1 ;

  // draw mouse rectangle
  fill(0) ;
  rect(mouseX,mouseY,mouseRectW,mouseRectH) ;

  // draw the rectangles, first checking to see if touching mouseRect as if they are
make them red
  if (hit_TwoRectsIntersect(rX1,rY1,rW1,rH1, mouseX, mouseY, mouseRectW,
mouseRectH) )
    fill(255,0,0) ;
  else
    fill(0) ;
  rect(rX1,rY1,rW1,rH1) ;

  if (hit_TwoRectsIntersect(rX2,rY2,rW2,rH2, mouseX, mouseY, mouseRectW,
mouseRectH) )
    fill(255,0,0) ;
  else
    fill(0) ;
  rect(rX2,rY2,rW2,rH2) ;
```

```
  if (hit_TwoRectsIntersect(rX3,rY3,rW3,rH3, mouseX, mouseY, mouseRectW,
mouseRectH) )
    fill(255,0,0) ;
  else
    fill(0) ;
  rect(rX3,rY3,rW3,rH3) ;

  if (hit_TwoRectsIntersect(rX4,rY4,rW4,rH4, mouseX, mouseY, mouseRectW,
mouseRectH) )
    fill(255,0,0) ;
  else
    fill(0) ;
  rect(rX4,rY4,rW4,rH4) ;

  if (hit_TwoRectsIntersect(rX5,rY5,rW5,rH5, mouseX, mouseY, mouseRectW,
mouseRectH) )
    fill(255,0,0) ;
  else
    fill(0) ;
  rect(rX5,rY5,rW5,rH5) ;



}
```

Simply put, the above is YUCKY! (That is precise techno-talk). Message: classes/ objects can make programming a lot easier!!

We create classes to make it easier to solve some sort of programming problem. Here is a problem: We want to be able to create sprites. The sprites should move about the screen by themselves in two dimensions. We should be able to load an image from a file into the sprite object. When we click on the sprite it should go away. We should be able to make it come back somehow to.

We DESIGN a solution to this problem. There are many design solutions. Here is one possible solution:

```
class MovingSprite
{
  // member data variables
  float rx, ry, rwidth, rheight, velX, velY ;
  PImage theImage ;
  boolean imageHidden ;
```

```
 MovingSprite(float inRx, float inRy, float inVelx, float inVely, String inImageName)
 // constructor number 1
 {
   rx = inRx ;
   ry = inRy ;
   theImage = loadImage(inImageName) ;
   rwidth = theImage.width ;
   rheight = theImage.height  ;
   velX = inVelx ;
   velY = inVely ;
   imageHidden = false ;

 }

 // setter methods
 void setX(float inRx) { rx = inRx;}
 void setY(float inRy) { ry = inRy;}
 void setWidth(float inW) { rwidth = inW;}
 void setHeight(float inH) { rheight = inH;}
 void setVelX(float inVx) { velX = inVx;}
 void setVelY(float inVy) { velY = inVy;}

 // getter methods
 float getX() { return rx;}
 float getY() { return ry;}
 float getWidth() { return rwidth;}
 float getHeight() { return rheight;}
 float getVelX() { return velX;}
 float getVelY() { return velY;}

 void updateLocation()
 {
   rx += velX ;
   ry += velY ;
   if ( (rx < 0) || ((rx+rwidth) > width) )
     velX *= -1 ;
   if ( (ry < 0) || ((ry+rheight) > height) )
     velY *= -1 ;
 }

 void drawSprite()
 {
   if (!imageHidden)
     image(theImage,rx,ry) ;
 }
```

```
 boolean containsPoint(float inPx, float inPy)
 {
   if ( (inPx > rx) && (inPx < (rx + rwidth) ) &&
      (inPy > ry) && (inPy < (ry+rheight) )  )
     return (true) ;
   else
     return(false) ;
 }

 boolean isHidden()
 // returns true if the image is currently hidden
 {
   if (imageHidden)
     return (true) ;
   else
     return (false) ;
 }

 void hideImage()
 {
   imageHidden = true ;
 }

 void unhideImage()
 {
   imageHidden = false ;
 }


}
```

Notice this classes member variables include “theImage” and “imageHidden” of type PImage and boolean respectively. Notice how these variables are initialized in the constructor. Also notice the methods containsPoint(), drawSprite(), isHidden(), hideImage() and unhideImage().

The following code shows examples of using this class. In this example we put four moving sprites on the screen moving in different directions. When the user clicks on one it disappears. When all four are gone they all come back but at a faster speed. Note the global variable “numNotHidden” is used to keep track of how many objects are still visible.

```
// global vars
```

```
MovingSprite ms1, ms2, ms3, ms4, ms5 ;
int numNotHidden ;


void setup()
{
  size(400,400) ;
  numNotHidden = 4 ;
  ms1 = new MovingSprite(width/2,height/2,random(1,2),random(2,3),"smiley1.png") ;
  ms2 = new MovingSprite(width/2,height/2,random(-2,-1),random(2,3),"smiley1.png") ;
  ms3 = new MovingSprite(width/2,height/2,random(1,2),random(-3,-2),"smiley1.png") ;
  ms4 = new MovingSprite(width/2,height/2,random(-2,-1),random(-2,-1),"smiley1.png") ;


}

void draw()
{
  background(0) ;

  ms1.updateLocation() ;
  ms1.drawSprite() ;
  ms2.updateLocation() ;
  ms2.drawSprite() ;
  ms3.updateLocation() ;
  ms3.drawSprite() ;
  ms4.updateLocation() ;
  ms4.drawSprite() ;
}

void mousePressed()
{
  if ( (ms1.containsPoint(mouseX,mouseY) ) && !ms1.isHidden() )
  {
    ms1.hideImage() ;
    numNotHidden-- ;
    println(numNotHidden) ;
  }
  if ( (ms2.containsPoint(mouseX,mouseY) ) && !ms2.isHidden() )
  {
    ms2.hideImage() ;
    numNotHidden-- ;
    println(numNotHidden) ;
  }
  if ( (ms3.containsPoint(mouseX,mouseY) ) && !ms3.isHidden() )
  {
```

```
    ms3.hideImage() ;
    numNotHidden-- ;
    println(numNotHidden) ;
  }
  if ( (ms4.containsPoint(mouseX,mouseY) ) && !ms4.isHidden() )
  {
    ms4.hideImage() ;
    numNotHidden-- ;
    println(numNotHidden) ;
  }

  if (numNotHidden == 0)
  {
    println("complete a round") ;
    resetTheSmileys() ;
    println(numNotHidden) ;
  }
}

void resetTheSmileys()
{
  ms1.setVelX( ms1.getVelX() * 1.5) ;
  ms1.setVelY( ms1.getVelX() * 1.5) ;
  ms1.unhideImage() ;
  ms2.setVelX( ms2.getVelX() * 1.5) ;
  ms1.setVelY( ms1.getVelX() * 1.5) ;
  ms2.unhideImage() ;
  ms3.setVelX( ms3.getVelX() * 1.5) ;
  ms3.setVelY( ms1.getVelX() * 1.5) ;
  ms3.unhideImage() ;
  ms4.setVelX( ms4.getVelX() * 1.5) ;
  ms4.setVelY( ms1.getVelX() * 1.5) ;
  ms4.unhideImage() ;

  numNotHidden = 4 ;
}
```

Note, the MovingRectangle and MovingSprite classes we have created are just two examples of a class. Again, the analogy we are using is that it is like tool-building: by using the class construct we have made better tools for programming: easier to use, easier to understand, less likely to introduce mistakes. Further, we can improve upon our tool by adding new methods or changing them.

We can create our own classes for anything we want. And, a well constructed class can be given to other programmers to use. They can use it without understanding its implementation (i.e. the code inside the class definition), all they need to know is the interface, i.e. the constructor, getters, setters, and how to call the additional methods. Just as most of use do not really understand how an automobile works inside, we still can use it. It would be way too much work to understand and tinker around with what is under the hood.

EXERCISE 13A
Modify the MovingSprite class to have a new method called MoveTowards(x,y). When you click on the screen the sprite will move to the point you clicked on. Hint: you will need to change velX and velY so that the sprite moves in the direction of the point (x,y). Suggestion: have the sprite stop once within a certain distance of the point. This will stop the sprite for shimmying back and forth at the point.

EXERCISE 13B
Create a new class called AnimatedSprite. When a sprite object is created it should play the animation sequence over and over. Assume the animation has exactly 4 frames (images). You should add startAnimation() and stopAnimation() methods so that you can stop and start the animation. The constructor should have whatever input parameters are needed/desired plus four parameters for the names of the animation frames.