SLUG Fest 2013
Introduction to Classes
John Hanna jehanna@optonline.net www.johnhanna.us

Classes

Classes provide a means of encapsulating *objects* with data and member functions similar to Java programming. Working with classes requires a 'different' way of thinking about programming: nouns rather than verbs. Rather than 'move the circle' we think – 'tell the circle to move'.

Here's a complete (but not very interesting) Lua example:

```
circle = class() -- the name of the class is circle

function circle:init(cx, cy, radius) -- called when a circle
                                     -- object is created
  self.x = cx
  self.y = cy
  self.r = radius
end

function circle:paint(gc)  -- the routine to draw the circle
  gc:drawArc(self.x, self.y, self.r, self.r, 0, 360)
end

c = circle(50, 50, 50) -- c is a circle object

function on.paint(gc)
  c:paint(gc)
end
```

Some big ideas:

From the Java Tutorial: A *class* is the blueprint from which individual objects are created.

From Wikipedia: In object-oriented programming, a **class** is a construct that is used to create instances of itself – referred to as *class instances*, *class objects*, *instance objects* or simply *objects*. A class defines constituent members which enable its instances to have state and behavior. Data field members (*member variables* or *instance variables*) enable a class instance to maintain state. Other kinds of members, especially *methods*, enable the behavior of class instances. Classes define the type of their instances. A class usually represents a noun, such as a person, place or thing. For example, a "Banana" class would represent the properties and functionality of bananas in general. A single, particular banana called "Hanna" would be an instance of the "Banana" class, an object of the type "Banana".

The class is only an abstract idea or plan. It is *not* a variable. The class definition can contain variables and 'member' functions.

The main difference between TI-BASIC programming and Lua programming is the way the user interaction is turned inside out. BASIC programs are in control: they read input from users and display results whenever their logic dictates. Lua scripts, on the other hand, are reactive. They may only accept input in response to events such as key presses. And they only display results indirectly in response to the system requesting a repaint. It takes a new way of thinking to write programs in this fashion.

-John Powers 1/14/13

SLUG Fest 2013
Introduction to Classes
John Hanna jehanna@optonline.net www.johnhanna.us

A powerful idea: 'dynamic variables'

Objects can be created 'on the fly' while the program is running and can 'respond' to events like mouse clicks and mouse movements…

```
-- the circle class definition --
circle=class()

function circle:init(x,y,r)
   self.x=x
   self.y=y
   self.r=r
end

function circle:paint(gc)
   gc:drawArc(self.x-self.r,self.y-self.r,2*self.r,2*self.r,0,360)
end
-- end of the circle class --

-- make a circle object --
c = circle(50,50,20)

-- make a list of objects --
list = {c}

function on.paint(gc)
    for i=1,#list do
       obj=list[i]
       obj:paint(gc)
    end
end

-- when you click the mouse, make another circle --
function on.mouseDown(x,y)
    c = circle(x,y,20)
    table.insert(list,c)
    platform.window:invalidate()
end
```

John Hanna jehanna@optonline.net www.johnhanna.us

Click 'n Drag

The final demo illustrates a technique for clicking and dragging an object.

```
-- the circle class definition --
circle=class()
selectedObject = nil

function circle:init(x,y,r)
    self.x=x
    self.y=y
    self.r=r
end

function circle:contains(x,y)
   if (x-self.x)^2 + (y-self.y)^2 < self.r^2 then
      selectedObject = self
   end
end

function circle:paint(gc)
    gc:drawArc(self.x-self.r,self.y-self.r,2*self.r,2*self.r,0,360)
end
-- end of the circle class --

-- make a circle object --
c = circle(150,50,20)
c2 = circle(75,80,10)
c3 = circle(180,100,15)
-- make a list of objects --
list={c, c2, c3}

function on.paint(gc)
    for i=1,#list do
       obj = list[i]
       obj:paint(gc)
    end
end

function on.mouseDown(x,y)
  for i=1,#list do
     obj=list[i]
     if obj:contains(x,y) then
      selectedObject = obj
     end
  end
end

function on.mouseMove(x,y)
  if selectedObject ~= nil then
     selectedObject.x = x
     selectedObject.y = y
  end
end

function on.mouseUp(x,y)
  selectedObject = nil
end
```