Computer Science 51
Spring 2006

Midterm Examination
8 March, 2006

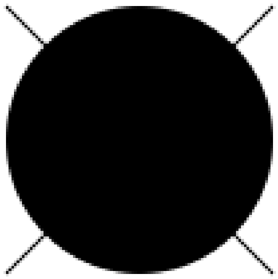
Question	Points	Score	Description
1	15		Class definition
2	10		Program analysis
3	15		Programming - Swing
4	13		Short Answers
Total	53		

This examination is closed book. You have 50 minutes to complete the exam.

Your Name (Please print) ______________________________

1. **Classes**:

   I would like you to design a class representing a draggable "Sun-like" object that should look like this (except for the color):



You can think of this "sun" as having four rays emanating out (though you can draw them with just two lines, if you think about it a bit). The class should take parameters to allow the sun to be created at any location and to be of any size(though of course it should be round!). It should always be yellow when it is created. Your sun should be "draggable," so it will need a `moveBy` method and a `contains` method (but no **`moveTo`** method).

To make things slightly more interesting, the sun's rays should disappear and the sun should always change color to orange when it is being moved. The method `shineOn`, should make the rays reappear and restores the sun to its usual yellow color.

You are to fill in the class below to implement this sun class. Fill in any needed constants, instance variables, local variables, formal parameters, and method bodies. You need not include comments.

```
dialect "objectdraw"
type Draggable = {
   moveBy(dx: Number, dy: Number) -> Done
   contains(pt: Point) -> Boolean
   shineOn -> Done
}

class sunAt(pt: Point) size (diameter: Number)
            on (canvas: DrawingCanvas) -> Draggable {
  def body: Graphic2D = filledOvalAt(pt) size (diameter @ diameter)
                on (canvas)
  body.color := colorGen.yellow
  def ray1: Line = lineFrom(pt) to (pt+(diameter @ diameter))
                on (canvas)
  ray1.color := colorGen.yellow

  def upperRight: Point = pt + (diameter @ 0)
  def lowerLeft: Point = pt + (0 @ diameter)
  def ray2: Line = lineFrom (upperRight) to (lowerLeft) on (canvas)
  def ray2.color := colorGen.yellow

  def orange: Color = colorGen.r(250) g (115) b (10)
  // move the sun, hide the rays, and change color to orange
  method moveBy(dx: Number, dy: Number) -> Done {
    body.moveBy(dx,dy)
    ray1.moveBy(dx,dy)
    ray2.moveBy(dx,dy)
    ray1.visible := false
    ray2.visible := false
    body.color := orange
  }

  // determine whether sun (including rays) contains the point
  method contains(pt: Point) -> Boolean {
    body.contains(pt) || ray1.contains(pt) || ray2.contains(pt)
  }

  // show the rays and change color to yellow
  method shineOn -> Done {
    body.color := colorGen.yellow
    ray1.visible := true
    ray2.visible := true
  }

}
```

2. Explain what the program on the following 2 pages does:
   i. When it starts

Prints the instructions on the screen

   ii. When the user presses and continues holding the mouse

A bubble (circle appears on the screen where the user pressed and grows slowly – staying centered at the same place – until the mouse is released.

   iii. When the user releases the mouse

The bubble floats slowly off the top of the screen. Tragically, it is never removed from the canvas!

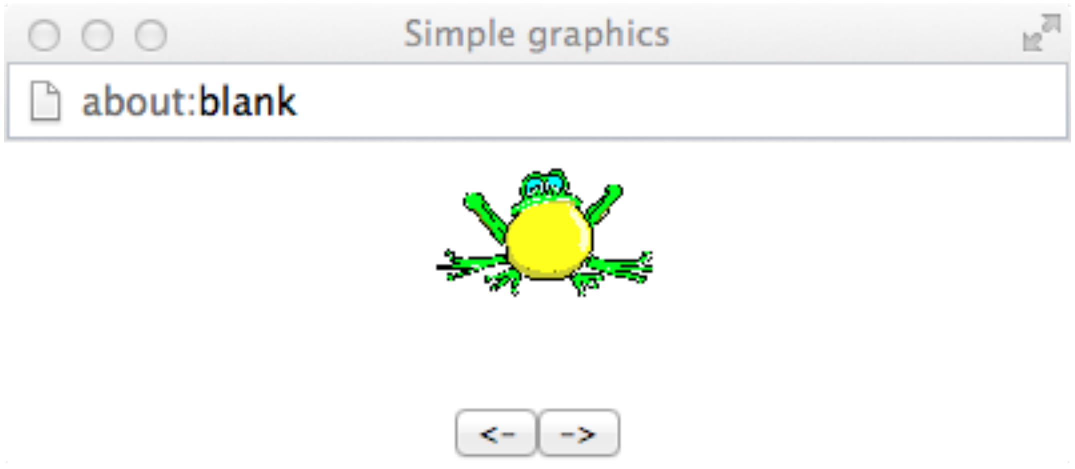
```
def mystery: GraphicApplication = object {
  inherit graphicApplicationSize(600 @ 600)

  def textPlace: Point = 150 @ 450
  var thing: Mystery
  textAt(textPlace)
       with ("hold the mouse down for a while")
       on (canvas)

  method onMousePress(point: Point) -> Done {
    thing := mysteryObjectAt (point) on (canvas)
    thing.start
  }

  method onMouseRelease(point: Point) -> Done {
    thing.release
  }

  startGraphics
}
```

```
type Mystery = {
  start -> Done
  release -> Done
}

class mysteryObjectAt(point: Point)
            on (canvas:DrawingCanvas) -> Mystery {
  def startDiameter: Number = 4
  def increase: Number = 2
  def pauseTime: Number = 50
  var growing: Boolean := true
  def startLocn: Point =
       point - ((startDiameter/2) @ (startDiameter/2))

  def circle: Graphic2D =
        framedOvalAt(startLocn)
          size (startDiameter @ startDiameter) on (canvas)

  method release -> Done {
    growing := false
  }

  method start -> Done {
    animator.while {growing} pausing (pauseTime) do {
      circle.width := circle.width + increase
      circle.height := circle.height + increase
      circle.moveBy(-increase/2,-increase/2);
    } finally {
      animator.while{(circle.y + circle.height) >= 0}
                     pausing (pauseTime) do {
         circle.moveBy(0,-increase);
      }
    }
  }
}
```

3. A very enthusiastic CS 51G student named Herb became so excited when he learned about GUI components that he wanted to go back and revise the programs he wrote for some of the earlier labs to use them. For example, Herb wanted to change the Frogger program so that the player would have to click on appropriate buttons to move the frog rather than just clicking on the canvas. Unfortunately, despite his enthusiasm, Herb doesn't really understand how to write programs using GUI components. So, we would like you to help him get started on the task of revising his Frogger by writing a program to illustrate how a set of buttons could control the hopping of the frog.

We don't want you to write all the code Herb will need. After all, this should be a learning experience for Herb. Instead, we just want you to write a simple program that will display the frog and two buttons on the screen. Clicking one of the buttons should make the frog "hop" to the left. Clicking the other button should make the frog hop to the right.

We have included a picture of what the display should look like while the program is running below. In particular, in this picture we show how we would like the buttons to appear in the display.



A "starter" for this program can be found on the next two pages. Please write all the additional code needed to implement the program described on this starter. Your program should NOT react to mouse events on the canvas, only to mouse presses on the buttons!

```
dialect "objectdraw"

def toTheHop = object {
  inherit graphicApplicationSize(400 @ 100)
  def frogStart: Point = 160 @ 10
  def frogWidth: Number = 83
  def frogHeight: Number = 48

  // distance frog should hop
  def hopSize: Number = 70

  def frog: Graphic2D =
     drawableImageAt(frogStart)
        size (frogWidth @ frogHeight)
        url("http://www.cs.pomona.edu/classes/….png")
        on (canvas)

  def goLeft: Button = buttonLabeled("<-")
  def goRight: Button = buttonLabeled("->")

  append(goLeft)
  append(goRight)

  goLeft.onMousePressDo{mouseEvent: MouseEvent ->
    frog.moveBy(-hopSize,0)
  }

  goRight.onMousePressDo{mouseEvent: MouseEvent ->
    frog.moveBy(hopSize,0)
  }

  startGraphics
}
```

4. Short questions:

a. Please rewrite the following method so that it does the same thing but uses better style (and is more compact):

```
method sameNums(x: Number, y: Number) -> Boolean {
  if (x == y) then {
    true
  } else {
    false
  }
}
```

Should be

```
method sameNums(x: Number, y: Number) -> Boolean {
  x == y
}
```

b. Simplify the following code in which `continuing` is a boolean variable:

```
if (continuing) then {

} else {
  print "Still working"
}
```

Should be

```
if (!continuing) then {
  print "Still working"
}
```

c. A beginning programmer has written the program below without fully understanding declarations and scope. What problems does the following code have?

```
class cWith(number: Number) {
    var number: Number := number

    method m -> Done{
          var number:Number
          if ( 18 > number) then{ ... }
    }
    ...
}
```

Lots of mistakes here: number is actually declared three times! Once as a parameter in c.with, once as an instance variable on the next line, and then a third time in method m. Presumably they didn't need either of the "var" declarations as they could just have used the parameter number. As it is, the fact that the last declaration of number is not initialized means that the program will crash when method m is executed.

d. Fill in the condition following the if for the following code. Assume myNumber has been declared as a Number.

```
if ( // myNumber is evenly divisible by 3 or 4 )
    then {
      // do something
}
```

The answer is

```
if ((myNumber % 3) == 0) || ((myNumber % 4) == 0))
    then {
      // do something
}
```