This exam is closed book, closed notes, closed computer, closed calculator, closed friends, etc. The only materials you may use are the GUI and Objectdraw cheat sheets that we will be providing at the start of the exam. Read the problem descriptions carefully and write your answers clearly and *legibly* in the space provided. You may use extra sheets of paper if you need more room, as long as the problem number is clearly labelled and your name is on the paper.

There are 4 questions (plus problem 0). You have 50 minutes for the exam, which is worth 100 points. If you have any questions, just raise your hand.

Problem 0: (1 point)

- Write your name at the top of this page.

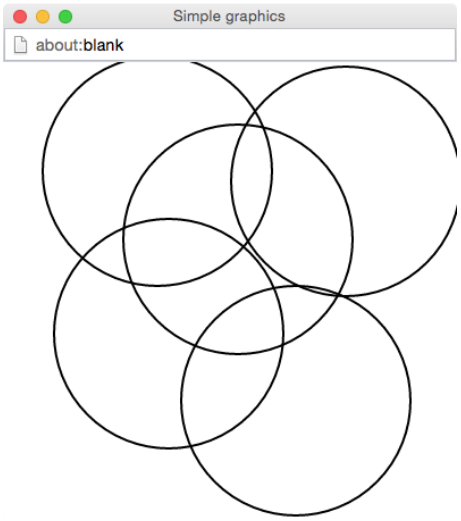
For grading:

Problem 0	1	
Problem 1	30	
Problem 2	24	
Problem 3	30	
Problem 4	15	
Total	100	

Problem 1: (30 points)

The program `rippleGenerator` is an object that inherits `graphicApplication.size (400, 400)` When the program starts, there is nothing shown in the window.

When the user clicks on the screen, a small circle (diameter 2) appears at that location on the screen. The circle then expands, staying centered at the same location, until its width or height (they are the same in this case) exceeds 200, at which point it disappears. The following is a screenshot showing the picture after new has been clicked several times:



Your program should use a class (`ripple`) that contains the animated object and has type `Animated` where

```
type Animated = {
    start -> Done
}
```

On the next page complete the code for `rippleGenerator` and `ripple`. You may assume that the window is 400 by 400. You do not need to define constants (just use magic numbers), nor do you need to include comments.

```
dialect "rtobjectdraw"
import "animation" as animator
def rippleGenerator: GraphicApplication = object {
    inherits graphicApplication.size (400, 400)









}
```

```
// add names and parameters as needed
class bubble.                                                    )
                                                              -> Animated {

    method start -> Done {

    }
}
```

Problem 2: (24 points)

1. The following are three methods designed to adjust an instance variable `score` according to these rules: a `score` below 50 should be increased by 30% and a `score` greater than or equal to 50 should be increased by only 10%. Next to each method indicate whether it works correctly. If it does not, indicate how the method malfunctions.

(i)

```
method adjust -> Done {
   if (score < 50.0) then {
        score := score * 1.3
   } elseif (score >= 50) then {
        score := score * 1.1
   }
}
```

(ii)

```
method adjust -> Done {
   if (score < 50.0) then {
        score := score * 1.3
   } else {
        score: = score * 1.1
   }
}
```

(iii)

```
method adjust -> Done {
   if (score < 50.0) then {
        score := score * 1.3
   }
   if (score >= 50) {
        score := score * 1.1
   }
}
```

2. Rewrite the following two methods so that they are more compact, but still do the same thing:

(i)

```
method aMethod (b: Boolean) -> Done{
   if (b && true) then {
      print ("One")
   }

   if (!b) then {
      print ("Two")
   }

   if ( b == true ) then {
      print ("Three")
   }
}
```

(ii)

```
method anotherMethod -> Boolean {
   if (check == false) then {
      true
   } else {
      false
   }
}
```

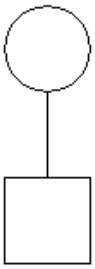
3. What is printed by the following method?

```
method begin -> Done {
    var x: Number := 10
    var y: Number := 0
    while {y < x} do {
        print "{y}, {x}"
        y := y+1
        x := x-1
    }
}
```

Problem 3: (30 points)

JackInTheBox is a class the extends WindowController and implements a basic jack-in-the-box game. When the program begins the canvas shows a box like so:

If the user clicks in the box three times, the jack "pops out" by showing a body and head like so:



If the user clicks in the box a fourth time, the jack is "put back into the box" by hiding the body and head, like it was in the beginning, and the game can be repeated. The full implementation of the `jackInTheBox` program follows. Your job is to fill in the jack class on the following page by filling in the methods used by JackInTheBox.

```
dialect "rtobjectdraw"
def jackInTheBox: GraphicApplication = object {
    inherits graphicApplication.size (400, 400)

    def theJack: Jack = jack.at (100 @ 300) on (canvas)

    method onMousePress (pt: Point) -> Done {
        if (theJack.boxContains(pt)) then {
            if (theJack.isHiding) then {
                theJack.turnCrank
                if (theJack.readyToPop) then {
                    theJack.pop
                }
            } else {
                theJack.hideJack
            }
        }
    }

    startGraphics
}
```

```
type Jack = {
    isHiding -> Boolean
    hideJack -> Done
    pop -> Done
    boxContains (pt: Point) -> Boolean
    turnCrank -> Done
    readyToPop -> Boolean
}

class jack.at(pt: Point) on (canvas: DrawingCanvas) -> Jack {









    // return true if the jack is in the box, false if out
    method isHiding -> Boolean {



    }

    // hide the jack and reset the jack crank
    method hideJack -> Done {









    }
```

```
    // show the jack!
    method pop -> Done {




    }

    // return true if the jack's box contains the point, false otherwise
    method boxContains (point: Point) -> Boolean {



    }

    // update the jack because the 'crank has been turned'
    method turnCrank -> Done {




    }

    // return true if the jack is ready to pop (the crank has been turned
    // three times)
    method readyToPop -> Boolean {




    }
}
```

Problem 4: (15 points) The mystery object draws a picture on the canvas. Please read the mystery program carefully and on the following page draw what this program produces.

```
dialect "rtobjectdraw"

def mystery: GraphicApplication = object {
    inherits graphicApplication.size (250 , 250)
    var rowCount: Number := 1
    var y: Number := 0
    var startWhite: Boolean := false

    while {rowCount <= 5} do {
        var columnCount: Number := 1
        var x: Number := 0
        if (startWhite) then {
            columnCount := 2
            x := 50
        }
        while {columnCount <= 3} do {
            filledRect.at (x @ y) size (50, 50) on (canvas)
            columnCount := columnCount + 1
            x := x + 100
        }
        y := y + 50
        rowCount := rowCount + 1
        startWhite := !startWhite
    }

    startGraphics
}
```

Please draw the mystery program's output here: