Handout for Multiple Language Implementations for a Simple Geometry Class

Java

```
class Shape
{
	public void talkToMe()
	{
		System.out.println("But I am a Shape first and foremost");
	}
}

class Rectangle extends Shape
{
	String recName;
	float width, height, area, perimeter;

	public Rectangle(String name, float width, float height)
	{
		initialize(name,width,height);
		System.out.println("A Rectangle has been created and initialized");
		displayYourself();
	}

	private void initialize(String name, float width, float height)
	{
		recName = name;
		this.width = width;
		this.height = height;
		area = width * height;
		perimeter = 2 * (width + height);
	}

	public void displayYourself()
	{
		System.out.println(" I am a Rectangle: "+recName);
		System.out.print("      width: "+width);
		System.out.print("      height: "+height);
		System.out.print("      area: "+area);
		System.out.println("    perimeter: "+perimeter);
	}
}

class Square extends Rectangle
{
	public Square(String name, float width, float height)
	{
		super(name,width,width);
		System.out.println("A Square has been created and initialized");
	}
}

public class geo
{
	public static void main(String args[])
	{
		System.out.println("Creating a new Rectangle");
		Rectangle r1 = new Rectangle("Rectangle r1",5,4);
		System.out.println("Creating a new Square");
		Square s1 = new Square("Square s1",6,6);
		System.out.println("Calling displayYourself on the new Square");
		s1.displayYourself();
		System.out.println("Calling talkToMe on the new Square");
		s1.talkToMe();
	}
}
```

C++:

```
#include <iostream.h>

class Shape
{
      public:
            void talkToMe();
};

void Shape::talkToMe()
{
      cout << "But I am a Shape first and foremost" << endl;
}

class Rectangle : public Shape
{
      private:
            char * recName;
            float width, height, area, perimeter;

      protected:
            void initialize(char * name, float width, float height);

      public:
            Rectangle(char * name, float width, float height);
            void displayYourself();
};

Rectangle::Rectangle(char * name, float width, float height)
{
      initialize(name,width,height);
      cout << "A Rectangle has been created and initialized" << endl;
      displayYourself();
}

void Rectangle::initialize(char * name, float width, float height)
{
      recName = name;
      this->width = width;
      this->height = height;
      area = width * height;
      perimeter = 2 * (width + height);
}

void Rectangle::displayYourself()
{
      cout << " I am a Rectangle: " << recName << endl;
      cout << "   width: " << width;
      cout << "     height: " << height;
      cout << "     area: " << area;
      cout << "     perimeter: " << perimeter << endl;
}

class Square : public Rectangle
{
      public:
            Square(char * name, float width, float height);
};

Square::Square(char * name, float width, float height) : Rectangle(name, width, height)
{
      cout << "A Square has been created and initialized" << endl;
}
```

```
void main(int argc, char * argv[] )
{
      cout << "Creating a new Rectangle" << endl;
      Rectangle * r1 = new Rectangle("Rectangle r1",5.0,4.0);
      cout << "Creating a new Square" << endl;
      Square * s1 = new Square("Square s1",6.0,6.0);
      cout << "Calling displayYourself on the new Square" << endl;
      s1->displayYourself();
      cout << "Calling talkToMe on the new Square" << endl;
      s1->talkToMe();
}
```

Common Lisp:

```
(in-package :user)

(defclass Shape () ())

(defclass Rectangle (Shape)
  ((recName :initarg :name :initform nil :accessor recName)
   (width :initarg :width :initform nil :accessor width)
   (height :initarg :height :initform nil :accessor height)
   (area :initform nil :accessor area)
   (perimeter :initform nil :accessor perimeter)
   ))

(defclass Square (Rectangle) ()
)

;;; define the INTERFACE for generic functions
(defgeneric initialize (rectangle)
  (:documentation
   "this is the initialize method for all types of Rectangles"))

(defgeneric displayYourself (rectangle)
  (:documentation
   "this is the display method for all types of Rectangles"))

(defgeneric talkToMe (shape)
  (:documentation
   "this is the talkToMe method for alltypes of Shapes"))

;;; define the IMPLEMENTATION METHODS of the Shape hierarchy
(defmethod initialize ((rectangle Rectangle))
  (setf (area rectangle) (* (width rectangle) (height rectangle)))
  (setf (perimeter rectangle) (* 2 (+ (width rectangle) (height rectangle))))
  )

(defmethod talkToMe ((shape Shape))
  (format t "But I am a Shape first and foremost"))

(defmethod displayYourself ((rectangle Rectangle))
  (format t " I am a Rectangle: ~A~%" (recName rectangle))
  (format t "    width: ~F" (width rectangle))
  (format t "    height: ~F" (height rectangle))
  (format t "    area: ~F" (area rectangle))
  (format t "    perimeter: ~F~%" (perimeter rectangle))
)

;;;define constructors for the derivative classes
(defun new-Rectangle (name width height)
  (format t "A Rectangle has been created and initialized~%")
  (let ((r (make-instance 'Rectangle :name name :width width :height height)))
    (initialize r)
    r)
)

(defun new-Square (name width height)
  (format t "A Square has been created and initialized~%")
  (let ((s (make-instance 'Square :name name :width width :height height)))
    (initialize s)
    s)
)

(defun doMain ()
  (format t "Creating a new Rectangle~%")
```

```
    (let ((myrec (new-Rectangle "Rectangle r1" 5.0 4.0)))
      (displayYourself myrec))
    (format t "Creating a new Square~%")
    (let ((mysq (new-Square "Square s1" 6.0 6.0)))
      (format t "Calling dsiplayYourself on the new Square~%")
      (displayYourself mysq)
      (format t "Calling talkToMe on the new Square~%")
      (talkToMe mysq))
)
```

Simula:

```
begin

      class Shape;
      begin
            procedure talkToMe;
            begin
                  OutText("But I am a Shape first and foremost");
                  OutImage
            end of getDefault;
      end;

      Shape class Rectangle(recName,width,height);! class with 3 parameters;
      Text recName;Real width, height;! Specification of parameters;
      begin
             Real area, perimeter; ! these are our attributes;

             procedure initialize;  ! method definition;
             begin
                  area := width * height;
                  perimeter := 2*(width + height)
             end of initialize;

             procedure displayYourself;   ! a Method;
             begin
                   OutText(" I am a rectangle: "); OutText(recName);
                   OutImage;
                   OutText("        width: "); OutFix(width,2,4);
                   OutText("    height: "); OutFix(height,2,4);
                   OutText("    area: "); OutFix(area,2,6);
                   OutText("    perimeter: "); OutFix(perimeter,2,6);
                   OutImage
             end of displayYourself;

            initialize;       ! body of Rectangle;
            OutText("A Rectangle has been created and initialized."); OutImage;
            displayYourself;
      end of Rectangle;

      Rectangle class Square;
      begin
             height := width;!body of Square;
             initialize;
             OutText("A Square has been created and initialized"); OutImage;
      end of Square;


      !Variables declared in the prefixed block: ;

      ref(Rectangle) R1;
      ref(Square) S1;

      !Block body - here the program starts: ;

      OutText("Creating a new Rectangle"); OutImage;
      r1 :- New Rectangle("Rectangle r1",5,4); OutText("Creating a new Square"); OutImage;
      s1 :- New Square("Square s1",6,6);
      OutText("Calling displayYourself on the new Square"); OutImage;
      s1.displayYourself;
      OutText("Calling talkToMe on the new Square"); OutImage;
      s1.talkToMe;
      OutImage;
end;
```

Ruby :

```
class Shape
     def talkToMe
          puts "But I am a Shape first and foremost"
     end
end

class Rectangle < Shape
     def initialize( name, width, height )
          puts "A Rectangle has been created and initialized"
          @recName = name
          @width = width
          @height = height
          @area = @width * @height
          @perimeter = 2 * (@width + @height)
          displayYourself
     end
     def displayYourself
          puts " I am a Rectacngle: " + @recName
          print "      width: " + @width.to_s
          print "      height: " + @height.to_s
          print "      area: " + @area.to_s
          puts "      perimeter: " + @perimeter.to_s
     end
end

class Square < Rectangle
     def initialize ( name, width, height )
          super(name,width,height)
          puts "A Square has been created and initialized"
     end
end

puts "Creating a new Rectangle"
r1 = Rectangle.new("Rectangle r1",5,4)
puts "Creating a new Square"
s1 = Square.new("Square s1",6,6)
puts "Calling displayYourslf on the new Square"
s1.displayYourself
puts "Calling talkToMe on the new Square"
s1.talkToMe
```