Ethan Roberts

CS 417

Design Patterns – Square

Roberts.E.assn06\_squares

**Problem 1. (Source Code):**

# Ethan Roberts

# CS 417 Topics in OOP

# This program will read a command and -

# display what action will be completed.

# Roberts.E.assn06

# Design Patterns

# This class contains associative stacks

**class** **Square**

attr\_reader :squareNumber

attr\_reader :length

attr\_reader :centerPoint

**def** **initialize**

@squareNumber **=** 0

@length **=** 0

@squareNumber **=** **[]**

#for pushing contents

#these three stacks are associative

@centerPointStack **=** Array**.**new**()**

@lengthStack **=** Array**.**new**()**

@cmdStack **=** Array**.**new**()**

#used as a temporary stack in case of undo/redo

# These stacks are associative

@otherStackCP **=** Array**.**new**()**

@otherStackLength **=** Array**.**new**()**

@otherStackCmd **=** Array**.**new**()**

**end**

**def** **createSquare(**i**,**j**)**

@centerPoint **=** **[**0**,**0**]**

@squareNumber **=** i

@length **=** j

@centerPointStack**.**push**(**@centerPoint**)**

@lengthStack**.**push**(**@length**)**

@cmdStack**.**push**(**"C"**)**

**return** **self**

**end**

**def** **moveSquare(**j**,**k**)**

@centerPoint **=** **[**j**,**k**]**

@centerPointStack**.**push**(**@centerPoint**)**

@lengthStack**.**push**(**@length**)**

@cmdStack**.**push**(**"M"**)**

**end**

# scaling square and increasing/decreasing length by "j"

**def** **scaleSquare(**i**,**j**)**

j **=** j**.**to\_i

@length **=** @length **+** j

@centerPointStack**.**push**(**@centerPoint**)**

@lengthStack**.**push**(**@length**)**

@cmdStack**.**push**(**"S"**)**

**end**

# Saving main stack contents into "otherStack" (temporary stack)

**def** **undo**

#if nothing is on the "main" stack

**if** **(**@lengthStack**.**empty?**)**

print "Nothing left to undo...\n\n"

**else**

@otherStackCP**.**push**(**@centerPointStack**.**pop**())**

@otherStackLength**.**push**(**@lengthStack**.**pop**())**

@otherStackCmd**.**push**(**@cmdStack**.**pop**())**

**end**

**if** **(!**@lengthStack**.**empty?**)** #if the stack is not empty, get element

@centerPoint **=** @centerPointStack**.**last**()**

@length **=** @lengthStack**.**last**()**

**end**

**end**

# Pulling "otherStack" (temporary stack) contents and pushing

# onto main stack

**def** **redo**

#if nothing is on the "redo" stack

**if** **(**@otherStackCP**.**empty?**)**

print "Nothing left to redo...\n\n"

**else**

@centerPointStack**.**push**(**@otherStackCP**.**pop**())**

@lengthStack**.**push**(**@otherStackLength**.**pop**())**

@cmdStack**.**push**(**@otherStackCmd**.**pop**())**

@centerPoint **=** @centerPointStack**.**last**()**

@length **=** @lengthStack**.**last**()**

**end**

**end**

**def** **printSquareContents**

print "Square Number: "

print **self.**squareNumber

print " "

print "Center Point: "

print **self.**centerPoint

print " "

print "Length: "

print **self.**length

print "\n"

**end**

**end**

myAry **=** **[]** #this array will contain Square objects

line **=** ""

userCmd **=** ""

i **=** 0

j **=** 0

k **=** 0

squareCounter **=** 0

**while** **(**userCmd **!=** "X" **&&** userCmd **!=** "x"**)**

print "Enter a command (Commands are: C, M, S, U, R, P, X):\n"

line **=** gets**.**upcase

line **=** line**.**split**(**' '**)**

userCmd **=** line**[**0**]**

**if** **(**userCmd **==** 'C'**)**

i **=** line**[**1**].**to\_i

j **=** line**[**2**].**to\_i

square **=** Square**.**new**()**

myAry**[**i**]** **=** square**.**createSquare**(**i**,**j**)**

squareCounter **=** squareCounter **+** 1

myAry**[**i**].**printSquareContents

**end**

**if** **(**userCmd **==** 'M'**)**

i **=** line**[**1**].**to\_i

j **=** line**[**2**].**to\_i

k **=** line**[**3**].**to\_i

myAry**[**i**].**moveSquare**(**j**,**k**)**

myAry**[**i**].**printSquareContents

**end**

**if** **(**userCmd **==** 'S'**)**

i **=** line**[**1**].**to\_i

j **=** line**[**2**].**to\_i

myAry**[**i**].**scaleSquare**(**i**,**j**)** #increases length of square

myAry**[**i**].**printSquareContents

**end**

**if** **(**userCmd **==** 'U'**)**

myAry**[**i**].**undo

myAry**[**i**].**printSquareContents

**end**

**if** **(**userCmd **==** 'R'**)**

myAry**[**i**].**redo

myAry**[**i**].**printSquareContents

**end**

**if** **(**userCmd **==** 'P'**)**

z **=** 1

**while** **(**z **<=** squareCounter**)**

print myAry**[**z**].**printSquareContents

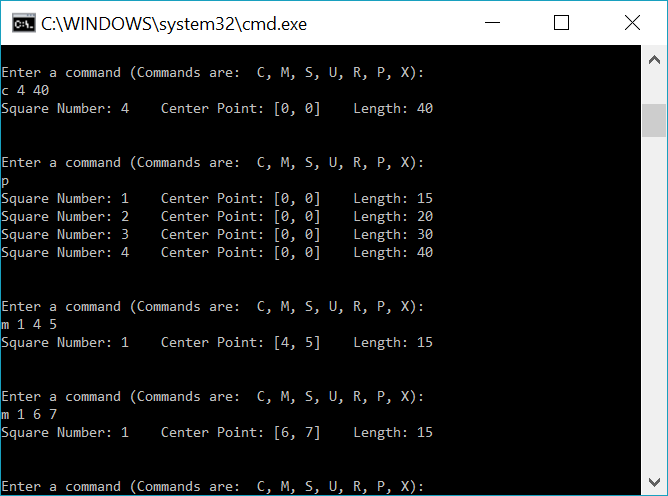
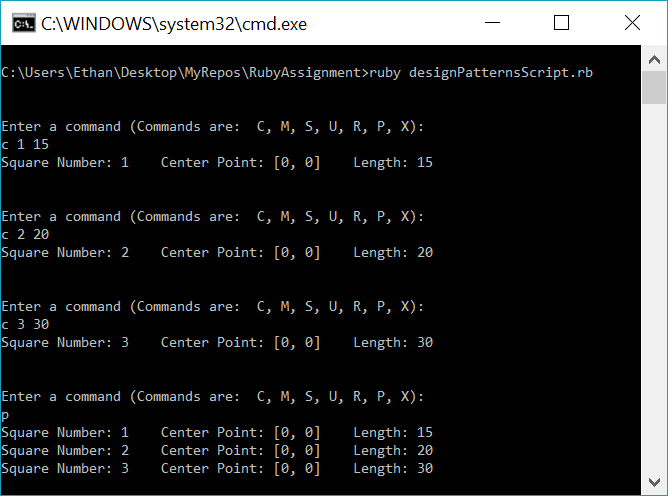
z **=** z **+** 1

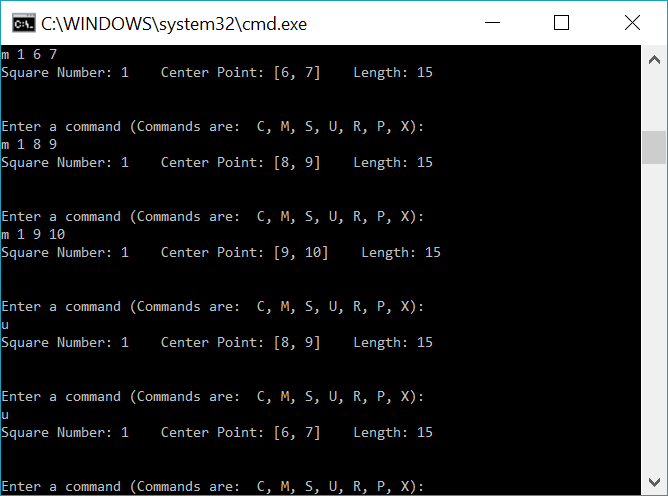
**end**

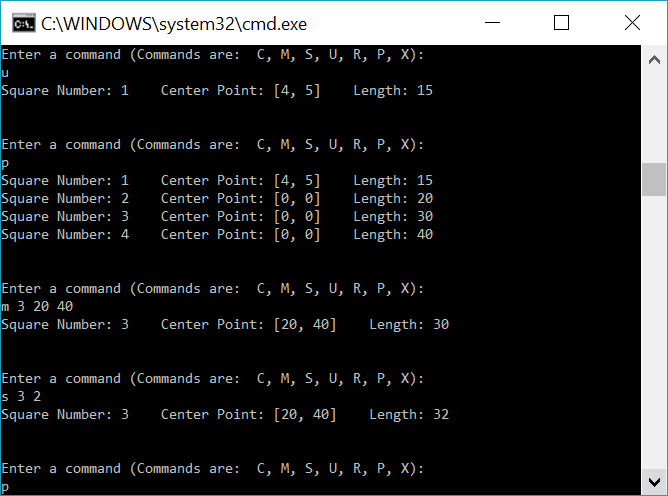
**end**

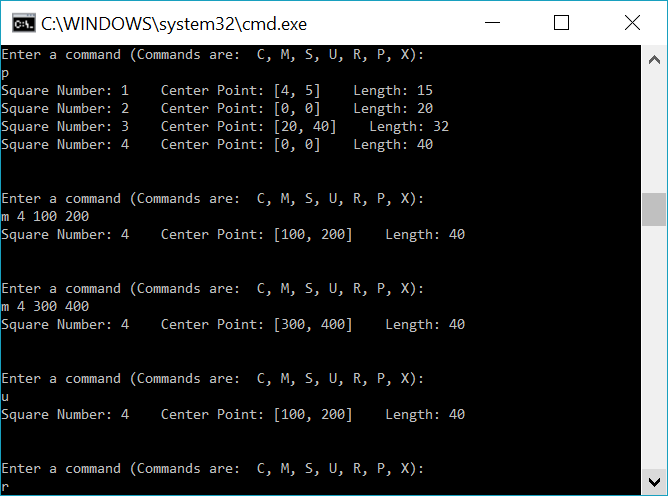
**end**

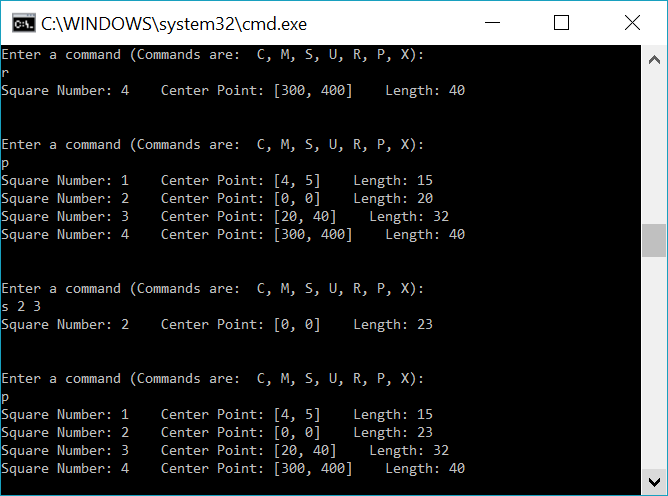
print "\n\nProgram Terminated\n"

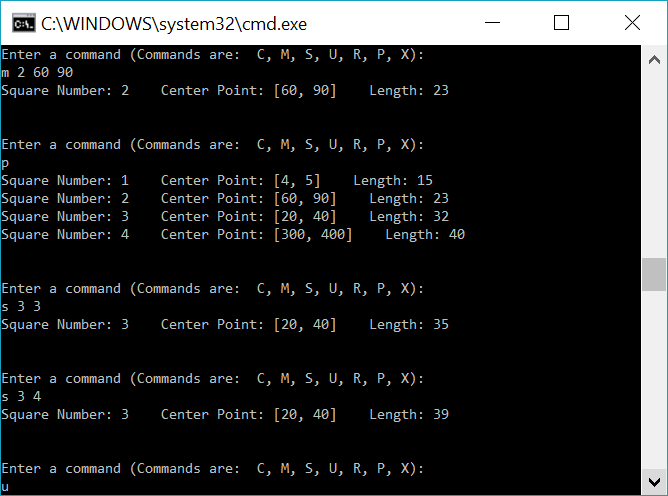
**Problem 1.(Output):**

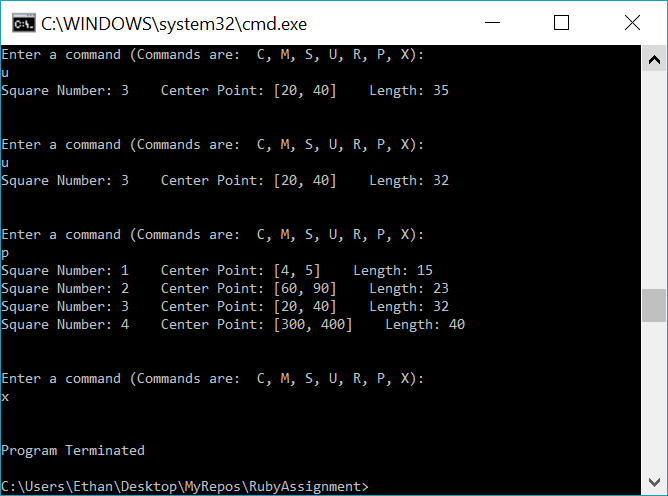




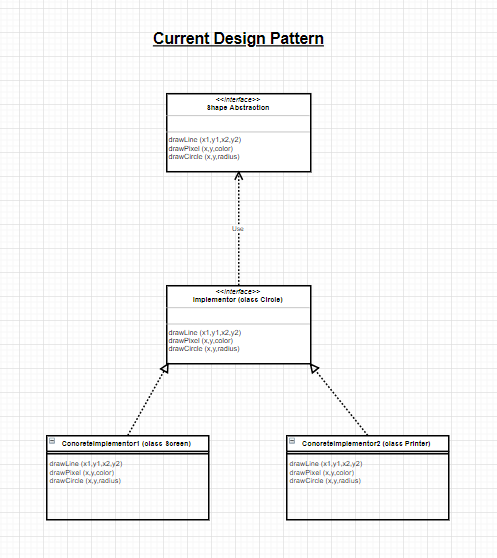


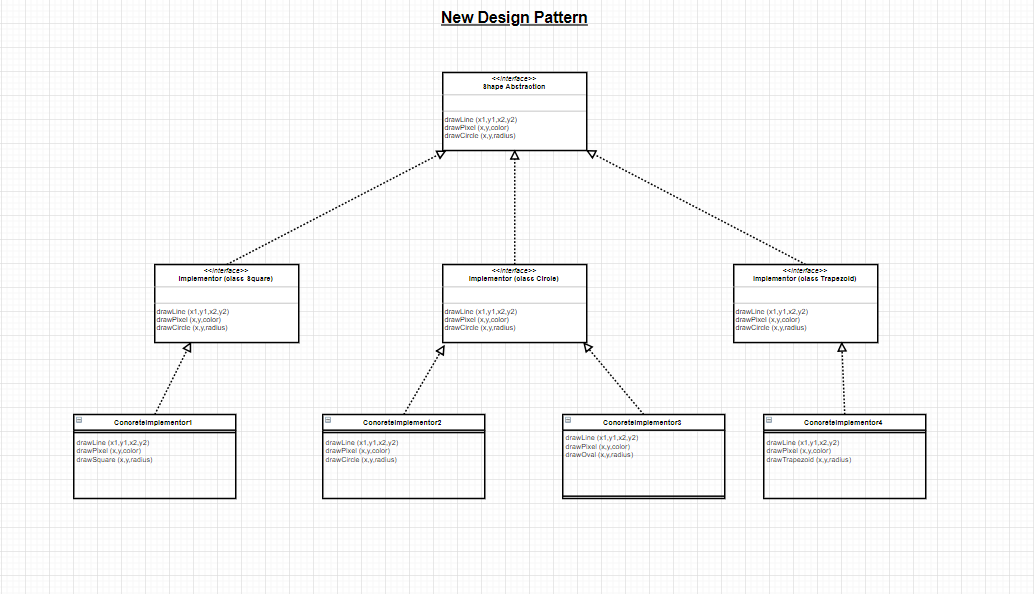






**Problem 2. UML Diagram:**





**Problem 2. Class Prototype**

# Ethan Roberts

# CS 417 Topics in OOP

# Design Patterns

# This is a prototype

**class** **MainAbstract** #main abstract implementation

# Main abstract interface that "Servant" class will implement for some shape

# regardless of if it has edges or no edges

**end**

**class** **Servant** **<** MainAbstract # servant class

#This class inherits the MainAbstract interface and it will

#be used by all shapes

**end**

**class** **Circle** **<** Servant

#using Servant class

**end**

**class** **Polygon** **<** Servant

#using servant class and this class is being extended

**end**

**class** **OutputType** # this class will be implemented for classes that are using outputs

**end**

**class** **XMLFormatter** **<** OutputType

**end**

**class** **OutputPolygon** **<** OutputType

**end**

**class** **OutputCircle** **<** OutputType

**end**