'''

Keni Mou

Kmou1@binghamton.edu

Assignment11 Part1-2

Lab Section 52

CA Kyle Mille

'''

import random

import tkinter.messagebox

#Constants

MIN\_SIDES=4

'''

User defined exception class (subclass of Exception)

'''

class BadArgument(Exception):

def \_\_init\_\_(self):

self.\_\_title="Bad Argument"

self.\_\_message="Sides must be greater than or equal to four"

def getTitle(self):

return self.\_\_title

def \_\_str\_\_(self):

return self.\_\_message

'''

Models a single multi-sided die

'''

#param sides(int)

#return true if sides is valid

def isValid(sides):

sidesStr=str(sides)

return sidesStr.isdigit() and int(sides)>=MIN\_SIDES

class DieMultiSided:

# ------------------------------------------------------

# Constructor

# param initial(int)

def \_\_init\_\_(self, sides=6, value= -1):

self.\_\_value = value

self.\_\_sides = sides

if not isValid(self.\_\_sides):

raise BadArgument

# --------------------------------------------------------

# Accessors

def getSides(self):

return self.\_\_sides

def getValue(self):

return self.\_\_value

# -------------------------------------------------------

# Mutators

# roll the dice

def roll(self):

value = random.randrange(1, (self.\_\_sides+1))

self.\_\_value = value

def reset(self):

self.\_\_value=-1

# string ------

def \_\_str\_\_(self):

return " " if self.\_\_value == -1 else "%s"%(self.\_\_value)

'''

# Tester for DieMultiSided class

def main():

print("Create default die")

die = DieMultiSided()

print("Value of sides: %d" % die.getSides())

print("Value (int): %d" % die.getValue())

print("Value (str): %s" % die)

die.roll()

print("Value (int): %d" % die.getValue())

print("Value (str): %s" % die)

print("Create 12-sided die")

die = DieMultiSided(12)

print("Value of sides: %d" % die.getSides())

print("Value (int): %d" % die.getValue())

print("Value (str): %s" % die)

inputStr = ''

while not inputStr:

die.roll()

print("Value: %d" % die.getValue())

print("Value: %s" % die)

inputStr = input("Press any key to exit loop: ")

print("Create die with invalid state")

try:

die = DieMultiSided(3)

print("Value of sides: %d" % die.getSides())

print("Value (int): %d" % die.getValue())

print("Value (str): %s" % die)

die.roll()

print("Value (int): %d" % die.getValue())

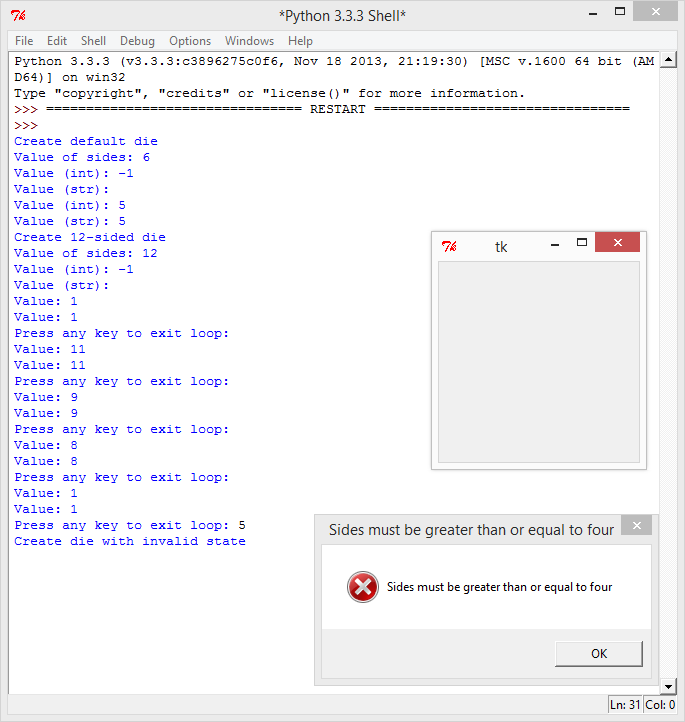
print("Value (str): %s" % die)

except BadArgument as err:

tkinter.messagebox.showerror(err.getTitle(), str(err))

main()

'''



'''

Keni Mou

Kmou1@binghamton.edu

Assignment11 Part3

Lab Section 52

CA Kyle Mille

'''

'''

Notes: should be accompanied by labeled sketch of GUI interface

This GUI lives inside its parent's window, so it doesn't have its own

all names start with self.\_\_

Provides control and view GUI for DieMultiSided model

Models single multi-sided die that can be rolled

Model:

die (DieMultiSided)

Views:

rollValueLabel (Label)

rollValue (StringVar)

Controllers:

dieLabel (Label)

numSidesEntryBox (Entry)

rollButton (Button)

Organizational widgets:

parentFrame (Frame)

dieFrame (Frame)

Other instance objects:

parent (DiceGUI) - owner of this instance

rollValue (StringVar)

Classes Used:

BadArgument

DieMultiSided

'''

# 1. define imports

from tkinter import \*

import dieMultiSided

# 2. define class

class DieGUI:

#-- Constructor ---------------------------------------------------------------

# 3. define constructor, parent parameter

# param parent GUI (DiceGUI) - owner of this instance

def \_\_init\_\_(self, parent):

# 4. Create null model for now (so we have a reference for later)

self.\_\_die =None # Will be created from dieStr.DieMultiSided(sides)

# 5. Get parent GUI object and get parent frame from it

# This objects widgets will live in parent frame or window

self.\_\_parent = parent

self.\_\_parentFrame = self.\_\_parent.getDiceFrame()

# 6. Create frame for this die

self.\_\_dieFrame = Frame(self.\_\_parentFrame)

# 7. Create static label and entry box for creating die

self.\_\_rollValueLabel=Label(self.\_\_dieFrame, text="Sides")

self.\_\_numSidesEntryBox=Entry(self.\_\_dieFrame,width=5)

# Bind entry box to createDie event handler

# User must specify number of sides before creation

self.\_\_numSidesEntryBox.bind('<Return>', self.createDie)

# 8. Create button controller

# set event handler to rollDie

self.\_\_rollButton=Button(self.\_\_dieFrame, text = "Roll",command=self.rollDie)

self.\_\_rollButton.config(state="disabled")

# 9. Create and set up StringVar and dynamic label viewer

# Initialize StringVar to ' '

self.\_\_rollValue=StringVar()

self.\_\_rollValue.set(" ")

self.\_\_dieLabel=Label(self.\_\_dieFrame, textvariable=self.\_\_rollValue)

# 10. Pack the widgets into the dieFrame

self.\_\_rollValueLabel.pack(side="left")

self.\_\_numSidesEntryBox.pack(side="left")

self.\_\_rollButton.pack(side="left")

self.\_\_dieLabel.pack(side="left")

# 11. Pack the dieFrame into the parent frame

self.\_\_dieFrame.pack()

#-- Accessors -----------------------------------------------------------------

# 12. Create Accessors

# return dieFrame (Frame)

def getFrame(self):

return self.\_\_dieFrame

# invoke get() (StringVar)

# return rollValue (StringVar)

def getRollValue(self):

rollValue=str(self.\_\_die.getValue())

return rollValue

#-- Mutators ------------------------------------------------------------------

# 13. Create Mutators

# Reset before next turn

# invoke:

# reset() (DieMultiSided)

# set() (StringVar)

# \_\_str\_\_() (DieMultiSided)

# enableRoll() (self)

def resetDie(self):

self.\_\_die.reset()

self.\_\_rollValue.set(" ")

self.enableRoll()

# Enable rollButton after all dice have been created and after each turn

# invokes:

# config (Button)

def enableRoll(self):

self.\_\_rollButton.config(state='normal')

#-- EVENT HANDLERs ----------------------------------------------------------

# Create die with given number of sides

# invoke:

# get() (Entry)

# config() (Entry)

# \_\_isValid() (DieGUI)

# \_\_str\_\_() (DieMultiSided)

# incrementNumberCreated() (DiceGUI)

# allDiceHaveBeenCreated() (DiceGUI)

# enableRollButtons() (DiceGUI)

def createDie(self, event):

# 14. Get number of sides on die string from entry box

rollValueStr=self.\_\_numSidesEntryBox.get()

try:

# 15. Check for invalid entry (e.g.,number of sides str isn't all digits)

# and if so, Raise BadArgument exception

if not dieMultiSided.isValid(rollValueStr):

raise dieMultiSided.BadArgument

# 16. Create model and disable entry box

self.\_\_die=dieMultiSided.DieMultiSided(int(rollValueStr))

self.\_\_numSidesEntryBox.config(state = 'disabled')

# 17. Let parent know that another die has been created

self.\_\_parent.incrementNumberCreated()

# print("line 156") #debug

# 18. Have parent enable all roll buttons if all dice have been created

if self.\_\_parent.allDiceHaveBeenCreated():

# print("line 160") #debug

self.\_\_parent.enableRollButtons()

# print("line 162") #debug

# 19. invalid entry

except dieMultiSided.BadArgument as err:

# 18. Warn user, clear entry box

messagebox.showerror(err.getTitle(), str(err))

# Roll die, set value, increment parent's roll counter, disable button

# Sum rolls if all other dice have been rolled

# invoke:

# roll() (DieStr)

# set() (StringVar)

# \_\_str\_\_() (DieMultiSided)

# config() (Button)

# incrementRollCounter() (DiceGUI)

# allDiceHaveBeenRolled() (DiceGUI)

# sumRolls() (DiceGUI)

def rollDie(self):

# 19. Roll the die and set the StringVar with the die value

self.\_\_die.roll()

rollValue=str(self.\_\_die.getValue())

self.\_\_rollValue.set(rollValue)

# 20. Let parent know that die has been rolled and disable roll button

self.\_\_parent.incrementRollCounter()

self.\_\_rollButton.config(state='disabled')

# 21. Have parent sum the rolls if all dice have been rolled

if self.\_\_parent.allDiceHaveBeenRolled():

self.\_\_parent.sumRolls()

'''

# -----------------------------------------------------------------------------

# Test Class

# Write minimal parent GUI class for testing purposes

class Parent:

def \_\_init\_\_(self):

# Create window and test widget frame

self.\_\_win = Tk()

self.\_\_parentWidgets = Frame(self.\_\_win)

# Create labels and IntVar for sum (of one die)

self.\_\_sumLabel = Label(self.\_\_parentWidgets, text = 'sum')

self.\_\_sumVar = IntVar()

self.\_\_sumVar.set(0)

self.\_\_sumValue = Label(self.\_\_parentWidgets, textvariable = self.\_\_sumVar)

# Pack widgets, pack frame

self.\_\_sumLabel.pack(side = 'left')

self.\_\_sumValue.pack(side = 'left')

self.\_\_parentWidgets.pack()

# Create frame for die GUI, create die GUI

self.\_\_diceFrame = Frame(self.\_\_win)

self.\_\_die = DieGUI(self)

# Pack frame for die GUI

self.\_\_diceFrame.pack()

# Start listener

mainloop()

# ---------------------------------------------------------------------------

# Write stubbed versions of all necessary parent GUI methods

# For testing purposes there will only be one die GUI at a time, so write as

# little code as possible

# return the frame that will hold the die GUIE

def getDiceFrame(self):

# print(1)

return self.\_\_diceFrame

# Only one will be created

def incrementNumberCreated(self):

# print(2)

return 1

# Only one will be rolled per turn

def incrementRollCounter(self):

# print(3)

return 1

# return True since the only one has been rolled

def allDiceHaveBeenRolled(self):

# print(4)

return True

# return True since the only one has been created

def allDiceHaveBeenCreated(self):

# print(5)

return True

# turn is over after one roll

def enableRollButtons(self):

# print(6)

self.\_\_die.enableRoll()

# only one has been rolled

def sumRolls(self):

# print(7)

self.\_\_sumVar.set(int(self.\_\_die.getRollValue().strip()))

self.enableRollButtons()

# -----------------------------------------------------------------------------

# Write main() tester class to create parent GUIS to exercise one at a time

def main():

reg1 = Parent() # create 4-sided die

# Press <Enter> after entering 4 in the entry box

# Keep on rolling until you are satisfied that it generates all values from

# 1-4 inclusive and only those values

# Click the x in the right-hand corner of the window when you are satisfied

reg2 = Parent() # create 6-sided die

# Press <Enter> after entering 6 in the entry box

# Keep on rolling until you are satisfied that it generates all values from

# 1-6 inclusive and only those values

# Click the x in the right-hand corner of the window when you are satisfied

reg3 = Parent() # create 12-sided die

# Press <Enter> after entering 12 in the entry box

# Keep on rolling until you are satisfied that it generates all values from

# 1-12 inclusive and only those values

# Click the x in the right-hand corner of the window when you are satisfied

bad = Parent() # create 3-sided die

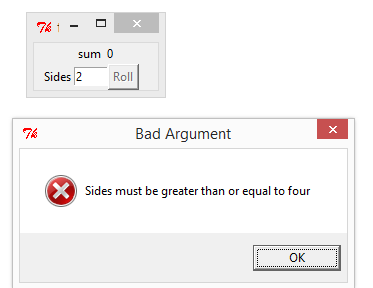
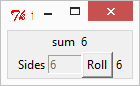
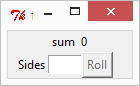
# Press <Enter> after entering 3 in the entry box

# It should not be possible to create a die with less than 4 sides

# Click the x in the right-hand corner of the window when you are satisfied

main()

'''



'''

Keni Mou

Kmou1@binghamton.edu

Assignment 11 Model

Lab Section 52

CA Kyle Mille

'''

class CounterWP:

def \_\_init\_\_(self):

self.\_\_value = 0

def getValue(self):

return self.\_\_value

def \_\_str\_\_(self):

return "Count" + str(self.\_\_value)

def increment(self):

self.\_\_value += 1

def decrement(self):

self.\_\_value -=1

def reset(self):

self.\_\_value = 0

def set(self, avalue):

self.\_\_value = avalue

def isNegative(self):

return self.\_\_value < 0

CounterWP()

'''

Keni Mou

Kmou1@binghamton.edu

Assignment11 Part4

Lab Section 52

CA Kyle Mille

'''

'''

Note: should be accompanied by labeled sketch of GUI interface

Provides control and view GUI for set of dice

that can be rolled and summed once per turn in a game

Note: all names start with self.\_\_

Output to DiceGUI labels:

sum (StringVar)

Input to DiceGUI:

numberDice (Entry)

reset (Button)

Modules imported:

tkinter

dieGUI

dieMultisided

counterWP

Classes Used:

DieGUI

DieMultisided

CounterWP

'''

# 1. Define imports

from tkinter import \*

import counterWP

import dieMultiSided

import dieGUI

# 2. Define class

class DiceGUI(object):

#-- Constructor --------------------------------------------------------

# 3. Define constructor

# Model:

# dice (list of DieGUI)

# Views:

# inputLabel (Label)

# resultLabel (Label)

# resultValue (Label)

# diceLabel (Label)

# sum (IntVar)

# Controls:

# entry (Entry)

# reset (Button)

# Organizational widgets:

# win (Tk)

# inputFrame (Frame)

# resultFrame (Frame)

# diceLabelFrame (Frame)

# diceFrame (Frame)

# Other instance variables and objects:

# numberDice (int)

# numberCreated (int)

# rollCounter (CounterWP)

def \_\_init\_\_(self):

# 4. Create main window

self.\_\_win = Tk()

#--------------------------------------------------------------------------

# 5. Create model, count variables, and counter

self.\_\_dice = [] # model

self.\_\_numberDice = 0

self.\_\_numberCreated = 0

self.\_\_rollCounter = counterWP.CounterWP() # secondary model

#--------------------------------------------------------------------------

# 6. Create user input frame

self.\_\_inputFrame=Frame(self.\_\_win)

# 7. Create static label and entry box for entering number of dice

# Bind entry box to createDice event handler

self.\_\_inputLabel=Label(self.\_\_inputFrame, text="Number of Dice:")

self.\_\_inputLabel.bind("<Return>",self.createDice)

self.\_\_entry=Entry(self.\_\_inputFrame, width=6)

self.\_\_entry.bind("<Return>", self.createDice)

# 8. Pack widgets into input frame

self.\_\_inputLabel.pack(side="left")

self.\_\_entry.pack(side="left")

# 9. Pack input frame in the main window

self.\_\_inputFrame.pack()

#--------------------------------------------------------------------------

# 10. Create result frame

self.\_\_resultFrame=Frame(self.\_\_win)

# 11. Create reset button, set its event handler to clearRolls and its

# current state to 'disabled'

self.\_\_reset=Button(self.\_\_resultFrame,text="Reset",command=self.clearRolls)

self.\_\_reset.config(state="disabled")

# 12. Create static result Label

self.\_\_resultLabel=Label(self.\_\_resultFrame,text="You rolled:")

# 13. Create sum IntVar and initialize to 0

self.\_\_sum=IntVar()

self.\_\_sum.set(0)

# 14. Create dynamic result value label

self.\_\_resultValue=Label(self.\_\_resultFrame,textvariable=self.\_\_sum)

# 15. Pack widgets into result frame

self.\_\_reset.pack(side="left")

self.\_\_resultLabel.pack(side="left")

self.\_\_resultValue.pack(side="left")

# 16. Pack result frame into main window

self.\_\_resultFrame.pack()

#--------------------------------------------------------------------------

# 17. Create dice label frame

self.\_\_diceLabelFrame=Frame(self.\_\_win)

# 18. Create static dice labels

self.\_\_diceSeparator = Label(self.\_\_diceLabelFrame,

text = '---------------------------')

self.\_\_diceLabel = Label(self.\_\_diceLabelFrame, \

text = 'DICE')

# 19. Pack dice label into dice label frame

self.\_\_diceLabel.pack()

# 20. Pack dice label frame into main window

self.\_\_diceLabelFrame.pack()

#--------------------------------------------------------------------------

# 21. Create the dice Frame -- but don't pack it yet

# Will be packed after number of DieGUI objects to be created is known

# (See createDice() event handler)

self.\_\_diceFrame=Frame(self.\_\_win)

#--------------------------------------------------------------------------

# 22. Start the main loop

mainloop()

#-- Predicates ---------------------------------------------------------

# 23. Create Predicates

# Check number of dice and number created count variables

def allDiceHaveBeenCreated(self):

# print("before comparison",self.\_\_numberDice,self.\_\_numberCreated) #debug

# if int(self.\_\_numberDice) == int(self.\_\_numberCreated): #debug

# print("after",self.\_\_numberDice,self.\_\_numberCreated) #debug

return int(self.\_\_numberDice) == self.\_\_numberCreated

# Compare roll count with number of dice

# invoke getValue() (CounterWP)

def allDiceHaveBeenRolled(self):

return self.\_\_rollCounter.getValue()==int(self.\_\_numberDice)

#-- Accessors ----------------------------------------------------------

# 24. Create accessors

# return diceFrame (Frame) - contains all dice

def getDiceFrame(self):

return self.\_\_diceFrame

# invoke getValue() (CounterWP)

# return count (int) of rolls this turn

def getRollCounterValue(self):

return self.\_\_rollCounter.getValue()

#-- Mutators -----------------------------------------------------------

# 25. Create Mutators

def incrementNumberCreated(self):

self.\_\_numberCreated += 1

## print(self.\_\_numberCreated) #debug

## print(self.\_\_numberDice) #debug

# invoke increment() (CounterWP)

def incrementRollCounter(self):

self.\_\_rollCounter.increment()

# invoke enableRoll() (DieGUI) on each die in collection

def enableRollButtons(self):

for each in self.\_\_dice:

each.enableRoll()

# invoke getRollValue() (DieGUI)

# invoke set() (IntVar)

# invoke config() (Button)

def sumRolls(self):

# Sum all rolls by iterating over each die in collection

# Watch out for str vs. int types

# Set IntVar to sum

# Enable reset button

sumNum = 0

for i in self.\_\_dice:

sumNum+=int(i.getRollValue())

self.\_\_sum.set(sumNum)

self.\_\_reset.config(state="normal")

#-- EVENT HANDLERS ---------------------------------------------------

# Create collection of dice requested by user

# invoke:

# get() (Entry)

# config() (Entry)

# delete() (Entry)

# isdigit() (str)

# append() (list of DiceGUI)

# pack() (Frame)

# showerror() (messagebox)

def createDice(self, event):

# 26. Get number of dice (as string) requested by user from entry box

self.\_\_numberDice=self.\_\_entry.get()

# 27. Check that number of dice string is all digits

if self.\_\_numberDice.isdigit():

# 28. Store as integer if it is

numberDiceInt=int(self.\_\_numberDice)

# 29. Disable entry box

self.\_\_entry.config(state="disabled")

# 30. Loop for number of dice to be created

for i in range(numberDiceInt):

# 31. Create dieGUI and append to list of dice

self.\_\_dice.append(dieGUI.DieGUI(self))

# 32. Pack the completed dice frame into the main window

self.\_\_diceFrame.pack()

# 33. Otherwise, handle invalid entry

else:

# 34. Warn user

messagebox.showerror("Invalid Input","You can only input a int")

# 35. Clear entry box

self.\_\_entry.delete(0,END)

# Reset roll counter, sum, all dice

# invoke:

# reset() (CounterWP)

# set() (IntVar)

def clearRolls(self):

# 36. Reset roll counter and sum

self.\_\_rollCounter.reset()

self.\_\_sum.set(0)

# 37. Loop to reset each die in collection

for i in self.\_\_dice:

i.resetDie()

# 38. Disable reset button

self.\_\_reset.config(state="disabled")

# 39. Create instance of class

DiceGUI()

