**11/24/15**

"""

Donato, Brandon

bdonato1@binghamton.edu

CS 110 - B57

Jia Yang

Assignment11Ex1

"""

'''

This class represents a patron

A Patron has a name, a status and zero or more books checked out

'''

class Patron:

# Class Variables ----------------------------------------------------------

# Maximum number of books Patron can take out (int)

MAX\_BOOKS\_OUT = 3

# Current status of this Patron (str)

# Will be combined with name of Patron

STATUS = [" can borrow up to 3 books", " can borrow two more books", \

" can borrow one more book", " must return book(s)"]

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

# params: name - name of Patron(str)

# initialize: self.\_\_name (str), to parameter name,

# self.\_\_numBooksOut (int) to 0, and self.\_\_status() (str)

# to STATUS[0]

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

# your code here

self.\_\_name = name

self.\_\_numBooksOut = 0

self.\_\_status = Patron.STATUS[0]

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

# True if less then max books checked out, False otherwise

def canCheckOutBooks(self):

return (self.\_\_numBooksOut < Patron.MAX\_BOOKS\_OUT)

**Page 1 of 14**

# True if books checked out, False otherwise

def hasCheckedOutBooks(self):

# your code here

return (self.\_\_numBooksOut > 0)

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

# returns: name (str)

def getName(self):

# your code here

return self.\_\_name

# returns: status (str)

def getStatus(self):

# your code here

return self.\_\_status

# returns: number of books out (int)

def getNumBooksOut(self):

# your code here

return self.\_\_numBooksOut

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

# set to STATUS indexed by number of books out

def \_\_updateStatus(self):

# your code here

self.\_\_status = Patron.STATUS[self.getNumBooksOut()]

# invokes: updateStatus()

def increment(self):

# your code here

self.\_\_numBooksOut += 1

self.\_\_updateStatus()

# invokes updateStatus()

def decrement(self):

# your code here

self.\_\_numBooksOut -= 1

self.\_\_updateStatus()

# Comparators --------------------------------------------------------------

# Already written for you:

**Page 2 of 14**

# You will need to include these in order to sort Patron objects

# Shows how two Patrons can be compared with respect to the < relationship

# params: other - another Patron object

# invokes: type()

# returns: True when they are not the same Patron and other is a Patron

# object and name of this Patron is lexicographically less than

# name of other Patron, False otherwise (bool)

def \_\_lt\_\_(self, other):

return (not self is other) and (type(self) == type(other)) and \

self.\_\_name < other.\_\_name

# Shows how two Patrons can be compared with respect to the == relationship

# params: other - another Patron object

# invokes: type()

# returns: True when both are same Patron OR both are Patron objects AND

# all attributes are equal, False otherwise (bool)

def \_\_eq\_\_(self, other):

return self is other or \

(type(self) == type(other) and \

self.\_\_name == other.\_\_name and \

self.\_\_status == other.\_\_status and \

self.\_\_numBooksOut == other.\_\_numBooksOut)

# Convert to str -----------------------------------------------------------

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

# your code here

return '\n' + self.\_\_name + self.\_\_status + ", " + \

str(self.\_\_numBooksOut) + " book(s) out"

"""

Donato, Brandon

bdonato1@binghamton.edu

CS 110 - B57

Jia Yang

Assignment11Ex2

"""

'''

This class represents a book with a title, author, status,

a patron to whom the book is checked out, and a list

of patrons waiting for it

'''

**Page 3 of 14**

class Book:

# Class Variables ----------------------------------------------------------

# index when book is first created (int)

NONE = 0

# index when book is loaned successfully (int)

SUCCESSFUL = 1

# index when patron is put on waiting list (int)

WAIT = 2

# index when request for loan is unsuccessful (int)

UNSUCCESSFUL = 3

# index when book is returned (int)

RETURNED = 4

# index when request for loan is invalid (int)

INVALID = 5

# status of most recent transaction with respect to this book (str)

# Will be combined with name of patron participating in transaction and

# and title of this book"""

TRANS\_STATUS = [" No transactions yet",

" successfully checked out ",

" has been put on waiting list for ",

" must return books before taking out ",

" has returned ",

" has recorded an invalid transaction re: "]

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

# Creates a new book with the given title and author

# params: title (str) and author (str) of book

# initialize: self.\_\_title (str) and self.\_\_author (str) to value of

# incoming parameters

# self.\_\_transactionStatus (str) to no transactions yet,

# self.\_\_patron (Patron) & self.\_\_waitList (list of Patrons)

# to null/empty values

def \_\_init\_\_(self, title, author):

# your code here

self.\_\_title = title

self.\_\_author = author

self.\_\_transactionStatus = Book.TRANS\_STATUS[Book.NONE]

**Page 4 of 14**

self.\_\_patron = None

self.\_\_waitlist = []

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

# returns: True when book is already loaned out, False otherwise (bool)

def isCheckedOut(self):

# your code here

return bool(self.\_\_patron)

# invokes len()

# returns: True if Patron(s) are waiting for book, False otherwise (bool)

def isReserved(self):

# your code here

return bool(self.\_\_waitlist)

# params: patron - a particular patron (Patron)

# returns: True when Patron has checked out book, False otherwise (bool)

def hasBook(self, patron):

# your code here

return (self.\_\_patron == patron)

# params: patron - a particular patron (Patron)

# returns: True when given Patron is on waiting list, False otherwise (bool)

def isInWaitList(self, patron):

# your code here

return (patron in self.\_\_waitlist)

# Both return and lend

# returns: True when previous transaction is "returned" and current

# transaction is "lend", False otherwise (bool)

def \_\_isTwoPartStatus(self):

# your code here

return "returned" in self.\_\_transactionStatus \

and ("\n" not in self.\_\_transactionStatus)

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

# returns: title of book (str)

def getTitle(self):

# your code here

return self.\_\_title

**Page 5 of 14**

# returns: author of book (str)

def getAuthor(self):

# your code here

return self.\_\_author

# returns: Patron who has checked out book (Patron)

def getPatron(self):

# your code here

return self.\_\_patron

# returns: record of latest book transaction (str)

def getTransactionStatus(self):

# your code here

return self.\_\_transactionStatus

# invokes: str()

# returns: str representation of waiting list for book (str)

def getWaitListStr(self):

# your code here

nameStr = "Waitlist: "

for names in self.\_\_waitlist:

nameStr += str(names)

return nameStr + '\n'

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

# This method delegates all responsibilities to private methods of class

# invokes: hasBook(), isInWaitList(), canCheckOutBooks(), isCheckedOut(),

# \_\_lendBook(), \_\_putInWaitList(), and \_\_setTransactionStatus()

# params: patron - patron trying to borrow book (Patron)

def borrowMe(self, patron):

# your code here

if self.hasBook(patron) or self.isInWaitList(patron):

self.\_\_setTransactionStatus(patron.getName(), Book.INVALID)

elif patron.canCheckOutBooks() and not self.isCheckedOut():

self.\_\_lendBook(patron)

self.\_\_setTransactionStatus(patron.getName(), Book.SUCCESSFUL)

elif patron.canCheckOutBooks() and self.isCheckedOut():

self.\_\_putInWaitList(patron)

self.\_\_setTransactionStatus(patron.getName(), Book.WAIT)

else:

self.\_\_setTransactionStatus(patron.getName(), Book.UNSUCCESSFUL)

**Page 6 of 14**

# Return book: release current patron, try to lend to waiting patron

# This method delegates all responsibilities to private methods of class

# invokes: isCheckedOut(), isReserved(), getName(),

# \_\_resetPatron,(), \_\_lendToNextPatron(), and

# \_\_setTransactionStatus()

def returnMe(self): # mutator controller

# your code here

if self.isCheckedOut():

self.\_\_setTransactionStatus(self.\_\_patron.getName(), Book.RETURNED)

self.\_\_resetPatron()

if self.isReserved():

self.\_\_lendToNextPatron()

# invokes: increment() (Patron class)

# params: patron - Patron borrowing book (Patron)

def \_\_lendBook(self, patron):

# your code here

patron.increment()

self.\_\_patron = patron

# invokes: decrement() (Patron class)

def \_\_resetPatron(self):

# your code here

self.\_\_patron.decrement()

self.\_\_patron = None

# Lend book to waiting patron if eligible; if not, remove from wait List

# invokes: isCheckedOut(), isReserved(),

# pop() (from list), borrowMe()

def \_\_lendToNextPatron(self): # waitList mutator

# your code here

while not self.isCheckedOut() and self.isReserved():

self.borrowMe(self.\_\_waitlist.pop(0))

# params: patron - Patron to put in waiting list (Patron)

# invokes: append() (to list)

def \_\_putInWaitList(self, patron): # waitList mutator

# your code here

if (not self.isInWaitList(patron)):

self.\_\_waitlist.append(patron)

# Creates string describing latest transaction

# Combines name of patron participation in transaction with

# status of most recent transaction and title of this book

**Page 7 of 14**

# params: name - name of Patron participating in transaction (str)

# index - index of transaction in TRANS\_STATUS (int)

# invokes: \_\_isTwoPartStatus()

def \_\_setTransactionStatus(self, name, index):# transStatus mutator

# your code here

if self.\_\_isTwoPartStatus():

self.\_\_transactionStatus += "\n"+ name + Book.TRANS\_STATUS[index] \

+ self.getTitle()

else:

self.\_\_transactionStatus = name + Book.TRANS\_STATUS[index] \

+ self.getTitle()

# Comparators --------------------------------------------------------------

# Already written for you:

# Include these in order to sort Book objects

# Shows how two Books can be compared with respect to the < relationship

# params: other - another object

# invokes: type()

# returns: True when they are not same Book and other is Book object and

# title of this Book is lexicographically lower than title of

# other Book, False otherwise (bool)"""

def \_\_lt\_\_(self, other):

return (not self is other) and (type(self) == type(other)) and \

self.\_\_title < other.\_\_title

# Shows how two Books can be compared with respect to the == relationship

# params: other - another object

# invokes: type()

# returns: True when both are same Book or both are Book objects and

# title and author are equal, False otherwise (bool)

def \_\_eq\_\_(self, other):

return self is other or \

(type(self) == type(other) and \

self.\_\_title == other.\_\_title) and\

self.\_\_author == other.\_\_author

# Convert to Str -----------------------------------------------------------

# invokes: str(), getWaitListStr()

# returns: str representation of Book object (str)

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

# your code here

**Page 8 of 14**

if self.isCheckedOut():

bookStr = "Title: %s\nAuthor: %s\nChecked out: %s\n%s" % \

(self.\_\_title, self.\_\_author, str(self.\_\_patron),\

self.getWaitListStr())

else:

bookStr = "Title : %s\nAuthor: %s\n%s\n%s" % (self.\_\_title, \

self.\_\_author, "Not checked out", \

self.getWaitListStr())

return bookStr

"""

Donato, Brandon

bdonato1@binghamton.edu

CS 110 - B57

Jia Yang

Assignment11Ex3

"""

from libraryModule import StringGeneratorForDictionaries

'''

This class represents a named library with books and patrons.

A library can add and remove Patrons, add and remove Books,

access information about its Patrons and Books, and keep track of

any transactions taking place with repect to it's Books and Patrons.

This class makes use of the StringGeneratorForDictionaries class found

in the libraryModule file.

'''

class Library:

#-- Class Variables ----------------------------------------------------

# Index when book not in library

NOT\_IN\_LIBRARY = 0

# Index when book added to library

ADD = 1

# Index when book removed from library

REMOVE = 2

# Index when patron not a member of library

NOT\_A\_MEMBER = 3

**Page 9 of 14**

# Index when patron becomes member of library

JOIN = 4

# Index when patron ends membership in library

LEAVE = 5

# Index when book information is accessed

ACCESS = 6

# Index when patron information is accessed

LOOK\_UP = 7

# Most recent transaction with respect to either a book or a patron

TRANS\_STATUS = [" is not in library",

" has been added to the library",

" has been removed from the library ",

" is not a library member ",

" has been added as a library member",

" has been removed as a library member",

" has been accessed",

" member files have been accessed"]

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

# Creates new dictionaries to hold books and patrons

# params: name - name of Library(str)

# initialize: self.\_\_name (str), to parameter name,

# self.\_\_books (dict of Book) and

# self.\_\_patrons() (dict of Patron) to empty dictionaries,

# self.\_\_transactionStatus (str) to TRANS\_STATUS with respect to

# book participating in transaction or

# patron participating in transaction

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

# your code here

self.\_\_name = name

self.\_\_books = {}

self.\_\_patrons = {}

self.\_\_transactionStatus = ""

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

# returns: name of library (str)

def getName(self):

# your code here

return self.\_\_name

**Page 10 of 14**

# returns: record of latest transaction (str)

def getTransactionStatus(self):

# your code here

return self.\_\_transactionStatus

# params: title of Book (str)

# invokes: inLibrary(), \_\_setTransactionStatus()

# returns: Book stored in library (Book)

def getBook(self, title):

# your code here

if self.inLibrary(title):

self.\_\_setTransactionStatus(title, "", Library.ACCESS)

bookStr = self.\_\_books.get(title)

else:

self.\_\_setTransactionStatus(title, "", Library.NOT\_IN\_LIBRARY)

bookStr = self.\_\_books.get(title)

return bookStr

# params: name of Patron who is member of library (str)

# invokes: isMember(), \_\_setTransactionStatus()

# returns: name of Patron (Patron) or None

def getPatron(self, name):

# your code here

if self.isMember(name):

self.\_\_setTransactionStatus("", name, Library.LOOK\_UP)

patronStr = self.\_\_patrons.get(name, None)

else:

self.\_\_setTransactionStatus("", name, Library.NOT\_A\_MEMBER)

patronStr = self.\_\_patrons.get(name, None)

return patronStr

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

# Checks if book is in library

# params: title - title of Book to search for in library (str)

# returns: True if in library, False otherwise (bool)

def inLibrary(self, title):

# your code here

return (title in self.\_\_books.keys())

# Checks if person is member of library

# params: name - name of Patron to search for in library (str)

**Page 11 of 14**

# returns: True if member of library, False otherwise (bool)

def isMember(self, patronName):

# your code here

return (patronName in self.\_\_patrons.keys())

# Checks if library has any books

# invokes: len()

# returns: True if library has any books, False otherwise (bool)

def hasBooks(self):

# your code here

return (len(self.\_\_books) > 0)

# Checks if library has any members

# invokes: len()

# returns: True if library has any members, False otherwise (bool)

def hasMembers(self):

# your code here

return (len(self.\_\_patrons) > 0)

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

# Set status for latest transaction

# params: title - title of Book participating in transaction (str)

# name = name of Patron participating in transaction (str)

# Note: one of the above should be an empty string

# index into TRANS\_STATUS (int)

def \_\_setTransactionStatus(self, title, name, index):

# your code here

self.\_\_transactionStatus = title + name + \

Library.TRANS\_STATUS[index]

# Adds book to library using its title as a key

# params: book - new Book to be added to library (Book)

# invokes: getTitle() (Book), \_\_setTransactionStatus()

def addBook(self, book):

# your code here

self.\_\_books[book.getTitle()] = book

self.\_\_setTransactionStatus(book.getTitle(), "", Library.ADD)

# Removes book from library and releases borrower if applicable

# params: title - title of Book to remove from library (str)

# invokes: pop() (list),

# isCheckedOut() (Book), getPatron (Book)

# decrement () (Patron)

**Page 12 of 14**

# inLibrary(), \_\_setTransactionStatus()

def removeBook(self, title):

# your code here

if self.inLibrary(title):

removedBook = self.\_\_books.pop(title)

self.\_\_setTransactionStatus(title, "", Library.REMOVE)

if removedBook.isCheckedOut():

patron = removedBook.getPatron()

patron.decrement()

# Adds patron to library using its name as a key

# params: patron - new Patron to add to library (Patron)

# invokes: getName (Patron), \_\_setTransactionStatus()

def addPatron(self, patron):

# your code here

self.\_\_patrons[patron.getName()] = patron

self.\_\_setTransactionStatus("", patron.getName(), Library.JOIN)

# Removes the patron and returns borrowed books if any

# params: name - name of Patron to remove from library (str)

# invokes: pop() (list),

# hasCheckedOutBooks() (Patron)

# getPatron (Book), returnMe (Book)

# isMember(), \_\_setTransactionStatus()

def removePatron(self, name):

# your code here

if self.isMember(name):

removedPatron = self.\_\_patrons.pop(name)

self.\_\_setTransactionStatus("", name, Library.LEAVE)

if removedPatron.hasCheckedOutBooks():

for book in list(self.\_\_books.values()):

if book.getPatron() == removedPatron:

book.returnMe()

#-- Convert to Str -----------------------------------------------------

# Generates string representation of library

# creates: StringGeneratorForDictionaries objects

# invokes: str(), getName(), hasBooks(), hasMembers(),

# getDictString() (StringGeneratorForDictionaries)

# returns: str representation of Library object (str)

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

**Page 13 of 14**

if self.hasBooks():

bookDictStr = StringGeneratorForDictionaries(self.\_\_books,\

"Book Dictionary")

bookStr = bookDictStr.getDictString()

if self.hasMembers():

patronDictStr = StringGeneratorForDictionaries(self.\_\_patrons,\

"Patron Dictionary")

patronStr = patronDictStr.getDictString()

else:

patronStr = "There are no patrons in the library"

else:

bookStr = "There are no books in the library"

if self.hasMembers():

patronDictStr = StringGeneratorForDictionaries(self.\_\_patrons,\

"Patron Dictionary")

patronStr = patronDictStr.getDictString()

else:

patronStr = "There are no patrons in the library"

return self.getName() +': \n'+ bookStr + "\n" + patronStr

**Page 14 of 14**