- ***Name Hamdan Hafeez Malik***
- ***Roll 480469***

Fop II project

You will use object-oriented programming (classes and inheritance) to build a program to monitor news feeds over the Internet. Your program will filter the news, alerting the user when it notices a news story that matches that user's interests (for example, the user may be interested in a notification whenever a story related to the Red Sox is posted).

```
# RSS Feed Filter
# Name:
# Collaborators:
# Time:

import feedparser
import string
import time
import threading
from project_util import translate_html
from mtTkinter import *
from datetime import datetime
import pytz


#======================
# Code for retrieving and parsing
# Google and Yahoo News feeds
# Do not change this code
#======================

def fetch_rss_feed(url):
    """
```

```
    Fetches news items from the RSS url and parses them.
    Returns a list of NewsArticle objects.
    """
    feed = feedparser.parse(url)
    entries = feed.entries
    articles = []
    for entry in entries:
        guid = entry.guid
        title = translate_html(entry.title)
        link = entry.link
        description = translate_html(entry.description)
        pubdate = translate_html(entry.published)

        try:
            pubdate = datetime.strptime(pubdate, "%a, %d %b %Y %H:%M:%S %Z")
            pubdate = pubdate.replace(tzinfo=pytz.timezone("GMT"))
        except ValueError:
            pubdate = datetime.strptime(pubdate, "%a, %d %b %Y %H:%M:%S %z")

        article = NewsArticle(guid, title, description, link, pubdate)
        articles.append(article)
    return articles

#=====================
# Data structure design
#========================================================================

# Problem 1
# TODO: NewsArticle

class NewsArticle:
    def __init__(self, guid, title, description, link, pubdate):
        self.guid = guid
        self.title = title
        self.description = description
        self.link = link
        self.pubdate = pubdate

    def get_guid(self):
        return self.guid

    def get_title(self):
        return self.title

    def get_description(self):
        return self.description

    def get_link(self):
```

```
        return self.link

    def get_pubdate(self):
        return self.pubdate

#======================================================================
# Triggers
#======================

class Trigger:
    def evaluate(self, article):
        raise NotImplementedError

# PHRASE TRIGGERS

# Problem 2
# TODO: PhraseTrigger

class KeywordTrigger(Trigger):
    def __init__(self, phrase):
        self.phrase = phrase.lower()

    def is_phrase_in(self, text):
        text = text.lower()
        for p in string.punctuation:
            text = text.replace(p, ' ')
        words = text.split()
        phrase_words = self.phrase.split()
        for i in range(len(words) - len(phrase_words) + 1):
            if phrase_words == words[i:i + len(phrase_words)]:
                return True
        return False

# Problem 3
# TODO: TitleTrigger

class TitleKeywordTrigger(KeywordTrigger):
    def evaluate(self, article):
        return self.is_phrase_in(article.get_title())

# Problem 4
# TODO: DescriptionTrigger

class DescriptionKeywordTrigger(KeywordTrigger):
    def evaluate(self, article):
        return self.is_phrase_in(article.get_description())

# TIME TRIGGERS
```

```
# Problem 5
# TODO: TimeTrigger
# Constructor:
#        Input: Time has to be in EST and in the format of "%d %b %Y %H:%M:%S".
#        Convert time from string to a datetime before saving it as an attribute.

class DateTimeTrigger(Trigger):
    def __init__(self, time_string):
        est = pytz.timezone("EST")
        self.time = est.localize(datetime.strptime(time_string, "%d %b %Y %H:%M:%S"))

# Problem 6
# TODO: BeforeTrigger and AfterTrigger

class BeforeTrigger(DateTimeTrigger):
    def evaluate(self, article):
        return article.get_pubdate().replace(tzinfo=pytz.timezone("EST")) < self.time

class AfterTrigger(DateTimeTrigger):
    def evaluate(self, article):
        return article.get_pubdate().replace(tzinfo=pytz.timezone("EST")) > self.time

# COMPOSITE TRIGGERS

# Problem 7
# TODO: NotTrigger

class NotTrigger(Trigger):
    def __init__(self, trigger):
        self.trigger = trigger

    def evaluate(self, article):
        return not self.trigger.evaluate(article)

# Problem 8
# TODO: AndTrigger

class AndTrigger(Trigger):
    def __init__(self, trigger1, trigger2):
        self.trigger1 = trigger1
        self.trigger2 = trigger2

    def evaluate(self, article):
        return self.trigger1.evaluate(article) and self.trigger2.evaluate(article)

# Problem 9
# TODO: OrTrigger
```

```
class OrTrigger(Trigger):
    def __init__(self, trigger1, trigger2):
        self.trigger1 = trigger1
        self.trigger2 = trigger2

    def evaluate(self, article):
        return self.trigger1.evaluate(article) or self.trigger2.evaluate(article)

#======================
# Filtering
#======================

# Problem 10
# TODO: Problem 10

def filter_articles(articles, triggerlist):
    filtered_articles = []
    for article in articles:
        for trigger in triggerlist:
            if trigger.evaluate(article):
                filtered_articles.append(article)
                break
    return filtered_articles



#=========================================================================
===


#======================
# User-Specified Triggers
#======================
# Problem 11
def read_trigger_config(filename):
    """
    filename: the name of a trigger configuration file

    Returns: a list of trigger objects specified by the trigger configuration
        file.
    """
    # We give you the code to read in the file and eliminate blank lines and
    # comments. You don't need to know how it works for now!
    trigger_file = open(filename, 'r')
    lines = []
    for line in trigger_file:
        line = line.rstrip()
```

```
        if not (len(line) == 0 or line.startswith('//')):
            lines.append(line)

    # TODO: Problem 11
    # line is the list of lines that you need to parse and for which you need
    # to build triggers

    print(lines) # for now, print it so you see what it contains!


SLEEPTIME = 120 #seconds -- how often we poll

def main_thread(master):
    # A sample trigger list - you might need to change the phrases to correspond
    # to what is currently in the news
    try:
        t1 = TitleTrigger("election")
        t2 = DescriptionTrigger("Trump")
        t3 = DescriptionTrigger("Clinton")
        t4 = AndTrigger(t2, t3)
        triggerlist = [t1, t4]

        # Problem 11
        # TODO: After implementing read_trigger_config, uncomment this line
        # triggerlist = read_trigger_config('triggers.txt')

        # HELPER CODE - you don't need to understand this!
        # Draws the popup window that displays the filtered stories
        # Retrieves and filters the stories from the RSS feeds
        frame = Frame(master)
        frame.pack(side=BOTTOM)
        scrollbar = Scrollbar(master)
        scrollbar.pack(side=RIGHT,fill=Y)

        t = "Google & Yahoo Top News"
        title = StringVar()
        title.set(t)
        ttl = Label(master, textvariable=title, font=("Helvetica", 18))
        ttl.pack(side=TOP)
                cont = Text(master, font=("Helvetica",14), yscrollcommand=scrollbar.set)
        cont.pack(side=BOTTOM)
        cont.tag_config("title", justify='center')
        button = Button(frame, text="Exit", command=root.destroy)
        button.pack(side=BOTTOM)
        guidShown = []
        def get_content(new_story):
            if new_story.get_guid() not in guidShown:
                cont.insert(END, new_story.get_title()+"\n", "title")
```

```
                cont.insert(END, "\n--------------------------------------------------------------\n", "title")
                cont.insert(END, new_story.get_description())
                cont.insert(END,
"\n**********************************************************************\n", "title")
                guidShown.append(new_story.get_guid())

        while True:

            print("Polling . . .", end=' ')
            # Get stories from Google's Top Stories RSS news feed
            stories = process("http://news.google.com/news?output=rss")

            # Get stories from Yahoo's Top Stories RSS news feed
            stories.extend(process("http://news.yahoo.com/rss/topstories"))

            stories = filter_stories(stories, triggerlist)

            list(map(get_content, stories))
            scrollbar.config(command=cont.yview)


            print("Sleeping...")
            time.sleep(SLEEPTIME)

    except Exception as e:
        print(e)


if __name__ == '__main__':
    root = Tk()
    root.title("Some RSS parser")
    t = threading.Thread(target=main_thread, args=(root,))
    t.start()
    root.mainloop()
```