Fundamentals of Programming II

By Saqib Nizar

OOP Project : Source Code

Made by :

Name	CMS
Subhan Bin Yousaf	481281
Momina Haq	468932
Muhammad Ali Shahzadah	466353
Abdul Manan	462449
Fahad Asif	462895

```
import feedparser
import string
import time
import threading
from mtTkinter import *
from datetime import datetime
import pytz
import html

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

#======================
# Utility function to translate HTML
#======================

def translate_html(html_content):
    """
    Replaces HTML character references with the corresponding characters.
    """
    return html.unescape(html_content)

#======================
# Code for retrieving and parsing
# Google and Yahoo News feeds
#======================

def process(url):
    """
    Fetches news items from the rss url and parses them.
    Returns a list of NewsStory instances.
    """
    feed = feedparser.parse(url)
    entries = feed.entries
    ret = []
    for entry in entries:
        guid = entry.get('id', None)
        title = translate_html(entry.get('title', ''))
        link = entry.get('link', '')
        description = translate_html(entry.get('description', entry.get('summary', '')))
        pubdate = translate_html(entry.get('published', ''))

        # Try parsing the date with different formats
        date_formats = ["%a, %d %b %Y %H:%M:%S %Z", "%a, %d %b %Y %H:%M:%S %z",
"%Y-%m-%dT%H:%M:%SZ"]
```

```
        for date_format in date_formats:
            try:
                pubdate = datetime.strptime(pubdate, date_format)
                pubdate = pubdate.replace(tzinfo=pytz.timezone("GMT"))
                break
            except ValueError:
                continue

        newsStory = NewsStory(guid, title, description, link, pubdate)
        ret.append(newsStory)
    return ret

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

class NewsStory:
    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(object):
```

```
    def evaluate(self, story):
        """
        Returns True if an alert should be generated
        for the given news item, or False otherwise.
        """
        raise NotImplementedError

# PHRASE TRIGGERS

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

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

class TitleTrigger(PhraseTrigger):
    def evaluate(self, story):
        return self.is_phrase_in(story.get_title())

class DescriptionTrigger(PhraseTrigger):
    def evaluate(self, story):
        return self.is_phrase_in(story.get_description())

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

class BeforeTrigger(TimeTrigger):
    def evaluate(self, story):
        story_pubdate = story.get_pubdate()
        if story_pubdate.tzinfo is None:
            story_pubdate = pytz.timezone('EST').localize(story_pubdate)
        return story_pubdate < self.time
```

```
class AfterTrigger(TimeTrigger):
    def evaluate(self, story):
        story_pubdate = story.get_pubdate()
        if story_pubdate.tzinfo is None:
            story_pubdate = pytz.timezone('EST').localize(story_pubdate)
        return story_pubdate > self.time

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

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

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

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

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

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

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

def filter_stories(stories, triggerlist):
    """
    Takes in a list of NewsStory instances.
    Returns: a list of only the stories for which a trigger in triggerlist fires.
    """
    filtered_stories = []
    for story in stories:
        for trigger in triggerlist:
            if trigger.evaluate(story):
```

```
            filtered_stories.append(story)
            break
    return filtered_stories

#======================
# User-Specified Triggers
#======================

def read_trigger_config(filename):
    trigger_map = {
        'TITLE': TitleTrigger,
        'DESCRIPTION': DescriptionTrigger,
        'BEFORE': BeforeTrigger,
        'AFTER': AfterTrigger,
        'NOT': NotTrigger,
        'AND': AndTrigger,
        'OR': OrTrigger
    }
    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)

    triggers = {}
    trigger_list = []

    for line in lines:
        parts = line.split(',')
        if parts[0] == 'ADD':
            for name in parts[1:]:
                trigger_list.append(triggers[name])
        else:
            trigger_name = parts[0]
            trigger_type = parts[1]
            if trigger_type in ['TITLE', 'DESCRIPTION']:
                triggers[trigger_name] = trigger_map[trigger_type](parts[2])
            elif trigger_type in ['BEFORE', 'AFTER']:
                triggers[trigger_name] = trigger_map[trigger_type](parts[2])
            elif trigger_type == 'NOT':
                triggers[trigger_name] = trigger_map[trigger_type](triggers[parts[2]])
            elif trigger_type in ['AND', 'OR']:
                triggers[trigger_name] = trigger_map[trigger_type](triggers[parts[2]], triggers[parts[3]])
```

```
    return trigger_list

SLEEPTIME = 120  # seconds -- how often we poll

def main_thread(master):
    try:
        triggerlist = read_trigger_config('triggers.txt')

        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=master.destroy)
        button.pack(side=BOTTOM)
        guidShown = []

        def get_cont(newstory):
            if newstory.get_guid() not in guidShown:
                cont.insert(END, newstory.get_title() + "\n", "title")
                cont.insert(END, "\n----------------------------------------------------------------\n", "title")
                cont.insert(END, newstory.get_description())
                cont.insert(END, "\n*\n", "title")
                guidShown.append(newstory.get_guid())

        while True:
            print("Polling . . .", end=' ')
            stories = process("http://news.google.com/news?output=rss")
            stories.extend(process("http://news.yahoo.com/rss/topstories"))
            stories = filter_stories(stories, triggerlist)
            list(map(get_cont, 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()
```