NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY

Fundamentals Of Programming – II

Lab Project

Student Names:

Mohammad Gulzaib	460917
Syed Fakhar Abbas	466960
Muhammad Dawood Saeed	465231
Haider Nawaz Cheema	480239
Jonathan Sharif	474228

Section:

ME – 15 (C)

Problem:

The problem in this project was to make a news forum which will fetch news articles from google and yahoo and show it on the tab. These articles will be based on the user recent activity.

Solution:

This code is designed to fetch, filter, and display news stories from RSS feeds based on user-specified triggers. The process begins by parsing RSS feeds using the `feedparser` library, which retrieves news items and processes them into `NewsStory` objects. Each `NewsStory` object contains details such as a unique identifier (guid), title, description, link, and publication date.

The main functionality revolves around various triggers that determine if a news story should be flagged based on specific criteria. These triggers include:

1. **PhraseTrigger**: Checks if a specific phrase is present within a given text.
2. **TitleTrigger**: Inherits from `PhraseTrigger` and checks the presence of a phrase in the story title.
3. **DescriptionTrigger**: Inherits from `PhraseTrigger` and checks the presence of a phrase in the story description.
4. **TimeTrigger**: A base class for triggers that involve time-based conditions.
5. **BeforeTrigger**: Inherits from `TimeTrigger` and checks if the story was published before a certain time.
6. **AfterTrigger**: Inherits from `TimeTrigger` and checks if the story was published after a certain time.
7. **NotTrigger**: Inverts the result of another trigger.
8. **AndTrigger**: Combines two triggers and is true if both triggers are true.
9. **OrTrigger**: Combines two triggers and is true if either trigger is true.

The configuration for these triggers is read from a file (`triggers.txt`), which specifies how each trigger is constructed and combined. The code includes error handling to manage incorrect configurations.

The main graphical user interface (GUI) is built using the `Tkinter` library, displaying the news stories that match the triggers. The interface includes a text widget to show the news content, a scrollbar for navigation, and an exit button.

A separate thread runs the main process, which continuously fetches and filters news stories every two minutes (adjustable via `SLEEPTIME`). Matching stories are displayed in the GUI, highlighting titles, publication dates, descriptions, and links. The program ensures the interface is regularly updated with new stories that meet the specified trigger conditions.

Importing Libraries and Modules

- **feedparser**: Used for parsing RSS feeds.
- **string**: Provides a list of punctuation characters used for cleaning text.
- **time, threading**: Used for managing the timing of news updates and threading for concurrent execution.
- **project_util, translate_html**: Custom modules for HTML translation.
- **mtTkinter**: A modified version of Tkinter that is thread-safe.
- **datetime, pytz**: Handle date and time operations, including time zone conversions.

Data Structures and Classes

NewsStory Class

The `NewsStory` class encapsulates information about a news story:

- **Attributes**:
  - `guid`: Globally unique identifier.
  - `title`: Title of the news story.
  - `description`: Description of the news story.
  - `link`: URL link to the news story.
  - `pubdate`: Publication date as a `datetime` object.
- **Methods**:
  - `get_guid()`, `get_title()`, `get_description()`, `get_link()`, `get_pubdate()`: Return corresponding attributes.

Triggers

Base Trigger Class

- **Trigger**: Abstract base class with the method `evaluate` that needs to be implemented by subclasses.

PhraseTrigger

- **PhraseTrigger**: Inherits from `Trigger`. Checks if a specific phrase is in the given text.
  - **Methods**:
    - `__init__(phrase)`: Initializes with a phrase, converting it to lowercase.
    - `is_phrase_in(text)`: Checks if the phrase is in the text after removing punctuation and splitting into words.

Specific Triggers

- **TitleTrigger**: Inherits from `PhraseTrigger` and checks if the phrase is in the story title.
- **DescriptionTrigger**: Inherits from `PhraseTrigger` and checks if the phrase is in the story description.

- **TimeTrigger**: Abstract base class for time-based triggers
    - **Methods**:
        - `__init__(time_string)`: Converts a time string to a `datetime` object in EST timezone.
- **BeforeTrigger**: Inherits from `TimeTrigger` and checks if the story was published before the specified time.
- **AfterTrigger**: Inherits from `TimeTrigger` and checks if the story was published after the specified time.
- **NotTrigger**: Inverts the result of another trigger.
- **AndTrigger**: Combines two triggers and returns true if both are true.
- **OrTrigger**: Combines two triggers and returns true if either is true.

Reading Trigger Configuration

- **read_trigger_config(filename)**: Reads a configuration file specifying triggers.
    - **File Format**: Each line specifies a trigger or an operation (`ADD`) to combine triggers into a list.
    - **Parsing**:
        - Ignores empty lines and comments.
        - Creates and stores triggers in a dictionary based on the configuration.
        - Supports `TitleTrigger`, `DescriptionTrigger`, `BeforeTrigger`, `AfterTrigger`, `NotTrigger`, `AndTrigger`, `OrTrigger`.
        - Handles errors for invalid configurations and logs messages.

Filtering Stories

- **filter_stories(stories, triggerlist, cont)**: Filters stories based on a list of triggers.
    - **Parameters**:
        - `stories`: List of `NewsStory` objects.
        - `triggerlist`: List of triggers to apply.
        - `cont`: Text widget to display log messages.
    - **Process**:
        - Iterates through stories and checks each story against all triggers.
        - If a story matches any trigger, it is added to the filtered list and displayed in the GUI.

Main GUI Thread

- **main_thread(master)**: Main function to initialize the GUI and start the process.
    - **Components**:
        - **GUI Elements**: Frame, scrollbar, label (title), text widget (content), and exit button.
        - **Trigger Configuration**: Reads triggers from a file.

- **Update Function**: Periodically fetches and filters news stories.
- **Threading**: Runs the main loop in a separate thread for concurrent execution.

Main Execution

- **Tkinter Main Loop**:
  - Initializes the Tkinter root window.
  - Starts the main thread to handle GUI and news updates.
  - Enters the Tkinter main event loop to keep the application running.

This detailed breakdown covers the main components and functionality of the code, explaining how it fetches, filters, and displays news stories using various triggers and a graphical user interface.

Code:

```
import feedparser

import string

import time

import threading

from project_util import translate_html

from mtTkinter import *

from datetime import datetime

import pytz



def process(url):

    """

    Fetches news items from the rss url and parses them.

    Returns a list of NewsStory-s.

    """

    feed = feedparser.parse(url)

    entries = feed.entries

    ret = []

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

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

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

# Problem 1

# TODO: NewsStory

class NewsStory:
    def __init__(self, guid, title, description, link, pubdate):
        """
        Initialize a NewsStory object.

        Args:
        guid (str): Globally unique identifier for the news story.
```

```
        title (str): Title of the news story.
        description (str): Description of the news story.
        link (str): URL link to the news story.
        pubdate (datetime): Publication date of the news story.
        """
        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

    def get_name(self):
        return self.__class__.__name__



# Problem 2
# TODO: PhraseTrigger
#============================================================================


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

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


# Problem 4
# TODO: DescriptionTrigger

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

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

    def __init__(self, time_string):

        est = pytz.timezone("EST")

        self.time = est.localize(datetime.strptime(time_string, "%d %b %Y %H:%M:%S"))

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







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



class BeforeTrigger(TimeTrigger):

    def evaluate(self, story):

        return story.get_pubdate().replace(tzinfo=pytz.timezone("EST")) < self.time



class AfterTrigger(TimeTrigger):

    def evaluate(self, story):

        return story.get_pubdate().replace(tzinfo=pytz.timezone("EST")) > 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)

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



#======================
# User-Specified Triggers
#======================
# Problem 11
#======================================================================


def read_trigger_config(filename):
    trigger_file = open(filename, 'r')
    lines = []
    for line in trigger_file:
        line = line.rstrip()
        if not (len(line) == 0 or line.startswith('//') or line.startswith('#')):
            lines.append(line)

    triggers = {}
    trigger_list = []

    for line in lines:
        parts = line.split(',')
        if len(parts) < 2:
            print(f"Invalid line (too short): {line}")
            continue

        trigger_name = parts[0].strip()
        trigger_type = parts[1].strip()
```

```
if trigger_type == 'TitleTrigger':
    if len(parts) != 3:
        print(f"Invalid line (incorrect number of arguments for TitleTrigger): {line}")
        continue
    triggers[trigger_name] = TitleTrigger(parts[2].strip())
    print(f"Created TitleTrigger: {trigger_name} with phrase {parts[2].strip()}")

elif trigger_type == 'DescriptionTrigger':
    if len(parts) != 3:
        print(f"Invalid line (incorrect number of arguments for DescriptionTrigger): {line}")
        continue
    triggers[trigger_name] = DescriptionTrigger(parts[2].strip())
    print(f"Created DescriptionTrigger: {trigger_name} with phrase {parts[2].strip()}")

elif trigger_type == 'BeforeTrigger':
    if len(parts) != 3:
        print(f"Invalid line (incorrect number of arguments for BeforeTrigger): {line}")
        continue
    triggers[trigger_name] = BeforeTrigger(parts[2].strip())
    print(f"Created BeforeTrigger: {trigger_name} with date {parts[2].strip()}")

elif trigger_type == 'AfterTrigger':
    if len(parts) != 3:
        print(f"Invalid line (incorrect number of arguments for AfterTrigger): {line}")
        continue
    triggers[trigger_name] = AfterTrigger(parts[2].strip())
    print(f"Created AfterTrigger: {trigger_name} with date {parts[2].strip()}")

elif trigger_type == 'NotTrigger':
```

```
    if len(parts) != 3:
        print(f"Invalid line (incorrect number of arguments for NotTrigger): {line}")
        continue
    if parts[2].strip() not in triggers:
        print(f"Invalid trigger name for NotTrigger: {parts[2].strip()}")
        continue
    triggers[trigger_name] = NotTrigger(triggers[parts[2].strip()])
    print(f"Created NotTrigger: {trigger_name} negating {parts[2].strip()}")

elif trigger_type == 'AndTrigger':
    if len(parts) != 4:
        print(f"Invalid line (incorrect number of arguments for AndTrigger): {line}")
        continue
    if parts[2].strip() not in triggers or parts[3].strip() not in triggers:
        print(f"Invalid trigger names for AndTrigger: {parts[2].strip()}, {parts[3].strip()}")
        continue
    triggers[trigger_name] = AndTrigger(triggers[parts[2].strip()], triggers[parts[3].strip()])
    print(f"Created AndTrigger: {trigger_name} combining {parts[2].strip()} and {parts[3].strip()}")

elif trigger_type == 'OrTrigger':
    if len(parts) != 4:
        print(f"Invalid line (incorrect number of arguments for OrTrigger): {line}")
        continue
    if parts[2].strip() not in triggers or parts[3].strip() not in triggers:
        print(f"Invalid trigger names for OrTrigger: {parts[2].strip()}, {parts[3].strip()}")
        continue
    triggers[trigger_name] = OrTrigger(triggers[parts[2].strip()], triggers[parts[3].strip()])
    print(f"Created OrTrigger: {trigger_name} combining {parts[2].strip()} and {parts[3].strip()}")

elif trigger_type == 'ADD':
```

```
            for name in parts[2:]:
                if name.strip() not in triggers:
                    print(f"Invalid trigger name in ADD: {name.strip()}")
                else:
                    trigger_list.append(triggers[name.strip()])
                    print(f"Added trigger to list: {name.strip()}")

    return trigger_list

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



def filter_stories(stories, triggerlist, cont):
    filtered_stories = []
    for story in stories:
        cont.insert(END, f"Checking story: {story.get_title()}\n")
        for trigger in triggerlist:
            cont.insert(END, f"  Against trigger: {trigger.get_name()}\n")
            if trigger.evaluate(story):
                cont.insert(END, f"    Trigger {trigger.get_name()} matched!\n")
                filtered_stories.append(story)
                get_cont(story, cont)
                break
    return filtered_stories


def get_cont(newstory, cont):
    cont.insert(END, f"{newstory.get_title()}\n", "title")
    cont.insert(END, f"Published at: {newstory.get_pubdate().strftime('%Y-%m-%d %H:%M:%S')}\n\n")
    cont.insert(END, f"{newstory.get_description()}\n")
    cont.insert(END, f"{newstory.get_link()}\n\n")
```

```
    cont.insert(END, "------------------------------------------------------------\n", "title")


def main_thread(master):
    try:
        triggerlist = read_trigger_config('triggers.txt')
        SLEEPTIME = 120

        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)
        button = Button(frame, text="Exit", command=root.destroy)
        button.pack(side=BOTTOM)
        guidShown = []

        def update_gui():
            stories = process("http://news.google.com/news?output=rss")
            filtered_stories = filter_stories(stories, triggerlist, cont)
            for story in filtered_stories:
                get_cont(story, cont)
            master.after(SLEEPTIME * 1000, update_gui)
```

```
            update_gui()


        except Exception as e:
            cont.insert(END, f"Error: {e}\n")


if __name__ == '__main__':
    root = Tk()
    root.title("Some RSS parser")
    thread = threading.Thread(target=main_thread, args=(root,))
    thread.start()
    root.mainloop()
```