Fundamentals of Programming II

By Saqib Nizar

OOP Project : Report

Made by :

Name	CMS
Subhan Bin Yousaf	481281
Muhammad Ali Shahzadah	466353

Process (url) function:

```
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
```

The function `process(url)`is in charge of obtaining news items from a specified RSS feed URL. It parses the XML data from the RSS feed using the `feedparser` package. It collects important data, including the GUID (a unique identity), title, description, link, and publication date, for every news item in the feed. The `NewsStory` class, a custom class intended to represent a single news story, is then used to encapsulate these details. Ultimately, a list of these {NewsStory} instances is returned by the function, enabling additional processing or analysis.

NewsStory class :

```
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
```

A single news story is shown by the `NewsStory` class. It acts as a storage space for different information related to a news item, such as the GUID, publication date, description, link, and title. With data encapsulation in mind, the class offers methods to access each of these attributes separately, including `get_guid()}, `get_title()}, `get_description()}, `get_link()}, and `get_pubdate()}. This encapsulation promotes code readability and maintainability by guaranteeing that the internal state of a NewsStory instance is secured and may only be accessed or modified through predefined methods.

Trigger Class :

```
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
```

All trigger types in the system use the {Trigger} class as their base class. It specifies `evaluate()` as a common interface function that subclasses need to implement. This function uses the criteria specified by the particular trigger subclass to determine whether a given news story should cause an alert. The `Trigger` class permits polymorphic behavior, which permits the system to employ several trigger types interchangeably by offering a consistent interface.

PhraseTrigger Class :

```
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
```

A subclass of `Trigger', the `PhraseTrigger` class focuses on creating triggers using particular phrases that appear in news articles. It has a predefined phrase for initialization and has a function called `is_phrase_in()', to determine whether the phrase appears in a supplied text. This class allows triggers to be defined based on news story text, and it is the fundamental building block for more specialized trigger subclasses like `TitleTrigger` and `DescriptionTrigger'.

TitleTrigger and DescriptionTrigger Classes :

```
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())
```

Specific terms present in news titles and descriptions are the triggers activated by these subclasses of 'PhraseTrigger'. The `evaluate()` method is implemented in a way that is more specific to each situation, yet the functionality of `PhraseTrigger` is retained. For instance, while `DescriptionTrigger` concentrates on trigger phrases found in descriptions, `TitleTrigger` concentrates on those found in news titles.

TimeTrigger Class :

```
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))
```

Time-based triggers, such as those that are activated before or after a specified time, are represented by the `TimeTrigger` class. After initializing with a time string, it transforms it into a `datetime` object by adding timezone details. With the help of this class, triggers can be constructed according to temporal criteria. For example, an alert can be set to go off if a news piece is published before or after a specific date and time.

BeforeTrigger and AfterTrigger Classes :

```
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
```

These `TimeTrigger’, subclasses focus on triggers that are activated either before or after a given time. To compare the news story's publishing time with the trigger time, they take over the `evaluate` method's functionality from `TimeTrigger`. This makes it possible to construct triggers according to temporal correlations, such sending out an alert in the event that a news article is published earlier or later.

NotTrigger, AndTrigger, and OrTrigger functions :

```
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)
```

These classes stand for composite triggers, which use logical operators like NOT, AND, and OR to combine numerous triggers. The `evaluate` method is implemented by each subclass, which specifies how constituent triggers are combined and assessed to decide if a news story should cause an alert. By combining basic triggers in different ways, composite triggers allow more complicated trigger conditions to be defined.

Filter_stories function:

```
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
```

Using a list of trigger objects as input, the `filter_stories` function filters a list of news stories. Every news story is iterated through to see if any of the triggers in the trigger list evaluate to 'True' for that particular story. A filtered list is returned when stories that meet at least one trigger criteria are added to it. With the help of this feature, users can filter news stories according to particular parameters, including text content or publication date, by applying a series of specified triggers.

read_trigger_config function:

```
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
```

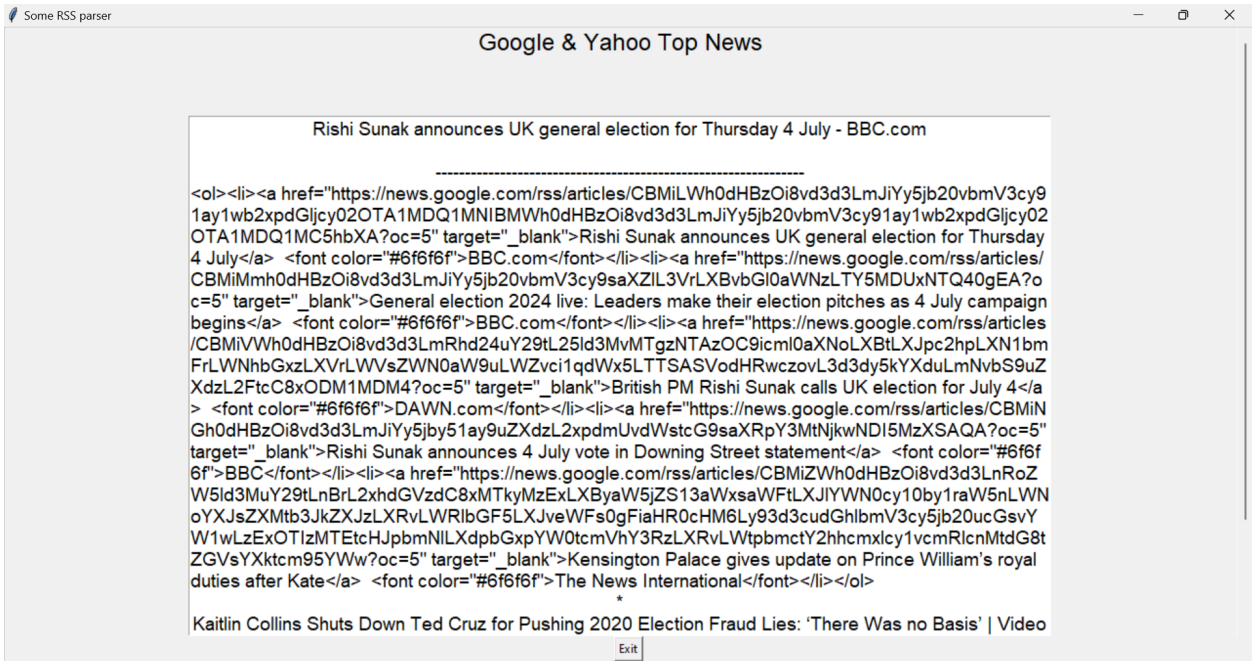
After reading a trigger configuration file, the `read_trigger_config` function builds trigger objects using the given triggers and their arguments. It generates trigger instances for every stated trigger after parsing the configuration file and returns a list of these trigger objects. Greater flexibility and customization of trigger behavior are made possible by this method, which lets users define bespoke trigger setups outside of the main code without changing the fundamental implementation.

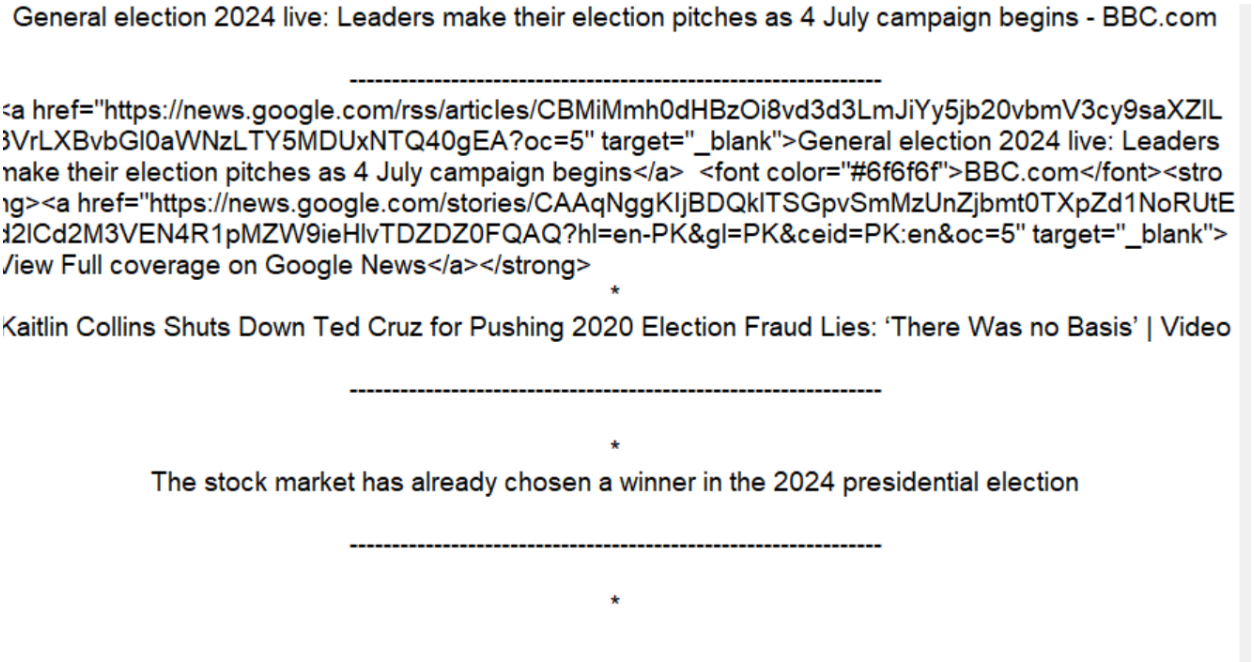
main_thread function:

```
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:
```

The program's primary execution is managed by the `main_thread` function, which is in charge of gathering, sorting, and presenting news articles. It uses the `mtTkinter` package to set up a graphical user interface (GUI) and polls RSS feeds continuously for fresh content. It filters news stories by using the selected triggers, and then shows the filtered stories in the GUI. It keeps the GUI responsive by running endlessly in a separate thread and updating it with fresh stories. This function coordinates the many parts of the program and makes sure it runs well by acting as its central control hub.

End Result :


Some RSS parser
Google & Yahoo Top News
Rishi Sunak announces UK general election for Thursday 4 July - BBC.com
---
Rishi Sunak announces UK general election for Thursday 4 July BBC.com
General election 2024 live: Leaders make their election pitches as 4 July campaign begins BBC.com
British PM Rishi Sunak calls UK election for July 4 DAWN.com
Rishi Sunak announces 4 July vote in Downing Street statement BBC
Kensington Palace gives update on Prince William's royal duties after Kate The News International
*
Kaitlin Collins Shuts Down Ted Cruz for Pushing 2020 Election Fraud Lies: 'There Was no Basis' | Video
Exit


General election 2024 live: Leaders make their election pitches as 4 July campaign begins - BBC.com
---
General election 2024 live: Leaders make their election pitches as 4 July campaign begins BBC.comView Full coverage on Google News
*
Kaitlin Collins Shuts Down Ted Cruz for Pushing 2020 Election Fraud Lies: 'There Was no Basis' | Video
---
*
The stock market has already chosen a winner in the 2024 presidential election
---
*