UNIVERSITY OF
WEST LONDON
The Career University

Applied Software Engineering

Coursework

Assignment – 2

Name: Aman Adish
Student ID: 21587065
Batch: L6- Batch 2

Contents

Task 1 – Implementation

This task involves developing a fully functional software solution for the "Advertisement Approval" use case defined in Assignment 1. It also demonstrates the implementation and utility of Object-Oriented Principles in the software. The system is designed to clearly illustrate the interactions between the Primary and Secondary Actors throughout the task flow.

I have used **Python Spyder** for the implementation of this software.

Use Case Model: Advertisement Approval

The Marketing Staff forwards an advertisement with all the details to the Editor for review. This makes sure the advertisement is ready for publication.

Primary Actor: Marketing Staff

Secondary Actor: Editor

Software Workflow

- Marketing Staff logs into the system.
- The system displays the list of recorded advertisements.
- Marketing Staff selects an adds Advertisement Details to forward to the Editor and validates all the details.
- Editor Logs into the system.
- The Editor approves/rejects the Advertisement.
- The system notifies the Marketing Staff that the submission was successful.
- The system can now track the status of advertisement.

Key Classes as per Assignment 1

- Marketing Staff
- Editor
- Advertisement
- System

Core Classes and Methods

This software implements the key classes from Assignment 1, in the following ways:

MarketingStaff Class

Made as child-class of StaffMember representing marketing personnel. They can:

- *record_ad_details()* - record advertisement details using commands
- *send_ad_to_editor ()* - Send validated advertisements to the editor for review

```
class MarketingStaff(StaffMember):
    def __init__(self, staff_id, name, department_id):
        super().__init__(staff_id, name, department_id)
# Records Ad details
    def record_ad_details(self, system, advertisement):
        command = AddAdvertisementCommand(system, advertisement)
        command.execute()
# Sending Ad to Editor
    def send_ad_to_editor(self, system, advertisement):
        command = ForwardAdvertisementCommand(system, advertisement)
        command.execute()
```

Editor Class

Made as a subclass of StaffMember representing editors who review advertisements. They can:

- *review_ad()* - approve, reject, or request revisions for ads in their queue.
- Notify users about the review decisions.

```
class Editor(StaffMember):
    def __init__(self, staff_id, name, department_id):
        super().__init__(staff_id, name, department_id)

# Reviewing Ad send by editor
# NOTE: Only Ads send by editor can be reviewd
    def review_ad(self, system, advertisement):
        if advertisement in system.editor_queue:
            decision = input(
```

```
                f"Approve, Reject, or Request Revision for advertisement
'{advertisement.advertisement_id}'? (approve/reject/revision): "
            ).lower()

            if decision == "approve":
                advertisement.set_status("Approved")
                system.notify_users(
                    f"The Advertisement with the ID number:
{advertisement.advertisement_id} is approved."
                )

            elif decision == "reject":
                advertisement.set_status("Rejected")
                system.notify_users(
                    f"The Advertisement with the ID number:
{advertisement.advertisement_id} is rejected."
                )

            elif decision == "revision":
                advertisement.set_status("Needs Revision")
                system.notify_users(
                    f"Advertisement with ID
{advertisement.advertisement_id} sent to Marketing Staff for revision."
                )

            else:
                print("Invalid decision.")
        else:
            print("Advertisement not in editor's queue.")
```

Advertisement Class

Represents an individual advertisement with properties like ID, advertiser details, size, publication date, placement, and status. It includes a set_status() method to update its current status (e.g., Approved, Rejected, Needs Revision). Additionally, it stores an optional file path for uploaded advertisement files.

```
class Advertisement:
    def __init__(self, advertisement_id, advertiser_details, size,
date_of_publication, placement):
        self.advertisement_id = advertisement_id #Can only be Int
        self.advertiser_details = advertiser_details
```

```
        self.size = size #Can only be Int
        self.date_of_publication = date_of_publication
        self.placement = placement
        self.status = "Pending" #Default
        self.file = None

    def set_status(self, status):
        self.status = status # Update setter

# How to upload Ad files in process ?
def upload_advertisement_file():
    # Create and configure the Tkinter root
    root = Tk()
    root.withdraw()  # Hide the root window

    try:
        # Open file dialog
        file_path = filedialog.askopenfilename(
            title="Select an Advertisement File",
            filetypes=[("All Files", "*.*"), ("Text Files", "*.txt"),
("Images", "*.jpg *.png")]
        )

        if file_path:
            print(f"File '{file_path}' uploaded successfully.")
            return file_path
        else:
            print("No file selected.")
            return None
    except Exception as e:
        print(f"An error occurred during file upload: {e}")
        return None
    finally:
        root.destroy()  # Ensure the Tkinter instance is properly closed
```

System Class

Manages the overall functionality, including user login/registration, storing advertisements, and interaction between Marketing Staff and Editors. It provides methods to record advertisement details, validate and forward ads, track ad statuses, and manage a queue for editors. It also sends notifications to users about the status of advertisements.

```
class System: # manages users, advertisements, and interactions between
them
    def __init__(self):
        self.logged_in_users = []
        self.advertisements = []  # List of advertisements
        self.editor_queue = []  # Editor's queue for pending
advertisements
        self.users = {"editor1": "editor_password", "marketing1":
"marketing_password"}

        # Preload advertisements (stored but not displayed until
requested)
        self.preload_advertisements()

    # Preload a few advertisements into the system
    def preload_advertisements(self):
        preloaded_ads = [
            Advertisement("100", "Advertiser West London ", 500,
"01/01/2024", "Homepage"),
            Advertisement("101", "Advertiser Stirling", 300, "15/01/2024",
"Sidebar"),
            Advertisement("102", "Advertiser Bath Spa", 400, "10/02/2024",
"Footer"),
            Advertisement("103", "Advertiser Wollonglong", 600,
"20/03/2024", "Homepage"),
        ]
        for ad in preloaded_ads:
            ad.set_status("Approved")  # Set default status
            self.advertisements.append(ad)
```

def login_user ():

This method handles the login process for users. It checks the provided username and password against the system's stored credentials. If the credentials are valid, the user is logged in successfully; otherwise, an error message is displayed.

```
 # Handle user login
    def login_user(self, username, password):
        if username in self.users and self.users[username] == password:
            print(f"Login successful for {username}!")
            return username
        else:
            print("Login failed. Invalid username or password.")
            return None
```

```
    # Register a new user
    def register_user(self, username, password):
        if username in self.users:
            print(f"Username '{username}' already exists. Please choose a
different username.")
        else:
            self.users[username] = password
            print(f"User '{username}' registered successfully!")
```

def record_ad_details():

This method records the details of a new advertisement. It ensures that the advertisement ID is unique before adding it to the system's list of advertisements. If a duplicate ID is found, an error message is displayed.

```
    # Record advertisement details in the system
    def record_ad_details(self, advertisement):
        # Check for duplicate advertisement IDs
        if any(ad.advertisement_id == advertisement.advertisement_id for
ad in self.advertisements):
            print(f"Advertisement ID '{advertisement.advertisement_id}'
already exists. Please choose a different ID.")
        else:
            self.advertisements.append(advertisement)
            print(f"Advertisement '{advertisement.advertisement_id}'
recorded successfully.")
```

def send_advertisement_to_editor():

This method forwards a validated advertisement to the editor's queue for review. It checks for queue capacity and ensures the advertisement is not already in the queue before submission. If successful, the advertisement's status is updated, and notifications are sent.

```
 # Send advertisement to the editor's queue
    def send_advertisement_to_editor(self, advertisement):
        if self.validate_ad_submission(advertisement):
            # Check for duplicates before adding to editor's queue
            if advertisement in self.editor_queue:
                print(f"Advertisement '{advertisement.advertisement_id}'
already in editor's queue.")
```

```
            elif len(self.editor_queue) >= 3:
                print("Editor queue is full. Cannot add new
advertisements.")
            else:
                self.editor_queue.append(advertisement)
                advertisement.set_status("Submitted")
                self.notify_users(f"Advertisement
'{advertisement.advertisement_id}' has been forwarded to the Editor.")
        else:
            print("Failed to forward advertisement.")
```

def track_ad_status():

This method retrieves and displays the current status of a specific advertisement. It helps users keep track of their advertisements' progress through the system.

```
    # Display advertisement status
    def track_ad_status(self, advertisement):
        print(f"Advertisement '{advertisement.advertisement_id}' status:
{advertisement.status}")


# Method to get advertisements in "Needs Revision" status
    def get_needs_revision_ads(self):
        return [ad for ad in self.advertisements if ad.status == "Needs
Revision"]

    # Send notifications to users
    def notify_users(self, message):
        print(f"Notification: {message}")

    # Validate an advertisement before submission
    def validate_ad_submission(self, advertisement):
        if not advertisement.advertiser_details:
            print("Invalid advertiser details.")
            return False
        if advertisement.size <= 0:
            print("Advertisement size must be greater than 0.")
            return False
        if not advertisement.placement:
            print("Advertisement placement is invalid.")
            return False
        return True
```

```
# View all recorded advertisements (called only when selected)
    def view_recorded_advertisements(self):
        if not self.advertisements:
            print("No advertisements have been recorded yet.")
        else:
            print("List of recorded advertisements:")
            for ad in self.advertisements:
                print(f"Advertisement ID: {ad.advertisement_id}, Status:
{ad.status}")
```

Integration of Object Oriented Programming Principles

This software implements the OOP Principles seamlessly in the following ways:

Aggregation

The relationship between **Marketing Staff** and **Advertisement** is an **aggregation**, as advertisements are associated with marketing staff but do not depend on them for their existence. Even if the Marketing Staff object is deleted, the Advertisement objects can remain part of the system.

Encapsulation

Attributes like **updateContent() in the Advertisement class** and **trackAdStatus() in Marketing Staff** are kept private. Direct access to these attributes is restricted, and controlled. This prevents unintended modifications, ensures data integrity, and seamless flow of software.

```
        self.advertisement_id = advertisement_id #Can only be Int
        self.status = "Pending" #Default
        self.file = None

    def set_status(self, status):
        self.status = status # Update setter
    def get_status(self):  # Controlled access
        return self.__status
```

Inheritance

The **Editor and Marketing Staff classes inherit attributes** (like id, name, and departmentID) from the parent class Staff Member. This **reduces redundancy**, as shared attributes and methods are defined in the parent class and reused in child classes.

```
# StaffMember base class
class StaffMember:
    def __init__(self, staff_id, name, department_id):
        self.staff_id = staff_id
        self.name = name
        self.department_id = department_id

class MarketingStaff(StaffMember):
    def __init__(self, staff_id, name, department_id):
        super().__init__(staff_id, name, department_id)

# Rest of the code…

class Editor(StaffMember):
    def __init__(self, staff_id, name, department_id):
        super().__init__(staff_id, name, department_id)
```

Composition

The **System** class has a **composition** relationship with **Advertisement** objects. This means the **Advertisement** objects are tightly bound to the lifecycle of the **System**. If the **System** is destroyed, all associated advertisements will also cease to exist.

NOTE: Here, the advertisements as a list is meant to be seen as the whole advertisement itself.

```
class System: # manages users, advertisements, and interactions between
them
    def __init__(self):
        self.logged_in_users = []
        self.advertisements = []  # List of advertisements

# Rest of the code…

    def record_ad_details(self, advertisement):
        # Check for duplicate advertisement IDs
```

```
        if any(ad.advertisement_id == advertisement.advertisement_id for
ad in self.advertisements):
```

Polymorphism

The Staff Member class defines general methods, but both Editor and Marketing Staff override this method with their specific implementations and functions unique to them, ensuring the behavior aligns with their respective responsibilities.

Abstraction

Complex processes are abstracted into methods such as **record_ad_details()** like how the method **hides the internal storage details** (e.g., using a list to store advertisements) and provides a simple interface for adding advertisements. Or, **notify_users(), which hides how notifications are delivered** (e.g., SMS output, email, etc.) and provides a simple interface for sending them

System Workflow

```
def main():
    system = System()

    while True:
        role = input("Welcome!\nPlease select your department.\nDo you
belong to Marketing or Editorial Department? (M/E): ").lower()

        while True:
            action = input("Please Register, Log In or Exit System\n*
Register (R)\n* Login (L)\n* Exit (E)? ").lower()
            if action == "l":
                username = input("Enter Username: ")
                password = input("Enter Password: ")
                user = system.login_user(username, password)
                if user:
                    break
            elif action == "r":
                username = input("Enter Username for Registration: ")
                password = input("Enter Password for Registration: ")
                system.register_user(username, password)
            elif action == "e":
                print("Exiting system.")
                return
            else:
                print("Invalid action.")
```

```
        if role == "m":
            marketing_staff = MarketingStaff(1, username, "Marketing")

            # Marketing staff must be logged in before performing actions
            if not user:
                print("You must log in to perform any operation.")
                continue

            choice = input(
                "Would you like to \n(1) Record Advertisement Details
\n(2) View Recorded Advertisements \n(3) Needs Revision? (1/2/3): "
            )
            if choice == "1":
                ad_id = input("Enter Advertisement ID: ")
                advertiser_details = input("Enter Advertiser details: ")

                # Validate size input
                while True:
                    try:
                        size = int(input("Enter Advertisement size: "))
                        break  # Exit loop if valid integer
                    except ValueError:
                        print("Invalid input. Please enter a valid number
for advertisement size.")

                date_of_publication = input("Enter Date of Publication: ")
                placement = input("Enter Placement details: ")

                ad = Advertisement(ad_id, advertiser_details, size,
date_of_publication, placement)
                marketing_staff.record_ad_details(system, ad)

                ad.file = upload_advertisement_file()

                choice = input("a) Validate Advertisement \nb) Send to
Editor \nc) Needs Revision: ").lower()

                if choice == "a":
                    if system.validate_ad_submission(ad):
                        system.notify_users(f"Advertisement
'{ad.advertisement_id}' validated.")
                        # Display advertisement details after file upload
                        print(f"Advertisement ID: {ad.advertisement_id}")
```

```
                        print(f"Advertiser Details:
{ad.advertiser_details}")
                        print(f"Size: {ad.size}")
                        print(f"Date of Publication:
{ad.date_of_publication}")
                        print(f"Placement: {ad.placement}")
                        print(f"Status: {ad.status}")
                        marketing_staff.send_ad_to_editor(system, ad)

                elif choice == "b":
                    marketing_staff.send_ad_to_editor(system, ad)

                elif choice == "c":
                    system.notify_users(f"Advertisement
'{ad.advertisement_id}' needs revision.")
            elif choice == "2":
                system.view_recorded_advertisements()
            elif choice == "3":
                needs_revision_ads = system.get_needs_revision_ads()
                if not needs_revision_ads:
                    print("No advertisements currently need revision.")
                else:
                    print("Advertisements that need revision:")
                    for ad in needs_revision_ads:
                        print(f"Advertisement ID: {ad.advertisement_id},
Status: {ad.status}")
        elif role == "e":
            editor = Editor(1, username, "Editorial")

            advertisement_id = input("Enter advertisement ID to review: ")
            for ad in system.editor_queue:
                if ad.advertisement_id == advertisement_id:
                    editor.review_ad(system, ad)

if __name__ == "__main__":
    main()
```

Output

Approved Advertisement with Tracking Status

```
Welcome!
Please select your department.
Do you belong to Marketing or Editorial Department? (M/E): M

Please Register, Log In or Exit System
* Register (R)
* Login (L)
* Exit (E)? R

Enter Username for Registration: Mar_example

Enter Password for Registration: Mar_pass
User 'Mar_example' registered successfully!

Please Register, Log In or Exit System
* Register (R)
* Login (L)
* Exit (E)? L

Enter Username: Mar_example

Enter Password: Mar_pass
Login successful for Mar_example!

Would you like to
(1) Record Advertisement Details
(2) View Recorded Advertisements
(3) Needs Revision? (1/2/3): 1

Enter Advertisement ID: 555

Enter Advertiser details: Sample Ad

Enter Advertisement size: 230

Enter Date of Publication: 12/09/2024

Enter Placement details: Header
Advertisement '555' recorded successfully.
File 'C:/Users/USER/Downloads/tick.png' uploaded successfully.

a) Validate Advertisement
b) Send to Editor
c) Needs Revision: b
Notification: Advertisement '555' has been forwarded to the Editor.

Welcome!
Please select your department.
Do you belong to Marketing or Editorial Department? (M/E): E
```

```
Please Register, Log In or Exit System
* Register (R)
* Login (L)
* Exit (E)? R

Enter Username for Registration: Edr_example

Enter Password for Registration: Edr_pass
User 'Edr_example' registered successfully!

Please Register, Log In or Exit System
* Register (R)
* Login (L)
* Exit (E)? L

Enter Username: Edr_example

Enter Password: Edr_pass
Login successful for Edr_example!

Enter advertisement ID to review: 555

Approve, Reject, or Request Revision for advertisement '555'? (approve/reject/revision): approve
Notification: The Advertisement with the ID number: 555 is approved.

Welcome!
Please select your department.
Do you belong to Marketing or Editorial Department? (M/E): M

Please Register, Log In or Exit System
* Register (R)
* Login (L)
* Exit (E)? L

Enter Username: Mar_example

Enter Password: Mar_pass
Login successful for Mar_example!
```

```
Would you like to
(1) Record Advertisement Details
(2) View Recorded Advertisements
(3) Needs Revision? (1/2/3): 2
List of recorded advertisements:
Advertisement ID: 100, Status: Approved
Advertisement ID: 101, Status: Approved
Advertisement ID: 102, Status: Approved
Advertisement ID: 103, Status: Approved
Advertisement ID: 555, Status: Approved

Welcome!
Please select your department.
Do you belong to Marketing or Editorial Department? (M/E): M

Please Register, Log In or Exit System
* Register (R)
* Login (L)
* Exit (E)? E
Exiting system.
```

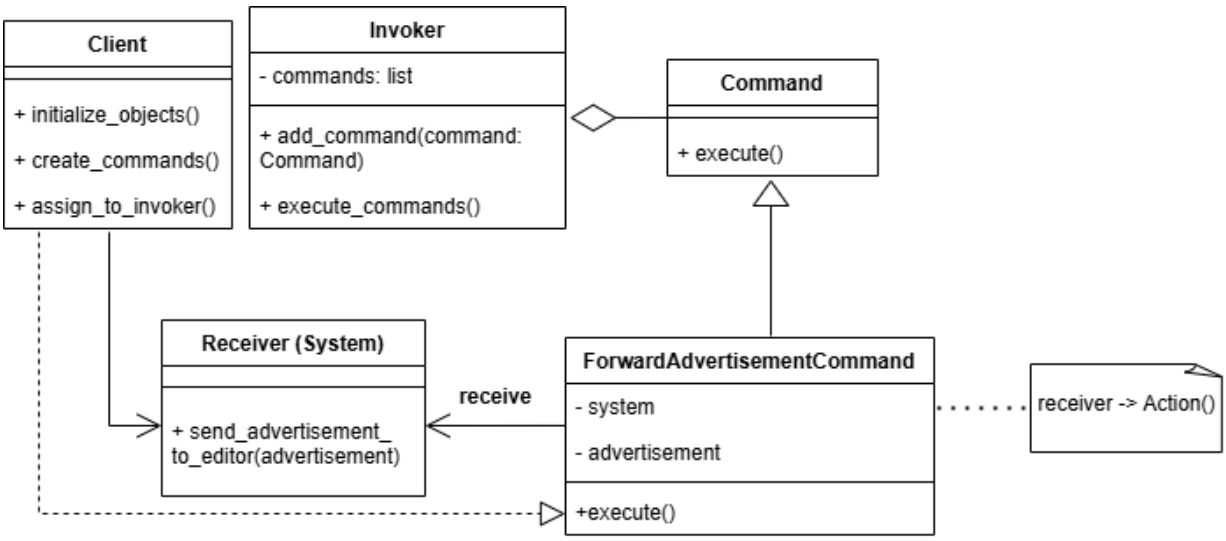
Task 2 – Design Patterns

The Command Pattern is a **behavioral design pattern** that encapsulates a request as an object, thereby allowing users to parameterize clients with queues, requests, and operations. It provides flexibility in how requests are executed.

The Command Pattern focuses on encapsulating actions, not modifying the core objects themselves, so most of the classes such as the MarketingStaff Advertisement and Editor remains unchanged.

There are five parts to the Command Pattern, below is an analysis of how the Command, Concrete Command, Invoker, Client, and Receiver are used in my implementation:

Class Diagram for ForwardAdvertisement Concrete Command



1. Command (Base Command)

The Command class serves as the **Command interface** or **abstract base class** in my implementation. It defines the execute() method, will be overridden by concrete commands to perform specific actions. It establishes a contract for all command classes, ensuring they provide an execute() method.

```
# Command pattern base class
class Command:
    def execute(self):
        raise NotImplementedError
```

2. Concrete Command

The Concrete Commands are classes **AddAdvertisementCommand**, **ForwardAdvertisementCommand**, and **TrackAdvertisementStatusCommand**. These classes implement the execute() method defined in the Command class and encapsulate the specific operations for each action (adding an advertisement, forwarding it to the editor, and tracking its status).

AddAdvertisementCommand - Encapsulates the action of recording advertisement details.

```
class AddAdvertisementCommand(Command):
    def __init__(self, system, advertisement):
        self.system = system
        self.advertisement = advertisement

    def execute(self):
        self.system.record_ad_details(self.advertisement)
```

ForwardAdvertisementCommand - Encapsulates the action of forwarding an advertisement to the editor.

```
class ForwardAdvertisementCommand(Command):
    def __init__(self, system, advertisement):
        self.system = system
        self.advertisement = advertisement

    def execute(self):
        self.system.send_advertisement_to_editor(self.advertisement)
```

TrackAdvertisementStatusCommand - Encapsulates the action of tracking the status of the advertisement.

```
class TrackAdvertisementStatusCommand(Command):
    def __init__(self, system, advertisement):
        self.system = system
        self.advertisement = advertisement

    def execute(self):
        self.system.track_ad_status(self.advertisement)
```

3. Invoker

The Invoker is responsible for executing commands. In this case, this role is played by the part of the code where we instantiate and call the execute() method on the command objects. The Invoker doesn't need to know the concrete command type; it only interacts with the Command interface.

NOTE: Invoker Class not specifically defined as a separate class

```
# Records Ad details
    def record_ad_details(self, system, advertisement):
        command = AddAdvertisementCommand(system, advertisement)
        command.execute()
# Sending Ad to Editor
    def send_ad_to_editor(self, system, advertisement):
        command = ForwardAdvertisementCommand(system, advertisement)
        command.execute()
```

4. Client

The Client is the part of the code that creates the command objects and typically invokes them via the Invoker. In this case, the Client is the section of the code where commands like AddAdvertisementCommand, ForwardAdvertisementCommand, and TrackAdvertisementStatusCommand are created and executed. It is responsible for initiating actions (e.g., recording or forwarding advertisements).

NOTE: Client Class not specifically defined as a separate class, same code as Invoker

5. Receiver

The **Receiver** is responsible for performing the actual work when a command is executed. It contains the business logic that is triggered when the command's execute() method is called. In my system, the **Receiver is the System class**. This class contains all the business logic for managing advertisements, such asRecording advertisement detailsSending the advertisement to the editor's queueTracking advertisement status etc.

Task 3 – OCL Implementation

The following OCL Constraint Rules were considered in my Assignment 1, the implementation is as follows:

Rule 1: Marketing Staff must be verified (logged in) before performing any operations

OCL: *context System::sendAdvertisementToEditor(advertisement : Advertisement) pre: self.currentUser.isLoggedIn() and advertisement.isValid() and not self.editorQueue->includes(advertisement)*

Implementation: In the System class The login_user method ensures that only verified users with valid credentials can log in.If login fails, the user is not allowed to perform any operations.

In the main function: After selecting Marketing, the system checks if the user is logged in using the user variable. If not logged in, operations are restricted.

```
    def login_user(self, username, password):
        if username in self.users and self.users[username] == password:
            print(f"Login successful for {username}!")
            return username
        else:
            print("Login failed. Invalid username or password.")
            return None

# IN MAIN FUNCTION

            # Marketing staff must be logged in before performing actions
            if not user:
                print("You must log in to perform any operation.")
                continue
```

Rule 2: Duplicate advertisements should not be forwarded to the Editor's queue.

OCL: *context Systeminv noDuplicateSubmissions:self.editorQueue.advertisements->forAll(a | a <> advertisement)*

Implementation: In the send_advertisement_to_editor method, the system checks if the advertisement is already in the editor's queue using this condition.

In the record_ad_details method, Before adding a new advertisement, the system verifies if the advertisement ID already exists in the recorded advertisements

```
 # Send advertisement to the editor's queue
    def send_advertisement_to_editor(self, advertisement):
        if self.validate_ad_submission(advertisement):
            # Check for duplicates before adding to editor's queue
            if advertisement in self.editor_queue:
                print(f"Advertisement '{advertisement.advertisement_id}'
already in editor's queue.")
            elif len(self.editor_queue) >= 3:
                print("Editor queue is full. Cannot add new
advertisements.")
```

Rule 3: The Editor's queue must have capacity before accepting new advertisements.

OCL: *context EditorQueue inv queueCapacity: self.advertisements->size() < self.maxCapacity*

Implementation: The system ensures the editor's queue does not exceed a specified capacity (e.g., 3 in this implementation), though it can be changed as per the Admin's will.

```
 # Send advertisement to the editor's queue
    def send_advertisement_to_editor(self, advertisement):
        if self.validate_ad_submission(advertisement):
            # Check for duplicates before adding to editor's queue
            if advertisement in self.editor_queue:
                print(f"Advertisement '{advertisement.advertisement_id}'
already in editor's queue.")
            elif len(self.editor_queue) >= 3:
                print("Editor queue is full. Cannot add new
advertisements.")
```

Alternate Code Flow

1. Editor Queue is Full ERROR

```
Enter Advertisement ID: 666

Enter Advertiser details: sample_editor_queue_full

Enter Advertisement size: 234

Enter Date of Publication: 11/11/2011

Enter Placement details: Footer
Advertisement '666' recorded successfully.
File 'C:/Users/USER/Downloads/Souvenir.png' uploaded successfully.

a) Validate Advertisement
b) Send to Editor
c) Needs Revision: b
Editor queue is full. Cannot add new advertisements.
```

2. Username already registered ERROR

```
Welcome!
Please select your department.
Do you belong to Marketing or Editorial Department? (M/E): M

Please Register, Log In or Exit System
* Register (R)
* Login (L)
* Exit (E)? R

Enter Username for Registration: marketing1

Enter Password for Registration: marketing_example
Username 'marketing1' already exists. Please choose a different username.
```

3. Viewing Prerecorded Advertisement

```
Would you like to
(1) Record Advertisement Details
(2) View Recorded Advertisements
(3) Needs Revision? (1/2/3): 2
List of recorded advertisements:
Advertisement ID: 100, Status: Approved
Advertisement ID: 101, Status: Approved
Advertisement ID: 102, Status: Approved
Advertisement ID: 103, Status: Approved
```

4. Wrong Variable ERROR

```
Enter Advertisement ID: 545

Enter Advertiser details: Sample_wrong_variable

Enter Advertisement size: wrong_variable
Invalid input. Please enter a valid number for advertisement size.

Enter Advertisement size: 230

Enter Date of Publication: 12/12/2012
```

5. Duplicate ID ERROR

```
Enter Advertisement ID: 545

Enter Advertiser details: sample_duplicate_ID

Enter Advertisement size: 123

Enter Date of Publication: 12/12/2024

Enter Placement details: Sidebar
Advertisement ID '545' already exists. Please choose a different ID.
```

6. Rejected Advertisement

```
Enter Password: aman1
Login successful for aman!

Enter advertisement ID to review: 545

Approve, Reject, or Request Revision for advertisement '545'? (approve/reject/revision): reject
Notification: The Advertisement with the ID number: 545 is rejected.

Welcome!
Please select your department.
Do you belong to Marketing or Editorial Department? (M/E): M

Please Register, Log In or Exit System
* Register (R)
* Login (L)
* Exit (E)? L

Enter Username: aman

Enter Password: aman1
Login successful for aman!

Would you like to
(1) Record Advertisement Details
(2) View Recorded Advertisements
(3) Needs Revision? (1/2/3): 2
List of recorded advertisements:
Advertisement ID: 100, Status: Approved
Advertisement ID: 101, Status: Approved
Advertisement ID: 102, Status: Approved
Advertisement ID: 103, Status: Approved
Advertisement ID: 545, Status: Rejected
```

7. Needs Revision

```
Enter advertisement ID to review: 555

Approve, Reject, or Request Revision for advertisement '555'? (approve/reject/revision): revision
Notification: Advertisement with ID 555 sent to Marketing Staff for revision.

Welcome!
Please select your department.
Do you belong to Marketing or Editorial Department? (M/E): M

Please Register, Log In or Exit System
* Register (R)
* Login (L)
* Exit (E)? L

Enter Username: aman

Enter Password: aman1
Login successful for aman!

Would you like to
(1) Record Advertisement Details
(2) View Recorded Advertisements
(3) Needs Revision? (1/2/3): 3
Advertisements that need revision:
Advertisement ID: 555, Status: Needs Revision
```

Task 4 – Software Testing and Verification Considerations

Unit testing focuses on testing the smallest, individual components (or units) of a software application in isolation. The goal is to ensure that each unit works as expected.

Test Driven Development

TDD is a software development approach where tests are written before the actual code is implemented. This methodology ensures the code meets the test criteria right from the start.

My code has utilized and approached this project with the TDD method and has achieved significant results, it used the **unittest Module** in Python to facilitate the testing process:

```
import unittest
from io import StringIO
from unittest.mock import patch
from ASE import System, Advertisement, MarketingStaff, Editor
```

This imports the unittest module and the relevant classes (System, Advertisement, MarketingStaff, and Editor) from my saved file with full implementation code named ASE.

```
    def setUp(self):
        """Set up the system and create necessary objects."""
        self.system = System()
        self.advertisement = Advertisement("104", "Advertiser Test", 250,
"01/04/2024", "Sidebar")
        self.marketing_staff = MarketingStaff(1, "MarketingUser",
"Marketing")
        self.editor = Editor(2, "EditorUser", "Editorial")
```

The setUp method initializes the testing environment by creating instances of System, Advertisement, MarketingStaff, and Editor objects, ensuring a consistent setup for each test case.

The other tests are marked with a symbol, test_ to help identify and easily understand helping in code readability:

Full Software Testing Code

```
import unittest
from io import StringIO
from unittest.mock import patch
from ASE import System, Advertisement, MarketingStaff, Editor

class TestSystem(unittest.TestCase):

    def setUp(self):
        """Set up the system and create necessary objects."""
        self.system = System()
        self.advertisement = Advertisement("104", "Advertiser Test", 250,
"01/04/2024", "Sidebar")
        self.marketing_staff = MarketingStaff(1, "MarketingUser",
"Marketing")
        self.editor = Editor(2, "EditorUser", "Editorial")

    def test_login_success(self):
        """Test login with correct credentials."""
        user = self.system.login_user("marketing1", "marketing_password")
        self.assertEqual(user, "marketing1")

    def test_login_failure(self):
        """Test login with incorrect credentials."""
        user = self.system.login_user("wrong_user", "wrong_password")
        self.assertIsNone(user)

    def test_register_user(self):
        """Test registering a new user."""
        self.system.register_user("new_user", "new_password")
        user = self.system.login_user("new_user", "new_password")
        self.assertEqual(user, "new_user")

    def test_record_ad_details(self):
        """Test recording advertisement details."""
        self.system.record_ad_details(self.advertisement)
        self.assertIn(self.advertisement, self.system.advertisements)

    def test_duplicate_advertisement_id(self):
        """Test handling of duplicate advertisement IDs."""
```

```
        duplicate_ad = Advertisement("100", "Advertiser Duplicate", 300,
"02/05/2024", "Footer")
        self.system.record_ad_details(duplicate_ad)
        with patch('sys.stdout', new_callable=StringIO) as mock_stdout:
            self.system.record_ad_details(duplicate_ad)
            self.assertIn("Advertisement ID '100' already exists",
mock_stdout.getvalue())

    def test_send_ad_to_editor(self):
        """Test sending advertisement to the editor's queue."""
        self.system.record_ad_details(self.advertisement)
        self.marketing_staff.send_ad_to_editor(self.system,
self.advertisement)
        self.assertIn(self.advertisement, self.system.editor_queue)

    def test_send_ad_to_editor_full_queue(self):
        """Test sending advertisement when the editor's queue is full."""
        self.system.editor_queue = [self.advertisement] * 3  # Simulate
full queue
        with patch('sys.stdout', new_callable=StringIO) as mock_stdout:
            self.marketing_staff.send_ad_to_editor(self.system,
self.advertisement)
            self.assertIn("Editor queue is full", mock_stdout.getvalue())

    def test_track_ad_status(self):
        """Test tracking advertisement status."""
        self.system.record_ad_details(self.advertisement)
        self.system.track_ad_status(self.advertisement)
        with patch('sys.stdout', new_callable=StringIO) as mock_stdout:
            self.system.track_ad_status(self.advertisement)
            self.assertIn("Advertisement '104' status: Pending",
mock_stdout.getvalue())

    def test_validate_ad_submission(self):
        """Test validation of advertisement before submission."""
        self.advertisement.advertiser_details = ""  # Invalid details
        valid = self.system.validate_ad_submission(self.advertisement)
        self.assertFalse(valid)

    def test_advertisement_submission_success(self):
        """Test advertisement submission with valid details."""
        self.advertisement.advertiser_details = "Advertiser Test"
        valid = self.system.validate_ad_submission(self.advertisement)
        self.assertTrue(valid)
```

```
    def test_editor_review_ad_approve(self):
        """Test the editor's decision to approve an advertisement."""
        self.system.record_ad_details(self.advertisement)
        self.marketing_staff.send_ad_to_editor(self.system,
self.advertisement)
        with patch('builtins.input', return_value="approve"):
            with patch('sys.stdout', new_callable=StringIO) as
mock_stdout:
                self.editor.review_ad(self.system, self.advertisement)
                self.assertIn("The Advertisement with the ID number: 104
is approved.", mock_stdout.getvalue())

    def test_editor_review_ad_reject(self):
        """Test the editor's decision to reject an advertisement."""
        self.system.record_ad_details(self.advertisement)
        self.marketing_staff.send_ad_to_editor(self.system,
self.advertisement)
        with patch('builtins.input', return_value="reject"):
            with patch('sys.stdout', new_callable=StringIO) as
mock_stdout:
                self.editor.review_ad(self.system, self.advertisement)
                self.assertIn("The Advertisement with the ID number: 104
is rejected.", mock_stdout.getvalue())

    def test_editor_review_ad_revision(self):
        """Test the editor's decision to request a revision."""
        self.system.record_ad_details(self.advertisement)
        self.marketing_staff.send_ad_to_editor(self.system,
self.advertisement)
        with patch('builtins.input', return_value="revision"):
            with patch('sys.stdout', new_callable=StringIO) as
mock_stdout:
                self.editor.review_ad(self.system, self.advertisement)
                self.assertIn("Advertisement with ID 104 sent to Marketing
Staff for revision.", mock_stdout.getvalue())

    def test_get_needs_revision_ads(self):
        """Test retrieving advertisements that need revision."""
        self.advertisement.set_status("Needs Revision")
        self.system.record_ad_details(self.advertisement)
        ads_needing_revision = self.system.get_needs_revision_ads()
        self.assertIn(self.advertisement, ads_needing_revision)
print("\nTEST CASES")
suite = unittest.TestLoader().loadTestsFromTestCase(TestSystem)
unittest.TextTestRunner(verbosity=2).run(suite)
```

Output Screenshot

```
test_advertisement_submission_success (__main__.TestSystem)
Test advertisement submission with valid details. ... ok
test_duplicate_advertisement_id (__main__.TestSystem)
Test handling of duplicate advertisement IDs. ... ok
test_editor_review_ad_approve (__main__.TestSystem)
Test the editor's decision to approve an advertisement. ... ok
test_editor_review_ad_reject (__main__.TestSystem)
Test the editor's decision to reject an advertisement. ... ok
test_editor_review_ad_revision (__main__.TestSystem)
Test the editor's decision to request a revision. ... ok
test_get_needs_revision_ads (__main__.TestSystem)
Test retrieving advertisements that need revision. ... ok
test_login_failure (__main__.TestSystem)
Test login with incorrect credentials. ... ok
test_login_success (__main__.TestSystem)
Test login with correct credentials. ... ok
test_record_ad_details (__main__.TestSystem)
Test recording advertisement details. ... ok
test_register_user (__main__.TestSystem)
Test registering a new user. ... ok
test_send_ad_to_editor (__main__.TestSystem)
Test sending advertisement to the editor's queue. ... ok
test_send_ad_to_editor_full_queue (__main__.TestSystem)
Test sending advertisement when the editor's queue is full. ... ok
test_track_ad_status (__main__.TestSystem)
Test tracking advertisement status. ... ok
test_validate_ad_submission (__main__.TestSystem)
Test validation of advertisement before submission. ...
----------------------------------------------------------------------
Ran 14 tests in 0.027s

OK
```

Validation Considerations

1. Advertiser Name Validation

In the validate_ad_submission method, it checks if the advertiser_details (the advertiser's name) is present. This ensures that the advertiser_details is not empty, thus ensuring a valid Advertiser Name.

```python
    def validate_ad_submission(self, advertisement):
        if not advertisement.advertiser_details:
            print("Invalid advertiser details.")
            return False
```

2. Advertisement Size Validation

In the validate_ad_submission method, Also there is a check to ensure the size of the advertisement is greater than 0. This checks that the size of the advertisement is positive.

```
        if advertisement.size <= 0:
            print("Advertisement size must be greater than 0.")

# Rest of the code…
   # Validate size input
                while True:
                    try:
                        size = int(input("Enter Advertisement size: "))
                        break  # Exit loop if valid integer
                    except ValueError:
                        print("Invalid input. Please enter a valid number
for advertisement size.")
```

3. Advertisement Placement Validation

Another condition within the validate_ad_submission method validates that the placement is not null or undefined.

```
        if not advertisement.placement:
            print("Advertisement placement is invalid.")
            return False
        return True
```

4. File Upload Validation

In the upload_advertisement_file method, it uses the filedialog.askopenfilename() method to open a file dialog and check if a file is selected. This ensures that a file is selected by the user and provides feedback if no file is chosen.

```
       if file_path:
            print(f"File '{file_path}' uploaded successfully.")
            return file_path
        else:
```

```
            print("No file selected.")
            return None
    except Exception as e:
        print(f"An error occurred during file upload: {e}")
        return None
    finally:
        root.destroy()  # Ensure the Tkinter instance is properly closed
```

#Note: The Outputs specifying the proof of Validation Considerations have been mentioned in the OCL Section

Conclusion

The implementation of the **Advertisement Approval** system effectively demonstrates the application of **Object-Oriented Principles**, including aggregation, encapsulation, inheritance, composition, polymorphism, and abstraction. The system ensures seamless interaction between Marketing Staff and Editors while maintaining data integrity and efficient workflow.

Through the use of the **Command Pattern**, the design supports flexibility and scalability, allowing easy addition of new functionalities. The system's validation process guarantees the accuracy of the advertisements, while the overall design adheres to best practices for object-oriented software development.

Test-Driven Development (TDD) was integral to the process, ensuring that each component was **thoroughly tested before implementation**, promoting reliable and bug-free code. Automated unit tests were written for key functionalities to validate correctness, with edge cases covered, allowing for continuous improvements and ensuring robust system performance.

References

GeeksforGeeks. (2018). Command Pattern - GeeksforGeeks. [online] Available at: https://www.geeksforgeeks.org/command-pattern/.

Unadkat, J. (2021). Test Driven Development (TDD) : Approach & Benefits. [online] BrowserStack. Available at: https://www.browserstack.com/guide/what-is-test-driven-development.

Bhakhra, S. (2020). Test Driven Development (TDD). [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/test-driven-development-tdd/.

Java, in (2021). The Command Pattern Explained and Implemented in Java | Behavioral Design Patterns | Geekific. [online] YouTube. Available at: https://youtu.be/UfGD60BYzPM?si=V9EHvamFlc0Mrnnu [Accessed 11 Jan. 2025].

YouTube. (2024). TDD - Test Driven Development (Red | Green | Refactor) | Example | Java Techie. [online] Available at: https://youtu.be/UzRa5cLma0g?si=OGxV1gqbwV4XK4tn [Accessed 11 Jan. 2025].

Blackboard Slides for Applied Software Engineering