**Practical Assignment 3**

**Course name: CST8002 050**

**Professor’s Name: Todd Keuleman**

**Name: Tanaz Imrankhan Pathan**

**Student Number: 041102857**

Table of Contents

[**Evidence of learning** 3](#_Toc193043627)

[**Overview** 3](#_Toc193043628)

[**Inheritance** 3](#_Toc193043629)

[**Polymorphism** 4](#_Toc193043630)

[**Program Demonstration via Screen Shots** 6](#_Toc193043631)

[**Pytest** 12](#_Toc193043632)

[**Source Code** 13](#_Toc193043633)

# **Evidence of learning**

## **Overview**

This project demonstrates the use of advanced programming principles like **inheritance** and **polymorphism** in the context of a structured application. The purpose of these concepts is to create reusable, maintainable, and flexible code. Inheritance allows one class to inherit properties and methods from another, while polymorphism enables a method to behave differently depending on the subclass.

## **Inheritance**

Inheritance in object-oriented programming allows one class (the subclass) to inherit methods and attributes from another class (the superclass). This promotes code reuse and modularity. In our project, **Facility** and **Record** are examples of how we leverage inheritance.

**Facility Class (Superclass)**

The Facility class provides the basic structure for representing a healthcare facility. It contains essential attributes like region, district, license number, name, and contact information. The class also includes a display() method that outputs basic information about the facility.

class Facility:

    def \_\_init\_\_(self, region, district, license\_number, name, type, address\_1, address\_2, address\_3, capacity, occupied, available, staff\_count, language, operator, online\_services):

        self.region = region

        self.district = district

        self.license\_number = license\_number

        self.name = name

        self.type = type

        self.address\_1 = address\_1

        self.address\_2 = address\_2

        self.address\_3 = address\_3

        self.capacity = capacity

        self.occupied = occupied

        self.available = available

        self.staff\_count = staff\_count

        self.language = language

        self.operator = operator

        self.online\_services = online\_services

    def display(self):

        return f"Facility Name: {self.name}, License Number: {self.license\_number}, Region: {self.region}"

**Hospital Class (Subclass)**

The Hospital class extends the Facility class, inheriting all the basic attributes and methods but adding additional hospital-specific attributes and methods. For example, a hospital may have emergency services, departments, and specific health programs.

class Hospital(Facility):

    def \_\_init\_\_(self, region, district, license\_number, name, type, address\_1, address\_2, address\_3, capacity, occupied, available, staff\_count, language, operator, online\_services, emergency\_services, departments):

        super().\_\_init\_\_(region, district, license\_number, name, type, address\_1, address\_2, address\_3, capacity, occupied, available, staff\_count, language, operator, online\_services)

        self.emergency\_services = emergency\_services

        self.departments = departments

    def display(self):

        # Overriding the display method to include emergency services and departments

        base\_info = super().display()

        return f"{base\_info}, Emergency Services: {self.emergency\_services}, Departments: {self.departments}"

In this case, the Hospital class inherits from Facility, reusing the display() method but enhancing it with additional attributes specific to hospitals, like emergency\_services and departments. The display() method is overridden in Hospital to show more detailed information.

**Why Use Inheritance?**

* **Reusability**: The Hospital class reuses the Facility class's methods and attributes, reducing the need for repetitive code.
* **Extendibility**: New facility types (e.g., Clinic, HealthCenter) can be created by extending the Facility class without duplicating the core functionality.

## **Polymorphism**

Polymorphism allows an object of a subclass to be treated as an object of its superclass, enabling method overriding where a subclass provides its own implementation of a method defined in the superclass.

In this project, **polymorphism** is demonstrated with the display() method. While the Facility class defines a basic display() method, the Hospital subclass overrides it to provide more specialized information.

**Polymorphic Behavior in Action**

When an instance of Facility or Hospital calls the display() method, Python uses **method overriding** to decide which version of display() to execute. This is an example of **polymorphism**, where the same method behaves differently depending on the object type.

def show\_facility\_details(facility):

    print(facility.display())

# Create objects

facility = Facility("Region1", "District1", "LN001", "Facility1", "Type1", "Address1", "Address2", "Address3", 100, 10, 20, 30, "English", "Operator1", "Yes")

hospital = Hospital("Region1", "District1", "LN001", "Hospital1", "Type1", "Address1", "Address2", "Address3", 200, 50, 150, 100, "English", "Operator1", "Yes", "Yes", "Emergency, Surgery")

# Calling the same method on different objects

show\_facility\_details(facility)  # Uses Facility's display

show\_facility\_details(hospital)  # Uses Hospital's display

Output:

Facility Name: Facility1, License Number: LN001, Region: Region1

Facility Name: Hospital1, License Number: LN001, Region: Region1, Emergency Services: Yes, Departments: Emergency, Surgery

Here, even though the show\_facility\_details() function expects a Facility object, it can also handle objects of the Hospital subclass due to polymorphism. This is useful for making the code more flexible and general while still allowing specialized behavior in subclasses.

**Why Use Polymorphism?**

* **Flexibility**: By allowing the display() method to behave differently based on the object type, we can write more generic and flexible code that works with both Facility and Hospital objects seamlessly.
* **Maintainability**: If the Facility class needs to be updated, such as changing how data is displayed, these changes will propagate through all subclasses, maintaining consistency.

# **Program Demonstration via Screen Shots**

A computer screen shot of a black background

AI-generated content may be incorrect.

This option allows you to load facility records from an external file (CSV). You will be prompted to provide the file path, and the system will read the data and load it into the system.

A computer screen shot of a computer program

AI-generated content may be incorrect.

This option displays all the facility records that have been loaded into the system. If there are no records, it will let you know.

A computer screen with white text

AI-generated content may be incorrect.

This option allows you to search for and view a specific facility by its **License Number**. You’ll be asked to enter the License Number, and the system will show the relevant facility's details if it’s found.

A screenshot of a computer program

AI-generated content may be incorrect.This option lets you add a new facility. You'll be prompted to input details such as region, district, license number, facility name, etc. The new facility will be added to the system and savedA screenshot of a computer program

AI-generated content may be incorrect.

This option enables you to update an existing facility’s information. You’ll search for the facility by its License Number and then be asked to enter the updated details. After editing, the system will replace the old record with the new one.

A screenshot of a computer program

AI-generated content may be incorrect.

This option allows you to delete a facility record. You’ll be prompted to provide the License Number of the facility you wish to delete, and once found, the record will be removed from the system.

A screen shot of a computer

AI-generated content may be incorrect.

After adding, editing, or deleting records, this option allows you to save the current facility records to a new file. The system generates a new filename (UUID-based) and writes the updated data to a CSV file.

A black screen with white text

AI-generated content may be incorrect.

This option exits the system, ending the program. It allows you to safely close the application.

# **Pytest**

import unittest  # Importing the unittest module to create and run tests

from model.facility import Facility  # Importing the Facility class from the model.facility module

from model.record import Record  # Importing the Record class from the model.record module (though it's not used in this test)

class TestPolymorphism(unittest.TestCase):  # Define a test case class inheriting from unittest.TestCase

    def test\_display\_method(self):  # Define the test method for testing polymorphism

        # Creating a Facility object with test data

        facility = Facility("Region1", "District1", "LN001", "Facility1", "Type1",

                             "Address1", "Address2", "Address3", 100, 10, 20, 30,

                             "English", "Operator1", "Yes")

        # Asserting that the display() method of the Facility object returns the expected string

        self.assertEqual(facility.display(), "Facility Name: Facility1, License Number: LN001, Region: Region1")

# The following block runs the test case when the script is executed directly

if \_\_name\_\_ == "\_\_main\_\_":

    unittest.main()  # Running the tests

**A screen shot of a computer

AI-generated content may be incorrect.**

# 

# **Source Code**

# business/business\_logic.py

# Importing the Facility class from the model.facility module to work with Facility objects

from typing import List

from model.facility import Facility

# Function to display all the records (facilities) by calling their 'display' method

def display\_records(records: List[Facility]):

    # Iterate over the list of records (facilities) and display each record's details

    for record in records:

        print(record.display())

# Function to add a new facility to the list of facilities

def add\_facility(facilities: List[Facility], facility: Facility) -> List[Facility]:

    # Append the new facility to the list

    facilities.append(facility)

    return facilities

# Function to edit an existing facility based on its license number

def edit\_facility(facilities: List[Facility], license\_number: str, updated\_facility: Facility) -> List[Facility]:

    # Iterate over the list of facilities

    for i, facility in enumerate(facilities):

        # Check if the current facility's license number matches the given license number

        if facility.license\_number == license\_number:

            # If found, update the facility at the current index with the new details

            facilities[i] = updated\_facility

            return facilities

    # If no matching facility is found, print an error message

    print(f"Facility with License Number {license\_number} not found.")

    return facilities

# Function to delete a facility based on its license number

def delete\_facility(facilities: List[Facility], license\_number: str) -> List[Facility]:

    # Filter the list of facilities, excluding the one with the matching license number

    facilities = [facility for facility in facilities if facility.license\_number != license\_number]

    return facilities

# Function to view facilities (either by license number or all facilities)

def view\_facilities(facilities: List[Facility], license\_number: str = None):

    # If a license number is provided, search for that specific facility

    if license\_number:

        for facility in facilities:

            if facility.license\_number == license\_number:

                print(facility.display())  # Display the found facility

                return

        # If no matching facility is found, print an error message

        print(f"Facility with License Number {license\_number} not found.")

    else:

        # If no license number is provided, display all facilities

        for facility in facilities:

            print(facility.display())

# models/facility.py

# Importing the parent class 'Record' from the model.record module

from model.record import Record

# Facility class inherits from the Record class, which contains common properties like 'license\_number' and 'facility\_name'

class Facility(Record):

    # Constructor to initialize all the attributes for the Facility object

    def \_\_init\_\_(self, region, district, license\_number, facility\_name, facility\_type,

                 facility\_address\_1, facility\_address\_2, facility\_address\_3,

                 max\_children, max\_infants, max\_preschool, max\_school\_age,

                 language\_of\_service, operator\_id, designated\_facility):

        # Calling the constructor of the parent class 'Record' to initialize common properties

        super().\_\_init\_\_(license\_number, facility\_name)

        # Initializing additional properties specific to Facility

        self.region = region  # Region where the facility is located

        self.district = district  # The district of the facility

        self.facility\_type = facility\_type  # Type of the facility (e.g., daycare, school)

        self.facility\_address\_1 = facility\_address\_1  # First line of the facility's address

        self.facility\_address\_2 = facility\_address\_2  # Second line of the facility's address

        self.facility\_address\_3 = facility\_address\_3  # Third line of the facility's address (optional)

        self.max\_children = max\_children  # Maximum number of children the facility can accommodate

        self.max\_infants = max\_infants  # Maximum number of infants the facility can accommodate

        self.max\_preschool = max\_preschool  # Maximum number of preschool-aged children the facility can accommodate

        self.max\_school\_age = max\_school\_age  # Maximum number of school-aged children the facility can accommodate

        self.language\_of\_service = language\_of\_service  # The language(s) in which services are offered

        self.operator\_id = operator\_id  # Unique ID of the operator responsible for the facility

        self.designated\_facility = designated\_facility  # Whether this facility is a designated facility (Yes/No)

    # A method to display the facility details in a formatted way

    def display(self):

        # Returning a string representation of the Facility with key details: name, license number, and region

        return f"Facility Name: {self.name}, License Number: {self.license\_number}, Region: {self.region}"

# models/record.py

from abc import ABC, abstractmethod

"""

    Abstract base class for different record types.

    Defines the structure for record classes and an abstract display method.

    """

class Record(ABC):

    def \_\_init\_\_(self, license\_number, name):

        self.license\_number = license\_number

        self.name = name

    @abstractmethod

    def display(self):

        pass

# persistence/file\_io.py

# Importing necessary libraries

import csv

import uuid

from typing import List

from model.facility import Facility

# Function to read facility records from a CSV file

def read\_facility\_records(file\_path: str) -> List[Facility]:

    facilities = []  # List to store the Facility objects

    try:

        # Open the file in read mode

        with open(file\_path, mode='r') as file:

            csv\_reader = csv.reader(file)  # Create a CSV reader object

            next(csv\_reader)  # Skip the header row

            for i, row in enumerate(csv\_reader):

                # Check if the row has exactly 15 columns

                if len(row) == 15:

                    # Unpack row data and create a Facility object

                    facility = Facility(\*row)

                    facilities.append(facility)  # Append the facility to the list

                if i == 99:  # Stop reading after 100 records

                    break

    except FileNotFoundError:

        print(f"Error: File '{file\_path}' not found.")  # Handle file not found error

    return facilities  # Return the list of facility objects

# Function to write facility records to a CSV file

def write\_facility\_records(file\_path: str, records: List[Facility]) -> None:

    try:

        # Open the file in write mode, with newline='' to prevent blank rows in CSV

        with open(file\_path, mode='w', newline='') as file:

            csv\_writer = csv.writer(file)  # Create a CSV writer object

            # Write the header row

            csv\_writer.writerow(['Region', 'District', 'License-Number', 'Facility-Name', 'Facility-Type',

                                 'Facility-Address-1', 'Facility-Address-2', 'Facility-Address-3',

                                 'Max-Number-of-Children', 'Max-Number-of-Infants', 'Max-Number-of-Preschool-Aged-Children',

                                 'Max-Number-of-School-Age-Children', 'Language-of-Service', 'Operator-Id', 'Designated-Facility'])

            # Iterate over the records and write each record as a row in the CSV file

            for record in records:

                csv\_writer.writerow([record.region, record.district, record.license\_number, record.facility\_name,

                                     record.facility\_type, record.facility\_address\_1, record.facility\_address\_2,

                                     record.facility\_address\_3, record.max\_children, record.max\_infants,

                                     record.max\_preschool, record.max\_school\_age, record.language\_of\_service,

                                     record.operator\_id, record.designated\_facility])

    except Exception as e:

        print(f"Error: Unable to write to file: {e}")  # Handle other potential errors

# Function to generate a unique filename using UUID

def generate\_uuid\_filename() -> str:

    # Generate and return a unique filename with .csv extension

    return str(uuid.uuid4()) + ".csv"

# presentation/user\_interface.py

# Import necessary modules and functions

import os

from business.business\_logic import add\_facility, edit\_facility, delete\_facility, view\_facilities, display\_records

from persistence.file\_io import read\_facility\_records, write\_facility\_records, generate\_uuid\_filename

from model.facility import Facility

# Define the UserInterface class to interact with the user

class UserInterface:

    def \_\_init\_\_(self):

        self.facilities = []  # Initialize an empty list of facilities

    # Method to display the main menu and process user input

    def show\_menu(self):

        while True:

            # Display the options available to the user

            print("\n--- Facility Management System ---")

            print("\n Program by Tanaz Pathan")

            print("1. Load data from file")

            print("2. View all records")

            print("3. View record by License Number")

            print("4. Add new record")

            print("5. Edit record")

            print("6. Delete record")

            print("7. Save data to new file")

            print("8. Exit")

            # Take user input for the selected menu option

            choice = input("Enter choice: ")

            # Execute the corresponding action based on the user's choice

            if choice == '1':

                self.load\_data()  # Load data from a file

            elif choice == '2':

                self.view\_all\_records()  # View all facility records

            elif choice == '3':

                self.view\_record\_by\_license\_number()  # View a specific record by License Number

            elif choice == '4':

                self.add\_record()  # Add a new facility record

            elif choice == '5':

                self.edit\_record()  # Edit an existing facility record

            elif choice == '6':

                self.delete\_record()  # Delete a facility record

            elif choice == '7':

                self.save\_data()  # Save data to a new file

            elif choice == '8':

                print("Exiting program.")  # Exit the program

                break  # Break the loop to exit the program

            else:

                print("Invalid choice, please try again.")  # Handle invalid choices

    # Method to load facility data from a CSV file

    def load\_data(self):

        file\_path = input("Enter file path to load data: ")  # Ask user for the file path

        if os.path.exists(file\_path):  # Check if the file exists

            self.facilities = read\_facility\_records(file\_path)  # Read records from the file

            print(f"Loaded {len(self.facilities)} records.")  # Display how many records were loaded

        else:

            print(f"Error: File '{file\_path}' not found.")  # Error if file does not exist

    # Method to view all the facility records

    def view\_all\_records(self):

        if not self.facilities:  # Check if the facilities list is empty

            print("No records to display.")  # Display message if no records are available

        else:

            display\_records(self.facilities)  # Call function to display all records

    # Method to view a specific facility record by License Number

    def view\_record\_by\_license\_number(self):

        license\_number = input("Enter License Number to search: ")  # Ask for the License Number

        # Search and display the facility record

        view\_facilities(self.facilities, license\_number)  # This function will handle the display logic

    # Method to add a new facility record

    def add\_record(self):

        print("Enter details for the new facility:")

        # Prompt the user to input the details of the new facility

        region = input("Region: ")

        district = input("District: ")

        license\_number = input("License Number: ")

        facility\_name = input("Facility Name: ")

        facility\_type = input("Facility Type: ")

        facility\_address\_1 = input("Facility Address Line 1: ")

        facility\_address\_2 = input("Facility Address Line 2: ")

        facility\_address\_3 = input("Facility Address Line 3: ")

        max\_children = input("Max Number of Children: ")

        max\_infants = input("Max Number of Infants: ")

        max\_preschool = input("Max Number of Preschool-Aged Children: ")

        max\_school\_age = input("Max Number of School-Aged Children: ")

        language\_of\_service = input("Language of Service: ")

        operator\_id = input("Operator ID: ")

        designated\_facility = input("Designated Facility (Yes/No): ")

        # Create a new Facility object using the provided details

        new\_facility = Facility(region, district, license\_number, facility\_name, facility\_type,

                                facility\_address\_1, facility\_address\_2, facility\_address\_3,

                                max\_children, max\_infants, max\_preschool, max\_school\_age,

                                language\_of\_service, operator\_id, designated\_facility)

        # Add the new facility to the facilities list

        self.facilities = add\_facility(self.facilities, new\_facility)

        print("New facility added.")  # Notify the user

    # Method to edit an existing facility record

    def edit\_record(self):

        license\_number = input("Enter License Number of the facility to edit: ")  # Ask for the License Number

        # Find the facility object that matches the License Number

        facility = next((f for f in self.facilities if f.license\_number == license\_number), None)

        if facility:

            # Facility found, prompt the user to update the details

            print(f"Editing facility: {facility.facility\_name} ({facility.license\_number})")

            # Ask the user to input the updated details, defaulting to current values if left empty

            region = input(f"Enter Region (current: {facility.region}): ") or facility.region

            district = input(f"Enter District (current: {facility.district}): ") or facility.district

            facility\_name = input(f"Enter Facility Name (current: {facility.facility\_name}): ") or facility.facility\_name

            facility\_type = input(f"Enter Facility Type (current: {facility.facility\_type}): ") or facility.facility\_type

            facility\_address\_1 = input(f"Enter Facility Address 1 (current: {facility.facility\_address\_1}): ") or facility.facility\_address\_1

            facility\_address\_2 = input(f"Enter Facility Address 2 (current: {facility.facility\_address\_2}): ") or facility.facility\_address\_2

            facility\_address\_3 = input(f"Enter Facility Address 3 (current: {facility.facility\_address\_3}): ") or facility.facility\_address\_3

            max\_children = input(f"Enter Max Number of Children (current: {facility.max\_children}): ") or facility.max\_children

            max\_infants = input(f"Enter Max Number of Infants (current: {facility.max\_infants}): ") or facility.max\_infants

            max\_preschool = input(f"Enter Max Number of Preschool-Aged Children (current: {facility.max\_preschool}): ") or facility.max\_preschool

            max\_school\_age = input(f"Enter Max Number of School-Aged Children (current: {facility.max\_school\_age}): ") or facility.max\_school\_age

            language\_of\_service = input(f"Enter Language of Service (current: {facility.language\_of\_service}): ") or facility.language\_of\_service

            operator\_id = input(f"Enter Operator ID (current: {facility.operator\_id}): ") or facility.operator\_id

            designated\_facility = input(f"Enter Designated Facility (current: {facility.designated\_facility}): ") or facility.designated\_facility

            # Create a new Facility object with the updated details

            updated\_facility = Facility(region, district, license\_number, facility\_name, facility\_type,

                                        facility\_address\_1, facility\_address\_2, facility\_address\_3,

                                        max\_children, max\_infants, max\_preschool, max\_school\_age,

                                        language\_of\_service, operator\_id, designated\_facility)

            # Replace the old facility with the updated one in the list

            self.facilities[self.facilities.index(facility)] = updated\_facility

            print("Facility record updated successfully.")  # Notify the user

        else:

            print("Facility not found.")  # Handle case where the facility is not found

    # Method to delete a facility record

    def delete\_record(self):

        license\_number = input("Enter License Number of the facility to delete: ")  # Ask for the License Number

        # Delete the facility from the list

        self.facilities = delete\_facility(self.facilities, license\_number)

        print(f"Facility with License Number {license\_number} deleted.")  # Notify the user

    # Method to save the facility data to a new file

    def save\_data(self):

        file\_name = generate\_uuid\_filename()  # Generate a unique filename using UUID

        # Save the facility records to the generated file

        write\_facility\_records(file\_name, self.facilities)

        print(f"Data saved to {file\_name}.")  # Notify the user

from presentation.user\_interface import UserInterface

if \_\_name\_\_ == "\_\_main\_\_":

    print ("Program by Tanaz Pathan")

    ui = UserInterface()

    ui.show\_menu()