CHRIST (Deemed to be) University Bangalore

Department of Computer Science

MCA171 Python Programming

Exercises IV & V

**DOMAIN**: **Medical Equipment Failure Prediction**

**TOJIN VARKEY SIMSON**

**2447253(MCA-B)**

(1) For your specific domain topic, create a class hierarchy to demonstrate the

concepts of inheritance and polymorphism. Follow these steps:

(a) Define an abstract base class that represents a general concept in your

domain. Include at least one abstract method.

(b) Create at least three subclasses that inherit from the base class. Each

subclass should represent a more specific concept within your domain

and should implement the abstract method(s) in a way that reflects its

unique characteristics.

(c) Instantiate objects of these subclasses and store them in a list.

(d) Use a loop to call the implemented methods on each object, demonstrat-

ing how polymorphism allows different behaviors to be executed based

on the object's subclass type.

(2) For your specific domain topic, you are asked to develop a system to analyze

various types of data from different sources. The system should be flexible

and extendable to support different types of data analysis. To achieve this,

you need to implement an abstract class for data analysis and handle various

exceptions that might arise during the analysis process. Follow the steps

mentioned below:

1

2

(a) Create an abstract base class named 'DataAnalyzer' with an abstract

method 'analyze'.

(b) Implement at least two subclasses, say, 'TextDataAnalyzer' and 'Numer-

icData Analyzer' that inherit from DataAnalyzer.

(c) Each subclass should implement the 'analyze' method and handle specific

exceptions relevant to its analysis logic.

(d) Handle exceptions such as 'KeyError', 'TypeError', 'ValueError', and a

custom exception 'AnalysisError' in each subclass.

(e) Test the implementation as,

Instantiate objects of each subclass and store them in a list.

(ii) Create a list of sample data entries.

(iii) Use a loop to call the ‘analyze' method on each object for each data

entry, demonstrating how different subclasses handle exceptions and

perform analysis.

Evaluation Rubrics:-

(1) Timly submission: 2 marks.

(2) Correctness and Clarity: 4 marks.

(3) Complexity and Validation: 2 marks.

(4) Viva Voice: 2 marks.

Submission Guidelines:**-**

• Generate the single .pdf file for the given questions separately. The file name

should be your register number followed by the program number.

• Upload the pdf files in Google Classroom on or before the deadline mentioned.

**OUTPUT**:

Q1:

from abc import ABC, abstractmethod

from datetime import datetime, timedelta

# Abstract Base Class

class MedicalEquipment(ABC):

def \_\_init\_\_(self, equipment\_id, last\_service\_date):

self.equipment\_id = equipment\_id

self.last\_service\_date = last\_service\_date

@abstractmethod

def predict\_failure(self):

"""Predict the likelihood of equipment failure."""

pass

# Subclasses

class MRI(MedicalEquipment):

def predict\_failure(self):

days\_since\_last\_service = (datetime.now() - self.last\_service\_date).days

failure\_probability = min(0.9, days\_since\_last\_service / 1000)

return "MRI Equipment " + self.equipment\_id + " has a " + str(int(failure\_probability \* 100)) + "% chance of failure."

class CTScanner(MedicalEquipment):

def predict\_failure(self):

days\_since\_last\_service = (datetime.now() - self.last\_service\_date).days

failure\_probability = min(0.8, days\_since\_last\_service / 800)

return "CT Scanner " + self.equipment\_id + " has a " + str(int(failure\_probability \* 100)) + "% chance of failure."

class Ultrasound(MedicalEquipment):

def predict\_failure(self):

days\_since\_last\_service = (datetime.now() - self.last\_service\_date).days

failure\_probability = min(0.7, days\_since\_last\_service / 600)

return "Ultrasound " + self.equipment\_id + " has a " + str(int(failure\_probability \* 100)) + "% chance of failure."

# Instantiate and store in a list

equipment\_list = [

MRI("MRI123", datetime.now() - timedelta(days=500)),

CTScanner("CT456", datetime.now() - timedelta(days=300)),

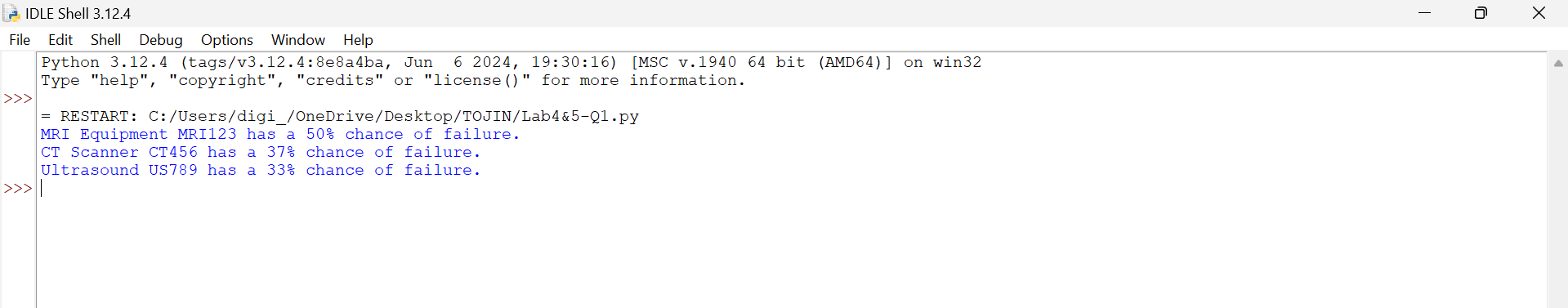
Ultrasound("US789", datetime.now() - timedelta(days=200))

]

# Demonstrate polymorphism

for equipment in equipment\_list:

print(equipment.predict\_failure())

****

**Q2:**

from abc import ABC, abstractmethod

# Custom Exception

class AnalysisError(Exception):

"""Custom exception for errors during analysis."""

pass

# Abstract Base Class

class DataAnalyzer(ABC):

@abstractmethod

def analyze(self, data):

"""Analyze the provided data."""

pass

# Subclass for Text Data Analysis

class TextDataAnalyzer(DataAnalyzer):

def analyze(self, data):

try:

# Check that data is a dictionary with a 'text' key

if not isinstance(data, dict):

raise TypeError("Data should be a dictionary.")

text = data.get('text')

if not isinstance(text, str):

raise ValueError("Text should be a string.")

# Dummy analysis: count words

word\_count = len(text.split())

return "Text analysis result: " + str(word\_count) + " words."

except KeyError as e:

raise KeyError("Missing expected key: " + str(e))

except TypeError as e:

raise TypeError("Type error: " + str(e))

except ValueError as e:

raise ValueError("Value error: " + str(e))

except Exception as e:

raise AnalysisError("An unexpected error occurred: " + str(e))

# Subclass for Numeric Data Analysis

class NumericDataAnalyzer(DataAnalyzer):

def analyze(self, data):

try:

# Check that data is a dictionary with a 'values' key

if not isinstance(data, dict):

raise TypeError("Data should be a dictionary.")

values = data.get('values')

if not isinstance(values, list):

raise ValueError("Values should be a list.")

if not all(isinstance(x, (int, float)) for x in values):

raise ValueError("All values should be numeric.")

# Dummy analysis: calculate mean

mean\_value = sum(values) / len(values)

return "Numeric analysis result: Mean value is " + str(round(mean\_value, 2)) + "."

except KeyError as e:

raise KeyError("Missing expected key: " + str(e))

except TypeError as e:

raise TypeError("Type error: " + str(e))

except ValueError as e:

raise ValueError("Value error: " + str(e))

except Exception as e:

raise AnalysisError("An unexpected error occurred: " + str(e))

# Sample Data

text\_data\_list = [

{"text": "Medical equipment requires regular maintenance."},

{"text": 12345}, # Invalid data

{} # Missing 'text' key

]

numeric\_data\_list = [

{"values": [1, 2, 3, 4, 5]},

{"values": "not a list"}, # Invalid data

{"values": [1, "two", 3]} # Non-numeric value

]

# Instantiate analyzers

analyzers = [

TextDataAnalyzer(),

NumericDataAnalyzer()

]

# Process text data

print("Text Data Analysis:")

for analyzer in analyzers:

if isinstance(analyzer, TextDataAnalyzer):

for data in text\_data\_list:

try:

result = analyzer.analyze(data)

print(result)

except AnalysisError as e:

print("Analysis error:", e)

except Exception as e:

print("Unexpected error:", e)

# Process numeric data

print("\nNumeric Data Analysis:")

for analyzer in analyzers:

if isinstance(analyzer, NumericDataAnalyzer):

for data in numeric\_data\_list:

try:

result = analyzer.analyze(data)

print(result)

except AnalysisError as e:

print("Analysis error:", e)

except Exception as e:

print("Unexpected error:", e)

