DPA – Lab 6

from typing import List, Optional

from datetime import datetime

class Factor:

def \_\_init\_\_(self, ID: int, Name: str, Status: str, Time: datetime):

self.ID = ID

self.Name = Name

self.Status = Status

self.Time = Time

def Show(self):

return f"ID: {self.ID}, Name: {self.Name}, Status: {self.Status}, Time: {self.Time}"

class Person(Factor):

def \_\_init\_\_(self, ID, Name, Status, Time, Habit: str, Location: str):

super().\_\_init\_\_(ID, Name, Status, Time)

self.Habit = Habit

self.Location = Location

def Move(self): pass

def GetIn(self): pass

def GetOut(self): pass

def Use(self): pass

class HomeAppliance(Factor):

def \_\_init\_\_(self, ID, Name, Status, Time, Location: str, EffectLevel: int):

super().\_\_init\_\_(ID, Name, Status, Time)

self.Location = Location

self.EffectLevel = EffectLevel

def SetStatus(self, status: str):

self.Status = status

class Environment(Factor):

def \_\_init\_\_(self, ID, Name, Status, Time, Temperature: float, Humidity: float, Illumination: float, NoiseLevel: float):

super().\_\_init\_\_(ID, Name, Status, Time)

self.Temperature = Temperature

self.Humidity = Humidity

self.Illumination = Illumination

self.NoiseLevel = NoiseLevel

def GetEnvironmentInfo(self):

return {

"Temperature": self.Temperature,

"Humidity": self.Humidity,

"Illumination": self.Illumination,

"NoiseLevel": self.NoiseLevel

}

class Internal(Environment):

def \_\_init\_\_(self, ID, Name, Status, Time, Temperature, Humidity, Illumination, NoiseLevel, Size: float):

super().\_\_init\_\_(ID, Name, Status, Time, Temperature, Humidity, Illumination, NoiseLevel)

self.Size = Size

def GetEnvironmentFromApplianceEffect(self): pass

class Weather(Environment):

def \_\_init\_\_(self, ID, Name, Status, Time, Temperature, Humidity, Illumination, NoiseLevel, Level: int):

super().\_\_init\_\_(ID, Name, Status, Time, Temperature, Humidity, Illumination, NoiseLevel)

self.Level = Level

def SetEffect(self): pass

class VirtualSpace:

def \_\_init\_\_(self, Size: float, Location: str, factors: List[Factor]):

self.Size = Size

self.Location = Location

self.factors = factors

def Show(self):

return f"Virtual Space: Size={self.Size}, Location={self.Location}, Factors={len(self.factors)}"

def GetEvent(self): pass

class DBConnection:

def \_\_init\_\_(self, ConnectionString: str):

self.ConnectionString = ConnectionString

def Read(self): pass

def Write(self): pass

def Close(self): pass

class Reasoning:

def \_\_init\_\_(self, dbConnection: DBConnection, refSmartHome: VirtualSpace):

self.dbConnection = dbConnection

self.refSmartHome = refSmartHome

def GetCases(self): pass

def DoReasoning(self): pass

def CaseMatching(self): pass

def GetEnvironmentInfo(self): pass

//Ex2.py

from typing import List, Optional

from datetime import datetime

class Factor:

def \_\_init\_\_(self, ID: int, Name: str, Status: str, Time: datetime):

self.ID = ID

self.Name = Name

self.Status = Status

self.Time = Time

def Show(self):

return f"ID: {self.ID}, Name: {self.Name}, Status: {self.Status}, Time: {self.Time}"

class Person(Factor):

def \_\_init\_\_(self, ID, Name, Status, Time, Habit: str, Location: str):

super().\_\_init\_\_(ID, Name, Status, Time)

self.Habit = Habit

self.Location = Location

def Move(self): pass

def GetIn(self): pass

def GetOut(self): pass

def Use(self): pass

class HomeAppliance(Factor):

def \_\_init\_\_(self, ID, Name, Status, Time, Location: str, EffectLevel: int):

super().\_\_init\_\_(ID, Name, Status, Time)

self.Location = Location

self.EffectLevel = EffectLevel

def SetStatus(self, status: str):

self.Status = status

class Environment(Factor):

def \_\_init\_\_(self, ID, Name, Status, Time, Temperature: float, Humidity: float, Illumination: float, NoiseLevel: float):

super().\_\_init\_\_(ID, Name, Status, Time)

self.Temperature = Temperature

self.Humidity = Humidity

self.Illumination = Illumination

self.NoiseLevel = NoiseLevel

def GetEnvironmentInfo(self):

return {

"Temperature": self.Temperature,

"Humidity": self.Humidity,

"Illumination": self.Illumination,

"NoiseLevel": self.NoiseLevel

}

class Internal(Environment):

def \_\_init\_\_(self, ID, Name, Status, Time, Temperature, Humidity, Illumination, NoiseLevel, Size: float):

super().\_\_init\_\_(ID, Name, Status, Time, Temperature, Humidity, Illumination, NoiseLevel)

self.Size = Size

def GetEnvironmentFromApplianceEffect(self): pass

class Weather(Environment):

def \_\_init\_\_(self, ID, Name, Status, Time, Temperature, Humidity, Illumination, NoiseLevel, Level: int):

super().\_\_init\_\_(ID, Name, Status, Time, Temperature, Humidity, Illumination, NoiseLevel)

self.Level = Level

def SetEffect(self): pass

class VirtualSpace:

def \_\_init\_\_(self, Size: float, Location: str, factors: List[Factor]):

self.Size = Size

self.Location = Location

self.factors = factors

def Show(self):

return f"Virtual Space: Size={self.Size}, Location={self.Location}, Factors={len(self.factors)}"

def GetEvent(self): pass

class DBConnection:

def \_\_init\_\_(self, ConnectionString: str):

self.ConnectionString = ConnectionString

def Read(self): pass

def Write(self): pass

def Close(self): pass

class Reasoning:

def \_\_init\_\_(self, dbConnection: DBConnection, refSmartHome: VirtualSpace):

self.dbConnection = dbConnection

self.refSmartHome = refSmartHome

def GetCases(self): pass

def DoReasoning(self): pass

def CaseMatching(self): pass

def GetEnvironmentInfo(self): pass

class DiseaseRule:

def \_\_init\_\_(self, name, factors):

self.name = name

self.factors = factors # e.g. {"Temperature": {"emergency": (>=35), ...}, ...}

def evaluate(self, environment):

results = {}

for factor, ranges in self.factors.items():

value = getattr(environment, factor, None)

if value is None:

results[factor] = "unknown"

continue

matched = False

for category, (low, high) in ranges.items():

if low <= value <= high:

results[factor] = category

matched = True

break

if not matched:

results[factor] = "out of range"

return results

class Reasoning:

def \_\_init\_\_(self, dbConnection=None, refSmartHome=None):

self.dbConnection = dbConnection

self.refSmartHome = refSmartHome

self.rules = []

def add\_rule(self, rule):

self.rules.append(rule)

def DoReasoning(self, environment, illness\_name):

for rule in self.rules:

if rule.name == illness\_name:

return rule.evaluate(environment)

return f"No rule found for illness: {illness\_name}"

illness\_rules = [

DiseaseRule("Asthma", {

"Humidity": {"emergency": (80, 100), "warning": (60, 79), "normal": (30, 59), "below normal": (0, 29)},

"NoiseLevel": {"emergency": (80, 200), "warning": (60, 79), "normal": (30, 59), "below normal": (0, 29)}

}),

DiseaseRule("Hypertension", {

"Temperature": {"emergency": (35, 50), "warning": (30, 34), "normal": (20, 29), "below normal": (0, 19)},

"NoiseLevel": {"emergency": (85, 200), "warning": (70, 84), "normal": (40, 69), "below normal": (0, 39)}

}),

DiseaseRule("Hypotension", {

"Temperature": {"emergency": (0, 10), "warning": (11, 15), "normal": (16, 25), "below normal": (26, 50)},

}),

DiseaseRule("Arthritis", {

"Humidity": {"emergency": (90, 100), "warning": (70, 89), "normal": (40, 69), "below normal": (0, 39)},

}),

DiseaseRule("Insomnia", {

"Illumination": {"emergency": (0, 10), "warning": (11, 30), "normal": (31, 70), "below normal": (71, 100)},

"NoiseLevel": {"emergency": (80, 200), "warning": (60, 79), "normal": (30, 59), "below normal": (0, 29)},

}),

DiseaseRule("COPD", {

"Temperature": {"emergency": (35, 50), "warning": (30, 34), "normal": (20, 29), "below normal": (0, 19)},

"Humidity": {"emergency": (90, 100), "warning": (70, 89), "normal": (40, 69), "below normal": (0, 39)},

}),

DiseaseRule("Migraine", {

"Illumination": {"emergency": (90, 100), "warning": (70, 89), "normal": (30, 69), "below normal": (0, 29)},

"NoiseLevel": {"emergency": (85, 200), "warning": (70, 84), "normal": (40, 69), "below normal": (0, 39)},

}),

DiseaseRule("Allergies", {

"Humidity": {"emergency": (80, 100), "warning": (60, 79), "normal": (30, 59), "below normal": (0, 29)},

"Illumination": {"emergency": (90, 100), "warning": (70, 89), "normal": (30, 69), "below normal": (0, 29)},

}),

DiseaseRule("Flu", {

"Temperature": {"emergency": (0, 10), "warning": (11, 15), "normal": (16, 25), "below normal": (26, 50)},

"Humidity": {"emergency": (90, 100), "warning": (70, 89), "normal": (40, 69), "below normal": (0, 39)},

}),

DiseaseRule("Depression", {

"Illumination": {"emergency": (0, 10), "warning": (11, 30), "normal": (31, 70), "below normal": (71, 100)},

})

]

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

class MockEnvironment:

def \_\_init\_\_(self, Temperature, Humidity, Illumination, NoiseLevel):

self.Temperature = Temperature

self.Humidity = Humidity

self.Illumination = Illumination

self.NoiseLevel = NoiseLevel

env = MockEnvironment(Temperature=28, Humidity=85, Illumination=25, NoiseLevel=90)

reasoning = Reasoning()

for rule in illness\_rules:

reasoning.add\_rule(rule)

print(reasoning.DoReasoning(env, "Asthma"))

//Ex3.py

class DiseaseRule:

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

self.name = name

self.factors = {}

def add\_condition(self, parameter, category, low, high):

self.factors.setdefault(parameter, {})[category] = (low, high)

def evaluate(self, env):

results = {}

for param, categories in self.factors.items():

val = getattr(env, param, None)

if val is None:

results[param] = "unknown"

continue

for cat, (low, high) in categories.items():

if low <= val <= high:

results[param] = cat

break

else:

results[param] = "out of range"

return results

def load\_rules\_from\_csv(csv\_path):

rules\_dict = {}

with open(csv\_path, newline='') as csvfile:

reader = csv.DictReader(csvfile)

for row in reader:

name = row["Disease"]

param = row["Parameter"]

cat = row["Category"]

low = float(row["Min"])

high = float(row["Max"])

if name not in rules\_dict:

rules\_dict[name] = DiseaseRule(name)

rules\_dict[name].add\_condition(param, cat, low, high)

return list(rules\_dict.values())

class MockEnvironment:

def \_\_init\_\_(self, Temperature, Humidity, Illumination, NoiseLevel):

self.Temperature = Temperature

self.Humidity = Humidity

self.Illumination = Illumination

self.NoiseLevel = NoiseLevel

def chatbot():

print("Hello! I can check how environmental factors affect your condition.")

rules = load\_rules\_from\_csv("disease\_rules.csv")

rule\_names = [r.name for r in rules]

print("Available diseases:", ", ".join(rule\_names))

name = input("Enter your disease: ").strip()

rule = next((r for r in rules if r.name.lower() == name.lower()), None)

if not rule:

print("Disease not found.")

return

try:

t = float(input("Temperature: "))

h = float(input("Humidity: "))

i = float(input("Illumination: "))

n = float(input("Noise Level: "))

except ValueError:

print("Invalid input.")

return

env = MockEnvironment(t, h, i, n)

result = rule.evaluate(env)

print("\n Diagnosis result for", rule.name)

for factor, category in result.items():

print(f" - {factor}: {category.upper()}")

print("Stay safe!")

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

chatbot()

//Homework.py

class DiseaseRule:

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

self.name = name

self.factors = {}

def add\_condition(self, parameter, category, low, high):

self.factors.setdefault(parameter, {})[category] = (low, high)

def evaluate(self, env):

results = {}

for param, categories in self.factors.items():

val = getattr(env, param, None)

if val is None:

results[param] = "unknown"

continue

for cat, (low, high) in categories.items():

if low <= val <= high:

results[param] = cat

break

else:

results[param] = "out of range"

return results

def load\_rules\_from\_csv(csv\_path):

rules\_dict = {}

with open(csv\_path, newline='') as csvfile:

reader = csv.DictReader(csvfile)

for row in reader:

name = row["Disease"]

param = row["Parameter"]

cat = row["Category"]

low = float(row["Min"])

high = float(row["Max"])

if name not in rules\_dict:

rules\_dict[name] = DiseaseRule(name)

rules\_dict[name].add\_condition(param, cat, low, high)

return list(rules\_dict.values())

class MockEnvironment:

def \_\_init\_\_(self, Temperature, Humidity, Illumination, NoiseLevel):

self.Temperature = Temperature

self.Humidity = Humidity

self.Illumination = Illumination

self.NoiseLevel = NoiseLevel

def SpeakText(command):

engine = pyttsx3.init()

engine.say(command)

engine.runAndWait()

def Listen():

r = sr.Recognizer()

with sr.Microphone() as source:

r.adjust\_for\_ambient\_noise(source, duration=0.5)

print("🎙️ Listening...")

audio = r.listen(source)

try:

text = r.recognize\_google(audio)

return text.lower()

except sr.UnknownValueError:

SpeakText("Sorry, I did not understand that.")

return None

except sr.RequestError as e:

SpeakText("Could not request results.")

return None

def voice\_bot():

SpeakText("Hello! I can check how environmental factors affect your condition.")

rules = load\_rules\_from\_csv("disease\_rules.csv")

rule\_names = [r.name for r in rules]

SpeakText("Please say the name of your disease.")

name = Listen()

if not name:

return

rule = next((r for r in rules if r.name.lower() == name), None)

if not rule:

SpeakText("Disease not found.")

return

try:

SpeakText("Say temperature")

t = float(Listen())

SpeakText("Say humidity")

h = float(Listen())

SpeakText("Say illumination")

i = float(Listen())

SpeakText("Say noise level")

n = float(Listen())

except (TypeError, ValueError):

SpeakText("Invalid input.")

return

env = MockEnvironment(t, h, i, n)

result = rule.evaluate(env)

SpeakText(f"Diagnosis result for {rule.name}")

for factor, category in result.items():

SpeakText(f"{factor} is {category}")

SpeakText("Stay safe!")

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

voice\_bot()

//disease\_rules.csv

Disease,Parameter,Category,Min,Max

Asthma,Humidity,emergency,80,100

Asthma,Humidity,warning,60,79

Asthma,Humidity,normal,30,59

Asthma,Humidity,below normal,0,29

Asthma,NoiseLevel,emergency,80,200

Asthma,NoiseLevel,warning,60,79

Asthma,NoiseLevel,normal,30,59

Asthma,NoiseLevel,below normal,0,29

Hypertension,Temperature,emergency,35,50

Hypertension,Temperature,warning,30,34

Hypertension,Temperature,normal,20,29

Hypertension,Temperature,below normal,0,19

Hypertension,NoiseLevel,emergency,85,200

Hypertension,NoiseLevel,warning,70,84

Hypertension,NoiseLevel,normal,40,69

Hypertension,NoiseLevel,below normal,0,39

Migraine,Illumination,emergency,90,100

Migraine,Illumination,warning,70,89

Migraine,Illumination,normal,30,69

Migraine,Illumination,below normal,0,29

Migraine,NoiseLevel,emergency,85,200

Migraine,NoiseLevel,warning,70,84

Migraine,NoiseLevel,normal,40,69

Migraine,NoiseLevel,below normal,0,39

Flu,Temperature,emergency,0,10

Flu,Temperature,warning,11,15

Flu,Temperature,normal,16,25

Flu,Temperature,below normal,26,50

Flu,Humidity,emergency,90,100

Flu,Humidity,warning,70,89

Flu,Humidity,normal,40,69

Flu,Humidity,below normal,0,39

Arthritis,Humidity,emergency,90,100

Arthritis,Humidity,warning,70,89

Arthritis,Humidity,normal,40,69

Arthritis,Humidity,below normal,0,39