**Міністерство освіти і науки України**

**Національний університет «Львівська політехніка»**

**Кафедра ЕОМ**

Звіт



**Лабораторна робота № 9**

“ ОСНОВИ ОБ’ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ У PYTHON”

з курсу “Кросплатформні засоби програмування”

Варіант: 4

Виконав:

ст.гр.КІ-306

Войтович П.Т.

Прийняв:

доцент кафедри ЕОМ

Олексів М.В.

**Львів 2024**

**Мета:** оволодіти навиками реалізації парадигм об’єктно-орієнтованого програмування використовуючи засоби мови Python.

**Теоретичні відомості:** Спадкування в ООП призначене для розширення функціональності існуючих класів шляхом утворення нових класів на базі вже існуючих. У Java реалізована однокоренева архітектура класів згідно якої всі класи мають єдиного спільного предка (кореневий клас в ієрархії класів) – клас Object. Решта класів мови Java утворюються шляхом успадковування даного класу. Будь-яке спадкування у мові Java є відкритим, при цьому аналогів захищеному і приватному спадкуванню мови С++ не існує. На відміну від С++ у Java можливе спадкування лише одного базового класу (множинне спадкування відсутнє). Спадкування реалізується шляхом вказування ключового слова class піcля якого вказується назва підкласу, ключове слово extends та назва суперкласу, що розширюється у новому підкласі.

Механізм поліморфізму забезпечує можливість присвоєння об’єктним змінним суперкласу об’єктів похідних класів та звертання з-під цих змінних до перевизначених у підкласі членів суперкласу. У Java всі об’єктні змінні є поліморфними. Поліморфізм реалізується за допомогою механізму динамічного (пізнього) зв’язування, який полягає у тому, що вибір методу, який необхідно викликати, відбувається не на етапі компіляції, а під час виконання програми.

У Java усі об’єктні змінні є типізовані. Механізми наслідування і поліморфізму дозволяють створювати нові типи (класи та інтерфейси) на базі вже існуючих та присвоювати об’єкти цих типів посиланням на об’єкти супертипу. В цьому випадку об’єкти підтипів мають ті самі елементи, що й об’єкти супертипу, тож таке висхідне приведення типів є безпечним і здійснюється компілятором автоматично. Проте присвоєння посиланню на об’єкт підтипу об’єкту супертипу не завжди є коректним, тому таке приведення вимагає явного приведення типів. При такому приведенні типів можливі дві ситуації:

• якщо посилання на об’єкт супертипу реально посилається на об’єкт підтипу, то приведення посилання на об’єкт супертипу до типу підтипу є коректним;

• якщо посилання на об’єкт супертипу посилається на об’єкт супертипу, то приведення посилання на об’єкт супертипу до типу підтипу викличе виключну ситуацію ClassCastExeption.

Абстрактні класи призначені бути основою для розробки ієрархій класів та не дозволяють створювати об’єкти свого класу. Вони реалізуються за допомогою ключового слова abstract. На відміну від звичайних класів абстрактні класи можуть містити абстрактні методи (а можуть і не містити). Абстрактні методи – це методи, що 6 оголошені з використанням ключового слова abstract і не місять тіла.

Інтерфейси вказують що повинен робити клас не вказуючи як саме він це повинен робити. Інтерфейси покликані компенсувати відсутність множинного спадкування у мові Java та гарантують визначення у класах оголошених у собі прототипів методів.

**Завдання:** 1. Написати та налагодити програму на мові Java, що розширює клас, що реалізований у лабораторній роботі №2, для реалізації предметної області заданої варіантом. Суперклас, що реалізований у лабораторній роботі №2, зробити абстрактним. Розроблений підклас має забезпечувати механізми свого коректного функціонування та реалізовувати мінімум один інтерфейс. Програма має розміщуватися в пакеті Група.Прізвище.Lab3 та володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.

2. Автоматично згенерувати документацію до розробленого пакету.

3. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub.

4. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації та завантажити його у ВНС.

5. Дати відповідь на контрольні запитання.

**Код програми:**

master.py

#Class Master implements master  
class Master:  
 #Constructor  
 def \_\_init\_\_(self, name\_master="", number=""):  
 self.name\_master = name\_master  
 self.number = number  
   
 #Method gets name master  
 def get\_name\_master(self):  
 return self.name\_master  
  
 # Method sets name master  
 def set\_name\_master(self, name\_master):  
 self.name\_master = name\_master  
  
 # Method gets number master  
 def get\_number(self):  
 return self.number  
  
 # Method sets number master  
 def set\_number(self, number):  
 self.number = number

breed.py

#Class Breed implements breed  
class Breed:  
 # Constructor  
 def \_\_init\_\_(self, breed=""):  
 self.breed = breed  
  
 # Method gets breed  
 def get\_breed(self):  
 return self.breed  
 # Method sets breed  
 def set\_breed(self, breed):  
 self.breed = breed

collar.py

#Class Collar implements collar  
class Collar:  
 #Constuctor  
 def \_\_init\_\_(self, collar=False):  
 self.collar = collar  
  
 # Method gets presence collar  
  
 def get\_presence(self):  
 return self.collar  
 # Method sets presence collar  
 def set\_presence(self, collar):  
 self.collar = collar

cat.py

from breed import Breed  
from master import Master  
from collar import Collar  
  
#Class Cat implements cat  
class Cat:  
 #Constructor  
 def \_\_init\_\_(self, name="None", breed="None", number="None", name\_master="None", info\_collar=False, location="None"):  
 self.name = name  
 self.breed = Breed(breed)  
 self.master = Master(number, name\_master)  
 self.collar = Collar(info\_collar)  
 self.energy = 5  
 self.food = 5  
 self.location = location  
 self.log\_activity(f"Cat's name: {self.name}, Cat's breed: {self.breed.get\_breed()}, Cat's master: {self.master.get\_number()}, {self.master.get\_name\_master()}, Cat's collar: {self.collar.get\_presence()}")  
  
 # Method gets name cat  
 def get\_name(self):  
 return self.name  
  
 # Method sets name cat  
 def set\_name(self, name):  
 self.name = name  
 self.log\_activity(f"New name for cat's: {self.name}")  
  
 # Method action play cat  
 def play(self, game):  
 if game == "mouse":  
 self.energy -= 1  
 self.food += 1  
 self.log\_activity(f"{self.name} plays with mouse: energy-1, food+1")  
 elif game == "bug":  
 self.energy += 1  
 self.food -= 1  
 self.location = "bugs"  
 self.log\_activity(f"{self.name} plays in bug: energy+1, food-1")  
 else:  
 self.energy -= 2  
 self.food -= 1  
 self.location = "outside"  
 self.log\_activity(f"{self.name} plays outside: energy-2, food-1")  
  
 # Method action sleep cat  
 def sleep(self):  
 self.energy += 1  
 self.log\_activity(f"{self.name} sleeps: energy+1")  
  
 # Method action clean cat  
 def clean(self):  
 self.energy -= 1  
 self.log\_activity(f"{self.name} cleans: energy-1")  
  
 # Method to activate a night\_vision cat's  
 def night\_vision(self, visor):  
 self.energy -= 1  
 self.log\_activity(f"{self.name} night vision {visor}: energy-1")  
  
 # Method to change location cat's  
 def set\_place(self):  
 self.log\_activity(f"{self.name} in {self.location}")  
  
 # Method action eat cat  
 def eat(self, food):  
 if food == "fish":  
 self.food += 1  
 self.log\_activity(f"{self.name} eats fish: food+1")  
 elif food == "meat":  
 self.food += 2  
 self.log\_activity(f"{self.name} eats meat: food+2")  
 else:  
 self.food -= 1  
 self.log\_activity(f"{self.name} doesn't want to eat candy: food-1")  
  
 # Method to show status cat's  
 def status(self):  
 if self.energy == 5 and self.food == 5:  
 self.log\_activity("Status: Nice")  
 elif self.energy < 5 or self.food < 5:  
 self.log\_activity(f"Status: {self.name} needs to sleep or eat")  
 else:  
 self.log\_activity(f"Status: {self.name} wants to play")  
 self.log\_activity(f"Energy level: {self.energy}, Food level: {self.food}")  
  
 # Method to show information about cat's  
 def info\_cat(self):  
 self.log\_activity(f"Cat's name: {self.name}, Cat's breed: {self.breed.get\_breed()}, Master's number: {self.master.get\_number()}, Master's name: {self.master.get\_name\_master()}, Cat's collar: {self.collar.get\_presence()}, Cat's energy: {self.energy}")  
  
 # Method to write action in file  
 def log\_activity(self, message):  
 with open("cat\_activity.log", "a") as f:  
 f.write(message + "\n")  
 print(message)

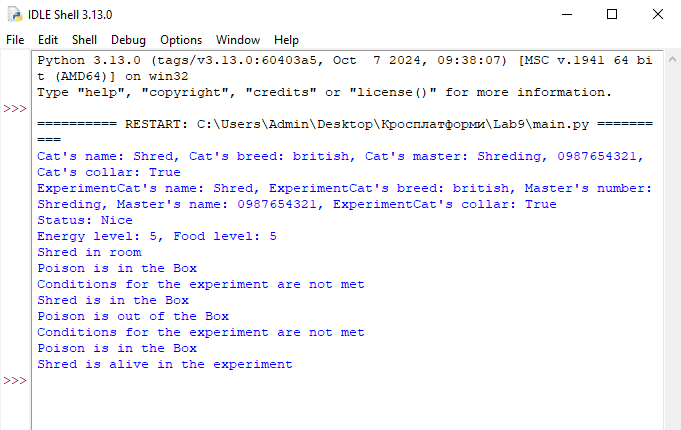
experiment\_cat.py

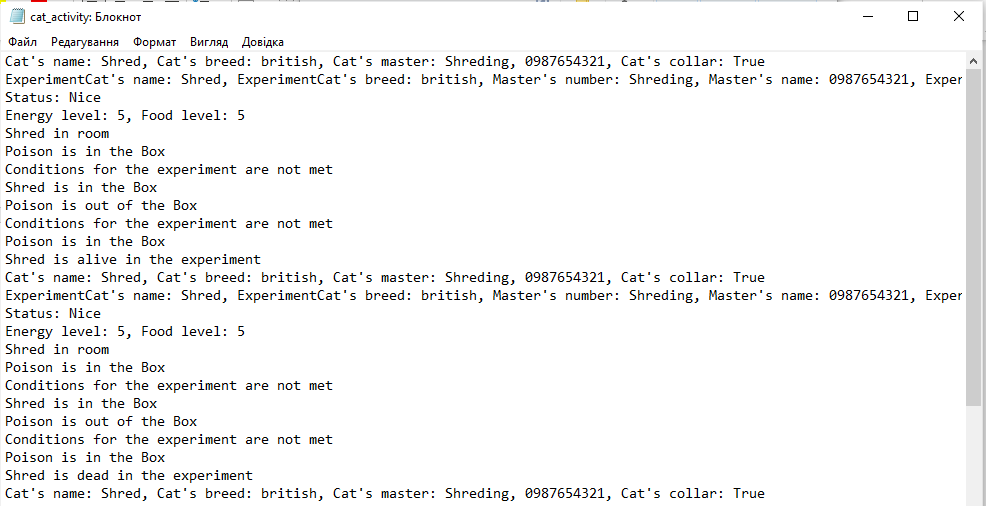
from cat import Cat  
import random  
  
# Class ExperimentCat implements experiment cat  
class ExperimentCat(Cat):  
 #Constructor  
 def \_\_init\_\_(self, name="None", breed="None", number="None", name\_master="None", info\_collar=False, location="None"):  
 super().\_\_init\_\_(name, breed, number, name\_master, info\_collar, location)  
 self.poison = 0  
 self.log\_activity("ExperimentCat's name: "+self.name+", ExperimentCat's breed: "+self.breed.get\_breed()+", Master's number: "+self.master.get\_number()+", Master's name: "+self.master.get\_name\_master()+", ExperimentCat's collar: "+str(self.collar.get\_presence()))  
  
 # Method to put cat in the box  
 def box(self):  
 self.location = "Box"  
 self.log\_activity(self.name+" is in the "+self.location)  
  
 # Method to put poison in the box  
 def put\_poison(self):  
 self.poison = 1  
 self.log\_activity("Poison is in the Box")  
  
 # Method to out poison in the box  
 def out\_poison(self):  
 self.poison = 0  
 self.log\_activity("Poison is out of the Box")  
  
 # Method to check status experiment  
 def check\_experiment(self):  
 if self.poison == 1 and self.location == "Box":  
 result = random.randint(0, 1)  
 if result == 0:  
 self.log\_activity(self.name+" is dead in the experiment")  
 else:  
 self.log\_activity(self.name+" is alive in the experiment")  
 else:  
 self.log\_activity("Conditions for the experiment are not met")

main.py

from experiment\_cat import ExperimentCat  
  
#Start project  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 shreding\_cat = ExperimentCat("Shred", "british", "0987654321", "Shreding", True, "room")  
 shreding\_cat.status()  
 shreding\_cat.set\_place()  
 shreding\_cat.put\_poison()  
 shreding\_cat.check\_experiment()  
 shreding\_cat.box()  
 shreding\_cat.out\_poison()  
 shreding\_cat.check\_experiment()  
 shreding\_cat.put\_poison()  
 shreding\_cat.check\_experiment()

**Результат:**

****

****

**Висновок:** оволодів навиками реалізації парадигм об’єктно-орієнтованого програмування використовуючи засоби мови Python.