OBYEKTGA YO‘NALTIRILGAN DASTURLASHDA POLIMORFIZM MISOLLARINI ISHLAB CHIQISH: BIRD, DOG VA CAT SINFLARI ASOSIDA.

Yusupov Mirsaid Abdulaziz o‘g‘li
Amaliy matematika va informatika
kafedrasi o‘qituvchisi
mirsaidbeky@gmail.com
Ma’murova Mashhuraxon Rahmatjon qizi
Farg‘ona davlat universiteti talabasi
mamurovmansurbek05@gmail.com

Annotatsiya: Ushbu maqolada obyektga yo‘naltirilgan dasturlash (OOP) tamoyillaridan biri - polimorfizm nazariy jihatdan tahlil qilinadi va uni dasturiy modellashtirish bo‘yicha amaliy natija keltiriladi. Maqola polimorfizmning rasmiy ta’rifini, turlarini, dizayn printsiplaridagi o‘rni va afzallik hamda cheklovlarini o‘rganadi. Amaliy qismda Animal ota sinfi va undan meros olgan Bird, Dog, Cat sinflari misolida runtime polimorfizmi metod overriding to‘liq ishlab chiqiladi, UML tasviri matn shaklida, kod namunasi, test va tahlil beriladi.

Kalit so‘zlar: Obyektga yo‘naltirilgan dasturlash, polimorfizm, vorislik, inkapsulyatsiya, Bird sinfi, Dog sinfi, Cat sinfi, metodni qayta yozish, polymorphism, obyekt modeli, dasturlash paradigmasi, abstraksiya.

Annotation: This article presents a theoretical analysis of polymorphism, one of the fundamental principles of object-oriented programming (OOP), and provides a practical implementation illustrating its application in software modeling. In the practical section, runtime polymorphism is demonstrated through method overriding using the Animal base class and its subclasses Bird, Dog, and Cat. A textual UML representation, code examples, test cases, and analytical discussion are also provided.

Keywords: Object-oriented programming, polymorphism, inheritance, encapsulation, Bird class, Dog class, Cat class, method overriding, polymorphism.

Аннотация: В данной статье рассматривается полиморфизм - один из ключевых принципов объектно-ориентированного программирования (ООП). В практической части реализован пример полиморфизма времени выполнения с использованием базового класса Animal и подклассов Bird, Dog и Cat, где метод speak() переопределяется в каждой производной структуре. Приведены текстовое UML-описание, программные фрагменты, тестовые случаи и аналитический разбор.

Ключевые слова: Объектно-ориентированное программирование, полиморфизм, наследование, инкапсуляция, класс Bird, класс Dog, класс Cat, переопределение метода, polymorphism.

Obyektga yo‘naltirilgan dasturlash paradigmasi zamonaviy dasturiy ta’minot loyihalarida keng qo‘llaniladi. U modullarni, komponentlarni va tizimning murakkab elementlarini abstraktsiyalash, qayta ishlatish hamda kengaytirishni osonlashtirish imkonini beradi. OOPning asosiy tamoyillari - inkapsulyatsiya, merosxo‘rlik va polimorfizm - dasturiy tiziml arning moslashuvchanligini va ishonchliligini oshirishda markaziy rol o‘ynaydi. Polimorfizm - bir nechta turdagi obyektlarga bitta umumiy interfeys orqali murojaat qilib, ularning turiga qarab turlicha xatti-harakatlarni amalga oshirish imkonini beruvchi printsipdir. Ushbu maqola polimorfizmni nazariy jihatdan tahlil qiladi va uni amalda qanday qo‘llash mumkinligini ko‘rsatadi. Polimorfizm (yunoncha poly - ko‘p, morph - shakl) - dasturlashda bir xil nomli operatsiyaning turli kontekstlarda turlicha ma’no kasb etishidir. OOP kontekstida u odatda bitta interfeys yoki ota sinf orqali ko‘plab voris sinflarning umumiy metodini turlicha amalga oshirishidir.

Polimorfizm ikki asosiy guruhga bo‘linadi:

1. Kompilyatsiya statik polimorfizm - metod overloading bir nom bilan bir nechta metod, argumentlar turiga yoki soniga qarab tanlanadi. Operator overloading ba’zi tillarda operatorlar turlicha aniqlanishi mumkin.
2. Ishlash dinamik polimorfizm - metod overriding ota sinfdagi metod voris sinflarda qayta aniqlanadi. Bu turi odatda vityual metodlar yoki interfeyslar orqali amalga oshiriladi va chaqirish vaqtida qaror qilinadi.

Polimorfizm SOLID printsiplariga mos keladi: tizim yangi sinflar qo‘shilganda mavjud kodni o‘zgartirmay kengaytirilishi kerak. Polimorfizm ham shu tamoyilni qo‘llashga yordam beradi.

Polimorfizmning afzalliklari va cheklovlari: Obyektga yo‘naltirilgan dasturlashda polimorfizm yondashuvi dastur arxitekturasini modulga ajratish, kodni qayta ishlatish va uni tartibli boshqarish imkonini beradi. Ushbu yondashuv orqali ishlab chiqilgan tizimlar yanada kengaytiriladigan bo‘lib, yangi sinflarni qo‘shish yoki mavjud funksionallikni yangilash nisbatan sodda tarzda amalga oshiriladi. Shuningdek, polimorfizm test jarayonlarini soddalashtiradi, chunki umumiy interfeyslar asosida turli obyektlarning xatti-harakatini yagona sxema orqali tekshirish mumkin. Biroq, polimorfizmning ayrim cheklovlari ham mavjud bo‘lib, xususan, to‘g‘ri abstraktsiya darajasini tanlash murakkab dizayn qarorlarini talab qiladi. Dinamik polimorfizmda funksiya tanlovi bajarilish vaqtiga qoldirilgani sababli, ba’zi hollarda samaradorlikka salbiy ta’sir ko‘rsatishi mumkin. Shuningdek, noto‘g‘ri

tuzilgan yoki juda chuqur merosxo‘rlik daraxti kodni tushunishni murakkablashtiradi va tizimning qo‘llab-quvvatlanishini qiyinlashtiradi.

Maqsad: Ushbu loyiha Animal nomli ota sinf va undan meros oluvchi Bird, Dog hamda Cat sinflaridan foydalanib polimorfizm tamoyilining amaliy qo‘llanilishini ko‘rsatishga qaratilgan. Loyiha doirasida umumiy interfeys vazifasini bajaruvchi speak() metodi barcha sinflarda mavjud bo‘ladi, biroq har bir sinfda u turlicha ma’noda qayta aniqlanadi. Shu orqali bir xil metod chaqiruvi turli obyektlar uchun o‘ziga xos natija berishi namoyish etiladi.

Loyihaning asosiy talablari quyidagilardan iborat:

1. Animal sinfi barcha voris sinflar uchun umumiy interfeyslarni belgilab beruvchi bazaviy sinf bo‘lishi lozim.
2. Bird, Dog va Cat sinflarida speak() metodi majburiy ravishda overriding qilinishi, ya’ni har bir sinf o‘ziga xos holda mazkur metodni qayta aniqlashi kerak.
3. Dastur umumiy massiv yoki ro‘yxat (list) orqali barcha hayvon obyektlarini bir xil tartibda iteratsiyalab, speak() metodini chaqirishi va har xil natijalar qayd etilishini ta’minlashi zarur.
4. Qo‘shimcha imkoniyat sifatida, har bir sinfga xos bo‘lgan atributlarni (masalan, Bird uchun wing_span, Dog uchun breed) kiritish mumkin. Bu kabi qo‘shimcha xususiyatlar polimorfizmning ishlashiga ta’sir qilmaydi, biroq sinflarning real modellashtirilishiga xizmat qiladi.
5. Yaratiladigan kod toza, testlanadigan va kengaytiriladigan bo‘lishi, ya’ni keyinchalik yangilanish va qo‘shimcha funksiyalar qo‘shishda muammosiz ishlashi talab etiladi.

Quyidagi kod namunasi polimorfizmni to‘liq amaliy ko‘rsatadi.

```
from abc import ABC, abstractmethod
from typing import List
class Animal(ABC):
    """
    Ota sinf: Animal.
    Abstrakt sinf sifatida speak() metodini majburiy qiladi.
    """
    def __init__(self, name: str):
        self.name = name
    @abstractmethod
    def speak(self) -> str:
        """
        Har bir voris sinf bu metodni qayta aniqlashi (override) shart.
        """
```

```
        pass
    def info(self) -> str:
        """Oddiy umumiy metod: hayvon nomini qaytaradi."""
        return f"Animal: {self.name}"
class Bird(Animal):
    def __init__(self, name: str, wing_span_cm: float):
        super().__init__(name)
        self.wing_span_cm = wing_span_cm
    def speak(self) -> str:
        return f"{self.name} (Bird): Chirq-chirq!"
    def fly_info(self) -> str:
        return f"Wing span: {self.wing_span_cm} cm"
class Dog(Animal):
    def __init__(self, name: str, breed: str):
        super().__init__(name)
        self.breed = breed
    def speak(self) -> str:
        return f"{self.name} (Dog): Vov-vov!"
    def breed_info(self) -> str:
        return f"Breed: {self.breed}"
class Cat(Animal):
    def __init__(self, name: str, color: str):
        super().__init__(name)
        self.color = color
    def speak(self) -> str:
        return f"{self.name} (Cat): Miyov-miyov!"
    def color_info(self) -> str:
        return f"Color: {self.color}"
def main():
    animals: List[Animal] = [
        Bird("Sparrow", 15.0),
        Dog("Rex", "German Shepherd"),
        Cat("Misty", "Gray")
    ]
    for a in animals:
        print(a.info())
        print(a.speak())
        print("---")
if __name__ == "__main__":
```

main()

1. Animal sinfi ABC abstract base class orqali abstrakt qilib belgilangan va speak() metodini @abstractmethod bilan majburiy qilgan. Bu voris sinflarning speak() metodini amalga oshirishini kafolatlaydi.
2. Bird, Dog, Cat sinflari speak() metodini o‘ziga xos tarzda qayta aniqlaydi.
3. main() funksiyasida turli obyektlar yagona animals listida joylanib, bir xil for sikli orqali boshqariladi - polimorfizmning markaziy belgisidir.

Polimorfizm yondashuvini amaliy tajriba asosida baholash maqsadida bir nechta test holatlari ko‘rib chiqildi. Birinchi holatda, tizimga yangi sinf qo‘shish jarayoni tahlil qilindi. Agar yangi sinf Animal bazaviy sinfidan meros olsa va speak() metodini o‘zida qayta aniqlasa, mavjud dastur arxitekturasi, xususan main() funksiyasidagi kod o‘zgarmaydi. Bu holat Open/Closed printsipining aniq namunasidir: tizim ochiq-kengaytiriladigan, ammo yopiq-o‘zgarmas bo‘lib qoladi. Ikkinchi holatda, agar voris sinf speak() metodini amalga oshirmasa, abstrakt sinf mexanizmi orqali TypeError yoki shunga o‘xshash xatoliklar erta bosqichda aniqlanishi mumkin. Bu dizayn nuqtai nazaridan muhim bo‘lib, noto‘g‘ri implementatsiyalar ishlab chiqish jarayonidayoq to‘xtatiladi. Uchinchi holatda sinflarga xos qo‘shimcha metodlardan masalan, fly_info(), breed_info(), color_info() kabi funksiyalardan foydalanish imkoniyatlari o‘rganildi. Ushbu metodlar polimorfik siklda bevosita chaqirilmaydi, ammo zarur hollarda turli obyektlarni isinstance() yordamida aniqlash yoki visitor pattern kabi rivojlangan dizayn yondashuvlarini qo‘llash orqali murakkab jarayonlarni boshqarish mumkin. Polimorfizm turli dasturlash tillarida o‘ziga xos usullar orqali amalga oshiriladi. Java tilida Animal abstrakt sinf yoki interfeys ko‘rinishida yaratiladi, speak() esa abstrakt metod sifatida ta’riflanadi. Polimorfik chaqiruvlar esa Animal a = new Dog(); a.speak(); kabi konstruktsiyalar orqali bajariladi; bu jarayonda @Override annotatsiyasi metodni to‘g‘ri qayta aniqlashni ta’minlashda muhim. C++ tilida esa virtual metodlar orqali dinamik polimorfizm tashkil etiladi, ota sinfdagi virtual void speak() = 0; ko‘rinishidagi toza abstrakt metodlar voris sinflar tomonidan to‘liq implementatsiya qilinishi shart. Shuningdek, C++ da destruktorning virtual e’lon qilinishi resurslarni to‘g‘ri boshqarish uchun zarur hisoblanadi. C# tilida polimorfizm virtual va override kalit so‘zlari yordamida amalga oshiriladi, shuningdek interfacelar keng qo‘llanadi. PHP va JavaScript tillarida esa prototip asosidagi modellardan yoki klass paradigmalaridan foydalaniladi; bu tillarning dinamik tabiatini inobatga olgan holda polimorfizm strukturalari mos ravishda shakllantiriladi. Arxitektura loyihalash jarayonida polimorfizm interfeyslar orqali umumiy boshqaruv qatlamlarini yaratish imkonini beradi. Bunday yondashuv ichki implementatsiyalarni mustaqil rivojlantirish, moduliylikni ta’minlash va sinflarning funksional chegaralarini aniq belgilashga xizmat qiladi. Har bir sinf o‘ziga tegishli mas’uliyatni bajaradi, bu esa tizimni boshqarishni soddalashtiradi va murakkab komponentlarni izchil tartibga soladi.

Bundan tashqari, polimorfik interfeyslar testlash jarayonini sezilarli darajada yengillashtiradi. Shunga qaramay, polimorfizmni qo‘llashda ayrim xavf va nosozliklar ham mavjud. Eng asosiylaridan biri noto‘g‘ri tanlangan abstraktsiyalar bilan bog‘liq bo‘lib, agar ota sinfning interfeyslari noaniq yoki keraksiz metodlarni o‘z ichiga olsa, voris sinflar majburiy ravishda ularga mos implementatsiya yozishga majbur bo‘ladi. Bu esa dizaynning ifloslanishiga olib keladi. Shuningdek, Liskov substitutability printsipiga qat’iy rioya qilish zarur bo‘lib, voris sinflar ota sinf o‘rnida qo‘llanganda dastur xatti-harakatlarini buzmasligi lozim. Aks holda, polimorfizm o‘z vazifasini bajara olmaydi va tizimning barqarorligi izdan chiqishi mumkin.

Umuman olganda, polimorfizm obyektga yo‘naltirilgan dasturlashning asosiy tamoyillaridan biri bo‘lib, kodni modulga ajratish, qayta ishlatish va tizimni kengaytirishga xizmat qiladi. Bird, Dog va Cat sinflari misolida ko‘rsatilgan polimorfizm ushbu tamoyilning sodda, tushunarli va pedagogik jihatdan qulay ko‘rinishini namoyon etadi. Bitta umumiy metod orqali turli obyektlarning o‘ziga xos xatti-harakatlarini boshqarish dastur arxitekturasining moslashuvchanligini sezilarli darajada oshiradi. To‘g‘ri tanlangan abstraktsiyalar, toza dizayn va obyektlararo munosabatlarning muvofiqligi polimorfizmning amaliy samaradorligini ta’minlaydi hamda katta ko‘lamli tizimlarda ishonchli ishlash imkonini yaratadi.

Foydalanilgan adabiyotlar:

1. Azizxo‘jayev A. Obyektga yo‘naltirilgan dasturlash asoslari
2. Yo‘ldoshev A., Xayitov B. Python dasturlash tili asoslari.
3. Toshmurodov F. Algoritmlar va dasturlash asoslari.
4. Rasulov M. C++ dasturlash asoslari.
5. Norqulov S. Kompyuter arxitekturasi va dasturlash.
6. Hamroyev Sh. Dasturlash paradigmalariga kirish.
7. Abdug‘aniyev R. Ma’lumotlar tuzilmalari va algoritmlar.
8. Qayumov A. Sun’iy intellekt va algoritmik yondashuvlar.
9. Raximov U. Dasturiy ta’minot injiniringi.
10. Xolmirzayev M. Obyektga yo‘naltirilgan tahlil va dizayn.
11. Qobilov B. Python va ma’lumotlar bilan ishlash.