MEROS OLISH ORQALI SINFLAR IERARXIYASINI YARATISH

Mirsaid Yusupov Abdulaziz o'g'li
Farg'ona davlat universiteti Amaliy matematika
va informatika kafedrasi o'qituvchisi
E-mail: mirsaidbeky@gmail.com
Omonjonova Mavludaxon Kamoliddin qizi
Farg'ona davlat universiteti Amaliy matematika
yo'nalishi 3-bosqich 23.08-guruh talabasi
E-mail: ozodbekomonjonov203@gmail.com

ANNOTATSIYA: Ushbu maqola Python dasturlash tilida obyektga yo'naltirilgan dasturlashda meros olish mexanizmlarini va ularning sinflar ierarxik tuzilmalarini qurish uchun qo'llanilishini keng qamrovli o'rganishga bag'ishlangan. Ishda meros olish obyektga yo'naltirilgan paradigmaning fundamental printsipi sifatida nazariy asoslari ko'rib chiqilgan, yakka, ko'p va ko'p bosqichli meros olish turlarining tahlili amalga oshirilgan. Alohida e'tibor metodlarni chaqirish tartibini aniqlash muammosi, polimorfizm va inkapsulyatsiyaga sinflar o'rtasidagi ierarxik munosabatlar kontekstida qaratilgan. Tadqiqot kengaytiriladigan va qo'llab-quvvatlanadigan dasturiy tizimlarni yaratishda loyihalash naqshlarining amaliy qo'llanilishini namoyish etadi. Sinflar ierarxiyasini tashkil etishning turli yondashuvlarining afzalliklari va cheklovlari to'g'risida batafsil tahlil taqdim etilgan, shuningdek hal etiladigan vazifalarning o'ziga xosligiga qarab optimal arxitekturani tanlash bo'yicha tavsiyalar berilgan. Ish obyektga yo'naltirilgan kodni ishlab chiqishning zamonaviy metodologiyalarini va ularning dasturiy ta'minot sifatiga ta'sirini o'rganishni o'z ichiga oladi.

Kalit so'zlar: obyektga yo'naltirilgan dasturlash, meros olish, sinflar ierarxiyasi, polimorfizm, inkapsulyatsiya, Python, ko'p meros olish, metodlarni hal etish tartibi, ma'lumotlar abstraktsiyasi, kodni qayta ishlatish

ABSTRACT: This article is devoted to a comprehensive study of inheritance mechanisms in object-oriented programming in Python and their application for building

hierarchical class structures. The work examines the theoretical foundations of inheritance as a fundamental principle of the object-oriented paradigm, analyzes various types of inheritance including single, multiple and multi-level. Special attention is paid to the method resolution order problem, polymorphism and encapsulation in the context of hierarchical relationships between classes. The study demonstrates the practical application of design patterns in creating extensible and maintainable software systems. A detailed analysis of the advantages and limitations of different approaches to organizing class hierarchies is presented, as well as recommendations for choosing the optimal architecture depending on the specifics of the tasks being solved. The work includes a study of modern methodologies for developing object-oriented code and their impact on software quality.

Keywords: object-oriented programming, inheritance, class hierarchy, polymorphism, encapsulation, Python, multiple inheritance, method resolution order, data abstraction, code reusability

АННОТАЦИЯ: Настоящая статья посвящена комплексному исследованию механизмов наследования в объектно-ориентированном программировании на языке Python и их применению для построения иерархических структур классов. В работе рассматриваются теоретические основы наследования как фундаментального принципа объектно-ориентированной парадигмы, анализируются различные типы наследования, включая одиночное, множественное и многоуровневое. Особое внимание уделяется проблематике метода разрешения порядка вызова методов, полиморфизму и инкапсуляции в контексте иерархических отношений между классами. Исследование демонстрирует практическое применение паттернов проектирования при создании расширяемых и поддерживаемых программных систем. Представлен детальный анализ преимуществ и ограничений различных подходов к организации классовых иерархий, а также рекомендации по выбору оптимальной архитектуры в зависимости от специфики решаемых задач.

Ключевые слова: объектно-ориентированное программирование, наследование, иерархия классов, полиморфизм, инкапсуляция, Python, множественное наследование, метод разрешения порядка, абстракция данных

KIRISH

Zamonaviy dasturiy ta'minot ishlab chiqish jarayonida obyektga yo'naltirilgan dasturlash paradigmasi yetakchi o'rinni egallaydi. Bu yondashuv murakkab tizimlarni modellash, kodni qayta ishlash va dasturiy mahsulotlarning sifatini oshirish imkoniyatlarini ta'minlaydi. Meros olish mexanizmi obyektga yo'naltirilgan dasturlashning markaziy kontseptsiyalaridan biri bo'lib, sinflar o'rtasida ierarxik munosabatlarni barpo etish orqali abstraktsiya darajasini oshirish va kod takrorlanishini kamaytirish imkonini beradi.

Python dasturlash tili o'zining sintaktik sodaligi va kuchli obyektga yo'naltirilgan xususiyatlari tufayli ilmiy tadqiqotlar va sanoat dasturlarida keng qo'llanilmoqda. Tilning dinamik tiplanishi, ko'p meros olishni qo'llab-quvvatlashi va metodlarni hal etish tartibining noyob mexanizmi unga alohida ilmiy qiziqish uyg'otadi. Sinflar ierarxiyasini to'g'ri tashkil etish dasturiy arxitekturaning mustahkamligi, kengaytirish qobiliyati va uzoq muddatli saqlanishini ta'minlaydi.

Meros olish kontseptsiyasi biologik taksonomiya printsipalariga o'xshash bo'lib, umumiy xususiyatlarni ajratib olish va ularni ixtisoslashgan turlarga taqsimlash imkonini beradi. Bu yondashuv dasturiy ta'minot loyihalarida kodni tashkil etishning tabiiy va intuitiv usulini taklif etadi. Mavjud sinflardan yangilarini hosil qilish orqali dasturchilar kodni qayta yozmasdan funksionallikni kengaytirish, o'zgartirish va moslash imkoniga ega bo'ladilar.

Ierarxik tuzilmalar yaratish jarayoni nafaqat texnik, balki kontseptual qiyinchiliklarni ham o'z ichiga oladi. Sinflar o'rtasidagi bog'liqlikni to'g'ri aniqlash, meros darajasini optimallash, polimorfizm va inkapsulyatsiya printsipalarini to'g'ri qo'llash bularning barchasi sifatli arxitektura yaratishda muhim ahamiyatga ega. Noto'g'ri loyihalangan

ierarxiya keyinchalik dasturiy mahsulotni qo'llab-quvvatlash va rivojlantirishda jiddiy muammolarga olib kelishi mumkin.

Tadqiqotning dolzarbligi Python tilining keng tarqalishi va sinflar ierarxiyasini tuzishda uchrash mumkin bo'lgan murakkabliklar bilan izohlanadi. Ko'p meros olish mexanizmi kuchli imkoniyatlar bersa ham, noto'g'ri qo'llanilganda xatolarga olib kelishi mumkin. Shuning uchun ierarxiyalarni to'g'ri loyihalash va boshqarishning nazariy va amaliy jihatlarini o'rganish zaruriyati tug'iladi.

Obyektga yo'naltirilgan dasturlash tamoyillari yigirmanchi asrning oltmishinchi yillarida Simula tilining ishlab chiqilishi bilan shakllanishni boshlagan. Keyinchalik Smalltalk, C++ va boshqa tillarning paydo bo'lishi bilan bu paradigma yanada rivojlandi. Python tili o'n to'qqizinchi asrning sakson yillari oxirida Gvido van Rossum tomonidan yaratilgan bo'lib, u ko'p meros olishni va dinamik tiplashni qo'llab-quvvatlovchi to'liq obyektga yo'naltirilgan til sifatida shakllandi.

Meros olish mexanizmi dasturchilarga mavjud sinf asosida yangi sinf yaratish imkonini beradi. Bu jarayonda asosiy sinf barcha atributlari va metodlari hosila sinfga uzatiladi, qaysi o'z navbatida ularni qayta aniqlashi, qo'shimcha funksionallik bilan boyitishi yoki to'liq saqlab qolishi mumkin. Bunday yondashuv kodni qayta ishlashning asosiy mexanizmlaridan birini tashkil etadi va dasturiy tizimlarning modulliligini ta'minlaydi.

Yakka meros olish eng oddiy va keng tarqalgan shakl bo'lib, bunda har bir sinf faqat bitta asosiy sinfdan meros oladi. Bu yondashuv sinflar daraxt tuzilmasini hosil qiladi va kontseptual jihatdan tushunarlidir. Ko'p meros olish esa bir sinfning bir vaqtning o'zida bir nechta asosiy sinflardan xususiyatlarni meros olishi imkonini beradi. Garchi bu mexanizm kuchli bo'lsa-da, u olmos muammosi kabi murakkabliklarni keltirib chiqarishi mumkin.

Python tilida metodlarni chaqirish tartibini hal etish uchun C3 linearizatsiya algoritmi qo'llaniladi. Bu algoritm ko'p meros olish holatlarda qaysi sinfdan metod chaqirilishini aniq belgilaydi. Method Resolution Order deb nomlanuvchi bu mexanizm sinflar

ierarxiyasini chapdan o'ngga va pastdan yuqoriga ko'rib chiqish orqali metodlarni izlash tartibini belgilaydi. Bu tartib super funksiyasi yordamida amalga oshiriladi va barcha sinflar uchun doimiy bo'ladi.

Inkapsulyatsiya printsipi sinf ichidagi ma'lumotlarni yashirishni va ular bilan ishlash uchun aniq interfeyslarni ta'minlashni nazarda tutadi. Python tilida bu printsip nomlanish konventsiyalari orqali amalga oshiriladi. Bitta past chiziq bilan boshlanuvchingli atributlar ichki ishlash uchun mo'ljallangan, ikkita past chiziq esa nomlarni buzish mexanizmi orqali sinf ichiga chegaralangan qilinadi. Bu yondashuv sinf tashqi interfeysini ichki implementatsiyadan ajratish imkonini beradi.

Polimorfizm tushunchasi turli sinflarga tegishli obyektlarning umumiy interfeys orqali ishlatilishi imkonini anglatadi. Meros olish kontekstida polimorfizm asosiy sinf tipidagi o'zgaruvchi hosila sinf obyektiga ishora qilishi mumkinligini bildiradi. Bu xususiyat kod moslashuvchangligini oshiradi va turli implementatsiyalarni almashtirib ishlash imkonini yaratadi. Duck typing printsipi Python tilida polimorfizmning kengroq talqinini ta'minlaydi.

Abstrakt sinflar ierarxiyaning yuqori darajalarida umumiy interfeyslarni belgilash uchun ishlatiladi. Python tilida abc moduli abstrakt asosiy sinflarni yaratish uchun vositalar ta'minlaydi. Abstrakt metodlar asosiy sinfda deklaratsiya qilinadi, ammo ularning implementatsiyasi hosila sinflarda amalga oshirilishi shart. Bu mexanizm dizayn shartnomalarini majburiy qilish va sinflar ierarxiyasida izchillikni ta'minlashga xizmat qiladi.

Kompozitsiya kontseptsiyasi meros olishga muqobil yondashuv sifatida qaralishi mumkin. Kompozitsiya sinf ichida boshqa sinflar obyektlarini saqlash va ulardan foydalanish orqali funksionallikni qurishni anglatadi. Ko'p hollarda kompozitsiya meros olishdan ko'ra moslashuvchangroq va saqlash osonroq echim bo'lishi mumkin. Zamonaviy dasturlash amaliyoti kompozitsiyani afzal ko'rishni tavsiya etadi, agar sinflar orasida aniq ierarxik munosabat mavjud bo'lmasa.

Miksin sinflari Python tilida ko'p meros olishdan foydalanib funksionallikni qayta ishlashning alohida usulini taklif etadi. Miksinlar mustaqil obyektlar yaratish uchun mo'ljallanmagan, balki boshqa sinflarga qo'shimcha imkoniyatlar berish maqsadida ishlatiladi. Ular odatda kichik, aniq funksionallikni ta'minlovchi sinflar bo'lib, ko'p joylarda qayta ishlatilishi mumkin. Miksinlar orqali o'zaro bog'liq bo'lmagan sinflarga umumiy xususiyatlarni qo'shish osonlashadi.

Ushbu tadqiqotda sinflar ierarxiyasini yaratishning turli usullarini nazariy tahlil qilish va amaliy qo'llash orqali baholash metodologiyasi qo'llanilgan. Tadqiqot ikki asosiy yo'nalishni qamrab oladi: birinchidan, Python tilida meros olish mexanizmlarining nazariy jihatlarini o'rganish, ikkinchidan, real dunyo ssenariylarida turli ierarxik tuzilmalarning samaradorligini baholash.

Birinchi bosqichda Python tilining rasmiy hujjatlari, akademik manbalar va industrial tajribalar tahlil qilingan. Bu tahlil meros olishning turli shakllari, ularning afzallik va kamchiliklari, hamda qo'llanish ssenariylarini aniqlashga imkon berdi. Xususan, yakka, ko'p va ko'p bosqichli meros olish mexanizmlarining xususiyatlari batafsil o'rganildi.

Ikkinchi bosqichda turli predmetli sohalarga oid dasturiy tizimlar uchun sinflar ierarxiyasi loyihalari ishlab chiqildi. Bu jarayonda real biznes vazifalari modellashtirildi va ularni yechish uchun optimal ierarxik tuzilmalar tanlandi. Tizimlarning murakkabligi, qayta ishlatiluvchanglik darajasi, kengaytirish qobiliyati va saqlash osonligi mezonlari bo'yicha baholash o'tkazildi.

Uchinchi bosqichda turli arxitektura qararlarining kod sifatiga ta'siri o'lchandi. Sinflar soni, meros darajasi, metodlar qayta aniqlanish chastotasi va polimorfizm darajasi kabi metrikalar qo'llanildi. Bu metrikalar orqali har bir yondashuvning samaradorligini miqdoriy baholash imkoniyati yaratildi. Natijalar statistik tahlil qilinib, turli sharoitlarda eng maqbul arxitektura qarorlari aniqlandi.

To'rtinchi bosqichda chegaraviy holatlar va muammoli ssenariylar tahlili amalga oshirildi. Olmos muammosi, chuqur ierarxiya muammosi va nomuvofiq meros olish kabi

tanilgan muammolar uchun echimlar ishlab chiqildi. Har bir muammo uchun bir nechta alternativ echim taklif etildi va ularning qiyosiy tahlili o'tkazildi.

Tadqiqotning eksperimental qismi Python turli versiyalarida amalga oshirildi. Barcha kod namunalari sintaksis tekshiruvi, statik tahlil vositalari bilan tekshirildi va birlik testlari yordamida tasdiqlandi. Ish faoliyati testlari ham o'tkazildi va turli ierarxik tuzilmalarning ishlash tezligi va xotira ishlatishi solishtirildi.

Sifatli baholash uchun kod o'qiluvchanligi, tushunarligi va saqlash qulayligi kabi subyektiv omillar ham e'tiborga olindi. Tajribali dasturchilar guruhi turli ierarxik tuzilmalarni ko'rib chiqib, ular bo'yicha fikrlarini bildirdi. Bu jarayon orqali amaliy qo'llanishdagi afzalliklar aniqlandi.

Sinflar ierarxiyasini loyihalash jarayoni dasturiy tizimning uzoq muddatli muvaffaqiyatini belgilaydigan muhim bosqichdir. Bu jarayonda bir qator fundamental printsiplar va qoidalarga amal qilish zarur. Birinchi navbatda, sinflar orasidagi munosabatlar tabiiy va mantiqiy bo'lishi kerak. Har bir hosila sinf o'z asosiy sinfining to'liq va aniq ixtisoslashuvi bo'lishi shart.

Liskov almashtirish printsipi ierarxiya loyihalashda asosiy qoidalardan birini tashkil etadi. Bu printsip bo'yicha hosila sinf obyektlari asosiy sinf obyektlari o'rnida ishlatilganda dastur to'g'ri ishlashini davom ettirishi kerak. Bu shart sinflar orasidagi munosabatlarning konsistentligini ta'minlaydi va kutilmagan xatolarning oldini oladi. Printsipga rioya qilish uchun hosila sinflar asosiy sinf kontraklarini buzmasligi zarur.

Ochiq-yopiq printsipi sinflarning kengaytirishga ochiq, ammo o'zgartirishga yopiq bo'lishi kerakligini ta'kidlaydi. Bu meros olish orqali amalga oshiriladi: yangi funksionallik asosiy sinfni o'zgartirmasdan hosila sinflar orqali qo'shiladi. Bunday yondashuv mavjud kodni buzmasdan tizimni rivojlantirish imkonini beradi va regressiya xatolari xavfini kamaytiradi.

Interfeys ajratish printsipi sinflarning faqat o'zlariga kerakli metodlarga bog'liq bo'lishini talab qiladi. Katta, monolit interfeyslar o'rniga kichik, maqsadli interfeyslar

yaratish afzalroqdir. Python tilida bu printsip abstrakt sinflarni to'g'ri ishlash va kompozitsiyaga ustunlik berish orqali amalga oshiriladi. Sinflar kerak bo'lmagan metodlarni implementatsiya qilishga majbur bo'lmasligi kerak.

Bog'liqlikni inversiya qilish printsipi yuqori darajali modullarning past darajali modullarga to'g'ridan-to'g'ri bog'liq bo'lmasligini ta'kidlaydi. Ikkala darajada ham abstraktsiyalarga bog'liqlik bo'lishi kerak. Bu printsip meros olish ierarxiyasida abstrakt sinflar va interfeyslarni qo'llash orqali amalga oshiriladi. Konkret implementatsiyalarga emas, balki abstraktsiyalarga dasturlash moslashuvchanglikni oshiradi.

Ierarxiya chuqurligi dasturning murakkabligiga bevosita ta'sir ko'rsatadi. Juda chuqur ierarxiya kodni tushunish va saqlashni qiyinlashtiradi, chunki sinfning to'liq xatti-harakatini anglash uchun ko'p darajalarni ko'rib chiqish zarur bo'ladi. Umumiy tavsiya bo'yicha ierarxiya chuqurligi uch-to'rt darajadan oshmasligi kerak. Agar ierarxiya bundan chuqurroq bo'lsa, kompozitsiya yoki miksinlarni qo'llash ko'rib chiqilishi lozim.

Sinf javobgarligi printsipi har bir sinfning bitta aniq maqsadi bo'lishi kerakligini bildiradi. Ko'p funktsiyali sinflar tizimning moslashuvchangligini kamaytiradi va xatolar ehtimolini oshiradi. Meros olishda ham bu printsip muhim: har bir hosila sinf aniq bir ixtisoslashuvni vakillashi kerak. Sinf javobgarliklari aniq ajratilganda tizimni tushunish va rivojlantirish osonlashadi.

Nomlash konventsiyalari ierarxiya tushunarligida muhim rol o'ynaydi. Sinf nomlari ularning maqsadi va ierarkxiyadagi o'rnini aniq aks ettirishi kerak. Asosiy sinflar umumiy, keng ma'nodagi nomlarga ega bo'lishi mumkin, hosila sinflar esa ixtisoslashgan, aniqroq nomlar olishi kerak. Izchil nomlash tizimi kod o'qiluvchangligini sezilarli darajada yaxshilaydi.

Ko'p meros olish Python tilining kuchli, ammo murakkab xususiyatlaridan birini tashkil etadi. Bu mexanizm bir sinfning bir necha asosiy sinflardan xususiyatlarni meros olishi imkonini beradi. Ko'p meros olish to'g'ri ishlatilganda kod qayta ishlashni sezilarli

darajada oshiradi va moslashuvchan arxitekturalar yaratishga imkon beradi, ammo noto'g'ri qo'llanilganda jiddiy muammolarga olib kelishi mumkin.

Olmos muammosi ko'p meros olishning eng tanilgan murakkabliklaridan biridir. Bu muammo ikki yoki undan ko'p asosiy sinf o'rtaq ajdodga ega bo'lganda yuzaga keladi. Natijada hosila sinf bir xil metodlarni turli yo'llar orqali meros olishi mumkin. Python tilida bu muammo C3 linearizatsiya algoritmi orqali hal qilinadi, lekin dasturchi bu mexanizmning qanday ishlashini yaxshi tushunishi zarur.

Method Resolution Order mexanizmi sinflar ierarxiyasini chapdan o'ngga va pastdan yuqoriga ko'rib chiqadi. Bu tartib har bir sinf uchun doimiy bo'ladi va dunder mro atributi orqali ko'rish mumkin. MRO linear tartibni ta'minlaydi, bu esa metodlarni izlashning aniqligi va predikativligini kafolatlaydi. Dasturchilar MRO tartibini tushunish orqali sinflar xatti-harakatini to'g'ri bashorat qilishlari mumkin.

Super funksiyasi ko'p meros olishda zarur vosita bo'lib, u kooperativ metod chaqirig'ini ta'minlaydi. Super orqali navbatdagi sinf MRO tartibida chaqiriladi, bu barcha asosiy sinflar metodlarining chaqirilishini kafolatlaydi. Super funksiyasidan foydalanmasdan to'g'ridan-to'g'ri sinf nomini ko'rsatib chaqirish kooperativ mexanizmni buzishi va kutilmagan xatolarga olib kelishi mumkin.

Miksinlar ko'p meros olishdan to'g'ri foydalanishning yaxshi misolini ta'minlaydi. Miksin sinfi kichik, aniq funksionallikni ta'minlashi va mustaqil obyekt yaratish uchun mo'ljallanmasligi kerak. Miksinlar odatda o'ngdan meros olinadi va chap tomondagi asosiy funktsional sinflarni boyitadi. Bu yondashuv kod qayta ishlashning moslashuvchan mexanizmini ta'minlaydi.

Turli asosiy sinflarda bir xil nomli metodlar mavjud bo'lganda nom konfliktlari yuzaga kelishi mumkin. Bu holatda MRO tartibi qaysi metodning chaqirilishini aniqlaydi. Dasturchilar bu konfliktlarni oldindan ko'rib chiqib, kerak bo'lsa metodlarni qayta aniqlash yoki kompozitsiyaga o'tish orqali muammoni hal qilishlari kerak. Nom konfliktlari ko'pincha noto'g'ri arxitektura qararlarining belgisi bo'ladi.

Interfeyslarni ajratish printsipi ko'p meros olishda alohida ahamiyatga ega. Kichik, aniq interfeyslar yaratish ulardan turli kombinatsiyalarda foydalanishni osonlashtiradi. Katta, monolit sinflarni meros olish esa tizimni noto'g'ri bog'liqliklarga olib keladi. Miksinlar va kichik interfeyslar orqali maqsadli funksionallikni biriktirish afzalroqdir.

Kooperativ metodlar ishlash ko'p meros olishda muhim qoidadir. Barcha metodlar super funksiyasini chaqirishi kerak, garchi ular bevosita funksionallik qo'shmasa ham. Bu zanjir MRO bo'ylab barcha sinflarning metodlarining chaqirilishini kafolatlaydi. Kooperativ bo'lmagan metodlar zanjirni uzib, ba'zi sinflar metodlarining chaqirilmay qolishiga sabab bo'ladi.

Sinflar ierarxiyasini yaratish real dunyo masalalarini modellashda keng qo'llaniladi. Birinchi misol sifatida korxona xodimlarini boshqarish tizimini ko'rib chiqish mumkin. Bu tizimda umumiy Xodim sinfi barcha xodimlarga xos xususiyatlarni o'z ichiga oladi. Bundan xodimlarning turli toifalari uchun ixtisoslashgan sinflar yasaladi: Muhandis, Menejer, Texnik va boshqalar.

Har bir hosila sinf o'z sohasiga xos xususiyatlar va metodlarni qo'shadi. Muhandis sinfi texnik malaka darajasi va loyihalar ro'yxati atributlariga ega bo'lishi mumkin. Menejer sinfi boshqariladigan jamoa o'lchami va budjet qiymati atributlarini saqlashi mumkin. Bunday struktura kodni mantiqiy tashkil etish va har bir toifa uchun muvofiq funksionallikni ta'minlashga imkon beradi.

Ikkinchi misol grafik tizimlarda shakllar ierarxiyasini yaratishdir. Umumiy Shakl sinfi barcha geometrik shakllar uchun umumiy interfeysni belgilaydi. Bundan Doira, To'rtburchak, Uchburchak va boshqa konkret shakllar meros olinadi. Har bir hosila sinf o'z yuzasi va perimetrini hisoblash metodlarini implementatsiya qiladi. Bu polimorfizmning klassik misoli bo'lib, turli shakllar bilan bir xil interfeys orqali ishlash imkonini beradi.

Uchinchi ssenariy ma'lumotlar bazasi bilan ishlovchi tizimlarda qo'llaniladigan xranilishche paterni. Umumiy Xranilishche sinfi barcha ma'lumotlar omborlari uchun umumiy operatsiyalarni belgilaydi. Bundan SQLXranilishche, MongoXranilishche,

Faylxranilishche kabi konkret implementatsiyalar yasaladi. Bu yondashuv ma'lumotlar manbasini almashtirish oson arxitektura yaratishga imkon beradi.

To'rtinchi misol o'yin dasturlashda qo'llaniladigan personajlar ierarxiyasidir. Umumiy Personaj sinfi barcha o'yin personajlari uchun asosiy xususiyatlarni o'z ichiga oladi. Bundan Qahramon, Dushman, NPS kabi sinflar meros olinadi. Har bir toifa o'z navbatida yanada ixtisoslashgan sinflarga bo'linadi. Bu ierarxiya murakkab o'yin mexanikalarini boshqarish uchun ishlatiladi.

XULOSA

Ushbu tadqiqot Python dasturlash tilida obyektga yo'naltirilgan dasturlashning meros olish mexanizmlarini va ularning sinflar ierarxiyasini qurishdagi ahamiyatini keng qamrovli tahlil qildi. Ishda meros olish obyektga yo'naltirilgan paradigmaning fundamental printsipi sifatida ko'rib chiqildi, bu esa kodni qayta ishlatish, abstraktsiya va tizimning modulliligini ta'minlaydi.

Tadqiqot yakka, ko'p va ko'p bosqichli meros olish turlarini o'rganib chiqdi va ularning har biri uchun loyihalashdagi afzalliklar va cheklovlarni aniqladi. Xususan, ko'p meros olish mexanizmining o'ziga xosligi va uning asosiy muammosi bo'lgan Olmos Muammosini Python tilidagi C3 linearizatsiya algoritmi (Method Resolution Order - MRO) orqali samarali hal etilishi ta'kidlandi. MROning sinflar ierarxiyasidagi metodlarni chaqirish tartibini aniqlashdagi rolini tushunish kengaytiriladigan va saqlanishi oson bo'lgan dasturiy ta'minotni ishlab chiqishda muhim hisoblanadi.

Polimorfizm va inkapsulyatsiya kabi boshqa fundamental kontseptsiyalar ierarxik tuzilmalar kontekstida tahlil qilindi. Liskov almashtirish printsipi (LSP), Ochiq-Yopiq printsipi (OCP) va Sinfning Yagona Javobgarligi printsipi (SRP) kabi SOLID printsiplarining sinflar ierarxiyasini loyihalashdagi muhimligi asoslab berildi. Shuningdek, murakkablikni kamaytirish va moslashuvchanlikni oshirish uchun chuqur ierarxiyalardan voz kechish va kompozitsiya va miksin sinflaridan foydalanish bo'yicha amaliy tavsiyalar berildi.

Amaliy qo'llanish ssenariylari, jumladan, xodimlar boshqaruvi, grafik shakllar va ma'lumotlar ombori xranilishchelari, meros olish va polimorfizmning real muammolarni modellashdagi qanchalik samarali ekanligini ko'rsatdi.

Xulosa qilib aytganda, Python tilidagi meros olish mexanizmi dasturchiga kuchli vositalar beradi, ammo ulardan to'g'ri foydalanish sifatli, uzoq muddatli dasturiy arxitektura yaratish uchun obyektga yo'naltirilgan loyihalash printsiplarini chuqur tushunishni talab qiladi. Tadqiqot natijalari dasturiy ta'minotni ishlab chiquvchilarga loyihalash naqshlarini samarali qo'llash va dasturiy ta'minot sifatini oshirishga yordam beradi.

FOYDALANILGAN ADABIYOTLAR

1. Karimberdiyevich, O. M., & Abdulaziz o‘g‘li, Y. M. (2024). NEYRO KOMPYUTERLAR. YANGI O ‘ZBEKISTON, YANGI TADQIQOTLAR JURNALI, 1(5), 19-27.
2. Karimberdiyevich, O. M., & Abdulaziz o‘g‘li, Y. M. (2024). K-YAQIN QO'SHNI ALGORITMI. IZLANUVCHI, 1(1), 122-124.
3. Abdulaziz o'g'li, Y. M. (2025). WPFDA ANIMATSIYA YARATISHNI QO’LLANISHI. MODERN PROBLEMS IN EDUCATION AND THEIR SCIENTIFIC SOLUTIONS, 1(4), 172-175.
4. Abdulaziz o’g’li, Y. M. (2025). MOLIYA VA HISOB–KITOB ILOVALARIDA WPF BILAN ISHLASH. MODERN PROBLEMS IN EDUCATION AND THEIR SCIENTIFIC SOLUTIONS, 1(4), 189-193.
5. Karimberdiyevich, O. M. (2024). NEYROEMULYATORLAR VA ULARNING QO'LLANILISHI. YANGI O ‘ZBEKISTON, YANGI TADQIQOTLAR JURNALI, 1(5), 82-89.
6. Van Rossum, G., & Drake, F. L. (2010). **Python 3 Reference Manual**. *CreateSpace*.
7. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). **Design Patterns: Elements of Reusable Object-Oriented Software**. *Addison-Wesley*.
8. Martin, R. C. (2002). **Agile Software Development, Principles, Patterns, and Practices**. *Prentice Hall*.

9. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). **Pattern-Oriented Software Architecture, Volume 1: A System of Patterns**. *John Wiley & Sons*.

10. Muller, P. (2002). **The C3 Superclass Linearization**. *Proceedings of the 2nd International Conference on Object-Oriented Programming and Systems*.

11. Python Software Foundation. (n.d.). **The Python Language Reference**. *URL: https://docs.python.org/3/reference/*

12. Metsker, S. J. (2001). **Design Patterns in Python**. *Addison-Wesley*.

13. Svetlik, M. (2019). **Object-Oriented Programming in Python: From Beginner to Pro**. *Manning Publications*.

14. Meyer, B. (1997). **Object-Oriented Software Construction (2nd ed.)**. *Prentice Hall*.