DEPENDENCY INJECTION BILAN ISHLASH. DEPENDENCY INJECTIONDAN AFZALLIKLARI. DEPENDENCY INJECTIONDAN FOYDALANISHNING QIYINCHILIKLARI

Qirg'izboyev Diyorbek Akmaljon o'g'li
Farg'ona davlat Universiteti Kompyuter ilmlari va dasturlash texnologiyalari yo'nalishi 2-kurs talabasi
Diyorbekqirgizboyev91@gmail.com
Yusupov Mirsaidbek Abdulaziz O'g'li
Farg'ona davlat universiteti amaliy matematika va informatika kafedrasi o'qituvchisi
mirsaidbeky@gmail.com

Anotatsiya: *Dependency Injection (DI) – bu dasturiy ta'minot arxitekturasidagi naqsh bo'lib, u klass bog'liqliklarini (unga bog'liq obyektlarni) konstruktor, metod yoki xususiyat orqali uzatish imkonini beradi.*

DI ning asosiy afzalliklari:

- *Komponentlar orasidagi bog'liqlikni kamaytiradi.*
- *Kodni test qilish jarayonini osonlashtiradi.*
- *Bog'liqliklarni boshqarishni soddalashtiradi.*

ASP.NET Core da DI mexanizmi ichki qurilgan bo'lib, u servislarni ro'yxatdan o'tkazish va ularni controller, middleware yoki boshqa komponentlarga kiritish imkonini beradi.

Kalit so'zlar: *Dependency Injection, Bo'shashmasdan bog'lanish va qayta foydalanish, Testlash, Barqarorlik va moslashuvchanlik, Miqyoslilik va kengaytma, Eski tizimlar, ob'ektga yo'naltirilgan dasturlashda, protsessual,* ***In'ektsiya,*** *API'lar, ma'lumotlar bazalari,* ***Ajratish.***

Аннотация: *Внедрение зависимостей (DI) — это шаблон в архитектуре программного обеспечения, который позволяет классу передавать зависимости (объекты, которые зависят от него) через конструктор, метод или свойство.*

Основными преимуществами DI являются:

• Уменьшает зависимости между компонентами.

• Упрощает процесс тестирования кода.

• Упрощает управление зависимостями.

ASP.NET Core имеет встроенный механизм DI, который позволяет регистрировать службы и внедрять их в контроллеры, промежуточное программное обеспечение или другие компоненты.

Ключевые слова*: Внедрение зависимостей, Слабая связанность и повторное использование, Тестирование, Стабильность и гибкость, Масштабируемость и расширяемость, Устаревшие системы, Объектно-ориентированное программирование, Процедурный, Внедрение, API, Базы данных, Разделение.*

Annotation: *Dependency Injection (DI) is a pattern in software architecture that allows you to pass dependencies (objects that depend on it) of a class through a constructor, method, or property.*

The main benefits of DI are:

• Reduces dependencies between components.

• Makes code testing easier.

• Simplifies dependency management.

ASP.NET Core has a built-in DI mechanism that allows you to register services and inject them into controllers, middleware, or other components.

***Keywords**: Dependency Injection, Loose coupling and reuse, Testing, Stability and flexibility, Scalability and extensibility, Legacy systems, object-oriented programming, procedural, Injection, APIs, databases, Decoupling.*

Kirish

Juda ko’p insonlar Dependency Injection (DI) haqida har-hil tafsilotlarga ega, va bu juda ko’plab muhokamalarga olib kelgan mavzu. Biz bilamizki juda ko’p discuss ya’ni muhokamalarda Dependency Injection (DI) haqida gap ketadi lekin buni aslida nima ekanligini va qanday ishlashini ko’pchilik bilmaydi va xato fikrlaydi. Biz shu maqola orqali Dependency Injection(Qaramlikdan qutqarish) haqida tiniqroq fikrga ega bo’lamiz. Dastlab biz Dependency nima ekanligi haqida gaplashamiz.

Dependency Injection (DI) - ob'ektlarni o'zi yaratish uchun sinfga tayanishdan ko'ra, sinfni ob'ektlar bilan to'ldirishga imkon beradigan usul.

Samarali qaramlikni boshqarish keng miqyosli va ta'minlanadigan tizimlarni yaratish uchun juda muhimdir. Qaramlikni in'ektsiya qilish (DI) dizayn shakli juda mashhur bo'lgan strategiyadir. Asosan, bog'liqlikni in'ektsiya qilish - bu komponentlar yoki ob'ektlarning qanday qurilishi va to'g'ri ishlash uchun zarur bo'lgan bog'liqliklarni qanday qo'lga kiritishini hal qiluvchi usul. Ob'ektga yo'naltirilgan dasturlashda Dependency Injection (DI) dizayn shakli tizim tarkibiy qismlari o'rtasidagi aloqani kamaytiradigan va kodni yanada modulli, sinovdan o'tkaziladigan va saqlanib turadigan usuldir. Sinflar odatda dasturiy ta'minotda o'z vazifalarini bajarish uchun ko'pincha boshqa sinflarga tayanadi.

Masalan: sinf ishlaydigan sinfga bog'liq bo'lishi mumkin . DI bo'lmasa, sinf to'g'ridan-to'g'ri o'z kodi ichida nusxani yaratadi yoki boshqaradi , bu esa ikki

sinfni mahkam bog'laydi. Ushbu yondashuv muammolarni keltirib chiqarishi mumkin, ayniqsa kelajakda sinflarni sinab ko'rish, kengaytirish yoki o'zgartirish kerak bo'lganda.

a) Dependency Injection bu muammoni sinf yaratishdan ko'ra, tashqi manbadan sinfga bog'liqliklarni (misoldagi kabi) kiritish orqali hal qiladi.EngineCar

b) Sodda qilib aytganda, DI sinfga kerak bo'lgan narsalarni (uning bog'liqliklarini) tashqaridan "in'ektsiya qilish" imkonini beradi, o'rniga, sinfni o'zi yaratish yoki boshqarishga ruxsat berish o'rniga.

Dependency Injectiondan qachon foydalanish kerak?

Quyida qaramlikni in'ektsiya qilish qimmatli yondashuv bo'lgan asosiy stsenariylar keltirilgan:

1) **Bo'shashmasdan bog'lanish va qayta foydalanish:**Ob'ektlar o'zlarining bog'liqliklarini yaratmaydi, qattiq ulanishlarni buzadi va ularni yanada mustaqil qiladi.

2) **Testlash:** Tashqi tizimlarga yoki xizmatlarga tayanmasdan alohida ob'ektlarni izolyatsiyada sinab ko'rishga imkon beruvchi bog'liqliklar uchun mock yoki test juftliklarini kiriting.

3) **Barqarorlik va moslashuvchanlik:** Bog'liqlikni in'ektsiya qilish ramkalari ko'pincha bog'liqliklarni boshqaradi, bu esa ularni kuzatib borish va sozlashni osonlashtiradi.

4) **Miqyoslilik va kengaytma:** Katta miqyosli ilovalarda DI murakkab qaramlik grafiklarini boshqarishga yordam beradi va osonroq miqyoslash va kengaytirish imkonini beradi.

5) **O'zaro kesish tashvishlari:** Kodni takrorlashni oldini olish va izchil yondashuvni targ'ib qilish, bir nechta komponentlarda ishlatiladigan jurnallar, xavfsizlik, keshlash yoki boshqa kesish muammolari uchun xizmatlarni kiriting.

Dependency Injectionni qachon ishlatish kerak emas?

Quyidagi holatlarda qaramlikni yuborishdan (DI) qochish kerak:

1) **Oddiy ilovalar**: Kichik yoki sodda loyihalar uchun DI-dan foydalanish keraksiz murakkablikni qo'shishi mumkin.

2) **Ishlash muammolari**: DI ramkalari ishlashga ta'sir qilishi mumkin bo'lgan ortiqcha xarajatlarni keltirib chiqarishi mumkin, ayniqsa real vaqtda yoki yuqori samarali ilovalarda.

3) **Bir nechta bog'liqliklar**: Agar sinfda faqat bir nechta oddiy bog'liqliklar bo'lsa, ularni qo'lda kiritish DIdan foydalanishdan ko'ra osonroq bo'lishi mumkin.

4) **Eski tizimlar**: DI-ni ishlatish uchun mahkam bog'langan eski tizimni qayta ko'rib chiqish ko'p vaqt talab qiluvchi va xavfli bo'lishi mumkin.

5) **Moslashuvchanlik etishmasligi** kerak: Agar bog'liqliklar tez-tez o'zgarmasa, DI haddan tashqari ko'p bo'lishi mumkin.

Dependency Injectiondan foydalanishning afzalliklari

Qaramlikka qarshi in'ektsiya dasturiy ta'minotni ishlab chiqish uchun juda ko'p foyda keltiradi.

1) **Modullilik va barqarorlikni oshirish:** Kod tarkibiy qismlarni o'z bog'liqliklaridan ajratish orqali toza va modulli bo'ladi.

2) **Yaxshilangan testlanish:** Mocks va stub bog'liqliklarni birlik sinovlari uchun osongina yuborish mumkin, bu esa komponentlarni ajratilgan sinovdan o'tkazishni osonlashtiradi.

3) **Qisqartirilgan bog'lanish va yaxshilangan bo'shashmasdan bog'lanish:** Komponentlar bo'shashmasdan bog'lanishni rag'batlantiradigan muayyan dasturlarga emas, balki mavhumliklarga bog'liq.

4) **Oson hamkorlik va qayta foydalanish:** Ishlab chiquvchilar bog'liqliklar haqida qayg'urmasdan asosiy funktsiyalarni amalga oshirishga e'tibor qaratishlari mumkin.

Dependency Injectiondan foydalanishning qiyinchiliklari

Qaramlikni in'ektsiya (DI) bilan birga keladigan ko'plab afzalliklar mavjud bo'lsa-da, potentsial salbiy tomonlarni tan olish va ularni dasturiy ta'minotni ishlab chiqish qarorlarida ko'rib chiqish juda muhimdir. Quyida xabardor bo'lishi kerak bo'lgan ba'zi asosiy muammolar keltirilgan:

1) **Ortib borayotgan murakkablik:** DI ramkalarini joriy etish yoki qo'lda in'ektsiya qilishni boshqarish kichik loyihalarga yoki oddiy kod bazalariga murakkablik qo'shishi mumkin.

2) **Ishlash vaqti xatolari:** Mos kelmaydigan bog'liqliklarni noto'g'ri konfiguratsiya qilish yoki in'ektsiya qilish, mahkam bog'langan koddagi kompilyatsiya vaqtidagi xatolarga qaraganda disk raskadrovka qilish qiyinroq bo'lgan ish vaqtidagi xatolarga olib kelishi mumkin.

3) **Ortiqcha xarajatlar va ishlash:** DI ramkalari, ayniqsa, mahkam bog'langan arxitekturalar bilan solishtirganda, xotiradan foydalanish va ishlash vaqti ishlashi jihatidan qo'shimcha xarajatlarni keltirib chiqarishi mumkin.

4) **Qaramlikni in'ektsiya qilishni o'zi tekshirish:** DI konfiguratsiyangizni va uning AOK qilingan bog'liqliklar bilan o'zaro ta'sirini sinab ko'rish to'g'ridan-to'g'ri ulangan komponentlarni sinash qiyinroq bo'lishi mumkin.

Dependency injection (DI) komponentlarni yaratish va foydalanishni ajratib turadigan dasturlash usuli. Dizayn shakli ob'ektga yo'naltirilgan dasturlashda paydo bo'ladi va SOLID tamoyillariga (xususan **S** va **D**) rioya qiladi. Shu kabi fikrlar protsessual yoki funktsional dasturlash kabi boshqa dasturiy paradigmalarga ham qo'llaniladi.

Atama kontseptsiyaning sirini ochishga yordam beradigan ikkita kalit so'zdan iborat:

Bog'liqliklar. Kod komponentlari o'z vazifalarini bajarish uchun ko'plab ob'ektlar va xizmatlarga (bog'liqliklarga) tayanadi. Tashqi resurslar, boshqa ob'ektlar yoki xizmatlar kabi turli xil bog'liqliklar mavjud.

In'ektsiya. Komponentlar ichki bog'liqliklarni yaratmaydi yoki bog'lamaydi DI. Buning o'rniga, texnika tashqi tomondan bog'liqliklarni ta'minlaydi (in'ektsiya qiladi). Ushbu yondashuv komponentlar va bog'liqliklar o'rtasida ajratish yaratadi.

Bog'liqliklar turli shakl va shakllarda bo'ladi. Ular komponentning ishlashi uchun juda muhimdir, ammo komponentning o'zi emas. **API'lar**, kutubxonalar, ma'lumotlar bazalari va boshqa tarkibiy qismlarning barchasi bog'liqlik sifatida ishlashi mumkin.

Qaramlikni in'ektsiya qilish qaramlikni boshqarish muammosini hal qiladi. Bog'liqlikni boshqarishni komponentdan ajratib, tashqi manba bog'liqliklarni yaratadi va boshqaradi.

Ajratish (yoki ajratish) komponentlar va modullar orasidagi ulanishlar sonini kamaytiradi. Kamroq ulanishlar soddalashtirilgan parvarishlash va moslashuvchanlikka olib keladi. Natijada, kod qayta foydalanish mumkin bo'ladi, chunki komponentlar mustaqil va turli kontekstlarda qo'llaniladi. Jamoalar doimiy muvofiqlashtirishni talab qilmaydi va modullarni parallel ravishda ishlab chiqishlari mumkin.

Boshqaruvning inversiyasi:

Yuqorida aytib o'tganimdek, Nazoratning inversiyasi - bu qaramlikni in'ektsiya qilish amalga oshiriladigan printsipdir. Bundan tashqari, nomidan ko'rinib turibdiki, Nazoratning inversiyasi asosan asosiy mas'uliyatni emas, balki sinfning turli xil qo'shimcha majburiyatlarini teskari qilish uchun ishlatiladi.

Agar sizga soddaroq so'zlar bilan tushuntirishim kerak bo'lsa, unda siz pishirish qobiliyatiga ega bo'lgan misolni ko'rib chiqing. IoC printsipiga ko'ra, siz boshqaruvni teskari qilishingiz mumkin, shuning uchun siz ovqat tayyorlash o'rniga, siz to'g'ridan-to'g'ri tashqaridan buyurtma berishingiz mumkin, bu erda siz eshigingizda ovqat olasiz. Shunday qilib, sizning eshigingizda sizga etkazib berilgan oziq-ovqat jarayoni Nazoratning inversiyasi deb ataladi.

Siz o'zingizni pishirishingiz shart emas, buning o'rniga siz oziq-ovqatni buyurtma qilishingiz va etkazib berish rahbariga siz uchun oziq-ovqat etkazib berishga ruxsat berishingiz mumkin. Shunday qilib, siz qo'shimcha majburiyatlarni o'z zimmangizga olishingiz shart emas va faqat asosiy ishingizga e'tiboringizni qarating.

Endi, siz Dependency Injection ortidagi printsipni bilasiz, men sizni Dependency Injection turlari orqali olib boraman.

Qaramlikni in'ektsiya turlari

Qaramlikni in'ektsiya qilishning asosan uch turi mavjud:

a) Konstruktorni yuborish: Ushbu turdagi injektsiyada injektor mijoz sinf konstruktori orqali bog'liqlikni ta'minlaydi.

b) Setter in'ektsiyasi / Xususiyatni in'ektsiya qilish: Ushbu turdagi in'ektsiya usuli mijoz tomonidan fosh etilgan setter usuliga bog'liqlikni kiritadi.

c) Interfeys qo‘yilgan Ushbu turdagi in'ektsiya injektori mijoz sinfiga qaramlikni ta'minlash uchun interfeysdan foydalanadi. Mijozlar qaramlikni qabul qiladigan setter usulini fosh etadigan interfeysni amalga oshirishlari kerak.

Xulosa

Xulosa qilib aytganda, Dependency Injection (DI) zamonaviy dasturiy ta'minot ishlab chiqishda muhim va kuchli pattern (andoza) hisoblanadi. U komponentlar orasidagi bog'liqlikni kamaytirish (decoupling) orqali kodning sezilarli darajada oshiradi. Garchi boshida Dependency Injection konsepsiyasi biroz murakkab tuyulishi mumkin bo'lsa-da, uning uzoq muddatli afzalliklari – toza, mustahkam va moslashuvchan dasturlar yaratish – dastlabki o'rganish sarf-xarajatlarini oqlaydi. Shu sababli, har bir dasturchi ushbu patternni o'rganishi va o'z loyihalarida qo'llashi dasturiy ta'minot sifatini yangi bosqichga olib chiqishda muhim qadam bo'lishini bilib oldim.

Foydalanilgan adabiyotlar ro’yxati:

1. " .NETda qaramlik kiritish "- Mark Seemann
2. " Harakatdagi bahor " - Kreyg Uolls
3. " Dizayn naqshlari: qayta foydalanish mumkin bo'lgan ob'ektga yo'naltirilgan dasturiy ta'minot elementlari- Erich Gamma, Richard Helm, Ralf Jonson, Jon Vlissides
4. " Toza kod: Agile dasturiy ta'minot hunarmandchiligi bo'yicha qo'llanma "- Robert C. Martin
5. Microsoft Docs - Dependency Injection in . NET
6. Spring Framework Documentation - Dependency Injection
7. " Java -da qaramlik kiritish - Baeldung"
8. Pluralsight: " .NETda qaramlik kiritish "
9. " TobelikInyeksiya tushuntirildi" - YouTube
10. " Tobelikni in'ektsiya qilish bo'yicha keng qamrovli qo'llanma " - Martin Fauler