**Fathullo To’yliyev**

**Java dasturlash tilidagi muhim mavzular jamlanmasi**





**ChatGPT asosida tuzilgan**

* **Java Cache**

**Java- da keshlash** *tez-tez ishlatiladigan* ***ma'lumotlarni*** *xotirada saqlash jarayoni* bo'lib,ularni *ma'lumotlar* *bazasidan* yoki boshqa tashqi manbalardan olish kerak bo'lmasdan tezda olish mumkin. Keshlash katta hajmdagi ma'lumotlarga kirishi yoki murakkab hisob-kitoblarni amalga oshirishi kerak bo'lgan ilovalarning ish faoliyatini yaxshilashi mumkin.

*Java ma'lumotlarni xotirada saqlash uchun ishlatilishi mumkin bo'lgan bir nechta keshlash mexanizmlarini taqdim etadi*. Java-da eng ko'p ishlatiladigan keshlash mexanizmlaridan ba'zilari quyidagilardir:

**HashMap** - hash map ma'lumotlarni saqlash uchun **key-value** juftliklaridan foydalanadigan to'plamdir. **Bu xotiradagi ma'lumotlarni keshlashning oddiy va tezkor usuli.**

**ConcurrentHashMap** - Bir vaqtning o'zida **HashMap** HashMapning **thread-safe** versiyasidir. U bir nechta **threadlar** tomonidan bir vaqtning o'zida kirishni **qo'llab-quvvatlash** uchun mo'ljallangan.

**Ehcache** - Ehcache *ochiq manbali keshlash kutubxonasi bo'lib, kengaytirilgan keshlash xususiyatlarini ta'minlaydi*, masalan*, taqsimlangan keshlash* va chiqarib tashlash siyosatlarini qo'llab-quvvatlash.

**Spring Cache** - Springkeshi bu Springilovalarida ma'lumotlarni keshlashning oddiy va izchil usulini ta'minlovchi keshlash abstraksiyasi.

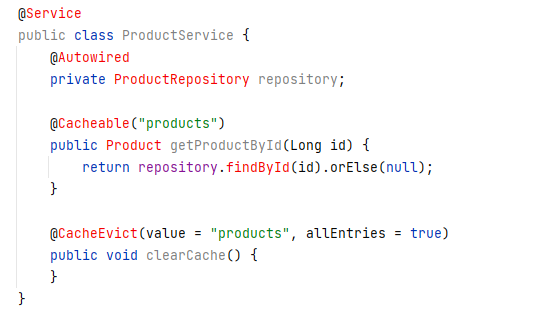
Keshlash Java ilovalari ish faoliyatini yaxshilash uchun kuchli vosita bo'lishi mumkin, ammo uni to'g'ri ishlatish muhimdir. Keshlashdan noto'g'ri foydalanish xatolar, xotiraning oqishi va boshqa ishlash muammolariga olib kelishi mumkin. Ilovangiz uchun keshlash strategiyasini sinchkovlik bilan ko'rib chiqish va kutilgandek ishlayotganiga ishonch hosil qilish uchun uni sinchiklab tekshirish muhimdir.

* **Spring Cache**: *Spring metod chaqiruvlari natijalarini keshlash uchun ishlatilishi mumkin bo'lgan keshlash abstraksiyasini ta'minlaydi*. Spring keshlash **Ehcache**, **Hazelcast** va **Redis** kabi turli xil *keshlash provayderlarini qo'llab-quvvatlaydi*. Misol uchun, *agar metod bir xil kirishlar uchun bir xil natijani qaytarsa, Spring natijani keshlashi va bir xil kirishlar bilan keyingi so’rov uchun uni keshdan qaytarishi mumkin.*  
  Sizga Java-da keshlashning ba'zi misollarini keltiraman.

**Web-ilovani keshlash**:

**Veb-ilovalar har bir so'rov uchun ma'lumotlarni olish uchun tez-tez qimmat ma'lumotlar bazasi so'rovlarini bajaradi**. *Keshlashdan foydalanib, so'rov natijalarini keshda saqlashingiz va keyingi so'rovlar uchun ma'lumotlar bazasi so'rovidan qochishingiz mumkin.*

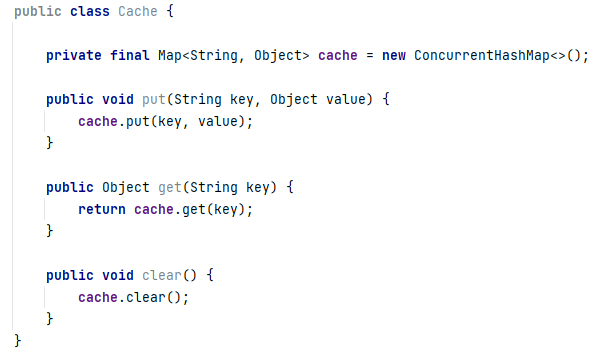
Masalan, Spring Framework keshlash abstraksiyasini taqdim etadi, undan siz ilovangizga keshlashni osongina qo'shish uchun foydalanishingiz mumkin. Spring Cache-dan qanday foydalanishga oddiy misol:



Ushbu misolda bizda @**Cacheable** bilan izohlangan **getProductById** metodiga ega **ProductService** mavjud. Ushbu izoh Spring metod natijasini keshda " **products** " deb nomlangan keshda saqlashni aytadi. Keshni tozalash uchun **clearCache** usulidagi @**CacheEvict** izohidan foydalaniladi.

**Xotira ichidagi keshlash:**

**Tez-tez ishlatiladigan ob'ektlarni xotirada saqlash uchun xotirada keshlashdan ham foydalanishingiz mumkin**. *Bu bajariladigan qimmat operatsiyalar sonini kamaytirish orqali ilovangiz ish faoliyatini yaxshilashi mumkin*. Quyida **ConcurrentHashMap** klassi yordamida xotiradagi keshlashdan qanday foydalanishga misol keltirilgan:



Ushbu misolda bizda oddiy **Cache** klassi mavjud bo'lib, u ob'ektlarni xotirada saqlash uchun **ConcurrentHashMap**-dan foydalanadi. Ob'ektni keshda saqlash uchun **put** metodi, keshdan ob'ektni olish uchun **get** metodi, keshni tozalash uchun **clear** metodi qo'llaniladi.

Bu Java ilovalarida keshlashdan qanday foydalanish mumkinligiga ikkita misol. **Keshlashdan foydalanib, siz ilovangizning ish faoliyatini yaxshilashingiz va bajariladigan qimmat operatsiyalar sonini kamaytirishingiz mumkin.**

**Brauzerni keshlash***: Veb-saytga tashrif buyurganingizda, brauzer rasmlar,* ***CSS*** *va* ***JavaScript*** *fayllari kabi statik resurslarni keshlaydi. Shunday qilib, keyingi safar veb-saytga tashrif buyurganingizda, brauzer resurslarni qayta yuklab olishga hojat qolmaydi, bu esa veb-saytni tezroq yuklash imkonini beradi*.

**Hibernate keshlash:** *Hibernate - bu Java ob'ektlarini ma'lumotlar bazasi jadvallari bilan taqqoslash uchun ishlatiladigan ob'ekt bilan bog'liq map (****ORM****) vositasi.* ***Hibernate******ma'lumotlar bazasiga keraksiz so'rovlardan qochish uchun keshlash mexanizmlarini taqdim etadi***. Misol uchun, *agar dastur bir xil ma'lumotlarni bir necha marta so'rasa, Hibernate so'rov natijalarini keshlashi va ma'lumotlar bazasiga bir nechta so'rovlar qilish o'rniga ularni keshdan qaytarishi mumkin*.

**Java Collections Framework**: **Java**, **ArrayList**, **HashMap** va **HashSet** kabi ma'lumotlar tuzilmalarini o'z ichiga olgan **Collections Framework**-ni taqdim etadi. *Ushbu ma'lumotlar tuzilmalari keshlashni amalga oshirish uchun ishlatilishi mumkin*. *Misol uchun, agar ilova tez-tez elementlar ro'yxatida qidiruv operatsiyasini bajarsa, qidiruv operatsiyasini qayta bajarishdan qochish uchun qidiruv natijalarini* ***HashMap****-da keshlashi mumkin*.

* **Java-da keshlash** - *bu dastur ma'lumotlar bazalari yoki fayllar kabi sekinroq ma'lumotlar manbalariga kirishni talab qilish sonini kamaytirish uchun tez-tez ishlatiladigan ma'lumotlarni xotirada saqlash orqali ilovalarning ishlashini yaxshilash uchun ishlatiladigan usul*.

Misol uchun, foydalanuvchi ma'lumotlarini olish uchun ma'lumotlar bazasiga tez-tez kiradigan dasturni tasavvur qiling. Ushbu ma'lumotlarni xotirada keshlash orqali bir xil ma'lumotlar uchun keyingi so'rovlar ma'lumotlar bazasiga qayta kirishga hojat qoldirmasdan tezroq amalga oshirilishi mumkin.

*Biroq, keshlashdan to'g'ri foydalanish muhim, chunki noto'g'ri foydalanish eskirgan yoki nomuvofiq ma'lumotlar kabi muammolarga olib kelishi mumkin*. *Shuning uchun keshlash strategiyalarini sinchkovlik bilan ko'rib chiqish va ularning kutilganidek ishlashiga ishonch hosil qilish uchun ularni sinab ko'rish va tasdiqlash muhimdir*.

* **Redis**

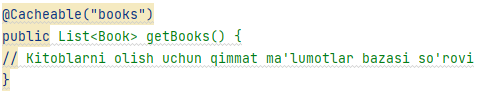
***Redis*** *-* ***ochiq manbali, xotiradagi ma'lumotlar tuzilmasi do'koni bo'lib, u ko'pincha ma'lumotlar bazasi, kesh va xabarlar vostachisi sifatida ishlatiladi*.** Redis tez va kengaytiriladigan ma'lumotlarga kirish va keshlashni ta'minlash uchun Spring Boot ilovalarida keng qo'llaniladi.

**Redis**-dan Spring Boot-da **Jedis** yoki **Lettuce Redis** mijozlari yordamida foydalanish mumkin, ular **Redis** ma'lumotlar tuzilmalari, masalan, **satrlar**, **xeshlar**, **ro'yxatlar**, **to'plamlar** va **tartiblangan to'plamlar** bilan o'zaro ishlash uchun qulay interfeyslarni ta'minlaydi. Ushbu mijozlar, shuningdek, **Redisning** **pub/sub** **xabar almashish** tizimini qo'llab-quvvatlaydi, bu esa dasturning turli qismlari o'rtasida real vaqt rejimida muloqot qilish imkonini beradi.

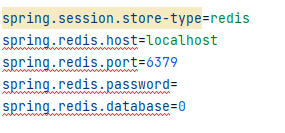
Spring Boot-da **Redis**-dan foydalanish ma'lumotlarga tezroq kirish va ma'lumotlar bazasiga yuklanishni kamaytirish, kengaytirilganlik va unumdorlik va katta hajmdagi ma'lumotlarni osonlik bilan boshqarish qobiliyati kabi bir qancha afzalliklarni berishi mumkin.

**Redisni Spring Boot bilan ishlatishning ba'zi misollari:**

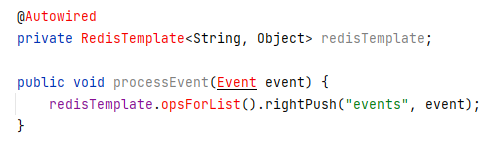
**1**.**Keshlash**: **Redis tez-tez kiriladigan ma'lumotlarni saqlash uchun kesh sifatida ishlatilishi mumkin**. Misol uchun, ma'lumotlar bazasiga yukni kamaytirish uchun Redis-da qimmat ma'lumotlar bazasi so'rovlari natijalarini keshlashingiz mumkin. Spring Boot-da siz metod natijalarini keshlash uchun @**Cacheable** izohidan foydalanishingiz mumkin. Mana bir misol:



**2.Sessiyani boshqarish**: Redis seans ma'lumotlarini taqsimlangan muhitda saqlash uchun ishlatilishi mumkin. Bu seans ma'lumotlarini yo'qotmasdan ilovangizni gorizontal ravishda o'lchash imkonini beradi. Spring Boot-da siz Spring-session-data-redis bog'liqligini qo'shish va uni application.properties faylida sozlash orqali Redis-ni seans do'koningiz sifatida sozlashingiz mumkin. Mana bir misol:



**3.Haqiqiy vaqtda ma'lumotlarni qayta ishlash**: Redis real vaqt rejimida katta hajmdagi ma'lumotlarni saqlash va qayta ishlash uchun real vaqtda ma'lumotlar ombori sifatida ishlatilishi mumkin. Masalan, siz Redis-dan foydalanuvchi voqealarini real vaqtda saqlash va real vaqtda tushunchalarni yaratish uchun ularni qayta ishlash uchun foydalanishingiz mumkin. Spring Boot-da siz Redis bilan ishlash uchun spring-data-redis kutubxonasidan foydalanishingiz mumkin. Mana bir misol:



Ushbu misolda biz **RedisTemplate**-dan voqealarni Redis ro'yxatiga o'tkazish uchun foydalanmoqdamiz. Keyinchalik tushunchalar yaratish uchun biz ushbu ro'yxatni real vaqtda qayta ishlashimiz mumkin.

**RabbitMQ**

***RabbitMQ*** *- bu tarqatilgan tizimning turli komponentlari o'rtasida xabar almashish imkonini beruvchi xabar vositachisi*. Spring Boot ilovasida RabbitMQ mikroservislar yoki boshqa komponentlar o'rtasida erkin bog'langan holda aloqani yoqish uchun ishlatilishi mumkin.

***RabbitMQ*** *xabarlar turli komponentlar tomonidan yuborilishi va qabul qilinishi mumkin bo'lgan queue- navbat yaratish orqali ishlaydi*. Navbatga xabar yuborilganda, RabbitMQ uni iste'molchi qabul qilish va qayta ishlashga tayyor bo'lgunga qadar saqlaydi. Bu komponentlarni ajratish imkonini beradi, chunki xabarlar asinxron tarzda yuborilishi va qabul qilinishi mumkin va jo‘natuvchi qabul qiluvchining o‘ziga xos xususiyatlarini bilishi shart emas.

*Spring Boot-da* ***RabbitMQ*** *Spring* ***AMQP*** *kutubxonasi yordamida osongina birlashtirilishi mumkin.* Bu dasturchilarga **RabbitMQ** ulanishlari va xabarlarni tinglovchilarni izohlar va Java klasslari yordamida sozlash imkonini beradi. Masalan, Spring Boot ilovasida **RabbitMQ** navbatiga xabar yuboradigan xizmat va bu xabarlarni qabul qiluvchi va qayta ishlovchi boshqa xizmat bo'lishi mumkin.

Umuman olganda, **RabbitMQ** kengaytiriladigan va ajratilgan tarzda taqsimlangan tizimning turli komponentlari o'rtasida aloqa o'rnatish uchun kuchli vosita bo'lishi mumkin.

**RabbitMQ** *- bu ilovaning turli qismlariga xabarlarni yuborish va qabul qilish orqali bir-biri bilan aloqa qilish imkonini beruvchi xabar almashish tizimi*. U jo'natuvchi va qabul qiluvchi o'rtasida vositachi bo'lib, xabarlarning ishonchli va samarali tarzda yetkazilishini ta'minlaydi.

**RabbitMQ** - bu xabarlar vositachisi bo'lib, u ilovalarga xabarlarni asinxron ravishda yuborish va qabul qilish orqali bir-biri bilan bog'lanish imkonini beradi. U turli xabar almashish tizimlari o'rtasida o'zaro hamkorlikni ta'minlaydigan **Advanced Message Queuing Protocol (AMQP)** standartiga amal qiladi.

RabbitMQ-dan Spring Boot ilovasida foydalanish uchun siz RabbitMQ bilan ishlashning oddiy va moslashuvchan metodini ta'minlovchi Spring **AMQP** frameworkidan foydalanishingiz mumkin. Framework xabar tinglovchilarini yaratish va xabarlarni yuborishni osonlashtiradigan izohlar va yordamchi dasturlarni taqdim etadi.

* Spring Boot ilovalarida RabbitMQ uchun umumiy foydalanish holatlari quyidagilardan iborat:

**Voqealarga asoslangan arxitekturalar**: *RabbitMQ nashr qilish-obuna tartibini amalga oshirish uchun ishlatilishi mumkin*, *bu erda voqealar mavzu bo'yicha nashr etiladi va bir nechta iste'molchilar tomonidan obuna bo'ladi.*

**Mikroservislar communication:** *RabbitMQ turli mikroservislar o'rtasidagi aloqani faollashtirish uchun ishlatilishi mumkin, bu ularga ishonchli va samarali tarzda xabarlar va ma'lumotlarni almashish imkonini beradi.*

**Task queue**: *RabbitMQ vazifalar navbatini amalga oshirish uchun ishlatilishi mumkin, bu erda vazifalar navbatga qo'shiladi va bir yoki bir nechta ishchi tomonidan qayta ishlanadi.*

**Umuman olganda, RabbitMQ kuchli xabar almashish tizimi bo'lib, Spring Boot ilovalarining ishonchliligi va kengaytirilishini yaxshilashga yordam beradi.**

Spring Boot-da RabbitMQ ko'pincha mikroservislar o'rtasida xabarga asoslangan aloqani amalga oshirish uchun ishlatiladi. Ramka Spring AMQP loyihasi orqali RabbitMQ bilan integratsiyani ta'minlaydi, bu ishlab chiquvchilarga Java ob'ektlari yordamida xabarlarni osongina yuborish va qabul qilish imkonini beradi.

RabbitMQ uni ishonchli xabar almashish tizimiga aylantiradigan bir nechta xususiyatlarni taqdim etadi, masalan, xabarni tasdiqlash, xabarning doimiyligi va xabarlarni yo'naltirish. Shuningdek, u xabarlar navbatlarga qanday yo'naltirilishini aniqlaydigan turli almashinuv turlarini qo'llab-quvvatlaydi, jumladan, to'g'ridan-to'g'ri, mavzu, sarlavhalar va fanout.

Umuman olganda, *RabbitMQ - bu taqsimlangan tizimdagi ilovalar va mikroservislar o'rtasidagi aloqani sezilarli darajada soddalashtiradigan kuchli xabar almashish tizimi*. Ishonchliligi, miqyoslanishi va boshqa xabar almashish tizimlari bilan o'zaro muvofiqligi tufayli u korporativ muhitda keng qo'llaniladi.

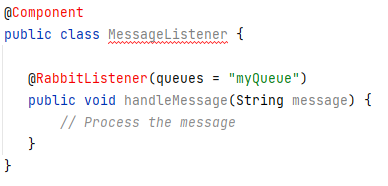
Ikki dastur o'rtasida xabarlarni yuborish va qabul qilish uchun Spring Boot-da RabbitMQ-dan foydalanishga misol:

Birinchidan, biz ishlab chiqaruvchi ilovasini o'rnatamiz. Ushbu misolda RabbitMQ-ga xabar yuborishni boshlash uchun REST so'nggi nuqtasidan foydalanamiz.

****

Ushbu kontroller xabarni satr sifatida qabul qiladigan va **RabbitTemplate** yordamida **RabbitMQ** almashinuviga yuboradigan metodga ega. Almashtirish va marshrutlash kaliti **convertAndSend** metodida ko'rsatilgan.

Keling, RabbitMQ-dan xabarlarni qabul qilish uchun iste'molchi ilovasini o'rnatamiz:

****

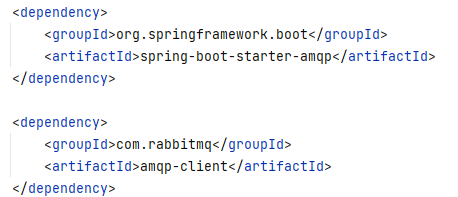
Bu oddiy Spring komponenti bo'lib, u ma'lum bir navbatdagi xabarlarni tinglaydi (ushbu misolda myQueue) va kelganda ularni qayta ishlaydi.

Bu ikki dastur endi RabbitMQ orqali xabar yuborish va qabul qilish orqali bir-biri bilan muloqot qilishi mumkin. Albatta, haqiqiy dunyo ilovasida ko'proq harakatlanuvchi qismlar bo'lishi mumkin, ammo bu misol RabbitMQ-dan Spring Boot-da qanday foydalanish mumkinligini oddiy namoyish etadi.

* **Spring Boot ilovasida RabbitMQ dan foydalanishga misollar.**

Aytaylik, bizda ikkita mikroservis bor, biri RabbitMQ-ga xabarlarni yuborish uchun, ikkinchisi esa bu xabarlarni qabul qilish uchun javobgardir.

Birinchidan, Spring Boot ilovamizda RabbitMQ ni sozlashimiz kerak. Buni pom.xml faylimizga quyidagi bog'liqliklarni qo'shish orqali amalga oshirishimiz mumkin:

****

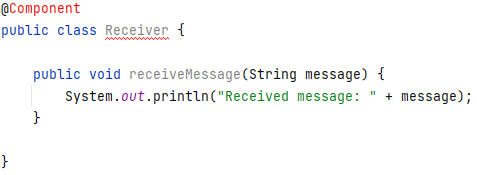
:

Keyin **RabbitMQ** uchun konfiguratsiya klassini aniqlashimiz kerak.

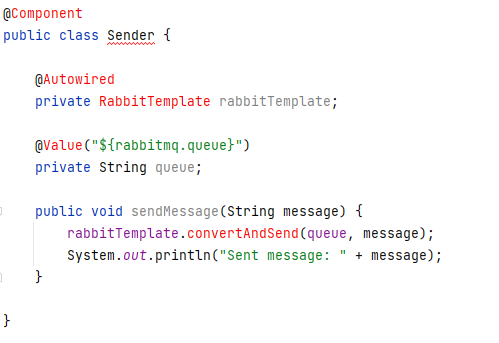


Ushbu konfiguratsiya klasida biz ulanish zavodini, Rabbit shablonini, navbatni, almashinuvni va bog'lashni aniqlaymiz. Shuningdek, biz xabarlarni tinglovchi konteynerini va xabar tinglovchi adapterini aniqlaymiz.

Keyin RabbitMQ dan xabarlarni qabul qiluvchi Qabul qiluvchi klassni aniqlaymiz:

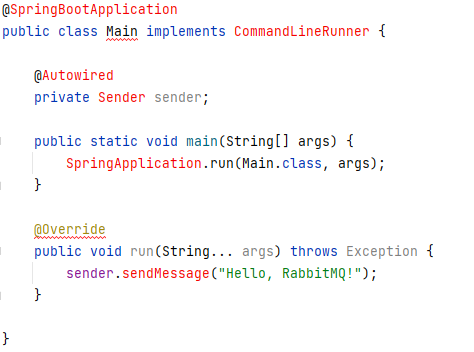
****

Nihoyat, RabbitMQ-ga xabar yuboradigan **Sender** klassini aniqlaymiz:

****

Bu klassda biz quyon shablonidan navbatga xabar yuborish uchun foydalanamiz.

Ushbu sinflardan foydalanish uchun biz oddiy asosiy klass yaratishimiz mumkin:



Ushbu ilovani ishga tushirganimizda, u RabbitMQ ga xabar yuboradi va Qabul qiluvchi klassi bu xabarni oladi va uni konsolga chop etadi.

Bu oddiygina misol, lekin u Spring Boot ilovasida RabbitMQ dan qanday foydalanish haqida fikr berishi kerak.

* **Bu erda RabbitMQ-dan foydalanishning bir nechta haqiqiy misollari keltirilgan:**

**Elektron tijorat:** *Elektron tijoratda RabbitMQ-ning mashhur qo'llanilishi buyurtmani qayta ishlashdir. Xaridor buyurtma berganida, RabbitMQ buyurtma ma'lumotlarini qayta ishlash uchun tegishli tizimga (inventarizatsiyani boshqarish, jo'natish, hisob-kitob va boshqalar) yo'naltirish uchun ishlatilishi mumkin. Bu buyurtmalarning to'g'ri va samarali bajarilishini ta'minlashga yordam beradi.*

**Moliyaviy xizmatlar**: *Moliyaviy xizmatlarda RabbitMQ savdo tizimlari o'rtasida real vaqt rejimida aloqa qilish uchun ishlatiladi. Misol uchun, savdo tizimi RabbitMQ-ga savdo amalga oshirilganda xabar yuborishi mumkin va boshqa tizim bu xabarni qabul qilib, hisob balanslarini yangilash yoki hisobotlarni yaratish uchun foydalanadi.*

**Sog'liqni saqlash**: Sog'liqni saqlash sohasida RabbitMQ yurak monitorlari, infuzion nasoslar va ventilyatorlar kabi tibbiy asboblar o'rtasida xabar almashish uchun ishlatilishi mumkin. Bu bemor to'g'risidagi ma'lumotlarning aniq va tez uzatilishini ta'minlashga yordam beradi va tibbiy xatolar xavfini kamaytiradi.

**Logistika va transport**: Logistika va transportda RabbitMQ transport rejimi, ob-havo sharoiti va boshqa omillar bo'yicha real vaqt ma'lumotlarini tahlil qilish orqali etkazib berish yo'nalishlarini optimallashtirish uchun ishlatilishi mumkin. Misol uchun, yuk mashinasi o'zining joylashuv ma'lumotlarini RabbitMQ-ga yuborishi mumkin va marshrutlash tizimi bu ma'lumotlarni oladi va undan eng samarali marshrutni hisoblash uchun foydalanadi.

**Bu ishonchli xabar almashish va tizimlar o'rtasida samarali aloqani ta'minlash uchun RabbitMQ-dan haqiqiy hayot stsenariylarida qanday foydalanish mumkinligiga bir nechta misollar.**

RabbitMQ – ilovalarga xabar almashish orqali bir-biri bilan muloqot qilish imkonini beruvchi xabarlar vostachisi. U xabarlar navbati modeliga amal qiladi, bu yerda xabarlar navbatga yuboriladi va ularni keyinchalik qabul qiluvchi ilova tomonidan olish mumkin. Bu turli xil ilovalar va tizimlarni oson integratsiyalash imkonini beradi va ular orasidagi aloqani sodda va ishonchli qiladi. Spring Boot-da RabbitMQ-dan xabar almashish tizimlarini amalga oshirish va kengaytiriladigan va xatolarga chidamli taqsimlangan tizimlarni yaratish uchun foydalanish mumkin.

* **Web Services (SOAP/REST)**

**Veb-xizmatlar (SOAP/REST) - bu Internet orqali turli dasturiy ta'minot tizimlari o'rtasidagi aloqani osonlashtirish uchun ishlatiladigan amaliy dasturlash interfeyslarining (API) ikkita keng tarqalgan turi**. SOAP (Oddiy ob'ektga kirish protokoli) va REST (vakillik holatini uzatish) zamonaviy veb-ishlab chiqishda keng qo'llaniladi va ularni turli stsenariylarga mos keladigan turli xususiyatlarga ega.

**SOAP** - bu **XML**-ga asoslangan xabarlarni kompyuter tizimlari o'rtasida almashish uchun protokol. SOAP xabarlar uchun standart konvert formatini, jumladan, header va asosiy qismni belgilaydi. Header bo'limida xabar haqidagi metama'lumotlar, masalan, uning manzili va so'ralgan operatsiya turi mavjud. Body bo'limi uzatilayotgan ma'lumotlarni o'z ichiga oladi.

Boshqa tomondan, REST - bu resurslarga kirish va boshqarish uchun HTTP so'rovlaridan foydalanadigan veb-xizmatlarni yaratish uchun arxitektura uslubi. RESTful veb-xizmatlari resurslar ustida operatsiyalarni bajarish uchun GET, POST, PUT va DELETE kabi HTTP usullariga tayanib, oddiy va yengil aloqa protokolidan foydalanadi. REST API ma'lumotlarni turli formatlarda, jumladan XML, JSON va oddiy matnda qaytaradi.

Spring Boot - bu SOAP va REST veb-xizmatlaridan foydalanadigan veb-ilovalarni ishlab chiqishni osonlashtiradigan mashhur Java frameworki. Spring Boot veb-xizmatlarni yaratish, joylashtirish va boshqarishni osonlashtiradigan turli xil vositalar va kutubxonalarni taqdim etadi.

SOAPni Spring Boot bilan ishlatish uchun dasturchilar S*pring Web Services* kutubxonasidan foydalanishlari mumkin. Ushbu kutubxona SOAP-ga asoslangan veb-xizmatlarni yaratish uchun vositalar to'plamini taqdim etadi, jumladan XML marshaling va unmarshaling, WSDL generatsiyasi va xabarlarni tekshirishni qo'llab-quvvatlash.

Spring Boot bilan REST-dan foydalanish uchun ishlab chiquvchilar Spring Web MVC kutubxonasidan yoki reaktiv dasturlash uchun Spring WebFlux kutubxonasidan foydalanishlari mumkin. Ushbu kutubxonalar RESTful veb-xizmatlarini yaratish uchun vositalar to'plamini taqdim etadi, jumladan so'rovlarni qayta ishlash, marshrutlash, kontentni muhokama qilish va javoblarni yaratish.

Umuman olganda, Spring Boot bilan veb-xizmatlardan (SOAP/REST) foydalanish veb-xizmatlarni ishlab chiqishning past darajadagi tafsilotlarini ko'rib chiquvchi vositalar va kutubxonalar to'plamini taqdim etish orqali veb-ilovalarni ishlab chiqishni soddalashtirishi mumkin, bu esa ishlab chiquvchilarga biznes mantig'iga e'tibor qaratish va funksionallik.

Veb-xizmatlari (SOAP/REST) Internet orqali ilovalar o'rtasida ma'lumot almashish uchun ishlatiladi. Ular har bir dastur tomonidan ishlatiladigan dasturlash tili, operatsion tizimi yoki platformasidan qat'i nazar, turli ilovalar uchun bir-biri bilan muloqot qilishning standartlashtirilgan usulini ta'minlaydi.

Spring Boot-da siz boshqa ilovalar tomonidan iste'mol qilinadigan veb-xizmatlarni yaratish uchun SOAP va REST veb-xizmatlaridan foydalanishingiz mumkin. SOAP - bu ilovalar o'rtasida ma'lumot almashish uchun HTTP orqali XML-ga asoslangan xabarlardan foydalanadigan xabar almashish protokoli. Boshqa tomondan, REST HTTP so'rovlaridan ma'lumotlarni yanada engilroq va moslashuvchan tarzda **GET**, **POST**, **PUT** va **DELETE** uchun ishlatadi.

Spring Boot-da SOAP va REST veb-xizmatlaridan foydalanish uchun siz mos ravishda *Spring Web Services va Spring Web MVC* modullaridan foydalanishingiz mumkin. *Ushbu modullar veb-xizmatlarni yaratish va HTTP so'rovlari va javoblarini qayta ishlash jarayonini soddalashtiradigan turli sinflar va izohlarni taqdim etadi.*

Veb-xizmatlar, shu jumladan SOAP va REST, Internet orqali turli tizimlar o'rtasida aloqa o'rnatish uchun ishlatiladigan texnologiyalar.

**SOAP** yoki **Simple Object Access Protocol** - *bu ilovalar o'rtasida tuzilgan ma'lumot almashish uchun protokol. U ma'lumotlar almashinuvi formati sifatida XML dan foydalanadi va ko'pincha korporativ ilovalarda qo'llaniladi.*

**REST** yoki **Representational State Transfer**, *veb-xizmatlarni yaratishda yengilroq yondashuvdir. Ilovalar o'rtasida ma'lumotlar va resurslarni uzatish uchun HTTP-dan foydalanadi. U ko'pincha veb va mobil ilovalarda qo'llaniladi.*

SOAP ham, REST ham Spring Boot bilan boshqa ilovalar tomonidan iste'mol qilinadigan veb-xizmatlarni yaratish uchun ishlatilishi mumkin. Ular turli dasturlash tillari va tizimlari oʻrtasida oʻzaro ishlash imkonini beradi, bu esa ilovalarning maʼlumotlar almashishi va almashishini osonlashtiradi.

* **Ba'zi misollar mavjud:**

**RESTful veb-xizmati misoli**: aytaylik, bizda foydalanuvchilarga mahsulotlarni **nomi**, **toifasi** va **narx** oralig'i bo'yicha qidirish imkonini beruvchi elektron tijorat ilovasi mavjud. Biz **HTTP GET** so'rovlarini mahsulot nomi, toifasi va narx oralig'i parametrlari bilan qabul qiladigan va JSON formatidagi mos mahsulotlar ro'yxatini qaytaradigan RESTful veb-xizmatini yaratishimiz mumkin.

SOAP veb-xizmati misoli: tasavvur qiling-a, bizda foydalanuvchilarga hisoblar o'rtasida pul o'tkazish imkonini beruvchi bank ilovasi mavjud. Biz hisob raqamlari, miqdori va tavsifi kabi tafsilotlar bilan o'tkazma so'rovini qabul qiladigan va o'tkazma muvaffaqiyatli yoki muvaffaqiyatli bo'lmaganligini ko'rsatuvchi javobni qaytaradigan SOAP veb-xizmatini yaratishimiz mumkin.

Ikkala misolda ham Spring Boot veb-xizmatning so'nggi nuqtalarini yaratish, kiruvchi so'rovlarni boshqarish va javoblarni belgilangan protokolga muvofiq formatlash uchun ishlatilishi mumkin (dam olish yoki sovun).

Veb-xizmatlar Spring Boot-da turli kutubxonalar va frameworklar yordamida osongina amalga oshirilishi mumkin. Masalan:

RESTful veb-xizmatlari uchun Spring Boot **Spring Web MVC** frameworkini taqdim etadi, bu esa ishlab chiquvchilarga RESTful API-larni osongina yaratishga imkon beradi. Bunga nazoratchi klassiga @**RestController** izohini qo'shish va HTTP usullari uchun so'rov mapping aniqlash orqali erishish mumkin.

SOAP veb-xizmatlari uchun Spring Boot Spring Web Services asosini taqdim etadi, bu esa ishlab chiquvchilarga SOAP so'nggi nuqtalarini osongina yaratishga imkon beradi. Bunga so'nggi nuqta sinfini Spring fasol sifatida belgilash va @Endpoint va @PayloadRoot izohlari yordamida so'nggi nuqta xaritasini sozlash orqali erishish mumkin.

Yuqoridagilarga qo'shimcha ravishda, Spring Boot JAX-WS, Apache CXF va Jersey kabi boshqa veb-xizmat texnologiyalarini qo'llab-quvvatlaydi, bu esa ishlab chiquvchilarga o'ziga xos talablari uchun eng yaxshi vositani tanlash imkonini beradi.

Spring Boot-da oddiy RESTful veb-xizmatini qanday yaratishga misol:



Ushbu misolda biz "/hello/{name}" so'nggi nuqtasida **HTTP GE**T so'rovlarini tinglaydigan RESTful veb-xizmatini aniqlaymiz. So'rov qabul qilinganda, "sayHello" meodi URL manzilidan olingan "name" parametri bilan chaqiriladi. Metod nom parametrini o'z ichiga olgan tabrik xabarini o'z ichiga olgan qatorni qaytaradi.

Umuman olganda, Spring Boot bilan veb-xizmatlardan (SOAP/REST) foydalanish Java ilovalarida veb-xizmatlarni yaratish va ulardan foydalanish jarayonini soddalashtirishi mumkin.

**Hibernate ORM**

Hibernate ORM- Object-Relational Mapping (ORM)(mulohazaviy xaritalash ramkasi.) - bu relyatsion ma'lumotlar bazalari bilan o'zaro aloqada bo'lish uchun Java-ga asoslangan ilovalar uchun ishlatiladigan ob'ekt- U Java ob'ektlarini ma'lumotlar bazasi jadvallariga va aksincha ko'rsatish orqali ma'lumotlar bazasiga kirishni soddalashtiradi. Hibernate bilan ishlab chiquvchilar past darajadagi SQL so'rovlarini yozishdan qochishlari va o'rniga yuqori darajadagi Java ob'ektlari bilan ishlashlari mumkin, bu esa yanada samarali va barqaror kodni yaratish imkonini beradi.

Hibernate keshlash, dangasa yuklash, optimistik qulflash va turli ma'lumotlar bazasi ishlab chiqaruvchilarini qo'llab-quvvatlash kabi bir qator xususiyatlarni qo'llab-quvvatlaydi. Shuningdek, u ishlab chiquvchilarga ob'ektga yo'naltirilgan sintaksisdan foydalangan holda ma'lumotlar bazasi so'rovlarini yozish imkonini beruvchi Hibernate Query Language (HQL) deb nomlangan so'rovlar tilini taqdim etadi.

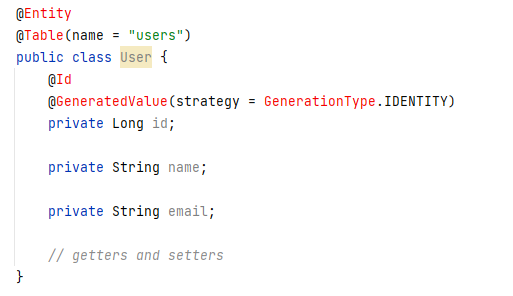
Hibernate-bu Java dasturlarida ma'lumotlar bazasini dasturlashni soddalashtiradigan ob'ekt-relyatsion mapping (ORM) doirasi*. Hibernate yordamida dasturchilar ma'lumotlar bazasi jadvallarini xuddi Java ob'ektlari bilan ishlayotgandek yaratishi va boshqarishi mumkin*, bu esa past darajadagi SQL so'rovlariga ehtiyojni yo'q qiladi.

Xulosa qilib aytganda, Hibernate ORM Java kodi va asosiy ma'lumotlar bazasi o'rtasida abstraksiya qatlamini ta'minlaydi, bu ma'lumotlar bazasiga asoslangan ilovalarni ishlab chiqish va saqlashni osonlashtiradi.

**Hibernate ORM ni ko'rsatishga yordam beradigan misol**:

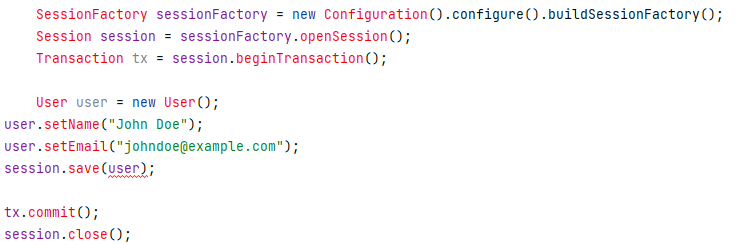
Aytaylik, bizda relyatsion ma'lumotlar bazasi bilan o'zaro aloqada bo'lishi kerak bo'lgan Java veb-ilovasi mavjud. SQL so'rovlarini qo'lda yozish o'rniga, biz Java klasslarini ma'lumotlar bazasi jadvallariga solishtirish va Java kodidan foydalangan holda CRUD (Yaratish, O'qish, Yangilash, O'chirish) operatsiyalarini bajarish uchun Hibernate ORM dan foydalanishimiz mumkin.

Birinchidan, biz ma'lumotlar bazasida jadvalni ko'rsatish uchun Java klassini yaratamiz. Aytaylik, bizda "ID", "name" va "email" ustunlari bo'lgan "foydalanuvchilar" deb nomlangan jadval mavjud. Bizning Java klassimiz quyidagicha ko'rinadi:



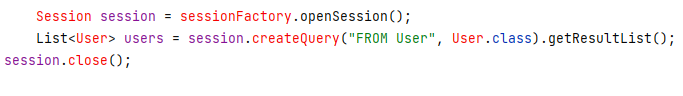
*Klassni doimiy ob'ekt sifatida belgilash uchun* @**Entity** izohidan va *jadval nomini belgilash uchun* @**Table** izohidan foydalanamiz. @**Id** izohi *id maydoni asosiy kalit ekanligini bildiradi* va @**GeneratedValue** *asosiy kalit qiymati avtomatik ravishda yaratilishini bildiradi*.

Keyinchalik, *sessiya yaratish va ma'lumotlar bazasi operatsiyalarini bajarish uchun* Hibernate'ning **SessionFactory**-dan foydalanishimiz mumkin. Mana yangi foydalanuvchi yaratish misoli:



Biz konfiguratsiya fayli yordamida avval **SessionFactory** yaratamiz va undan yangi Sessiyani ochish uchun foydalanamiz. Keyin biz yangi tranzaksiyani boshlaymiz, yangi Foydalanuvchi misolini yaratamiz, uning xususiyatlarini o'rnatamiz va **session.save**() usuli yordamida *ma'lumotlar bazasiga saqlaymiz*. Nihoyat, biz tranzaktsiyani bajaramiz va sessiyani yopamiz.

*Shuningdek, biz ma'lumotlar bazasini so'rash va ma'lumotlarni olish uchun Hibernate-dan foydalanishimiz mumkin*. Mana barcha foydalanuvchilarni olishning bir misoli:



Biz *yangi seans ochamiz*, Hibernate'ning **HQL (Hibernate Query Language)** yordamida *so'rov yaratamiz* va uni **session.createQuery**() yordamida bajaramiz. **getResultList**() metodi *foydalanuvchi ob'ektlari ro'yxatini qaytaradi*.

Umuman olganda, **Hibernate ORM** yuqori darajadagi <<object-relational mapping>> (ob'ektga aloqador xaritalash) tizimini taqdim etish orqali *ma'lumotlar bazasiga kirishni soddalashtiradi*, bu esa dasturchiga *SQL so'rovlarini to'g'ridan-to'g'ri yozish o'rniga* Java ob'ektlari bilan ishlash imkonini beradi.

**Project management -Loyihani boshqarish**

Loyihani *boshqarish muayyan loyiha maqsadlari va vazifalariga erishish uchun resurslarni rejalashtirish, tashkil etish va boshqarish jarayonini anglatadi*. U loyiha talablarini aniqlash, loyiha rejasini tuzish, resurslarni taqsimlash, taraqqiyotni kuzatish va loyihaning belgilangan hajm, byudjet va jadval doirasida bajarilishini ta'minlash uchun xavflarni boshqarish kabi tadbirlar majmuasini o'z ichiga oladi.

Loyihani samarali boshqarish loyihalarni o'z vaqtida, byudjet doirasida va kerakli sifatda bajarilishini ta'minlash uchun turli xil vositalar va usullardan foydalanishni o'z ichiga oladi. Ushbu vositalar va usullarga misollar loyihalarni rejalashtirish va kuzatish dasturlarini, risklarni boshqarish tizimlarini, manfaatdor tomonlarni boshqarish usullarini va aloqa rejalarini o'z ichiga oladi.

Loyihani boshqarish murakkab loyihalarni amalga oshiruvchi tashkilotlar uchun juda muhim, chunki bu ularga loyihalarni o'z vaqtida, byudjet doirasida va kerakli sifatda yakunlash imkonini beradi. Bu, shuningdek, loyiha muvaffaqiyatsizligi bilan bog'liq xavflarni kamaytirishga yordam beradi, jamoa a'zolari o'rtasidagi aloqa va hamkorlikni yaxshilaydi, loyihaning shaffofligi va hisobdorligini oshiradi.

Umuman olganda, loyihalarni boshqarish biznes operatsiyalarining muhim jihati bo'lib, u loyihalar muvaffaqiyatli yakunlanishini ta'minlaydi va kerakli natijalarga erishish uchun resurslarni samarali rejalashtirish, tashkil etish va boshqarishni talab qiladi.

**SOLID**

**SOLID** - *bu ob'ektga yo'naltirilgan dasturlashning besh tamoyilini ifodalovchi qisqartma bo'lib*, *ular yanada barqaror va kengaytiriladigan kodni yaratishga yordam beradi*. Bu tamoyillar:

**Yagona javobgarlik printsipi (SRP):** Klassda faqat bitta mas'uliyat yoki o'zgarish uchun sabab bo'lishi kerak.

**Ochiq/yopiq printsip (OCP):** Klass kengaytirish uchun ochiq, lekin o'zgartirish uchun yopiq bo'lishi kerak.

**Liskov almashtirish printsipi (LSP):** Subtiplar dasturning to'g'riligini o'zgartirmasdan ularning asosiy turlari bilan almashtirilishi kerak.

**Interfeyslarni ajratish printsipi (ISP):** Klass o'zlari foydalanmayotgan interfeyslarni amalga oshirishga majburlanmasligi kerak.

**Dependency Inversion Principle (DIP):** Yuqori darajadagi modullar past darajadagi modullarga bog'liq bo'lmasligi kerak. Ikkalasi ham abstraktsiyalarga bog'liq bo'lishi kerak.

Ushbu tamoyillar Java Spring Boot ilovalariga quyidagi usullarda qo'llanilishi mumkin:

**SRP**: Har bir klass faqat bitta mas'uliyatga ega bo'lishi kerak va har qanday qo'shimcha funktsiyalar alohida klassda amalga oshirilishi kerak. Masalan, *boshqaruvchi* klassi faqat HTTP so'rovlarini bajarishi va biznes mantiqini xizmat klassiga topshirishi kerak.

**OCP**: Mavjud kodni o'zgartirish o'rniga, mavjud klasslarni kengaytirish yoki yangi interfeyslarni amalga oshirish orqali yangi funksiyalar qo'shilishi kerak. Bunga Spring Boot-ning **bean** in'ektsiyasi va avtokonfiguratsiya xususiyatlari yordamida erishish mumkin.

**LSP**: Kichik klasslar muammo tug'dirmasdan o'zlarining ota-onalar klasslarini almashtirishlari kerak. Misol uchun, agar service klassi interfeysga bog'liq bo'lsa, ushbu interfeysni amalga oshiradigan har qanday klass muammo tug'dirmasdan bog'liqlik sifatida kiritilishi mumkin.

**ISP**: Klasslar faqat kerakli interfeyslarni amalga oshirishi kerak va har qanday qo'shimcha interfeyslar alohida klasslarda amalga oshirilishi kerak. Bu keraksiz bog'liqlikni kamaytirishga va xizmat ko'rsatishni yaxshilashga yordam beradi.

**DIP**: Yuqori darajadagi modullar (masalan, boshqaruvchilar) aniq amalga oshirishga emas, balki abstraktsiyalarga (masalan, interfeyslarga) bog'liq bo'lishi kerak. Bu komponentlarni ajratish va tizimni yanada moslashuvchan va kengaytiriladigan qilishga yordam beradi.

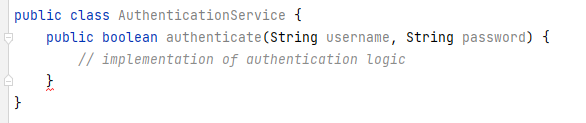
**Umuman olganda, ushbu SOLID tamoyillariga rioya qilish yanada barqaror va kengaytiriladigan Java Spring Boot ilovalarini yaratishga yordam beradi.**

Java Spring Boot ilovasida SOLID tamoyillarini qanday qo'llash mumkinligiga misol:

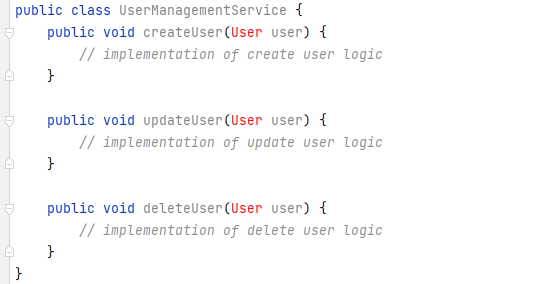
Keling, yagona mas'uliyat tamoyilini (**SRP**) ko'rib chiqaylik, unda klassni o'zgartirish uchun faqat bitta sabab bo'lishi kerak. Boshqacha qilib aytganda, **klass faqat bitta mas'uliyatga ega bo'lishi kerak**.

Bizda foydalanuvchi *autentifikatsiyasi* va *foydalanuvchi boshqaruvi* uchun mas'ul bo'lgan **UserService** klassimiz bor deylik. Bu **SRP**ni buzadi, chunki klass bir nechta mas'uliyatga ega. *Autentifikatsiya* va *foydalanuvchi boshqaruvi* mas'uliyatini alohida klasslarga ajratish uchun biz ushbu klassni qayta tahrirlashimiz mumkin.

*Birinchidan*, foydalanuvchi autentifikatsiyasini boshqaradigan **AuthenticationService** klassni yaratamiz:



Keyinchalik, foydalanuvchi boshqaruvini boshqaradigan **UserManagementService** klassini yaratamiz:



Nihoyat, biz ushbu yangi klasslardan foydalanish uchun **UserService** klassini yangilaymiz:



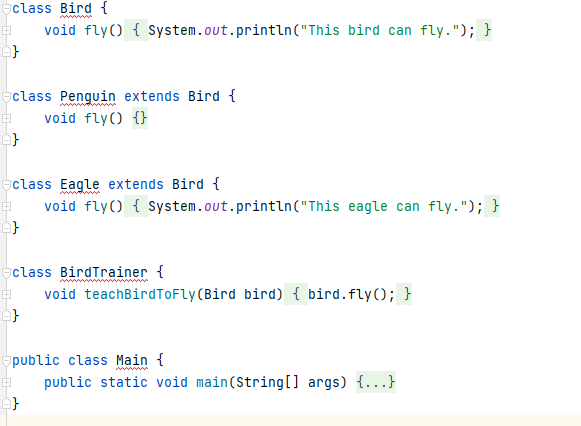
Endi **UserService** klassida faqat bitta mas'uliyat bor, ya'ni autentifikatsiya va foydalanuvchilarni boshqarish xizmatlarini tashkil qilish. Autentifikatsiya va foydalanuvchilarni boshqarish xizmatlari endi SRP ga rioya qiladigan tegishli majburiyatlari uchun javobgardir.

* **Liskov almashtirish printsipi (LSP):** ushbu printsipga ko'ra*, yuqori klass ob'ektlari dasturning to'g'riligini buzmasdan kichik klass ob'ektlari bilan almashtirilishi kerak*. Boshqacha qilib aytganda, bola klass dasturning funksionalligini buzmasdan ota-klassni almashtirishi kerak.

Pastagi misolda bizda ikkita kichik sinf **Pingvin** va **Eagle** bo'lgan **Qushlarning** super klassi mavjud. **Pingvin** ham, **Eagle** ham **Birddan** fly**()** funksiyasini meros qilib oladi. Biroq, pingvinlar ucha olmasligi sababli, **Penguin** klassi hech narsa qilmaslik uchun **fly**() metodini bekor qiladi.

Shuningdek, bizda **Bird** obyektini parametr sifatida qabul qiluvchi **teachBirdToFly**() funksiyasiga ega **BirdTrainer** klassi mavjud. Funksiya oddiygina **Bird** ob'ektidagi **fly**() funksiyasini chaqiradi.

Asosiy funksiyada biz **BirdTrainer** misolini va ikkita **Qush** ob'ektini - **burgut** va **pingvinni** yaratamiz. Keyin ikkala ob'ektda **teachBirdToFly**() funksiyasini chaqiramiz. **Eagle** *ucha olganligi sababli*, chiqish "Bu burgut ucha oladi". Biroq,Pingvin ucha olmasligi sababli, "Bu qush ucha oladi".



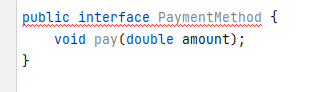
* **Interfeys izolyatsiyasi printsipi (ISP):** bu *tamoyil mijoz foydalanmayotgan interfeyslarga bog'liq bo'lmasligi kerakligini bildiradi*. Boshqacha qilib aytganda, hamma narsani qamrab oladigan bitta katta interfeysga ega bo'lishdan ko'ra, mijozlar ehtiyojlariga xos bo'lgan bir nechta kichik interfeyslarga ega bo'lish yaxshiroqdir.

Ushbu misolda bizda **harakatlanish**, **ovqatlanish** va **uxlash** funksiyalari bilan **Hayvonlar** **interfeysi** mavjud. Keyin biz mos ravishda ucha oladigan **qushlar** va **ovqatlanadigan** hayvonlar uchun **Flyable** va **Eatable** kichikroq interfeyslarni yaratamiz. **Bird** klassi **Animal** va **Flyable** interfeyslarini, **Penguin** klassi esa **Animal** va **Eatable** interfeyslarini amalga oshiradi. Bu ko'proq moslashuvchanlik va kodni yaxshiroq tashkil qilish imkonini beradi, chunki endi kerak bo'lganda hayvonlarning turli xatti-harakatlari uchun yangi interfeyslarni qo'shishimiz mumkin.



* **Open-Closed:** tamoyilini amalga oshirishga misol:

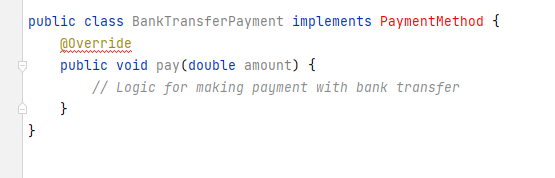
Aytaylik, bizda to'lovni amalga oshirish uchun yagona to'lov metodi mavjud **PaymentMethod** interfeysi mavjud. Biz yangi toʻlov usuli qoʻshmoqchimiz, lekin mavjud kodimiz oʻzgarmasligiga ishonch hosil qilmoqchimiz.



Endi biz **PaymentMethod** interfeysining ikkita ilovasini yaratamiz, **CreditCardPayment** va **PaypalPayment**.



Keling, yangi **BankTransferPayment** to'lov metodini qo'shishimiz kerak deb faraz qilaylik. Mavjud **PaymentMethod** interfeysi va uni amalga oshiradigan klasslarni o'zgartirish o'rniga biz quyida ko'rsatilgandek **PaymentMethod** interfeysining yangi ilovasini yaratamiz:

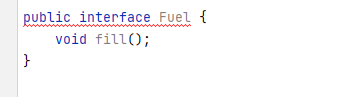


Endi biz yangi **BankTransferPayment** dasturini kodimizda mavjud kodni o'zgartirmasdan foydalanishimiz mumkin va shu bilan **Ochiq-yopiq** tamoyilga amal qilamiz.

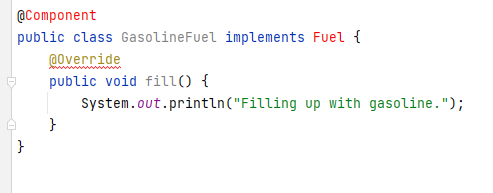
**Dependency Inversion Principle (DIP):** bu tamoyil *yuqori darajadagi modullar quyi darajadagi modullarga bog'liq bo'lmasligi kerakligini bildiradi*. *Ikkalasi ham abstraktsiyalarga bog'liq bo'lishi kerak*. *Abstraktsiyalar tafsilotlarga bog'liq bo'lmasligi kerak. Tafsilotlar abstraktsiyalarga bog'liq bo'lishi kerak*.

Misol uchun, bizda ishga tushirilishi yoqilg'i sarfiga bog'liq bo'lgan avtomobillar sinfiga egamiz. Biz Yoqilg'i interfeysini yaratishimiz va ma'lum bir dastur o'rniga Avtomobil sinfini unga bog'liq qilishimiz mumkin. Shunday qilib, kelajakda biz avtomobil sinfini o'zgartirmasdan, turli xil yoqilg'iga osongina o'tishimiz mumkin.

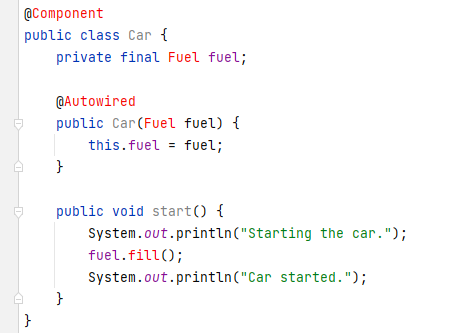
Birinchidan, biz yoqilg'i interfeysini aniqlaymiz:



Keyinchalik, **Fuel** interfeysini amalga oshiradigan **GasolineFuel** klassini yaratamiz:

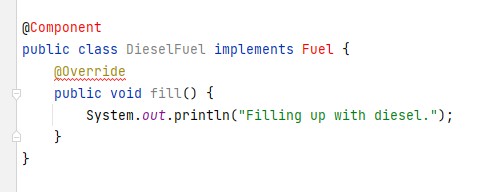


Keyin, biz **Fuel** interfeysiga bog'liq bo'lgan **Avtomobil** klassini yaratamiz:



Yuqoridagi kodda biz **Car** *konstruktoriga yoqilg'iga bog'liqlikni kiritish uchun* @**Autowired** izohidan foydalanamiz. Buni amalga oshirish orqali **Car** klassi ma'lum bir dasturga emas, balki **Fuel** interfeysiga bog'liq. Bu bizga **Car** klassini o'zgartirmasdan **Fuel** interfeysining boshqa ilovasiga osongina o'tish imkonini beradi.

Misol uchun, agar biz **Fuel** interfeysini amalga oshiradigan **DieselFuel** nomli yangi klass yaratsak, **DieselFuel** klassidagi @**Component** izohini o'zgartirish orqali uni **Car** klassida ishlatishga o'tishimiz mumkin:



Buni amalga oshirish orqali **Car** klassi endi **Car** klassining o'zida hech qanday o'zgarishsiz **GasolineFuel** ilovasi o'rniga **DieselFuel** ilovasidan foydalanadi.

**Spring MVC**

**Spring MVC (Model-View-Controller)** *Java veb-ilovalarini yaratish uchun mashhur veb- frameworkdir.* *U MVC pattern(naqsh)ga asoslangan veb-ilovalarni tashkil qilish va tuzish funksiyasini taqdim etadi*, bu dasturni uchta asosiy komponentga ajratadi: **Model**, **View** va **Controller**.

**Model** - *ma'lumotlar va ilovaning biznes mantiqini ifodalaydi*. **View** *ma'lumotlarni foydalanuvchiga qulay tarzda taqdim etish uchun javobgardir*. **Controller** - *foydalanuvchi so'rovlarini ko'rib chiqadi va dastur oqimini boshqaradi.*

Spring MVC HTTP so'rovlari va javoblarini boshqarish, URL manzillarini kontrollerlarga mapping qilish va **View** larni ko'rsatish uchun **DispatcherServlet**, **HandlerMapping**, **ViewResolver** va boshqalar kabi turli xil komponentlarni taqdim etadi. Shuningdek, u **View** larni ko'rsatish uchun **JSP**, **Thymeleaf** va boshqalar kabi turli xil shablon texnologiyalarini qo'llab-quvvatlaydi.

Ishlab chiquvchilar Spring MVC-dan osongina sinovdan o'tkazilishi va o'rnatilishi mumkin bo'lgan kengaytiriladigan, modulli va texnik xizmat ko'rsatadigan veb-ilovalarni yaratish uchun foydalanishlari mumkin.

*Aytaylik, bizda kitoblar ro'yxatini ko'rsatadigan oddiy veb-ilova mavjud.* **Sarlavha**, **muallif** va **nashr sanasi** kabi xususiyatlarga ega **Kitob** klassini yaratishimiz mumkin. Keyin biz kitoblar ro'yxatini boshqaradigan **BookService** klassini va kiruvchi veb-so'rovlarni bajaradigan va **BookService** bilan o'zaro aloqada bo'lgan **BookController** klassini yaratishimiz mumkin.



* CI/CD

CI/CD – (Continuous Integration)*uzluksiz joylashtirish degan ma’noni anglatadi*, *bu kod o‘zgarishlarini yaratish, sinovdan o‘tkazish va joylashtirish jarayonini avtomatlashtirish uchun dasturiy ta’minotni ishlab chiqishda foydalaniladigan amaliyotlar to‘plamidir.*

**CI(Continuous Integration)** - *bu kod o'zgarishlarini umumiy omborga tez-tez integratsiyalash va uning funktsional va xatosiz ekanligiga ishonch hosil qilish uchun kodni avtomatik ravishda yaratish va sinovdan o'tkazish amaliyotini anglatadi. Bu har qanday muammoni ishlab chiqish jarayonida erta aniqlashni ta'minlaydi, xatolar va boshqa xatolar ehtimolini kamaytiradi.*

**Continuous Integration** dasturchilarga yangi xususiyatlar va yangilanishlarni foydalanuvchilarga tez va xavfsiz ravishda chiqarish imkonini beruvchi ishlab chiqarish muhitiga kod oʻzgarishlarini avtomatik ravishda joylashtirish amaliyotini nazarda tutadi. Bu ishlamay qolish vaqtini minimallashtiradi va tezroq iteratsiya va kod o'zgarishlari haqida fikr-mulohazalarni bildirish imkonini beradi.

CI/CD amaliyotlari birgalikda *ishlab chiqish jarayonining* asosiy qismlarini avtomatlashtirish hamda kod o‘zgarishlarining sinchiklab tekshirilishi va o‘z vaqtida va xavfsiz tarzda joylashtirilishini ta’minlash orqali jamoalarga yuqori sifatli dasturiy ta’minotni tezroq va samarali yetkazib berishga yordam beradi.

Ko'rib turganingizdek, CI/CD quvur liniyasi bir necha bosqichlardan iborat:

**SCM**: **Source Control Management**(Manbani boshqarish boshqaruvi) Bu yerda dasturchilar **Git** yoki **SVN** kabi kod o'zgarishlarini saqlaydilar.

Subversion degan ma'noni anglatuvchi **SVN** bu fayl va kataloglarga kiritilgan o'zgarishlarni kuzatib borish uchun dasturchilar tomonidan foydalaniladigan dasturiy vositadir.

Buni kodingiz uchun vaqt mashinasi sifatida tasavvur qiling. Bu bir nechta odamlarga bir-birining o'zgarishlarini tasodifan yozmasdan bitta kod bazasida ishlashga imkon beradi.

**SVN** faylga kiritilgan har bir oʻzgartirishni, oʻzgartirishni kim amalga oshirganini va qachon kiritilganligini kuzatib boradi. Bu vaqt o'tishi bilan koddagi o'zgarishlar tarixini ko'rishni osonlashtiradi.

**SVN**, shuningdek, dasturchilarga kodlar bazasining " branches " yaratishga imkon beradi, ular mustaqil ishlash mumkin bo'lgan kodning alohida nusxalari. Bu bir vaqtning o'zida bir nechta odam loyihaning turli funktsiyalari ustida ishlayotganda foydali bo'lishi mumkin.

Umuman olganda, **SVN** dasturchilarga loyihalarda samaraliroq va kamroq xatolar bilan hamkorlik qilishga yordam beradi va vaqt o'tishi bilan kodlar bazasidagi o'zgarishlarni kuzatishni osonlashtiradi.

**Uzluksiz integratsiya** (**CI**): Agar dasturchi kod o'zgarishlarini **SCMga** kiritsa, **Jenkins**, **TravisCI** yoki **CircleCI** kabi **CI** server avtomatik ravishda kod o'zgarishlarini tuzadi va sinab ko'radi. Agar kod sinovlardan o'tsa, u foydalanishga tayyor.

**Jenkins** va **TravisCI** ikkalasi ham dasturiy ta'minotni yaratish, sinovdan o'tkazish va joylashtirish jarayonini *avtomatlashtiradigan* uzluksiz integratsiya (CI) serverlaridir.

**Integratsiya** deganda *bir butunlikni yaratish uchun turli qismlar yoki komponentlarni birlashtirish tushuniladi.* Dasturiy ta'minotni ishlab chiqishda integratsiya turli dasturchilar tomonidan kiritilgan kod o'zgarishlarini yagona dasturiy ta'minot ilovasiga **birlashtirishni** anglatadi. Integratsiya, shuningdek, o'zaro bog'langan tizimlarni yaratish uchun dasturiy ta'minotni boshqa texnologiyalar bilan birlashtirishni ham nazarda tutishi mumkin.

Integratsiya dasturiy ta'minotni ishlab chiqishning turli darajalarida, ya'ni alohida funktsiyalar yoki modullarni integratsiyalashdan butun tizimlar yoki ilovalarni integratsiya qilishgacha bo'lishi mumkin. Bu dasturiy ta'minotni ishlab chiqish jarayonining muhim jihati hisoblanadi, chunki u dasturiy ta'minotning turli qismlarini mo'ljallangan tarzda birgalikda ishlashini va ilovaning funktsional va ishlash talablariga javob berishini ta'minlashga yordam beradi.

Integratsiya shuningdek*, veb-ilovani ma'lumotlar bazasi bilan integratsiyalash yoki mobil ilovani bulutli xizmat bilan integratsiya qilish kabi dasturiy ilovalarni boshqa tizimlar yoki texnologiyalar bilan integratsiyalash jarayonini ham anglatishi mumkin*. *Ushbu turdagi integratsiya zamonaviy korxonalar va tashkilotlarning ehtiyojlarini qondira oladigan murakkab, o'zaro bog'liq tizimlarni yaratish uchun juda muhimdir*.

**Jenkins** - bu ochiq manbali **CI** **server** bo'lib, u *bir nechta platformalarda dasturiy ta'minotni yaratish, sinovdan o'tkazish va joylashtirishni avtomatlashtirish uchun ishlatilishi mumkin*. U juda ko'p turli xil plaginlarni qo'llab-quvvatlaydi, bu uni juda sozlanishi va kengaytirilishiga imkon beradi.

**TravisCI** - *bulutga asoslangan CI serveri bo'lib, u GitHub omborlarida joylashtirilgan kodni sinab ko'rish va joylashtirishga qaratilgan*. U **CI** quvurlarini sozlash uchun oddiy va ishlatish uchun qulay interfeysni taqdim etadi va u katta ish yuklarini boshqarish uchun avtomatik ravishda o'lchaydi.

**Jenkins** ham, **TravisCI** ham dasturchilarga o'zlarining dasturiy ta'minotini qanday qurish, sinovdan o'tkazish va joylashtirish kerakligini ko'rsatadigan quvurlarni aniqlashga imkon beradi. Ushbu quvurlar **Git** kabi manbalarni boshqarish (SCM) vositasiga kod yuborilganda avtomatik ravishda ishga tushirilishi yoki dasturchi tomonidan qo'lda ishga tushirilishi mumkin.

**Jenkins** va **TravisCI** kabi **CI** **serverlari** kod o'zgarishlari haqida tezkor fikr-mulohazalarni taqdim etish, kodni yaratish va sinovdan o'tkazish jarayonini avtomatlashtirish va joylashtirish jarayonini soddalashtirish orqali dasturiy ta'minotni ishlab chiqish jarayonini soddalashtirishga yordam beradi. Umuman olganda, ushbu vositalar ishlab chiquvchilarga yuqori sifatli dasturiy ta'minotni tezroq va kamroq xatolar bilan chiqarishga yordam beradi.

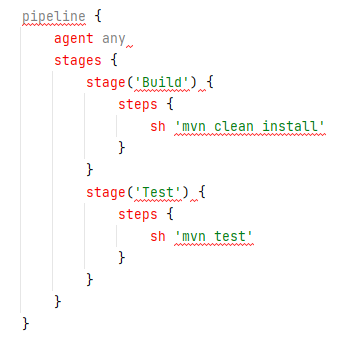
**Uzluksiz yetkazib berish (CD- Continuous Delivery):** CI serveri kod o'zgarishlarini artefaktga to'playdi va **QA** jamoasi tomonidan yakuniy sinovdan o'tkazish uchun uni sahnalash muhitiga joylashtiradi.

**Uzluksiz joylashtirish (CD- Continuous Deployment):** Agar kod o'zgarishlari yakuniy testlardan o'tsa, **CI** server kod o'zgarishlarini ishlab chiqarish muhitiga joylashtiradi va ularni oxirgi foydalanuvchilarga taqdim etadi.

Ushbu bosqichlarni avtomatlashtirish orqali **CI/CD** dasturchilarga minimal inson aralashuvi bilan kod o'zgarishlarini tez ishlab chiqish, sinab ko'rish va joylashtirish imkonini beradi. Bu dasturiy ta'minot yangilanishlarini chiqarish uchun zarur bo'lgan vaqt va harakatni sezilarli darajada qisqartirishi va kod har doim funktsional va dolzarb bo'lishini ta'minlashi mumkin.

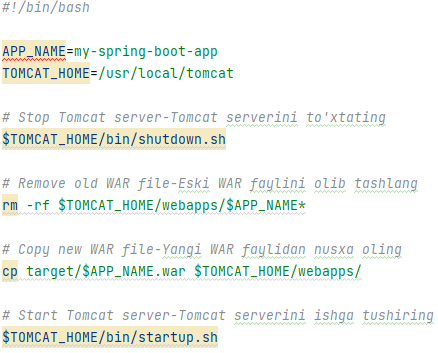
Spring Boot-dan foydalangan holda CI/CD-ga misol:

**1.Uzluksiz integratsiya** (**CI**): Agar dasturchi **Git** kabi manbalarni boshqarish (**SCM**) vositasiga kod o'zgarishlarini kiritsa, **Jenkins** yoki **TravisCI** kabi **CI** server avtomatik ravishda kod o'zgarishlarini tuzadi va sinab ko'radi. CI quvur liniyasini belgilaydigan **Jenkinsfile** misoli:



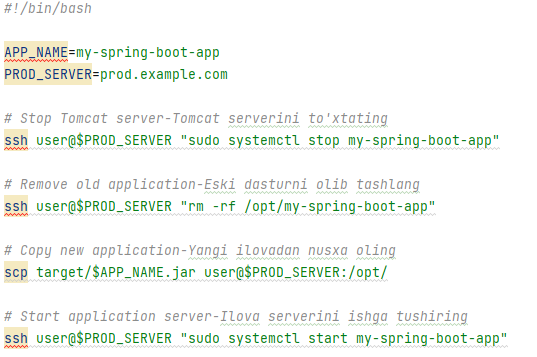
Ushbu quvur liniyasi ikki bosqichni belgilaydi: qurish va sinovdan o'tkazish. Qurilish bosqichida quvur liniyasi Spring Boot ilovasini quradigan va bajariladigan **JAR** faylini yaratadigan **mvn clean install** buyrug'ini bajaradi. Test bosqichida quvur liniyasi dastur uchun barcha avtomatlashtirilgan testlarni bajaradigan **mvn test** buyrug'ini bajaradi.

**2.Uzluksiz yetkazib berish** (**CD**): **CI serveri** kod o'zgarishlarini artefaktga to'playdi va QA jamoasi tomonidan yakuniy sinovdan o'tkazish uchun uni sahnalash muhitiga joylashtiradi. Bu erda Spring Boot dasturini *Tomcat serveriga* o'rnatadigan *skript* misoli keltirilgan



Ushbu skript Tomcat serverini to'xtatadi, ilovaning eski versiyalarini o'chiradi, ilovaning yangi versiyasini (CI quvur liniyasi tomonidan yaratilgan) nusxalaydi va Tomcat serverini qayta ishga tushiradi.

**3.Uzluksiz joylashtirish (CD):** Agar kod o'zgarishlari yakuniy testlardan o'tsa, CI server kod o'zgarishlarini ishlab chiqarish muhitiga joylashtiradi va ularni oxirgi foydalanuvchilarga taqdim etadi. Spring Boot ilovasini ishlab chiqarish serveriga joylashtiradigan skriptga misol:



Ushbu skript dastur **serverini to'xtatadi**, **ilovaning eski versiyalarini o'chiradi**, ilovaning **yangi versiyasini (***CI quvur liniyasi tomonidan yaratilgan***) nusxalaydi** va **dastur serverini qayta ishga tushiradi**.

Umuman olganda, ushbu misol CI/CD-dan Spring Boot ilovalarini yaratish, sinovdan o'tkazish va joylashtirish jarayonini avtomatlashtirish uchun qanday foydalanish mumkinligini ko'rsatadi, bu esa dasturchilarga yangi xususiyatlarni tezroq va kamroq xatolar bilan chiqarishga yordam beradi.

* **Dependency injection**

**Dependency injection** (**DI**) - *dasturiy ta'minotni ishlab chiqishda ishlatiladigan dizayn namunasi bo'lib, kodning moslashuvchanligi va barqarorligini oshirishga qaratilgan.* **Dependency injection** qilishning asosiy g'oyasi tizimning tarkibiy qismlari o'rtasidagi to'g'ridan-to'g'ri bog'liqliklarni har bir komponentning o'ziga bog'liqliklarini yaratish yoki olish o'rniga tashqi vositalar orqali ularning bog'liqliklarini kiritish orqali olib tashlashdir.

Boshqacha qilib aytadigan bo'lsak, *ob'ekt o'zining bog'liqligini yaratish o'rniga, u o'zining bog'liqligini ta'minlash uchun tashqi mexanizmga tayanadi*. Ushbu tashqi mexanizm odatda ob'ektlarni yaratish va kiritishni boshqaradigan framework yoki konteynerdir.

**Dependency injection**ning **afzalliklari** **kodni qayta ishlatish**, **sinovdan o'tkazish** va **texnik xizmat ko'rsatishni yaxshilashni** o'z ichiga oladi. Bu butun dasturni o'zgartirmasdan komponentlar yoki modullarni almashtirish orqali ilovaning harakatini o'zgartirishni osonlashtiradi. Bundan tashqari, u ko'proq modulli kodga imkon beradi, chunki komponentlar mustaqil ravishda ishlab chiqilishi va sinovdan o'tkazilishi mumkin.

* **Dependency injection**ning uch turi mavjud:

**Constructor Injection**: *Bog'liqliklar uning konstruktori orqali komponentga uzatiladi.*

**Setter Injection**: Bog'liqliklar komponentga setter metodlari orqali uzatiladi.

**Interfeys injection**: Bog'liqliklar,komponentga komponent amalga oshiradigan interfeys orqali uzatiladi.

Umuman olganda, **Dependency injection** qilish ***moslashuvchan*** *va* ***barqaror*** *dasturiy ta'minot tizimlarini yaratish uchun kuchli texnikadir*.

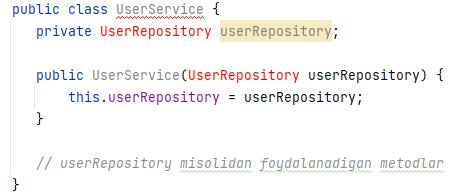
**Dependency injection**ni ko'rsatishga yordam beradigan misol:

Aytaylik, bizda veb-ilovada *foydalanuvchi* ma'lumotlarini boshqarish uchun mas'ul bo'lgan **UserService** deb nomlangan klass mavjud. Bu klass *foydalanuvchi* ma'lumotlarini **olish** va **saqlash** uchun **UserRepository**-ga bog'liq. **DI** bo'lmagan dasturda **UserService** klassi **UserRepository** klassining o'zi namunasini yaratish uchun javobgar bo'ladi:



Biroq, bu **UserService** va **UserRepository** klasslari o'rtasida qattiq bog'lanishni keltirib chiqaradi, bu **UserRepository** dasturini amalga oshirishni o'zgartirishni yoki **UserService**-ni alohida sinovdan o'tkazishni qiyinlashtiradi.

**Dependency injection** bilan biz **UserService** va **UserRepository** klasslarini ajratishimiz mumkin. **UserRepository** nusxasini to'g'ridan-to'g'ri **UserService**-da yaratish o'rniga, biz **UserService**-ga uning konstruktori orqali UserRepository nusxasini kiritamiz:



Shunday qilib, ***UserService*** *klassi o'z konstruktori orqali uzatiladigan* ***UserRepository*** *interfeysining har qanday ilovasidan foydalanishi mumkin*. Misol uchun, ishlab chiqarish kodida biz ma'lumotlar bazasiga kiradigan dasturdan foydalanishimiz mumkin, sinov paytida esa qattiq kodlangan foydalanuvchi ma'lumotlarini qaytaradigan soxta dasturdan foydalanishimiz mumkin.

Mana, Spring kabi **dependency injection** tizimidan foydalangan holda **UserRepository**-ning o'ziga xos ilovasi bilan **UserService** namunasini qanday yaratishimiz mumkinligiga misol:



**Ushbu misolda biz** **UserService** va **UserRepository** uchun **Bean** yaratadigan Spring konfiguratsiya klassini aniqlaymiz. **userRepository**() **Bean**ni **userService**() metodiga o'tkazish orqali **UserService** **UserRepository**-ga bog'liqligini aniqlaymiz. Ilovani ishga tushirganimizda, Spring **UserRepository** **DatabaseUserRepository** ilovasi bilan **UserService** namunasini yaratadi.

**Bean**

* **@Bean**-bu Springga "**bean**" deb nomlangan ob'ekt yaratish va uni siz uchun boshqarishni aytishning bir metodi. Bu sizning ilovangizning turli qismlarida ob'ektdan foydalanishni osonlashtiradi.

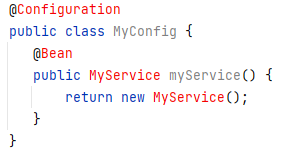
Misol uchun, agar sizda elektron pochta xabarlarini yuborish kabi ba'zi ishlarni bajaradigan klasslaringiz bo'lsa, *klassning namunasini yaratish va unga nom berish uchun* **@Bean**-*dan foydalanishingiz mumkin*. Keyin, siz **@Bean**ni olish uchun **@Autovired** izohidan foydalanishingiz va uni **kontroller** *kabi ilovangizning boshqa qismlarida ishlatishingiz mumkin.*

Bu ob'ekt yaratish va uni kodingizning boshqa qismlarida ishlatishga imkon beradigan yorliqqa o'xshaydi. Shuningdek, u ob'ektni ishlatishdan oldin sozlash imkonini beradi.

Xonani spring konteyner va xonadagi odamlarni **Bean** deb tasavvur qiling. Kimdir eshikdan kirdida menga "*Talaba*" kerak dedi. lekin xonada 2 yoki undan ko'p talaba bor shunda sizga aniqlashtirish uchun nimadir kerak bo'ladi, misol uchun "*Moliyada o'qiydigan talaba*". O'sha ishni **Qualifier** annotatsiyasi orqali qiladi

* **@Bean**-*ixtiyoriy turdagi ob'ektlar dastur ishga tushishi bilan ob’ektlar konteyneriga yig'iladi*,shu konteynerdan *shu tipdagi obekt qayerga kerak bo’ladigan bo’lsa bitta ob’ektni hamma* ***@Autowired*** *qilingan joyga chaqirib beradi,yani har safar qaysidir classda obekt kerak bo’ladigan bo’lsa yangi obekt ochmasdan hamma obektlarni birdan yig'ib olib keyin so’ralgan joylarga tarqatib beradi.*
* **@Bean**-bu Spring Boot-dagi metod darajasidagi izoh, bu metod spring konteynerini boshqarish uchun **bean** komponentini yaratishini ko'rsatish uchun ishlatiladi. Class **@Configuration** bilan izohlanganda, **@Bean** bilan izohlangan metodlar spring konteyneri tomonidan **bean** **komponentlarini** *yaratish* va *ularni dastur kontekstida komponent sifatida ro'yxatdan o'tkazish* *uchun qayta ishlanadi*. Bu Spring dasturidagi ob'ektlarni osongina sozlash va boshqarish imkonini beradi.

Misol uchun, Agar sizda "**MyService**" deb nomlangan klass bo'lsa va uni ilovangizning boshqa qismlariga osongina kiritish uchun uni **bean** komponenti sifatida sozlashni xohlasangiz, siz **@Configuration** klassini yaratishingiz va misollar yaratadigan metodni qo'shishingiz kerak.va **@Bean** izohli **MyService** misolini qaytaradi.



**MyService** metodi Spring tomonidan **MyService** namunasini yaratish va uni dastur kontekstida **bean** sifatida ro'yxatdan o'tkazish uchun chaqiriladi. Keyin ushbu **beanni** ilovangizning boshqa qismlariga kiritish uchun **@Autowired** dan foydalanishingiz mumkin.

**Microservices**

***Mikroservislar*** *dasturiy ta'minot ilovalarini yaratishning arxitekturaviy yondashuvi bo'lib, u* ***katta dasturni kichikroq****,* ***mustaqil ravishda joylashtiriladigan xizmatlarga bo'lib****,* ***API kabi yengil mexanizmlar orqali bir-biri bilan bog'lanishi mumkin*.**

Har bir **mikroservis** ilova doirasida muayyan funktsiyani bajarish uchun mo'ljallangan va o'zining ma'lumotlarini saqlash, qayta ishlash va boshqa xizmatlar bilan aloqa qilish uchun javobgardir. Ushbu yondashuv an'anaviy monolit arxitekturalar bilan solishtirganda ko'proq moslashuvchanlik, kengayish va barqarorlikni ta'minlaydi.

Spring Boot-da **mikroservislar** turli xil frameworklar va vositalar yordamida amalga oshirilishi mumkin, *masalan*, **Spring Cloud**, **mikroservislarga** asoslangan ilovalarni ishlab chiqishni **soddalashtirish** uchun **tools**(asboblar) va **kutubxonalar** to'plamini taqdim etadi. **Spring Boot** va **Spring Cloud** yordamida dasturchilar mikroservislarni osongina **yaratishi**, **joylashtirishi** va **boshqarishi** va **REST API** yoki xabarga asoslangan protokollar yordamida bir-biri bilan muloqot qilishiga imkon beradi.

Aytaylik, siz **foydalanuvchilarga** **mahsulotlarni** **ko‘rib chiqish**, **ularni savatga** qo‘**shish** **va to‘lovni amalga oshirish imkonini beruvchi elektron tijorat ilovasini yaratmoqdasiz**. *An'anaviy monolit arxitekturada ushbu funktsiyalarning barchasi bitta dasturda birlashtirilgan*.

Biroq, **mikroservislar** **arxitekturasi** bilan siz **ilovani kichikroq**, **mustaqil** ravishda joylashtiriladigan xizmatlarga bo'lishingiz mumkin. **Misol uchun**, sizda **mahsulot ma'lumotlarini olish** va **ko'rsatish** bilan shug'ullanadigan mahsulot katalogi **servicesi**, *savatdan narsalarni* **qo'shish** va **olib tashlashni** boshqaradigan **xarid qilish** savatchasi **servicesi** va **to'lovni qayta ishlash** bilan shug'ullanadigan **to'lov** **servicesi** bo'lishi mumkin.

Ushbu **servicelar** har biri boshqa **serviceslar** bilan aloqa o'rnatish uchun o'z **API**-ga ega bo'ladi, masalan, **xarid qilish** **savatchasi** **servicesi** mahsulot **ma'lumotlarini olish uchun** mahsulot katalogi **servicesi**ga chaqirish va **xarid qilish** **servicesi** **tovarlarni olish** uchun **savat** **servicesi**ni chaqirish qobiliyatiga ega bo'ladi. savatda.

Ilovani **kichikroq**, **ko'proq** yo'naltirilgan **serviceslar**ga bo'lish orqali siz **yaxshi miqyoslilik**, **nosozliklarni izolyatsiya qilish** va texnologiya tanlashda **moslashuvchanlik** kabi afzalliklarga ega bo'lasiz. Bundan tashqari, siz butun ilovani qayta joylashtirishni talab qilmasdan, har bir **servicesni** mustaqil ravishda joylashtirishingiz va yangilashingiz mumkin. Bu vaqt o'tishi bilan ilovani takrorlashni va yaxshilashni osonlashtiradi.

Bu yerda mahsulot ma'lumotlarini **oladigan** va **ko'rsatadigan** **mikroservisning** oddiy Spring Boot misoli:

1.Birinchidan, Spring Initializr kabi vosita yordamida yangi Spring Boot loyihasini yaratasiz.

2.Keyin, loyihangizga **RESTful API** yaratish uchun **Spring Web** va maʼlumotlar bazasi bilan oʻzaro ishlash uchun **maʼlumotlar bazasi drayveri** kabi zarur bogʻliqliklarni qoʻshasiz.

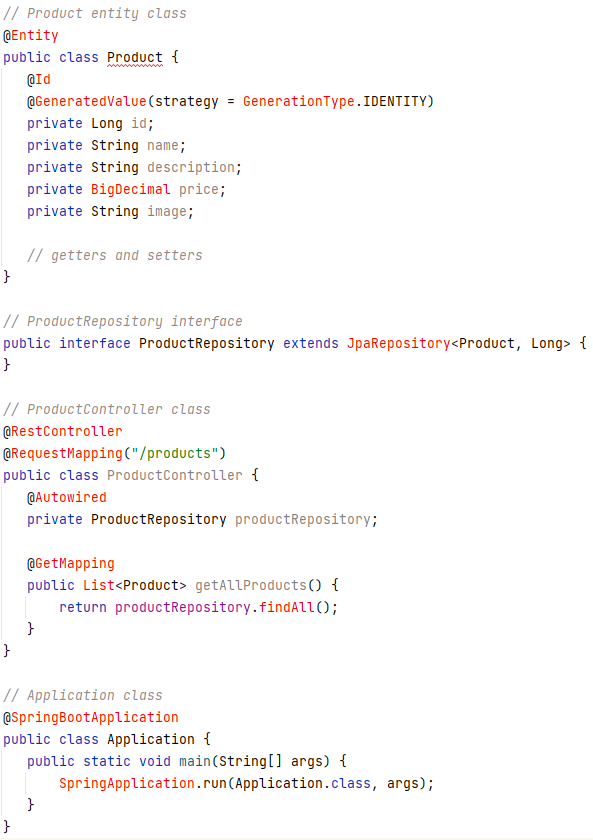
3.Keyin, **identifikator**, **nom**, **tavsif**, **narx** va **rasm** kabi xususiyatlarga ega mahsulotni ko'rsatish uchun **Mahsulot** ob'ekt klassini yaratasiz.

4.Shuningdek, siz **Spring Data JPA** ning **JpaRepository** interfeysini kengaytiruvchi **ProductRepository** interfeysini yaratasiz, bu esa **ma'lumotlar** **bazasi** bilan o'zaro ishlash metodlarini taqdim etadi.

5.Keyin, mahsulot ma'lumotlarini **olish** uchun **REST** so'nggi nuqtasini belgilaydigan **ProductController** klassini **yaratasiz**. Oxirgi nuqta **ma'lumotlar bazasidan mahsulot ma'lumotlarini olish** va uni **JSON** formatida qaytarish uchun **ProductRepository**-dan foydalanadi.

6.Nihoyat, siz dasturni ishga tushirasiz va **Postman** yoki **cURL** kabi vosita yordamida sinab ko'rasiz.

Ushbu qadamlarni ko'rsatish uchun ba'zi bir namuna kodlari:



Ushbu misol ma'lumotlar bazasidan **barcha** mahsulotlarni **oladigan** va **"/products"** so'nggi nuqtasiga **GET** so'rovi yuborilganda ularni **JSON** **formatida** **qaytaradigan** oddiy **mikroservisni** belgilaydi. E'tibor bering, bu oddiy misol va haqiqiy stsenariyda siz **pagination**(sahifalash), **filtrlash** va **saralash** kabi qo'shimcha funktsiyalarni qo'shishni xohlaysiz.

Bu Spring Boot **mikroxizmati** misoli qanday ishlashi haqida batafsil ma'lumot:

**Mahsulot** ob'ekt klassi **klass** va **ma'lumotlar bazasi** jadvali o'rtasidagi mapping(xaritalash)ni aniqlash uchun **JPA** izohlaridan foydalanadi. @**Id** izohi asosiy kalitni belgilaydi va @**GeneratedValue** izohi **JPA***ga identifikator strategiyasidan foydalangan holda avtomatik ravishda id-identifikatorni yaratishni aytadi.*

**ProductRepository** *interfeysi* **JpaRepository** interfeysini **kengaytiradi**, u **Mahsulot** ob'ekti uchun asosiy **CRUD** (**yaratish**, **o'qish**, **yangilash**, **o'chirish**) operatsiyalarini ta'minlaydi. Bu sizga hech qanday **SQL** yozmasdan ma'lumotlar bazasi operatsiyalarini osongina bajarish imkonini beradi.

**ProductController** klassi @**RestController** bilan izohlanadi, bu esa Springga bu klassda **RESTful API** so'nggi nuqtalari mavjudligini aytadi. @**RequestMapping** izohi ushbu kontrollerdagi barcha so'nggi nuqtalar uchun asosiy **URL** manzilini belgilaydi.

**getAllProducts**() metodi @**GetMapping** bilan izohlanadi, bu esa Springga bu metod **"/products**" so'nggi nuqtasiga **GET** so'rovlarini bajarishi kerakligini aytadi. Ushbu metod **productRepository**.**findAll**() metodi yordamida *ma'lumotlar bazasidan barcha mahsulotlarni* ***oladi*** va ularni **Mahsulot** ob'ektlari ro'yxati sifatida **qaytaradi**. Spring avtomatik ravishda ro'yxatni **JSON** formatiga o'zgartiradi va uni response bodyga **qaytaradi**.

Nihoyat, Ilova klassi @**SpringBootApplication** bilan izohlanadi, bu @**Configuration**, @**EnableAutoConfiguration** va @**ComponentScanni** birlashtirgan qulaylik izohidir. Ushbu izohlar Springga loyihani konfiguratsiya klasslari uchun skanerlashni, klass yo'li va boshqa sozlamalar asosida dasturni **avtomatik sozlashni** va loyihani kontrollerlar va omborlar kabi komponentlar uchun skanerlashni aytadi. **main**() metodi oddiygina Spring ilovasini ishga tushiradi.

Umuman olganda, bu misol Spring Boot yordamida oddiy **mikroservis** yaratish qanchalik osonligini ko'rsatadi. Bir nechta izohlar va ba'zi asosiy kodlar bilan siz *ma'lumotlar bazasidan ma'lumotlarni* ***oladigan*** *va uni standart formatda qaytaradigan* ***RESTful API******yaratishingiz*** *mumkin*. Bu yerdan siz mikroservisingizga ko'proq funksionallik va murakkablik kiritish uchun ushbu poydevorga asos solishingiz mumkin.

**Mikroservislarning amaldagi hayotiy misollari:**

**Netflix**: Netflix o'z platformasini *yaratish* va *ishlatish* uchun mikroservislardan *foydalanadigan* kompaniyaning ajoyib namunasidir. Netflix-da tavsiyalar, kontentni yetkazib berish va foydalanuvchi autentifikatsiyasi kabi turli funktsiyalarni bajaradigan 500 dan ortiq mikroservislar mavjud. Ilovani kichikroq, yoʻnaltirilgan serviceslarga boʻlish orqali Netflix oʻz platformasini osonlik bilan kengaytira oladi, serviceslarni mustaqil ravishda yangilaydi va nosozliklarga chidamliligini oshiradi.

**Uber**: Uber o'z platformasini quvvatlantirish uchun mikroservislarga tayanadigan yana bir kompaniya. Uber sayohatni **boshqarish**, **jo'natish** va **to'lovlarni qayta ishlash** kabi funktsiyalar uchun **mikroservislardan** foydalanadi. Ilovani kichikroq, yoʻnaltirilgan xizmatlarga boʻlish orqali Uber katta hajmdagi soʻrovlarni osonlik bilan hal qilishi, yangi funksiyalarni tezroq oʻrnatishi va platformasining ishonchliligini oshirishi mumkin.

**Amazon**: Amazon mikroservislardan foydalanishda peshqadam hisoblanadi. Amazon *mahsulot tavsiyalari*, *foydalanuvchi autentifikatsiyasi* va *buyurtmani qayta ishlash* kabi funktsiyalar uchun *mikroservislardan foydalanadi*. Ilovani kichikroq, yo'naltirilgan serviceslarga bo'lish orqali Amazon o'z platformasining *kengayishi* va *ishonchliligini* oshirishi, ishlab chiqish va joylashtirish vaqti va narxini kamaytirishi mumkin.

**PayPal**: PayPal *to'lovlarni qayta ishlash*, *firibgarlikni aniqlash* va *mijozlarni qo'llab-quvvatlash* kabi funktsiyalarni bajarish uchun mikroservislardan foydalanadi. PayPal ilovasini kichikroq va yoʻnaltirilgan serviceslarga boʻlish orqali *katta hajmdagi tranzaktsiyalarni amalga oshirishi*, *platformasining tezligi va ishonchliligin*i oshirishi va tezroq innovatsiyalar yaratishi mumkin.

**Airbnb**: Airbnb qidiruv, bron qilish va xabar almashish kabi funksiyalar uchun mikroservislardan foydalanadi. Ilovasini kichikroq va yoʻnaltirilgan serviceslarga boʻlish orqali Airbnb oʻz platformasining kengayishi va ishonchliligini oshirishi, yangi funksiyalarni tezroq oʻrnatishi va foydalanuvchi tajribasini shaxsiylashtirishi mumkin.

Umuman olganda, mikroservislar so'nggi yillarda tobora ommalashib bormoqda, chunki kompaniyalar kengaytiriladigan, moslashuvchan va tezkor platformalarni yaratishga intilmoqda. Murakkab ilovalarni kichikroq, ko'proq yo'naltirilgan serviceslarga bo'lish orqali kompaniyalar o'zlarining innovatsiyalar qilish, mijozlarning o'zgaruvchan ehtiyojlariga javob berish va raqobatchilardan oldinda turish qobiliyatini yaxshilashlari mumkin.

**Multithreading**

**Multithreading** - bu bir vaqtning o'zida dasturning bir nechta threadlarini bajarish usuli.

Informatika fanida **thread** - bu operatsion tizim tomonidan rejalashtirilishi mumkin bo'lgan eng kichik ishlov berish birligi. **Thread** - bu bir xil xotira va resurslarni baham ko'radigan, bir xil jarayondagi boshqa oqimlar bilan bir vaqtda ishlashi mumkin bo'lgan engil jarayon.

Oddiy so'z bilan aytganda, **thread** - bu *kompyuter dasturining bir vaqtning o'zida bir nechta ishlarni bajarish usuli*. Buni bir vaqtning o'zida bir nechta narsani qila oladigan odam kabi o'ylab ko'ring - masalan, telefonda gaplashish va bir vaqtning o'zida elektron pochta yozish. Xuddi shunday, kompyuter dasturida bir vaqtning o'zida har biri o'z ishini bajaradigan bir nechta mavzular bo'lishi mumkin. Bu dasturlarning tezroq ishlashiga va sezgir bo'lishiga yordam beradi.

**Thread**lar odatda **veb-serverlar**, **video o'yinlar** va **video tahrirlash** dasturlari kabi yuqori unumdorlik va sezgirlikni talab qiladigan ilovalarda **qo'llaniladi**.

Shuni ta'kidlash kerakki, **Thread**lar o'zlarining xotira maydonida ishlaydigan dasturning mustaqil nusxalari bo'lgan jarayonlardan farq qiladi. Jarayonlar xotirani bir-biri bilan baham ko'ra olmasa-da, bir xil jarayon ichidagi oqimlar xotira va resurslarni almashishi mumkin, bu ularni yengilroq va samaraliroq qiladi.

**Thread**ni dastur, vazifani bajarish uchun bajaradigan bir qator ko'rsatmalar sifatida tasavvur qiling. **Multithreading** bilan dastur bir vaqtning o'zida bir nechta ish zarralarini bajarishi mumkin, ya'ni u bir vaqtning o'zida bir nechta vazifalarni bajarishi mumkin.

Misol uchun, siz internetdan fayllarni yuklab oluvchi dasturni ishga tushiryapsiz deylik. Ko'p ish zarralari yordamida siz bitta faylni bitta faylni yuklab olishi mumkin, ikkinchisi esa bir vaqtning o'zida boshqa faylni yuklab olmoqda. Bu dasturga bir vaqtning o'zida bitta faylni yuklab olishdan ko'ra tezroq yuklab olishni yakunlash imkonini beradi.

**Multithreading** odatda **video o'yinlar**, **video tahrirlash** dasturlari va **veb-serverlar** kabi yuqori unumdorlik va sezgirlikni talab qiluvchi dasturiy ilovalarda qo'llaniladi.

*Spring Boot kontekstida* ***multithreading*** *frameworki bir vaqtning o'zida bir nechta dasturni bajarish qobiliyatini anglatadi.*

Spring Boot multithreadingni ishlatish uchun bir nechta xususiyatlarni taqdim etadi, masalan, **thread** pool(hovuz)larini **aniqlash** va **sozlash** qobiliyati, vazifalarni bajarish uchun ishlatilishi mumkin bo'lgan **thread**lar to'plamidir.

Spring Boot shuningdek, **asinxron dasturlashni** qo'llab-quvvatlaydi, bu dasturning asosiy **thread**ini *bloklamasdan* *vazifalarni bir vaqtda bajarishga imkon beruvchi dasturlash modelidi*r. Bu tarmoq so'rovlarini bajarish yoki uzoq muddatli hisob-kitoblarni bajarish kabi vazifalar uchun foydali bo'lishi mumkin.

**Asinxron dasturlash** - *bu dasturga bir vaqtning o'zida bir nechta vazifalarni bajarishga imkon beradigan, har bir vazifani keyingisini boshlashdan oldin tugashini kutmasdan, dasturlash usuli.*

Buni bir vaqtning o'zida bir nechta taomlarni pishiradigan oshpaz kabi tasavvur qiling. *Bir idish pechkada qaynayotganda*, u*lar boshqa taom uchun ingredientlarni tayyorlashni boshlashlari mumkin*. Shunday qilib, ular barcha idishlarni tezroq pishirishni tugatishlari mumkin.

Xuddi shunday, **asinxron dasturlashda** dastur bitta vazifani bajarishni boshlashi mumkin va shu vazifaning tugashini kutayotganda, u boshqa vazifani bajarishni boshlashi mumkin. Bu dasturga **vaqt** va **resurslardan** yaxshiroq foydalanish imkonini beradi va dasturning ishlashi va javob berish qobiliyatini yaxshilashga yordam beradi.

Umuman olganda, **multithreading** Spring Boot-ning **muhim xususiyati** bo'lib, u **ilovalarning ishlashi** va **kengayishini** **yaxshilashga** yordam beradi.

* Spring Boot ilovasida **multithreadingni** qanday amalga oshirishga misol:

Ushbu misolda bizda **MyService** deb nomlangan Spring Boot servesi mavjud bo'lib, u **so'rovlarni qayta ishlash** uchun javobgardir. **ProcessRequest** metodi @**Async** bilan izohlanadi, u Springga ushbu metodni **thread**da bajarishni aytadi.

*Yangi so'rov* kelganda, Spring Boot avtomatik ravishda **processRequest** metodini bajarish uchun yangi **thread** yaratadi. Bu bir vaqtning o'zida bir nechta so'rovlarni qayta ishlash imkonini beradi, bu esa dasturning ishlashi va javob berish qobiliyatini oshiradi.

**ProcessRequest** metodi ichida biz tarmoq so'rovini qilish yoki murakkab hisoblashni amalga oshirish kabi har qanday vaqt talab qiladigan vazifani bajarishimiz mumkin. Vazifa bajarilgandan so'ng, **MyRepository** ob'ekti yordamida natijani ma'lumotlar bazasiga **saqlashimiz** mumkin.

Umuman olganda, bu Spring Boot ilovasida unumdorlik va miqyosni yaxshilash uchun **multithreading**dan qanday foydalanish mumkinligiga faqat bir misol.

