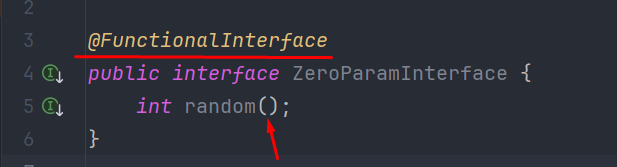
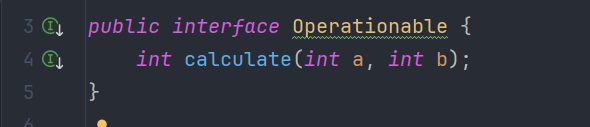
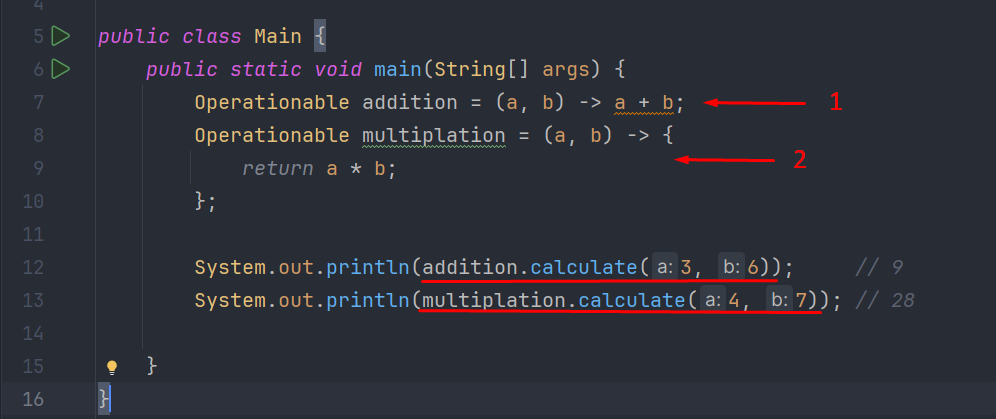
-Lambda expression(LE) lar bilan ishlashda bizga eng avvalo Functional Interface(FI) kerak bo’ladi. FI bu 3-qatorda yozilgan annotatsiyadir. FI bu Interfaceda hech bo’lmaganda bitta bo’lsa ham, abstract method bo’lishidir, aniqrog’i kamida bitta bo’lsa ham bo’lishi shart, aks holda xatolik beradi. Pastdagi misolda bizda bitta **random()** nomli abstract method mavjud:

-Lambda expression(LE) ni ifodalash uchun bizga Interface lar kerak bo’ladi. Pastdagi misolda bizda bitta abstract calculate() method bor.

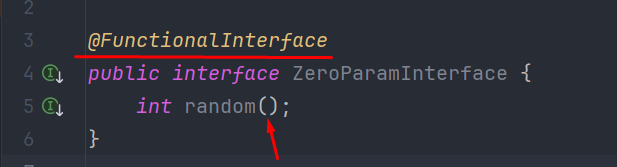


Endi shu methodni Functional Interface qoidasidan foydalanib, LE ni hosil qilamiz. 1 va 2 bilan ko’rsatilgan joylarda LE yozilyapti. Aynan mana shu joylarda LE ni body(tanasi) yozilyapti. Keyin esa 12-13-qatorlarda uni chaqiryapmiz:

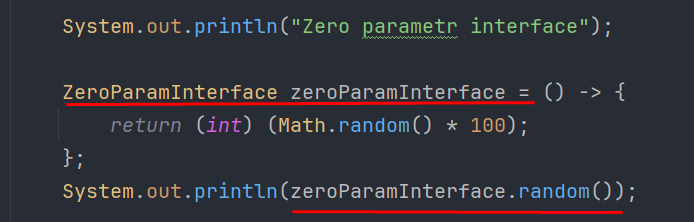


LE da 3 xil parameter type qiladigan function lar bo’ladi:

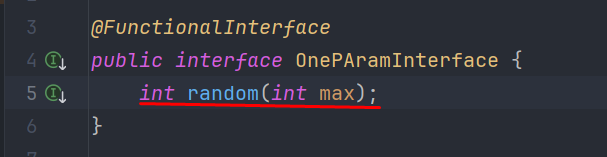
1. Zero parameter(Parameteri yo’q)



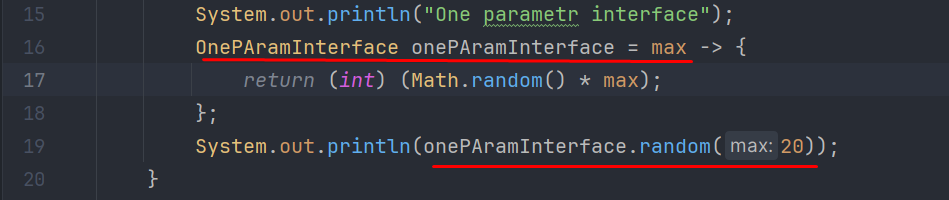
Endi shu random() methodini bodysini yozamiz va chaqiramiz:



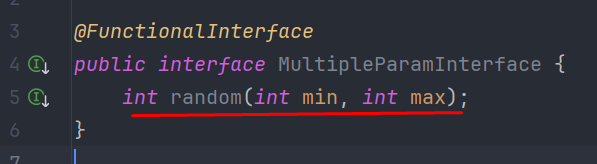
1. One parameter



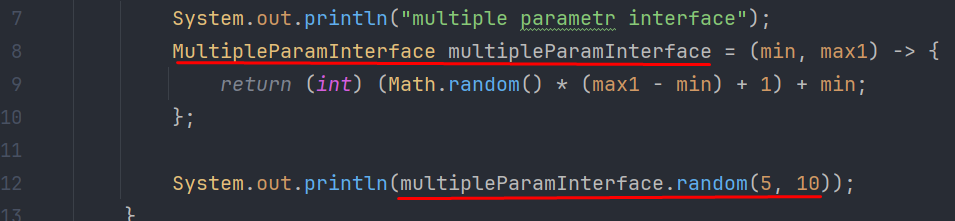
Endi shu random() methodimizni bodysini yozamiz va chaqiramiz:

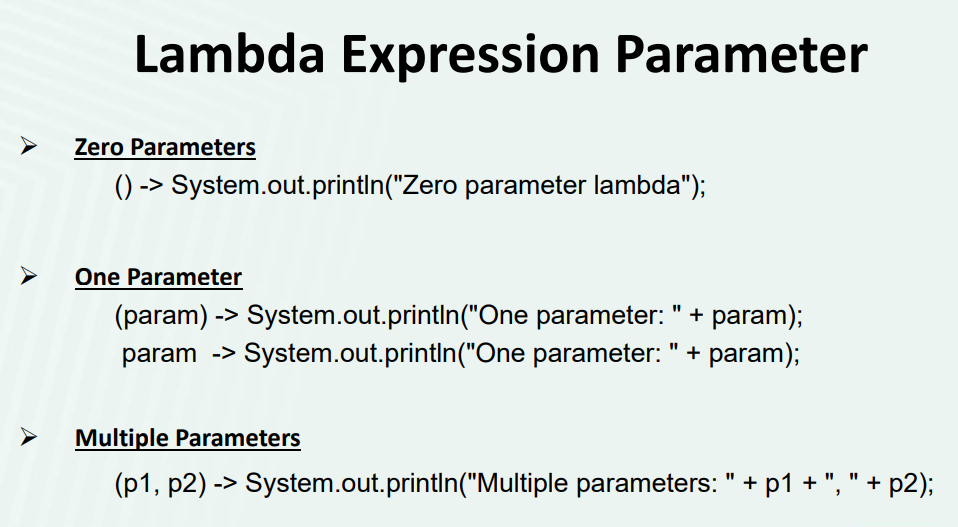


1. Multi parameter

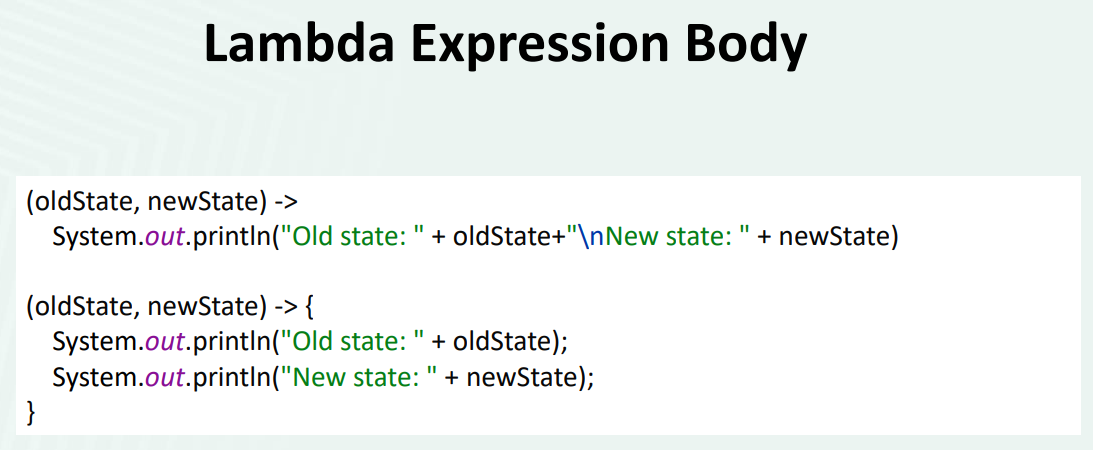


Endi shu random() methodimizni bodysini yozamiz va chaqiramiz:

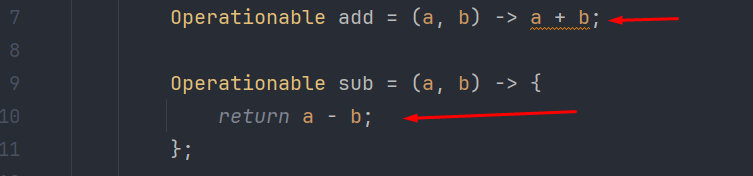




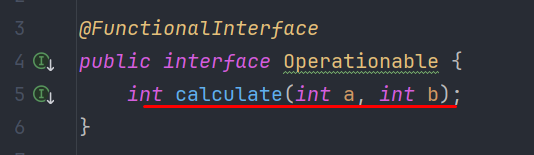
LE da body qismi faqat bitta statementdan iborat bo’lsa, u holda **{}** olish shart emas yoki olsak ham bo’ladi, xatolik bermaydi, lekin 2 va undan ortiq statementdan iborat bo’lsa, olish shart:



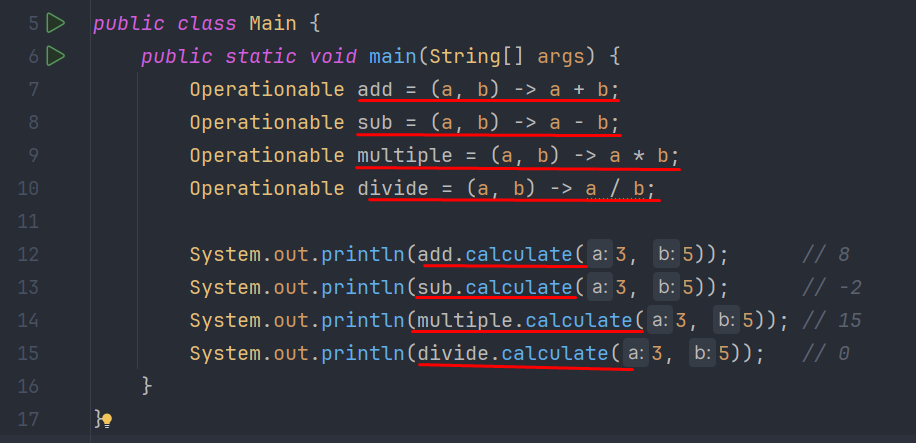
Pastda 7-qatorda istasak **return** ni qo’ymasak ham bo’ladi ixtiyoriydir, xuddi shunday 9-qatorda ham **return** ni qo’ysak ham bo’ladi. Lekin statementimiz bir qatordan iborat bo’lsa, u holda **return** ni yozish shart emas:



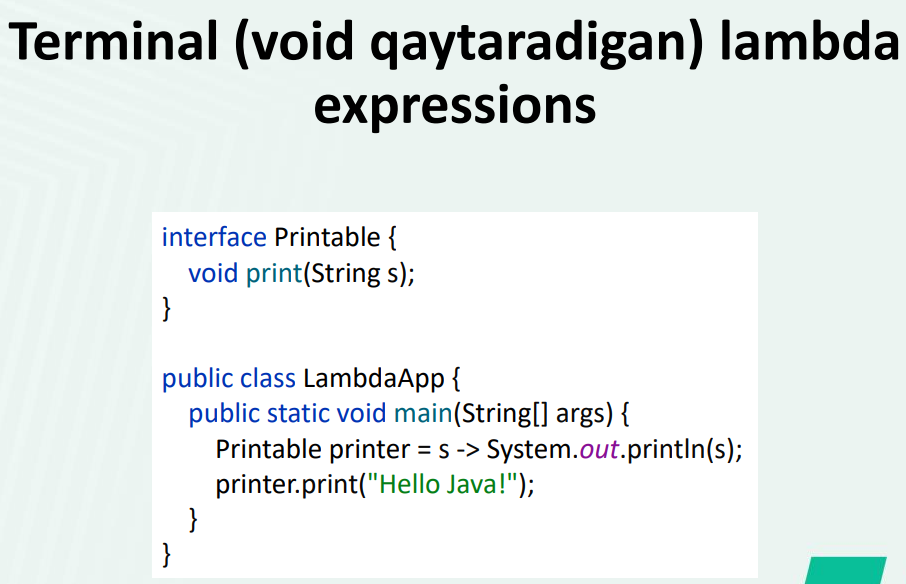
Agar istasak bitta abstract method yozib qo’yib, uni bodysini har xil function uchun har xil yozish mumkin. Masalan, pastda calculate() nomli abstract method bor:



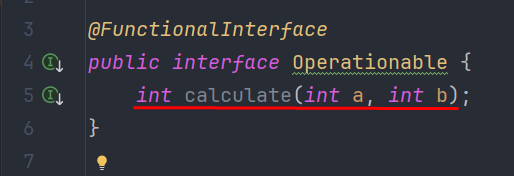
Pastda esa shu abstract methodni bodysini 7-8-9-10-qatorlarda har xil qilib yozdik, ya’ni bitta calculate() methodni bodysini har xil realizatsiya qildik:

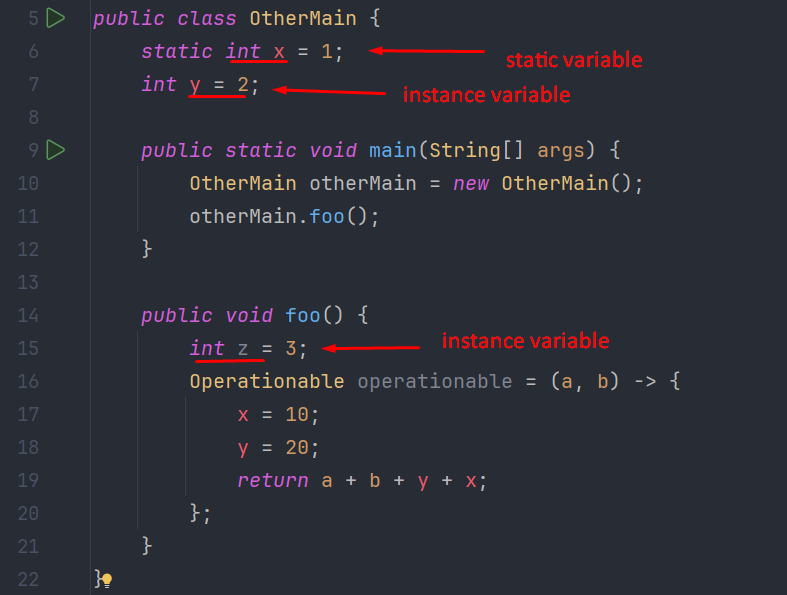


Bizda terminal(void qaytaradigan) lambda expression ham bor. Bu expression hech nima qaytarmaydi. Shuning uchun void lambda expression deyiladi:

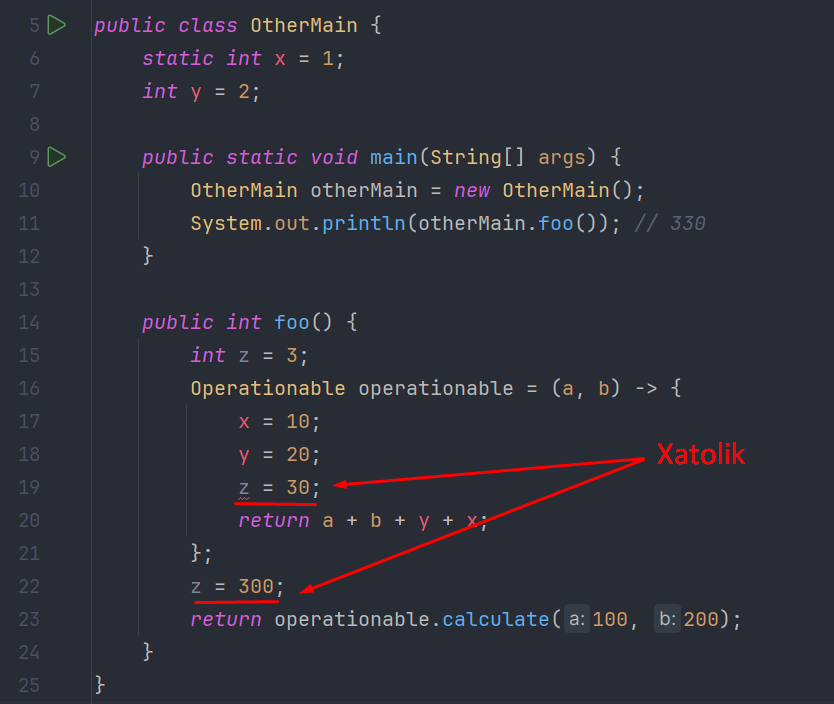


LE larda static, instance va local o’zgaruvchilarni ishlatsa ham bo’ladi. Abstract calculate() method yaratib olamiz:

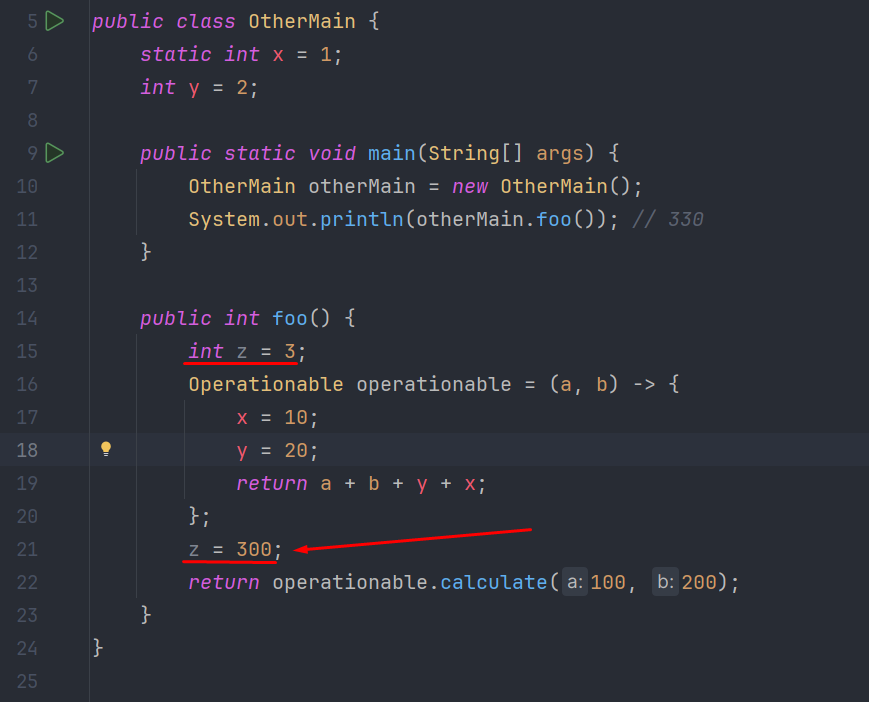


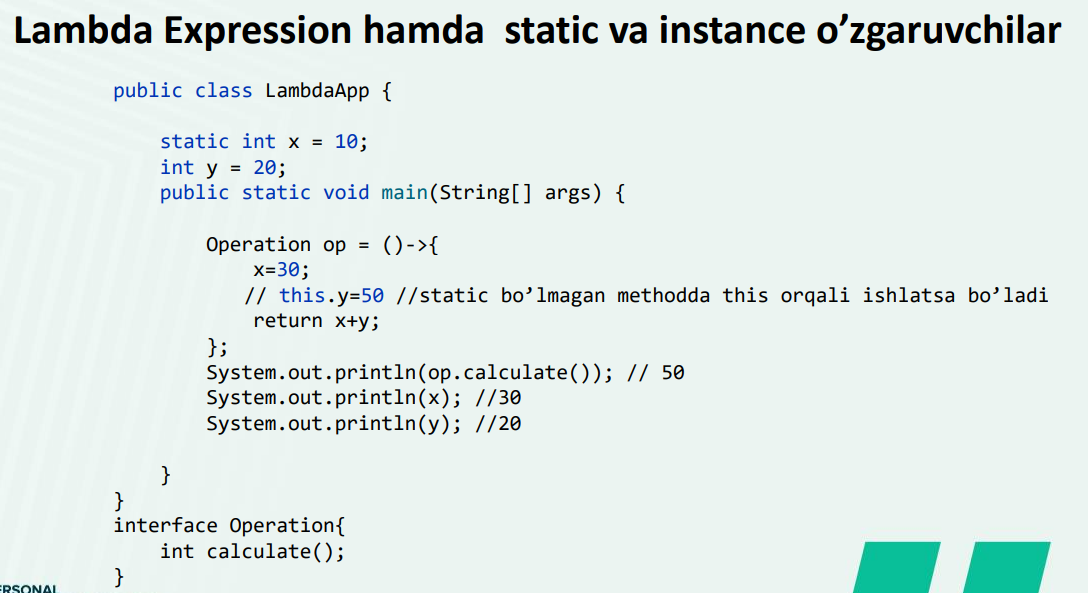


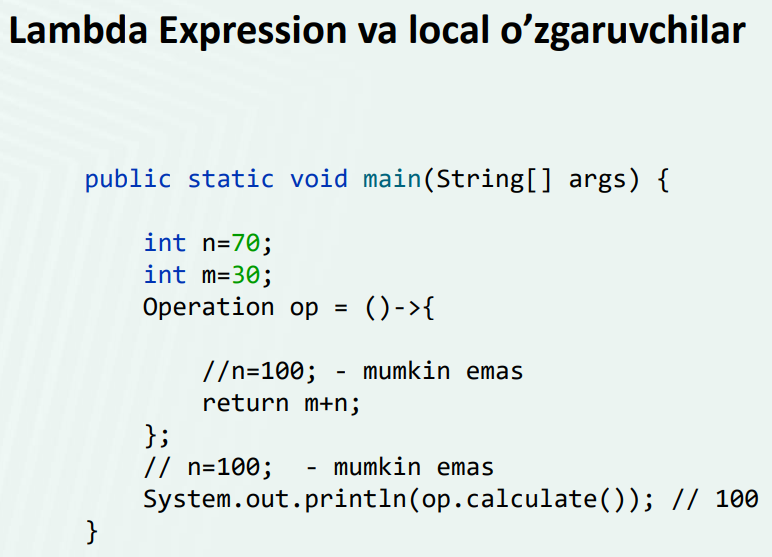
Lekin bu yerda bir narsani bilish kerakki, agar instance o’zgaruvchini LE ni ichida ishlatadigan bo’lsak, u holda bu local o’zgaruvchi avtomatik ravishda **final** bo’lib qoladi, ya’ni constantaga aylanib qoladi. Keyin uni qiymatini o’zgartirmoqchi bo’lsak, xatolik beradi. Pastda ko’rish mumkinki, 19-qatorda LE ni ichida o’zgartirsak yoki 22-qatorda foo() methodni ichida o’zgartirsak ham xatolik beradi:



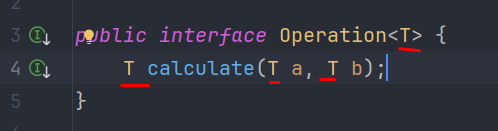
Lekin LE ni ichida local o’zgaruvchini ishlatmasak, u holda bu o’zgaruvchini methodni ichida chaqirsak ham xatolik bermaydi:



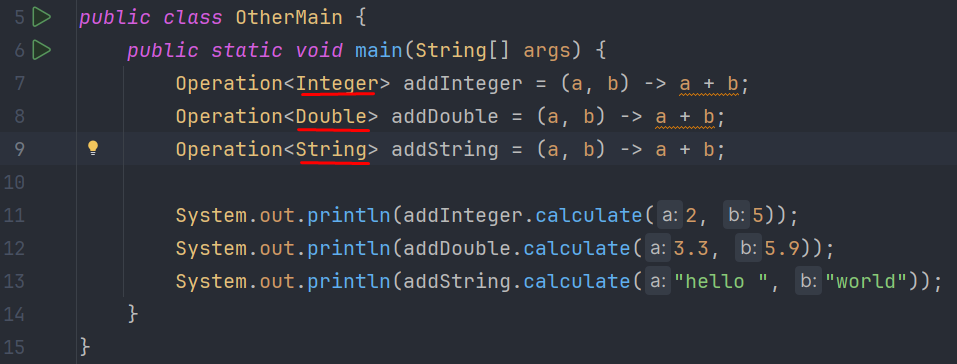


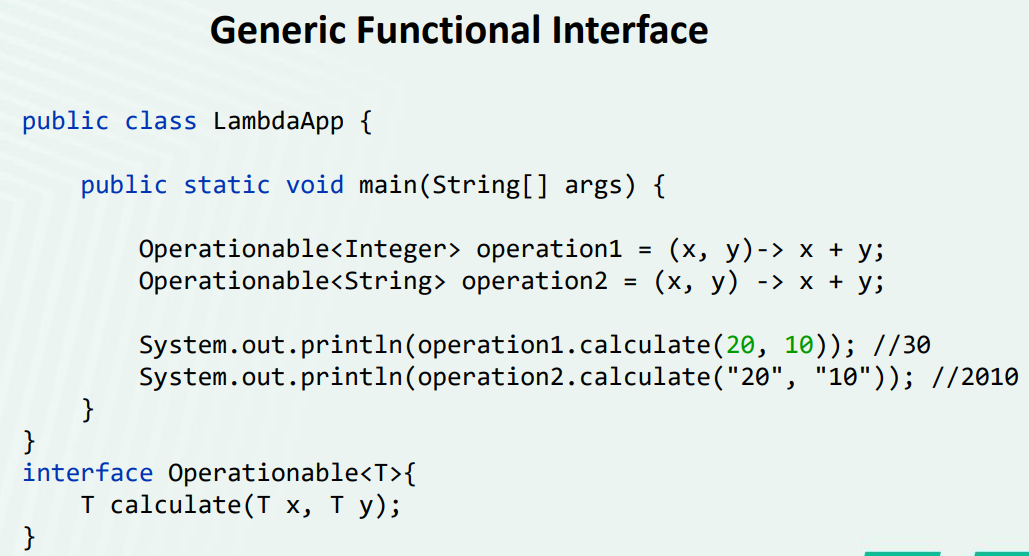


Agar istasak LE da generic typelardan foydalanish ham mumkin.

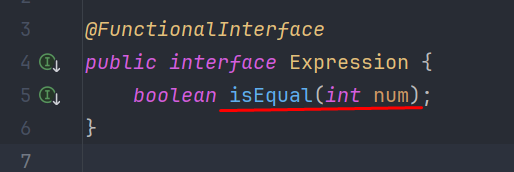


Pastda undan foydalanish berilgan:

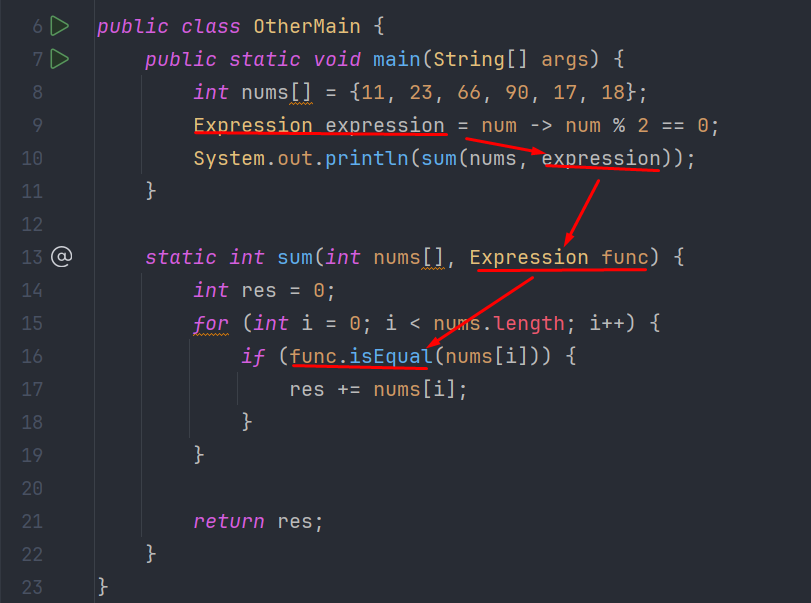




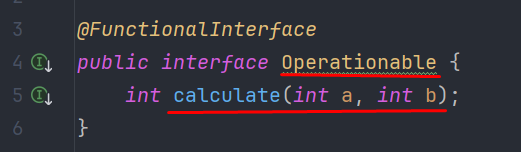
Agar istasak, biz method parametriga LE ni berib yuborishimiz mumkin. Buning uchun **isEqual()** nomli abstract method yaratamiz:



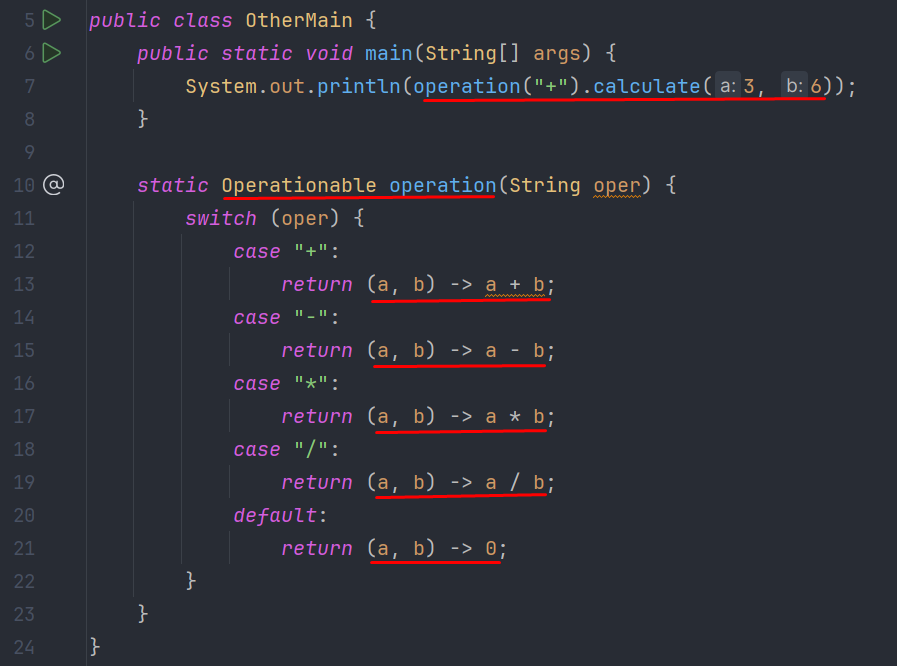
Pastda 13-qatorda **sum()** methodini ichida LE ni berib yuboryapmiz va undan 16-qatorda foydalanyapmiz:



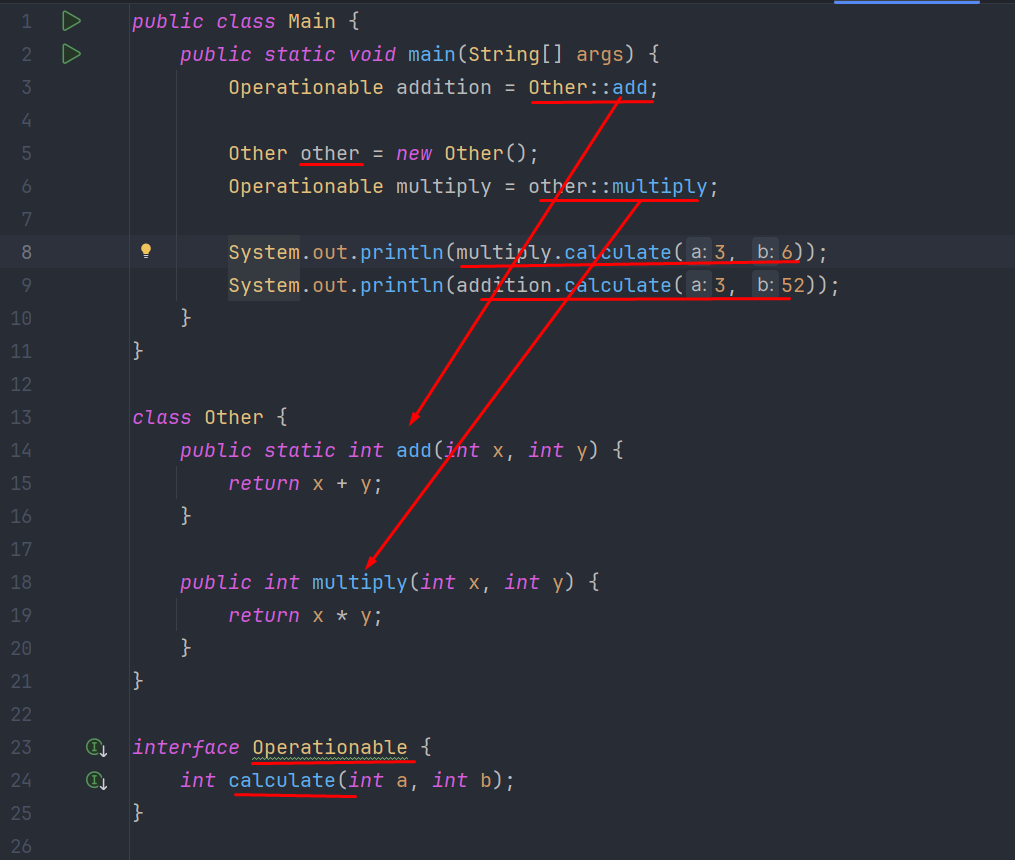
Agar istasak biz LE ni methodni return typeda ham jo’natishimiz mumkin:



Pastda return typeda LE ni return qilyapti:



LE da method reference degan tushuncha bor. U nima degani misol bilan ko’ramiz.



Constructor reference.

