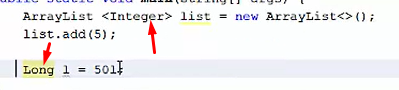
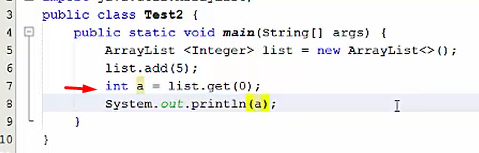


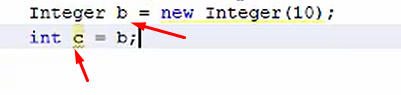
Pastdagi misolda Integer va Long lar autoboxing bo’layapti.



Pastdagi misolda esa AL ga Integer bo’lib qo’shilgan 5 soni, list.get(0) deb olganda primitive type int ga o’girilyapti. Demak bu yerda Unboxing qoidasi ketyapti:

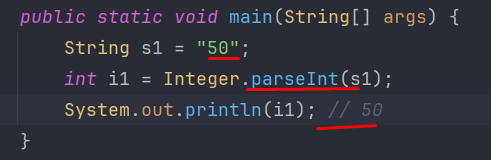


Pastdagi misolda ham int c = b; qismida unboxing qoidasi ketypati:

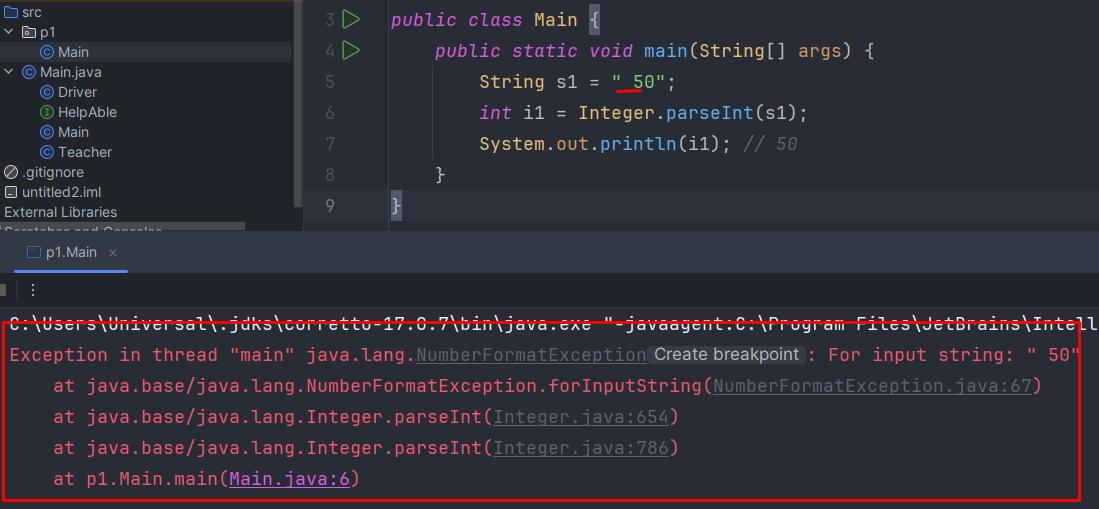


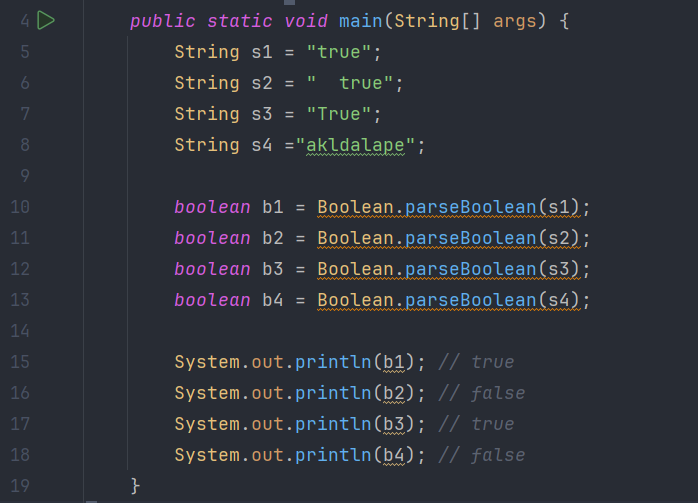
Parse

Pastdagi misolda string typedagi sonni Integerga parse qildik:

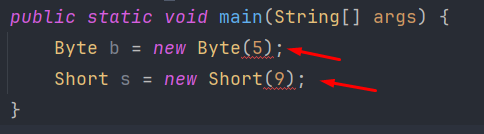


Agar string sonni probel bilan yozsak, u holda compile timeda xatolik emas, balki runtimeda exception beradi:

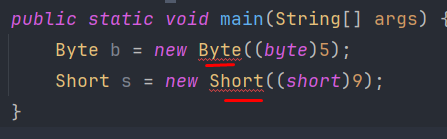


Boolean bilan ishlashda sal boshqacharoq bo’ladi: 

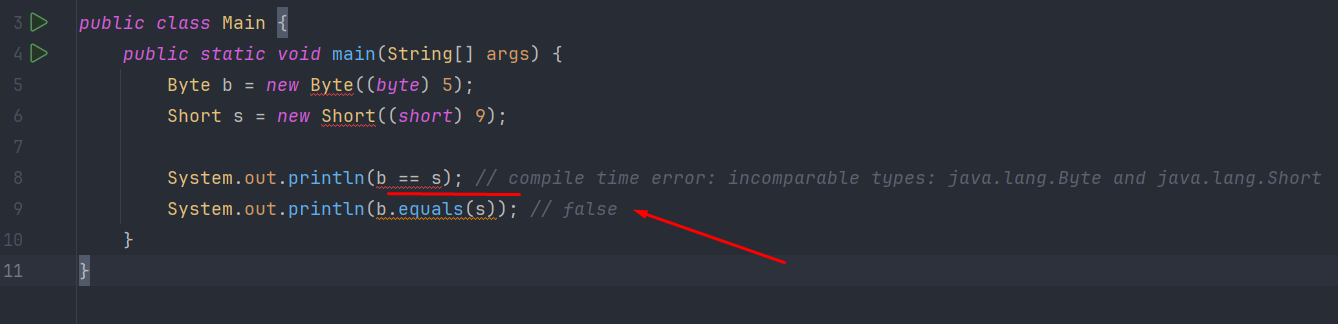
Pastdagi misolda xatolik beryapti compile timeda:



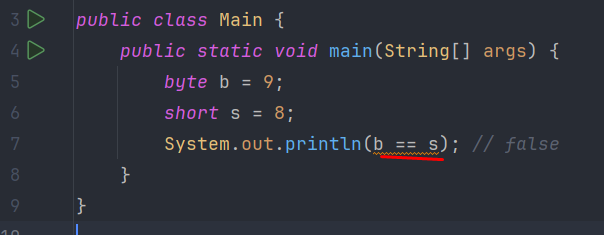
Sababi 5 va 9 sonlarimiz int type dadir. Shuning uchun biz ularni o’zini type ga casting qilishimiz kerak:



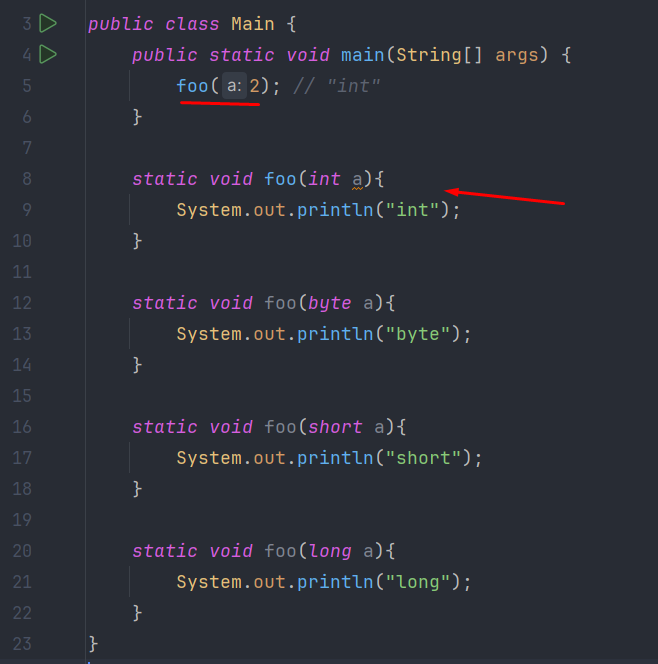
Biz javada **==** yordamida 2 ta har typeli objectlarni tenglashtira olmaymiz. Compile timeda xatolik beradi. Sababi new Byte() va new Short() deb yaratilyapti. Demak ular objectlardir. Har xil typeli bo’lgani uchun xatolik beradi:



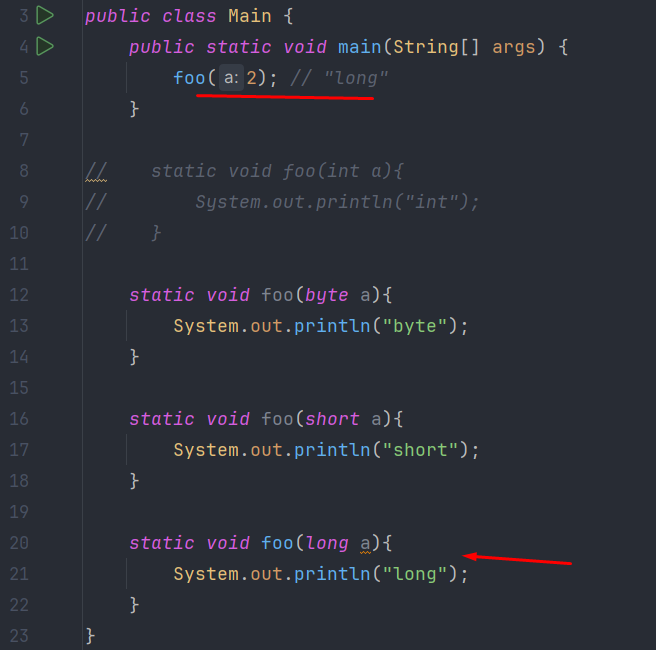
Lekin primitive typelarda esa unday emas to’g’ri ishlaydi:



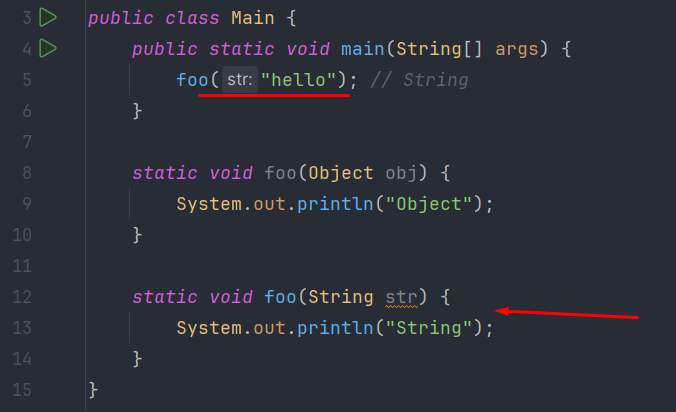
Pastdagi misolda ko’rish mumkinki, 2 ni yozsak int typelini chaqirdi, sababi int type default typedir bu 4 ta byte, short, int va long type uchun:



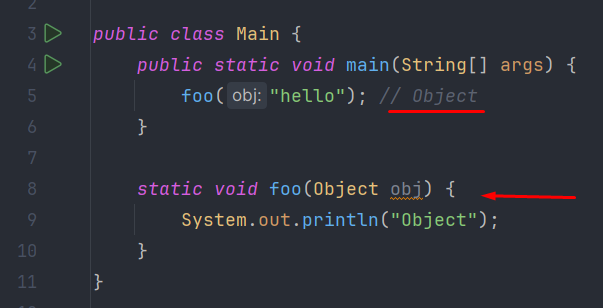
Lekin **int** ni qabul qiladigan methodni commentga olsak, u holda **long** ni chaqiradi. Sababi bizda default holatda int bo’lishi kerak edi, lekin int yo’q, bunda int dan kattaroq typeli long ni chaqiradi:



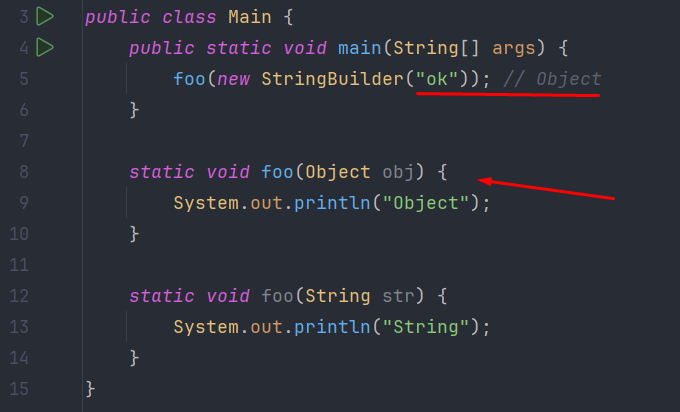
Pastdagi misolni ko’radigan bo’lsak, “hello” so’zi Object va String ga ham mos tushadi. Lekin aniqrog’i String ga mos tushadi:



Agar string ni olib tashlasak, u holda Object ni mos tushaveradi:



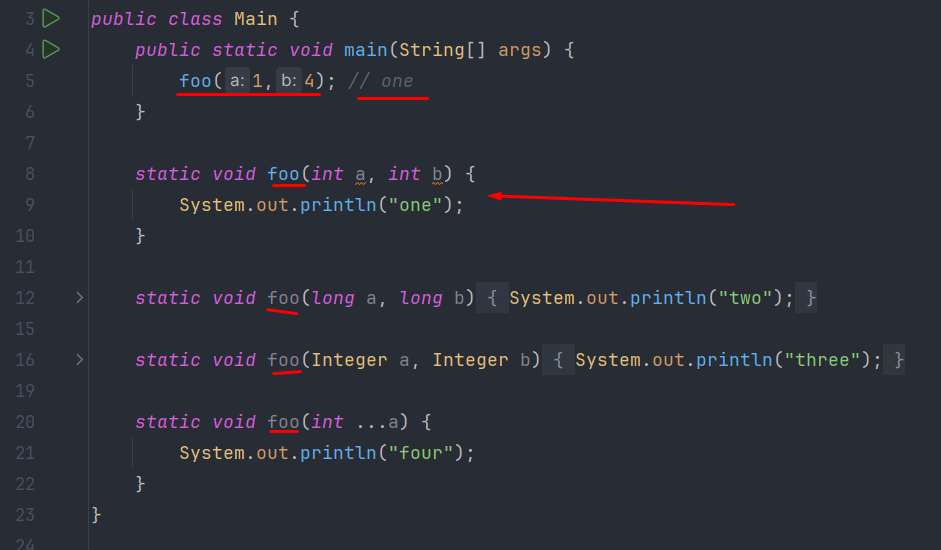
Agar **StringBuilder()** beradigan bo’lsak, u holda Stringga mos tushmaydi, Object ga mos tushadi. Shuning uchun Object ni chiqaradi:



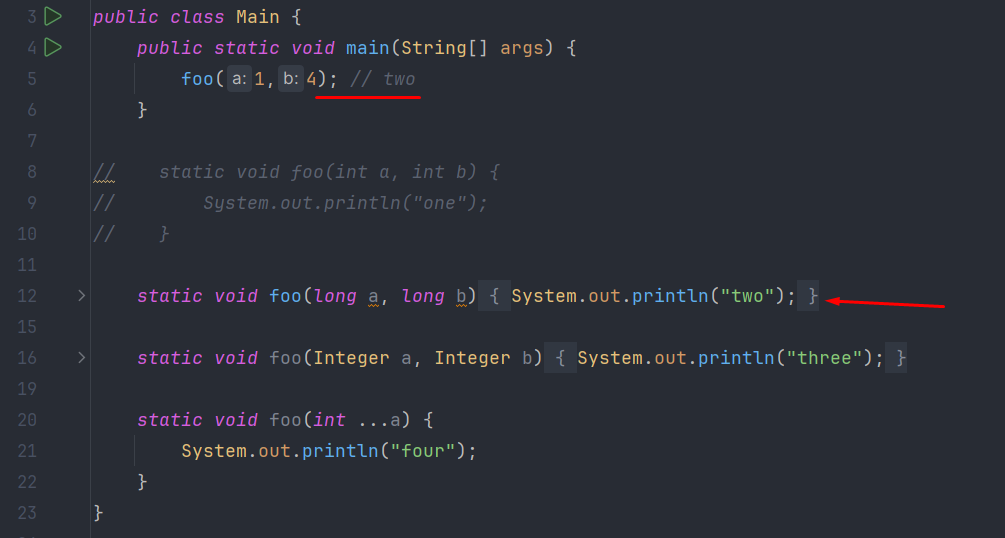
**Method overloading da methodlarni muhimlilik darajasi**



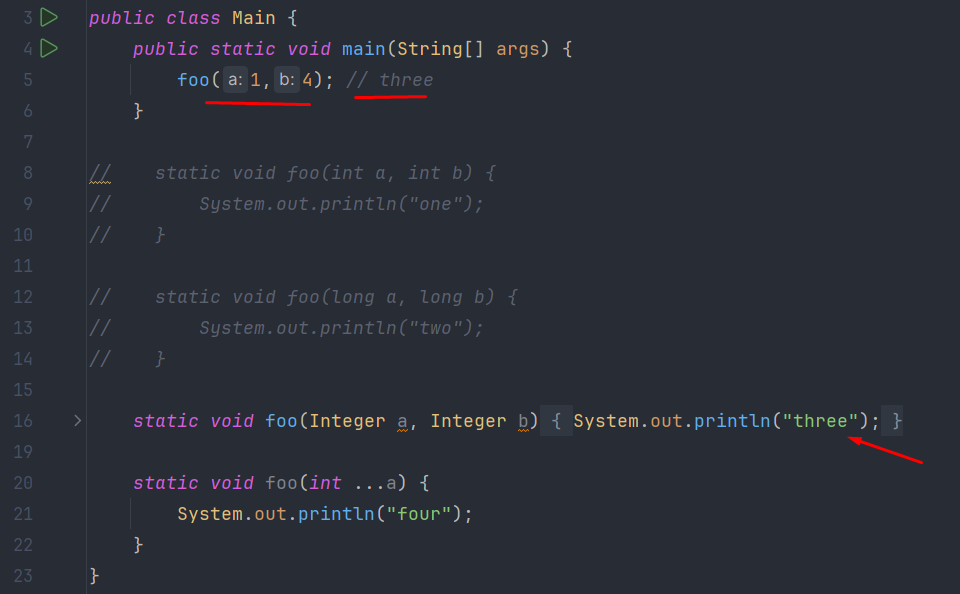
Pastdagi misolda **foo()** method parameteri **int** typeli bo’lganini chaqiradi. Sababi eng mos tushadigani shudir. Bu qoida yuqoridagi **screenshootni** **1-**qoidasiga mos tushadi:



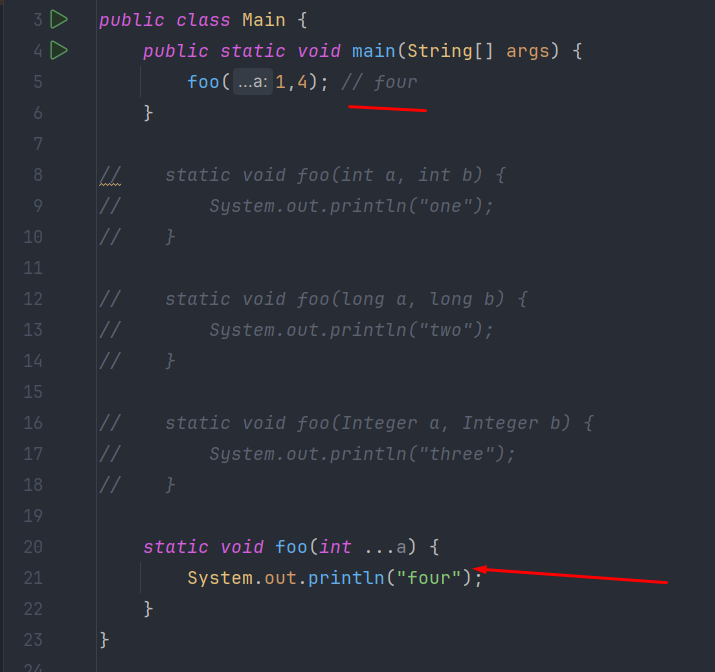
Yuqoridagi qoidani 2-qismiga ko’ra kattaroq primitive typelisi chaqiriladi. Bu yerda biz int typelisini commentga oldik. Endi int dan kattarog’i esa long bo’lgani uchun long typelisi chaqirilyapti:



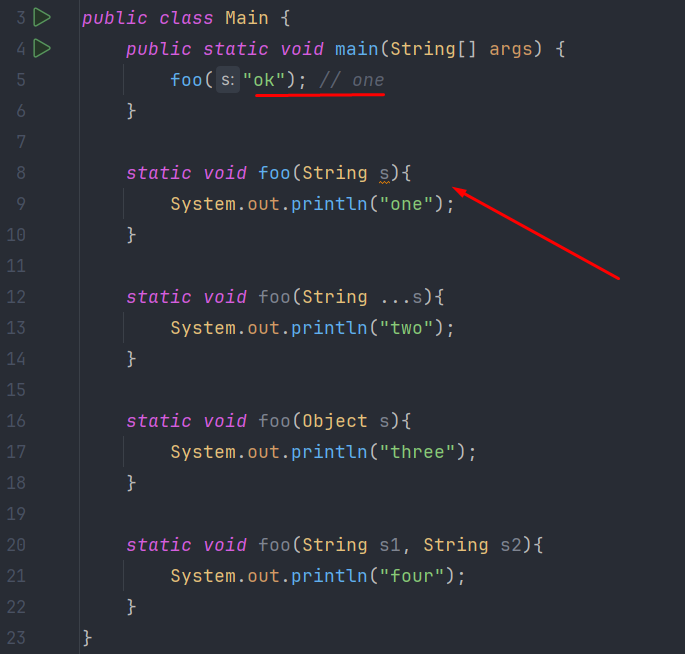
3-qoidaga ko’ra Autoboxed type chaqiriladi.Bu yerda 1 va 4 int typeda bo’lgani uchun, int va long typelisini biz commitga olganimiz uchun Autoboxed bo’lgani uchun Integer lisini chaqiradi:



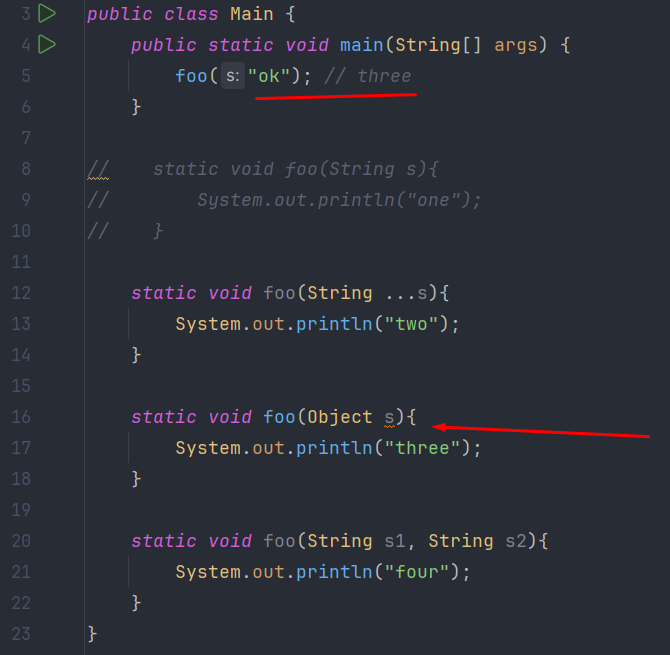
**4**-qoidaga muvofiq agar yuqoridagi **3** ta qoidaga mos tushadigan holat bo’lmasa, oxiri **varargs** li holatni qidiradi:



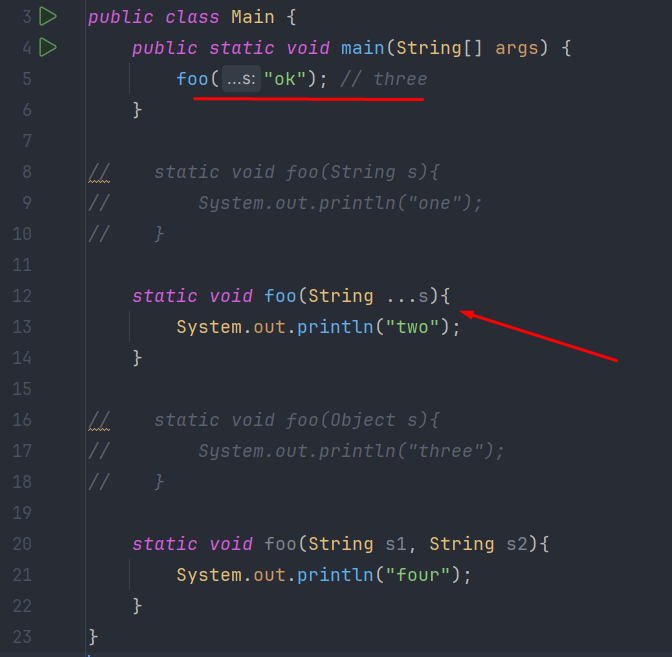
Endi boshqa reference type yordamida yuqoridagi 4 ta qoidani ko’ramiz. 1-qoidaga ko’ra eng aniq mos tushuvchi holat 8-qatordagi **foo()** methodidir:



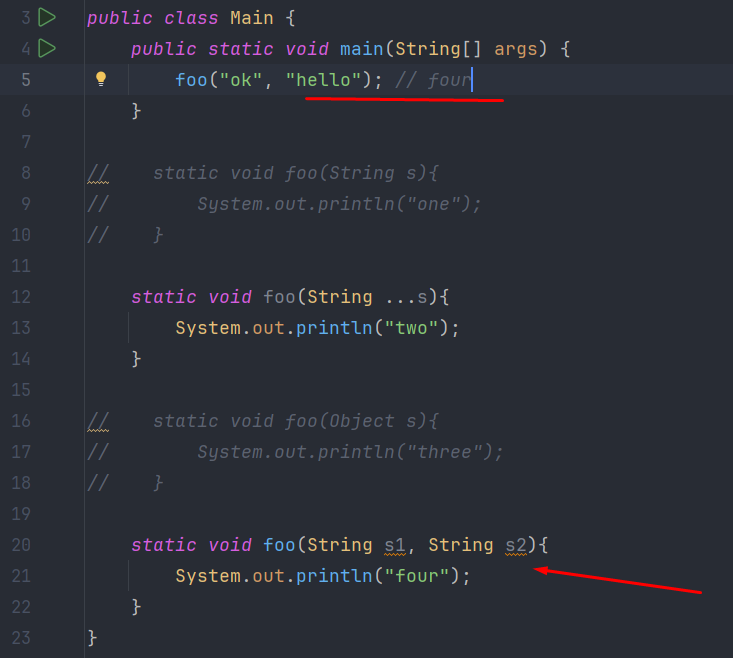
2-qoidaga muvofiq agar eng aniq mos tushuvchi typelisi topilmasa, u holda o’zidan kattaroq bo’lgan typeni qidiradi. 2-qoidada agar parameter reference type bo’lsa, u holda o’zini parent classini qidirib topadi. String ni parent classi bu yerda Object typedir. Shuning uchun pastdagi holatda Object typelisini qidirib topadi:



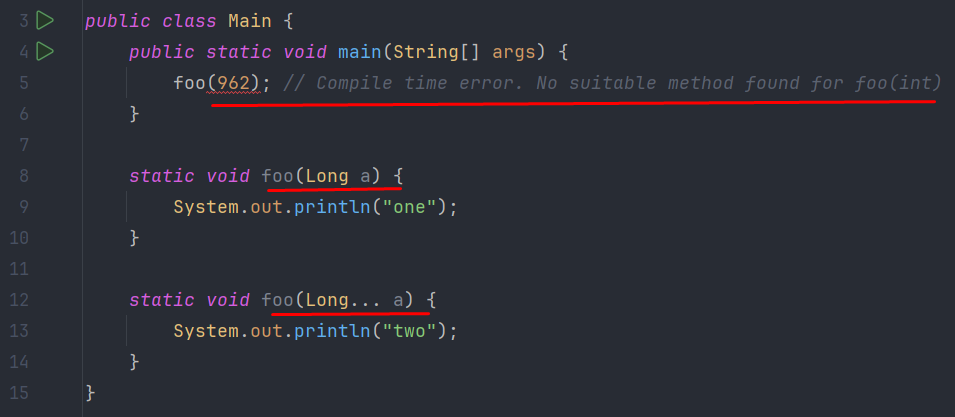
4-qoidaga ko’ra varargs mos tushadi. Sabai parametrida bitta “ok” so’zi bor xolos:



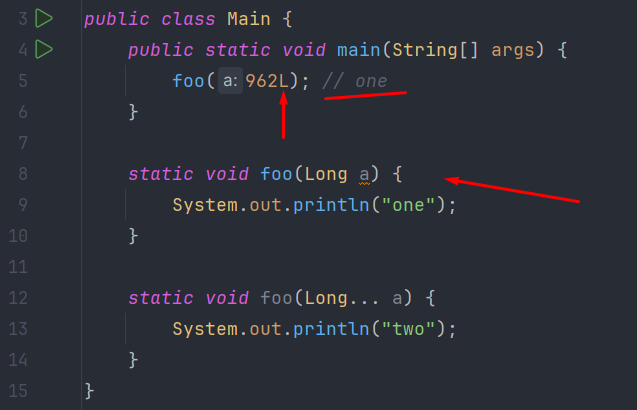
Pastdagi holatda eng aniq tushadigani bu 20-qatordagi foo() methodidir: To’g’ri bu yerda varargs ham mos tushadi, lekin eng anig’I 2 ta parameter oladiganidir:



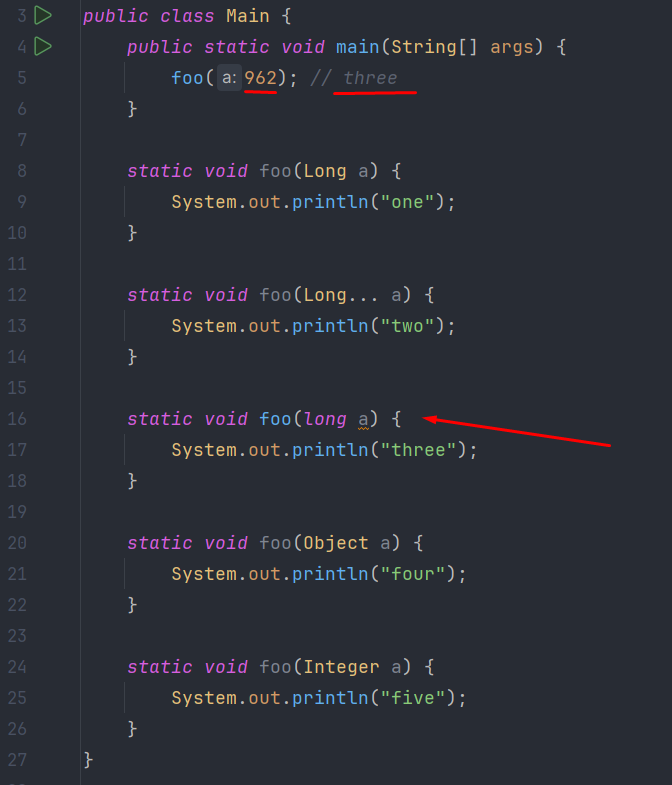
Pastdagi misolni ko’radigan bo’lsak, 2 ta foo() method e’lon qilganmiz. 2 lasi ham Long typeni qabul qiladi. Lekin 5-qatorda 962 sonini berib chaqirsak, u holda compile time error chiqyapti. Sababi 962 soni int typedadir. Biz bilamizki byte, short, int, long typelari uchun default type bu int typedir. Shuning uchun pastdagi holatda 962 ni int type deb qabul qilyapti compilator:



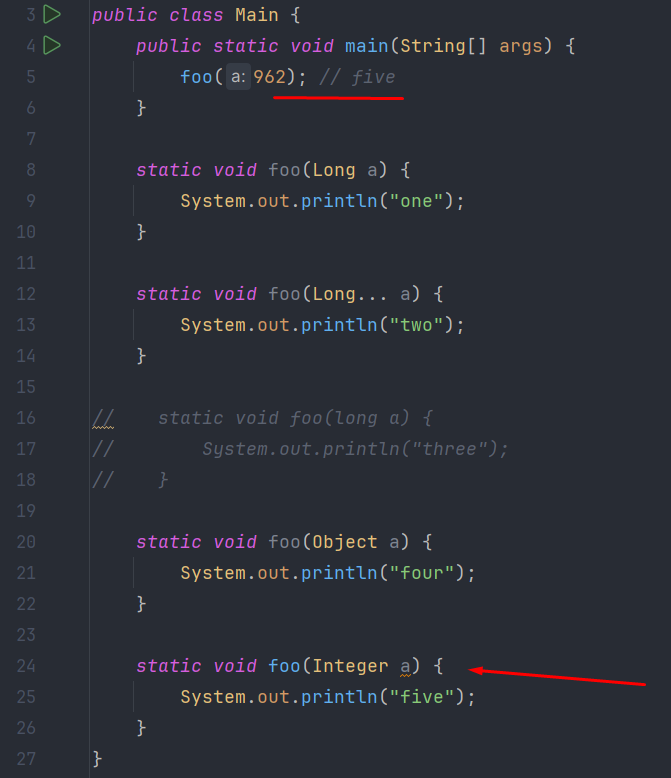
Agar 962L deb bersak, biz long typeli son berdik degan bo’lamiz. Shunda error chiqmaydi. Bu holatda esa 8-qatordagi foo() methodini chaqiradi. Sababi Autoboxing qoidasi ketypati, ya’ni 962L bu primitive long typedir, shuni esa o’zini mos bo’lgan reference type Long ga o’giryapti. Bundan tashqari eng aniq tushadigani ham shu 8-qatordagi methoddir:



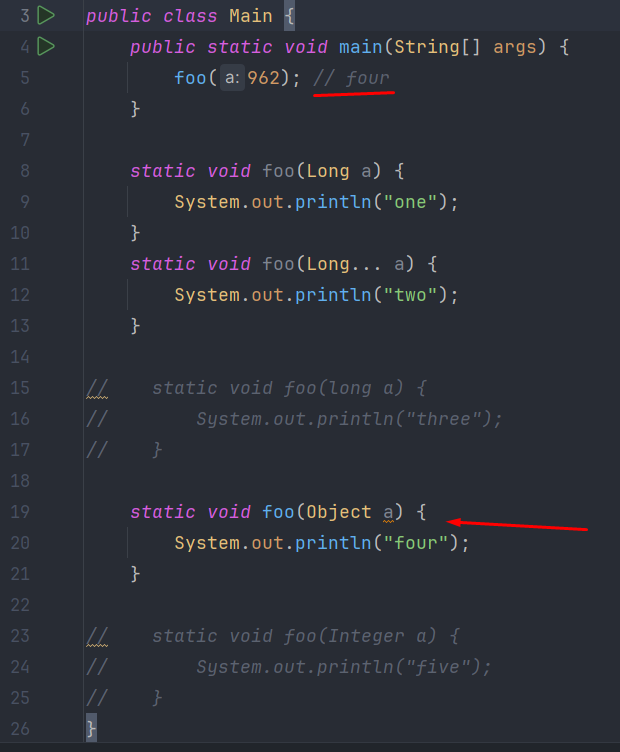
Agar yana boshqa methodlar qo’shib, 962 ni bersak, bu safar 16-qatordagi foo() method chaqiriladi. Sababi 962 int typeda bo’lib, primitive typedir. Yuqoridagi qoidani 2-qismiga ko’ra primitive typeda o’ziga eng aniq mos tushadigan type bo’lmasa, u holda o’zidan kattaroq primitive typeni oladi. Bu yerda kattaroq primitive type bu long dir:



Agar long typedagisini commentga olib chaqirsak, u holda 24-qatordagi foo() methodni chaqiradi. Bu yerda Autoboxing qoidasi ketyapti. Primitive type o’zini mos reference typega o’girilyapti. 962 int type bo’lgani uchun, uni reference type esa Integer dir:



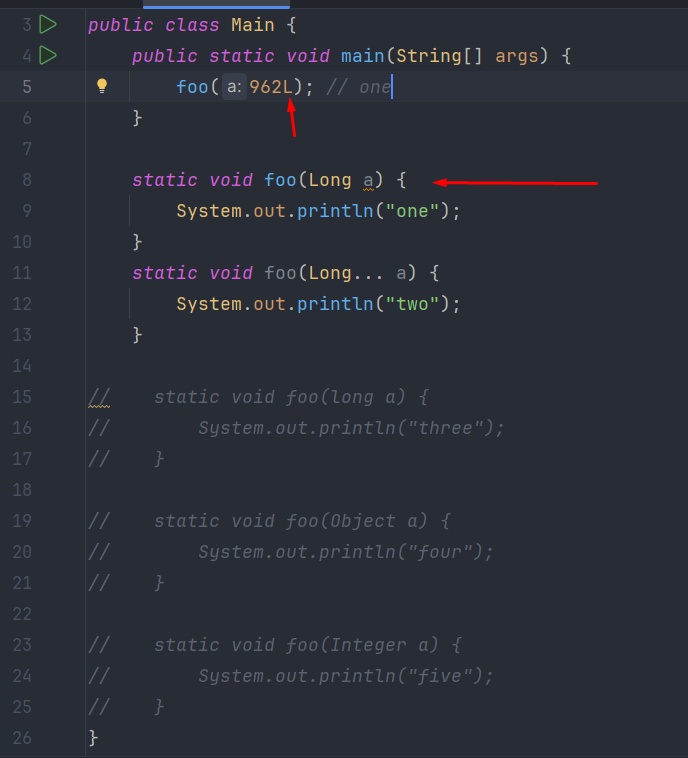
Agar Integer typelini ham commentga oladigan bo’lsak, u holda Object typelisi chaqiriladi. Sababi Object hamma classlar uchun ota classdir, Integer class uchun ham:



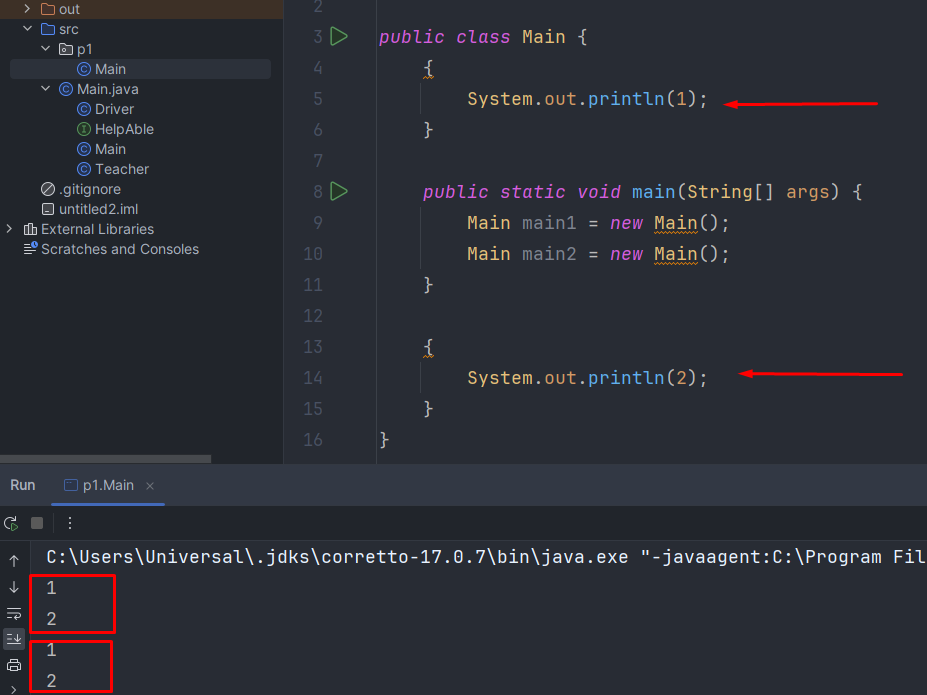
Agar object typelisini commentga oladigan bo’lsak, u holdaxatolik kelib chiqadi. Sababi 962 soni default holatda int typedadir. 8- va 11-qatorlardagi foo() methodlar esa long type qabul qiladi. Shuning uchun error beradi:



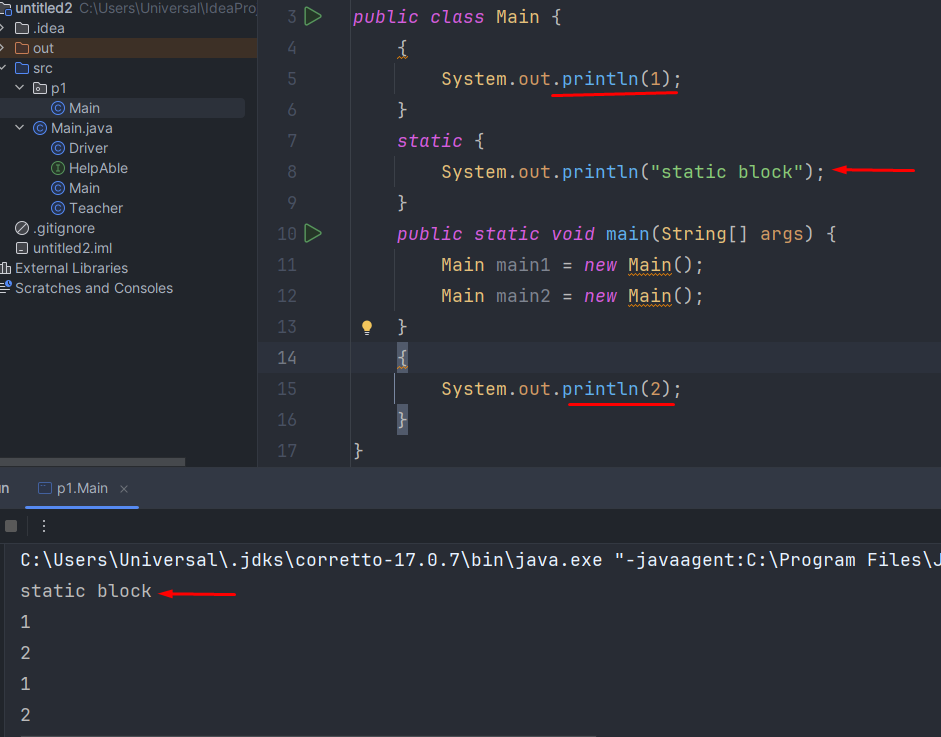
Agar 962L qilib long type bilan chaqiradigan bo’lsak, u holda xatolik yo’qoladi. Chunki Long typeni berib yuborypmiz:



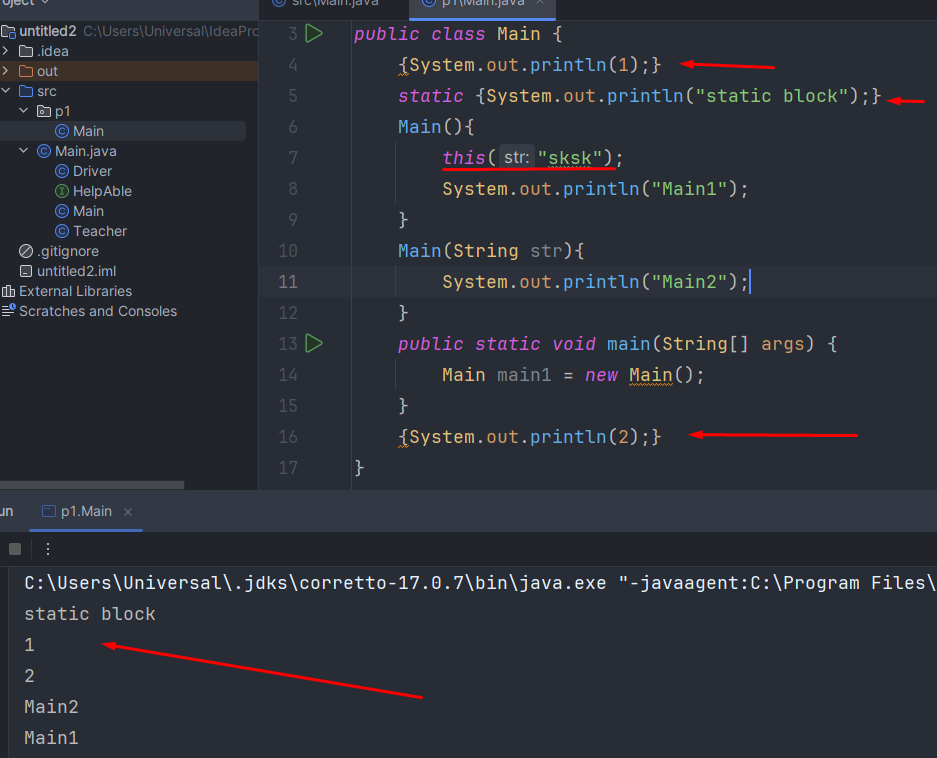
Initiliazer initial block(IIB) har safar constructor chaqirilganda chaqiriladi. Shuning uchun pastda 2 marta 1 va 2 yozuvini ko’rdik:



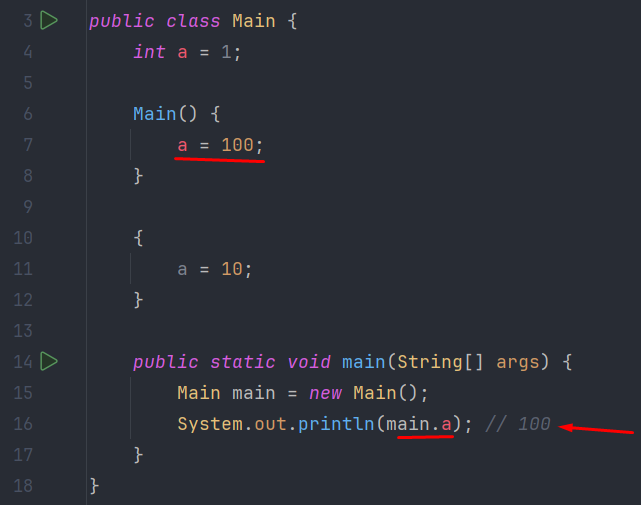
Static block ham bor bo’lib, bu block class load bo’lganda faqat 1 marta chaqiriladi. IIB dan shu tomonlari bilan farq qiladi:



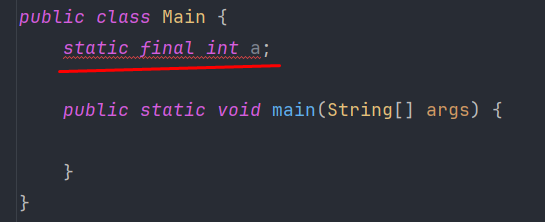
Yana boshqa holatda ishlatilishi:



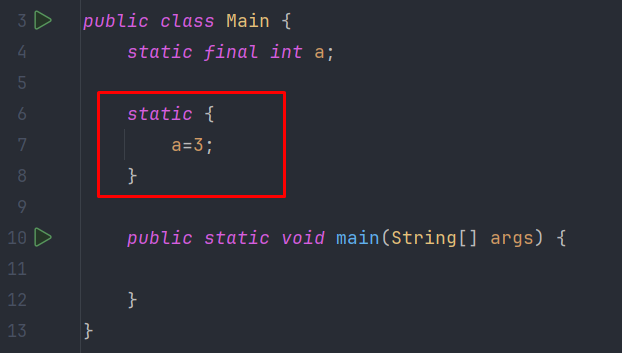
Pastdagi ifodaga e’tiborli bo’lish kerak. Pastda 4-qatorda a=1; deb a o’zgaruvchisi e’lon qilindi. Undan keyin 6-qatorda Main() nomli constructor e’lon qilindi va uni ichida a=100; qilindi. Undan keyin esa 10-qatorda IIB ochildi va a=10; qilindi. Endi 16-qatorda esa a ni chaqirdik. Natija esa a=100; bo’ldi. Nega constructorni ichidagi 100 ni oldi? Sababi IIB constructordan keyin bajariladi. Bizni holatda constructor eng oxirida bajarilgani uchun a=100; bo’ldi:



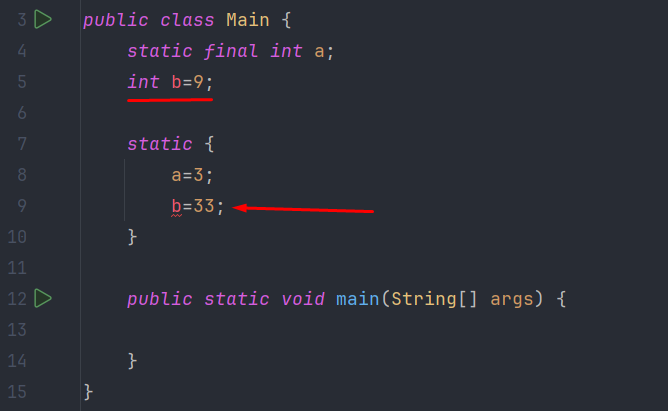
Biz bilamizki static final o’zgaruvchini e’lon qilgan zaxoti unga qiymat berish shart, aks holda error beradi:



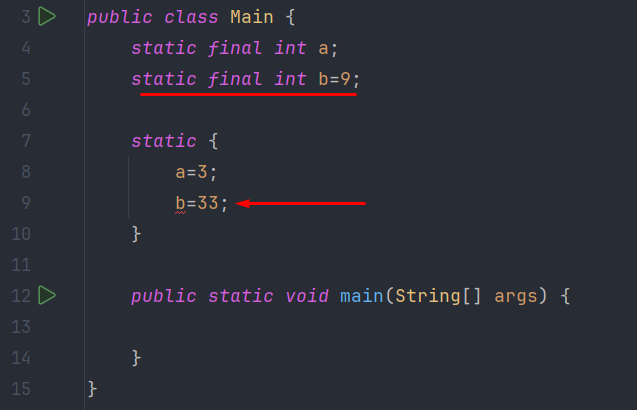
Lekin static blockda boshlang’ich qiymat bersa, xatolik yo’qoladi:



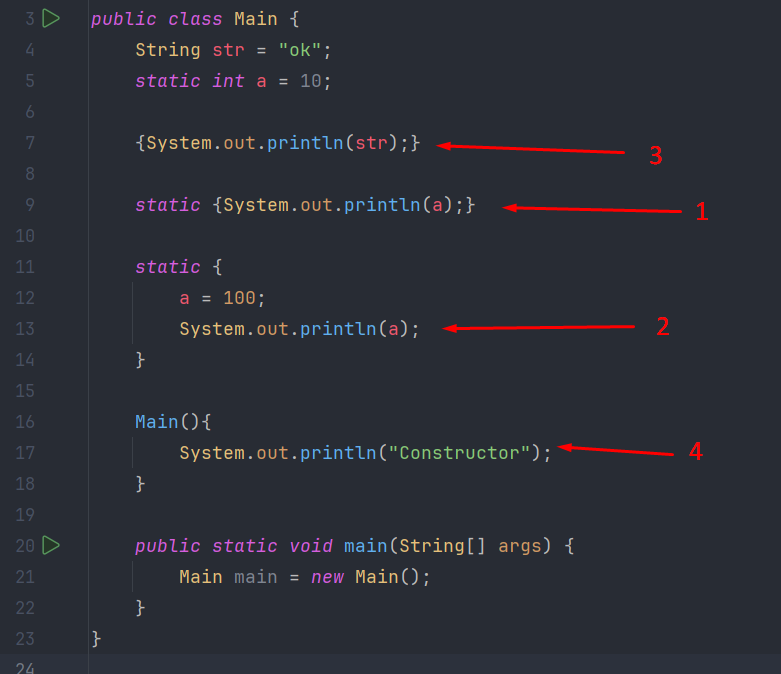
Static blockda instance o’zgaruvchini ishlatib bo’lmaydi, aks holda xatolik beradi. Sababi b o’zgaruvchi instance(object) o’zgaruvchi bo’lgani uchun, biz uni static blockda chaqira olmaymiz, chunki static block class load bo’lganda ishga tushadi. Bu vaqtda hali object ham yaratilmagan bo’ladi:



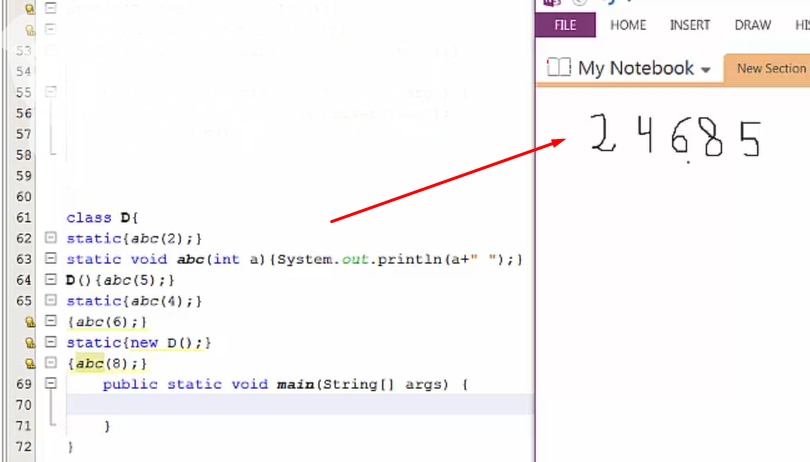
final static o’zgaruvchiga qiymat bergandan keyin unga 2-marta static blockda qiymat berib bo’lmaydi. Chunki final o’zgaruvchilar constanta hisoblanadi:



Pastdagi misolda chaqirilish ketma-ketligi. Har doim eng birinchi static block chaqiriladi, undan keyin IIB va oxirida constructor chaqiriladi:



Pastdagi misolni ko’ramiz:



Ota-bola munosabatida static va IIB blocklarni chaqirilish ketma-ketligi tartibi. 4-qatorda Dog classdan object olganda, eng avval Animal(ota) classni static blocki keyin esa Dog(bola) classni static blocki chaqiriladi. Sababi static block eng avval class load bo’lganda bajariladi, keyin esa 14-qatordagi

