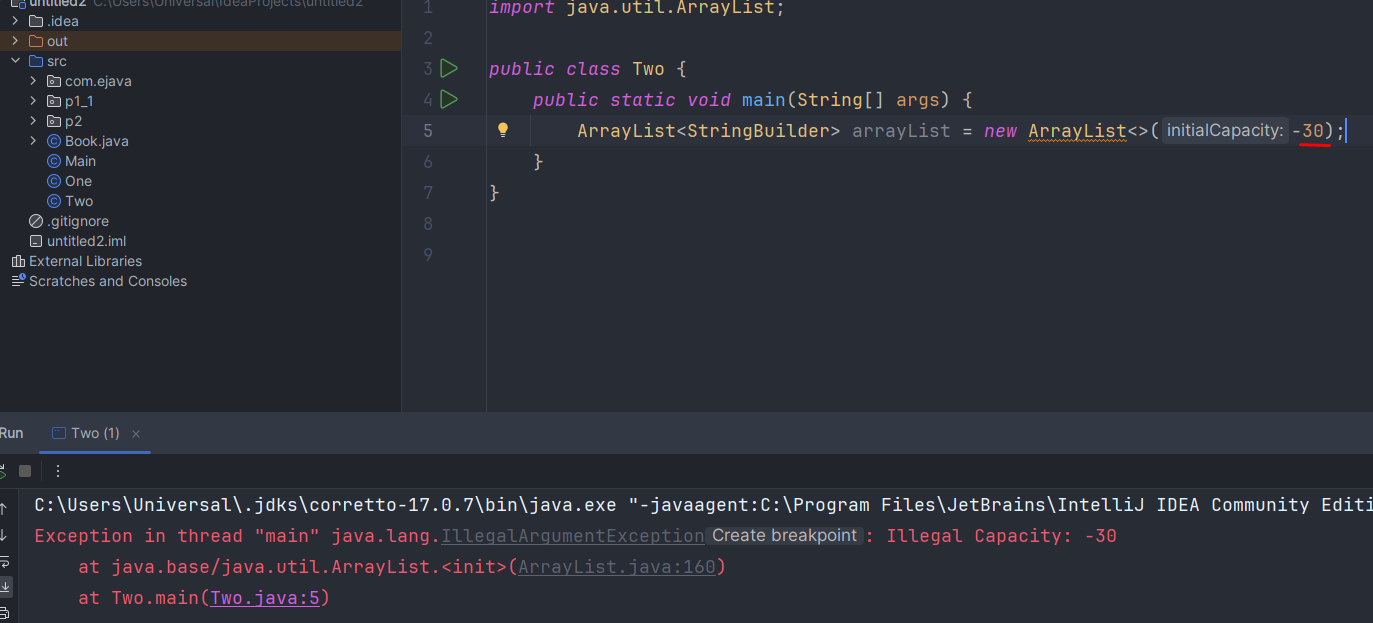
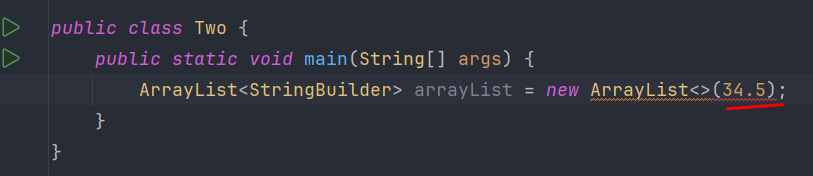
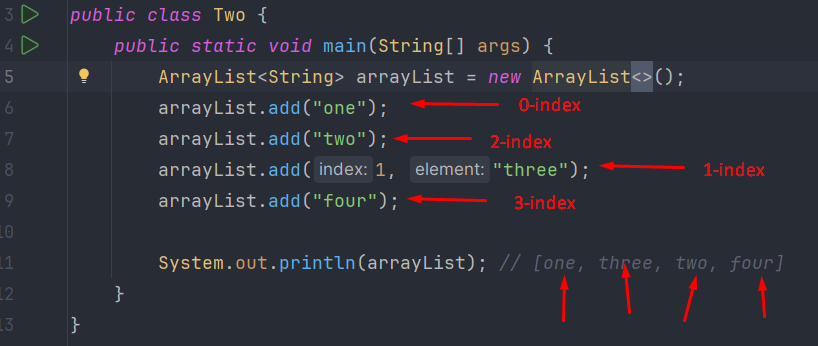
**ArrayList(AL)** da **capacity** siga biz manfiy son berishimiz mumkin, chunki capacity faqat **int** typedagi sonni oladi. Bunda hech qanday compile time error bermaydi, lekin run qilgan runtime error olamiz. Demak mumkin emas ekan manfiy qiymat berish capacity ga:



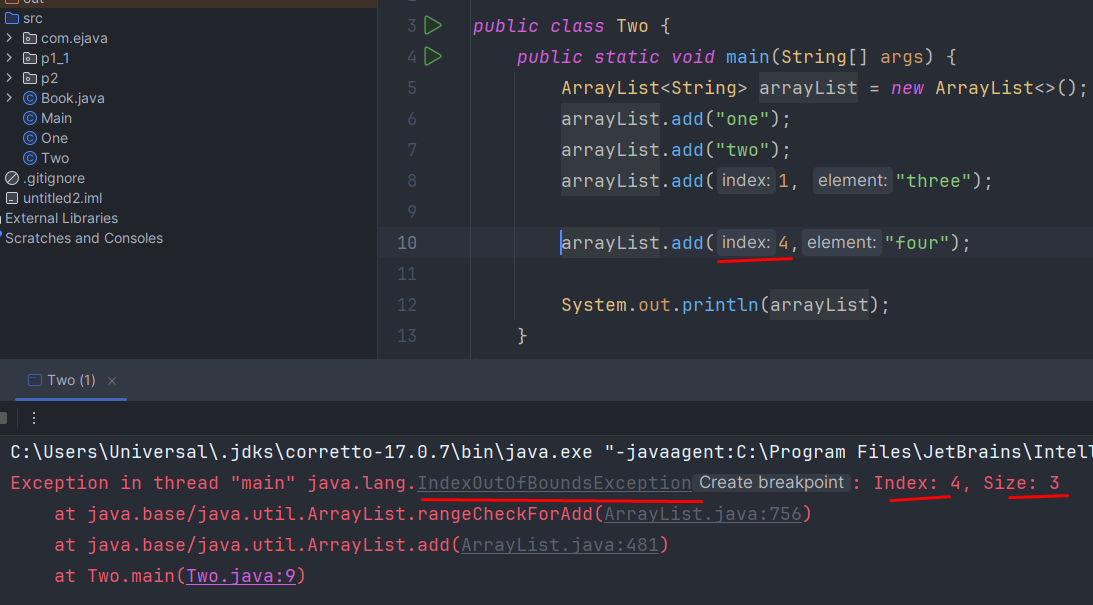
Lekin AL ga **float** yoki **double** typedagi son bera olmaymiz, chunki faqat **int** type ni qabul qiladi:



AL da **add(elem)** va **add(index, elem)** larni ishlatilishi:

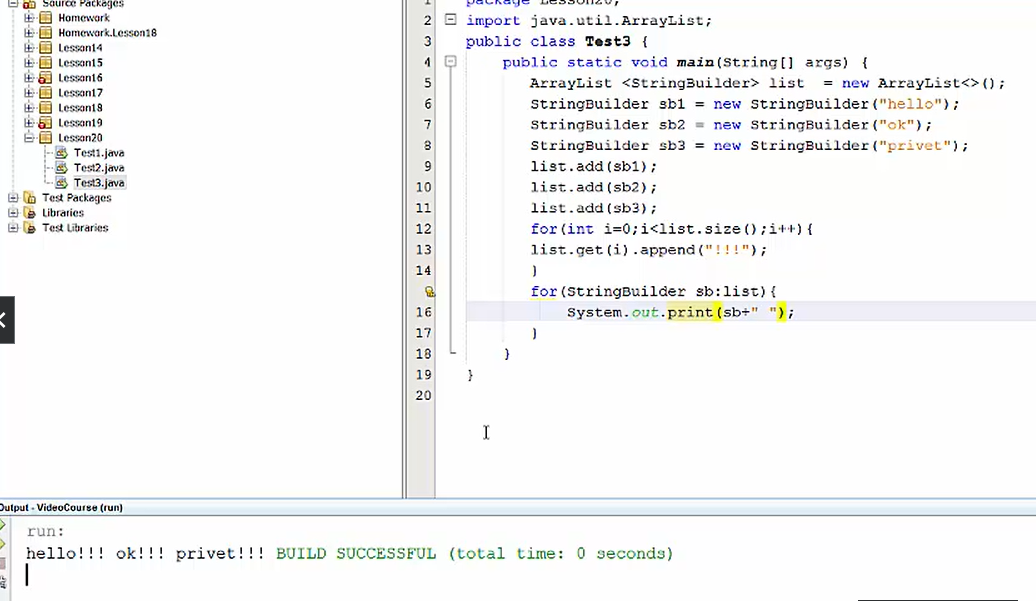


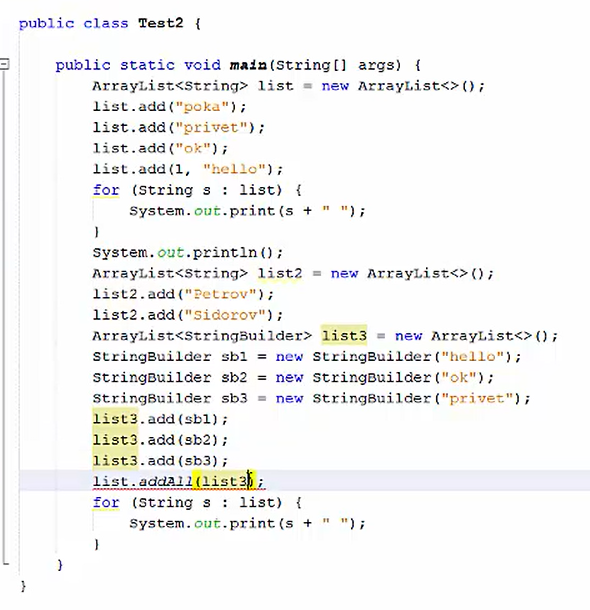
AL da uzunligidan ortiqcha element qo’shmoqchi bo’lsak, u holda error beradi. Pastda **AL** ga **3-indexni** qo’shishimiz kerak edi, lekin uzunligidan ortiqcha indexni ko’rsatganimiz uchun xatolik beradi:

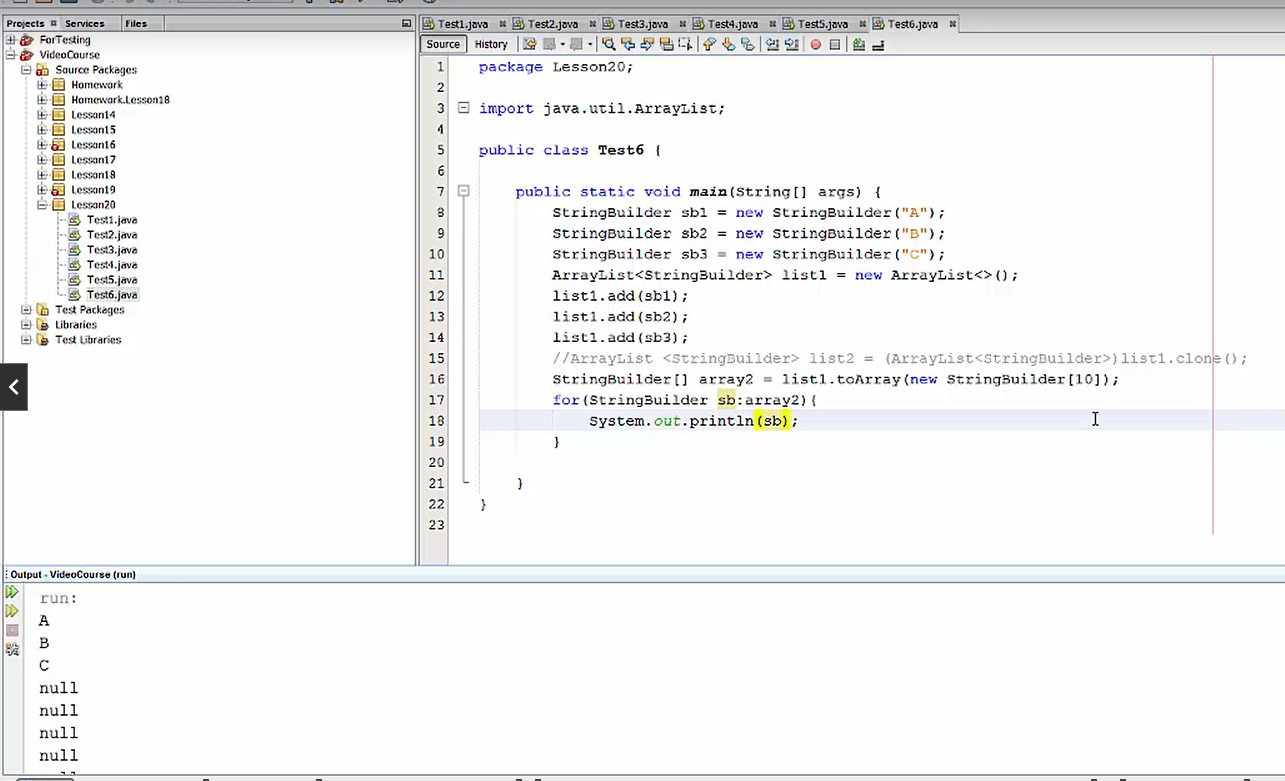


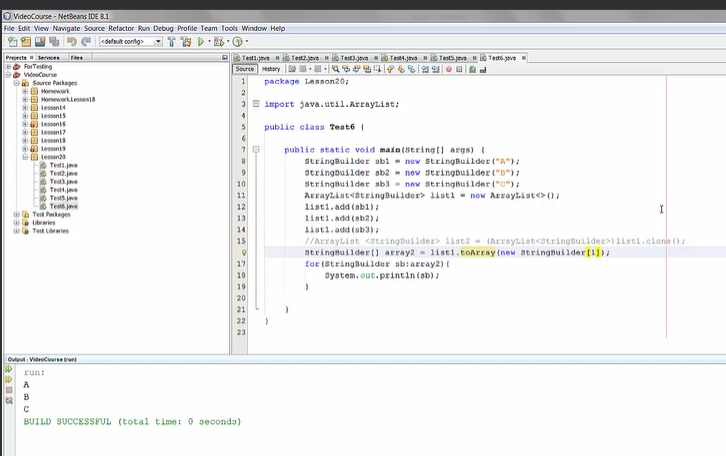
AL da **set()** methodi Stringdagi **replace()** methodi kabi ishlaydi, ya’ni berilgan **index** ga **set()** methodni ichidagi qiymat yoziladi. Eskisini o’chirib yangisini yozadi:

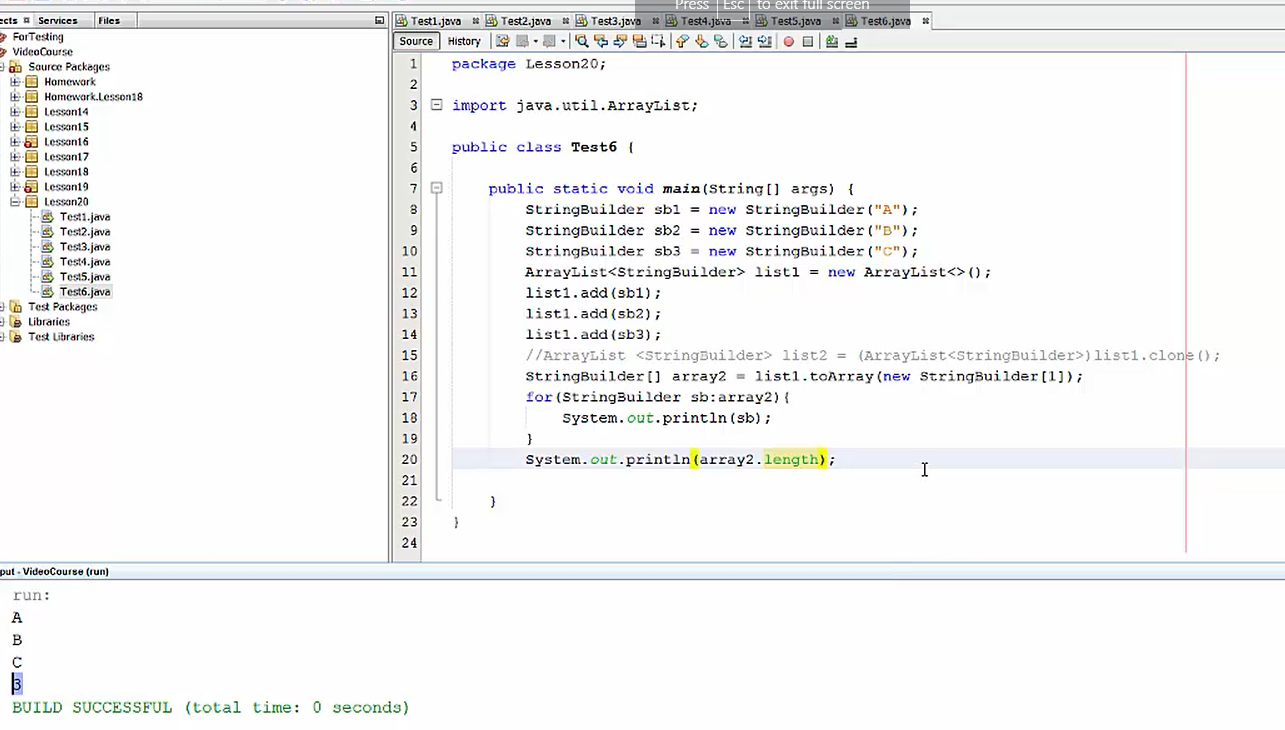




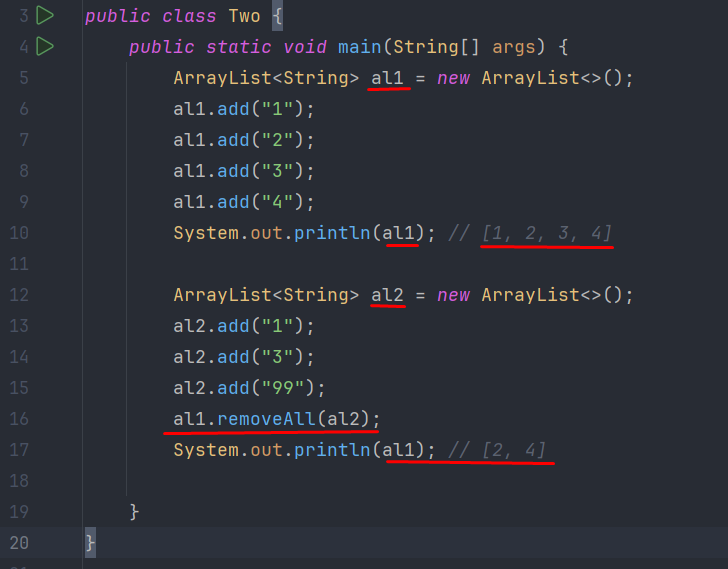




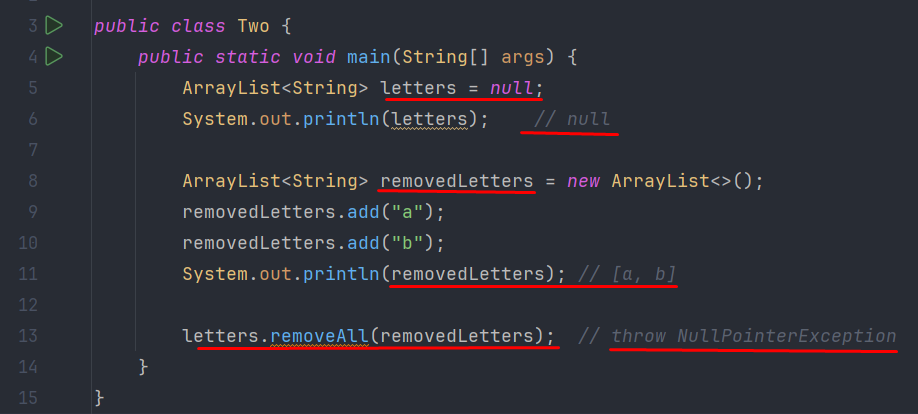




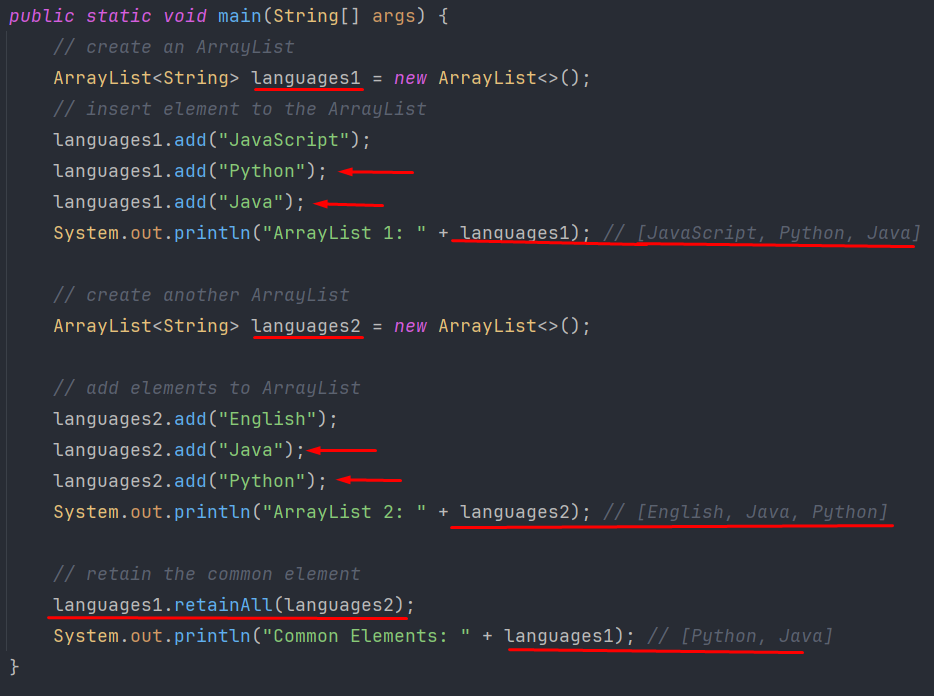
**arrayList.removeAll(Collection c)** method parameterga **Collection** qabul qiladi. Bu collection **arrayList** dan o’chirilishi mumkin bo’lgan elementlar. Pastda misolda ko’ramiz. **al1** nomli arraylist bor bizda, **al2** arraylist dagi elementlardan **al1** da borlari o’chiriladi, ya’ni **al1** dan **al2** da borlari ya’ni **“1”** va **“3”** elementlar o’chirildi:



Bu method bilan ishlayotgancha **NullPointerException** ga ehtiyot bo’lish kerak:



**al1.retainAll(al2)** bu method - **al1** collectionda **al2** collectionda bor bo’lgan elementlar saqlanib qoladi, ya’ni **2** la collection larda umumiy bo’lgan elementlar saqlanib qolinadi: Pastda ham **languages1** va **languages2** arraylistlarda umumiy bo’lgan **“Python”** va **“Java”** elementlar **languages1** collectionda saqlanib qolinyapti, qolganlari ya’ni umumiy bo’lmaganlari esa **languages1** collectiondan o’chirilyapti:



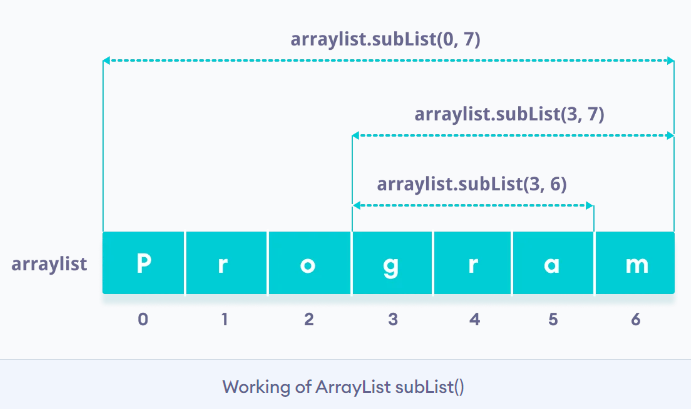
**Al1.containsAll(al2)** bu method – agar **al1** collectionda **al2** collection dagi hamma element bor bo’lsa, u holda **true** qaytaradi, aks holda **false** qaytaradi. Pastdagi misolda **languages2** dagi hamma elementlar **languages1** da mavjud emas, masalan **languages2** da **“English”** nomli element, **languages1** da mavjud emas, shuning uchun **false** qaytaradi:



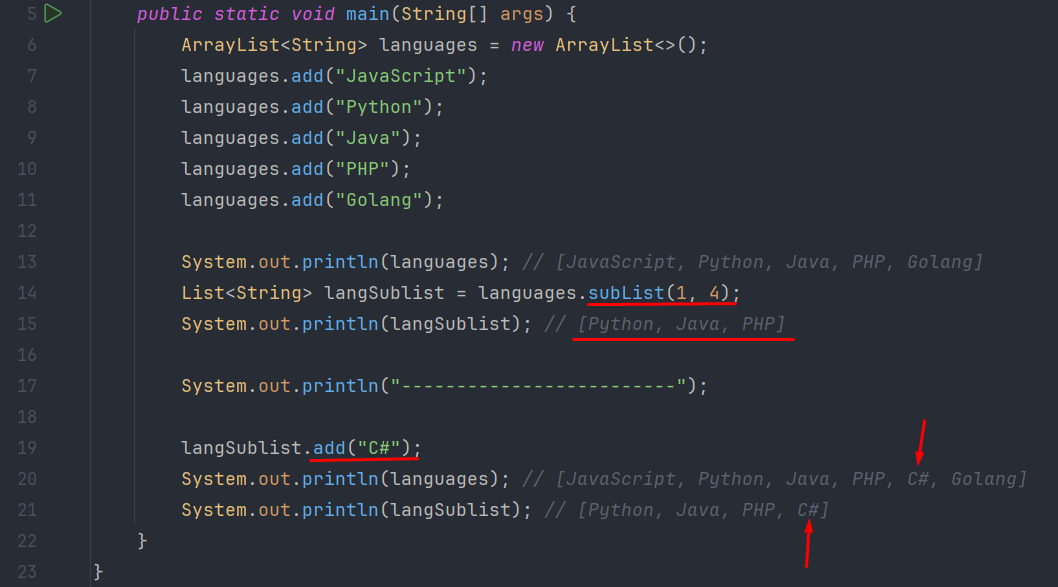
Pastdagi holatda esa **languages2** da bor bo’lgan elementlar, **languages1** da ham mavjud shuning uchun **true** qaytaradi:



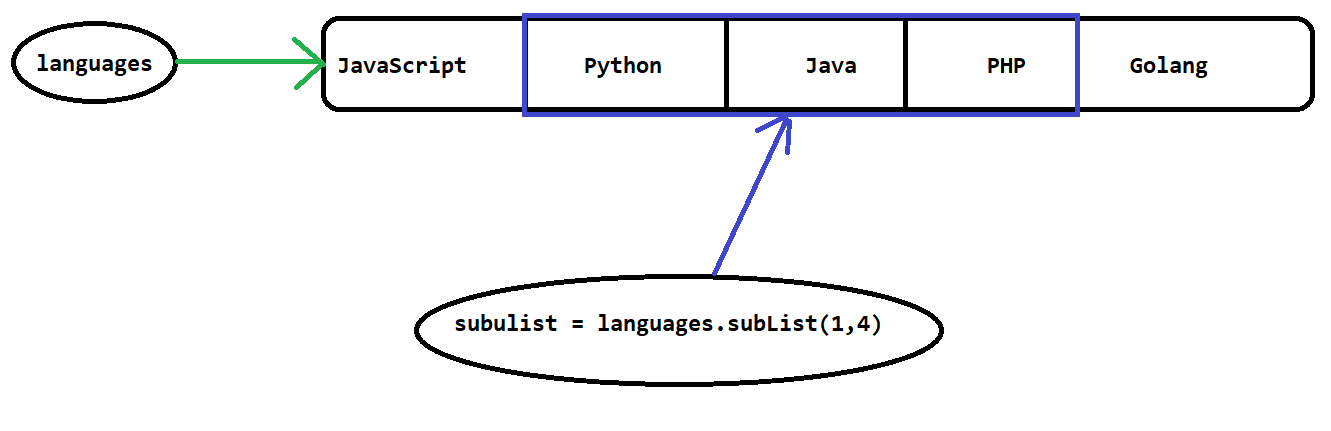
**ArrayList.subList(int startIndex, int endIndex)** bu method arrayListdan **startIndex** dan boshlab **endIndex** gacha bo’lgan qismini kesib oladi:



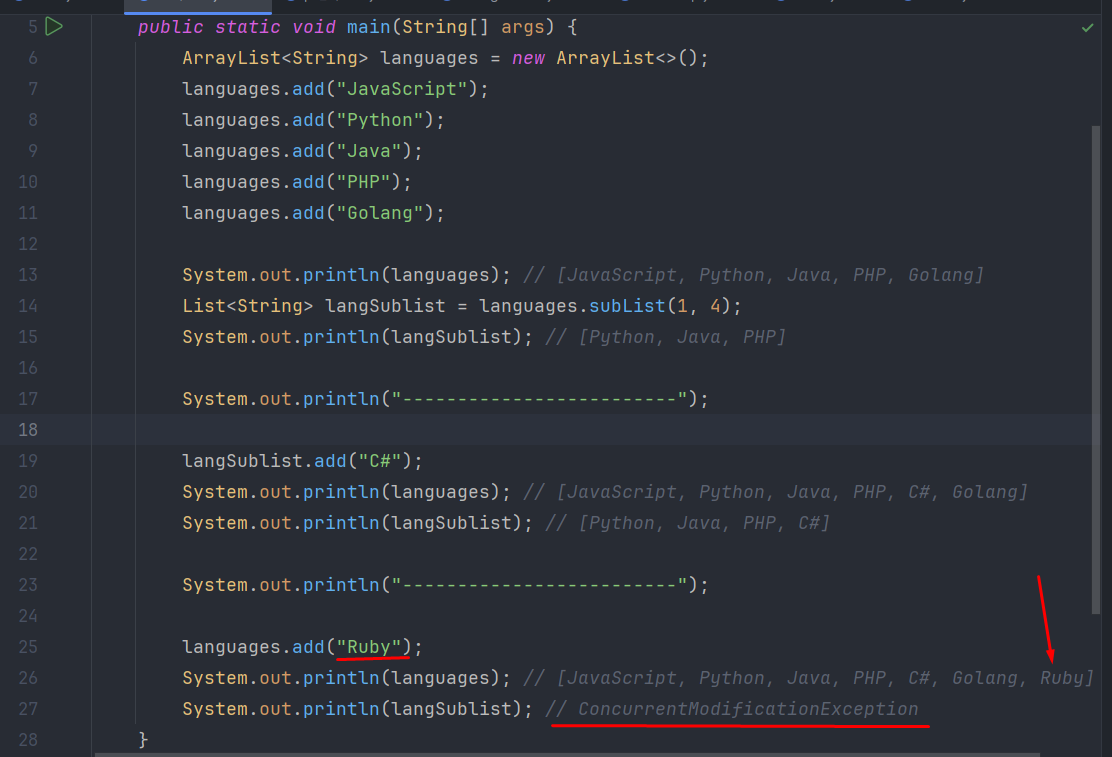
Pastdagi misolda ko’radigan bo’lsak, **langSublist** da languages dan **1**-indexdan **4**-indexgacha bo’lganlari kesib olib o’zlashtirilyapti:



**19**-qatorda sublistdan dan turib biz **“C#”** nomli element qo’shyapmiz. Bu o’zgarish esa asl **languages** listda ham aks etadi, **20-21**- qatorlarda ko’rsatilgan. Demak sublist asl listga reference qilib turibdi. **Sublist** orqali qilgan o’zgarish asl **languages** listda ham aks etyapti:

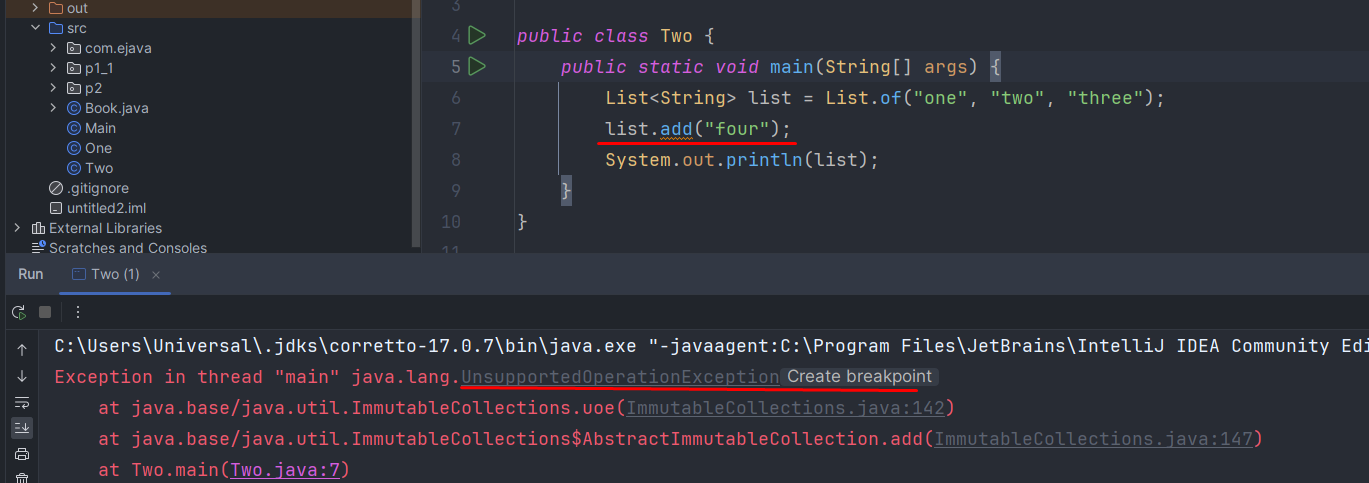


Lekin asl languages listda qilingan o’zgarish, xatolikka olib keladi, shu joyi ham bor. Pastdagi misolda asl languages listidan turib “Ruby” nomli element qo’shyapmiz 25-qatorda. Lekin bu o’zgarish asl languages listda hech qanday xatoliksiz qo’shiladi. Lekin langSublist sublistda xatolikka sabab bo’ladi(27-qator):



Nega xatolik beradi? Sababi qandaydir o’zgarish bo’ladigan bo’lsa, bu o’zgarish subList orqali bo’lishi kerak. Bu degani structural modificationlar(structural o’zgarishlar) sublist orqali va shu sublist methodlari orqali bo’lishi kerak(19-qator), aks holda xatolik beradi. Agar biz asl listdan sublist kesib olib ushlab turgan elementlarni o’chirib yuborsak, u holda biz sublistni stukturasini buzib yuborgan bo’lamiz, ya’ni sublist dagi elementlar ham o’chib ketganini ko’rishimiz mumkin. Bunday holatda sublistni strukturasi ham o’zgarib ketadi. Sublistni strukturasi esa o’zgarmasligi kerak. Shunday holatlarda shu sublistni o’zidan turib, element qo’shish, o’chirish, update qilish kerak, lekin asl listdan emas.

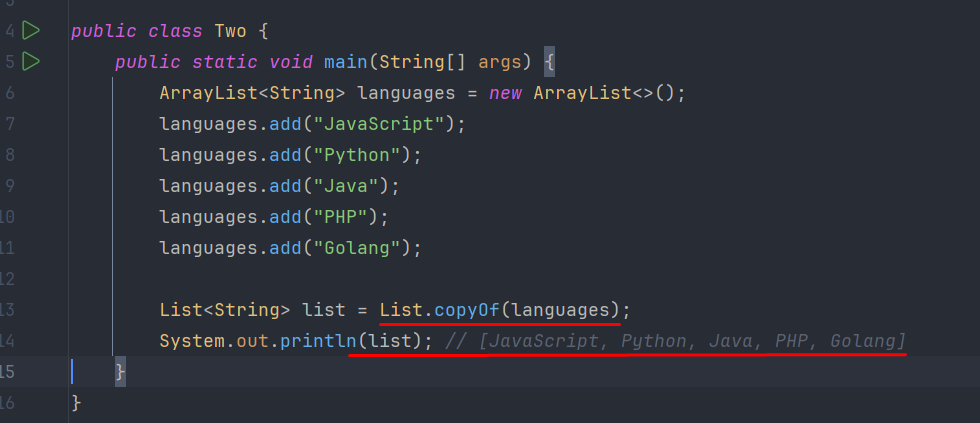
**List.of()** bu method yordamida listimizga hech qanday o’zgarish kiritgani qo’ymaymiz. Pastda shu holat berilgan. **7**-qatorga list ga element qo’shmoqchi bo’lyapmiz, lekin uni qo’shgani qo’yamayapti, xatolik beryapti:



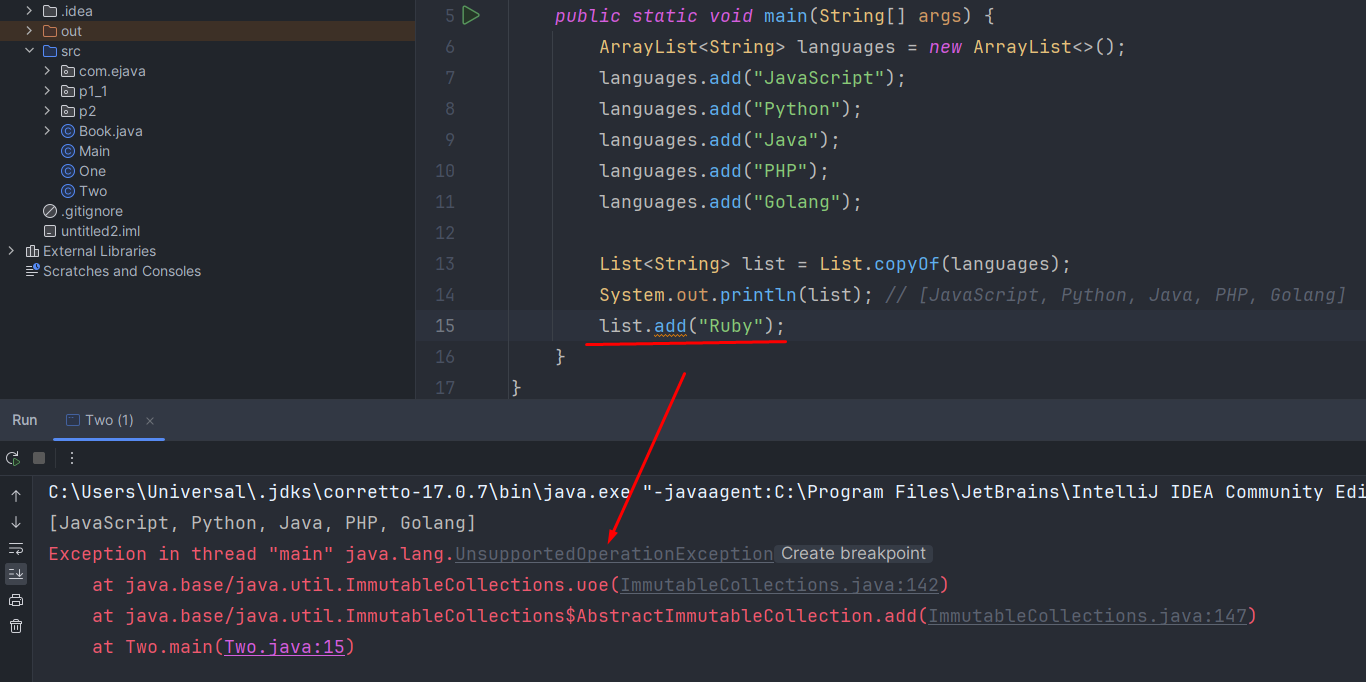
**List.of()** method o’zida **null** ni saqlamaydi, xatolik beradi:



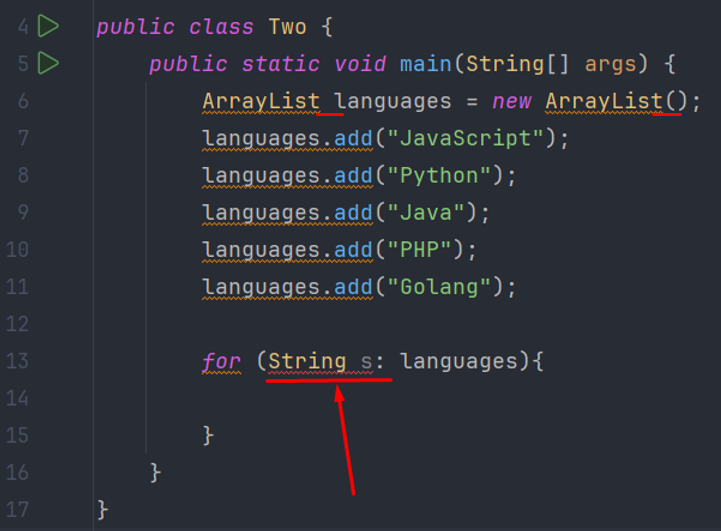
**List.copy()** bu method biror bir listdan copy oladi. Pastda **list** o’zgaruvchisiga **languages** dan copy olib o’zlashtiryapmiz:



Lekin eslab qolish kerak bo’lgan joyi, shundaki **List.of()** methodga o’xshab, bu method orqali copy olib, uni boshqa bir o’zgaruvchiga o’zlashtirsak, u holda bu o’zgaruvchini o’zgartib bo’lmaydi, ya’ni **immutable** bo’ladi. Pastda ham shu jarayon ketyapti **list** o’zgaruvchisiga endi boshqa qiymat qo’shib, o’chirib yoki update qilib bo’lmaydi, aks holda xatolik chiqadi:



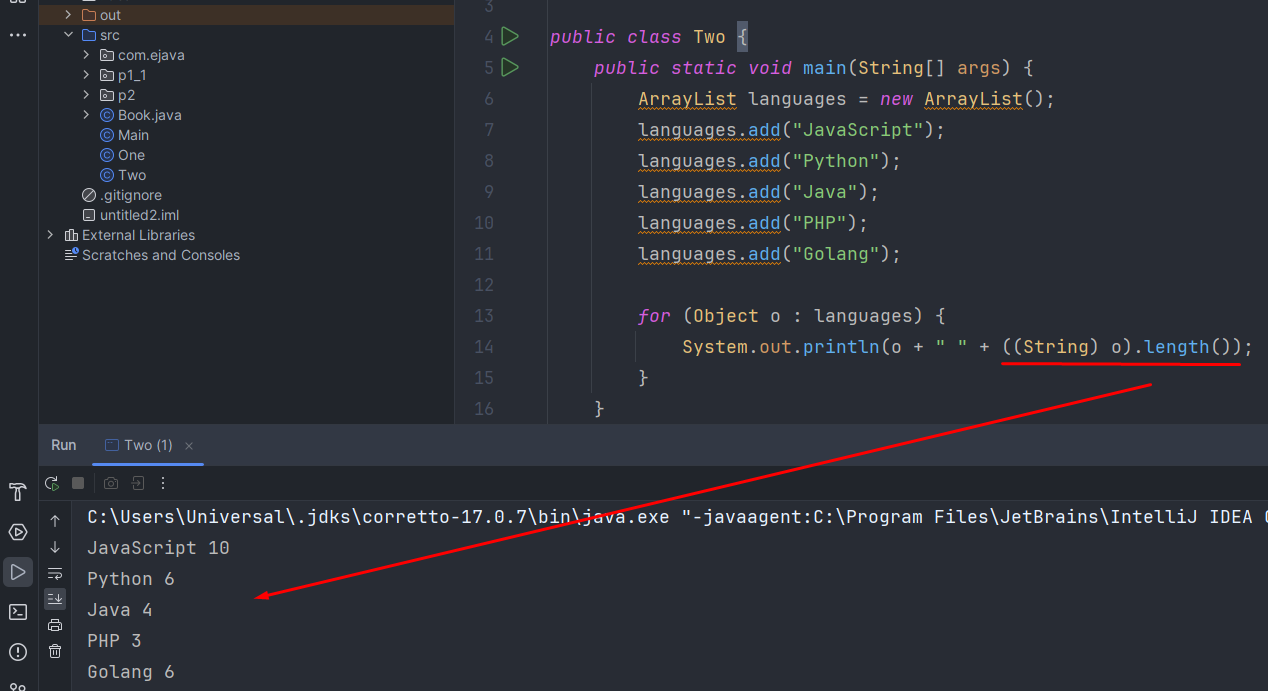
AL da agar biz qo’shmoqchi bo’lgan elementlarimizni typeni generic typeni ichida bermasak, u holda bu AL ni default type **Object** class bo’ladi. Bu degani bu AL ga istalgan typedagi qiymat qo’shishimiz mumkin degani. Masalan, pastda biz faqat String typeli element qo’shdik languages nomli AL ga. Lekin e’tibor bersangiz **13**-qatorda biz endi shu elementni forEach orqali loop qilmoqchi bo’lsak, xatolik beradi. Chunki languages o’zgaruvchimizni type **String** emas balki **Object** type dir:



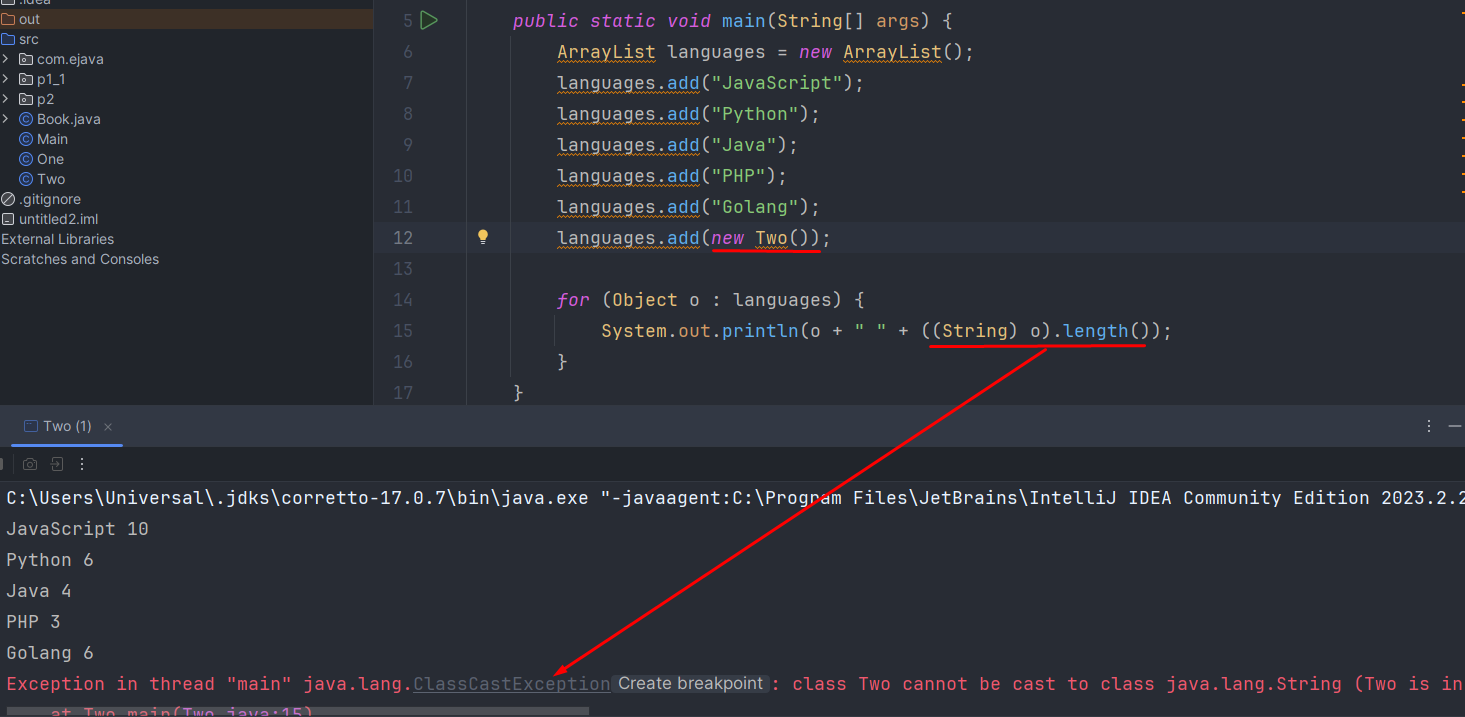
Agar typeni **Object** qilsak xatosiz ishlaydi:



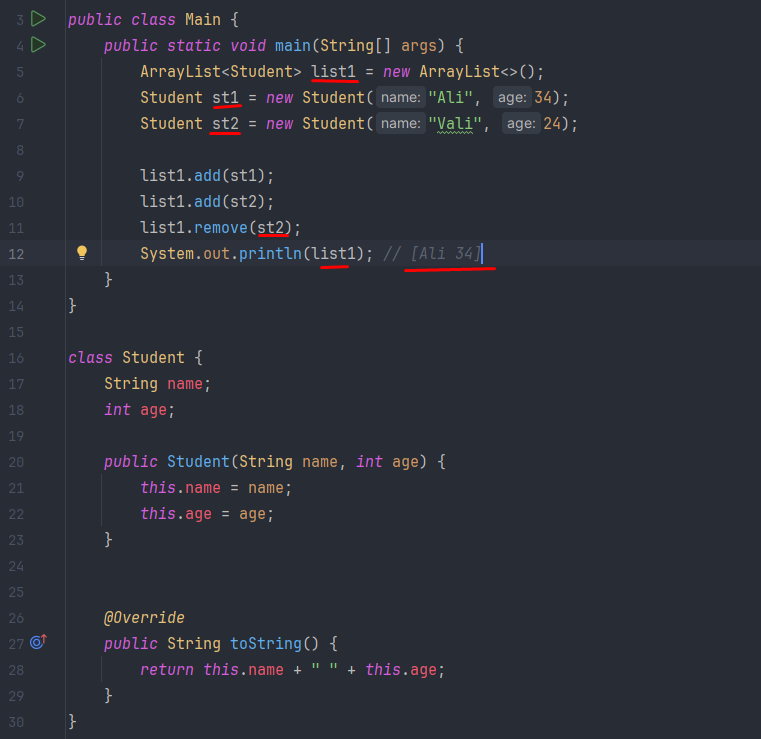
Har bir elementni uzunligini chiqarib beramiz, bunda **o** ni **Stringga** o’girish kerak.



Lekin typeni **Object** qilishni yomon tomoni ham bor, masalan pastda biz 12**-**qatorda boshqa object **Two** ni ham qo’shishimiz mumkin, chunki **languages** istalgan object oladi, type **Object** bo’lgani uchun. Lekin **forEach** da **length()** method faqat **String** type uchun ishlaydi, **Two** objectda esa **length()** nomli method yo’q, bunday vaziyatda xatolik beradi:



AL.da remove() m-di bo’lib, u o’ziga parameter sifatida 2 ta type qabul qiladi. Birinchisi biz o’chirmoqchi bo’lgan elementimizni indexini, 2-esa o’chirmoqchi bo’lgan objectimizni berishimiz kerak. Bizni 2-holat qiziqtiradi. Masalan, pastdagi misolda biz st2 objectni o’chiryapmiz va o’chayapti:



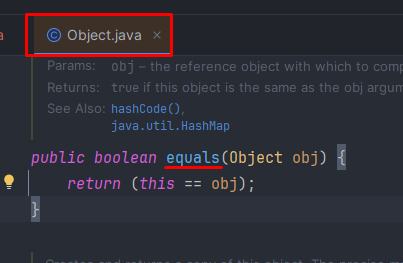
Endi keeling yangi st3 object yaratib o’chirib ko’raylik. E’tibor bergan bo’lsangiz st2 objectning ichidagi qiymatlar st3 objectniki bilan bir xil. Lekin remove() m-di orqali o’chirib ko’rsak, o’chmaganini ko’rish mumkin. Chunki st2 va st3 objectlarni ichidagi qiymatlar bir xil bo’lsa hamki, ular har xil joyga reference qilyapti. Shuning uchun o’chirib bo’lmaydi:

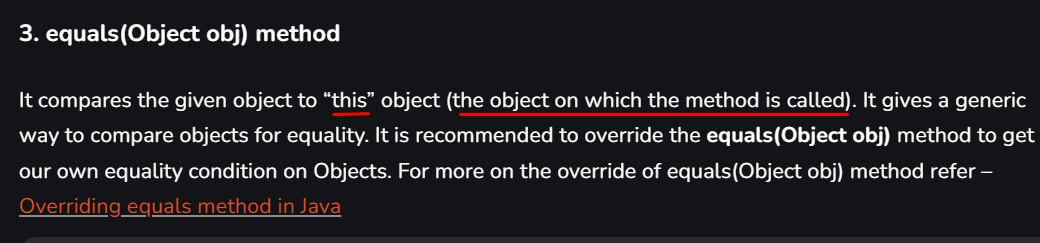


Endi savol tug’ilishi mumkin. Unda ichidagi qiymatlar bir xil bo’lsa, yangi object bo’lishiga qaramasdan qanday qilb o’chirish mumkin. Buni yo’li **Student** classimizda **equals()** m-dini override qilib qayta yozishdir. Biz **equlas()** m-dni qayta yozganimizda istalgan field bo’yicha ularni tenglini tekshirib, teng bo’lsa **true** aks holda **false** qaytaraishimiz mumkin. Pastdagi holatda **remove()** m-d o’chiradi:

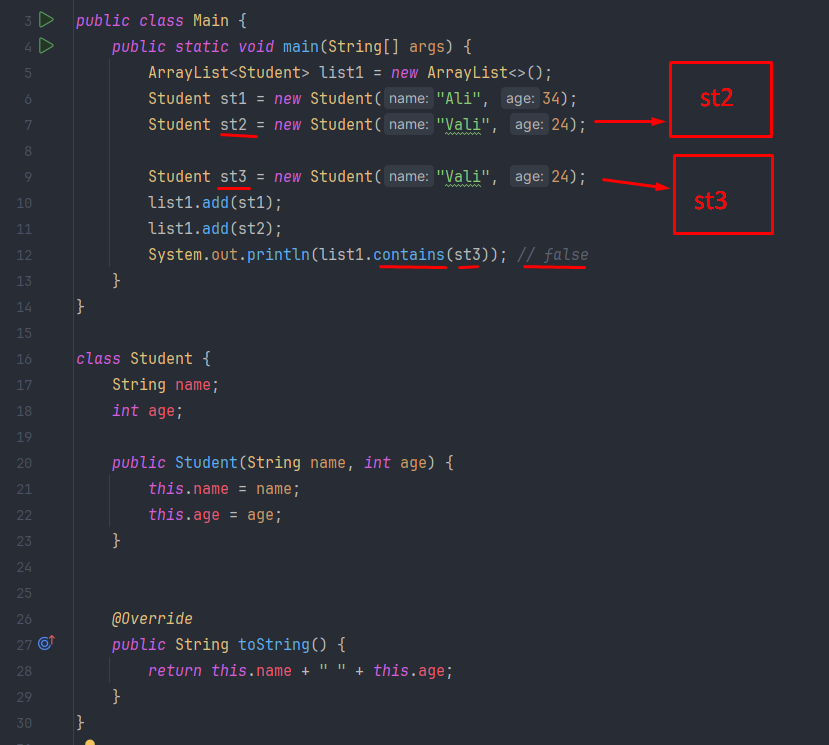
Yuqoridagi holatda o’chirishga sabab **AL**.dagi **remove()** m-d Object classidagi **eqauls()** m-dini ishlatyapti.Biz o’sha **equals()** m-ni override qilib qayta yozdik.

**Object** classidagi **equals()** m-dini umumiy ko’rinishi quyidagicha:

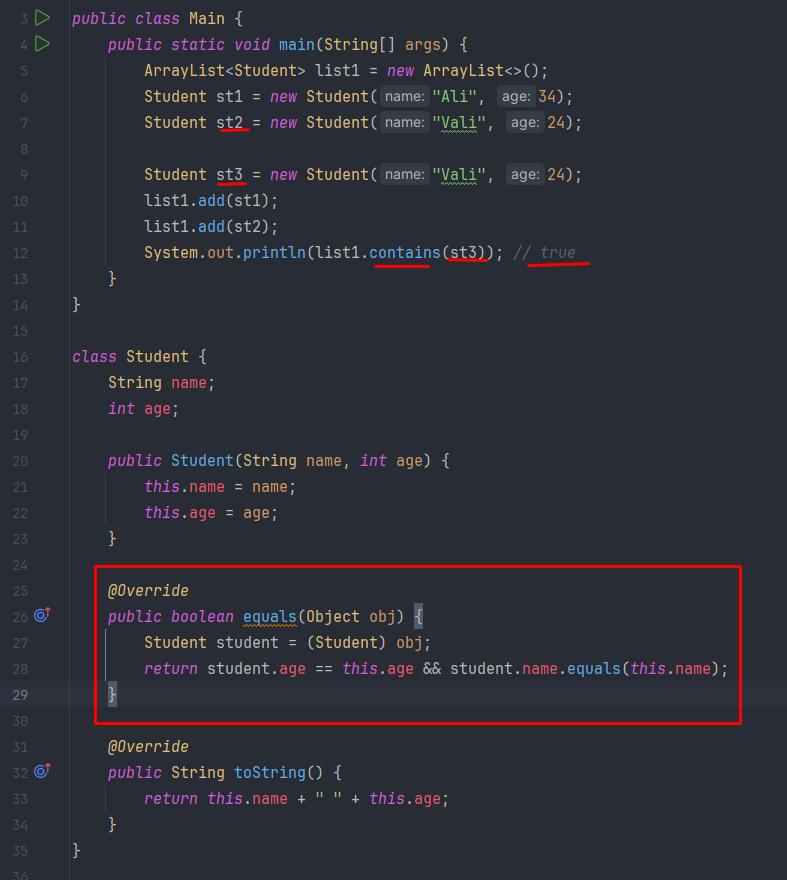




Xuddi shunday remove() m-dga o’xshab, contains() m-d ham st2 va st3 objectlar garchi bir xil qiymat saqlayotgan bo’lsa hamki, ularni reference turlichadir. Shuning uchun pastdagi holatda false qaytadi contains() m-ddan:



Lekin biz equals() m-dini override qilsak, u holda true qiymat qaytadi:



ArrayList methodlarida pastda qizil bilan belgilanganlarida yuqoridagi **remove()** va **contains()** methodlarida bo’lgan holat yuz beradi. Shuning uchun belgilangan methodlarda ham **equals()** methodini bemalol **override** qilishimiz mumkin:

