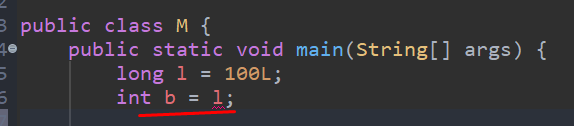
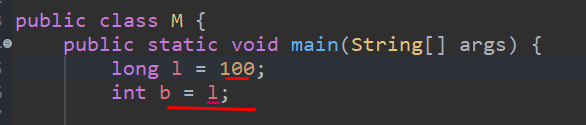


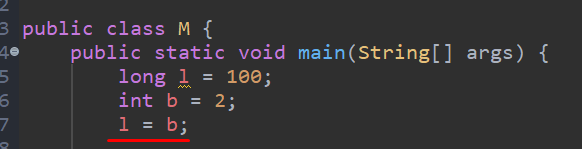
* Biz **long** typedagi o’zgaruvchini **int** type dagi o’zgaruvchiga o’zlashtira olmaymiz. Sababi **long** typeni oraliq chegarasiga kiradigan hamma sonlar ham, **int** type oraliq chegarasiga kirmaydi. **Long** typeni oraliq chegarasi katta. Shuning uchun xatolik beradi:



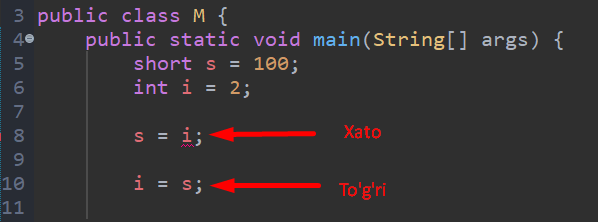
Farqi yo’q xattoki **l** o’zgaruvchini oxiridan **L** ni o’chirsak ham:



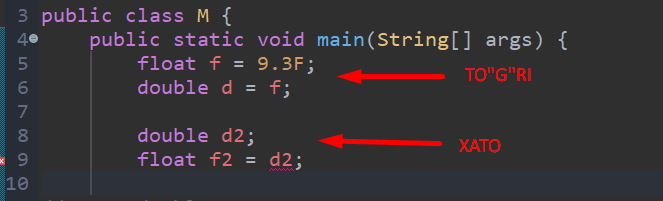
Lekin **int** typedagi o’zgaruvchini **long** typedagi o’zgaruvchiga o’zlashtirsak, xato bermasdan ishlaydi. Sababi **long** type ni oraliq chegarasiga **int** ni oraliq chegarasi tushadi:



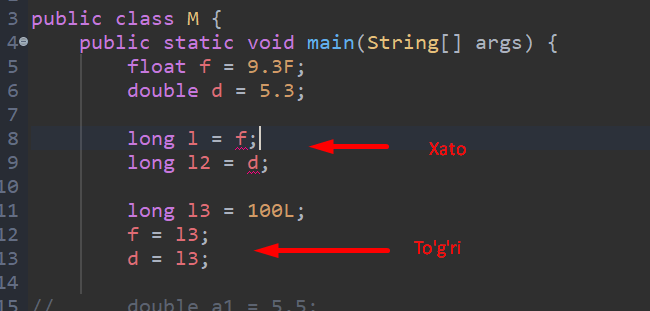
Xuddi shunday **short** typeli o’zgaruvchiga **int** typeli o’zgaruvchini o’zlashtirib bo’lmaydi(**8-qator**), chunki **short** typeni oraliq chegarasi **int** nikidan kichkina. Lekin aksi bo’lishi mumkin(**10-qator**):



Xuddi shunday **double** va **float** uchun ham. **float** ni **double** ga o’zlashtirish mumkin, lekin aksi mumkin emas xato bo’ladi:

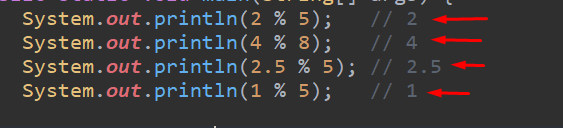


Bundan tashqari **long** typega **float** va **double** ni o’zlashtirib bo’lmaydi. Sababi **float** va **double** ni oraliq chegarasi **long** nikidan kattadir(**8-9-qator**). Lekin aksi bo’lishi mumkin, ya’ni **float** va **double** ga **long** ni o’zlashtirishimiz mumkin(**12-13-qator**):

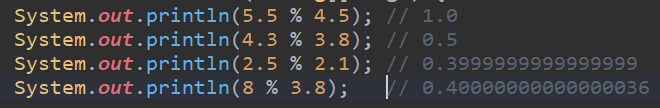


**Xulosa qiladigan bo’lsak, kichkina oraliq typeli o’zgaruvchiga katta oraliq type li o’zgaruvchini o’zlashtirib bo’lmas ekan, lekin aksi bo’lishi mumkin!!!**

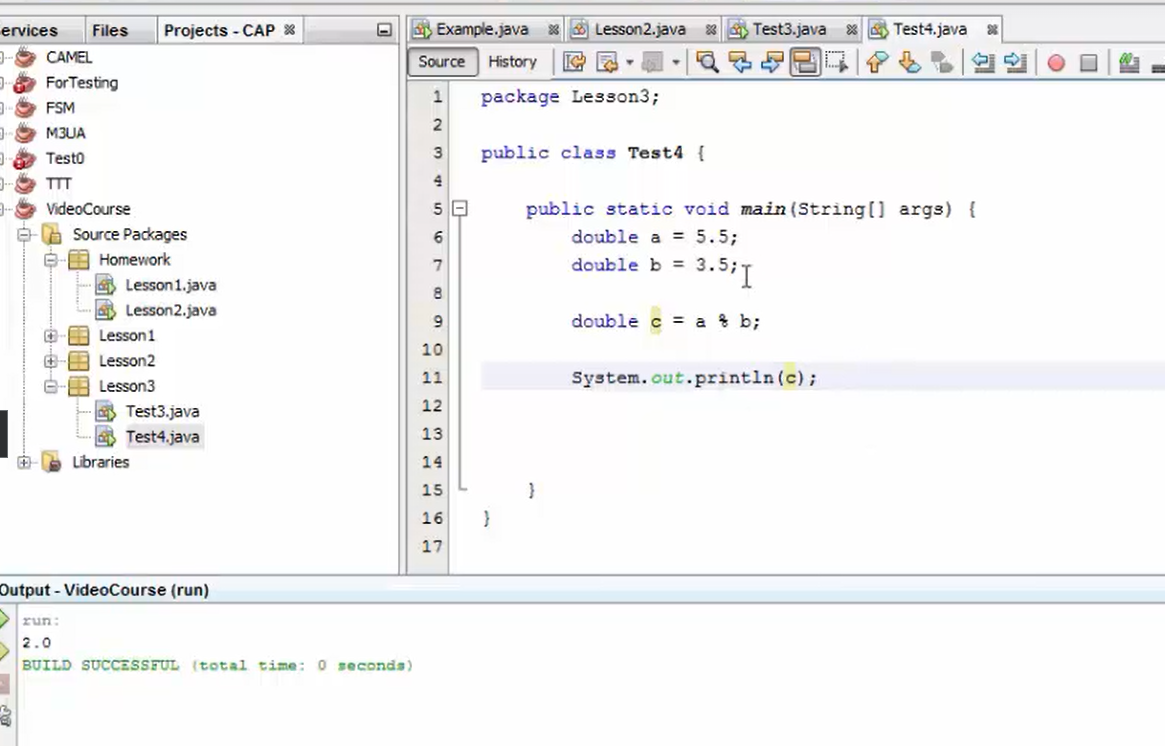
* Agar bizda **bo’linuvchi**miz **bo’luvchi**dan kichkina bo’lsa, bunday holatlarda qoldig’ini olishda, har doim **bo’linuvchi**miz chiqadi natija sifatida:

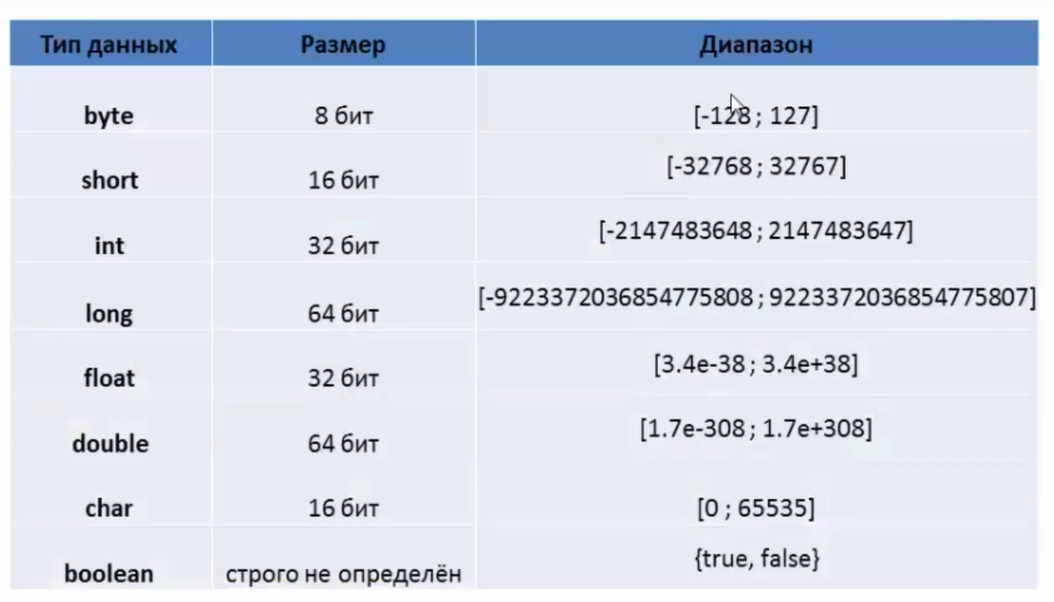


Agar sonimiz float-pointing bo’lsa, qoldiq sifatida yetmay qolgani ayriladi:

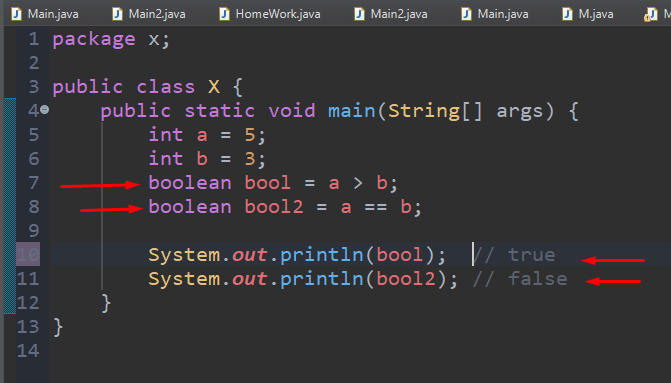


Boshqa misol yana:

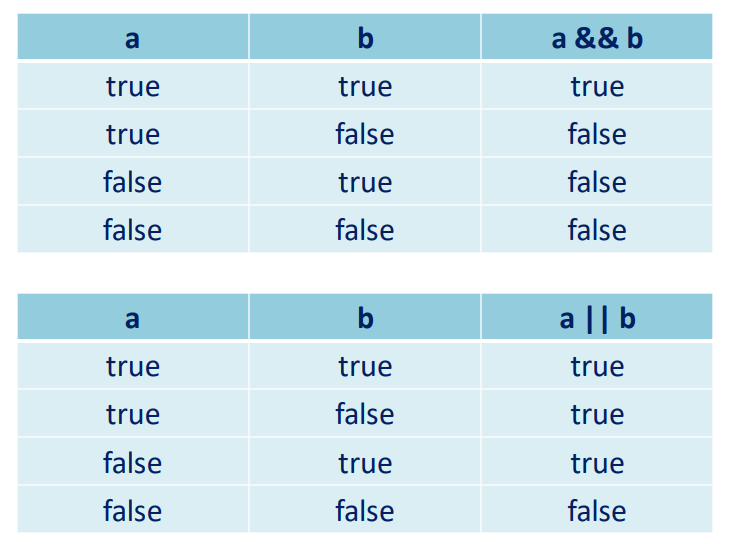




* **Boolean** li type ga bemalol expressionni tayinlashimiz mumkin. Bunda faqat shu expressionimiz albatta Boolean qaytarishi zarur. Ularga **>, <, >=, <=, ==, !** kabilar kiradi:

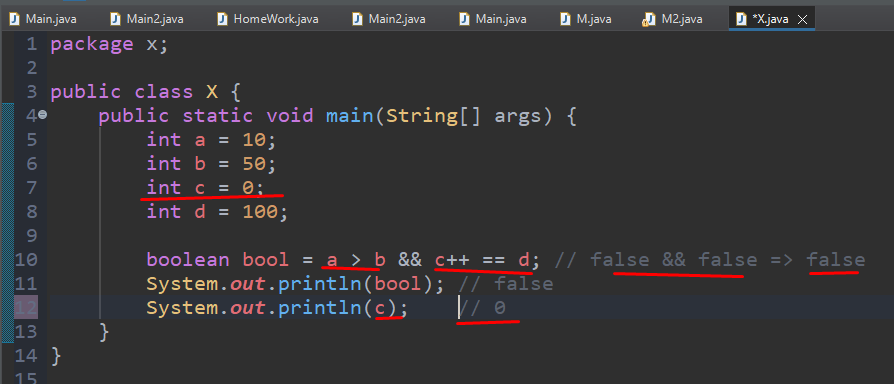


**- Logical** operators:

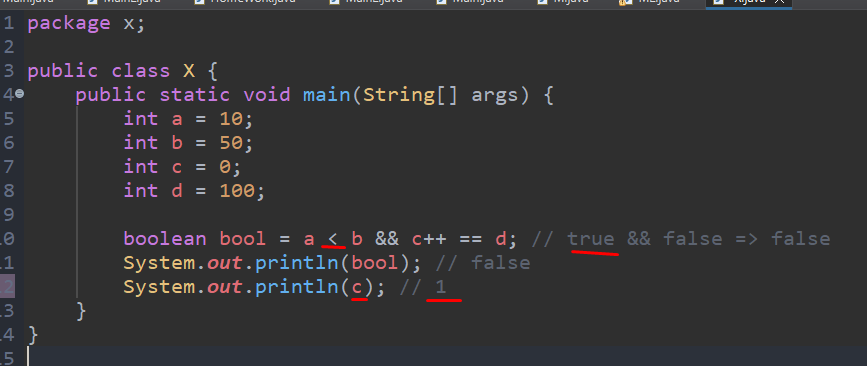


Logical **&&(AND)** va **||(OR)** operatorlarini ishlash prinsipini bilib olaylik.

**AND** operatorida aqalli bittagina **false** bo’lsa, undan keyingi qismini ko’rib o’tirmasdan to’xtatib chiqib ketadi. Masalan, pastdagi misoldan ko’rish mumkinki **a=10** va **b=50** dir. **a>b** ifoda o’z-o’zidan **false** qaytaradi. Tabiiyki **&&** dan keying qism(**c++ == d**)ni ko’rmasdan tashlab ketadi. Shuning uchun **c++** qismi bajarilmaydi va **c=0** ligicha qolaveradi:

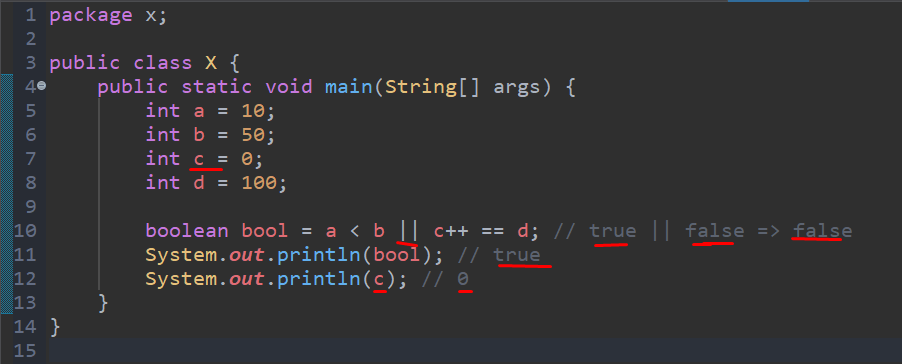


Endi yuqoridagi kodda faqatgina **a>b** ni **a<b** ga o’zgartiramiz. **10<50** ifoda **true** qaytadi, shuning uchun **&&** dan keyingi qismga o’tadi va **c++ == d** ifodani bajarib tekshirib ko’radi. **c++** qismi bajarilgani uchun, **c=1** bo’ladi:

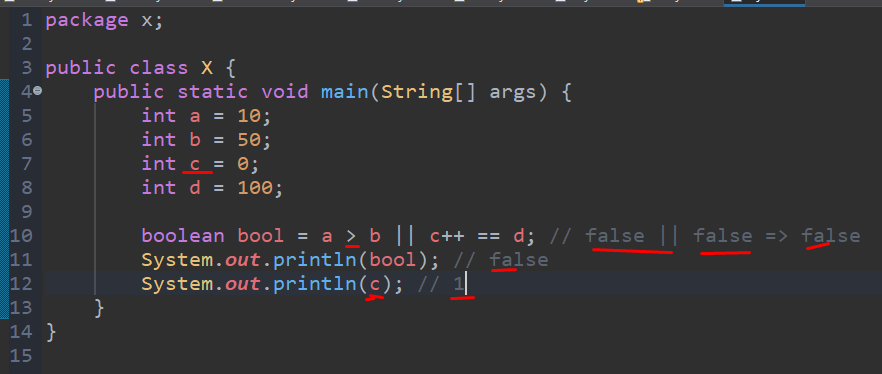


Demak, **AND(&&)** da aqalli bitta **false** uchrasa, o’sha **false** dan keyingi logical operatorlarni va ifodalarni ko’rib ham o’tirmaydi.

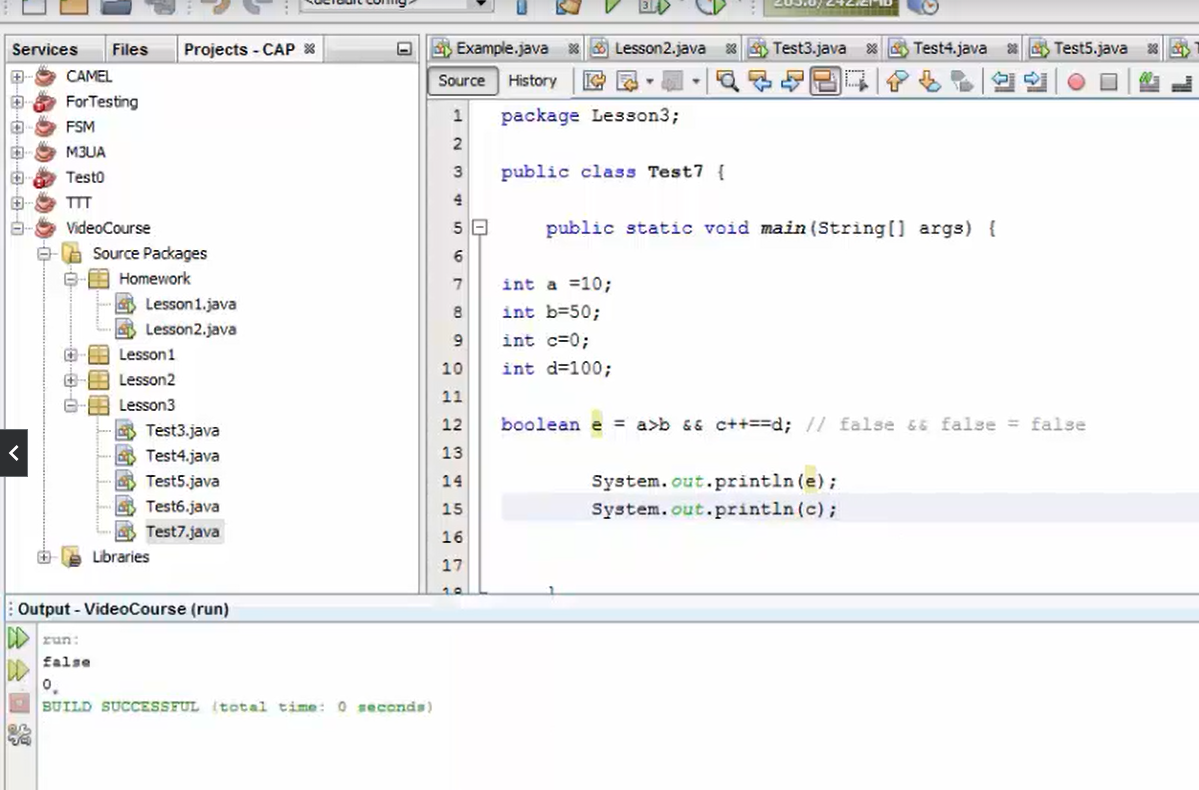
**OR(||)** da ham xuddi shunday ishlaydi, faqat bitta **true** bo’lguncha ko’radi. Agar bitta **true** uchrasa, undan keying logical operatorlarni va ifodalarni ko’rib o’tirmaydi. Pastdagi misolda ko’rish mumkinki, **a<b** ya’ni **10<50** dan **true** qaytadi. **OR** da birinchi **true** ni ko’rsa, kodni keying qismi **c++ == d** bajarilmaydi va **c++** ifoda ham bajarilmaydi **va c=0** ligicha qolib ketadi:

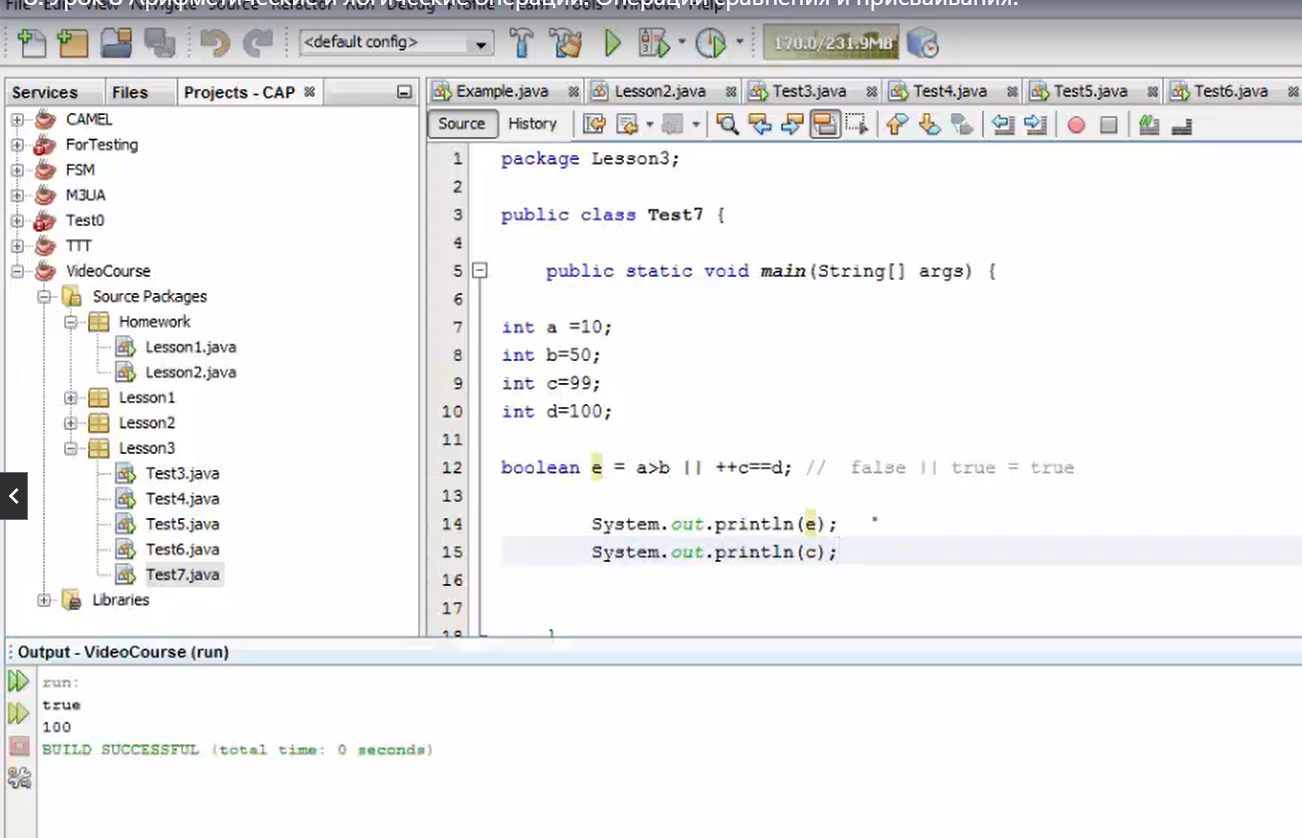


Yuqoridagi koddan faqat **a<b** ni **a>b** ga o’zgartiramiz. Shunda **10>50** dan **false** qaytadi. **OR** bo’lgani uchun keyingi **true** ni qidirish uchun **c++==d** ifodaga o’tib tekshiradi. Shu payti **c++** ifoda ham bajariladi va **c=1** bo’ladi:

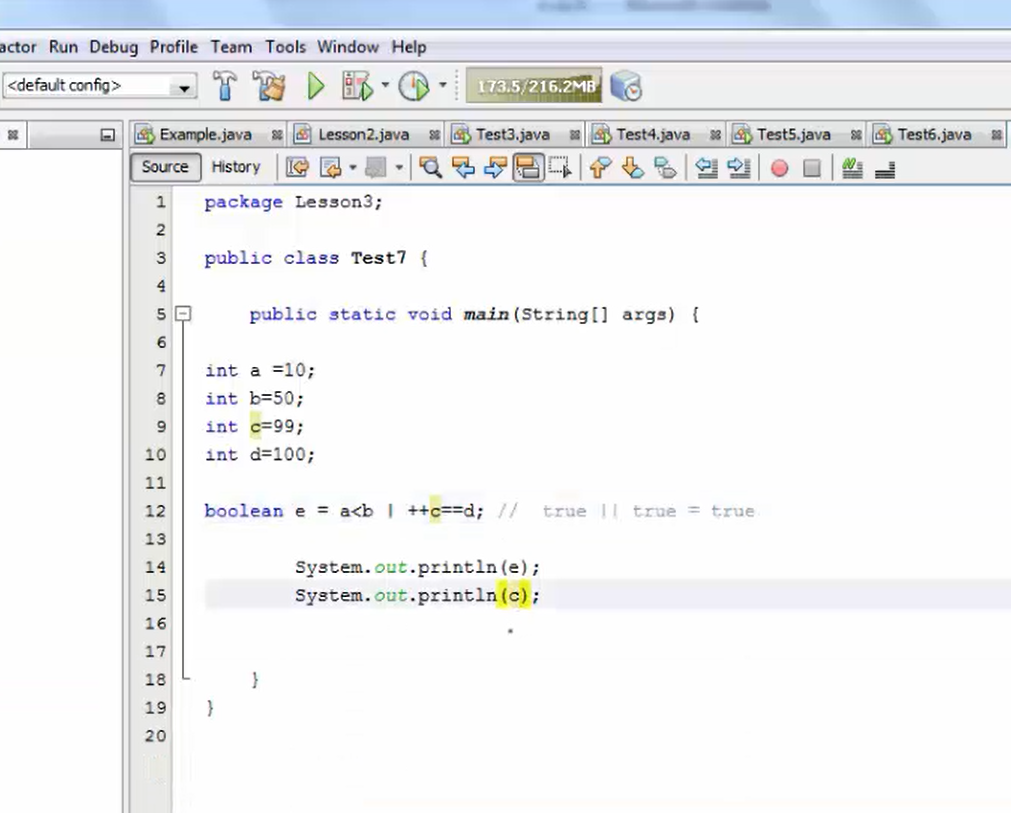


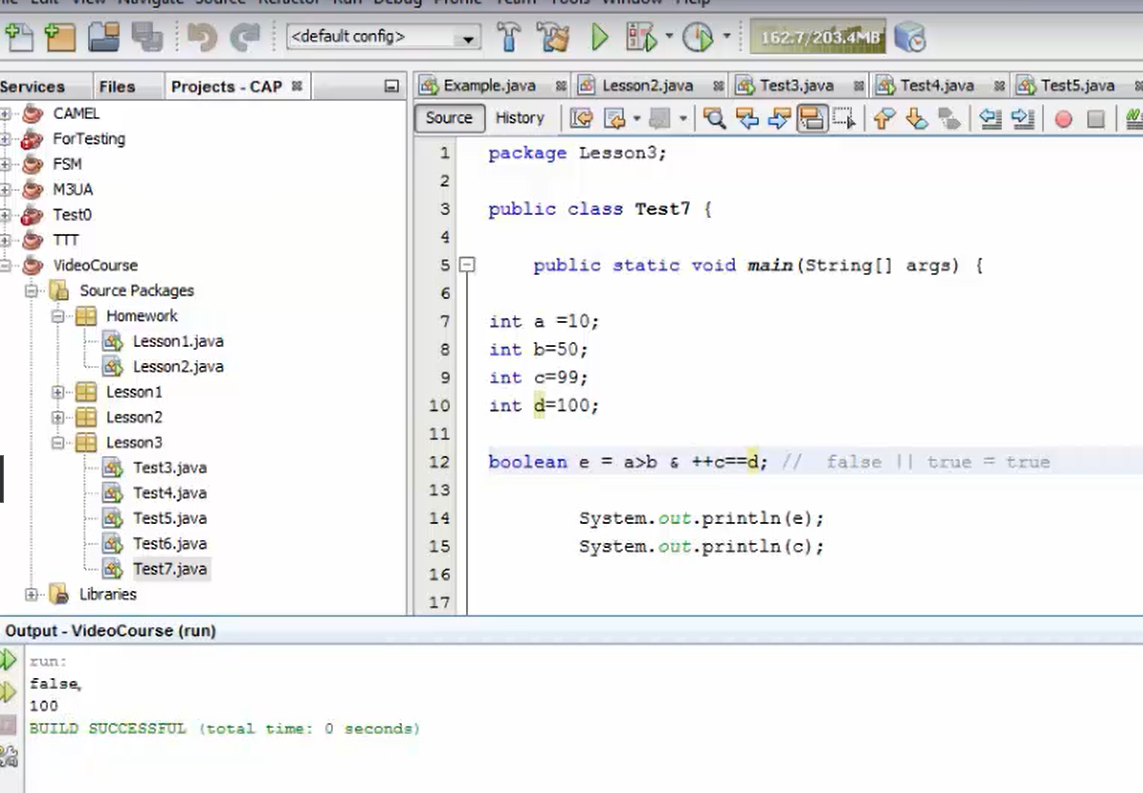
Yana boshqa misollar:

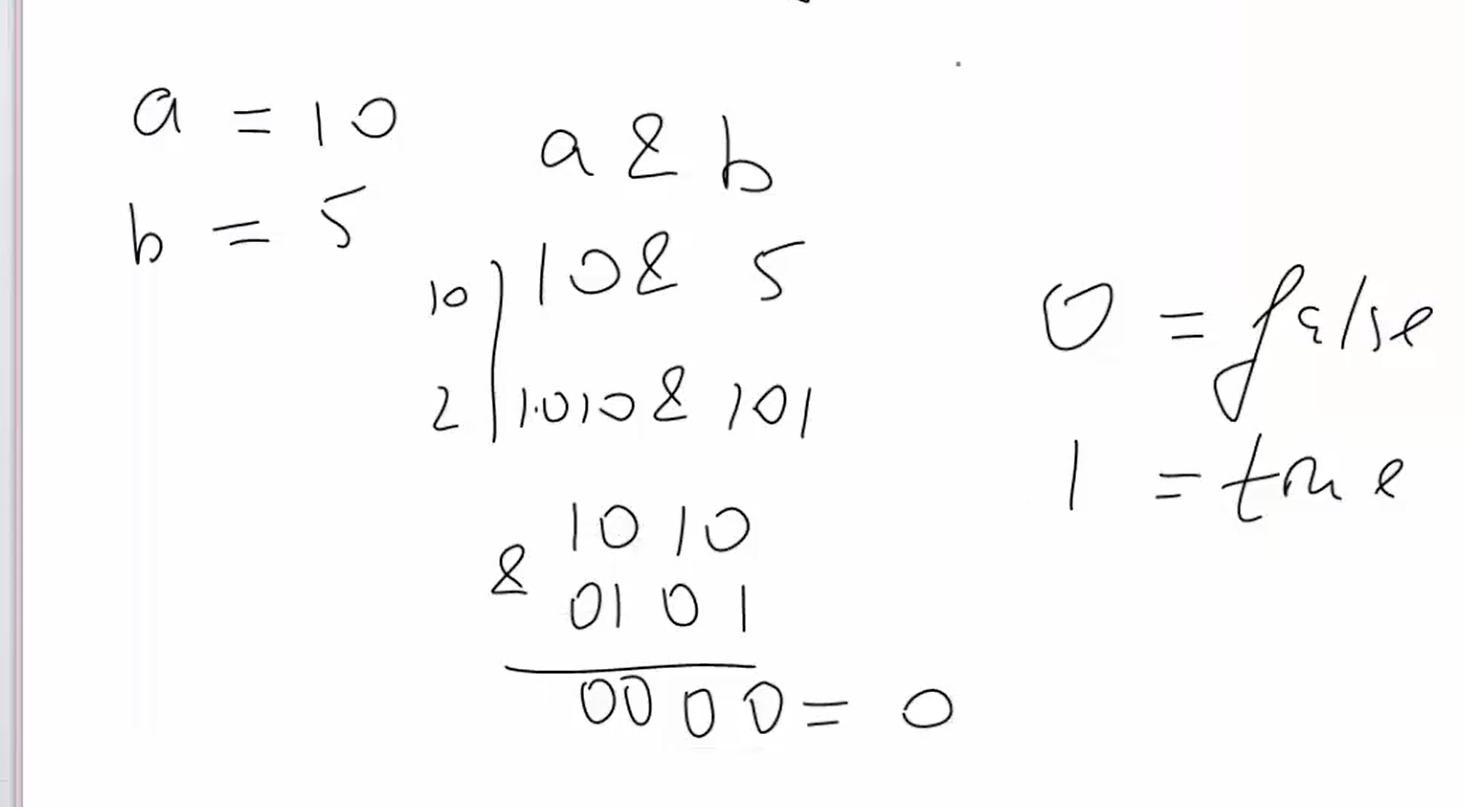




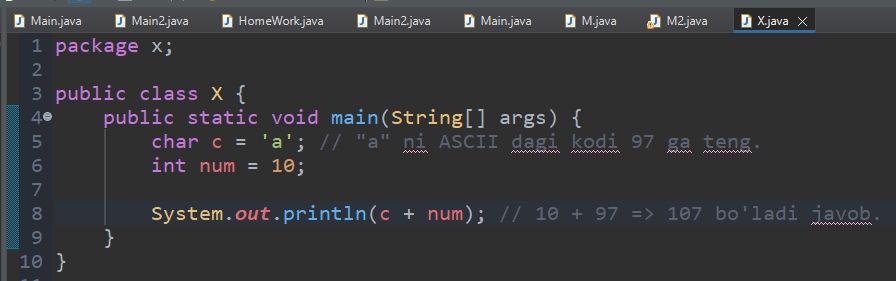
**Bitwise operatorlar:**







**Char** type bilan ishlashda bir qiziq holat bor. Agar biz **char** type ga biror **simvol**ni o’zlashtirib, shunga sonni qo’shsak, u holda shu **char** simvolni **ASCII** dagi kodini oladi va shu kodga **int** type o’zgaruvchini qo’shadi:



Yoki boshqa bir misol:

