სიის სცენა ყველა კლასისთვის ერთნაირია და შესაძლებელია დაიწეროს ერთი კომპლექტი (fxml / stage / controller), რომელიც ყველა სიისთვის ერთნაირ ვიზუალს გამოიტანს, თუმცა runtime-ში გადაწყვეტს რომლი კლასის ელემენტებით აავსოს სია.

ამ წესს არ ექვემდებარება balanceAccount განყოფილება იმ მიზეზით რომ მისი ელემენტების ვისუალი არის treeTable და არა table. ამიტომ მას მოემსახუროს სხვა კომპლექი (fxml / stage / controller), რომელიც პასუხისმგებელი იქნება treeTable-თან ურთიერთობაზე. თუ მომავალში გამოჩნდა რაიმე განყოფილება რომელსაც ასევე TreeTable ჭირდება თავისი სიის დასახატად, მისთვის უკვე გვექნება გამზადებული კომპლექტი, რომელიც ანალოგიურად დახატავს და ანალოგიურად იურთიერთებს ამ ახალი კლასის სიასთან ისე, როგორც balanceAccount-თან მოქმედებდა.

თუ გამოჩნდება კიდევ ახალი განყოფილება, რომელიც საერთოდ განსხვავებული ტიპის ცხრილის ვიზუალს იყენებს ვიდრე table და treeTable-ებია, მაშინ იმისთვის კვლავ ახალი კომპლექტი შეიქმნება.

იმისათვის რომ კომპლექტში stage-თან და fxml-თან ერთად შევიდეს controller-ი, საჭიროა reAssignTable მეთოდი თვითონ არ ქმნიდეს ცხრილში გამოსაჩენი ელემენტების სიას. არამედ ეს სია გადაეცემოდეს პარამეტრად ისე, როგორც ეს MiniTablesController-შია. ამით ფაქტიურად რამოდენიმე საერთო ვიზუალის მქონე სიის განყოფილებებს მოემსახურება ერთი კომპლექტი და თავიდან ავირიდებთ კოდის განმეორადობას.

რაც შეეხება დიალოგის შემთხვევას, აქაც შეიძლება საერთო კომპლექტის შემოტანა თუმცა მხოლოდ იმ ტიპის კლასებისთვის, რომლებსაც მსგავსი დიალოგის ვიზუალი აქვთ. მაგალითად, process type კლასებს აქვთ საერთო ვიზუალი (იგივე ვიზუალს ატარებენ invoiceReissuing, invoiceClarify, clientStatus და სხვა). თუმცა ცვადია არსებობს კლასები რომლეთა ვიზუალი განსხვავებულია process type-ებისგან. მათთვის იგივე სცენა არ გამოგვადგება. შესაბამისად არც ამ სცენის კონტროლერი. Stage-იც რომელიც ხატავს სცენას და ურთიერთობს ამ სცენის კონტროლერთან მეტწილად დამოკიდებულია მათზე, ამიტომ გამოდის რომ განსხვავებული ვიზუალის მქონე დიალოგს ჭირდება სხვა კომპლექტი fxml / stage / controller-ისა. მაგალითად კლიენტს სხვა კომპლექტი უნდა ქონდეს, ინვოისს სხვა, process type-ს სხვა და ა.შ.

დიალოგის შემთხვევაც იგივე გამოდის, რაც სიის დროს გავაკეთეთ. უბრალოდ სიის სახით წარმოდგენა ბევრ შემთხვევაშია ერთმანეთის მსგავსი და უკეთესად ჩანს კომპლექტის სარგებლიანობა. დიალოგის შემთხვევაში კი ცოტაა შემთხვევა, როცა სხვადასხვა კლასისთვის დიალოგის ვიზუალი ერთმანეთს გავს.

განსახილველია როგორ მოხდება runtime-ში კლასის შესაბამისი დიალოგის გამოჩენა. ამაზე დავწერ მოგვიანებით....

ფილტრიც ანალოგიურია, როგორც დიალოგი - ახალი კომპლექტი გაკეთდება ფილტრისთვისაც, რომელიც დიალოგის კომპლექტის დონეზე იქნება package-ებში.

თუ შევეცდებით დახატვის მხარე ჯავას კოდში გადავიტანოთ ეს იდეურ ცვლილებას არ გამოიწვევს, ანუ ვერ შევძლებთ ცლიენტის და ინვოისის დიალოგი ერთმანეთს დავამსგავსოთ მათ მაინც ცალ-ცალკე უნდა ხატონ სცენა, ოღონდ ხატვის პროცესი იქნება ჯავას მხარეს.

ComboBox\_ების შემთხვევაში 2 ვარიანტია:

ან generic-ები უნდა იყვნენ, მაგრამ ელემენტების სია უნდა დაესეტოთ გარედან, იმ კლასიდან სადაც აღიწერებიან.

ან უნდა იყვნენ კონკრეტულ კლასზე მიბმული და მაშინ უზრუნველყოფენ ამ კლასის სიის ჩატვირთვას ელემენტებად თავიანთ კონტრუქტორშივე.

ATableView კი არის generic მაგრამ სიას თუ რა ელემენტებისგან უნდა შედგებოდეს გადაეცემა გამომძახებელი კლასიდან (AfB-ში ეს არის EditorPanelController-ის setTableDataList მეთოდი)