Veikka Puhalainen

100884045

Computer Science, 1st year

18.4.2024

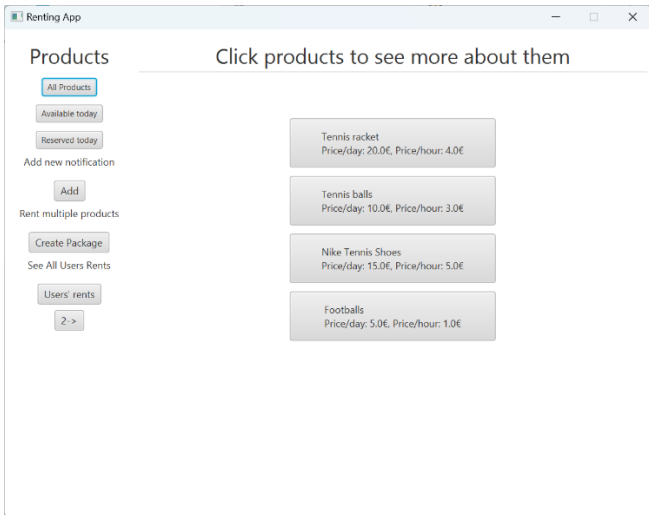
Renting App Documentation

I created a rental application with a medium level of complexity. The basic idea of the application is that users can simultaneously act as both renters and list their own items for rent. Users can add items they want to rent through a form, providing all the necessary details for each item individually. Users can browse all listings, check the reservation status, and delete listings if needed. Additionally, user can browse all users' items that they have rented.

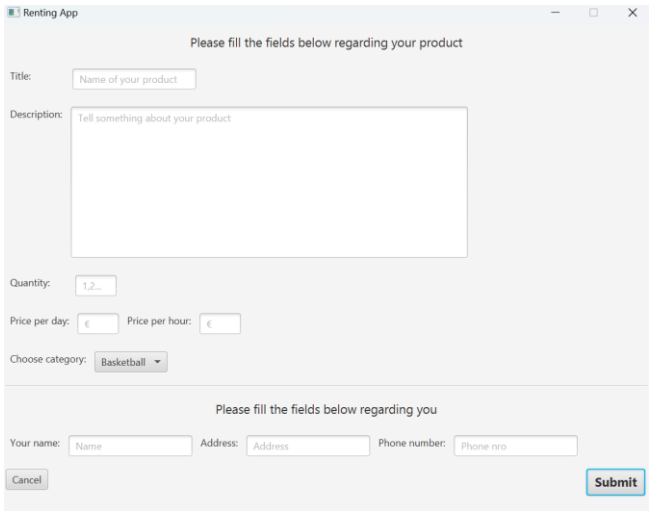
All the details entered in the form for each listing are visible. Users can comment on listings using the comment section. The rental status of the item is displayed, indicating whether it is available immediately, and if not, when it will be available, as well as upcoming rentals and their durations. To visualize the rental status, a calendar is displayed, showing the availability with clear color codes and some text.

Instruction manual

In homepage's left side there is buttons user can press. First three buttons filter the notifications that are shown in homepage's right side. Options are all, available today and reserved today.

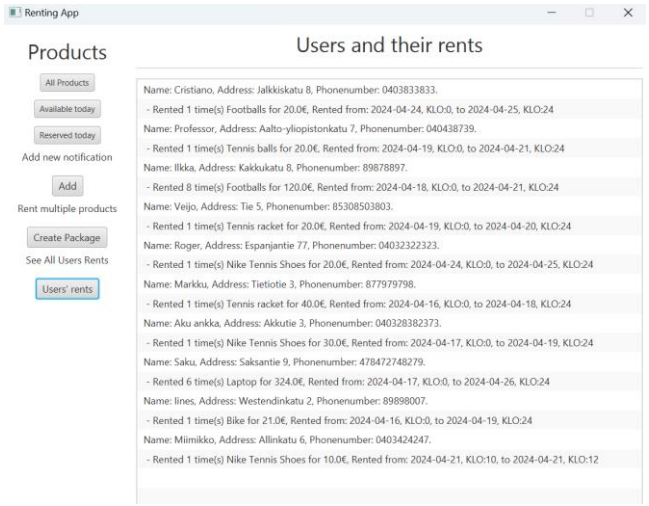


Below those buttons is add button which pressing, notification form will pop up. There user can write information of his product such as name, description, prices etc. When pressing submit button from right bottom corner, notification is added to homepage if all information was added correctly.



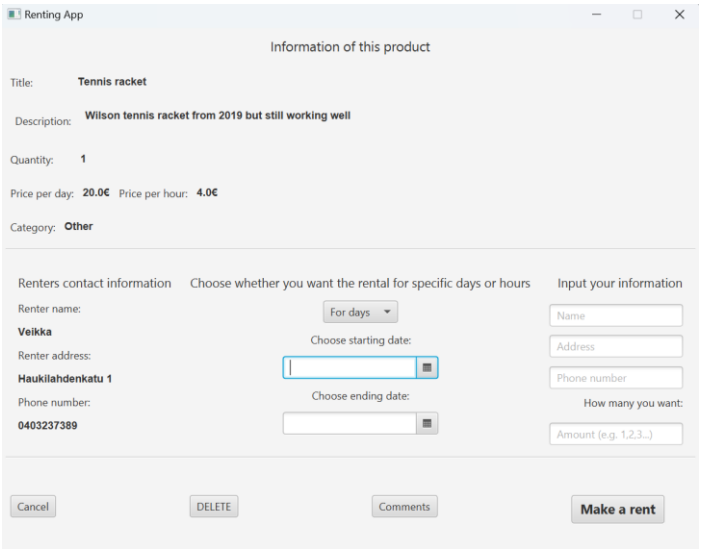
Next button is create package button which allows user to rent multiple products at a same time. This happens by pressing create package button and then pressing every product you want to rent. Pressing same products many times doesn't add them to package more than once. Now when pressing package done button, rent making page opens where user can rent all these products for same time.

Second last button is for seeing every user that has made rents in this app and all rents he has made. When pressing Users' rents a list opens where first you see a user's information and after that every product user has rented.



The last button is for changing pages. By pressing 2-> you can open page 2 etc. Overall, there are 4 pages which can hold max 6 notifications each. Pages are filled from 1 to 4.

In right side of homepage are all products that can be rented. By pressing a product, a new window opens where you can see details of this product, information of renter, calendar in which reserved days and times are shown and some more buttons.

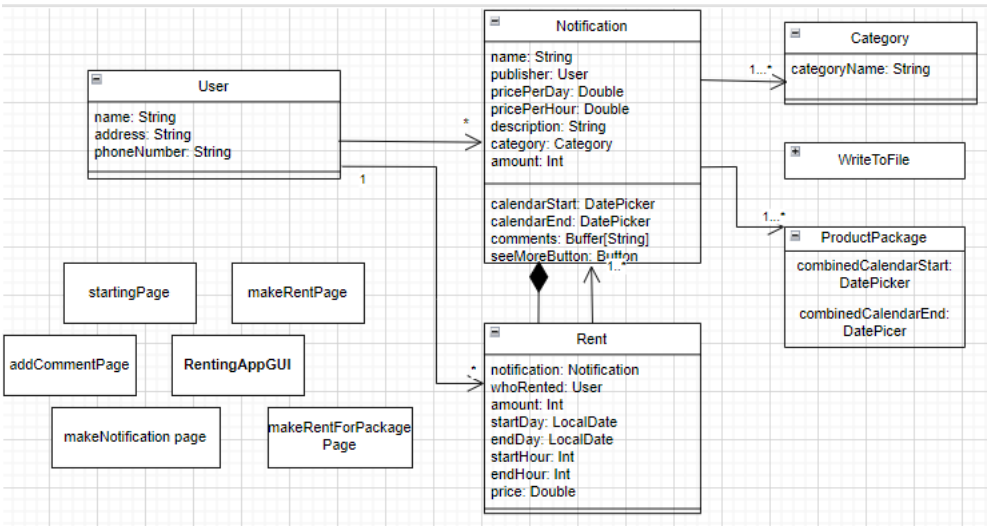


First let's look at how to rent a product. User can choose renting to either for days or hours. By renting for days user chooses starting and ending day of the rent. Starting and ending hours are automatically from 0 to 24 so product is rented for entire day every day. After filling information of himself, product can be rented for desired time, if possible, from make a rent button. If user chooses to rent for hours, he chooses day he wants to rent and starting and ending hours of that rent.

Comments of this product can be seen from comments button below. All comments are anomaly. New ones can be added by writing a comment to text field and pressing add comment button. Product can also be deleted by pressing delete button next to comments button in rent making page.

Structure

Project's final UML:



Classes' methods:

User:

def makeNotification: Creates a new notification from given information.

Notification:

def countDays: Returns a buffer of days between two given LocalDates.

def daysBetweenSandE: Returns days between this notification rents start and end day “middle days”.

def allReservedDays: Returns every reserved day of this notification.

def schedule: Returns buffer of tuples which has every this notifications rents’ start and end days and hours and amount of the rent.

def setCells: Adjusts every cell in this notification’s calendars so that reserved days are pink and they show text of how long this rent is for.

def button: Returns this notification’s button.

ProductPackage:

def countDays: Returns a buffer of days between two given LocalDates.

def daysBetweenSandE: Returns days between this packages all notifications’ rents start and end day “middle days”.

def allReservedDays: Returns every reserved day of this packages notifications.

def schedule: Returns buffer of tuples which has this packages notifications rents’ start and end days and hours and amount of the rent

def setCells: Adjusts every cell in this package’s calendars so that reserved days are pink and they show text of how long this rent is for. This takes count every packages notifications’ rents.

WriteToFile:

def writeNotifToFile: Writes given notification as json to “jsonFileNotif.txt” which has all notifications as a list.

def writeRentToFile: Writes given rent as json to “jsonFileRent.txt” which has all rents as a list.

def deleteNotification: Deletes given notification from file “jsonFileNotif.txt”

def deleteRent: Deletes given rent from file “jsonFileRent.txt”

def readRentsFromFile: Gives every rent of “jsonFileRent.txt” as a list by decoding the json file.

GUI’s methods:

def checkRentDays: Checks if given days and hours are correct so that rent can be executed for this desired period. This is run when rent is for days.

def checkRentHours: Checks if given day and hours are correct so that rent can be executed for this desired period. This is run when rent is for hours.

def countDays: Counts number of days between given start and end day.

def addComment: Checks if given comment is not empty and adds it to notification's comments.

def clearView2(): Clears every field from notification making page (view2).

def clearView3: Clears every field and calendar from rent making page (view3).

def clearView5: Clears every field and calendar from package rent making page (view5).

def countRentPrice: Counts rent price of given notifications for given time.

def createNewRent: Checks if all fields regarding making a rent are filled correctly and creates new rent.

def createNewRentForPackage: Checks if all fields regarding making a rent of package are filled correctly and creates new rent for all given notifications.

def createNewNotification: Checks if all fields in notification making page are filled correctly and creates a new notification by these information and adds this button to homepage.

def addToPackage: Adds given notification to temporally set of notifications for making a package.

def addPackageButtons: Changes homepage's product buttons so that their onAction now adds them to package.

def updateView3: Event listener for rent making page's choosebox where user chooses rent for days or for hours. This method changes the showing calendar and text fields.

def updateView5: Event listener for package rent making page's choosebox where user chooses rent for days or for hours. This method changes the showing calendar and text fields.

def makeRentPage: Fills rent making page with given notification's informations.

def makeRentPageForPackage: Fills rent making page with all given notifications' informations.

def makeCommentsPage: Shows comment page of given notification.

def deleteNotification: Deletes given notification from "jsonFileNotif.txt"

def updateStartPage: Updates start page how user wants. If value all is true every notification is shown, if available is true only products available today are shown, if both are false, reserved today products are shown. Divides notifications to 4 pages starting from page 1 and sets their onActions to show rent making page.

def deleteExpiredRents: This is run only when starting app. Reads every rent from "jsonFileRent.txt", and checks the ending days and hours of them. If rent has ended, it gets deleted from file.

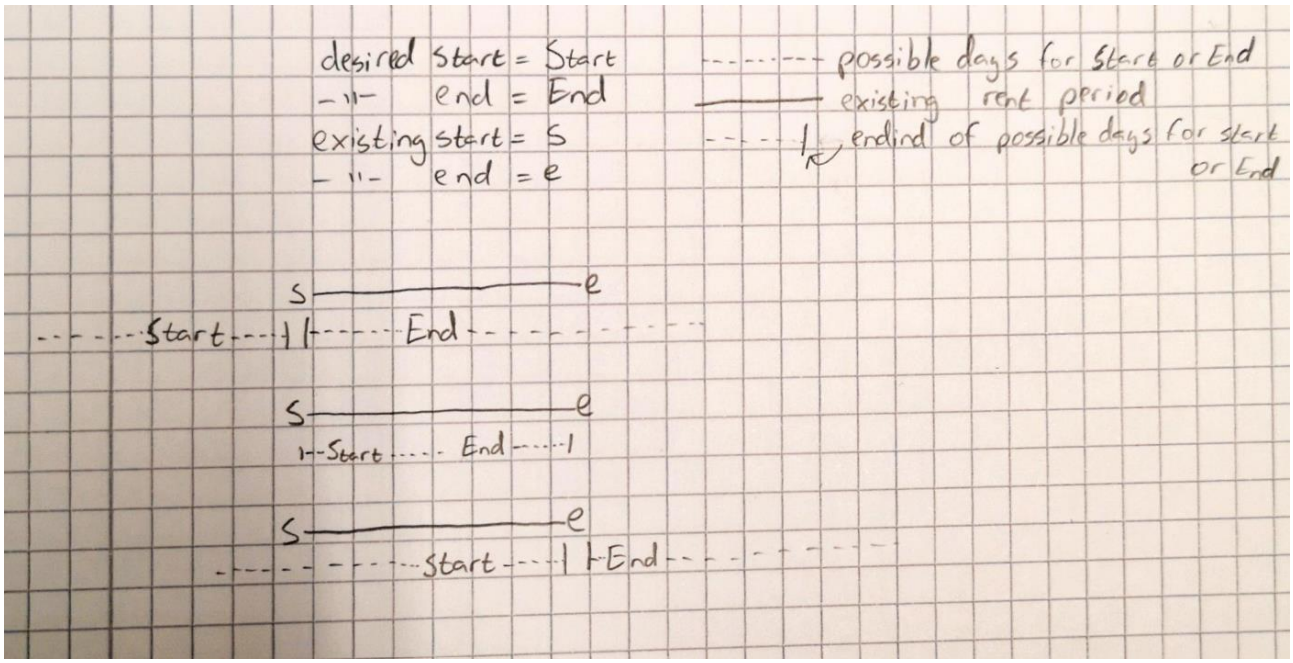
def seeUserData: Shows every user who has made rents and what he has rented in homepages right side in a listView.

Algorithms

One self-made algorithm is used when creating a new rent and checking if times are overlapping. Rent can be done for days or for hours and algorithm changes a bit so let's see first when renting for days.

In a short, rents can overlap in four ways: If desired time overlaps existing start day, if desired time overlaps existing end days, if desired time goes over both existing end day and start day or if desired time is inside existing start and end day. After getting all existing rents and their time periods we can start checking these.

First, we check if overlap happens by full days. We can compare desired days in three ways to see if overlaps happen in any of the four ways mentioned earlier.. Picture shows these 3 situations better:



Then we compare if start days or end days are the same day. If they are rent can't be done because start hour is 0 and end hour is 24 automatically when renting for days. We check if only start days are equal and if end days are equal. Also we check if existing start day equals desired end day or the other way.

When renting for hours algorithm changes a bit. If start days are equal but ending days not, rent can't be done because this existing rent is for days. If end days match but start doesn't that means existing rent has been made for days so ending hour is 24 and rent can't be done. If start and end days both match desired day, we check as shown in picture above the overlaps but using hours instead of days.

Data Structures

For making package of products I am using a mutable.Set because that handles itself situations when user tries to add same product more than once to package. I am also using mutable.Buffers for storing for example comments of a notification. Buffer is the most used data structure in this app.

Files

In this app every notification and rent are written to text files as Json objects. Json enables to write easily objects to text file and they are also easy to access and easy to read by user himself if he wants to. Here is example of notifications written to Json file:


```
{
  "name" : "Tennis racket",
  "publisher" : {
    "name" : "Veikka",
    "address" : "Haukilahdenkatu 1",
    "phoneNumber" : "0403237389"
  },
  "pricePerDay" : 20.0,
  "pricePerHour" : 4.0,
  "description" : "Wilson tennis racket from 2019 but still working well",
  "category" : {
    "categoryName" : "Other"
  },
  "amount" : 1
},
{
  "name" : "Tennis balls",
  "publisher" : {
    "name" : "Veijo Kauppinen",
    "address" : "Tietotie 4",
    "phoneNumber" : "03003489438"
  },
  "pricePerDay" : 10.0,
  "pricePerHour" : 3.0,
  "description" : "10 balls total brand new",
  "category" : {
    "categoryName" : "Other"
  },
  "amount" : 1
},
```


And here is example of rent:

```
[
  {
    "notification" : {
      "name" : "Tennis racket",
      "publisher" : {
        "name" : "Veikka",
        "address" : "Haukilahdenkatu 1",
        "phoneNumber" : "0403237389"
      },
      "pricePerDay" : 20.0,
      "pricePerHour" : 4.0,
      "description" : "Wilson tennis racket from 2019 but still working well",
      "category" : {
        "categoryName" : "Other"
      },
      "amount" : 1
    },
    "whoRented" : {
      "name" : "Markku",
      "address" : "Tietiotie 3",
      "phoneNumber" : "877979798"
    },
    "amount" : 1,
    "startDay" : "2024-04-16",
    "endDay" : "2024-04-18",
    "startHour" : 0,
    "endHour" : 24,
    "price" : 40.0
  },
```


Testing

My testing changed from the original plan. I tested only through the GUI and the console. GUI testing was intuitive and easy to start since I started making the project by making GUI. Also, I realized that lot of methods like showing a button of notification is just better to test with GUI than unit testing.

First, I tested notification creating simply in console by printing made notification to there. After that it was easy to test if notification button was shown in homepage when creating a button feature.

After that next was making a rent. For that I used same strategy as with creating notifications. First, I printed completed rent to console and then I wrote it to file and checked that everything matched and wrong inputs where handled. For testing if days or hours overlap when making a rent, I wrote examples down and tested them through GUI again, since I was showing alerts if rent was not possible to do it was easy and fast way to see if my algorithm worked. Gladly I tested it many times because there were a lot of issues at beginning.

Main thing that helped me with testing was to write wrong input handlers at beginning to every method. That way it was smoother path to continue with different scenarios for example when creating a new notification.

Lacks and issues

One main issue if thinking this app as a real-life app is when making a rent for product for example football that has amount of 3, the app lets user to rent that football only one rent at a time even

though there would be two more footballs left. So, app checks if user tries to rent too many products like 6 footballs but making two rents for 1 football isn't accepted.

I made the algorithm to check overlapping rents first and I thought that I would add this extra feature later. For my surprise, adding this feature was way harder than I thought. My approach was to get all rents that overlap desired rent and check the amount of those rents and then calculate all amounts together and see if there were any footballs left. When I started to execute this idea I realized that for example if the desired rent is for 2 footballs from Monday to Friday and there are two existing rents each for 1 football one from Monday to Tuesday and the other from Wednesday to Thursday. When calculating overlaps and checking if amounts are legal, I would see that there are two overlaps and each have 1 football: 2+1+1=4 so rent can't be done. But clearly, we can see that because the two already existing rents don't overlap each other, the rent should be able to do. This was just simple example, and it goes more complicated when the rent time goes bigger, and more and more overlapping rents come. I simply couldn't find way for checking every overlap and their overlap and so on so I went back to original and decided that rents can't overlap in any way.

Another lack is mainly visual. When watching product packages calendar, if some products' rents for the whole day, the datepicker doesn't show that product's name or quantity. It shows only text "one of this package's product is rented for the whole day". This isn't major issue since the days when some product of package is rented are always red in calendar so issues with that doesn't come but it's more blemish.

3 best things and 3 worst things

3 Best things:

1.The GUI looks overall good in my opinion. I was very motivated to put time to it look good. Maybe there exists some button or label that isn't right in the middle but overall, it looks clear.

2.The datepicker that shows reserved days looks in my opinion nice and I am happy that I managed to get the reserved days shown in same calendar you choose the dates. It required to do a one identical datepicker more for each notification, but I think that it isn't an issue.

3.The way you can make a rent for multiple products in same time is smooth and because its similar to make a rent for one product its also easy.

3 Worts things:

1.Like I mentioned in lacks and issues, probably the worst thing is that you can't make overlapping rents for product which quantity would allow it.

2.The way text is showed in datepicker changes a bit. If product is rented for whole day, it doesn't show the "${amount}x ${notification.name} is rented for whole day" but "${notification.name is rented for the whole day}".

3.The GUI has a lot of code, maybe too much. By that I mean that it would make the whole project's code clearer if there were more methods in classes rather than almost all inside the GUI.

When coding I tried to do this but then I had issues with dependencies, so it became easier to put everything inside GUI.

Deviations from the plan, realized process and schedule

The process changed a lot from original plan since my assistant suggested that starting to make GUI first would be the best option and it was. Because of that I didn't stress too much of following the original plan, so my realized schedule was something like this:

First, I started making the GUI just to show a simple homepage and notification making page and implemented classes like rent, category, user and notification. I hadn't planned the GUI completely so after first sprint meeting my assistant helped me with some ideas and explained how actually the app should work and I got better ideas of what and how I am going to do this project.

Then I started to make the rent making page which took more time than I expected. There were issues with showing correct notification's information in page, so I needed to make pretty big adjustments to the app's structure. Also, I made the app write every notification to the files as json.

After that I started making the actual rent stuff by adding datepicker and making rent of how user chooses the dates and inputs other stuff. Then there was last sprint meeting and we discussed more about what I needed to still add to the app so it would fill all criteria.

After that, I focused 100 precent to the project and grinded about 10 hours a day for almost a week to add missing features like writing rents to file, updating calendar depending on product's reservations, adding comments, deleting notifications and seeing every users' rented products and overall fixed and commented the code.

What I learned from this was that I need to focus more on planning because it saves a lot of time from me in the future. That requires putting more time to actually see if there is for example a good implemented library in scala for calendar rather than just imaging that I need to make a whole new class for calendar. Time I put overall to this project was maybe a little more than I planned.

Final evaluation

The renting app works well and I am happy of the features it has. There isn't many things that are missing, one thing that is missing is the overlapping rents for different quantities. What I could add to this in future is the overlapping rents, maybe implement different type of calendar to show the reserved days better and make an actual calendar instead of a datepicker and add more ways to see statistics of different products and users. I think those would be some cool and useful things to add.

For adding these new features I think that this project's structure is made pretty easy to add them. Also modifying the project is made easy because of the structure.

If I would start now from the scratch, I would put more time on planning the project and design from the start better the way I want to add required features. Now the way I worked was that I

added some feature first and after that I thought how I could add this another feature. That led to some issues and step backs.

References

I have used websites stackoverflow (https://stackoverflow.com/), scala documentation (https://docs.scala-lang.org/) and baeldung (https://www.baeldung.com/scala/) when making my project.

The core of the method setCells in Notification and ProductPackage classes was looked from stackoverflow under header "JavaFx DatePicker color single cell". There was example of how to change a datepicker's cell color and add text to it. What I did myself was that I changed this java code to scala so that datepicker could be added to vbox (scala datepicker didn't accept java callback and scala vbox didn't accept java datepicker etc.) I added my own logic (the way method handles rented days if they are start day, end day or middle day) and added own texts and color it shows. Also I modified this version to ProductPackage so that it does same thing but can handle multiple notifications once and modifies texts regarding the notification that is reserved.

https://stackoverflow.com/questions/42542312/javafx-datepicker-color-single-cell