Rental manager documentation

Basant Khattab, 100707973
Tietotekniikka
2022-
25.04.2023

General description

My project is an implementation of a rental system where items can be rented a user for a given period of time. The project consists of five main classes: Item, Renter,RentalRecord, Reservation and RentalManager, and a GUI. The RentalManager class acts as the central hub for managing items, renters, and rental records.

The Item class represents an item that can be rented, and it includes properties such as name, description, item type, price, hourly, daily, weekly, and monthly rates, and available count. It has a method for calculating the rental price based on the duration and the rates. The Renter class represents a user who can rent items and includes a list of rental records and comments.

The RentalRecord class represents a rental record and includes information such as the renter, the rented item, the count, the start and end time, and the price. The Reservation class is similar to the RentalRecord class but represents a reservation and includes information such as the reservation name, the rented items, their counts and the start and end time.

The RentalManager class includes lists of items, renters, rental records, reservations, and rented items. It has methods for adding and removing items, renters, and rental records. It also has methods for renting and returning items, calculating rental prices,adding reservations and saving and reading files.

The user interface is what the program's user interacts with. It can be used to add items, remove items, rent items, return items, add renters, remove renters, add reservations, get the list of items by type and many other things. The user interface also contains all the accounting methods and their buttons. This is due to me having trouble with the originally planned AccountingManager class.

The project has been implemented at an intermediate to difficult level, missing only one requirement per level.

User interface

To start using the user interface you have to run the GUI object. After running the program you are greeted with the main page of the program. The main page consists of the item, accounting and record views along with the load button which loads everything previously done. The first thing to do is press that button.

After loading the user can choose which view they want to use. The items view takes you a page where all the main commands of the program are located. Each command has its own button. After pressing a button, for example the rent item button, the program gives you instructions as to what to input. Each command has an enter button that fulfils its respective task. After pressing enter the program tells you if your request has gone through or not.

In the case of the rent item button, the program asks the user to input the renter's name, item name and count first. After it there are date pickers and text boxes where the user should choose the renting timeframe. If the input is in the correct form and the items are free the interface displays a success message and the item is rented. This can be checked from the "Check rented items" button.

There are also back buttons for every command and view that take you back either to the back to the view page or main page depending on where the button is located.

The item view page also has the "save" button that saves all the data so you can close the program without worry. It also has a "current situation" button that tells you the current situation based on the items rented and items available along with some other self explanatory buttons.

The second view is the accounting view which consists of the buttons incomebyitem, expensesbyitemtype, expensesbyitem. Where you need to follow the instructions to choose the dates and itemtype or item. After inputting them the user either gets 0.0€ as the prompt or the correct amount of money if everything is correct. The view also has a "Gained demand" button that displays a pie chart of all items that have been rented in proportion to how often they have been rented.

The last view is the records view that only has one button "rental records" that displays all rental records when clicked.

Program structure

The program is split into several classes to model the different aspects of the rental management system. The final realised class structure consists of five classes:

Renter: Represents a person who rents items. It has fields for the renter's name, rental records, and comments. The methods of the Renter class include addRentalRecord, removeRentalRecord, and addComment.

Item: Represents a rental item. It has fields for the item's name, description, itemType, price, hourlyPrice, dailyPrice, weeklyPrice, monthlyPrice, and availableCount. The methods of the Item class include addComment and getPrice.

RentalRecord: Represents a record of a rental. It has fields for the renter, item, count, rentStart, rentEnd, and price.

Reservation: Represents a reservation. It has fields for the reservation name, the rented items, their counts and the start and end time

RentalManager: This class represents the core of the rental management system. It has fields for the items, renters, rental records, reservations, and rented items. The methods of the RentalManager class include addItem, removeItem, rentItem, returnItem, addRenter, removeRenter, addReservation, and getItemsByType along with the saving and loading methods for all files.

The RentalManager class models the main part of the problem domain of the program, managing the rental of items, keeping track of rental records, renters, reservations, and rented items. The Renter and Item classes are used to model the rental records and rental items. The RentalRecord class is used to model the individual rental records.

The relationships between classes are as follows: The GUI class has a RentalManager object, which is used to implement all the interactive components of the GUI. RentalManager has a ListBuffer of items, renters, rental records, reservations, and rented items. Each Renter has a ListBuffer of rental records, and each Item has a ListBuffer of comments. The RentalRecord has a reference to both a Renter and an Item.

The GUI is a crucial part of the program, and it is implemented as a single object that contains all interactive components of the software. This GUI is responsible for managing the four different scenes in the program, including the main view, item view, accounting view, and record view. The GUI also includes all the accounting methods and their buttons. Originally, there was a plan to use an AccountingManager class to handle this. However, due to some technical difficulties, I decided to integrate the accounting methods into the GUI object itself.

The essential methods in the classes include:

Renter:

- addRentalRecord: adds a new rental record to the renter's rental records.
- removeRentalRecord: removes a rental record from the renter's rental records.
- addComment: adds a new comment to the renter's comments.

Item:

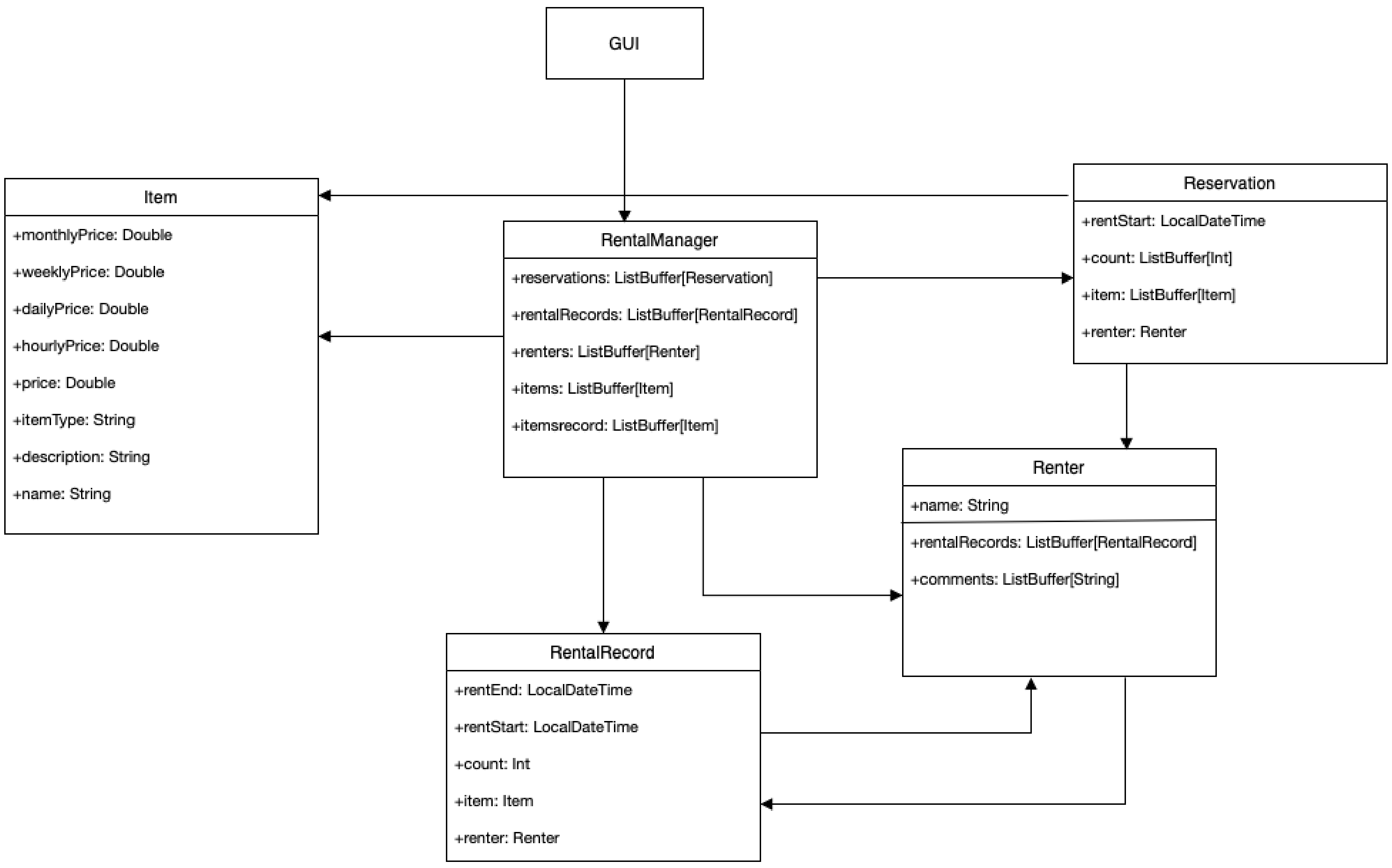
- addComment: adds a new comment to the item's comments.
- getPrice: calculates the price for renting the item for a given duration.

RentalManager:

- addItem: adds a new item to the RentalManager's list of items.
- removeItem: removes an item from the RentalManager's list of items.
- rentItem: rents an item to a renter for a specified duration and updates the RentalManager's lists accordingly.
- returnItem: updates the RentalManager's lists and returns a rental record.
- addRenter: adds a new renter to the RentalManager's list of renters.
- removeRenter: removes a renter from the RentalManager's list of renters.
- addReservation: adds a reservation to the RentalManager's list of reservations.
- getItemsByType: returns a list of items of a specific type.
- Save and Load methods: These methods are responsible for saving and loading the program's data, which includes rental records, items, item records, and rented items,

to and from separate files. These methods allow the program to be closed and reopened without losing any saved information.

Rental manager system UML



Algorithms

The Rental Manager program includes several algorithms to perform tasks such as renting, returning items, adding and removing renters, and adding and removing items, among others. The algorithms are designed to make sure the rental process is smooth and the rented items are correctly accounted for.

The algorithm for calculating the rental price is based on the duration of the rental and the price of the item. It returns the price of renting the item for the given duration, based on a formula that calculates the total cost of renting the item, including hourly, daily, weekly, and monthly rates.

The rental algorithm takes several steps to rent an item. It checks if the item is available, calculates the cost based on the rental period, updates the available count of the item, creates a rental record, and adds it to the renter's rental records. If the item is not available, the algorithm throws an exception.

The return item algorithm updates the available count of the item, removes the rental record from the renter's rental records, and adds the rental record to the rental records.

The add and remove renter algorithms add and remove renters from the renter list, respectively.

The add and remove item algorithms add and remove items from the item list, respectively. The remove item algorithm removes the item based on the name and the available count of the item.

The add reservation algorithm checks if all items in the reservation are available for the rental period and adds the reservation to the reservations list.

If an item is not available for the rental period or there are not enough items to complete the reservation, the algorithm throws an exception.

The gainedDemand algorithm in the GUI counts the number of times each item has been rented and displays the data in a pie chart. It starts by getting a list of all rental records from rentalmanager.rentalRecords. It then counts the number of times each item was rented by iterating through the rental records, and storing the count in a map (itemCounts) with the item name as the key. Finally, it adds the data from itemCounts to the chartData observableArrayList of PieChart.Data, creates a new PieChart object with the data, and displays the chart in a VBox.

The incomebyitemtype algorithm calculates the total income for a specific item type during a specified time period. The user inputs the item type, starting date, and ending date, and the algorithm retrieves the corresponding rental records from the database. It then calculates the total cost of the rentals and displays the result to the user. The algorithms for incomebyitem, expensesbyitemtype, expensesbyitem all work in the same way.

The program's algorithms are designed to ensure that the rental process is smooth and the rented items are correctly accounted for. Alternative solutions to the algorithms could be using a database to keep track of the rented items and renters, or using an external API to calculate the rental price. However, I believe that the Rental Manager program's design using lists and local calculation is an efficient and effective solution for managing rental items.

Data structures

The program uses several mutable collections from the Scala to store and process the data. Specifically, it uses ListBuffer to store lists of items, renters, rental records, reservations, and rented items. ListBuffer was chosen because it provides efficient implementation of the += and -= operators to add and remove elements, and the list operations provided by the Seq trait.

The Renter class contains a list of RentalRecord and a list of comments, both implemented as ListBuffer. The Item class has a list of comments, which is also implemented as ListBuffer.

The RentalManager class has a list of itemsrecords, which contains all the items added to the program. It also has lists of items, renters, rentalRecords, and reservations, implemented as ListBuffer. The rented items are stored in the renteditems list. The itemtypes list contains all the unique item types available.

ListBuffer was chosen as the primary data structure because it provides efficient implementation of adding and removing elements to the end of the list in constant time, and the random access of elements in linear time. ListBuffer is also a mutable data structure, which means it can be updated in-place, making it more suitable for the nature of the program.

Other data structures that could have been used include ArrayBuffer, HashMap, and HashSet. ArrayBuffer provides efficient implementation of random access, but it is less efficient than ListBuffer when adding and removing elements to the end of the list. HashMap and HashSet are good for implementations of lookup and insertion, but they do not preserve the order of elements, making them less suitable for some operations in the program.

All the data structures used in the program are mutable. I did consider immutability when designing the program, I didn't see it necessary, as the program does not require data structures that are immutable.

Files and Internet access

The Rental Manager program deals with text files in which the data is presented in comma-separated values (CSV) format.

The data files are saved in the same directory as the source code file, and they are named as follows:

items.csv: Contains information about the items available for rent, such as their name, description, price, type, and availability count.

item_record.csv: Contains information about all items that have been added to the program.

renters.csv: Contains information about the renters, such as their name and rental records.

rental_records.csv: Contains information about all rental records, such as the renter's name, the rented item's name, the rental start and end dates, the rental count, and the rental price.

The program loads the data from these files when the load button in the GUI is pressed and saves any changes made to the data back to the files when the program is saved.

Testing

The program was tested mainly using the GUI with many different user stories. The system was done by using the GUI to test the functionality of the program as a whole instead of testing the classes. While writing the code for the GUI I tested all RentalManager buttons using printing, for example in the case of the rentItem method, I tested if the rented items ListBuffer was updated correctly by printing its contents. I also tested each method by seeing how the method deals with incorrect input.

Any errors I noticed were and I continued the testing again until no issues were seen. The file management was tested using the save and load buttons in the GUI and by printing each ListBuffer and comparing the contents to before.

To be completely transparent the testing process was very rushed and not done as well as I had originally intended. The main issues being error situations, That I didn’t account for as well as I should have.

The accounting view in the GUI does not pass the tests in the way I intended and that is due to problems with the algorithms. The methods incomebyitemtype, incomebyitem, expensesbyitemtype, expensesbyitem all don’t work as planned. They calculate the total sum of any RentalRecords found within the given timeframe even though they might last longer. Another issue is the adding and removing of reservations. This hasn’t been tested well at all, as I realised one day before the deadline that I had misunderstood the requirements. At first you could only reserve one item at a time but now, to fulfil the requirements, more than one item can be reserved at a time.

The testing process was very similar to the plan, just very rushed towards the end and a lot less thorough.

Known bugs and missing features

As stated in the Testing paragraph the methods incomebyitemtype, incomebyitem, expensesbyitemtype, expensesbyitem all don’t work as planned. They calculate the total sum of any rental records found within the given timeframe even though the record lasts longer than the end date of the timeframe.

Another issue is the adding and removing of reservations. This might have bugs as it wasn’t properly tested. Also, the reservations are not saved into files due to time constraints so they cause issues when loading the program since some items “disappear”.

My original plan was to implement every requirement for the entire project as to get the highest possible grade but realised early on in the implementation that this would prove very difficult. I have managed to get creative and fulfil almost every requirement except for adding pictures to items and implementing a calendar from which you can see the situation in relation to an item / premises.

Another issue is the lack of a scrolling bar that I just could not get to work. If for example, there are too many rentalRecords it is not possible to view them all in the records view.

If I could continue the project I would use most of the time to test the project more thoroughly which is the part I lacked the most. I would also add the option of adding images of the goods to the program. The item view in the GUI would have a picture button that would show all the pictures for the items. The calendar would also be implemented using the same idea. I would also try harder to have the scroll bar work and implement the methods needed to also save reservations.

3 best sides and 3 weaknesses

The three best sides of my project are:

1. The GUI is easy to use and intuitive. I’m also just very excited that I figured out how to create a GUI
2. The accounting view in the user interface includes numbers as well as a visual view in the form of a pie chart. This was also super fun to implement!
3. That the program can be saved and loaded without losing most of the data

The three weaknesses are:

1. A few things such as comments and reservations don’t get saved in the program. This is due to time constraints.
2. A lot of things have to be manually typed into the program instead of being able to just select them, for example when adding comments you have to type the item's name instead of just being able to see all the items you have automatically.
3. The rent item button doesn’t let you rent an item if you have spaces between the commas.

Deviations from the plan, realised process and schedule

The project was implemented back-end first. Due to me spending lots of time planning, starting with the back-end logic was the easiest solution. In the first two weeks I had trouble with the version control and could not push anything onto version. After solving this issue I started on implementing the RentalManager class. I had already started running behind schedule at this point.

The next two weeks I finished the back-end and started working on the GUI, which brought along a lot of problems. At this stage of the project I lost motivation and only worked during the weekends due to feeling stuck with the user interface.

The two weeks after that I started to get a hang of scalaFX and worked on completing the GUI slowly. I missed a weekend of working on the project due to school related events. At this point in the project I realised that I would have to get creative and cut corners in order to finish the project in time and I think I did decently well, seeing as I also had other difficult courses.

The last few weeks were spent implementing the files and making sure the project fulfilled the requirements. I also tested the program to make sure there were no huge underlying issues that I didn't notice while implementing. The last few days were spent fixing them.

The biggest difference from the plan was the schedule. I did everything in the planned order but I was definitely behind schedule most of the time. Another difference is the accounting manager class which I created and implemented but decided not to use during the last moments of programming the project.

During the process I learned a lot of useful lessons. The biggest one being the importance of planning when it comes to designing software. I don't believe I would've been able to complete the project , had I not planned well in the beginning. I also learned that time management is extremely important as it is very easy to take it easy in the beginning and end up being very rushed towards the end, which is why it's important to set smaller deadlines. I learned this the hard way. Another thing is trusting in your skills as a programmer. At first I didn't believe I would get this done since it was all very new to me, but I had to trust myself so I could finish on time.

Final evaluation

I believe the overall quality of the program is decent. The program meets most of the requirements and the code is well-structured and easy to read. The program also demonstrates good use of object-oriented programming principles, and I believe the solution methods and data structures used are appropriate.

I do however believe that there are many areas where the program could be improved. For example, some parts of the program could benefit from more complicated and detailed error handling, and alot functionalities could be made more user-friendly (highlighted in other paragraphs). Also, although the program is designed to handle a range of inputs, there may be certain cases where it could produce unexpected results.

Future improvements could be made to the program's error handling and user interface. Also more testing should be conducted to identify and resolve any potential edge cases.

In terms of the program's structure, I think it is suitable for making changes or extensions. The program is organized and modular, which should make it relatively easy to modify or add new features.

If I were to start the project again from the beginning, there are many things that I would do differently. I would conduct more thorough testing and debugging throughout the development process to identify and resolve issues earlier on. I would also explore alternative solutions and approaches to ensure that the final program is as optimised and efficient as possible.

Overall I am very happy and satisfied with this project. A lot of it was brand new to me and I'm excited to have been able to create a program from start to finish completely alone. As a new programmer this was a huge step to feeling more confident in my choices of study field and to get a taste of what software developing is like.

List of requirements and justification implementation

Easy:

- The program keeps track of rented goods or premises.

  Self explanatory

- There can be many different and different types of goods, e.g. camping equipment, sports equipment, more special kitchen items, clothes.

  Self explanatory

- There may be one or more copies of each.

  Self explanatory

- In the program, you can view the information of an individual type of goods or premises separately and add your own comments to them.

  Check items button in GUI and check/add comments in item view.

- Renting can be done for an arbitrary period, but in such a way that the daily, weekly or monthly pricing differs from each other.

  In RentalManager class

- A record is kept of the renters (name, contact information, rented items), which can be viewed separately and comments can be attached to them.

  RentalRecord class and add comments to renter in in comment section of the item view.

- Each rental includes information on what was rented, how many items, to whom and for what time, and the cost charged.

  RentalRecord class

- When the item(s) are returned or the premises is freed, this is recorded in the program.

  ReturnItem and RentItem method algorithms

- You can ask the program what the current situation is (how many items/premises are available and how many are rented, and when something will become available if everything is currently rented).

  Current situation button in GUI

- Information about accounting and transactions is stored in a file and can be read from there again.

  Save and Load methods in RentalManager

Medium difficulty:

- Graphical user interface

  Self explanatory

- All transactions are entered on a form where the status of each item (item/premises) is visible (how many items are available, when will the item be returned next time if it is not available now, etc.).?

Current situation button

- You can add new goods and types of goods to the program and delete them.

Add and remove item buttons

- Renting can be done not only for a day but also for an hourly price.

Self explanatory

- You can make advance reservations for things that may depend on each other and availability.

Reservation class and GUI

- The reservation may concern a number of things (e.g. 2 tents, 1 set of cooking utensils, 2 backpacks, 3 sleeping pads), in which case they must be freely available at the desired time.

Reservation class and GUI

- The program has a calendar from which you can see the situation in relation to an item / premises.

Not implemented

Demanding:

- Images of the goods or facilities are also visible.

Not implemented

- Damage or loss of goods can be recorded in the program and a summary can be obtained.

Comments. Since the items can be deleted without side effects and the record of the item is still saved just adding comments and then later deleting the item is sufficient.

- The program can produce views that show the gained demand and income of each good, the goods type together or premises within the desired period.

Gained demand button in GUI

- The program calculates the expenses and income for the desired time interval and makes a summary of them by product or by type of goods (respectively by premises, if they are available). The costs include, for example, the purchase of products, storage, repair, etc., depending on the situation.

Accounting view in GUI

References

I used a lot of stackoverflow and old github projects to get a reference. For the GUI I used the recommended video playlist in the course material. I also used the documentation for scalaFX to figure out commands that I couldn't find in the youtube playlist.