Person A – 680033128, Person B – 69

14/02/20

14:45-17:15

Met at the café ground floor of Harrison to discuss timings for future meetings as well as begin evaluating the required functionality of the program as well as consideration for the best ways to implement said functionality. Created a shared work file so we can be viewing the log and code on our computers whenever we meet up and easily make changes to the code whilst writing our report. Planned what kinds of data structures would be used for different parts of the program.

15/02/20

12:30-14:15

Library group study rooms. Today we began by evaluating the exception classes as well as the basic classes sent with the project briefing. We considered storing data within a dictionary type structure where the key is the 8-character hex identifier is the key and the value contain a list of information pertaining to different configurations of beanbags. For stock indexing to work with this data structure we would need to include a field for quantity such that any beanbags which are interchangeable with identical details are counted correctly as these could have identical hex identifiers or different identifiers if the differed in colour. We needed to assess the ObjectArrayList to see how best to implement the desired data structure without the need for additional modules.

16/02/20

13:30-14:40

Devonshire house

Drafted an example data structure listing all the information required to for an individual bean bag. This was done using the examples from the Class, Strings, Array Workshop. Person A suggested testing manipulation of data by starting off with a simple function to edit the quantity of an item using its Hex identifier. Person B attempted to implement this class and create a driver class to run all the methods contained withing the package. The Week 5 Quiz included an example of reading from text files. We considered later implementing this feature in our program so that the state of the stock manager is saved upon closing, but first needed to ensure all our functionality worked.

17/02/20

9:35-10:25

Met in group study room bottom floor library. Used draw.io to create a UML classes diagram. Considered the different data structures that could be implemented. Settled on an ObjectArrayList which operates with a dictionary structure using keys and value. Instances of an object can have identical names as they are stored in different memory locations as such it would be possible to count the quantity of beanbags in stock by counting the number of objects created. When an item with an identical hex ID is sold you can simply delete one of the instances of the matching objects.

18/02/20

10:00-14:00

In between our only lecture of the day we spent most of the time at Peter Chalk centre completing our final draft of the UML diagram and setup a GitHub repository which would allow us to keep track of code revisions and later add the package upon completion as one our project on LinkedIn. Person B wanted to begin the project by implementing a simple class responsible for sending data to the ObjectArrayList as mention previously the object names would be the HexID of the beanbags. Contained within each object would be all the information relating to a beanbag. We started with a single beanbag and wanted to attempt to change the data held by the object. The simplest test case we devised would be to change. Person A suggested a more detailed revision of the availability of both partners, so we decided to add our availability to each other’s calendars. We also took this opportunity to reconfigure the directories to match those suggested in the “Jar File Creation Walkthrough” which was recently released. Found a nice website call Draw.io which allowed for seamless completion of our UML file which had previously been drawn out on paper. This was based on the diagram given in the Class, String, Array workshop.

20/02/20

12:40-14:20

Took a break yesterday from meetings. Today we started to break down the briefing that we had been given to shortlist the key information provided and bullet point the key features that we needed to implement in our system. The package is required to:

* keep track of several attributes:
  + an 8-digit hex id (String)
  + the manufacturer (String),
  + the bean bag name (String),
  + the priceInPence (**int**),
  + the year of manufacture (**short**),
  + the month of manufacture (**byte**),
  + the number of a type of bean bag (**int**) both remaining and later sold,
  + an optional free text component containing any additional information (String).
* Ability to reserve items currently in stock
  + reserved items must only be sold to customer with matching reservation number.
  + customers can cancel reservations and add item back to general item stock.
  + we would need to keep track of an attribute relating to reservationNumber (**int**)
* Ability to change price of items
  + must ensure that and items reserved are sold at the lower price.

Therefore, it seemed appropriate to split the problem up with separate classes for dealing with the reservation system, tracking sales totals and managing the store’s stock.

Person A said that from the information provide it looks like it would be possible to create multiple different types objects containing varying attributes for example one type of objects for stock management and separate type of object for reservations whilst still being able to store all these object within the same ObjectArrayList class provided. Person B liked the sound of this idea but said it would be wise to start by declaring the required attributes as null variable within their appropriate classes. This would allow for compartmentalisation of different kinds of functions within the overall package where similar function relating to the same feature would be grouped together.

23/02/20

Person B released we would need two constructor methods one which included a reference to the optional description variable containing the free text and a second identically named method without this variable. When running the package, the class selects the appropriate method based on the number of variables included when calling the method. Person A remembered that no variable for quantity need be included in the constructor as when multiple identically named beanbags are in stock, they would each be referenced as separate objects in memory within the java virtual machine.

23/02/20

Labelled methods getter and setter declared private variables alongside the public methods to call said variables. Created a new class called Stock responsible for managing any object containing information relating to the details of beanbags in stock. As discussed above we intend to segregate functionality within the program such that we are implementing several smaller classes rather than one large class containing all functionality. We had a discussion at length about the benefits of separating different parts of the package some of the benefits include upgradability of the system, readability, error checking/bug fixes, reliability. This may however come at the cost of a less efficient system in terms of the amount of storage required to store a greater number of objects in the ObjectArray. This is as a result of splitting up reservations and beanbag details into two separate objects. The table below details the two different methods that could be implemented in our system. The latter is the format that we have decided to go with however as we continue to add functionality to our program, we will continually evaluate this data structure is the most appropriate.

Upon later review of our code person B realised that within the reservations object we would need to store the price of the item at the time it was reserved. This would allow us to later compare this price with the current price of any identical items in general stock and change the customer the lower of the two prices. As such Person A change the code to include a field for the original price. We also included an additional getter method in order to retrieve this new field from the object.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8-digit identifier  String | Manufacturer  String | Name String | Manufacturer String | Price double | Year of manufacture  int | Month of manufacture int | Optional description String |
| A1B2C3D4 | Argos | Recliner | Argos | 29.99 | 2019 | 11 | Teal Blue |

The above would be a stock object

|  |  |  |  |
| --- | --- | --- | --- |
| 8-digit identifier  String | Reserved  Boolean | Reservation Reference String | Customer Name  String |
| A1B2C3D4 | True | Order-1234 | Jane Doe |

This object would be for reservations

The first possibility for data handling would require two separate objects for any reservation but would lead to better readability within the program and all the other benefits listed above. The second method below incorporates all data into a single object so takes up less storage but any changes made to the code would be more difficult to make due to reduced readability hence bug fixes and error checking would be far more difficult hindering development of the package classes.

The below an object combining both stock and reservations

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8-digit identifier  String | Manufacturer  String | Name  String | Price double | Year of manufacture int | Month of manufacture int | Optional description String | Reserved Boolean | Reservation Reference String | Customer Name String |
| A1B2C3D4 | Argos | Recliner | 29.99 | 2019 | 11 | Teal Blue | True | Order-1234 | Jane Doe |

Following on from this Person B suggested that for the reservations class we could follow the same approach as the stock class such that any reservation objects are named using the reservation number the only difference being that for reservation we would never have any identical objects.

Person B found and error with Person A’s code even when calling the reservation object by its reservation reference we would still need access to the hex identifier to change the availability of reserved items. We updated our Reserved class to include a field for the identifier within the object.

Currently the reserved class only includes a setter method for changing the Boolean value referring to whether the item is reserved. If this is set to false, then the reservation object is removed from the ObjectArrayList and available for purchase withing the general store. Later we may choose to include setter methods for changing the beanbag that a customer has decided to reserve or the customer name relating to that reservation however to keep the initial program simple currently these field are only accessible through getter method. Therefore, in order to change any details relating to a reservation the store would first need to cancel the original reservation before creating a new reservation entirely. This currently isn’t a particularly major issue but is something we could revisit at a later stage depending on the functionality required. In some instanced this implementation may be better than the alternative as it reduced the possibility of accidental deletion or editing of details relating to items that a customer has already decided to reserve.

29/02/20

With the separate objects for stock and reservations Person A remembered that you would need a way to store the details relating to reserved items. This could be done in one of two ways: the first being to include an optional reserved field in the stock object which would require yet another constructor method with space for this field. This particular method would only be invoked whenever any reservations are made. This reserved variable would only need to be set when the Boolean value is true. By including this addition method and relying on the same stock objects as before we could use the same setter methods from the Stock class to make changes to reservations instead of having to include duplicate methods within the Reservations class. The alternative is to copy over all the details relating to the item for use in the reserved object. This would reduce the need for an additional constructor method and would not affect storage required, as you remove the stock object from the ObjectArrayList and replace it with a reserved object with identical details.

Person B noted that we had not even begun to incorporate any functionality into our program and already we had the possibly of using four different implementations for our data structure. As such we decided to keep things as simple as possible for the time being whilst still allowing ourselves to make any necessary changes to our design in the future, should we decide that our chosen structure was no longer suitable. Our UML diagram was of significant use to us in allowing us to plan ahead, making sure we would not later have to compromise on functionality, whilst ensuring we did not encounter the headache of needing to rewrite large portions of code at later stages of development.

20/02/20

12:40-14:20

Amory Cafe

In our OOP lecture today, we learnt about inheritance and the keywords 'super' and 'extends' to refer to parent and child classes. Person A considered that we could have the Reservations class inherit the constructor method from Stock classes. To do this this subclass needs to extend the superclass as such we could include additional parameters for 'reserved (Boolean)’, 'reference (String)', 'customer (String)'. The access modifier for an overriding method can allow for more but not less access than the overridden method so this approach should be suitable for our application.

24/02/20

9:40-10:20 & 12:40-14:20

Devonshire house and Peter Chalk

Started coding for our project Person A took on the role of the driver whilst Person B was the observer in this instance. We began by creating a new Store class and copying over all the constructor methods listed in the BeanBagStore interface class. From here Person B suggested that we work by first creating a template of all the getter and setter methods described in the briefing and highlighting any key areas where we could potentially reuse parts of the code by overriding certain methods. After we had completed the outline of our project and what is was going to look like, we could then begin filling in the simpler methods responsible for creating beanbag objects and coded getters and setters for accessing and changing the quantity of a beanbag stored in the object.

Person B had realised that the data types of each parameter that we needed to state were given in the BeanBagStore interface as such Person A changed some of the variable types such as month from int to byte and year from int to short. We also took this opportunity to create a separate class containing our main method used to control the rest of the package.

26/02/20

12:40-14:15

Library Group Study Room

This session Person B was the driver and Person A the observer. Person A realised that later on in development whenever we got round to creating the getter method for year of manufacturer if would be possible to use a switch statement to convert the byte data type to a String types containing the full name of the month in which the beanbag was manufactured. This got us thinking about other useful java features that we had learn in lectures that could be use in our program. Person A determined that we would need some way to enforce the 8-digit hexadecimal naming scheme described in the brief such that only String values conforming to the standard were allowed. Person B suggest it may be possible to use the regex search pattern for input validation when giving each beanbag a unique identifier, we would revisit these discussions in later stages of development.

27/02/20

12:40-14:00

Library Group Study Room

Started coding for our project

Further to the discussion above regarding the reuse of code Person A noted that it may be possible to override the setter method for addBeanBags for reuse when selling beanbags. When we get to this part of the project, we will evaluate whether this is the case or if there is a better solution. Person B realised that we had missed declaration of two parameters, namely reservationNumber and filename used to identify reserved sales and store the contents of BeanBagStore respectively.

29/02/20

12:50 -