CS4227 Software Design & Architecture

Computer Manufacturer and Retailer

Michael Lucas 13112732

Conor Duggan 13127004

Cian Prendergast 12127191

Daniel O’Connor 13141252

Darren Stack 13129783

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CS4227: Software Design & Architecture Guidance on Marking Scheme for Team-Based Project: Semester 1, 2016-2017 | | | | | | |
| Name 1:  Name 2: Name 3: Name 4:  Name 5: | | | ID1:  ID2: ID3: ID4:  ID5: | | | |
|  | Item | Detailed Description | | Marks  Allocated | | Marks  Awarded |
| Sub- | Total |
| 1-2 | Presentation | General Presentation  Adherence to guidelines i.e front cover sheet, blank marking scheme, table of contents | |  | 2 |  |
| 3 | Requirements | Narrative & Use Case Diagram  Light weight SAMPLE Use Case  Descriptions  Discussion on NFRs with a focus on architectural use cases (quality attributes) | | 1  1  2 | 4 |  |
| 4 | Discussion on  Architectural and  Design Patterns | The Interceptor architectural pattern.  Discussion of 6 patterns  One must be selected through self guided research. | | 2  2  1 | 5 |  |
| 5 | System  Architecture |  | |  | 5 |  |
| 6 | Structural and  Behavioural  Diagram | Class and interaction diagram with  Correct application of design patterns  All patterns integrated | | 2  1  1 | 4 |  |
| 7 | Code | Matches/Supports/Realises diagrams  Interceptor pattern correctly implemented.  Design Patterns correctly implemented  Exposes intent, supporting documentation, naming conventions clearly identify design patterns used  At least 4 packages, one developed by each team member | | 2  3  6  1  P/F | 12 |  |
| 8 | Added Value | Two examples, 3 marks each. | |  | 6 |  |
| 9 | Testing | Design of test cases  Automated  Analysis of results | | 1  1  1 | 3 |  |
| 10 | Issues | Satisfactorily documented.  No marks awarded. | |  | N/A |  |
| 11 | Evaluation /  Critique | Is it the case that the patterns selected  supported relevant architectural use cases? If not, why not? Any alternatives? | |  | 3 |  |
| 12 | References |  | |  | 1 |  |
|  | Interview  Week 11 or 12 | Competent code inspection  Working demo | |  | (P/F) |  |
|  | SUB-TOTAL (A) | | | | 45 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| PENALTIES | | | |
|  | Description | Member 1 | Member 2 |
| 1 | Late Submission |  |  |
| 2 | Failure to contribute to coding effort |  |  |
| 3 | Failure to contribute to writing of report |  |  |
| 4 | Failure to report problems with team dynamics |  |  |
| 5 | Failure to contribute to demo week 13 |  |  |
|  | Sub-total (B) |  |  |

Table of Contents Page

Narrative ……………………………………………………………………………………………. 5

UseCases …………………………………………………………………………………………… 5

* Use Case Descriptions ………………………………………………………………. 6

Non-Functional Requirements (NRFs) …………………………………………………. 7

Design Patterns

* Visitor ………………………………………………………………………………………. 8
* Decorator………………………………………………………………………………….. 9
* Factory …………………………………………………………………………………….. 9
* Composite ……………………………………………………………………………….. 10
* Observer ………………………………………………………………………………….. 10
* Memento …………………………………………………………………………………. 11
* Command…………………………………………………………………………………. 11
* Interceptor Architectural Pattern ……………………………………………….. 12

System Architecture …………………………………………………………………………… 14

* UML Workbench
* Implementation Language

UML Class Diagram ……………………………………………………………………………. 16

UML Sequence Diagram ……………………………………………………………………… 17

Code Contribution ………………………………………………………………………………. 18

Added Values …………………………………………………………………………………….. 20

* Tactics for applying quality attributes
* GitHub
* Applying metrics to the code

Testing ………………………………………………………………………………………………. 22

Issues ……………………………………………………………………………………………….. 23

Critique ……………………………………………………………………………………………… 24

References ………………………………………………………………………………………... 25

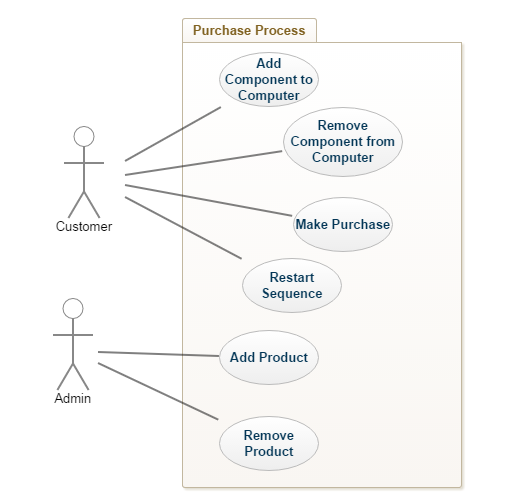
Code ………………………………………………………………………………………………….. 26

Narrative

For this project we decided to create a framework for a PC manufacturer and retailer to be used by a third party. It was created to allow customers to create a customisable Computer System. Customers will be able to pick and choose from a wide array of component types for their individual system. Administrators for the company should be able to add or remove any products they need to. They can customise the products based on the attributes such as name, price, stock etc. The framework will also provide a shipping cost to the user which will be calculated based on the weight. Potential users of the framework will be able to configure the products to suit their needs.

The use of design patterns is to help benefit people trying to incorporate the framework. These design patterns help the coders easily extend the framework to provide their own spin on things. This allows for numerous different types of systems to be created. The framework was developed in Eclipse and Java which allows for extensive portability due to Java Virtual Machines being available on a huge amount of platforms.

Use Cases



|  |  |
| --- | --- |
| Use Case 1 | Add Component to Computer |
| Actor Action | System Response |
| 1. Customer enters number of product. | 2. System adds the choice to the component. |
| 3. Customer is brought to next screen of next product. |  |
|  | Alternative System response |
|  | The choice is invalid and you’re returned back to the screen to select again. |

|  |  |
| --- | --- |
| Use Case 2 | Remove Component from Computer |
| Actor Action | System Response |
| 1. Customer enters -1 | 2. The System removes component and goes to previous component screen |
| 3. Customer is brought to previous screen. |  |

|  |  |
| --- | --- |
| Use Case 4 | Restart Sequence |
| Actor Action | System Response |
| 1. User selects ‘N’ | 2. Returned to initial screen |
| 3. Customer is brought to first screen |  |

|  |  |
| --- | --- |
| Use Case 5 | Add Product |
| Actor Action | System Response |
| 1. Admin specifies what type of component they want to add. | 2. Returns the attributes of the type of Product the user has selected. |
| 3. Admin enters attributes of Product to be added | 4. The Product is added to the Database. |

|  |  |
| --- | --- |
| Use Case 6 | Remove Product |
| Actor Action | System Response |
| 1. User enters number as choice for product to remove | 2. Product is removed from the list and screen is brought back to display of all available products. |

|  |  |
| --- | --- |
| Use Case 3 | Make Purchase |
| Goal in Context | Go through the components and update all them in the database. Calculate total cost of Computer System and display it to the customer in a receipt. |
| Preconditions | We have a Computer System with components. |
| Successful End Condition | Receipt is created with list of components and their prices. Stock is updated for all. |
| Failed End Condition | Receipt is not shown or stock not updated. |
| Primary, Secondary actors. | Customer, Database Handler, Shipping Service |
| Trigger | Customer selects “Y” when shown summary of products selected. |
| Description | 1. Customer selects “Y” when shown summary.  2. Changes are made to the stocks of components selected.  3. Shipping Price is calculated and total price is calculated.  4. Receipt is shown to the customer with all components and all prices displayed. |
| Extensions |  |
| Variations | 1. Customer enters “N” when shown summary and is brought back to initial screen to select components. |

Non-Functional Requirements

Extensibility

The need for good extensibility of a software project is imperative. With new features constantly being added to most projects, code additions are very likely to be made. Therefore, when planning how to carry out the project it should be created with a view of making future enhancements simple.

Portability

With the vast amount of operating systems and hardware available in the current market, building software to work across these platforms is essential. That is why it is necessary to outline the target audience and plan your projects design for those runtime environments at the beginning.

Performance

Recently, more than ever, the performance and speed of a product is possibly the most sought after NFR. In today’s society, immediate response is what every customer expects of a product. Any significant lag in response time and the customer will choose a competitive product to use.

Maintainability

Maintainability is a very important requirement in any software project. There will almost always need to be changes or additions made to the code in the future, making the need for it to be maintainable vital. Code should be written with the view that a new developer looking at it in the future is able to understand it with relative ease.

Interoperability

Interoperability is the ability for a system to exchange information with itself and be able to make use of it. Interoperability can also mean the exchanging of information between different software, but for the context of this project we will look to incorporate it as an information messaging system within our software.

Design Patterns

# **Visitor**

For our project we decided to use the Visitor design pattern to handle all the shipping cost calculations for each computer component we sell. From research (Amazon.com, 2016) we found that the shipping cost of items is often calculated based on the weight of the item sold. Therefore, we added a weight attribute to the Part interface which would be implemented for each Component object created. When calculating the total shipping cost after the user has picked out all their items, if it cost less than €100 we multiply the weight of the item by a chosen value (2 or 3). If the item costs more than €100, we ship the item at no extra cost.

Each time the user chooses a item/Component type object in the ProductList class, it gets added to an ArrayList of type VisitableElement. The accept() method in the VisitableElement class then gets called for each Component object and passing in a ShippingVisitor object at the same time, which calculates the cost of the shipping for that item. The getTotalShippingCost() method in ShippingVisitor is then called on its object to give back a final accumulated cost of shipping for all the items.

# **Decorator**

After the user has decided to purchase their items and confirmed their order they are presented with a receipt in the final class, ReceiptUI. To help create the receipt we used the Decorator design pattern to add header and footer sections surrounding the list of purchased items. This design pattern works by building different Decorator object types on top of each other.

“Receipt headerReceipt = new ThankYouReceipt(new HeaderReceipt(new BasicReceipt()));”

To create the header section, we first create a Receipt object of type BasicReceipt, which simply contains a “\nNote:” string, then a HeaderReceipt type containing the string “The Computer Shop” and finally a type ThankYouReceipt adding a string thanking the customer for shopping with us. We then print the list of items purchased after the output of this Receipt object.

Similarly, to create the footer section of the receipt we used the same Decorator Receipt logic but this time we used the types BasicReceipt and FooterReceipt to concatenate an output string asking the user to “come back again”.

Factory

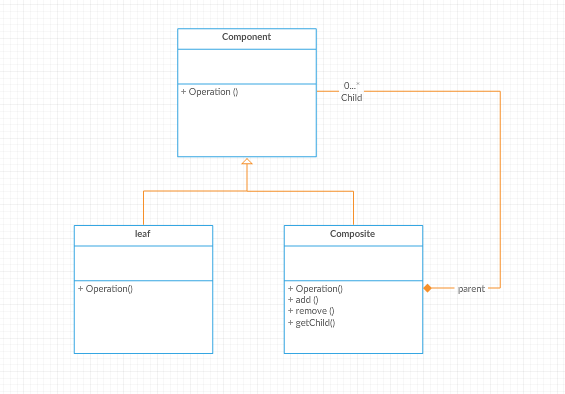
For our project, we decided to implement the Factory Design Pattern to help create different objects that extended the Component class. In order to accomplish this, we created our Factory Pattern class and made a method for each individual component that would take in all of the parameters and return an object of that corresponding component (Gamma, 1995).

“public CPU getCPU(int componentId, String componentName, double price, String typeOfComponent, String series, String CPUSocketType, double weight){

return new CPU(componentId, componentName, price, typeOfComponent, series, CPUSocketType, weight);”

Then in our DataControl class, we had our program read through our database of components and specifically looked for a parameter (in this case it was typeOfComponent) to help determine just what object we would be creating. We then ran the typeOfComponent through a switch case statement, and when there was a match, we call the Factory pattern class passing the parameters of the matched component and in this returning the newly created object and storing it in an ArrayList of components.

Composite



The composite pattern is a structural design pattern. We decided to use this design pattern in our project because we needed to treat a group of products as if they were all one product in the case of a Laptop where each could consist of an array of components such as a CPU, GPU, Motherboard, etc.

To do this, the pattern creates a class that contains a group of its own objects and provides a way to add, remove and get form this group of objects. Our analysis class diagram will show the composite tree structure to represent the part whole hierarchy. But essentially the ComputerSystem class is the composite class and it inherits from Component. GPU, CPU, Motherboard, RAM etc. can all be considered as Leaf components and they all inherit from the component class. The ComputerSystem can then add and remove to its Arraylist of components, and get the children at a specific index of that arraylist.

Observer

In our application the observer design pattern is used to keep the display of components from the product list up to date. The AdminProductList acts as the subject. It contains an array list of products that can be can be modified by the admin through methods in the class (e.g. add product, remove product). If these methods are called, the notifyObservers method is then called. In our example, the observer is AdminProductDisplay. It contains methods that return the data for displaying the list of products in the UI. It has an update method that pulls an up to date display from the Subject class. This update method is called in the notifyObservers method in AdminProductList meaning the display is aware of any state change that is made.

The advantages of this design pattern are the separation of display and data. Many different displays of products are possible. It also ensures that all displays of products are up to date with the current version of the data as the displays are updated immediately.

Memento

The memento design pattern was implemented to allow users to undo their selections when choosing the components to add to their computer system.

The memento design pattern uses three classes, memento, caretaker and originator. The memento contains the state of the object that may want to be restored later. The originator creates and stores mementos. The caretaker can restore an object state from Memento.

The Memento captures and externalizes an object's internal state so that the object can later be restored to that state.

Each time the user adds a component to the computer system, a memento of the computer system's state is created by the originator and the caretaker stores the state of the computer system so it can be restored later if necessary. This process also works if the user wants to skip adding a component to the computer system.

originator.set(tempComputerSystem);

caretaker.addMemento(originator.storeInMemento());

We added the option for the user to enter ‘-1’ when selecting components in order for them to undo their previously selected component. Two counters were created called savedSystems and currentSystem. The counter savedSystem keeps track of how many states of the computer system have been saved in the Memento arrayList inside Caretaker.java. When the user wants to undo their previous selection, the counter currentSystem is decremented, and this value represents which index should be retrieved from the memento arraylist inside the caretaker class. The users current computer system is then set to the computer system retrieved from the caretaker class, which allowed the user to effectivly undo their previous component selection.

Command

The command design pattern is used for decrementing the stock for each component that users add to their computer system. When a user has confirmed their component choices for their computer system, the completed computer system is passed to the decrementStock.java class inside StockManager. An invoker class is created which takes in DecreaseStock objects and adds them to an arrayList. The DecreaseStock objects constructor takes in a Component object. When DecreaseStock’s execute() method is called, the decrementStock() method is called for the component inside of the DecreaseStock object. This method decrease’s the Part/Component’s stock by one. The adjustStock() method inside of the DataControl class, then adjusts othe database / StockList.txt file by taking in the component name and whether to increment or decrement the stock for that component.

public static void decrementStock(ComputerSystem computerSystem){

Invoker stock = new Invoker();

int length = computerSystem.getComponents().size();

for(int i = 0;i < length;i++) {

DecreaseStock decrease = new DecreaseStock(computerSystem.getChildAtIndex(i));

stock.takeOrder(decrease);

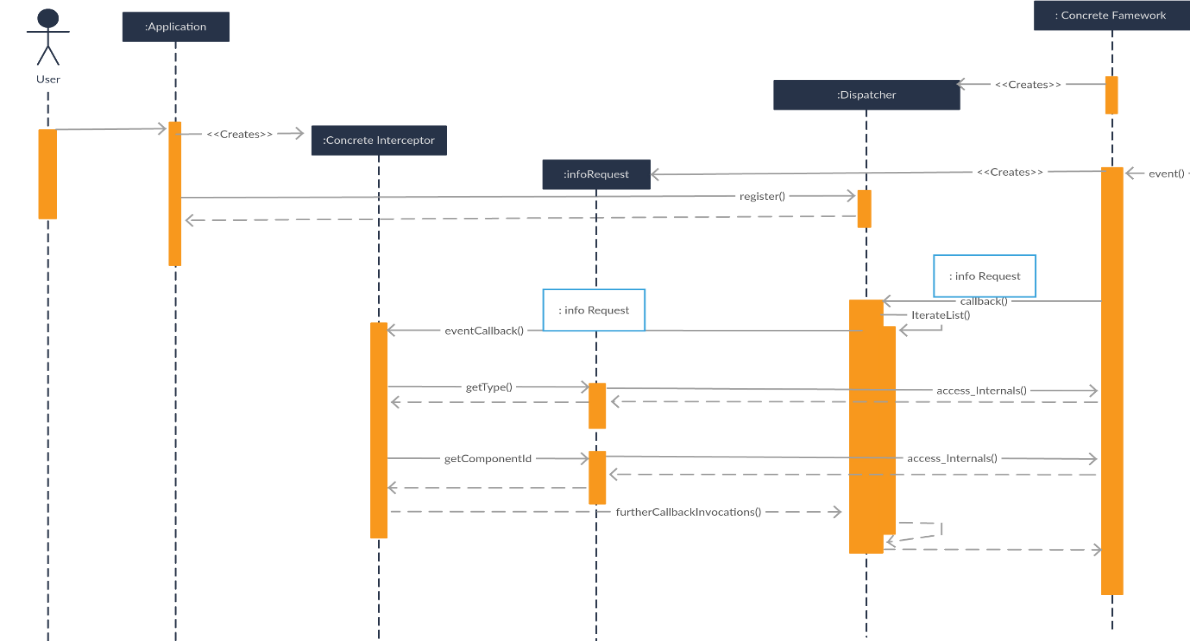
}

stock.placeOrders();

}

The purpose of the Command design pattern is to encapsulate the decrementStock() as an object called DecreaseStock which parameterises clients requests and support operations. It also decouples the object that invokes the operation (Invoker.java) from the one that performs it (decreaseStock.java).

Interceptor Architectural Pattern

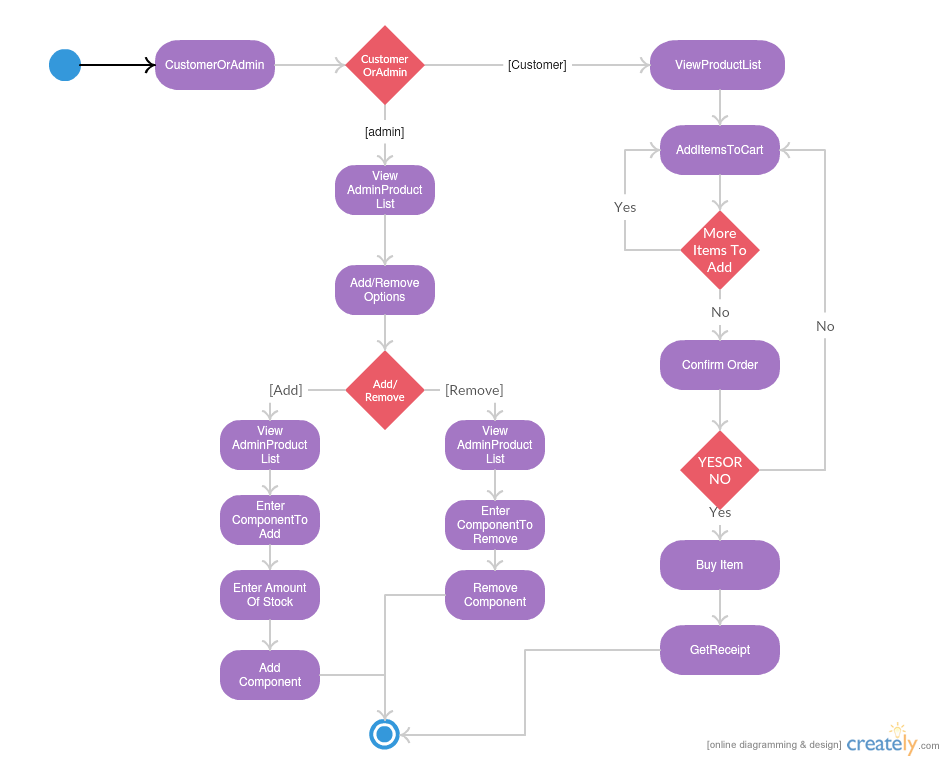


It is quite important to follow some kind of design guidelines to achieve extensible software. There are many articles on the internet describing the benefits of modular design in any level.

We chose to use the interceptor design pattern because we needed a way to build on our NFRs such as extensibility, flexibility and re-usability. We wanted a way of evolving the system without breaking any of our existing APIs which is where interceptors come to use. We wanted a separation of concerns so that it would make our life easier when it came to refactoring one of the elements or even replacing it with something more functional, as well as needing loose coupling making it easier to maintain and test.

It’s a very basic use of interception, so that you can see it in action. Logging to a logging file without bloating your business logic code with unnecessary responsibilities.

Process



From this state machine we established that there were 3 areas where we would consider putting interception points for out interceptor Architectural pattern, after add Component, after remove component, and after buying a Computer System.

Identify and model the interception Points



Specify the context object

We decided to settle with a single context object, with one interface as we were only going to have 1 interception point that passes parameters of the same type.

The context object supports reading and changing to the target object to support correct logging to the system. If we wanted in the future, we could use the context object to modify parameters, enforce encryption, validation or change behaviour reflectively. It has methods getType(), and getComponentId(), to get all of the relevant information for logging about that event.

We determined that we would only pass contextObjects to a concrete interceptor once only to create a concrete object on registration.

Specify the interceptors

We made one interceptor called infoRequest that the concrete framework will invoke with one designated callback hook(loggingServiceRequest) as we only had interception point in the group.

Specify the dispatchers.

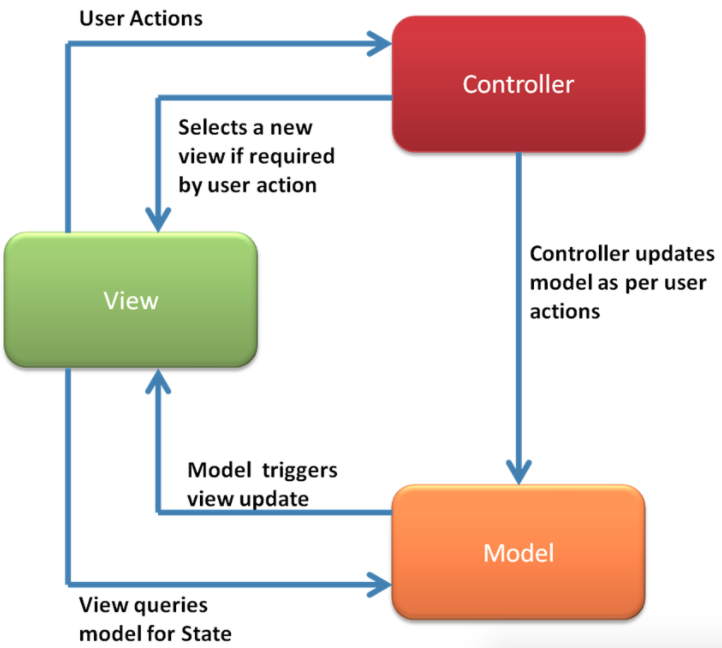
To register the interceptor, we had a static arrayList of Interceptors to store the Interceptors in. We passed the interceptor to a synchronised register method which then saved the interceptor in the static arraylist. We made the register method synchronised just to make sure that the interceptors were added in the correct order and one at a time.

When the interception occurs – the framework callbacks to the dispatcher, then the dispatcher invokes the designated callback hook(loggingServiceRequest) of all its concrete registered interceptors.

System Architecture

The system was split into 3 main subsystems: Business Layer, Data Layer and the User Interface Layer.

We implemented the Model View Controller (MVC) Design pattern (Jaggavarapu, M. 2016). which would make it easier to add any new features into the project as it developed. The MVC separates business logic from the user interface in order to allow for code reuse in the system and to reduce coupling between layers. The system is separated into the model, view and controller, where the view represents the GUI, the controller controls the information and the model updates the data.



The interceptor architectural pattern was also used so we could easily add new features or refactor existing elements in our project without breaking any of the existing APIs.

UML Workbench

We chose Creately, Gliffy and Genmymodel as our UML workbenches to draw the class and sequence diagrams due to the fact that they produce clear, easy to understand diagrams and are also free to use.

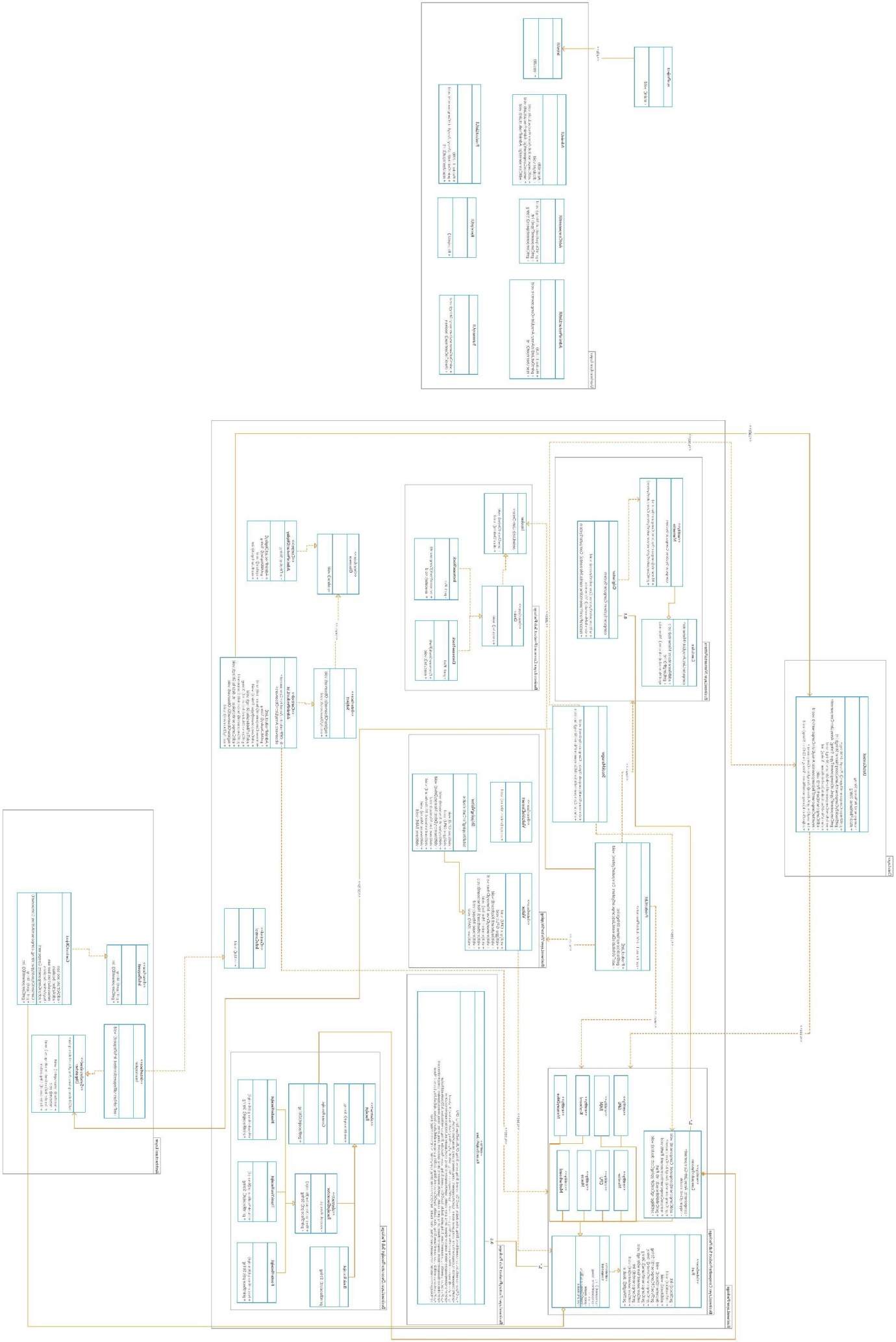
We found Creately more useful for implementing the larger class diagram as it was much easier to read and navigate compared to the alternatives. Gliffy was used ti create the sequence diagrams as we felt they were easier to create and understand compared to Creately.

Genmymodel was used to crate the use case diagrams as it was free to use and was the best of our options at creating the use case diagrams.

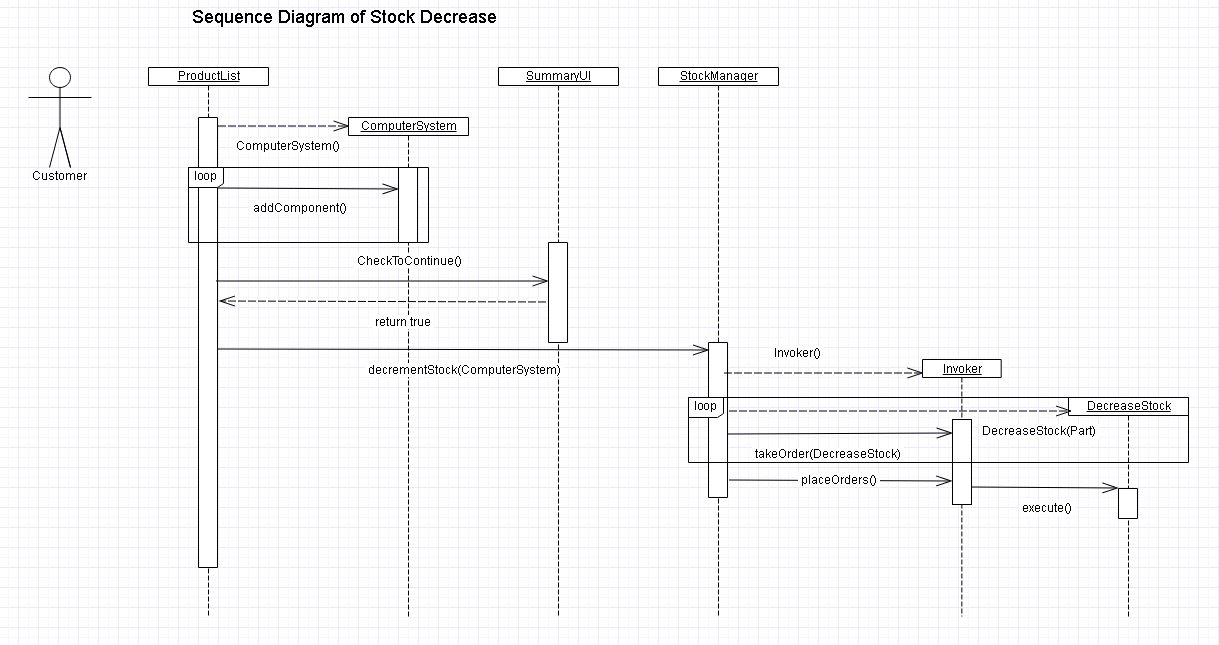
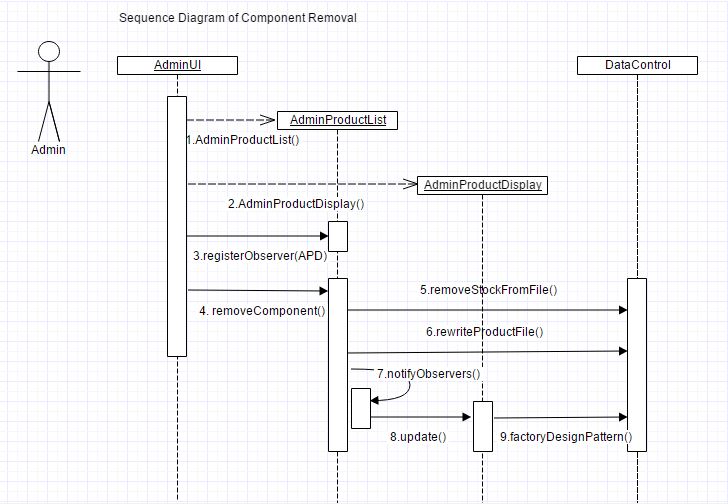
Implementation Language

We chose the Java programming language for our implementation. We chose Java as it is one of the most portable programming languages available, which means that our system is available to architecture that supports the Java Virtual Machine (JVM). Java has a JVM for many of the modern architectures which mean that our system will be available to many more users than if we used the C++ programming language.

Our program also does not rely on speed which is another reason why we didn't use C++. The added security, through Java’s garbage collector and the fact that all members of the team were comfortable programming in Java is what helped us make our decision.



Sequence Diagrams



Code Contribution

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Class | Michael | Conor | Daniel | Cian | Darren | Lines in File |
| ArchitechturalLayer Package |  |  |  |  |  |  |
| ContextObject |  |  |  |  | 47 | 47 |
| Dispatcher |  |  | 27 |  |  | 27 |
| infoRequest |  |  | 6 |  |  | 6 |
| Interceptor |  |  | 6 |  |  | 6 |
| BusinessLayer Package |  |  |  |  |  |  |
| IntroControl |  |  |  |  | 39 | 39 |
| ProductList |  | 146 |  | 64 | 4 | 214 |
| runProject | 24 |  |  |  |  | 24 |
| StockManager |  |  | 37 |  |  | 37 |
| BusinessLayer.CommandProduct Sub package |  |  |  |  |  |  |
| DecreaseStock |  |  |  | 17 |  | 17 |
| IncreaseStock |  |  |  | 18 |  | 18 |
| Invoker |  |  |  | 21 |  | 21 |
| Order |  |  |  | 5 |  | 5 |
| BusinessLayer.CompositeProduct  Sub Package |  |  |  |  |  |  |
| Component | 8 | 11 | 65 | 14 |  | 98 |
| ComputerSystem |  | 13 | 68 | 3 |  | 84 |
| CPU |  | 3 | 43 |  |  | 46 |
| GPU |  | 3 | 43 |  |  | 46 |
| Keyboard |  | 3 | 43 |  |  | 46 |
| MemoryDrive |  | 3 | 39 |  |  | 42 |
| Monitor |  | 3 | 51 |  |  | 54 |
| Motherboard |  | 3 | 52 |  |  | 55 |
| Mouse |  | 3 | 51 |  |  | 54 |
| Part |  |  | 11 |  |  | 11 |
| RAM |  | 3 | 42 |  |  | 45 |
| BusinessLayer.DecoratorReceipt Sub Package |  |  |  |  |  |  |
| BasicReceipt |  | 18 |  |  |  | 18 |
| CreateReceipt |  | 37 |  |  |  | 37 |
| FooterReceipt |  | 22 |  |  |  | 22 |
| HeaderReceipt |  | 22 |  |  |  | 22 |
| Receipt |  | 9 |  |  |  | 9 |
| ReceiptDecorator |  | 16 |  |  |  | 16 |
| ThankYouReceipt |  | 22 |  |  |  | 22 |
| BusinessLayer.MementoPattern Sub Package |  |  |  |  |  |  |
| Caretaker |  |  |  | 19 |  | 19 |
| Memento |  |  |  | 14 |  | 14 |
| Originator |  |  |  | 22 |  | 22 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Class | Michael | Conor | Daniel | Cian | Darren | Lines in File |
| BusinessLayer.FactoryProduct Sub Package |  |  |  |  |  |  |
| FactoryDesignPattern | 47 |  |  |  |  | 47 |
| BusinessLayer.ObserverPackage |  |  |  |  |  |  |
| AdminProductDisplay |  |  |  |  | 94 | 94 |
| AdminProductList |  |  |  |  | 89 | 89 |
| Observer |  |  |  |  | 8 | 8 |
| Subject |  |  |  |  | 7 | 7 |
| BusinessLayer.VisitorShipping Sub Package |  |  |  |  |  |  |
| ShippingVisitor |  | 68 |  |  |  | 68 |
| VisitableElement |  | 6 |  |  |  | 6 |
| Visitor |  | 15 |  |  |  | 15 |
| DataLayer Package |  |  |  |  |  |  |
| DataControl | 221 |  |  | 162 | 15 | 398 |
| UserInterface Layer Package |  |  |  |  |  |  |
| AdminUI |  |  |  |  | 67 | 67 |
| IntroUI | 34 |  |  |  |  | 34 |
| ProductListUI | 21 |  |  |  |  | 21 |
| ReceiptUI |  | 21 |  |  |  | 21 |
| SummaryUI |  |  | 40 |  |  | 40 |
| Test Package |  |  |  |  |  |  |
| customerTest |  |  | 42 |  |  | 42 |
| Total Added Vertically | 355 | 477 | 666 | 359 | 370 | 2,227 |

Added Values

Tactics for handling quality attributes:

We added value to this project by integrating tactics for handling quality attributes into our development through testing with Junit, versioning with Github and code metrics with SonarCube

Github:

GitHub is a web-based Git repository hosting service. It offers distributed version control and source code management (SCM).

It was a crucial part of the development process that made it far easier to work on. We were in a group of 5, so we obviously could not have been emailing each other snippets of code all the time. It would have been impossible. Github changed all of this. With the github application we could see all of the commits that everyone had made to the code. Everyone could see all the changes that were made to the code, and by simply typing “git pull” you would have a completely up to date version of the project. It often happened that 2 or 3 people were active during the same hour on Github, pushing commits to the same file. Often this caused merge conflicts that would have otherwise gone unnoticed.

We could also always go through the history of a file and go see all the changes made in every pull request. It was a very nice feature, because if you were ever wondering why a piece of code was present often you could find out why by cycling through the PRs and finding the PR that put it in and reading through the commit.

One thing that we probably should have done in retrospect was code reviews on the PR, each PR should have been approved by at least two other developers ensure the coding standard was on par (devinea, et al. 2016). Often we would find variables named with a single letter or similar silly mistakes. If we had used PR code reviews properly, I have no doubt but that SonarCube would have had much less issues with the code quality before we started fixing those issues.

Applying Metrics to the Code:

To help evaluate the quality of our source code, we used a software quality metrics tool called SonarQube. We chose SonarQube due to the huge amount of metric features it provides compared with competitors (Tomas, Escalona and Mejias, 2013). By running SonarQube from within the project folder it was able to analyse our source code for bugs, violations and bad code smells which may fail any of the vast array of software quality rules provided with SonarQube.

Once we were happy that we were finished the code aspect of our project, bar some minor additions or changes, we decided to apply the SonarQube software to our project. On its first run the results came back as seen in Figure 1 below.

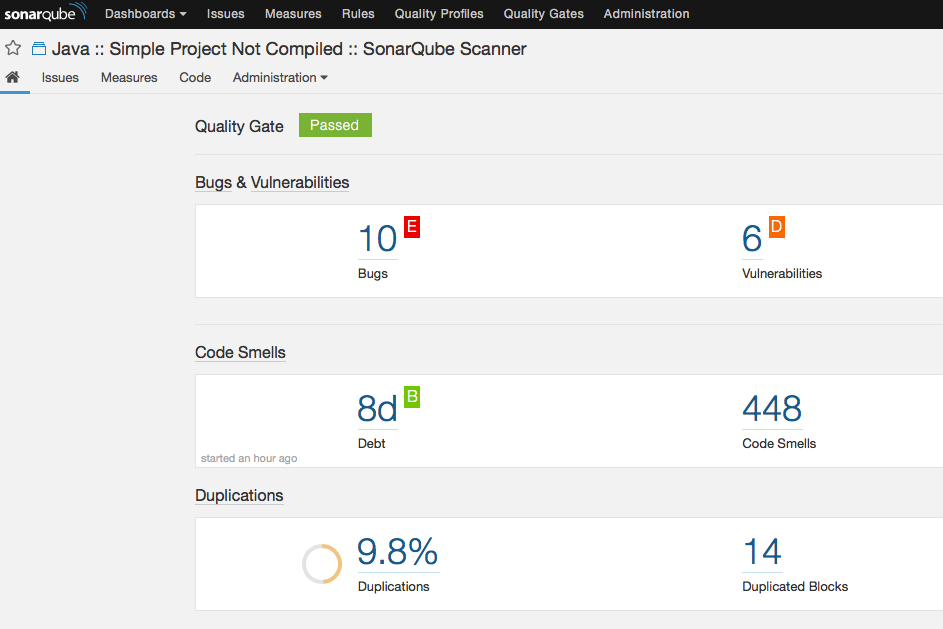


Figure 1: First analysis of our project using SonarQube.

From Figure 1 we see that it identified 10 bugs, 6 vulnerabilities and 448 code smells and applied ratings of E for Reliability, D for Security and B for Maintainability respective of the bugs, vulnerabilities and code smells found. However, looking through the code smells that were identified we decided to categorise some of them as not vitally important to fix in relation to this particular project. It is something we would certainly look at solving if we were to continue this project in the future. Some of the rules identifying code smells we decided we won’t fix include:

* Public types, methods and fields (API) should be documented with Javadoc.
  + Write code comments explaining each method, the parameters it takes and values it may return.
* Standard outputs should not be used directly to log anything.
  + We want to use standard out for this project so this rule enforcement isn’t necessary.
* Package, Interface, method and variable names should comply with a naming convention.
  + As there is only a small, constant number of people working on the project we all agreed to use a naming scheme relevant to what it was identifying and therefore not needing to create regex patterns to be enforced.

Removing rules such as these reduced the number of code smells down to 182. We then began to solve the various bugs, vulnerabilities and code smells by dealing with elements such as exception handling, Cyclomatic complexity measure failures, removing unused imports and variables, re-phrasing code and removing some duplicate blocks of code. In the end, comparing our initial SonarQube evaluation results in Figure 1 with the evaluation done after the refactoring in Figure 2, we saw a great increase in quality of our code with A-grades achieved in Reliability, Security and Maintainability.

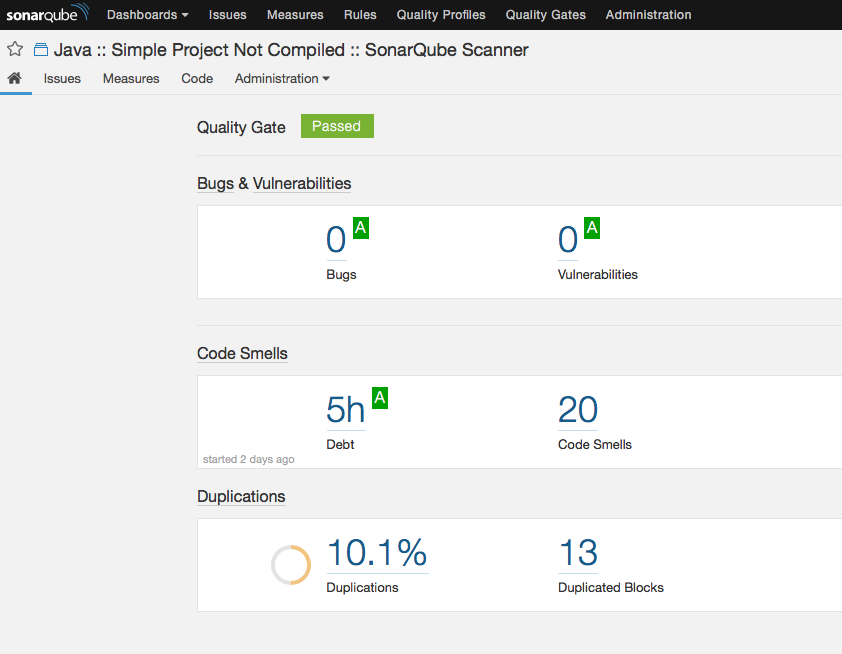


Figure 2: Final analysis of our project using SonarQube

Testing

One way to safeguard software to application breaking bugs is through the efficient use of unit tests. We decided to include automated testing into our application as it is a huge part of quality assurance.

The most important part thing to do when creating tests is to emulate an end to end user. So you might be tempted to say add more End to End tests right? But end to end test are expensive. They take up a lot of time! On top of that – when they do fail, they are not as efficient at pointing out the source of the bugs as integration and unit tests. It’s much easier to have a developer who understands their own code to write basic integration tests to their API that will run in seconds as opposed to starting your E2E tests (That an automation tester wrote) going to get a cup of coffee while they run and come back to it having not passed and still not understanding why!

Imagine a system where almost all the testing is done through E2E tests – if the sign in is broken, or something early in the E2E test is broken – almost 100% of your E2E tests are going to fail.

This is why we decided to only integrate unit tests. They are small, fast, reliable, and isolates failures. And if we really want to simulate some sort of E2E test, we can just string all of these unit tests into a simple integration test verifying that they coherently work together.

The 3 tests created were “testComposite”, “testReceipt”, and “testVisitor”, each just looks after its own design pattern and makes sure that they are operating correctly.

“testComposite” Gets the price of a computerSystem and ensures that it is the same as the price of all its composite components combined

“testReceipt” Ensures that the receipt gives back the correct name and price of each chosen component.

“testVisitor” ensures that the visitor pattern is working correctly by making sure that only products that cost under 100 euro contribute to the price of shipping. And that the price of shipping per component is equal to the weight multiplied by 3.

Issues Encountered

Memento

We experienced some issues with the implementation of the memento design pattern. After selecting a component to add to the user’s computer system, a memento capturing the state of the computer system is saved in an arrayList of type Memento in the caretaker class. However, when testing this we discovered that each time a memento of the computer system was saved, the state of the most recent computer system would duplicate itself onto all of the existing elements in the arraylist. This defeated the purpose of the memento design pattern as only the most recent state of the computer system would be saved, instead of all the states as components were added to the computer system.

We solved this problem by creating a new computer system inside the addMemento() method in the ProductList class, and we copied all of the components from the existing computer system and added them to this new ‘temporary’ computerSystem inside of the addMemento() class. This temporary computer system was then added to the originator which created a memento object. This memento object was then passed to the caretaker object, which saved it in the Memento ArrayList.

private void addMemento(ComputerSystem mainComputerSystem, Originator originator, Caretaker caretaker){

ComputerSystem tempComputerSystem = new ComputerSystem(1, "Laptop", "ComputerSystem", "Windows", 00.00);

ArrayList<Component> l = mainComputerSystem.getComponents();

for(int y = 0;y < l.size();y++) {

tempComputerSystem.addComponent(l.get(y));}

//adding a memento of the current computer system

originator.set(tempComputerSystem);

caretaker.addMemento(originator.storeInMemento());

}

This resolved our duplication issues and allowed the memento design pattern to work as expected.

Composite

In past attempts of previous projects, we made the mistake of implementing 2 different interfaces. One for the composite class and one for the leaf components. This was to ensure that ComputerSystem could not add another instance of a composite class to its ArrayList of components – why would a computer system ever include another computer system? However, we were told that this was unnecessary and not really abiding by the composite design pattern, for this project we implemented the composite with only one interface.

Critique of NFRs

Extensibility

The use of our chosen design and architectural patterns will help to ensure easier extensibility should additions be made in the future. Code following these standard development patterns mean that developers are able understand and enhance the code base with ease. With regards to the Factory Design Pattern, should the user decide to incorporate another component into the program, they would only need to add the unique features of that component in the component class. The command design pattern is a pattern that contributes to extensibility in our system. It encapsulates a request as an object, decrease stock or increase stock, therefore letting you parameterize clients requests and support operations. It also decouples the object that invokes the operation, Invoker.java from the one that performs it, increaseStock.java and decreaseStock.java.

Portability

We used Java as the programming language for our project. One of the major advantages of Java is the accompanying Java Virtual Machine (JVM). It allows any installation of the JVM to run a compilation of a Java project, greatly simplifying portability.

Performance

Without carrying out tests on our project we are quite happy with its performance and response times. There is no noticeable lag or delay in any operations it carries out. If we are to continue work with this project applying performance tests on our code, with particular interest on use case operations, would be of significant importance, especially if the code base is to grow to quite large.

Maintainability

To successfully support maintainability, we have added comments to some of our code and tried to keep the Cyclomatic complexity measure of our methods low. We also achieved an A-grade for maintainability in our SonarQube analysis.

Interoperability

The interceptor architectural pattern is an example of interoperability as it has many different messages between different pieces of software. Initially an event on the framework side triggers a creation of a context object which either modifies the framework or reads details from it. In our case, the context object just reads. It then relays these details through messages sent back to an Interceptor on the application side. This allows various third party developers to communicate with their framework in their own way through their individual concrete interceptors.

Usability

To improve the usability of our system we implemented the memento design pattern, which allows users to undo any component selection they made when creating their computer system. If the memento design pattern was not implemented users would have to delete their computer system and start again if they wanted to make any changes.

References

* Applying Metrics to Source Code: Tomas, P., Escalona, M. and Mejias, M. (2013). Open source tools for measuring the Internal Quality of Java software products. A survey. *Computer Standards & Interfaces*, 36(1), pp.244-255.
* MVC Design Pattern: Jaggavarapu, M. (2016). *Presentation Patterns : MVC, MVP, PM, MVVM*. [online] Manoj Jaggavarapu. Available at: https://manojjaggavarapu.wordpress.com/2012/05/02/presentation-patterns-mvc-mvp-pm-mvvm/
* Factory Design Pattern: Gamma, E., 1995. *Design patterns: elements of reusable object-oriented software*. Pearson Education India.
* PR Code Reviews: devinea, et al. (2016). *Try BUILD in Docker by devinea · Pull Request #51 · SAP/BUILD*. [online] GitHub. Available at: https://github.com/SAP/BUILD/pull/51
* Visitor Design Pattern: Amazon.com, (2016). *Amazon.com Help: Seller Shipping Rates*. [online] Amazon.com. Available at: https://www.amazon.com/gp/help/customer/display.html/ref=hp\_left\_v4\_sib?ie=UTF8&nodeId=201910890 [Accessed 7 Nov. 2016].

5

Code

ArchitecturalLayer – ContextObject.java

package ArchitecturalLayer;

import BusinessLayer.CompositeProduct.Component;

public class ContextObject implements infoRequest{

private boolean editAction;

private boolean addAction;

private boolean removeAction;

private boolean buyAction;

public Component actionComponent;

public ContextObject(String type, Component actionComponent){

this.actionComponent = actionComponent;

switch(type){

case "edit":editAction= true;

break;

case "add":addAction= true;

break;

case "remove":removeAction= true;

break;

case "Bought":buyAction= true;

break;

default:

System.*out*.println("Invalid");

break;

}

}

@Override

public String getType() {

if(editAction)

return "EDIT COMPONENT";

else if (addAction)

return "ADD COMPONENT";

else if(removeAction)

return "REMOVE COMPONENT";

else if(buyAction)

return "BOUGHT COMPUTERSYSTEM";

else return "INVALID ACTION";

}

@Override

public int getComponentID() {

return actionComponent.getComponentId();

}

}

ArchitecturalLayer – Dispatcher.java

package ArchitecturalLayer;

import java.util.ArrayList;

public class Dispatcher {

static ArrayList<Interceptor> *listOfInterceptors* = new ArrayList<>();

public synchronized void register(Interceptor i){

*listOfInterceptors*.add(i);

}

public synchronized void remove() {

// Method left empty

}

public synchronized void iterate\_list(infoRequest context) {

for(int i=0;i< *listOfInterceptors*.size(); i++) {

Interceptor ic = *listOfInterceptors*.get(i);

ic.loggingServiceRequest(context);

}

}

public Dispatcher theInstance() {

return this;

}

}

ArchitecturalLayer – infoRequest.java

package ArchitecturalLayer;

public interface infoRequest {

public String getType();

public int getComponentID();

}

ArchitecturalLayer – Interceptor.java

package ArchitecturalLayer;

@FunctionalInterface

public interface Interceptor {

public void loggingServiceRequest(infoRequest context);

}

BusinessLayer – IntroControl.java

package BusinessLayer;

import java.io.File;

import java.io.FileWriter;

import java.io.PrintWriter;

import java.util.logging.Level;

import java.util.logging.Logger;

import ArchitecturalLayer.Dispatcher;

import ArchitecturalLayer.Interceptor;

import ArchitecturalLayer.infoRequest;

import BusinessLayer.ObserverProduct.AdminProductList;

public class IntroControl {

private static final Logger *LOGGER* = Logger.*getLogger*( AdminProductList.class.getName() );

public void setUp(){

Interceptor myInterceptor = new Interceptor () {

public void loggingServiceRequest(infoRequest context) {

String description = context.getType();

int componentID = context.getComponentID();

File loggingFile = new File("LoggingFile.txt");

try {

FileWriter logWriter = new FileWriter(loggingFile , true);

PrintWriter out = new PrintWriter(logWriter);

out.println(description + "," + componentID + "," + System.*currentTimeMillis*());

out.close();

} catch (Exception e) {

*LOGGER*.log(Level.*SEVERE*, "context", e);

}

}

};

Dispatcher dispatcher = new Dispatcher();

dispatcher.register(myInterceptor);

}

}

BusinessLayer – ProductList.java

package BusinessLayer;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import ArchitecturalLayer.ContextObject;

import ArchitecturalLayer.Dispatcher;

import BusinessLayer.CompositeProduct.\*;

import BusinessLayer.VisitorShipping.\*;

import BusinessLayer.MementoPattern.\*;

import DataLayer.DataControl;

import UserInterfaceLayer.\*;

public class ProductList {

private List<VisitableElement> cartItems;

private Dispatcher dispatcher;

public ProductList() throws IOException {

Originator originator = new Originator();

Caretaker caretaker = new Caretaker();

boolean summaryConfirmToContinue = false;

dispatcher = new Dispatcher();

while(!summaryConfirmToContinue) {

String [] listOfComponentOptions = new String[] {"CPU", "GPU", "Keyboard", "MemoryDrive", "Monitor", "Motherboard", "Mouse", "RAM"};

ComputerSystem computerSystem = new ComputerSystem(1, "Laptop", "ComputerSystem", "Windows", 00.00);

addMemento(computerSystem,originator,caretaker);

cartItems = new ArrayList<>();

//saved systems = the amount of mementos created

//current system is used to return to the previous memento

int savedSystems = 0;

int currentSystem = 0;

// Loop for each component type in listOfComponentOptions array

for(int i = 0;i < listOfComponentOptions.length;) {

// Initialise an ArrayList to be filled with component type stored in index

ArrayList<Component> listOfComponentTypeOptions = new ArrayList<>();

// Ask for an ArrayList of all products of index component type

listOfComponentTypeOptions = DataControl.*getComponentTypeList*(listOfComponentOptions[i]);

if (!listOfComponentTypeOptions.isEmpty()) {

String outputComponentListString = "\n\nPlease choose a component for your computer:\n\n";

for (int index = 0; index < listOfComponentTypeOptions.size(); index++) {

outputComponentListString += "Choice: " + (index + 1) + "\t" + " " + listOfComponentTypeOptions.get(index).getComponentDetails() +"\n\n";

}

outputComponentListString += "Press 0 to skip component.\nPress -1 to undo selection.\n";

ProductListUI.*printOutput*(outputComponentListString);

int userChoice = *readUserComponentChoice*(listOfComponentTypeOptions.size());

if (userChoice >= -1) {

//if the user chooses 0 nothing is added to the computer system

if(userChoice >= 0)

computerSystem.addComponent(listOfComponentTypeOptions.get(userChoice));

addMemento(computerSystem, originator, caretaker);

savedSystems++;

currentSystem++;

i++;

/\* Whenever a product is added to the computer system the currentSystem must be equal to

\* the number of savedSystems to ensure that when the user wants to undo a selection

\* it always returns the last added memento\*/

currentSystem = savedSystems;

}

//if the user enters -1 (undo)

else if(userChoice == -2) {

// user cannot undo if no components have been added to the computer system

if(i <= 0)

System.*out*.print("No components currently in computer system");

else {

/\* .clear() removes all the components in a computer system

\* this is needed in order to set the main computer system as the previously saved memento

\*/

computerSystem.clear();

currentSystem--;

undo(currentSystem, computerSystem,originator,caretaker);

i--;

}

}

}

else {

ProductListUI.*printOutput*("We are currently out of all components of type" + listOfComponentOptions[i] + ".\nPlease consider returning after we restock our products");

}

}

// Visitor Design Pattern - Get Shipping Cost

ShippingVisitor shippingVisitor = new ShippingVisitor();

addToVisitableElementList(computerSystem);

for (VisitableElement item: cartItems)

item.accept(shippingVisitor);

double shippingCostTotal = shippingVisitor.getTotalShipping();

double totalCostBeforeShipping = computerSystem.getPrice();

double totalCostAfterShipping = totalCostBeforeShipping + shippingCostTotal;

computerSystem.setShippingCost(totalCostAfterShipping);

SummaryUI.*printOutSummary*(computerSystem.getSummary());

summaryConfirmToContinue = SummaryUI.*checkToContinue*();

StockManager.*decrementStock*(computerSystem);

ContextObject context = new ContextObject("Bought" , computerSystem);

dispatcher.iterate\_list(context);

ReceiptUI aReceipt = new ReceiptUI(computerSystem);

}

}

public int getStock(String compName) throws FileNotFoundException{

return DataControl.*getStockByComponentName*(compName);

}

private void addToVisitableElementList(ComputerSystem computerSystem) {

ArrayList<Component> componentList = computerSystem.getComponents();

for (Component component: componentList) {

switch(component.getTypeOfComponent()) {

case "CPU" :

CPU cpu = (CPU) component;

cartItems.add(cpu);

break;

case "GPU" :

GPU gpu = (GPU) component;

cartItems.add(gpu);

break;

case "Keyboard" :

Keyboard keyboard = (Keyboard) component;

cartItems.add(keyboard);

break;

case "MemoryDrive" :

MemoryDrive memoryDrive = (MemoryDrive) component;

cartItems.add(memoryDrive);

break;

case "Monitor" :

Monitor monitor = (Monitor) component;

cartItems.add(monitor);

break;

case "Motherboard" :

Motherboard motherboard = (Motherboard) component;

cartItems.add(motherboard);

break;

case "Mouse" :

Mouse mouse = (Mouse) component;

cartItems.add(mouse);

break;

case "RAM" :

RAM ram = (RAM) component;

cartItems.add(ram);

break;

default:

System.*out*.println("Invalid Component Type. ProductList.java");

break;

}

}

}

private void addMemento(ComputerSystem mainComputerSystem, Originator originator, Caretaker caretaker){

ComputerSystem tempComputerSystem = new ComputerSystem(1, "Laptop", "ComputerSystem", "Windows", 00.00);

ArrayList<Component> l = mainComputerSystem.getComponents();

for(int y = 0;y < l.size();y++) {

tempComputerSystem.addComponent(l.get(y));

}

//adding a memento of the current computer system

originator.set(tempComputerSystem);

caretaker.addMemento(originator.storeInMemento());

}

private void undo(int previousMemento, ComputerSystem computerSystem, Originator originator, Caretaker caretaker) {

ComputerSystem computerSystem1 = new ComputerSystem(1, "Laptop", "ComputerSystem", "Windows", 00.00);

computerSystem1 = originator.restoreFromMemento(caretaker.getMemento(previousMemento));

System.*out*.print("\n\n\nMemento computer system" + computerSystem1.getSummary());

//adding the components from the previous memento to the computer system

for(int j = 0;j < (computerSystem1.getComponents()).size();j++) {

Component component = computerSystem1.getChildAtIndex(j);

computerSystem.addComponent(component);

}

}

private static int readUserComponentChoice(int amountOfComponents) {

boolean acceptableInput = false;

int checkedUserChoice;

String uncheckedUserChoice = "";

while (!acceptableInput) {

ProductListUI.*printOutput*("\nEnter choice number: ");

uncheckedUserChoice = ProductListUI.*readUserInput*();

if (uncheckedUserChoice.matches("[0-9]+|-1") && Integer.*parseInt*(uncheckedUserChoice) <= amountOfComponents)

acceptableInput = true;

else

ProductListUI.*printOutput*("Error: Invalid. Must be an available choice number");

}

checkedUserChoice = Integer.*parseInt*(uncheckedUserChoice);

return checkedUserChoice - 1;

}

}

BusinessLayer – runProject.java

package BusinessLayer;

import java.util.logging.Level;

import java.util.logging.Logger;

import BusinessLayer.ObserverProduct.AdminProductList;

import UserInterfaceLayer.IntroUI;

public class runProject {

private static final Logger *LOGGER* = Logger.*getLogger*( AdminProductList.class.getName() );

public static void main(String[] args){

try{

IntroUI program = new IntroUI();

}

catch (Exception e) {

*LOGGER*.log(Level.*SEVERE*, "context", e);

}

}

}

BusinessLayer – StockManager.java

package BusinessLayer;

import java.io.FileNotFoundException;

import BusinessLayer.CommandProduct.DecreaseStock;

import BusinessLayer.CommandProduct.Invoker;

import BusinessLayer.CompositeProduct.ComputerSystem;

import DataLayer.DataControl;

public class StockManager {

private StockManager() {

//Adding private constructor to hide implicit public one

}

public static void decrementStock(ComputerSystem computerSystem){

Invoker stock = new Invoker();

int length = computerSystem.getComponents().size();

for(int i = 0;i < length;i++) {

DecreaseStock decrease = new DecreaseStock(computerSystem.getChildAtIndex(i));

stock.takeOrder(decrease);

}

stock.placeOrders();

}

public static boolean checkIsComponentInStock(String componentName) throws FileNotFoundException {

int currentStock = DataControl.*getStockByComponentName*(componentName);

boolean inStock;

if (currentStock > 0 ){

inStock = true;

}

else {

inStock = false;

}

return inStock;

}

}

BusinessLayer.CommandProduct – DecreaseStock.java

package BusinessLayer.CommandProduct;

import BusinessLayer.CompositeProduct.\*;

public class DecreaseStock implements Order {

private Part part;

public DecreaseStock(Part part){

this.part = part;

}

public void execute() {

part.decrementStock();

}

}

BusinessLayer.CommandProduct – IncreaseStock.java

package BusinessLayer.CommandProduct;

import BusinessLayer.CompositeProduct.\*;

public class IncreaseStock implements Order {

private Part part;

public IncreaseStock(Component part){

this.part = part;

}

public void execute() {

part.addStock();

}

}

BusinessLayer.CommandProduct – Invoker.java

package BusinessLayer.CommandProduct;

import java.util.ArrayList;

import java.util.List;

public class Invoker {

private List<Order> orderList = new ArrayList<Order>();

public void takeOrder(Order order){

orderList.add(order);

}

public void placeOrders(){

for (Order order : orderList) {

order.execute();

}

orderList.clear();

}

}

BusinessLayer.CommandProduct – Order.java

package BusinessLayer.CommandProduct;

public interface Order {

void execute();

}

BusinessLayer.CompositeProduct – Component.java

package BusinessLayer.CompositeProduct;

import java.util.logging.Level;

import java.util.logging.Logger;

import BusinessLayer.ObserverProduct.AdminProductList;

import DataLayer.DataControl;

public abstract class Component implements Part {

private int componentId;

private String componentName;

private double price;

private String typeOfComponent;

private double weight;

private static final Logger *LOGGER* = Logger.*getLogger*( AdminProductList.class.getName() );

/\*\*

\* Construct a new component using the provided item

\* and price.

\*

\* @param componentId the componentId of the component.

\* @param stock the amount left to sell for the component

\* @param componentName the componentName of the component.

\*/

public Component(int componentId, String componentName, double price, String typeOfComponent, double weight) {

this.componentId = componentId; //Generate new Id function should be used here

this.componentName = componentName;

this.price = price;

this.typeOfComponent = typeOfComponent;

this.weight = weight;

}

public int getComponentId() {

return componentId;

}

public void setComponentId(int newComponentId) {

componentId = newComponentId;

}

public String getComponentName() {

return componentName;

}

public void setComponentName(String newComponentName) {

componentName = newComponentName;

}

public double getPrice(){

return price;

}

public void setPrice(double newPrice){

price = newPrice;

}

public String getTypeOfComponent() {

return typeOfComponent;

}

public void setTypeOfComponent(String typeOfComponent) {

this.typeOfComponent = typeOfComponent;

}

public double getWeight() {

return this.weight;

}

public String getComponentDetails() {

return "\nComponent ID: " + this.componentId + "\nComponent Name: " + this.componentName +

"\nPrice: " + this.price + "\nType: " + this.typeOfComponent;

}

public void decrementStock() {

try {

DataControl.*adjustStock*(this.componentName, "decrement");

} catch (Exception e) {

*LOGGER*.log(Level.*SEVERE*, "context", e);

}

}

public void addStock() {

try {

DataControl.*adjustStock*(this.componentName, "increment");

} catch (Exception e) {

*LOGGER*.log(Level.*SEVERE*, "context", e);

}

}

@Override

public String toString(){

String output;

output = componentId + "," + componentName + "," + price + "," + typeOfComponent + "," + weight;

return output;

}

}

BusinessLayer.CompositeProduct – ComputerSystem.java

package BusinessLayer.CompositeProduct;

import java.text.DecimalFormat;

import java.util.ArrayList;

public class ComputerSystem extends Component{

private String OS;

private ArrayList<Component> components;

private double shippingCost;

public ComputerSystem(int componentId, String componentName, String typeOfComponent, String OS, double weight) {

super(componentId, componentName, 0, typeOfComponent, weight);

this.OS = OS;

this.components = new ArrayList<>();

this.shippingCost = 0.0;

}

public void addComponent(Component component) {

components.add(component);

}

public ArrayList<Component> getComponents() {

return components;

}

public void removeComponent(Part component) {

components.remove(component);

}

public Component getChildAtIndex(int i) {

return components.get(i);

}

public String getOS() {

return OS;

}

public void setOS(String OS) {

this.OS = OS;

}

public void setShippingCost(double shippingCost) {

this.shippingCost = shippingCost;

}

public double getPrice() {

double unitCost = 0;

if(!components.isEmpty()){

for (int i = 0; i < components.size(); i++) {

unitCost += components.get(i).getPrice();

}

}

unitCost += shippingCost;

return unitCost;

}

public String getComponentDetails(){

return super.getComponentDetails() + "\nOperating System: " + this.OS;

}

public String getSummary() {

String summary = "";

if(!components.isEmpty()){

for (int i = 0; i < components.size(); i++) {

String compInfo = components.get(i).getComponentName() + "\t\t:" + components.get(i).getPrice() + "\n";

summary += compInfo;

}

summary += "\nShipping Cost = " + new DecimalFormat("##.##").format(shippingCost);

summary += "\nTotal Price incl. Shipping = " + new DecimalFormat("##.##").format(getPrice()) + "\n";

}

return summary;

}

public String toString(){

String output;

output = super.toString();

output += "," + OS;

return output;

}

public void clear() {

components.removeAll(components);

}

}

BusinessLayer.CompositeProduct – CPU.java

package BusinessLayer.CompositeProduct;

import BusinessLayer.VisitorShipping.VisitableElement;

import BusinessLayer.VisitorShipping.Visitor;

public class CPU extends Component implements VisitableElement {

private String series;

private String CPUSocketType;

public CPU(int componentId, String componentName, double price, String typeOfComponent, String series, String CPUSocketType, double weight) {

super(componentId, componentName, price, typeOfComponent, weight);

this.series = series;

this.CPUSocketType = CPUSocketType;

}

public String getSeries() {

return series;

}

public void setSeries(String series) {

this.series = series;

}

public String getCPUSocketType() {

return CPUSocketType;

}

public void setCPUSocketType(String CPUSocketType) {

this.CPUSocketType = CPUSocketType;

}

public String getComponentDetails(){

return super.getComponentDetails() + "\nCPU series: " + this.series + "\nCPU SocketType:" + this.CPUSocketType;

}

public String toString(){

String output;

output = super.toString();

output += "," + series + "," + CPUSocketType;

return output;

}

// For Visitor

@Override

public void accept(Visitor visitor) {

visitor.visit(this);

}

}

BusinessLayer.CompositeProduct – GPU.java

package BusinessLayer.CompositeProduct;

import BusinessLayer.VisitorShipping.VisitableElement;

import BusinessLayer.VisitorShipping.Visitor;

public class GPU extends Component implements VisitableElement {

private String memory;

private int displayPorts;

public GPU(int componentId, String componentName, double price, String typeOfComponent, String memory, int displayPorts, double weight) {

super(componentId, componentName, price, typeOfComponent, weight);

this.memory = memory;

this.displayPorts = displayPorts;

}

public String getMemory() {

return memory;

}

public void setMemory(String memory) {

this.memory = memory;

}

public int getDisplayPorts() {

return displayPorts;

}

public void setDisplayPorts(int displayPorts) {

this.displayPorts = displayPorts;

}

public String getComponentDetails(){

return super.getComponentDetails() + "\nGPU Memory: " + this.memory + "\nGPU DisplayPorts:" + this.displayPorts;

}

public String toString(){

String output;

output = super.toString();

output += "," + memory + "," + displayPorts;

return output;

}

// For Visitor

@Override

public void accept(Visitor visitor) {

visitor.visit(this);

}

}

BusinessLayer.CompositeProduct – Keyboard.java

package BusinessLayer.CompositeProduct;

import BusinessLayer.VisitorShipping.VisitableElement;

import BusinessLayer.VisitorShipping.Visitor;

public class Keyboard extends Component implements VisitableElement {

private String keySwitchType;

private boolean backlit;

public Keyboard(int componentId, String componentName, double price, String typeOfComponent, String keySwitchType, boolean backlit, double weight) {

super(componentId, componentName, price, typeOfComponent, weight);

this.keySwitchType = keySwitchType;

this.backlit = backlit;

}

public String getKeySwitchType() {

return keySwitchType;

}

public void setKeySwitchType(String keySwitchType) {

this.keySwitchType = keySwitchType;

}

public boolean isBacklit() {

return backlit;

}

public void setBacklit(boolean backlit) {

this.backlit = backlit;

}

public String getComponentDetails(){

return super.getComponentDetails() + "\nKeySwitchType: " + this.keySwitchType + "\nBacklit: " + this.backlit;

}

public String toString(){

String output;

output = super.toString();

output += "," + keySwitchType + "," + backlit;

return output;

}

// For Visitor

@Override

public void accept(Visitor visitor) {

visitor.visit(this);

}

}

BusinessLayer.CompositeProduct – MemoryDrive.java

package BusinessLayer.CompositeProduct;

import BusinessLayer.VisitorShipping.VisitableElement;

import BusinessLayer.VisitorShipping.Visitor;

public class MemoryDrive extends Component implements VisitableElement {

private int driveCapacity;

private boolean SSD;

public MemoryDrive(int componentId, String componentName, double price, String typeOfComponent, int driveCapacity, boolean SSD, double weight) {

super(componentId, componentName, price, typeOfComponent, weight);

this.driveCapacity = driveCapacity;

this.SSD = SSD;

}

public int getDriveCapacity() {

return driveCapacity;

}

public void setDriveCapacity(int driveCapacity) {

this.driveCapacity = driveCapacity;

}

public boolean isSSD() {

return SSD;

}

public void setRPM(boolean SSD) {

this.SSD = SSD;

}

public String toString() {

return super.toString() + "," + this.driveCapacity + "," + this.SSD;

}

public String getComponentDetails(){

return super.getComponentDetails() + "\nDriveCapacity: " + this.driveCapacity + "\nSSD T/F : " + this.SSD;

}

// For Visitor

@Override

public void accept(Visitor visitor) {

visitor.visit(this);

}

}

BusinessLayer.CompositeProduct – Monitor.java

package BusinessLayer.CompositeProduct;

import BusinessLayer.VisitorShipping.VisitableElement;

import BusinessLayer.VisitorShipping.Visitor;

public class Monitor extends Component implements VisitableElement {

private String screenResolution;

private boolean curved;

private boolean is3d;

public Monitor(int componentId, String componentName, double price, String typeOfComponent, String screenResolution, boolean curved, boolean is3d, double weight) {

super(componentId, componentName, price, typeOfComponent, weight);

this.screenResolution = screenResolution;

this.curved = curved;

this.is3d = is3d;

}

public String getScreenResolution() {

return screenResolution;

}

public void setScreenResolution(String screenResolution) {

this.screenResolution = screenResolution;

}

public boolean isCurved() {

return curved;

}

public void setCurved(boolean curved) {

this.curved = curved;

}

public boolean is3d() {

return is3d;

}

public void setIs3d(boolean is3d) {

this.is3d = is3d;

}

public String getComponentDetails(){

return super.getComponentDetails() + "\nScreenResolution: " + this.screenResolution + "\nCurved T/F: " + this.curved + "\nIs 3D T/F: " + this.is3d;

}

public String toString(){

String output;

output = super.toString();

output += "," + screenResolution + "," + curved + "," + is3d;

return output;

}

// For Visitor

@Override

public void accept(Visitor visitor) {

visitor.visit(this);

}

}

BusinessLayer.CompositeProduct – Motherboard.java

package BusinessLayer.CompositeProduct;

import BusinessLayer.VisitorShipping.VisitableElement;

import BusinessLayer.VisitorShipping.Visitor;

public class Motherboard extends Component implements VisitableElement {

private String cpuSocketType;

private String series;

private String memoryStandard;

public Motherboard(int componentId, String componentName, double price, String typeOfComponent, String cpuSocketType, String series, String memoryStandard, double weight) {

super(componentId, componentName, price, typeOfComponent, weight);

this.cpuSocketType = cpuSocketType;

this.series = series;

this.memoryStandard = memoryStandard;

}

public String getCpuSocketType() {

return cpuSocketType;

}

public void setCpuSocketType(String cpuSocketType) {

this.cpuSocketType = cpuSocketType;

}

public String getSeries() {

return series;

}

public void setSeries(String series) {

this.series = series;

}

public String getMemoryStandard() {

return memoryStandard;

}

public void setMemoryStandard(String memoryStandard) {

this.memoryStandard = memoryStandard;

}

public String getComponentDetails(){

return super.getComponentDetails() + "\nCPU Socket Type: " + this.cpuSocketType + "\nSeries" + this.series + "\nMemoryStandard: " + this.memoryStandard;

}

public String toString(){

String output;

output = super.toString();

output += "," + cpuSocketType + "," + series + "," + memoryStandard;

return output;

}

// For Visitor

@Override

public void accept(Visitor visitor) {

visitor.visit(this);

}

}

BusinessLayer.CompositeProduct – Mouse.java

package BusinessLayer.CompositeProduct;

import BusinessLayer.VisitorShipping.VisitableElement;

import BusinessLayer.VisitorShipping.Visitor;

public class Mouse extends Component implements VisitableElement {

private int dpi;

private boolean programmableButtons;

private boolean dpiSwitching;

public Mouse(int componentId, String componentName, double price, String typeOfComponent, int dpi, boolean programmableButtons, boolean dpiSwitching, double weight) {

super(componentId, componentName, price, typeOfComponent, weight);

this.dpi = dpi;

this.programmableButtons = programmableButtons;

this.dpiSwitching = dpiSwitching;

}

public int getDpi() {

return dpi;

}

public void setDpi(int dpi) {

this.dpi = dpi;

}

public boolean isProgrammableButtons() {

return programmableButtons;

}

public void setProgrammableButtons(boolean programmableButtons) {

this.programmableButtons = programmableButtons;

}

public boolean isDpiSwitching() {

return dpiSwitching;

}

public void setDpiSwitching(boolean dpiSwitching) {

this.dpiSwitching = dpiSwitching;

}

public String getComponentDetails(){

return super.getComponentDetails() + "\nMouse DPI: " + this.dpi + "\nProgrammable Buttons: " + this.programmableButtons + "DPI Switching : " + this.dpiSwitching;

}

public String toString(){

String output;

output = super.toString();

output += "," + dpi + "," + programmableButtons + "," + dpiSwitching;

return output;

}

// For Visitor

@Override

public void accept(Visitor visitor) {

visitor.visit(this);

}

}

BusinessLayer.CompositeProduct – Part.java

package BusinessLayer.CompositeProduct;

public interface Part {

/\*\*

\* Get the ID of this Component.

\*

\* @return the ID of this Component.

\*/

int getComponentId();

void setComponentId(int newComponentId);

/\*\*

\* Get the componentName of this Component.

\*

\* @return the componentName of this Component.

\*/

String getComponentName();

void setComponentName(String newComponentName);

/\*\*

\* Get a String that describes this Component. Note that

\* this method overrides the toString method inherited

\* from Object.

\*

\* @return a String describing this Component.

\*/

String getTypeOfComponent();

String getComponentDetails();

void addStock();

void decrementStock();

double getWeight();

}

BusinessLayer.CompositeProduct – RAM.java

package BusinessLayer.CompositeProduct;

import BusinessLayer.VisitorShipping.VisitableElement;

import BusinessLayer.VisitorShipping.Visitor;

public class RAM extends Component implements VisitableElement {

private int gigaBytes;

private String multiChannel;

public RAM(int componentId, String componentName, double price, String typeOfComponent, int gigaBytes, String multiChannel, double weight) {

super(componentId, componentName, price, typeOfComponent, weight);

this.gigaBytes = gigaBytes;

this.multiChannel = multiChannel;

}

public int getGigaBytes() {

return gigaBytes;

}

public void setGigaBytes(int gigaBytes) {

this.gigaBytes = gigaBytes;

}

public String getMultiChannel() {

return multiChannel;

}

public void setMultiChannel(String multiChannel) {

this.multiChannel = multiChannel;

}

public String getComponentDetails(){

return super.getComponentDetails() + "\nGigaBytes: " + this.gigaBytes + "\nMultiChannel" + this.multiChannel;

}

public String toString(){

String output;

output = super.toString();

output += "," + gigaBytes + "," + multiChannel;

return output;

}

// For Visitor

@Override

public void accept(Visitor visitor) {

visitor.visit(this);

}

}

BusinessLayer.DecoratorRecipt – BasicReceipt.java

package BusinessLayer.DecoratorReceipt;

public class BasicReceipt extends Receipt {

@Override

public String printReceipt() {

return "\nNote:";

}

}

BusinessLayer.DecoratorRecipt – CreateReceipt.java

package BusinessLayer.DecoratorReceipt;

import java.text.DecimalFormat;

import java.util.ArrayList;

import BusinessLayer.CompositeProduct.\*;

public class CreateReceipt {

private static ComputerSystem *computerSystem*;

public CreateReceipt(ComputerSystem computerSystem) {

this.*computerSystem* = computerSystem;

}

public static String getReceipt() {

// Create heading section of receipt

Receipt headerReceipt = new ThankYouReceipt(new HeaderReceipt(new BasicReceipt()));

String returnString = headerReceipt.printReceipt();

// Create purchased products section of receipt

returnString += "\nDetails of your Computer System purchase:\n";

ArrayList<Component> computerComponents = *computerSystem*.getComponents();

for(Component component : computerComponents) {

returnString += component.getTypeOfComponent() + "\t\u20ac" + component.getPrice() + "\t" + component.getComponentName() + "\n";

}

double totalCost = *computerSystem*.getPrice();

returnString += "\nTotal Cost:\t\u20ac" + new DecimalFormat("##.##").format(totalCost);

// Create footing section of receipt

Receipt shippingReceipt = new FooterReceipt(new BasicReceipt());

returnString += shippingReceipt.printReceipt();

return returnString;

}

}

BusinessLayer.DecoratorRecipt – FooterReceipt.java

package BusinessLayer.DecoratorReceipt;

public class FooterReceipt extends ReceiptDecorator {

public FooterReceipt(Receipt receipt) {

super(receipt);

}

@Override

public String printReceipt() {

return super.printReceipt() + " Please come back again! ";

}

}

BusinessLayer.DecoratorRecipt – HeaderReceipt.java

package BusinessLayer.DecoratorReceipt;

public class HeaderReceipt extends ReceiptDecorator {

public HeaderReceipt(Receipt receipt) {

super(receipt);

}

@Override

public String printReceipt() {

return super.printReceipt() + " The Computer Shop ";

}

}

BusinessLayer.DecoratorRecipt –Receipt.java

package BusinessLayer.DecoratorReceipt;

public abstract class Receipt {

public abstract String printReceipt();

}

BusinessLayer.DecoratorRecipt –ReceiptDecorator.java

package BusinessLayer.DecoratorReceipt;

public abstract class ReceiptDecorator extends Receipt {

private Receipt receipt;

public ReceiptDecorator(Receipt receipt) {

this.receipt = receipt;

}

@Override

public String printReceipt() {

if (receipt != null)

return receipt.printReceipt();

return null;

}

}

BusinessLayer.DecoratorRecipt –ThankYouReceipt.java

package BusinessLayer.DecoratorReceipt;

public class ThankYouReceipt extends ReceiptDecorator {

public ThankYouReceipt(Receipt receipt) {

super(receipt);

}

@Override

public String printReceipt() {

return super.printReceipt() + "\n\nThank you for shopping with us ";

}

}

BusinessLayer.FactoryProduct –FactoryDesignPattern.java

package BusinessLayer.FactoryProduct;

import BusinessLayer.CompositeProduct.\*;

public class FactoryDesignPattern {

public CPU getCPU(int componentId, String componentName, double price, String typeOfComponent, String series, String CPUSocketType, double weight){

return new CPU(componentId, componentName, price, typeOfComponent, series, CPUSocketType, weight);

}

public GPU getGPU(int componentId, String componentName, double price, String typeOfComponent, String memory, int displayPorts, double weight){

return new GPU(componentId, componentName, price, typeOfComponent, memory, displayPorts, weight);

}

public Keyboard getKeyboard(int componentId, String componentName, double price, String typeOfComponent, String keySwitchType, boolean backlit, double weight){

return new Keyboard(componentId, componentName, price, typeOfComponent, keySwitchType, backlit, weight);

}

public MemoryDrive getMemoryDrive(int componentId, String componentName, double price, String typeOfComponent, int driveCapacity, boolean SSD, double weight){

return new MemoryDrive(componentId, componentName, price, typeOfComponent, driveCapacity, SSD, weight);

}

public Monitor getMonitor(int componentId, String componentName, double price, String typeOfComponent, String screenResolution, boolean curved, boolean is3d, double weight){

return new Monitor(componentId, componentName, price, typeOfComponent, screenResolution, curved, is3d, weight);

}

public Motherboard getMotherboard(int componentId, String componentName, double price, String typeOfComponent, String cpuSocketType, String series, String memoryStandard, double weight){

return new Motherboard(componentId, componentName, price, typeOfComponent, cpuSocketType, series, memoryStandard, weight);

}

public Mouse getMouse(int componentId, String componentName, double price, String typeOfComponent, int dpi, boolean programmableButtons, boolean dpiSwitching, double weight){

return new Mouse(componentId, componentName, price, typeOfComponent, dpi, programmableButtons, dpiSwitching, weight);

}

public RAM getRAM(int componentId, String componentName, double price, String typeOfComponent, int gigaBytes, String multiChannel, double weight){

return new RAM(componentId, componentName, price, typeOfComponent, gigaBytes, multiChannel, weight);

}

public ComputerSystem getComputerSystem(int componentId, String componentName, String typeOfComponent, String OS, double weight){

return new ComputerSystem(componentId, componentName, typeOfComponent, OS, weight);

}

}

BusinessLayer.MementoPattern –Caretaker.java

package BusinessLayer.MementoPattern;

import java.util.\*;

public class Caretaker {

ArrayList<Memento> computerList = new ArrayList<>();

public void addMemento(Memento m) {

computerList.add(m);

}

public Memento getMemento(int index) {

return computerList.get(index);

}

public int getLength() {

return computerList.size();

}

}

BusinessLayer.MementoPattern –Memento.java

package BusinessLayer.MementoPattern;

import BusinessLayer.CompositeProduct.\*;

public class Memento {

private ComputerSystem computerSystem;

public Memento(ComputerSystem computerSystem) {

this.computerSystem = computerSystem;

}

public ComputerSystem getComputerSystem() {

return computerSystem;

}

}

BusinessLayer.MementoPattern –Originator.java

package BusinessLayer.MementoPattern;

import BusinessLayer.CompositeProduct.\*;

public class Originator {

private ComputerSystem computerSystem;

public void set(ComputerSystem computerSystem) {

this.computerSystem = computerSystem;

}

public Memento storeInMemento() {

return new Memento(computerSystem);

}

public ComputerSystem restoreFromMemento(Memento memento) {

computerSystem = memento.getComputerSystem();

return computerSystem;

}

}

BusinessLayer.ObserverProduct –AdminProductDisplay.java

package BusinessLayer.ObserverProduct;

import java.io.FileNotFoundException;

import java.util.Scanner;

import java.util.logging.\*;

import DataLayer.\*;

public class AdminProductDisplay implements Observer {

private String allDisplay;

private static final Logger *LOGGER* = Logger.*getLogger*( AdminProductDisplay.class.getName() );

public AdminProductDisplay() throws FileNotFoundException{

allDisplay = "\n\nPlease choose a component for your computer:\n";

for (int i = 0; i < DataControl.*factoryDesignPattern*().size(); i++) {

allDisplay += "\n\nChoice: " + (i + 1) + "\t" + " " + DataControl.*factoryDesignPattern*().get(i).getComponentDetails();

}

}

public String getAllDisplay() {

return allDisplay;

}

@Override

public void update() {

try {

allDisplay = "\n\nPlease choose a component for your computer:\n";

for (int i = 0; i < DataControl.*factoryDesignPattern*().size(); i++) {

allDisplay += "\n\nChoice: " + (i + 1) + "\t" + " " + DataControl.*factoryDesignPattern*().get(i).getComponentDetails();

}

} catch (Exception e) {

*LOGGER*.log(Level.*SEVERE*, "context", e);

}

}

public String addComponent(String type) throws FileNotFoundException{

String result = "String componentName, double price, String typeOfComponent, double weight";

switch(type){

case "CPU":

result += ", String series, String CPUSocketType";

break;

case "GPU":

result += ", String memory, int displayPorts";

break;

case "Keyboard":

result += ", String keySwitchType , booelan backlit";

break;

case "MemoryDrive":

result += ", int driveCapacity, boolean SSD";

break;

case "Monitor":

result += ", String screenResolution, boolean curved, boolean is3d";

break;

case "Motherboard":

result += ", String cpuSocketType, String series, String memoryStandard";

break;

case "Mouse":

result += ", int dpi, boolean programmableButtons, boolean dpiSwitching";

break;

case "RAM":

result += ", int gigaBytes, String multiChannel";

break;

default:

result = "Invalid Input";

break;

}

return result;

}

public int readUserInput() {

boolean acceptableInput = false;

int checkedUserChoice;

String uncheckedUserChoice = "";

Scanner in = new Scanner(System.*in*);

while (!acceptableInput) {

System.*out*.println("\nEnter choice number: ");

uncheckedUserChoice = in.nextLine();

if (uncheckedUserChoice.matches("[0-9]+"))

acceptableInput = true;

else

System.*out*.println("Error: Invalid. Must be a number");

}

checkedUserChoice = Integer.*parseInt*(uncheckedUserChoice);

return checkedUserChoice;

}

}

BusinessLayer.ObserverProduct –AdminProductList.java

package BusinessLayer.ObserverProduct;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.util.ArrayList;

import java.util.logging.Level;

import java.util.logging.Logger;

import ArchitecturalLayer.ContextObject;

import ArchitecturalLayer.Dispatcher;

import BusinessLayer.CompositeProduct.\*;

import DataLayer.DataControl;

public class AdminProductList implements Subject {

private ArrayList<Component> listOfProducts;

private ArrayList<Observer> observers;

private Dispatcher dispatcher;

private static final Logger *LOGGER* = Logger.*getLogger*( AdminProductList.class.getName() );

public AdminProductList() throws FileNotFoundException{

listOfProducts = new ArrayList<>();

observers = new ArrayList<>();

listOfProducts = DataControl.*factoryDesignPattern*();

dispatcher = new Dispatcher();

}

public void removeComponent(int choice) throws IOException{

ContextObject context = new ContextObject("remove" , listOfProducts.get(choice - 1));

dispatcher.iterate\_list(context);

DataControl.removeStockFromFile(listOfProducts.get(choice-1).getComponentName());

listOfProducts.remove(choice - 1);

DataControl.*rewriteComponentFile*(listOfProducts);

notifyObservers();

}

public void addToFile(String details, int stock) throws FileNotFoundException{

DataControl.*writeNewComponentToFile*(details, stock);

listOfProducts = DataControl.*factoryDesignPattern*();

ContextObject c = new ContextObject("add" , listOfProducts.get(listOfProducts.size()-1));

dispatcher.iterate\_list(c);

notifyObservers();

}

@Override

public void registerObserver(Observer O) {

observers.add(O);

}

@Override

public void notifyObservers() {

Observer o;

try {

for(int i = 0; i < observers.size();i++ )

{

o = observers.get(i);

o.update();

}

} catch (Exception e) {

*LOGGER*.log(Level.*SEVERE*, "context", e);

}

}

}

BusinessLayer.ObserverProduct –Observer.java

package BusinessLayer.ObserverProduct;

@FunctionalInterface

public interface Observer {

public void update();

}

BusinessLayer.ObserverProduct –Subject.java

package BusinessLayer.ObserverProduct;

public interface Subject {

public void registerObserver(Observer O);

public void notifyObservers();

}

BusinessLayer.VisitorShipping –ShippingVisitor.java

package BusinessLayer.VisitorShipping;

import BusinessLayer.CompositeProduct.\*;

public class ShippingVisitor implements Visitor {

private double totalShippingForCart;

public double getTotalShipping() {

return totalShippingForCart;

}

@Override

public void visit(CPU cpu) {

if(cpu.getPrice() < 100.0) {

totalShippingForCart += cpu.getWeight() \* 2;

}

}

@Override

public void visit(GPU gpu) {

if(gpu.getPrice() < 100.0) {

totalShippingForCart += gpu.getWeight() \* 3;

}

}

@Override

public void visit(Keyboard keyboard) {

if(keyboard.getPrice() < 100.0) {

totalShippingForCart += keyboard.getWeight() \* 2;

}

}

@Override

public void visit(MemoryDrive memoryDrive) {

if(memoryDrive.getPrice() < 100.0) {

totalShippingForCart += memoryDrive.getWeight() \* 2;

}

}

@Override

public void visit(Monitor monitor) {

if(monitor.getPrice() < 100.0) {

totalShippingForCart += monitor.getWeight() \* 2;

}

}

@Override

public void visit(Motherboard motherboard) {

if(motherboard.getPrice() < 100.0) {

totalShippingForCart += motherboard.getWeight() \* 2;

}

}

@Override

public void visit(Mouse mouse) {

if(mouse.getPrice() < 100.0) {

totalShippingForCart += mouse.getWeight() \* 2;

}

}

@Override

public void visit(RAM ram) {

if(ram.getPrice() < 100.0) {

totalShippingForCart += ram.getWeight() \* 2;

}

}

}

BusinessLayer.VisitorShipping –VisitableElement.java

package BusinessLayer.VisitorShipping;

@FunctionalInterface

public interface VisitableElement {

public void accept(Visitor visitor);

}

BusinessLayer.VisitorShipping –Visitor.java

package BusinessLayer.VisitorShipping;

import BusinessLayer.CompositeProduct.\*;

public interface Visitor {

public void visit(CPU cpu);

public void visit(GPU gpu);

public void visit(Keyboard keyboard);

public void visit(MemoryDrive memoryDrive);

public void visit(Monitor monitor);

public void visit(Motherboard motherboard);

public void visit(Mouse mouse);

public void visit(RAM ram);

}

DataLayer.VisitorShipping –DataControl.java

package DataLayer;

import java.util.ArrayList;

import java.util.logging.Level;

import java.util.logging.Logger;

import BusinessLayer.StockManager;

import BusinessLayer.CompositeProduct.\*;

import BusinessLayer.FactoryProduct.\*;

import BusinessLayer.ObserverProduct.AdminProductList;

import java.io.\*;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.nio.file.StandardOpenOption;

import java.util.\*;

public class DataControl {

public static final String *componentFileName* = "ComponentList.txt";

public static final String *stockFileName* = "StockList.txt";

private static final Logger *LOGGER* = Logger.*getLogger*( AdminProductList.class.getName() );

private DataControl() {

//Adding private constructor to avoid implicit public one

}

public static ArrayList<String> getUniqueComponentTypes() throws FileNotFoundException {

File componentFile = new File(*componentFileName*);

FactoryDesignPattern component = new FactoryDesignPattern();

ArrayList<Component> componentList = new ArrayList<>();

Scanner reader = new Scanner(componentFile);

while(reader.hasNextLine()){

String LineFromFile = reader.nextLine();

String[] pieceOfLine = LineFromFile.split(",");

switch(pieceOfLine[4]){

case "CPU":

CPU cpuComponent = component.getCPU(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], pieceOfLine[4],

pieceOfLine[5], Double.*parseDouble*(pieceOfLine[6]));

componentList.add(cpuComponent);

break;

case "GPU":

GPU gpuComponent = component.getGPU(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], pieceOfLine[4],

Integer.*parseInt*(pieceOfLine[5]), Double.*parseDouble*(pieceOfLine[6]));

componentList.add(gpuComponent);

break;

case "Keyboard":

Keyboard keyboardComponent = component.getKeyboard(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], pieceOfLine[4],

Boolean.*parseBoolean*(pieceOfLine[5]), Double.*parseDouble*(pieceOfLine[6]));

componentList.add(keyboardComponent);

break;

case "MemoryDrive":

MemoryDrive memoryComponent = component.getMemoryDrive(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], Integer.*parseInt*(pieceOfLine[4]),

Boolean.*parseBoolean*(pieceOfLine[5]), Double.*parseDouble*(pieceOfLine[6]));

componentList.add(memoryComponent);

break;

case "Monitor":

Monitor monitorComponent = component.getMonitor(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], pieceOfLine[4],

Boolean.*parseBoolean*(pieceOfLine[5]), Boolean.*parseBoolean*(pieceOfLine[6]), Double.*parseDouble*(pieceOfLine[7]));

componentList.add(monitorComponent);

break;

case "Motherboard":

Motherboard motherboardComponent = component.getMotherboard(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], pieceOfLine[4],

pieceOfLine[5], pieceOfLine[6], Double.*parseDouble*(pieceOfLine[7]));

componentList.add(motherboardComponent);

break;

case "Mouse":

Mouse mouseComponent = component.getMouse(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], Integer.*parseInt*(pieceOfLine[4]),

Boolean.*parseBoolean*(pieceOfLine[5]), Boolean.*parseBoolean*(pieceOfLine[6]), Double.*parseDouble*(pieceOfLine[7]));

componentList.add(mouseComponent);

break;

case "RAM":

RAM ramComponent = component.getRAM(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], Integer.*parseInt*(pieceOfLine[4]),

pieceOfLine[5], Double.*parseDouble*(pieceOfLine[6]));

componentList.add(ramComponent);

break;

default:

System.*out*.println("Invalid Component type. DataControl.java -> Method: getUniqueComponentTypes()");

break;

}

}

reader.close();

ArrayList<String> uniqueComponentArrayList = new ArrayList<>();

for (int i = 0; i < componentList.size(); i++) {

if (!(uniqueComponentArrayList.contains(componentList.get(i).getTypeOfComponent())))

uniqueComponentArrayList.add(componentList.get(i).getTypeOfComponent());

}

return uniqueComponentArrayList;

}

public static int getStockByComponentName(String compName) throws FileNotFoundException {

File componentFile = new File(*stockFileName*);

int amountOfItemsInStock = 0;

Scanner reader = new Scanner(componentFile);

while(reader.hasNextLine()){

String LineFromFile = reader.nextLine();

String[] pieceOfLine = LineFromFile.split(",");

if(compName.matches(pieceOfLine[0])){

amountOfItemsInStock = Integer.*parseInt*(pieceOfLine[1]);

}

}

reader.close();

return amountOfItemsInStock;

}

public static ArrayList<Component> getComponentTypeList(String componentType) throws FileNotFoundException {

//Filled with factory stuff

//Method will take in param of component type e.g. CPU

//Factory stuff will search txt files for products matching CPU

//Create ArrayList of Component type filled with all CPUs

//Return ArrayList

File componentFile = new File(*componentFileName*);

FactoryDesignPattern component = new FactoryDesignPattern();

ArrayList<Component> componentList = new ArrayList<>();

Scanner reader = new Scanner(componentFile);

while(reader.hasNextLine()){

String LineFromFile = reader.nextLine();

String[] pieceOfLine = LineFromFile.split(",");

if (StockManager.*checkIsComponentInStock*(pieceOfLine[1]) && componentType.equals(pieceOfLine[3])) {

switch(componentType){

case "CPU":

CPU cpuComponent = component.getCPU(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], pieceOfLine[5],

pieceOfLine[6], Double.*parseDouble*(pieceOfLine[4]));

componentList.add(cpuComponent);

break;

case "GPU":

GPU gpuComponent = component.getGPU(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], pieceOfLine[5],

Integer.*parseInt*(pieceOfLine[6]), Double.*parseDouble*(pieceOfLine[4]));

componentList.add(gpuComponent);

break;

case "Keyboard":

Keyboard keyboardComponent = component.getKeyboard(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], pieceOfLine[5],

Boolean.*parseBoolean*(pieceOfLine[6]), Double.*parseDouble*(pieceOfLine[4]));

componentList.add(keyboardComponent);

break;

case "MemoryDrive":

MemoryDrive memoryComponent = component.getMemoryDrive(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], Integer.*parseInt*(pieceOfLine[5]),

Boolean.*parseBoolean*(pieceOfLine[6]), Double.*parseDouble*(pieceOfLine[4]));

componentList.add(memoryComponent);

break;

case "Monitor":

Monitor monitorComponent = component.getMonitor(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], pieceOfLine[5],

Boolean.*parseBoolean*(pieceOfLine[6]), Boolean.*parseBoolean*(pieceOfLine[7]), Double.*parseDouble*(pieceOfLine[4]));

componentList.add(monitorComponent);

break;

case "Motherboard":

Motherboard motherboardComponent = component.getMotherboard(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], pieceOfLine[5],

pieceOfLine[6], pieceOfLine[7], Double.*parseDouble*(pieceOfLine[4]));

componentList.add(motherboardComponent);

break;

case "Mouse":

Mouse mouseComponent = component.getMouse(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], Integer.*parseInt*(pieceOfLine[5]),

Boolean.*parseBoolean*(pieceOfLine[6]), Boolean.*parseBoolean*(pieceOfLine[7]), Double.*parseDouble*(pieceOfLine[4]));

componentList.add(mouseComponent);

break;

case "RAM":

RAM ramComponent = component.getRAM(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], Integer.*parseInt*(pieceOfLine[5]),

pieceOfLine[6], Double.*parseDouble*(pieceOfLine[4]));

componentList.add(ramComponent);

break;

default:

System.*out*.println("Invalid Component type. DataControl.java -> Method: getComponentTypeList()");

break;

}

}

}

reader.close();

return componentList;

}

public static void writeNewComponentToFile(String details, int stock) throws FileNotFoundException {

int nextID = *checkNextAvailableId*(*componentFileName*);

String lineToAppendToComponentFile = nextID + ",";

String [] detailsSplit = details.split(",");

for(int i=0;i < detailsSplit.length;i++){

lineToAppendToComponentFile += detailsSplit[i].trim() + ",";

}

String lineToAppendToStockFile = detailsSplit[0] + "," + stock + "\n";

lineToAppendToComponentFile += "\n";

try {

Files.*write*(Paths.*get*(*componentFileName*), lineToAppendToComponentFile.getBytes(), StandardOpenOption.*APPEND*);

}

catch (Exception e) {

*LOGGER*.log(Level.*SEVERE*, "context", e);

}

try {

Files.*write*(Paths.*get*(*stockFileName*), lineToAppendToStockFile.getBytes(), StandardOpenOption.*APPEND*);

}

catch (Exception e) {

*LOGGER*.log(Level.*SEVERE*, "context", e);

}

}

public static int checkNextAvailableId(String textFileName) throws FileNotFoundException {

int nextAvailableId = 0;

File searchTextFile = new File(textFileName);

Scanner lineIn = new Scanner(searchTextFile);

while (lineIn.hasNextLine()) {

String aLineFromFile = lineIn.nextLine();

String [] splitLineFromFile = aLineFromFile.split(",");

if (Integer.*parseInt*(splitLineFromFile[0]) > nextAvailableId)

nextAvailableId = Integer.*parseInt*(splitLineFromFile[0]);

}

lineIn.close();

nextAvailableId++;

return nextAvailableId;

}

public static void editComponent(Part part) throws IOException {

File searchTextFile = new File(*componentFileName*);

Scanner lineIn = new Scanner(searchTextFile);

ArrayList<String[]> fileList = new ArrayList<>();

while (lineIn.hasNextLine()) {

String aLineFromFile = lineIn.nextLine();

String [] splitLineFromFile = aLineFromFile.split(",");

fileList.add(splitLineFromFile);

}

FileWriter writer = new FileWriter(*componentFileName*);

for(int k = 0;k < fileList.size();k++) {

for(int u = 0;u < fileList.get(k).length;u++) {

writer.write(fileList.get(k)[u]);

if (u != fileList.get(k).length) {

writer.write(",");

}

}

writer.write("\n");

}

lineIn.close();

writer.close();

}

public static void rewriteComponentFile(ArrayList<Component> components) throws IOException{

FileWriter writer = new FileWriter(*componentFileName*);

PrintWriter out = new PrintWriter(writer);

Component c;

for(int i =0; i < components.size();i++){

System.*out*.println("[info] : ------ Rewriting File (DataControl) ------");

c= components.get(i);

out.println(c.toString());

}

out.close();

}

public static ArrayList<Component> factoryDesignPattern() throws FileNotFoundException{

File componentFile = new File(*componentFileName*);

FactoryDesignPattern component = new FactoryDesignPattern();

ArrayList<Component> componentList = new ArrayList<>();

Scanner reader = new Scanner(componentFile);

while(reader.hasNextLine()){

String LineFromFile = reader.nextLine();

String[] pieceOfLine = LineFromFile.split(",");

switch(pieceOfLine[3]){

case "CPU":

CPU cpuComponent = component.getCPU(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], pieceOfLine[5],

pieceOfLine[6], Double.*parseDouble*(pieceOfLine[4]));

componentList.add(cpuComponent);

break;

case "GPU":

GPU gpuComponent = component.getGPU(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], pieceOfLine[5],

Integer.*parseInt*(pieceOfLine[6]), Double.*parseDouble*(pieceOfLine[4]));

componentList.add(gpuComponent);

break;

case "Keyboard":

Keyboard keyboardComponent = component.getKeyboard(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], pieceOfLine[5],

Boolean.*parseBoolean*(pieceOfLine[6]), Double.*parseDouble*(pieceOfLine[4]));

componentList.add(keyboardComponent);

break;

case "MemoryDrive":

MemoryDrive memoryComponent = component.getMemoryDrive(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], Integer.*parseInt*(pieceOfLine[5]),

Boolean.*parseBoolean*(pieceOfLine[6]), Double.*parseDouble*(pieceOfLine[4]));

componentList.add(memoryComponent);

break;

case "Monitor":

Monitor monitorComponent = component.getMonitor(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], pieceOfLine[5],

Boolean.*parseBoolean*(pieceOfLine[6]), Boolean.*parseBoolean*(pieceOfLine[7]), Double.*parseDouble*(pieceOfLine[4]));

componentList.add(monitorComponent);

break;

case "Motherboard":

Motherboard motherboardComponent = component.getMotherboard(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], pieceOfLine[5],

pieceOfLine[6], pieceOfLine[7], Double.*parseDouble*(pieceOfLine[4]));

componentList.add(motherboardComponent);

break;

case "Mouse":

Mouse mouseComponent = component.getMouse(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], Integer.*parseInt*(pieceOfLine[5]),

Boolean.*parseBoolean*(pieceOfLine[6]), Boolean.*parseBoolean*(pieceOfLine[7]), Double.*parseDouble*(pieceOfLine[4]));

componentList.add(mouseComponent);

break;

case "RAM":

RAM ramComponent = component.getRAM(Integer.*parseInt*(pieceOfLine[0]), pieceOfLine[1], Double.*parseDouble*(pieceOfLine[2]), pieceOfLine[3], Integer.*parseInt*(pieceOfLine[5]),

pieceOfLine[6], Double.*parseDouble*(pieceOfLine[4]));

componentList.add(ramComponent);

break;

default:

System.*out*.println("Invalid Component Type. DataControl.java -> Method: factoryDesignPattern()");

break;

}

}

reader.close();

return componentList;

}

public static void adjustStock(String componentName, String incOrDec) throws IOException {

File searchTextFile = new File(*stockFileName*);

//Construct the new file that will later be renamed to the original filename.

File tempFile = new File(searchTextFile.getAbsolutePath() + ".tmp");

BufferedReader br = new BufferedReader(new FileReader(searchTextFile));

PrintWriter pw = new PrintWriter(new FileWriter(tempFile));

String line = null;

//Read from the original file and write to the new

//unless content matches data to be removed.

while ((line = br.readLine()) != null) {

String aLineFromFile = line;

String [] splitLineFromFile = aLineFromFile.split(",");

if (!splitLineFromFile[0].equals(componentName)) {

pw.println(line);

pw.flush();

}

else {

int stock = Integer.*parseInt*(splitLineFromFile[1]);

if (incOrDec.matches("increment")){

stock++;

}

else if(incOrDec.matches("decrement")){

stock--;

}

pw.println(componentName + "," + stock + ",");

pw.flush();

}

}

pw.close();

br.close();

//Delete the original file

if (!searchTextFile.delete()) {

System.*out*.println("Could not delete file");

return;

}

//Rename the new file to the filename the original file had.

if (!tempFile.renameTo(searchTextFile))

System.*out*.println("Could not rename file");

}

public static void removeStockFromFile(String componentName) throws IOException{

File searchTextFile = new File(*stockFileName*);

//Construct the new file that will later be renamed to the original filename.

File tempFile = new File(searchTextFile.getAbsolutePath() + ".tmp");

BufferedReader br = new BufferedReader(new FileReader(searchTextFile));

PrintWriter pw = new PrintWriter(new FileWriter(tempFile));

String line = null;

//Read from the original file and write to the new

//unless content matches data to be removed.

while ((line = br.readLine()) != null) {

String aLineFromFile = line;

String [] splitLineFromFile = aLineFromFile.split(",");

if (!splitLineFromFile[0].equals(componentName)) {

pw.println(line);

pw.flush();

}

}

pw.close();

br.close();

//Delete the original file

if (!searchTextFile.delete()) {

System.*out*.println("Could not delete file");

return;

}

//Rename the new file to the filename the original file had.

if (!tempFile.renameTo(searchTextFile))

System.*out*.println("Could not rename file");

}

}

DataLayer.Interceptor –infoRequest.java

package Interceptor;

//ContextObject Interface

public interface infoRequest {

public String getType();

public int getComponentID();

}