DCM Computing (Team 9)

CS4125 System Analysis and Design

Conor Duggan	13127004
Can Prendergast	12127191
Daniel O'Connor	13141252
Darren Stack	13129783
Michael Lucas	13112732

CS4125: Systems Analysis and Design. Semester 2, 2015-2016

Guidance on the MARKING SCHEME for Team-Based Project: Version 1 (9th February 2016 - Week 3)

Name:	ID:
Name:	ID:
Name:	ID:
Name	ID:

	Item	Detailed Description	Marks Allocated		Marks Awarded
			Sub-total	Total	
	Presentation	• General Presentation • Adherence to guidelines i.e front cover sheet, blank marking scheme, table of contents		**1**	
3	Narrative	Narrative description of business scenario		**1**	
4	SLC	Discuss and justify SLC & risk mgt strategy?		**1**	
5	Project Plan	Plan specifying timeline, deliverables, and roles.		**1**	
6	Requirement	• Use case diagram(s) • Structured use case descriptions(s) • Non-functional (quality) attributes • Screen shots / report formats	1 2 2 1	**6**	
7	System Architecture	System architecture diagram with interfaces		**2**	
8	Analysis Sketches	• Method used to identify candidate classes • Analysis Class diagram with generalisation, composition, multiplicity, dialog, control, entity, interfaces, pre and post conditions, etc. • Interaction diagram • Entity relationship diagram with cardinality	1 2 1 1	**5**	
9	Code	• Compiles and runs • MVC • Design pattern(s) • Concurrency	2 2 2 2	**8**	
10	Design blueprints - based on code	• Architectural diagram • Design class diagram • State chart.	2 2 2	**6**	
11	More UML diagrams	Component and Deployment diagrams - based on the implementation.		**2**	
12		Description of patterns and approach to concurrency support.		**2**	
13	Critique	Evaluate the analysis & design artefacts.		**2**	
14	References			**1**	
15	Online Assessment	Week 6: Use cases Week 8: Class diagram	1 1	**2**	
16	Interview Week 13 (Pass/Fail basis)			**P/F**	
	Sub-total (A)			**40**	

PENALTIES

	Description	TM1	TM2	TM3	TM4
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)				

FINAL MARKS AWARDED

	(A-B)	TM1	TM2	TM3	TM4

Table of Contents

Narrative

Overview of the current market

The PC industry has taken a hit in recent years. Sales in PCs and PC parts have been falling each year since 2006. This is due to a lack of innovation. Most manufacturers see the PC industry as quick and easy money and they mass produce PCs and PC parts with a “one size fits all approach”. Each year manufacturers release new iterations of PCs and laptops which are indistinguishable from the previous model.

This lack of focus on specific user’s needs has caused the PC industry to become tired and boring. There is a lack of customisation in the current market and if someone wants to build a customer machine themselves, they must go to a lot of trouble in order to find the exact parts they need.

DCM Computing

DCM Computing is a manufacturer and online retailer for PC parts and pre made machines and laptops. We aim to provide a range of PC components ranging in performance. This enables users to pick and choose the parts they need in order to fulfil their needs. For example, if a customer is looking to build a PC or laptop for home use, such as browsing the web and emailing, we would suggest that the customer purchase some of our lower performing processors and lower resolution monitor, whether it be for a laptop or desktop PC. The option of choosing these lower performing components helps keeps the cost down for the customer and prevents them from paying for performance which they do not need.

Similarly, a customer may be looking for a higher performing PC, whether it be for PC gaming or for professional work with 3D modelling. In this case, we have higher performing components such as dedicated graphics cards and high resolution monitors. We pride ourselves on having a wide range of components in order to help our customers create their own PC which match their needs perfectly.

We also dedicate much of our time into research in order to push the boundaries of computer performance. As computers become more widely used in both our personal lives and professional lives, it is important to continue updating our products in order to provide the best possible performance for our customers.

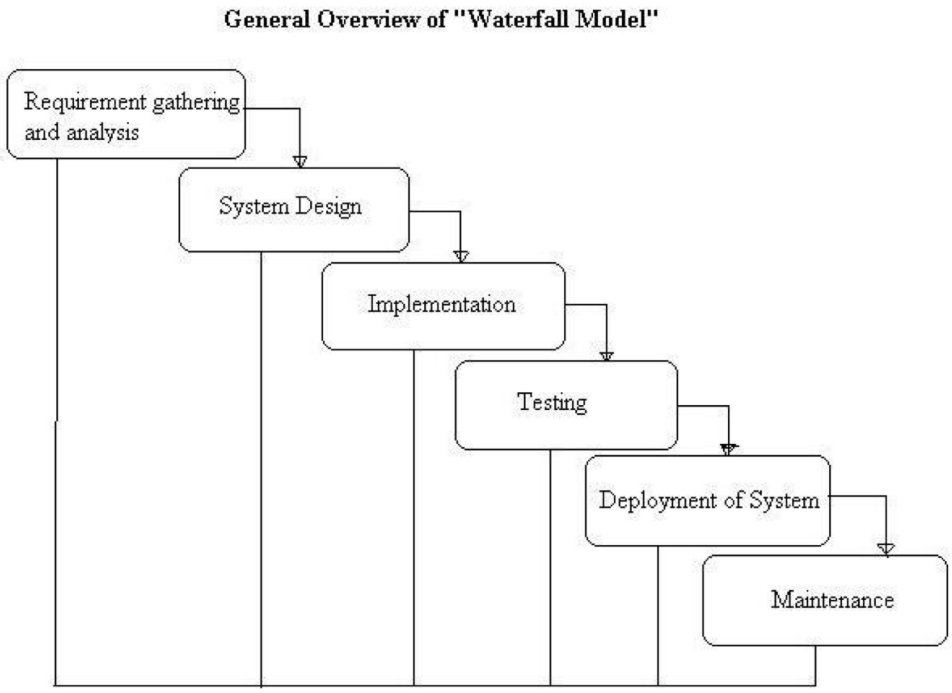
We also provide a range of pre built PCs and laptops with different levels of performance and price. Although our main aim is to allow customers to customise their own computer to their liking, the option of having pre built machines benefits using who have no prior experience building a custom PC or laptop. Each product in our pre built category comes with a list of components that make up the machine, along with a description which explains exactly what the computer is capable of.

Software Life Cycle Model

Waterfall Lifecycle

When further investigating the Waterfall model, we noticed a few flaws in the idea which wouldn't be appropriate for our project. They included:

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage of the project.
- Not a good model for complex and object-oriented projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.
- Poor model for long and ongoing projects

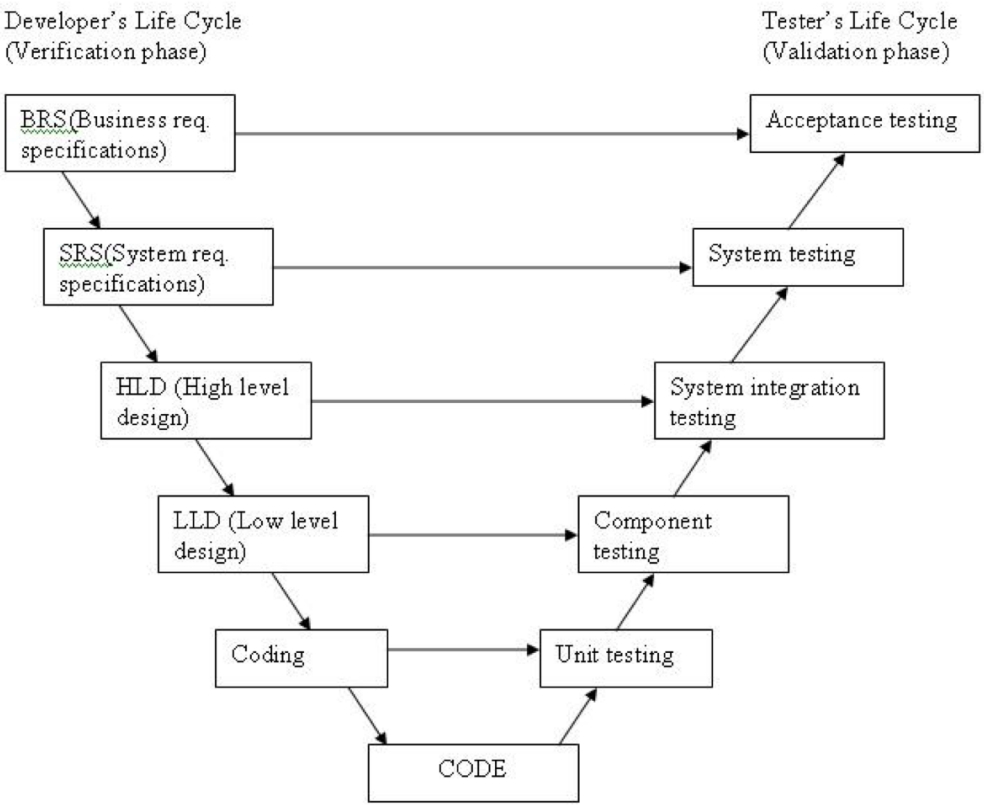


V-Model Lifecycle

We also decided to take a look into the V-Model as well but due to the following problems listed below, we decided not to take this method either.

- Very rigid and least flexible.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.

- If any changes happen midway, then the test documents along with requirement documents has to be updated.



Agile Development Value Proposition

We chose the agile development process for this project as it is the most flexible of all the software development choices.

While making the decision we looked at more rigid processes such as the “waterfall model” and the “v-model” but decided that the potential for the project would be much too high. We would prefer a short time between initial project time investment and either failing fast or knowing that an approach will work - So we quickly established our needs as a development team:

- Working software to be delivered frequently.
- Close, daily cooperation between developers.
- Continuous attention to technical excellence and good design.
- Regular adaptation to changing circumstances.
- Even late changes in requirements need to be implemented

It was clear that the agile development process was best suited to our needs. Always having a working product, starting with the very first sprint, so that it was harder for the project to go wrong. It seemed to fit as well because a typical scrum team is usually between 5 and 9 people – and as we are a team of 5 that suits us perfectly.

Instead of big specs, we'd discuss requirements together in meetings. Instead of wondering what anyone is doing in our project, we'd collaborate around a task-board (we used a free website called "Trello") discussing progress. Instead of long project plans, we'd discuss what's right for the project as we go and the team is empowered to make decisions. In turn this helps to create highly motivated, high performance teams that are highly cooperative.

A key principle of agile development is that testing is integrated throughout the lifecycle, enabling regular inspection of the working product as it develops. This this will allow us to make adjustments if necessary and gives the product team early sight of any quality issues.

Unlike the waterfall model in agile model very limited planning is required to get started with the project. Agile assumes that the end users' needs are ever changing in a dynamic business and IT world. Changes can be discussed and features can be newly effected or removed based on feedback. This effectively gives the customer the finished system they want or need.

On agile projects, every member of the project team has the opportunity to know how the project is going at any given time. Daily scrum meetings and weekly sprint reviews offer concrete ways to see progress.

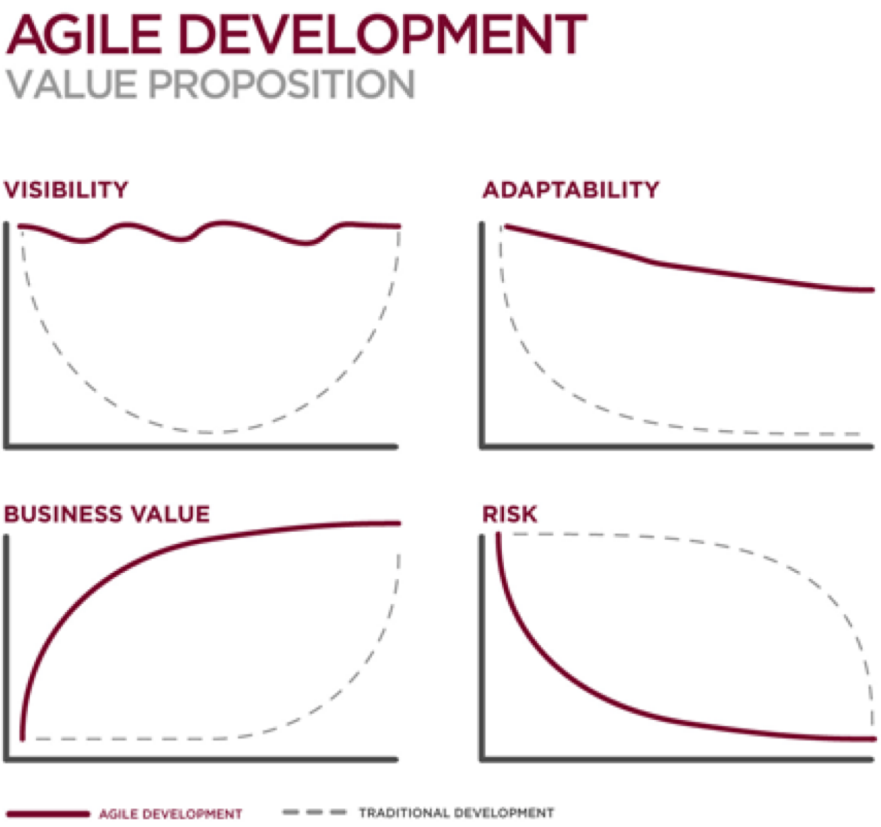
By delivering working, tested, deployable software on an incremental basis, agile development delivers increased value, visibility, and adaptability much earlier in the life cycle, significantly reducing project risk.

Agile vs. Waterfall

Waterfall challenges

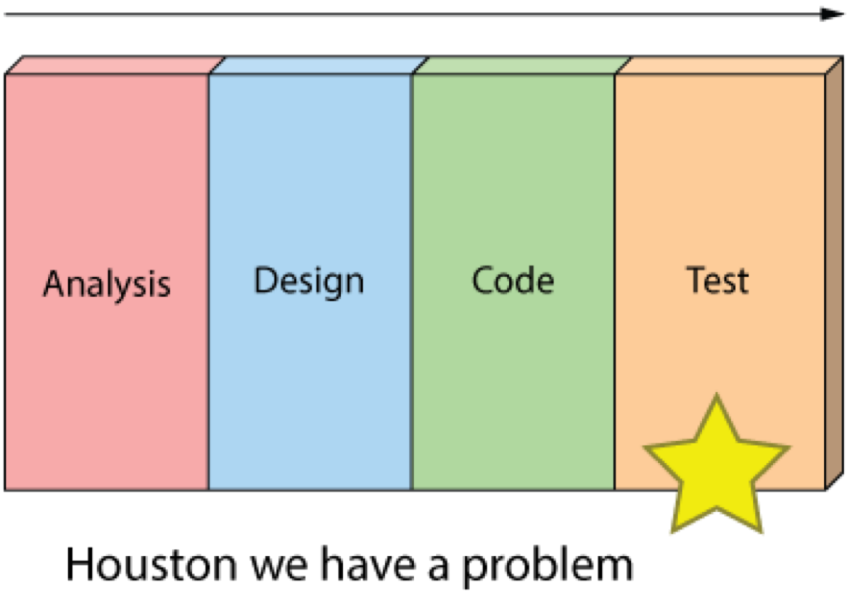
Poor quality

First off, when the project starts to run out of time, testing is the only phase left.



Poor visibility

Secondly, because working software isn't produced until the end of the project, you never really know where you are on a Waterfall project. That last 20% of the project always seems to take 80% of the time.



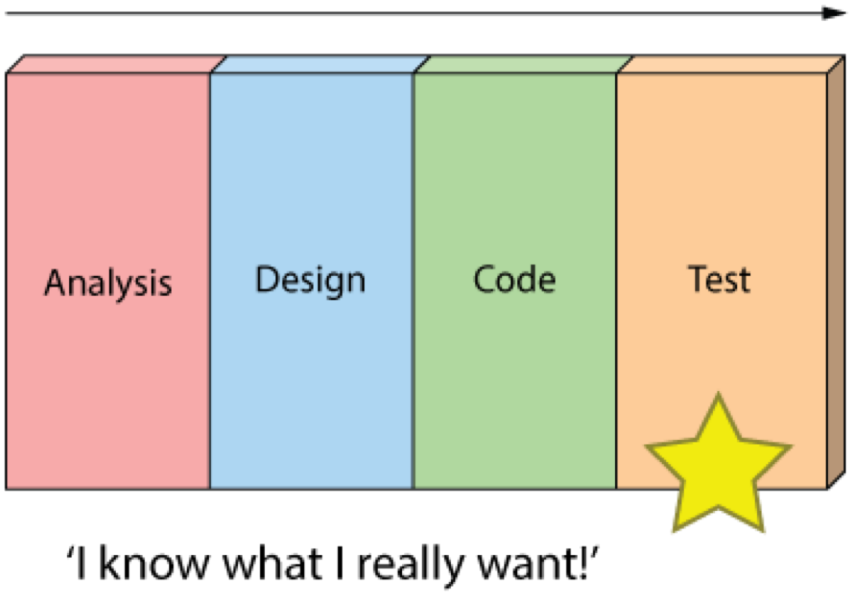
Too risky

Thirdly you've got schedule risk because you never know if you are going to make it until the end.

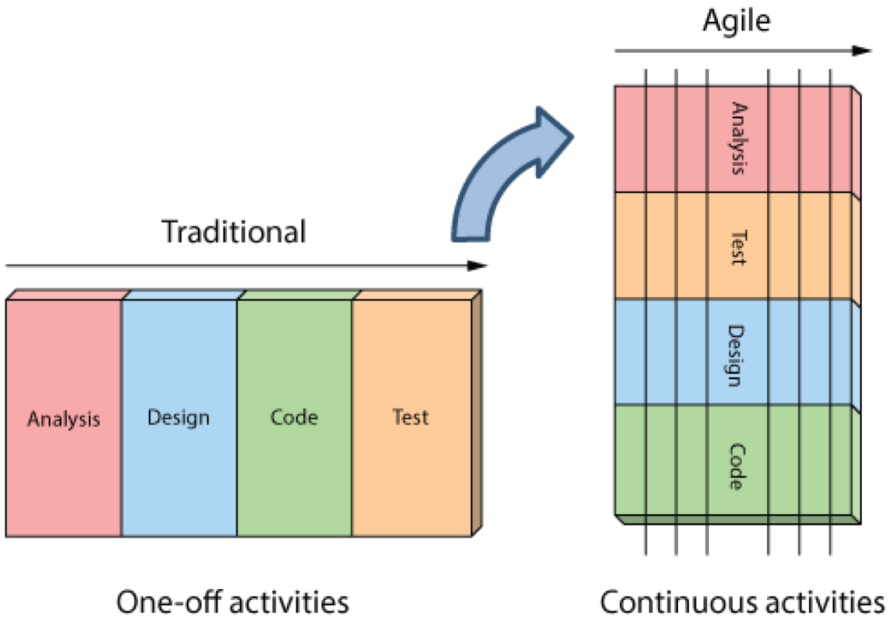
You've got technical risk because you don't actually get to test your design or architecture until late in the project.

And you've got product risk because don't even know if you are building the right until it's too late to make any changes.

And finally, most importantly, it's just not a great way for handling change.



The Agile Approach



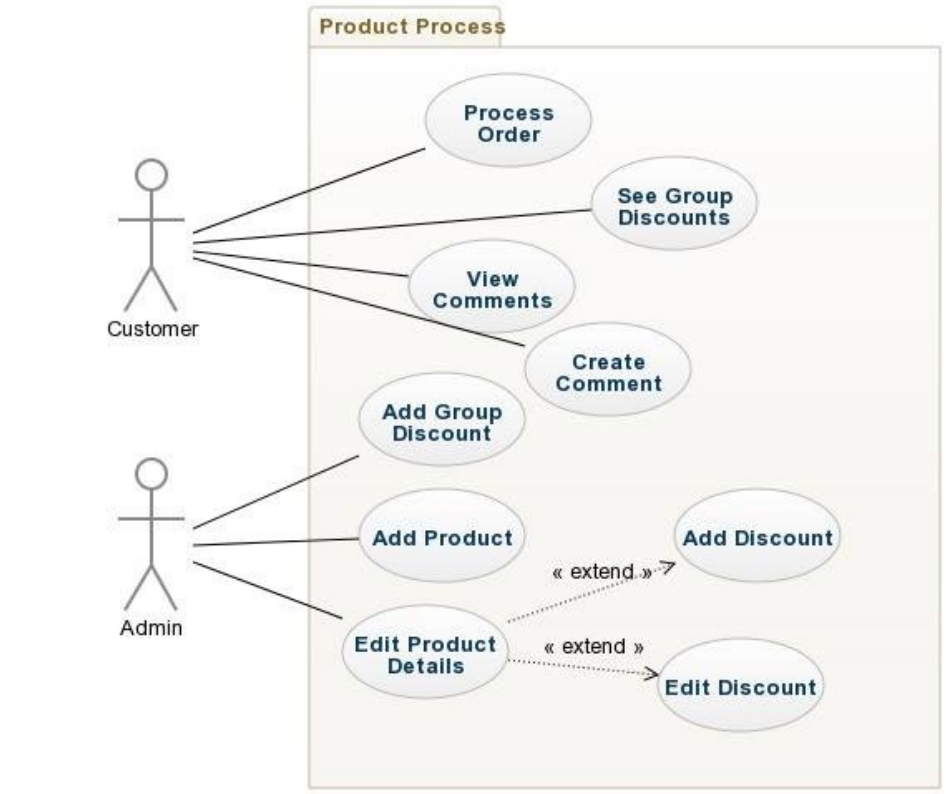
Project Plan

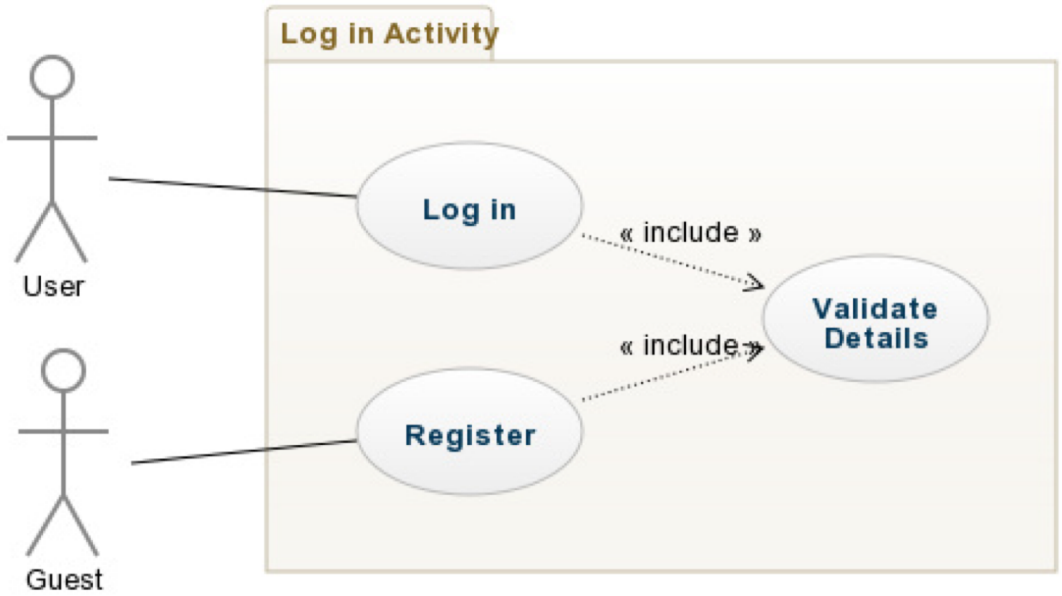
Job Name	Job Description	Allocated To	Week
Narrative	Narrative description of business scenarios	Cian	5
Presentation	Compony Logo/Design Cover Page	Conor	5
Software Life Cycle	Discussion of the Software Model Used	Michael & Daniel	5
Project Plan	Specifying Jobs and Roles	Conor	4
Requirements	Use Case Diagrams	Darren & Michael	6
	Use Case Descriptions	Darren & Michael	6
	Structured Use Case Description	Conor	6
	Non Functional Requirements	Darren	6
	Tactics for Handling Quality Attributes	Daniel	6
	Screenshots of GUI	Conor	12
System Architecture	Architecture Description and Diagram	Cian	8
Analysis Sketches	Identify Candidate Classes	Cian	6
	Class Diagram	Michael & Conor	7
	Interaction Diagrams	Cian & Darren	7
	Entity Relationship Diagram	Daniel	7
Code	Code Implementation	Group	8-12
Design	Architectural Diagram	Cian	12
	Design Class Diagram	Conor & Daniel	12
	State Chart	Conor & Darren	12
More UML Diagrams	Component Diagram	Michael & Cian	12
	Deployment Diagram	Michael & Darren	
Patterns	Description of Patterns	Group	12
Concurrency	Approach to Concurrency Support	Michael	12
Critique	Critique on Analysis and Design	Group	12
References	Sources used for learning/information	Group	4-12

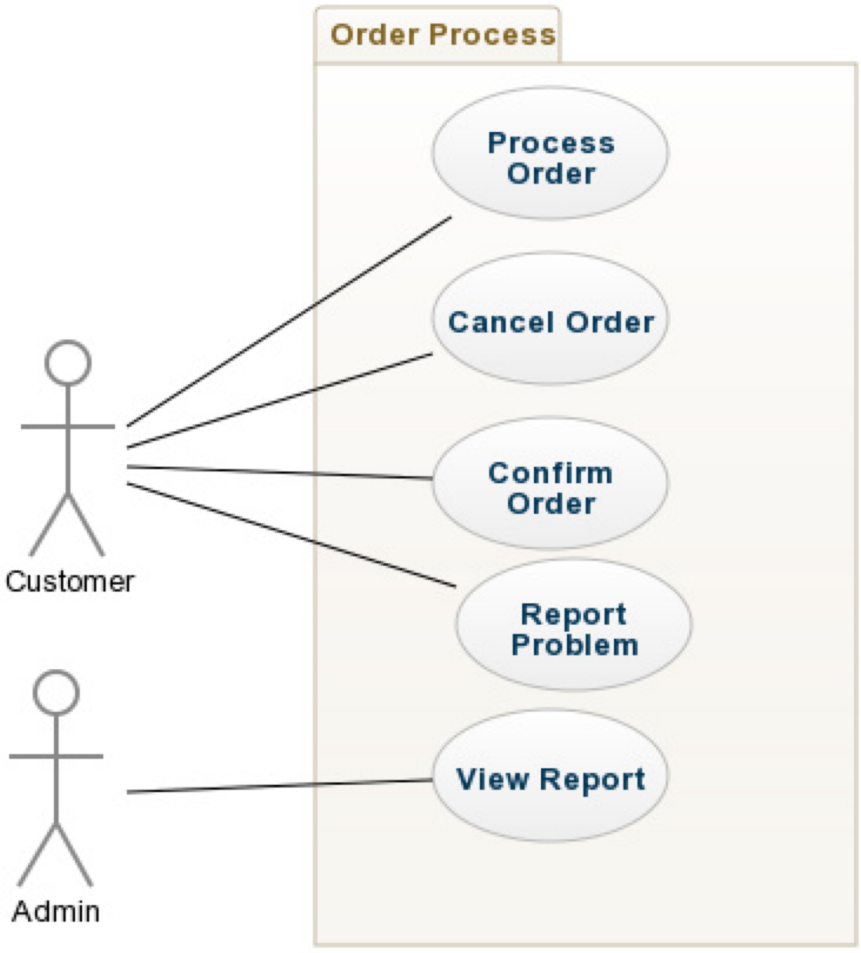
Requirements

Functional Requirements

Use Case Diagrams:






Order Process
Process Order
Cancel Order
Confirm Order
Report Problem
View Report
Customer
Admin

Structured Use Case Descriptions

Use Case 1	**Log In**
Actor Action	System Response
1. User enters a username and password and then clicks log in	2. The system will check if these details are valid or not. If they are valid they will be brought to a home screen.
	Alternative System Response
	The details are invalid, an error message will appear notifying the user that they have entered invalid details.

Non-Functional Requirement

Security: All user data should be encrypted with a hashed password so that their details cannot be accessed by malicious third parties.

Use Case 2	**Log Out**
Actor Action	System Response
1. User clicks button with the text 'Log Out'.	2. When clicked the user will be logged out of their account and the application ends.

Non-Functional Requirement

Usability: Users can be logged out with one click no matter what the page.

Security: There must be no trace of the user after they log out.

Use Case 4	**Create Comment**
Actor Action	System Response
1. Customer clicks *Add Comment* on a the ProductListUI.	2. A screen is displayed with a textbox.
3. Customer writes a comment on the product. The Customer then clicks *Submit*.	4. The comment is stored to list of comments for that product.

Non-Functional Requirement

Security: Users should only be able to make comments for bought products.

Use Case 5	**Create PC**
Actor Action	System Response
1. Customer clicks *Create PC*	2. A screen is displayed with fields for all the different parts of the PC (Monitor, Processor. RAM etc.)
3. The customer adds the product they want for each part of the PC. The customer clicks *Buy*	4. An order is placed for all the parts the customer wanted.

Non-Functional Requirement

Usability: Customer should easily be allowed to find the parts they are looking for.

Use Case 6	**Create Laptop**
Actor Action	System Response
1. Customer clicks *Create Laptop*	2. A screen is displayed with fields for all the different parts of the Laptop (CPU, Processor. RAM etc.)
3. The customer adds the product they want for each part of the Laptop. The customer clicks *Buy*	4. An order is placed for all the parts the customer wanted.

Non-Functional Requirement

Usability: Customer should easily be allowed to find the parts they are looking for.

Use Case 7	**Edit Credit Card Details**
Actor Action	System Response
1. Customer is submitting an order.	**2.** A screen is displayed with field asking for a new credit card number if they want to change the one associated with their account.
3. The Customer enters details and clicks *Proceed*.	**4.** Customer taken to Order Summary screen.
	Alternative System Response
	The credit card is invalid and the user is not allowed to proceed.

Non-Functional Requirement

Security: System should implement a defence to stop any chance of data being stolen.

Use Case 8	**Edit Address Details**
Actor Action	System Response
1. Customer is submitting an order.	**2.** A screen is displayed with field asking for a new address if they want to change the one associated with their account.
3. The Customer enters details and clicks *Proceed*.	**4.** Customer taken to Order Summary screen.

Non-Functional Requirement

Security: System should implement a defence to stop any chance of data being stolen.

Use Case 9	**Register as User**
Actor Action	System Response
1. User clicks into the register tab on the page	2. Once clicked, the system will redirect them into the registration page that will grant them access to become a Customer.
3. User will enter credentials that the system asks them and will click enter	4. The system will check the details that the user has provided and will send an email of confirmation to the users email address.
5. The user will open the email then and finally confirms that they are a user.	
	Alternative System Response
	4a. User does not receive an email of confirmation. (User did not provide proper email address)

Non-Functional Requirement

Security: Process of registration must be secured so that no personal information will be leaked onto another party. Problems like these can put the user at risk.

Use Case 10	**Add Product**
Actor Action	System Response
1. Admin clicks into *Add new Product*.	2. Displays page in regards to adding a new product.
3. Admin provides details of new product and then clicks ‘Finish’.	4. System is updated and saves newly added product on webpage

Non-Functional Requirement

Scalability: New items should be able to be added to increase the range of products available.

Use Case 11	**Edit Product Details**
Actor Action	System Response
1. Admin clicks into product and then clicks on ‘Edit’	2. Displays page where admin can change product information.
3. Admin edits information and clicks ‘Finish’	4. The system is updated and the new product information is now displayed.

Use Case 12	**Apply Discount**
Actor Action	System Response
1.Admin clicks into product and then clicks into *Add New Product Discount*	2. Displays page where the admin can edit product information
3. Admin ticks on the box that grants a discount, enters the value of discount and clicks ‘Finish’.	4. The system will update accordingly and display new prices with discounts applied accordingly.

Use Case 13	**Remove Discount**
Actor Action	System Response
1. Should the product already have a discount linked to it, go into it and click ‘Edit’	2. Displays page where the admin can edit product information
3. Admin then clicks the product and clicks ‘Finish’.	4. The system will update accordingly and display the correct price without discount applied.

Use Case 14	**Add Group Discount**
Actor Action	System Response
1. Admin clicks *Add new Group Discount*	2. Displays page where the admin can add the discount
3. Admin then selects their products and enters their appropriate discount.	4. The system will update accordingly and add the new Group Discount to the database.

Use Case 15	**See Group Discounts**
Actor Action	System Response
1. Customer clicks *View Group Discounts*	2. Displays page where all the Group Discounts are displayed.

Non-Functional Requirement

Performance: Should find all Group Discounts within 5 seconds.

Use Case 16	**View Comments**
Actor Action	System Response
1. Customer chooses a Product and clicks *View Comments.*	2. Displays page where all the Comments for selected products are displayed.

Non-Functional Requirement

Performance: Should find all Comments within 5 seconds.

NFR:

Usability

Maintainability

Scalability

Extensibility

Availability

Security

Portability

USE CASE 3	Process Order
Goal in Context	Customer requests purchase of item(s) from our company, if it is in stock we accept purchase, ship the item(s) and bill the customer.
Scope and Level	
Preconditions	We know the customer, the items they want to purchase, their address to ship to and the details of their purchase method.
Success End Conditions	The customer has received the item(s) they purchased and we have received the payment for them.
Failed End Conditions	The customer does not receive the item(s) they want to purchase and we do not receive payment.
Primary, Secondary, Actors	Customer, Company, Credit card company, bank, shipping service.
Trigger	The customer clicks *Purchase* button in their cart.
DESCRIPTION	1. Customer browses online store and picks out the item(s) that they want to purchase and add it to their cart. 2. Company's website checks to make sure the item(s) in the cart are in stock. 3. Customer clicks *Purchase* button in their cart. 4. Customer is asked to confirm the shipping address and credit card information associated with their account. 5. Company checks if it is a valid credit card number. 6. Company captures address and credit card information and an invoice is created with the cost of items and shipping. 7. Company creates the purchase order and ships the item(s) to the customer.
EXTENSIONS	5a. The customer may want to change the shipping address or credit card associated with their account. 5a1. Allow the customer to edit their details. 6a. Credit card details may be invalid of incorrect. 6a1. Have the customer review their credit card details and enter valid information. 8a. Customer returns the items(s). 8a1. Handle the returned goods and reimburse the customer.
VARIATIONS	1. Customer may want to complete the purchase over the phone. 5. Customer may want to pay using a debit card. 5. Customer may want to ship to an address that is not their own.

RELATED INFORMATION	Process Order
Priority	Top
Performance	< 10 minutes to proceed from cart to confirmation of purchase.
Frequency	Over 1000/day
Channel to Actors	Interactive, Database
OPEN ISSUES	
Due Date	Release 1.0
…any other management info…	
Superordinates	
Subordinates	Log In (Use Case 1)

Tactics for Handling Quality Attributes

Quality Assurance should be responsible for preventing defects, not merely for finding them. For this to happen, QA should be moved up to the front of the development cycle. Since we've chosen to use Github to keep our repository of code this could not be easier with the use of pull requests, versioning and branches! Another way to safeguard against application breaking bugs is by enforcing the ***efficient*** use of Unit, Integration and E2E(end to end) tests.

Pull Requests initiate discussion about your commits. Anyone can see exactly what changes would be merged if they accept your request. Each pull request should be reviewed by at least two other developers before the code can be successfully merged into the master branch. Once a Pull Request has been opened, the developer reviewing your changes may have questions or comments. Perhaps the coding style doesn't match the rest of the project, the change is missing unit or integration tests, or maybe everything looks great and kudos are in order. Pull Requests are designed to encourage and capture this type of conversation. Please go to https://github.com/SAP/BUILD/pull/51 to see an example of a good productive pull request between an architect and three senior devs discussing an infrastructure issue.

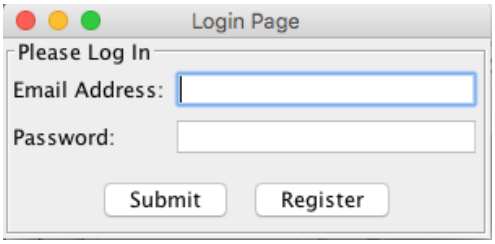
In the case of some small refactoring – a quick read through of the code might be sufficient but otherwise it's really easy to pull down the branch that the changes were made on and test it locally on your own machine without any need to bother the person who requested the PR.

With the use of versioning, it's easy to revert to an older version of a module. Perhaps a "Very High" or "Epic" bug is introduced to the system. All of the commits can be easily reverted to the latest version of the application that worked. Of course with the correct use of Unit, Integration and E2E tests this should be unlikely to happen.

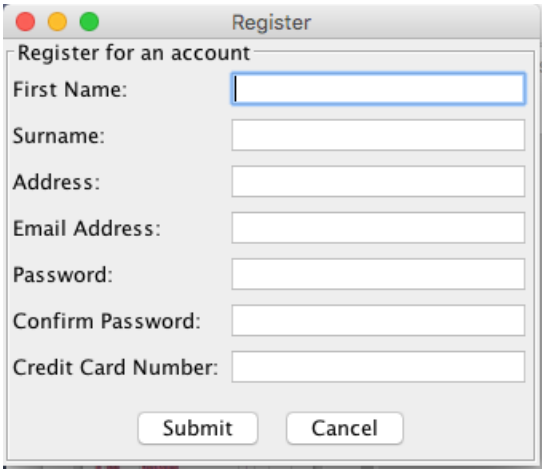
Automated testing is a huge part of quality assurance. And the most important thing to do when creating tests is to emulate an end user – so you might be tempted to say add more End to End test right? And to an extent you are correct but end to end tests are expensive. They take a lot of time and when they do fail they are not as efficient at pinpointing the source of the bugs as integration and unit tests. Just read this article slating the use of E2E tests by google http://googletesting.blogspot.ie/2015/04/just-say-no-to-more-end-to-end-tests.html . It's much easier to have a developer who understands their own code write basic integration tests to test the 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.

Screenshots of GUI

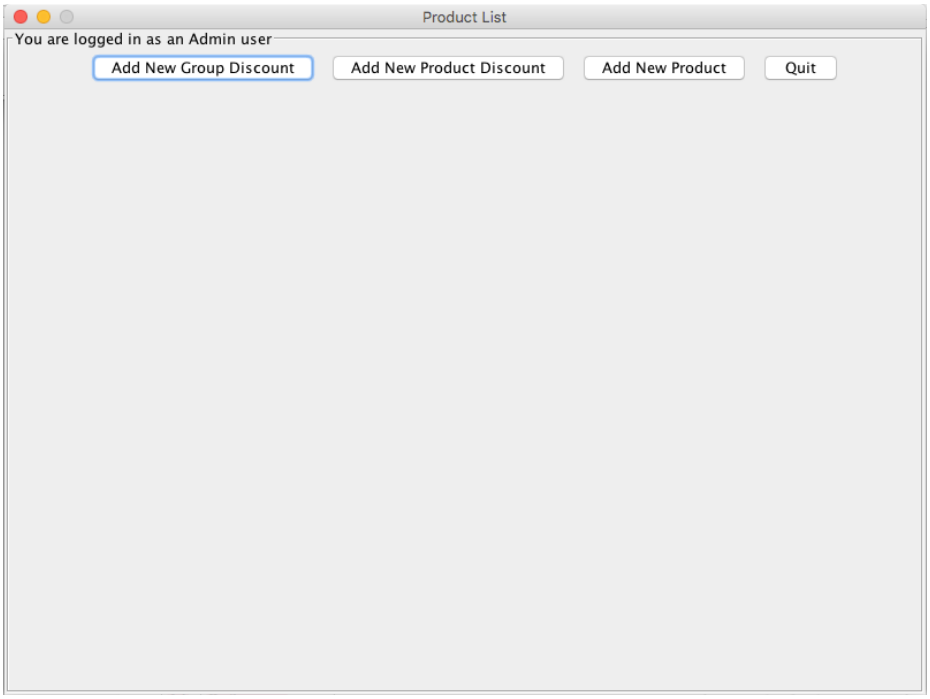
Login UI:



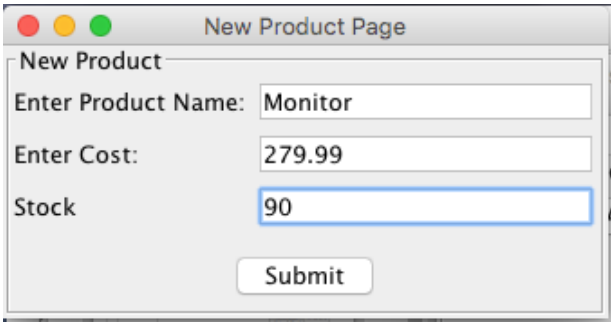
Register UI:

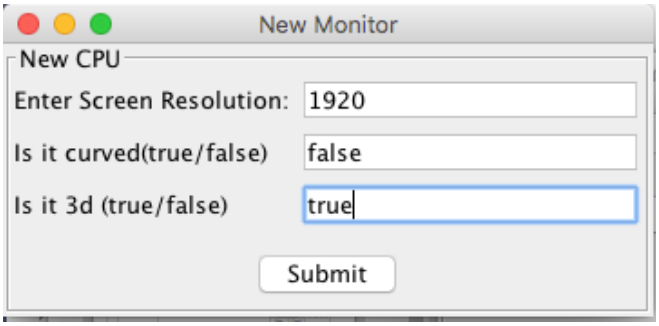


Admin Home UI:

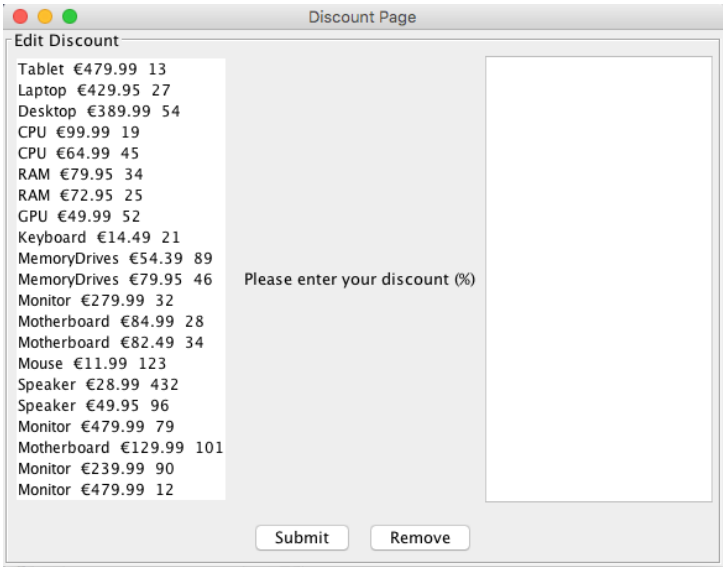


Add New Product UI:

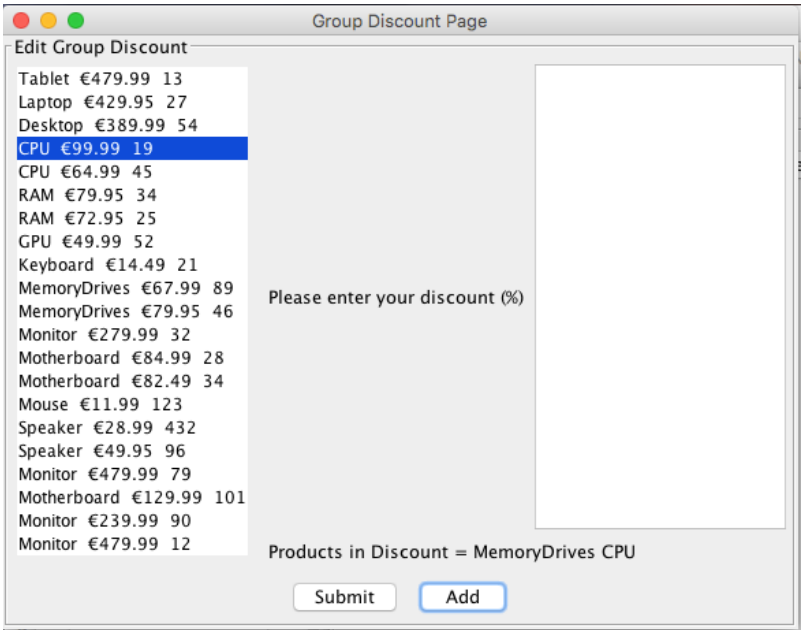




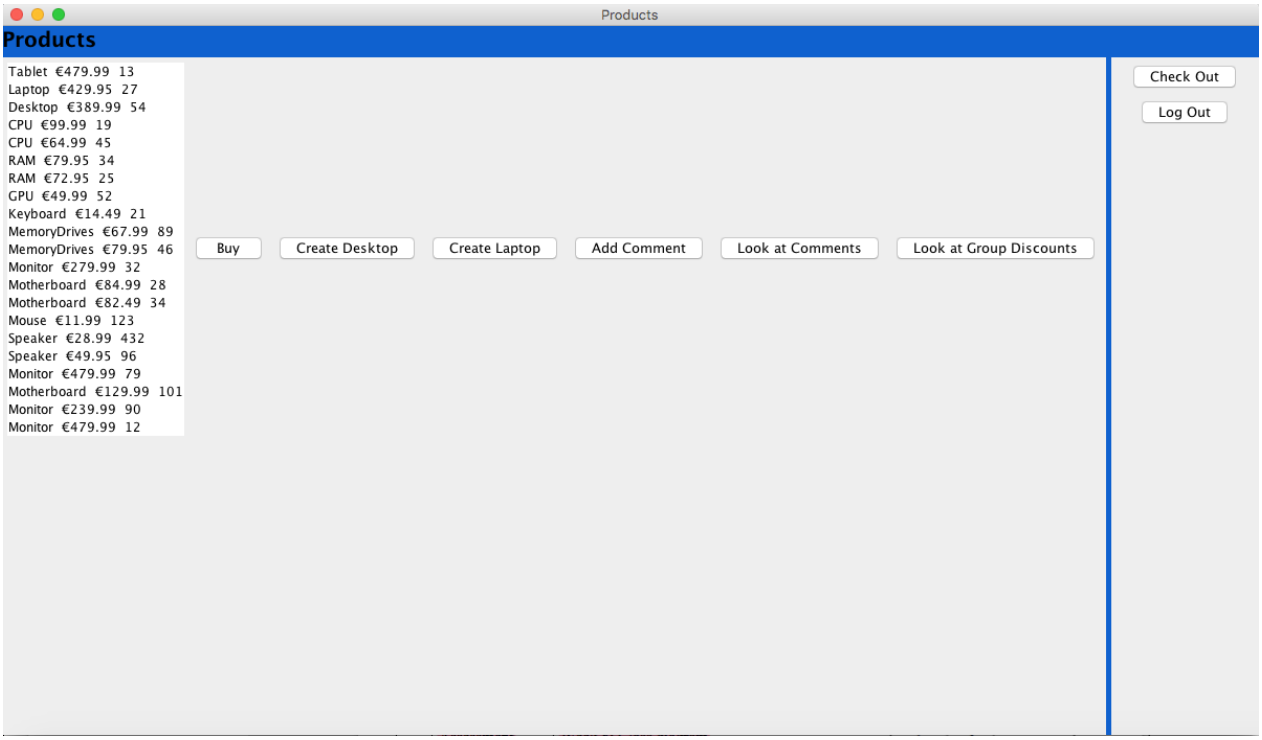
Add Product Discount UI:



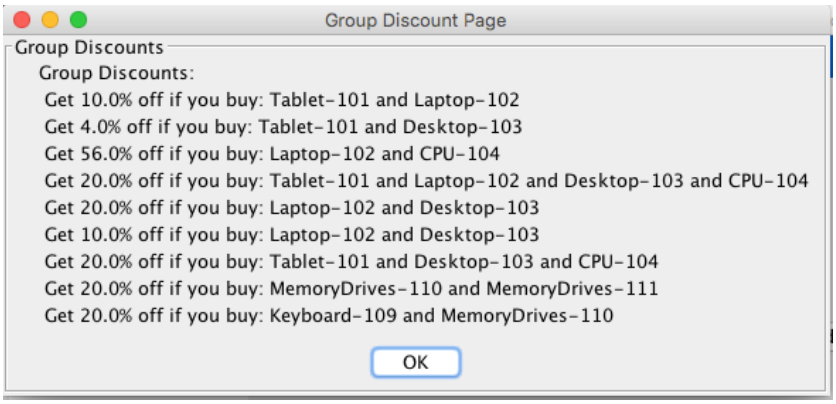
Add Group Product Discount UI:



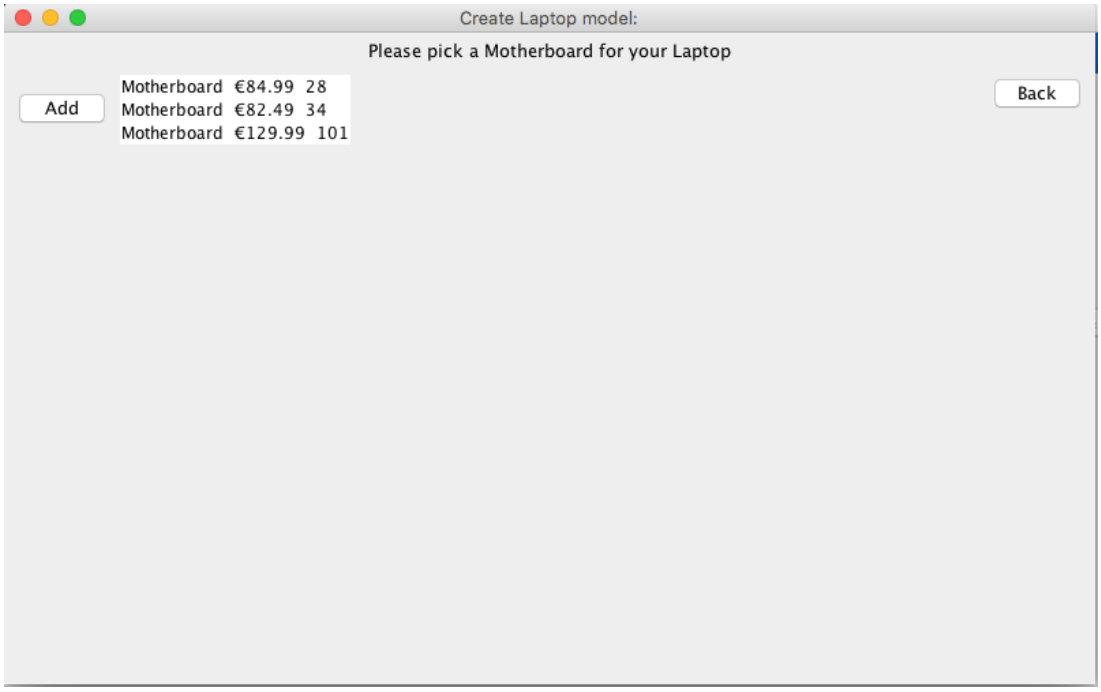
Customer Home UI:



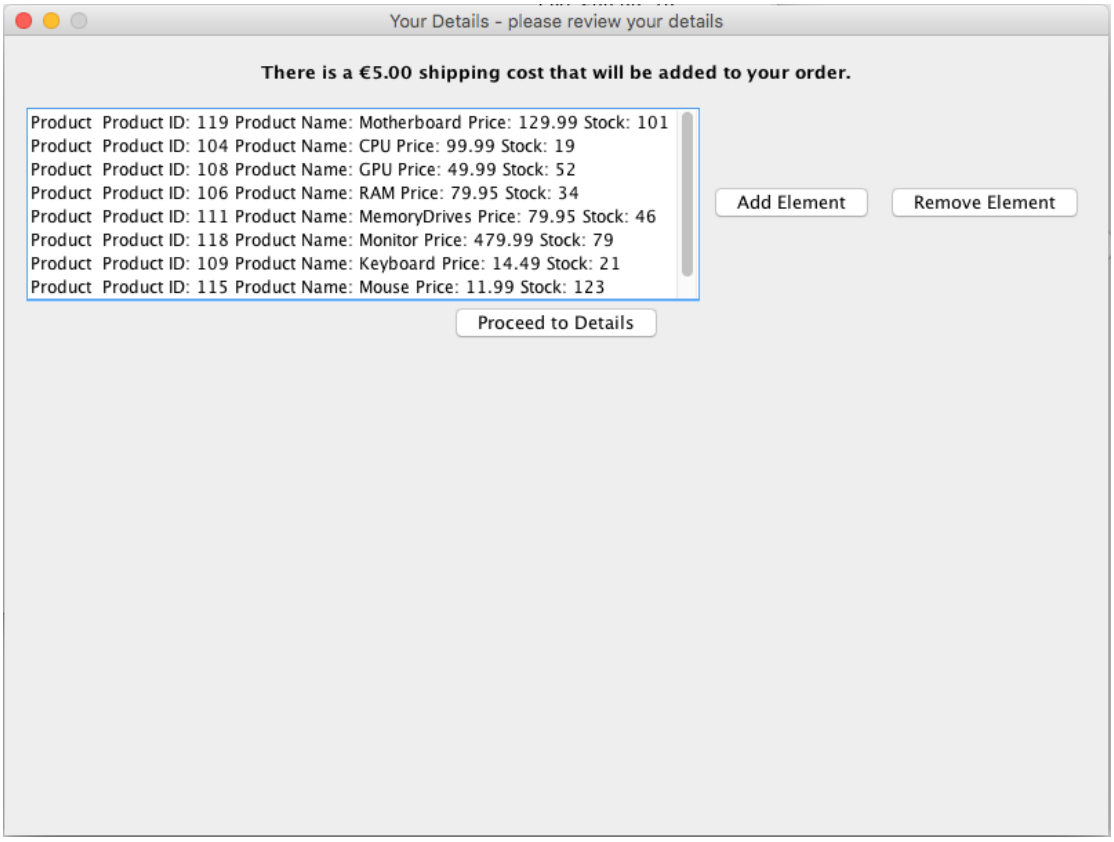
View Group Discounts:



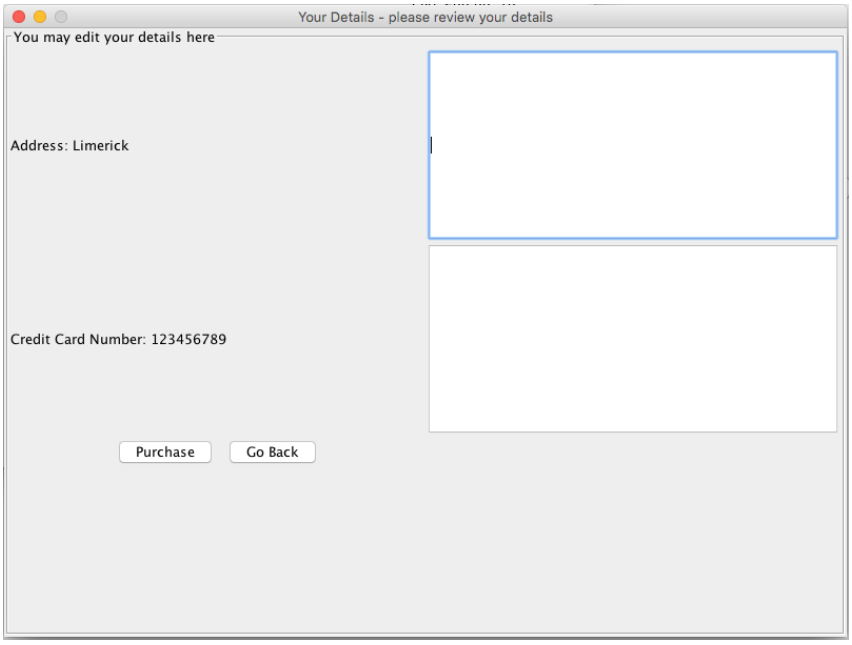
Create Laptop Model UI:



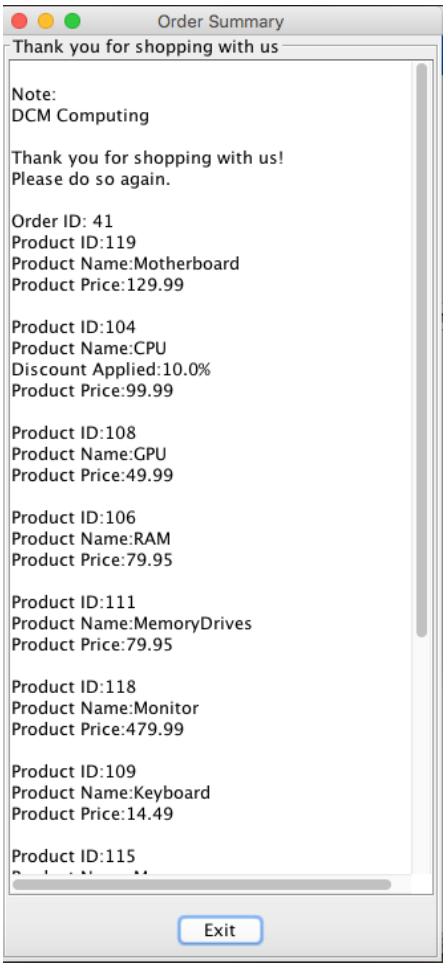
Cart UI:



Edit Customer Details UI:



Order Summary UI (Receipt):



System Architecture

Discussion

Architectural Decisions Taken:
The system was split into three subsystems for our implementation: the Business Layer, the Data Layer and the User Interface Layer.

We implemented a range of Model View Controller (MVC) design patterns in order to help our system be more flexible and easier to maintain. MVC separates business logic from the user interface. In our implementation we separated the user interface layer, the data layer and the business layer. By using MVC it allowed for code reuse within the system and it reduced coupling between the layers.

We also included a range of design patterns including the factory method which was implemented to help create proper subclasses of products so the system can easily determine which product was chosen by the customer at runtime.

UML Workbench

We chose Gliffy and Creately as our main UML workbenches due to the fact that they produce clear, easy to understand diagrams and they were also free to use. Each of the workbenches also provided some templates which aided us in creating our own diagrams for this implementation. We found it easier to divide our work between 2 different workbenches in order to avail of the strengths that each workbench possessed.

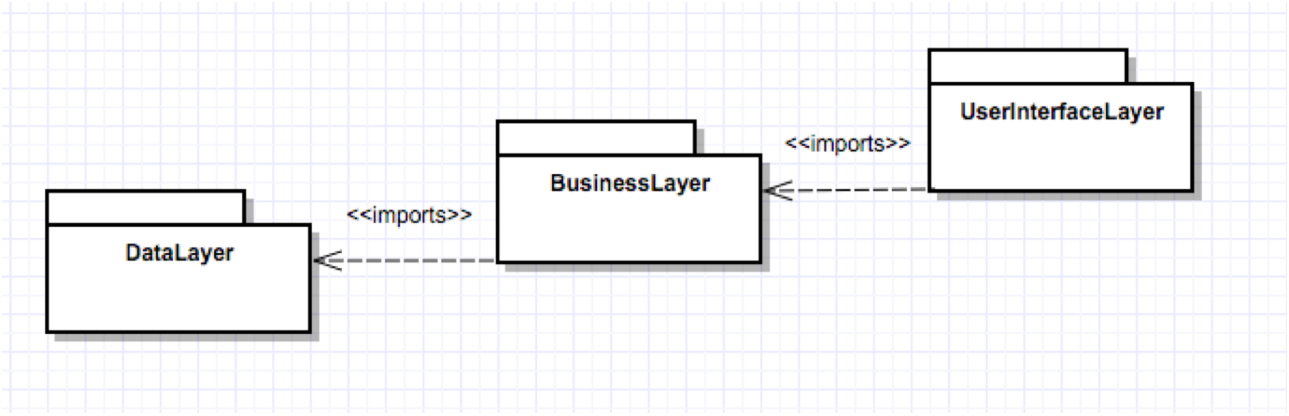
For example we found Creately much easier to use for the large analysis class diagram and state charts due to the fact that larger diagrams could be created easier and were much easier to read and navigate. We then used Gliffy for creating other diagrams, including the state diagrams and the sequence diagram.

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 the many architectures which support the Java Virtual Machine (JVM). Java has a JVM for many of the modern architectures which means 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.

Architecture Diagram



Analysis Sketches

Identifying Candidate Classes

Candidate Objects:

We decided to use the Data Driven Design (DDD) method for our project and we listed many potential candidate classes. We used the noun identification technique to find the potential classes.

Initial List:

Product	Support	CPU	Window	Order
Email	Mouse	Admin	PC	Notification
Voucher	Monitor	Click	Profile	ReportProblem
GraphicsCard	Address	HardDrive	Laptop	Website
Accept	Customer	Member	User	RAM

Heuristics:

1. Attribute - Purple
2. Too vague? - Red
3. Operation - Blue
4. Out of Scope - Orange
5. Equivalent - Green

Filtered List:

Member	Admin	Customer	Order	Voucher
ReportProblem	Product	CPU	GraphicsCard	HardDrive
RAM	Monitor	Mouse	Laptop	PC

Member:

A Member is a user that is either an Admin or a Customer. They have the ability to view and buy the products that are available to purchase on the website.

Admin:

An Admin is user of the website whose job is to maintain the site, add new products and deal with any problems that have been reported by another user. They would also be able to purchase items if they want.

Customer:

A Customer is a user of the website whose main purpose it to purchase items. They create an account in order to be able to choose and purchase the items that they require.

Order:
When a customer chooses the items they wish to purchase they create an order. This contains the information of all the items they wish to purchase, their account details, shipping details and credit card information.

Voucher:
Sometimes the Customer may obtain a discount voucher for the website. When they are purchasing their items, the Customer has the option to input a voucher code to get a discount on the final price of their items.

ReportProblem:
If a Customer encounters a problem while they are using the website they will be able to report it. When an Admin user logs on they will then be able to see any reported problems and deal with them.

Product:
Products are the items that are being sold on the website. The website sells a lot of different types a=of products such as CPUs, GPUs, hard drives and RAM. A product can be described with a ProductID, type, brand and price.

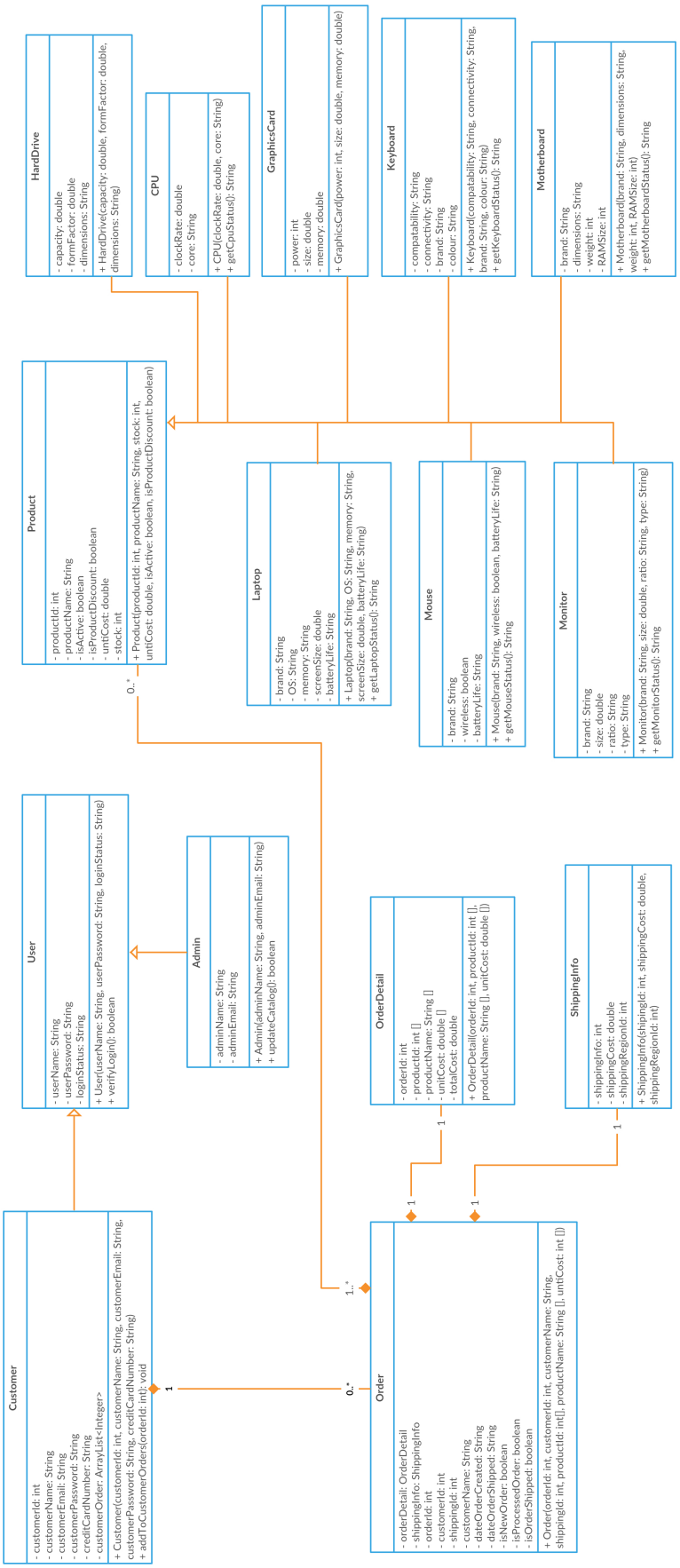
User:
A user is someone who has created an account on our website, which enables them to buy any of the products listed. All users have a unique user name, password and the system also keeps a record of some of their payment details, in order to make the payment process less tedious.

Discount:
A discount lists a product which has some sort of discount. The admin is the only person who can edit which products do have a discount and also the percentage off that product.

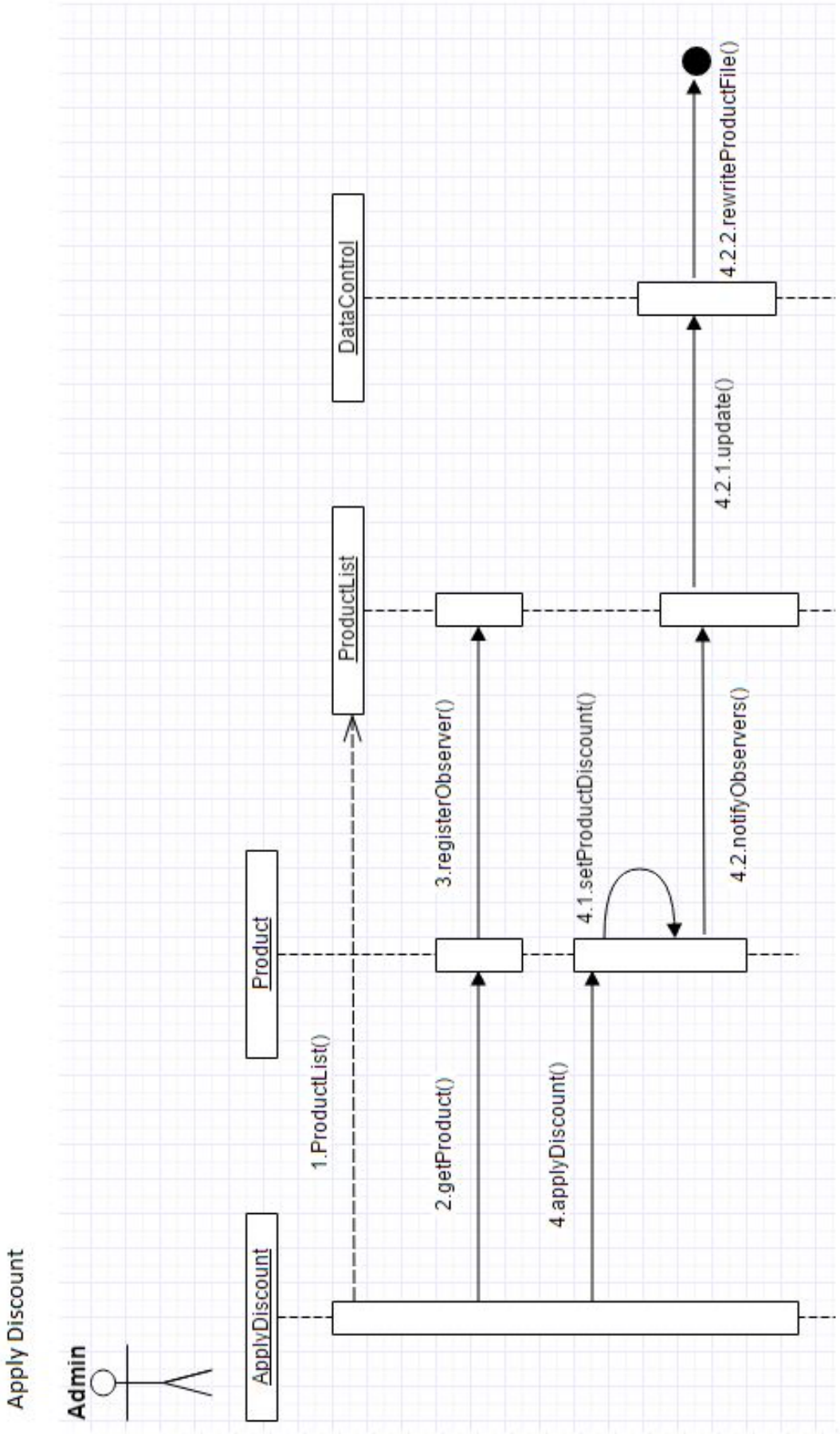
Laptop:
A customer will have the ability to go onto the website and purchase either a pre-configured laptop or choose the parts they want included in it e.g CPU, RAM etc.

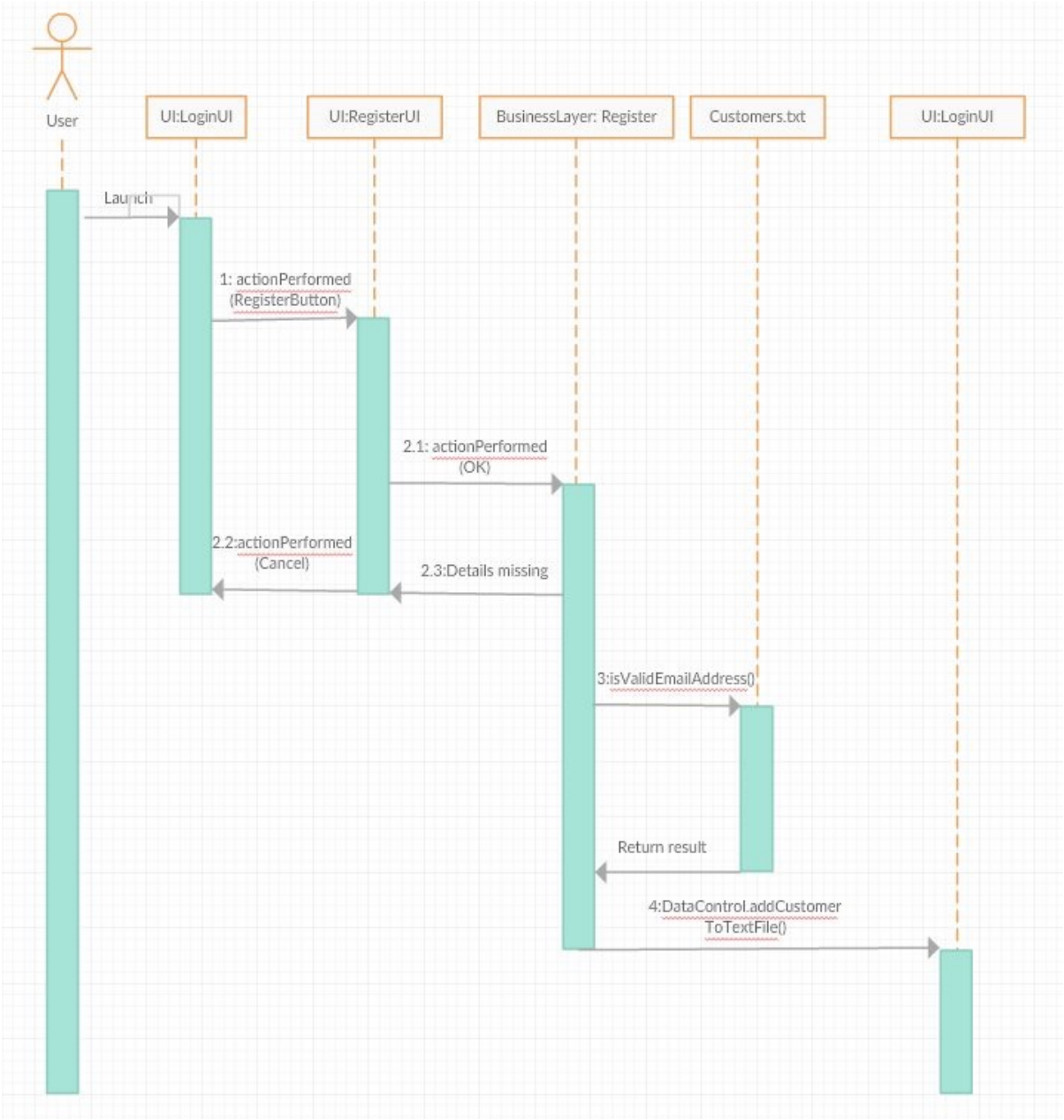
PC:
Customers have the option on the website to build their own PC by choosing the exact parts they want. PC contains a list of products that make up the PC however there can only be one type of product in each PC. For example, when building a PC a customer should not be able to pick 2 CPUs as only one is needed in a PC.

Class Diagram (Iteration 1)

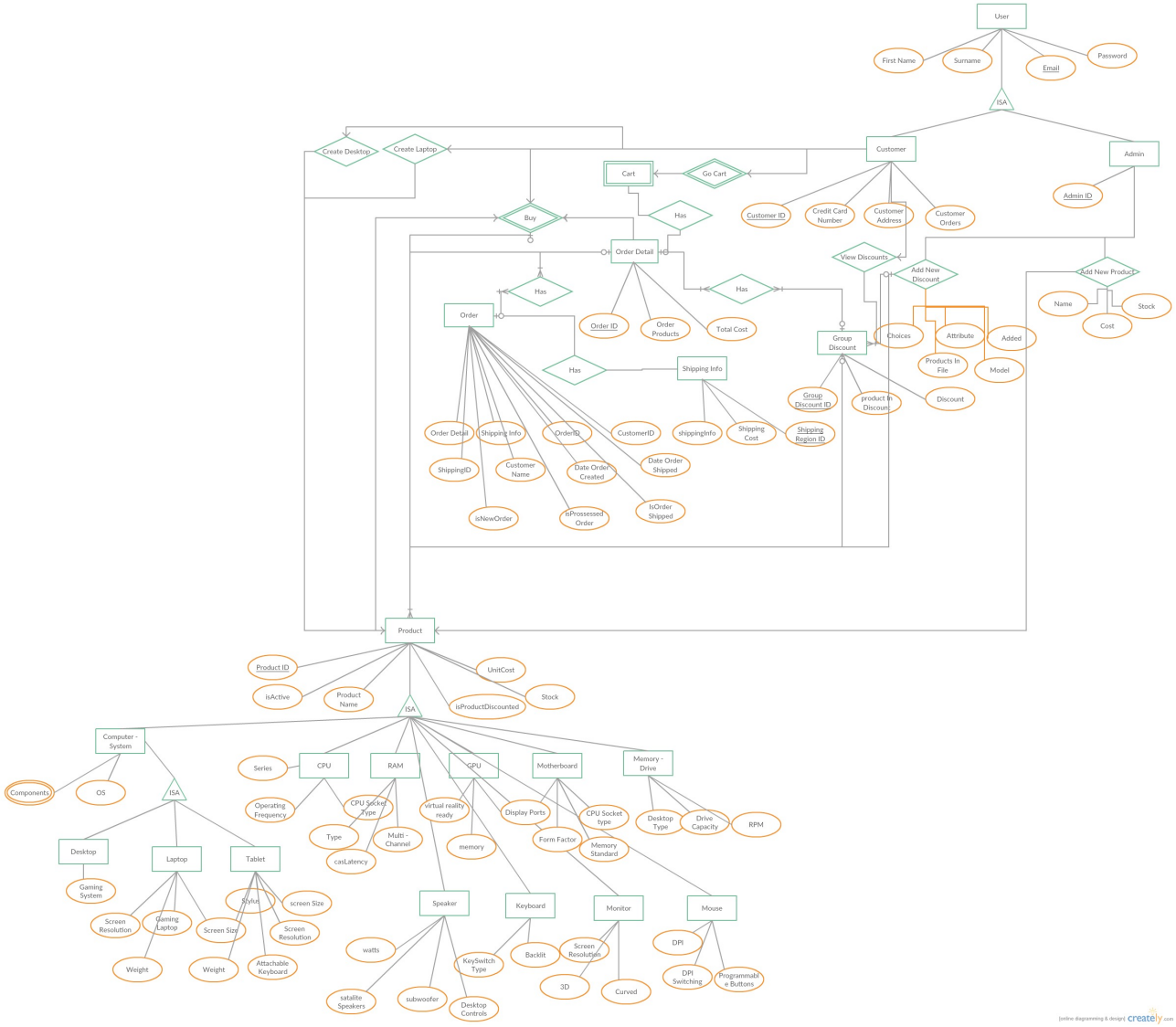


Interaction Diagrams




User
UI:LoginUI
UI:RegisterUI
BusinessLayer: Register
Customers.txt
UI:LoginUI
Launch
1: actionPerformed
(RegisterButton)
2.1: actionPerformed
(OK)
2.2:actionPerformed
(Cancel)
2.3:Details missing
3:isValidEmailAddress()
Return result
4:DataControl.addCustomer
ToTextFile()

Entity Relationship Diagram



Code Implementation

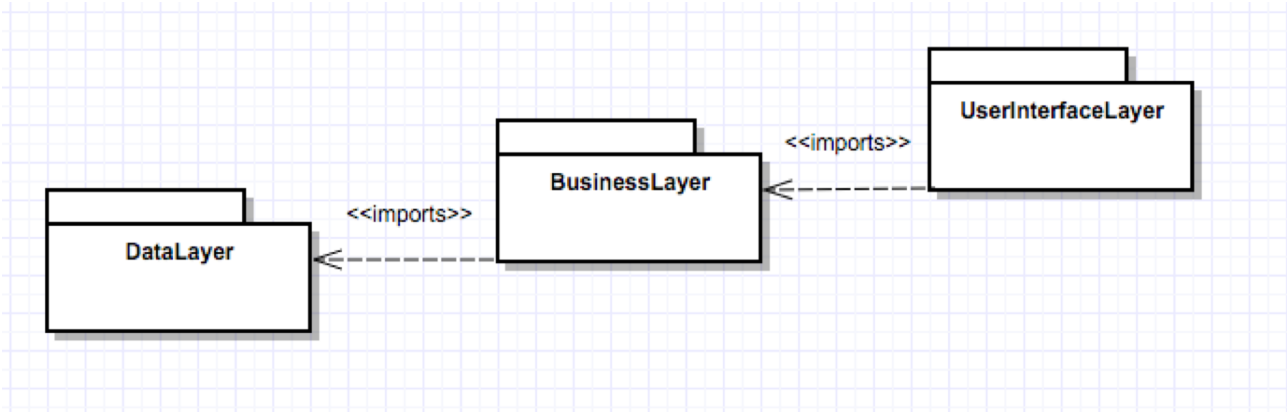
Class	Conor	Cian	Daniel	Darren	Michael	Lines In File
BusinessLayer Package						0
AddNewComment		55				55
AddNewGroupDiscount				37		37
AddNewProduct				35		35
ApplyDiscount				26		26
BusinessLayerDataControl	96					96
CreateLaptop			131			131
CreatePC			134			134
EditOrderDetails	30					30
Login		42				42
Observer				10		10
OrderSummary	45			15		60
productFactoryDesign					54	54
ProductList					47	47
Register		52				52
Subject				9		9
						0
BusinessLayer. CustomerClasses Sub Package						0
User		49				49
Admin		15				15
Customer		57				57
						0
BusinessLayer. OrderClasses Sub Package						0
BasicReceipt	9					9
GroupDiscount				81		81
HeaderReceipt	13					13
Order	137					137
OrderDetail	100			17		117
Receipt	6					6
ReceiptDecorator	16					16
ShippingInfo	28					28
ShippingReceipt	13					13
ThankYouReceipt	13					13
						0
BusinessLayer. ProductClasses Sub Package						0
Component			17			17
ComputerSystem			44			44
CPU					46	46
Desktop			35			35
GPU					56	56
Keyboard					36	36
Laptop			62			62
MemoryDrives					46	46
Monitor					46	46
Motherboard					57	57
Mouse					45	45
Product			179			179

Class	Conor	Cian	Daniel	Darren	Michael	Lines In File
RAM					47	47
Speaker					57	57
System					11	11
Tablet			60			60
						0
DataLayer Package						0
DatabaseInterface	30			3		33
DataControl	310	30		62	78	480
						0
UserInterfaceLayer Package						0
AddNewCommentUI		123				123
AddNewGroupDiscountUI				147		147
AddNewProductUI	56	169		417	74	716
ApplyDiscountUI				127		127
CartUI		100				100
CreateLaptopUI			194			194
CreatePCUI			201			201
EditOrderDetailsUI	91					91
GetCommentsUI		92				92
GetGroupDiscountsUI				89		89
LoginUI		100				100
OrderSummaryUI	35			16		51
ProductListUI					291	291
RegisterUI		119				119
						0
Total Added Vertically (per person)	1028	1003	1057	1091	991	5170

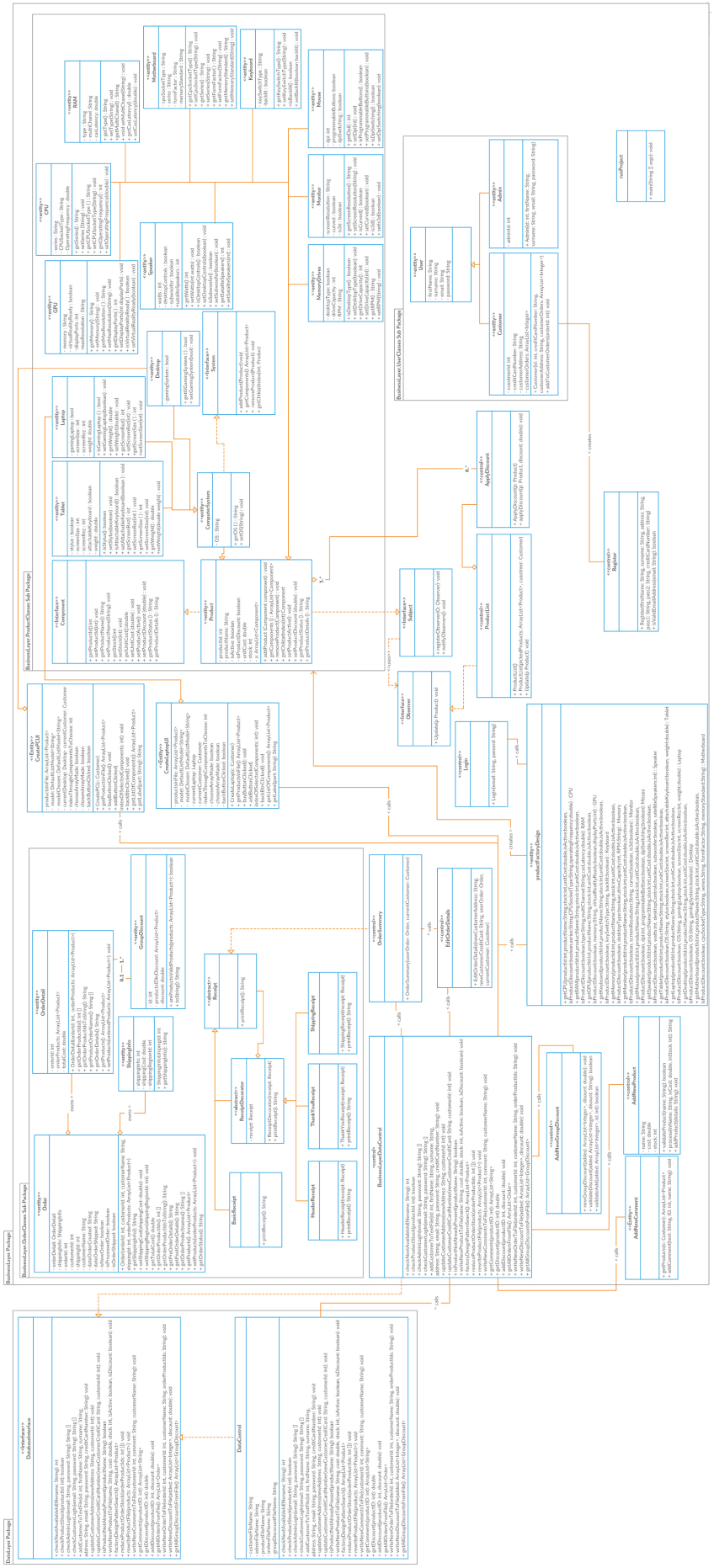
Class files have been appended to the end of this report.

Design Artefacts

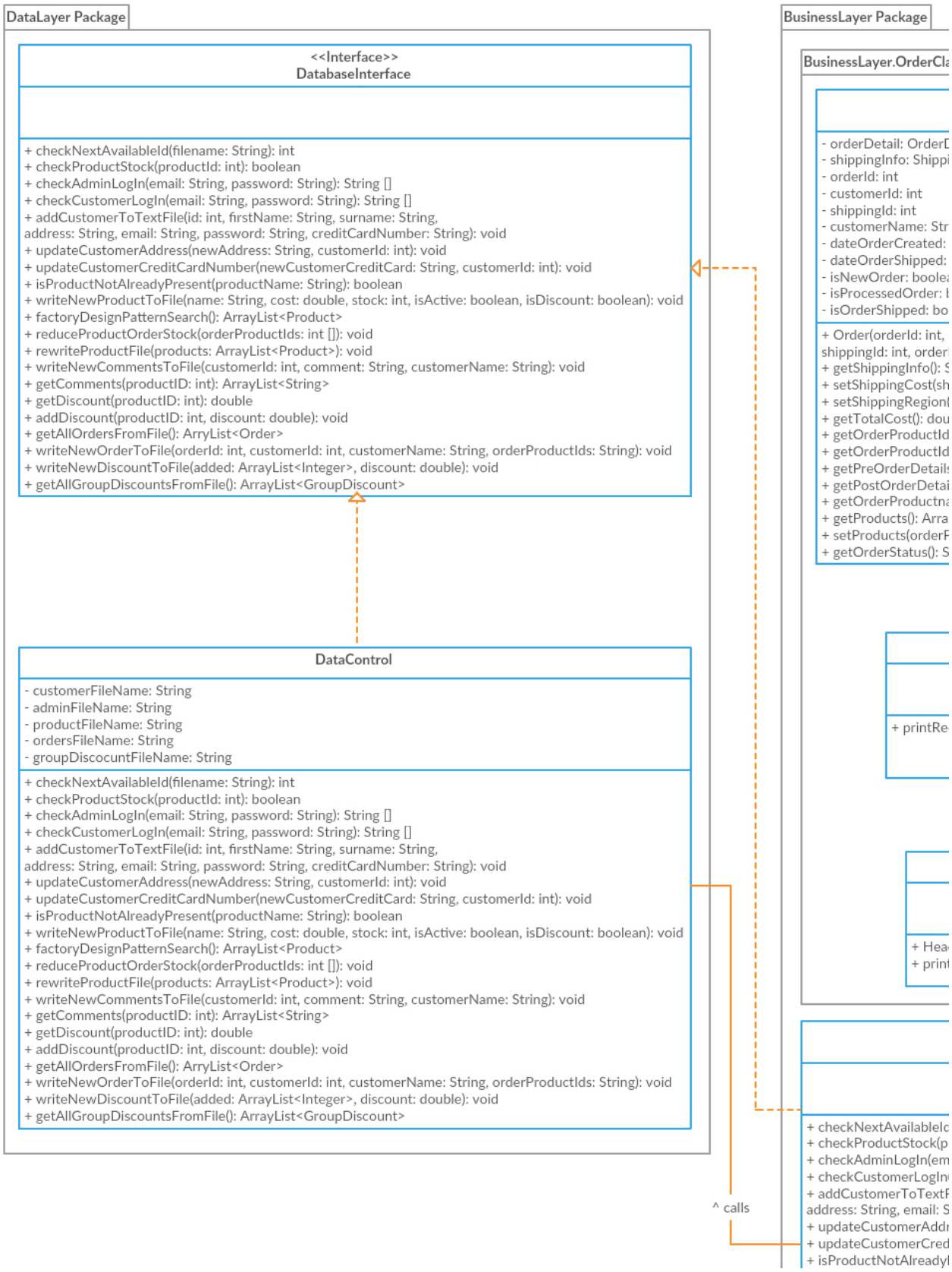
Architectural Diagram

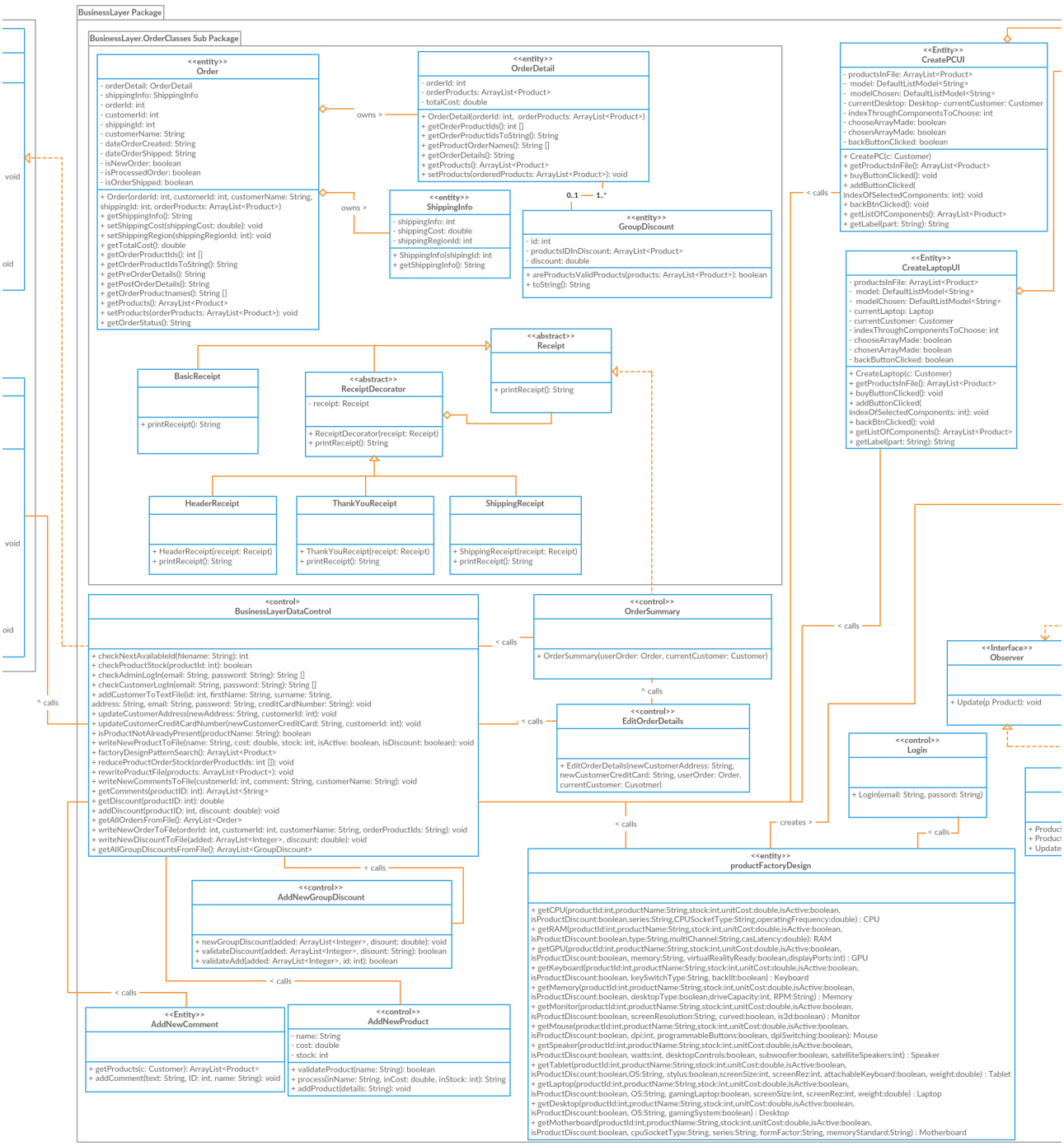


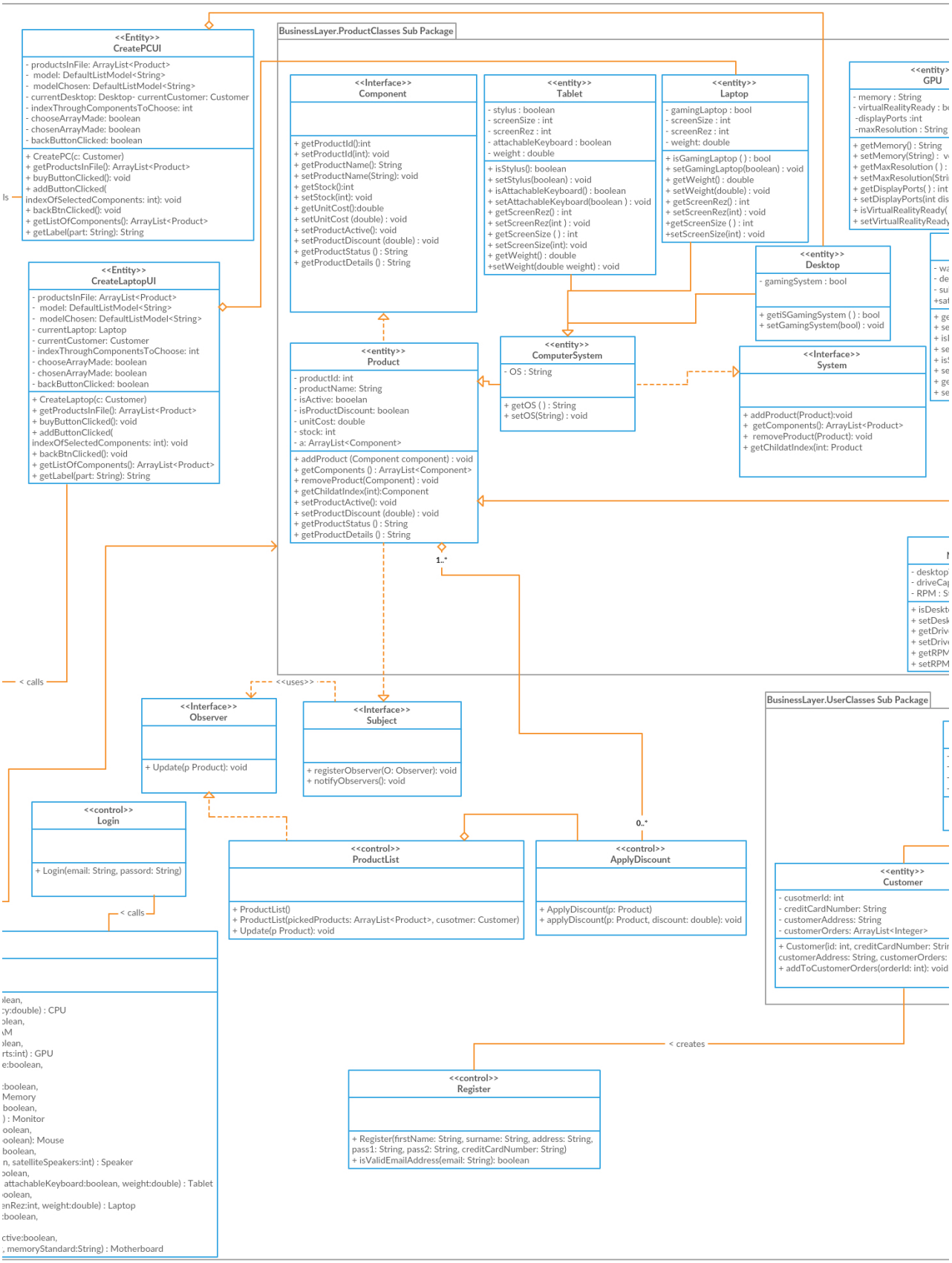
Design Class Diagram (Full)

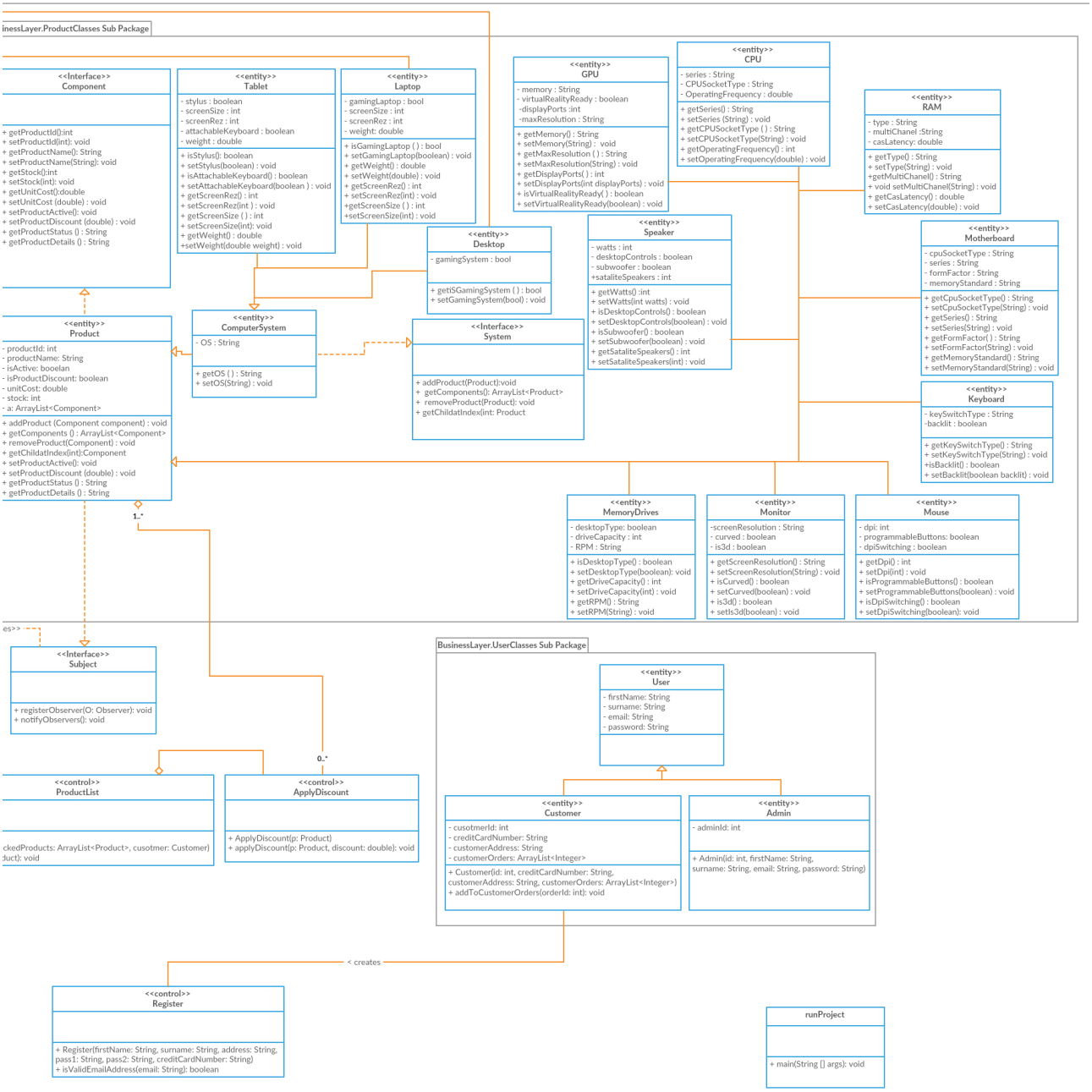


Design Class Diagram (Split Left to Right)



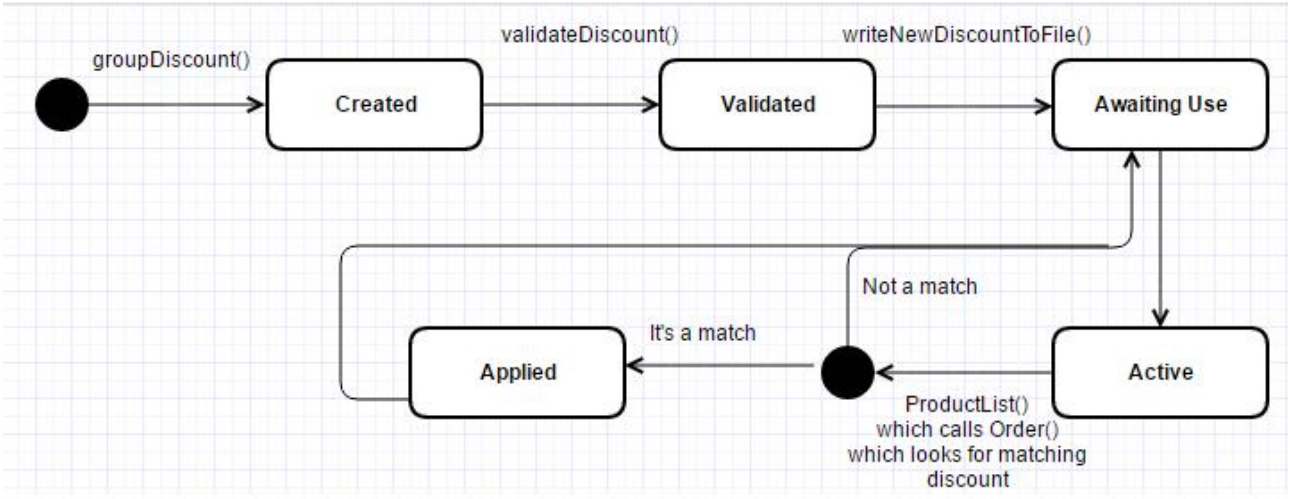

BusinessLayer Package
BusinessLayer.OrderClasses Sub Package
<<entity>>
Order
- orderDetail: OrderDetail
- shippingInfo: ShippingInfo
- orderId: int
- customerId: int
- shippingId: int
- customerName: String
- dateOrderCreated: String
- dateOrderShipped: String
- isNewOrder: boolean
- isProcessedOrder: boolean
- isOrderShipped: boolean
+ Order(orderId: int, customerId: int, customerName: String, shippingId: int, orderProducts: ArrayList<Product>)
+ getShippingInfo(): String
+ setShippingCost(shippingCost: double): void
+ setShippingRegion(shippingRegionId: int): void
+ getTotalCost(): double
+ getOrderProductIds(): int []
+ getOrderProductIdsToString(): String
+ getPreOrderDetails(): String
+ getPostOrderDetails(): String
+ getOrderProductnames(): String []
+ getProducts(): ArrayList<Product>
+ setProducts(orderProducts: ArrayList<Product>): void
+ getOrderStatus(): String
owns >
<<entity>>
OrderDetail
- orderId: int
- orderProducts: ArrayList<Product>
- totalCost: double
+ OrderDetail(orderId: int, orderProducts: ArrayList<Product>)
+ getOrderProductIds(): int []
+ getOrderProductIdsToString(): String
+ getProductOrderNames(): String []
+ getOrderDetails(): String
+ getProducts(): ArrayList<Product>
+ setProducts(orderedProducts: ArrayList<Product>): void
owns >
<<entity>>
ShippingInfo
- shippingInfo: int
- shippingCost: double
- shippingRegionId: int
+ ShippingInfo(shipingId: int
+ getShippingInfo(): String
0..1
1..*
<<entity>>
GroupDiscount
- id: int
- productsIDInDiscount: ArrayList<Product>
- discount: double
+ areProductsValidProducts(products: ArrayList<Product>): boolean
+ toString(): String
<<abstract>>
Receipt
+ printReceipt(): String
BasicReceipt
+ printReceipt(): String
<<abstract>>
ReceiptDecorator
- receipt: Receipt
+ ReceiptDecorator(receipt: Receipt)
+ printReceipt(): String
HeaderReceipt
+ HeaderReceipt(receipt: Receipt)
+ printReceipt(): String
ThankYouReceipt
+ ThankYouReceipt(receipt: Receipt)
+ printReceipt(): String
ShippingReceipt
+ ShippingReceipt(receipt: Receipt)
+ printReceipt(): String
<<Entity>>
CreatePCUI
- productsInFile: ArrayList<Product>
- model: DefaultListModel<String>
- modelChosen: DefaultListModel<String>
- currentDesktop: Desktop- currentCustomer: Customer
- indexThroughComponentsToChoose: int
- chooseArrayMade: boolean
- chosenArrayMade: boolean
- backButtonClicked: boolean
+ CreatePC(c: Customer)
+ getProductsInFile(): ArrayList<Product>
+ buyButtonClicked(): void
+ addButtonClicked(
indexOfSelectedComponents: int): void
+ backBtnClicked(): void
+ getListOfComponents(): ArrayList<Product>
+ getLabel(part: String): String
< calls
<<Entity>>
CreateLaptopUI
- productsInFile: ArrayList<Product>
- model: DefaultListModel<String>
- modelChosen: DefaultListModel<String>
- currentLaptop: Laptop
- currentCustomer: Customer
- indexThroughComponentsToChoose: int
- chooseArrayMade: boolean
- chosenArrayMade: boolean
- backButtonClicked: boolean
+ CreateLaptop(c: Customer)
+ getProductsInFile(): ArrayList<Product>
+ buyButtonClicked(): void
+ addButtonClicked(
indexOfSelectedComponents: int): void
+ backBtnClicked(): void
+ getListOfComponents(): ArrayList<Product>
+ getLabel(part: String): String
<<control>>
BusinessLayerDataControl
+ checkNextAvailableId(filename: String): int
+ checkProductStock(productId: int): boolean
+ checkAdminLogIn(email: String, password: String): String []
+ checkCustomerLogIn(email: String, password: String): String []
+ addCustomerToTextFile(id: int, firstName: String, surname: String, address: String, email: String, password: String, creditCardNumber: String): void
+ updateCustomerAddress(newAddress: String, customerId: int): void
+ updateCustomerCreditCardNumber(newCustomerCreditCard: String, customerId: int): void
+ isProductNotAlreadyPresent(productName: String): boolean
+ writeNewProductToFile(name: String, cost: double, stock: int, isActive: boolean, isDiscount: boolean): void
+ factoryDesignPatternSearch(): ArrayList<Product>
+ reduceProductOrderStock(orderProductIds: int []): void
+ rewriteProductFile(products: ArrayList<Product>): void
+ writeNewCommentsToFile(customerId: int, comment: String, customerName: String): void
+ getComments(productID: int): ArrayList<String>
+ getDiscount(productID: int): double
+ addDiscount(productID: int, discount: double): void
+ getAllOrdersFromFile(): ArrayList<Order>
+ writeNewOrderToFile(orderId: int, customerId: int, customerName: String, orderProductIds: String): void
+ writeNewDiscountToFile(added: ArrayList<Integer>, discount: double): void
+ getAllGroupDiscountsFromFile(): ArrayList<GroupDiscount>
< calls
<<control>>
OrderSummary
+ OrderSummary(userOrder: Order, currentCustomer: Customer)
^ calls
<<control>>
EditOrderDetails
+ EditOrderDetails(newCustomerAddress: String, newCustomerCreditCard: String, userOrder: Order, currentCustomer: Cusotmer)
< calls
<<Interface>>
Observer
+ Update(p Product): void
<<control>>
Login
+ Login(email: String, passord: String)
< calls
creates >
<<control>>
AddNewGroupDiscount
+ newGroupDiscount(added: ArrayList<Integer>, disount: double): void
+ validateDiscount(added: ArrayList<Integer>, disount: String): boolean
+ validateAdd(added: ArrayList<Integer>, id: int): boolean
<<Entity>>
AddNewComment
+ getProducts(c: Customer): ArrayList<Product>
+ addComment(text: String, ID: int, name: String): void
<<control>>
AddNewProduct
- name: String
- cost: double
- stock: int
+ validateProduct(name: String): boolean
+ process(inName: String, inCost: double, inStock: int): String
+ addProduct(details: String): void
<<entity>>
productFactoryDesign
+ getCPU(productId:int,productName:String,stock:int,unitCost:double,isActive:boolean, isProductDiscount:boolean,series:String,CPUSocketType:String,operatingFrequency:double) : CPU
+ getRAM(productId:int,productName:String,stock:int,unitCost:double,isActive:boolean, isProductDiscount:boolean,type:String,multiChannel:String,casLatency:double): RAM
+ getGPU(productId:int,productName:String,stock:int,unitCost:double,isActive:boolean, isProductDiscount:boolean, memory:String, virtualRealityReady:boolean,displayPorts:int) : GPU
+ getKeyboard(productId:int,productName:String,stock:int,unitCost:double,isActive:boolean, isProductDiscount:boolean, keySwitchType:String, backlit:boolean) : Keyboard
+ getMemory(productId:int,productName:String,stock:int,unitCost:double,isActive:boolean, isProductDiscount:boolean, desktopType:boolean,driveCapacity:int, RPM:String) : Memory
+ getMonitor(productId:int,productName:String,stock:int,unitCost:double,isActive:boolean, isProductDiscount:boolean, screenResolution:String, curved:boolean, is3d:boolean) : Monitor
+ getMouse(productId:int,productName:String,stock:int,unitCost:double,isActive:boolean, isProductDiscount:boolean, dpi:int, programmableButtons:boolean, dpiSwitching:boolean): Mouse
+ getSpeaker(productId:int,productName:String,stock:int,unitCost:double,isActive:boolean, isProductDiscount:boolean, watts:int, desktopControls:boolean, subwoofer:boolean, satelliteSpeakers:int) : Speaker
+ getTablet(productId:int,productName:String,stock:int,unitCost:double,isActive:boolean, isProductDiscount:boolean,OS:String, stylus:boolean,screenSize:int, screenRez:int, attachableKeyboard:boolean, weight:double) : Tablet
+ getLaptop(productId:int,productName:String,stock:int,unitCost:double,isActive:boolean, isProductDiscount:boolean, OS:String, gamingLaptop:boolean, screenSize:int, screenRez:int, weight:double) : Laptop
+ getDesktop(productId:int,productName:String,stock:int,unitCost:double,isActive:boolean, isProductDiscount:boolean, OS:String, gamingSystem:boolean) : Desktop
+ getMotherboard(productId:int,productName:String,stock:int,unitCost:double,isActive:boolean, isProductDiscount:boolean, cpuSocketType:String, series:String, formFactor:String, memoryStandard:String) : Motherboard
^ calls
void
oid


BusinessLayer.ProductClasses Sub Package
<<Entity>>
CreatePCUI
- productsInFile: ArrayList<Product>
- model: DefaultListModel<String>
- modelChosen: DefaultListModel<String>
- currentDesktop: Desktop- currentCustomer: Customer
- indexThroughComponentsToChoose: int
- chooseArrayMade: boolean
- chosenArrayMade: boolean
- backButtonClicked: boolean
+ CreatePC(c: Customer)
+ getProductsInFile(): ArrayList<Product>
+ buyButtonClicked(): void
+ addButtonClicked(
indexOfSelectedComponents: int): void
+ backBtnClicked(): void
+ getListOfComponents(): ArrayList<Product>
+ getLabel(part: String): String
<<Interface>>
Component
+ getProductId():int
+ setProductId(int): void
+ getProductName(): String
+ setProductName(String): void
+ getStock():int
+ setStock(int): void
+ getUnitCost():double
+ setUnitCost (double) : void
+ setProductActive(): void
+ setProductDiscount (double) : void
+ getProductStatus () : String
+ getProductDetails () : String
<<entity>>
Tablet
- stylus : boolean
- screenSize : int
- screenRez : int
- attachableKeyboard : boolean
- weight : double
+ isStylus(): boolean
+ setStylus(boolean) : void
+ isAttachableKeyboard() : boolean
+ setAttachableKeyboard(boolean) : void
+ getScreenRez() : int
+ setScreenRez(int) : void
+ getScreenSize ( ) : int
+ setScreenSize(int): void
+ getWeight() : double
+setWeight(double weight) : void
<<entity>>
Laptop
- gamingLaptop : bool
- screenSize : int
- screenRez : int
- weight: double
+ isGamingLaptop ( ) : bool
+ setGamingLaptop(boolean) : void
+ getWeight() : double
+ setWeight(double) : void
+ getScreenRez() : int
+ setScreenRez(int) : void
+getScreenSize ( ) : int
+setScreenSize(int) : void
<<entity>>
GPU
- memory : String
- virtualRealityReady : b
-displayPorts :int
-maxResolution : String
+ getMemory() : String
+ setMemory(String) :
+ getMaxResolution ( ) :
+ setMaxResolution(Stri
+ getDisplayPorts( ) : int
+ setDisplayPorts(int dis
+ isVirtualRealityReady(
+ setVirtualRealityReady
<<Entity>>
CreateLaptopUI
- productsInFile: ArrayList<Product>
- model: DefaultListModel<String>
- modelChosen: DefaultListModel<String>
- currentLaptop: Laptop
- currentCustomer: Customer
- indexThroughComponentsToChoose: int
- chooseArrayMade: boolean
- chosenArrayMade: boolean
- backButtonClicked: boolean
+ CreateLaptop(c: Customer)
+ getProductsInFile(): ArrayList<Product>
+ buyButtonClicked(): void
+ addButtonClicked(
indexOfSelectedComponents: int): void
+ backBtnClicked(): void
+ getListOfComponents(): ArrayList<Product>
+ getLabel(part: String): String
<<entity>>
Desktop
- gamingSystem : bool
+ getiSGamingSystem ( ) : bool
+ setGamingSystem(bool) : void
<<entity>>
Product
- productId: int
- productName: String
- isActive: booelan
- isProductDiscount: boolean
- unitCost: double
- stock: int
- a: ArrayList<Component>
+ addProduct (Component component) : void
+ getComponents () : ArrayList<Component>
+ removeProduct(Component) : void
+ getChildatIndex(int):Component
+ setProductActive(): void
+ setProductDiscount (double) : void
+ getProductStatus () : String
+ getProductDetails () : String
<<entity>>
ComputerSystem
- OS : String
+ getOS ( ) : String
+ setOS(String) : void
<<Interface>>
System
+ addProduct(Product):void
+ getComponents(): ArrayList<Product>
+ removeProduct(Product): void
+ getChildatIndex(int: Product
1..*
< calls
<<uses>>
<<Interface>>
Observer
+ Update(p Product): void
<<Interface>>
Subject
+ registerObserver(O: Observer): void
+ notifyObservers(): void
BusinessLayer.UserClasses Sub Package
<<control>>
Login
+ Login(email: String, passord: String)
0..*
<<control>>
ProductList
+ ProductList()
+ ProductList(pickedProducts: ArrayList<Product>, cusotmer: Customer)
+ Update(p Product): void
<<control>>
ApplyDiscount
+ ApplyDiscount(p: Product)
+ applyDiscount(p: Product, discount: double): void
<<entity>>
Customer
- cusotmerId: int
- creditCardNumber: String
- customerAddress: String
- customerOrders: ArrayList<Integer>
+ Customer(id: int, creditCardNumber: Strin
customerAddress: String, customerOrders:
+ addToCustomerOrders(orderId: int): void
< calls
< creates
<<control>>
Register
+ Register(firstName: String, surname: String, address: String,
pass1: String, pass2: String, creditCardNumber: String)
+ isValidEmailAddress(email: String): boolean

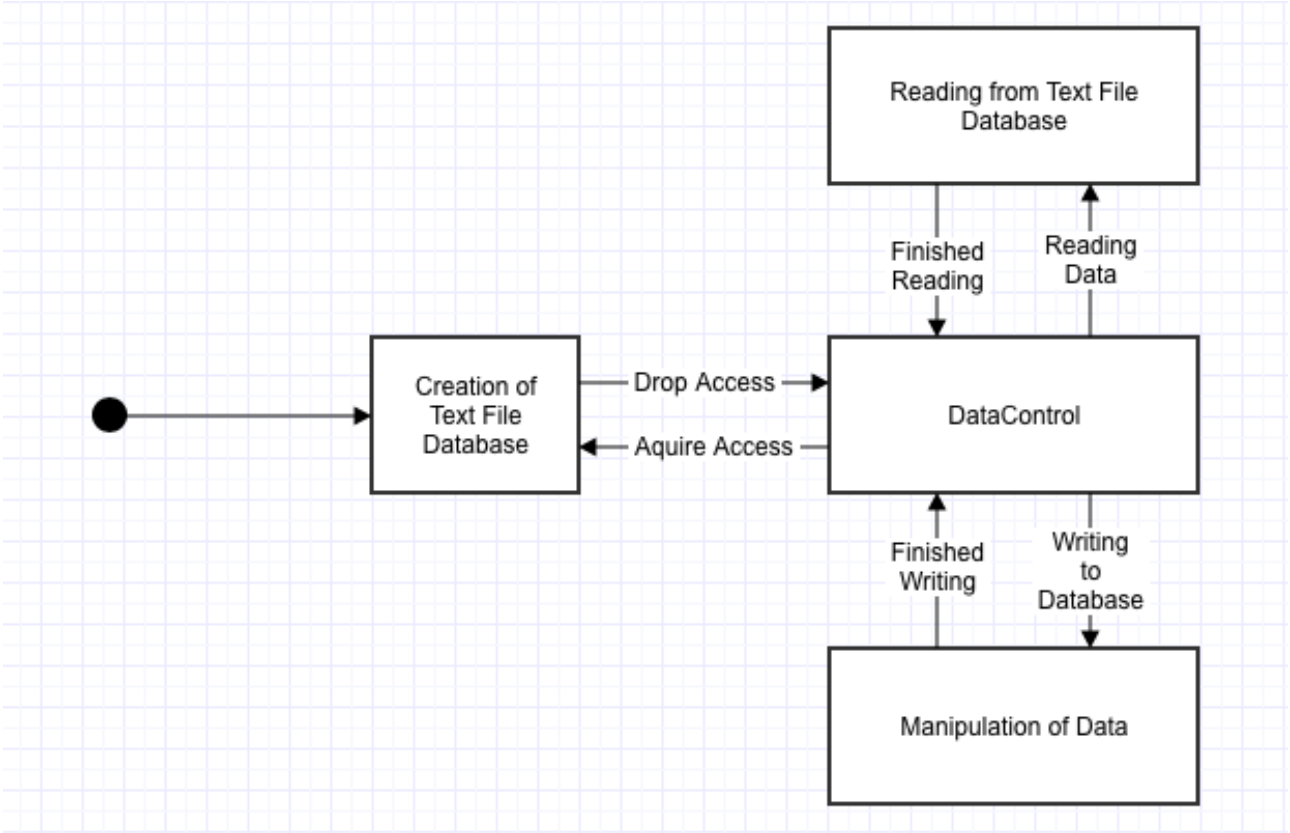

inessLayer.ProductClasses Sub Package
<<Interface>>
Component
+ getProductId():int
+ setProductId(int): void
+ getProductName(): String
+ setProductName(String): void
+ getStock():int
+ setStock(int): void
+ getUnitCost():double
+ setUnitCost (double) : void
+ setProductActive(): void
+ setProductDiscount (double) : void
+ getProductStatus () : String
+ getProductDetails () : String
<<entity>>
Tablet
- stylus : boolean
- screenSize : int
- screenRez : int
- attachableKeyboard : boolean
- weight : double
+ isStylus(): boolean
+ setStylus(boolean) : void
+ isAttachableKeyboard() : boolean
+ setAttachableKeyboard(boolean) : void
+ getScreenRez() : int
+ setScreenRez(int) : void
+ getScreenSize ( ) : int
+ setScreenSize(int): void
+ getWeight() : double
+setWeight(double weight) : void
<<entity>>
Laptop
- gamingLaptop : bool
- screenSize : int
- screenRez : int
- weight: double
+ isGamingLaptop ( ) : bool
+ setGamingLaptop(boolean) : void
+ getWeight() : double
+ setWeight(double) : void
+ getScreenRez() : int
+ setScreenRez(int) : void
+getScreenSize ( ) : int
+setScreenSize(int) : void
<<entity>>
GPU
- memory : String
- virtualRealityReady : boolean
-displayPorts :int
-maxResolution : String
+ getMemory() : String
+ setMemory(String) : void
+ getMaxResolution ( ) : String
+ setMaxResolution(String) : void
+ getDisplayPorts( ) : int
+ setDisplayPorts(int displayPorts) : void
+ isVirtualRealityReady( ) : boolean
+ setVirtualRealityReady(boolean) : void
<<entity>>
CPU
- series : String
- CPUSocketType : String
- OperatingFrequency : double
+ getSeries() : String
+ setSeries (String) : void
+ getCPUSocketType ( ) : String
+ setCPUSocketType(String) : void
+ getOperatingFrequency() : int
+ setOperatingFrequency(double) : void
<<entity>>
RAM
- type : String
- multiChanel :String
- casLatency: double
+ getType() : String
+ setType(String) : void
+getMultiChanel() : String
+ void setMultiChanel(String) : void
+ getCasLatency() : double
+ setCasLatency(double) : void
<<entity>>
Desktop
- gamingSystem : bool
+ getISGamingSystem ( ) : bool
+ setGamingSystem(bool) : void
<<entity>>
Speaker
- watts : int
- desktopControls : boolean
- subwoofer : boolean
+sataliteSpeakers : int
+ getWatts() :int
+ setWatts(int watts) : void
+ isDesktopControls() : boolean
+ setDesktopControls(boolean) : void
+ isSubwoofer() : boolean
+ setSubwoofer(boolean) : void
+ getSataliteSpeakers() : int
+ setSataliteSpeakers(int) : void
<<entity>>
Motherboard
- cpuSocketType : String
- series : String
- formFactor : String
- memoryStandard : String
+ getCpuSocketType() : String
+ setCpuSocketType(String) : void
+ getSeries() : String
+ setSeries(String) : void
+ getFormFactor( ) : String
+ setFormFactor(String) : void
+ getMemoryStandard() : String
+ setMemoryStandard(String) : void
<<entity>>
Product
- productId: int
- productName: String
- isActive: booelan
- isProductDiscount: boolean
- unitCost: double
- stock: int
- a: ArrayList<Component>
+ addProduct (Component component) : void
+ getComponents () : ArrayList<Component>
+ removeProduct(Component) : void
+ getChildatIndex(int):Component
+ setProductActive(): void
+ setProductDiscount (double) : void
+ getProductStatus () : String
+ getProductDetails () : String
<<entity>>
ComputerSystem
- OS : String
+ getOS ( ) : String
+ setOS(String) : void
<<Interface>>
System
+ addProduct(Product):void
+ getComponents(): ArrayList<Product>
+ removeProduct(Product): void
+ getChildatIndex(int: Product
<<entity>>
Keyboard
- keySwitchType : String
-backlit : boolean
+ getKeySwitchType() : String
+ setKeySwitchType(String) : void
+isBacklit() : boolean
+ setBacklit(boolean backlit) : void
<<entity>>
MemoryDrives
- desktopType: boolean
- driveCapacity : int
- RPM : String
+ isDesktopType() : boolean
+ setDesktopType(boolean): void
+ getDriveCapacity() : int
+ setDriveCapacity(int) : void
+ getRPM() : String
+ setRPM(String) : void
<<entity>>
Monitor
-screenResolution : String
- curved : boolean
- is3d : boolean
+ getScreenResolution() : String
+ setScreenResolution(String) : void
+ isCurved() : boolean
+ setCurved(boolean) : void
+ is3d() : boolean
+ setIs3d(boolean) : void
<<entity>>
Mouse
- dpi: int
- programmableButtons: boolean
- dpiSwitching : boolean
+ getDpi() : int
+ setDpi(int) : void
+ isProgrammableButtons() : boolean
+ setProgrammableButtons(boolean) : void
+ isDpiSwitching() : boolean
+ setDpiSwitching(boolean): void
1..*
<<Interface>>
Subject
+ registerObserver(O: Observer): void
+ notifyObservers(): void
0..*
<<control>>
ProductList
ckedProducts: ArrayList<Product>, cusotmer: Customer)
duct): void
<<control>>
ApplyDiscount
+ ApplyDiscount(p: Product)
+ applyDiscount(p: Product, discount: double): void
BusinessLayer.UserClasses Sub Package
<<entity>>
User
- firstName: String
- surname: String
- email: String
- password: String
<<entity>>
Customer
- cusotmerId: int
- creditCardNumber: String
- customerAddress: String
- customerOrders: ArrayList<Integer>
+ Customer(id: int, creditCardNumber: String, customerAddress: String, customerOrders: ArrayList<Integer>)
+ addToCustomerOrders(orderId: int): void
<<entity>>
Admin
- adminId: int
+ Admin(id: int, firstName: String, surname: String, email: String, password: String)
< creates
<<control>>
Register
+ Register(firstName: String, surname: String, address: String, pass1: String, pass2: String, creditCardNumber: String)
+ isValidEmailAddress(email: String): boolean
runProject
+ main(String [] args): void

State Charts

Apply Group Discount:

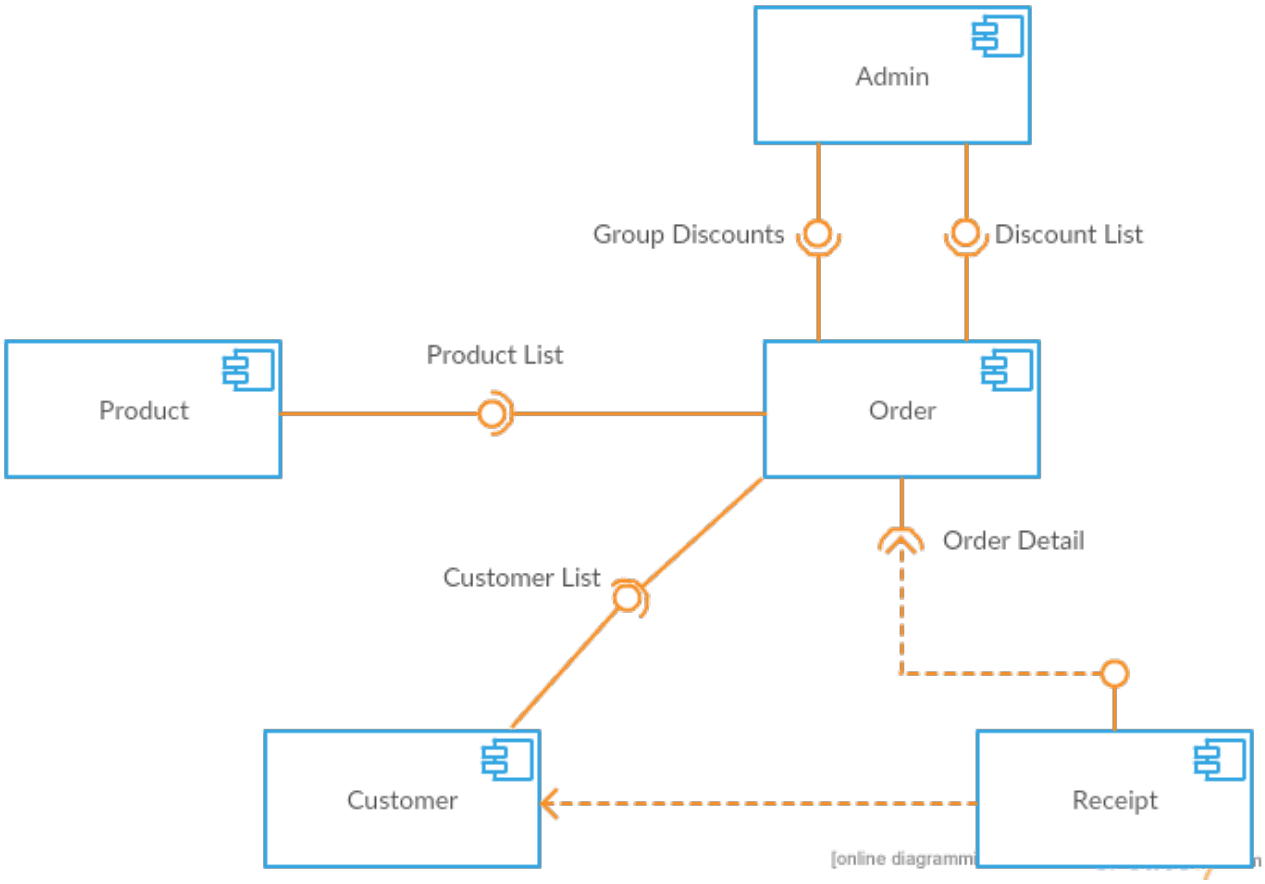


Database Manager:

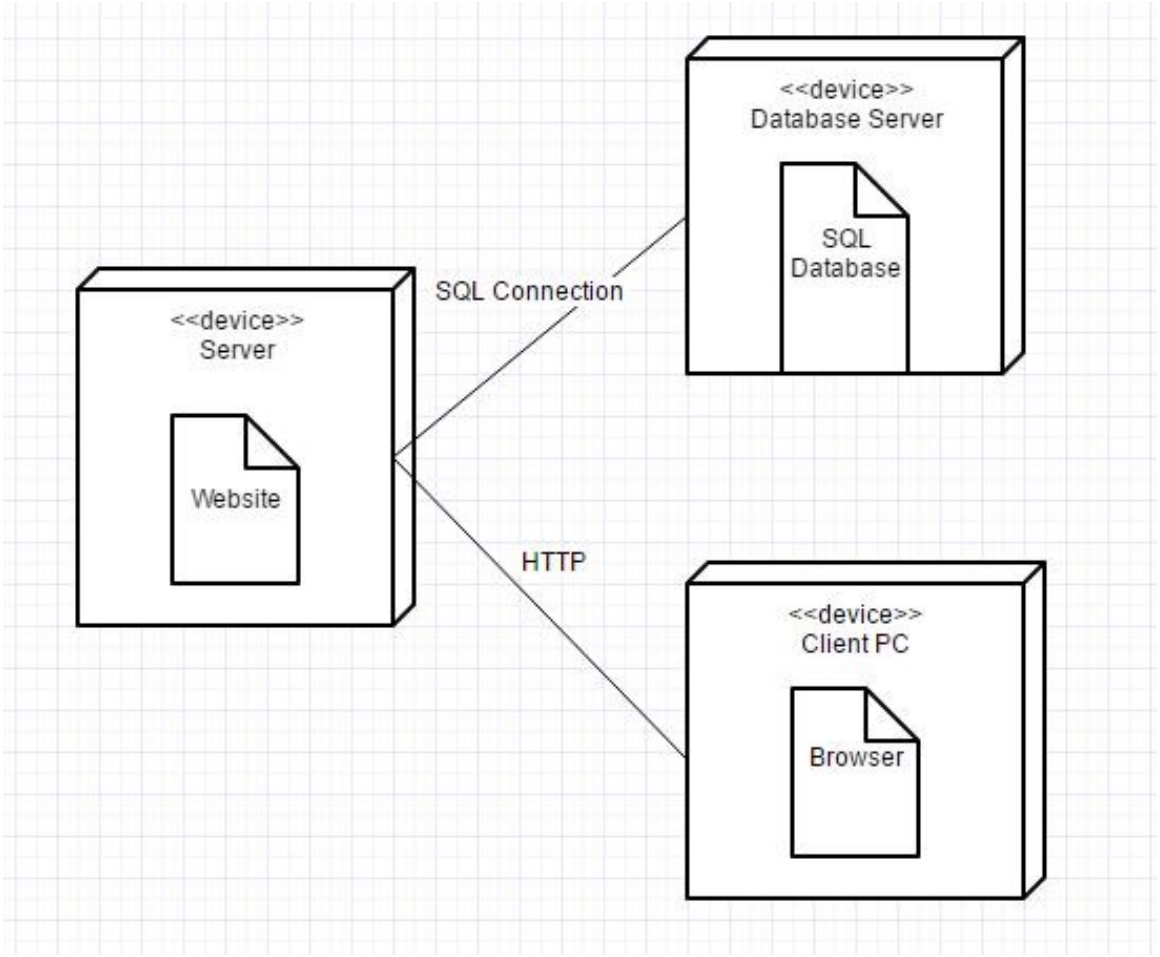


More UML Diagrams

Component Diagram



Deployment Diagram



Description of Patterns

Composite:

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 PC or 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. the classes inheriting form “System” will have an array list of components.

Observer:

The Observer Design pattern is used to keep viewers of data up to date as the data changes. The Data implements the Subject Interface and the viewer of the Data implements the Observer Interface. In our case the Subject is any Product and the Observer is the Product List. If an Admin wants to apply a discount to a product they go through the ApplyDiscountUI. This goes through the ApplyDiscount class that registers an instance of ProductList as an Observer of the Admin’s selected product. When the discount is applied, it changes the attributes in that Product. This requires the ProductList to be updated as it’s out of date. To do this, the notifyObservers method is called. This goes through the different Observers(just ProductList) and calls the Update method.

Factory:

The design pattern allows for the abstraction of implementation logic and allows this to be delegated to concrete subclasses. In our project, we applied this design pattern to help create types of objects that implement the Product class. When the application reads in all these different objects, it will not know when to instantiate a new object or what specific subclasses to create. With this design pattern, it creates a high cohesion and determines which exact subclass will be chosen and created at run-time. This method also demonstrates extensibility as the majority of the object creation will take place in the factory method itself, rather than in the application.

Decorator:

We used the Decorator design pattern when creating our order receipts that we display to the customer. The last UI that the customer will see is the OrderSummaryUI. This displays the products that they have bought, their price and the total cost. We used the Decorator pattern to create a header and footer note that get displayed at the top and bottom of the receipt. The header contains a thank you note and the footer a note about shipping charges. The OrderSummary class calls the header and shipping decorators through the use of the *Receipt* interface and the BasicReceipt concrete component class.

Concurrency Support

Even though we did not incorporate the state of concurrency in the final implementation of our project, we were still able to come up with a couple of ideas that would be able to implement this into future iterations. One example would be the threading of users being registered into the system, as well as checking for when more of more users and making comments to one of the same products. As for the threading of several products that are being bought by the customers, as well as other products are being added to the subsystem by the admin. For this there would be a faster response time to show the current available amount of stock that is remaining for that product. One other final piece to note for the issue of concurrency would be that of the threading of printing out receipts for customers. With several customers accessing the site and purchasing products simultaneously, the demand for confirming their purchase will want to be of a fast paste. With the running of several threads would be the best case to resolve this issue

Critique

- When comparing the artefacts in our analysis stage to what we had with our result, we noticed that the layout of the project remained the same. As for the longevity of our project, the concept we have provided will become inefficient as we continue to scale further up. The reason behind this is that using if/else statements in finding if group statements match would be considered poor practice for a major company's website. As well, in the observer design pattern, the action in which rewriting the entire text file should the user change a discount or price on a product is fine for now, but should the file contain a few thousand products, we should eventually move over to a proper SQL database as it would be able to handle this type of volume.
- During our first iteration, we decided on the concept of hard-coding in our products and customers into the system. This idea was not efficient due to the fact that in later iterations, we would have the idea of adding more products or customers to the system, making this concept difficult to implement. We created separate text files to store all the information in regards to orders, customers, etc. This implementation of the text file databases lead to our project being more scalable and reliable come our end result.
- Originally, we were incorporating all of our classes into one default package. We were then made clear of that we needed to separate our system models from our UI classes. After going over the MVC approach, we then separated our system into 3 separate packages, each demonstrating the concept of the Model, View and Control layers respectively. The separation of responsibilities allows more flexibility with our code in later iterations and easier to then implement our design patterns into the system.
- One critique that we had for the Composite design pattern is that the Product class that inherits from component is present and is not even abstract, as it has no reason to be instantiated. However, throughout the code the many Objects of type product were being created by the time we realised the error. Instead of refactoring a lot of the work we had done (which would have been disastrous at this late stage) we decided to accept it as it was. Although it does not follow the typical Composite design pattern it works just the same – just with one unnecessary class in between.
- Early on in our iteration, we developed the concept that the user would be permitted to have the choice of either buying individual products or buying packages of computer systems that already included the individual products. This idea seemed too complex to implement in our code, and with time permitting, we decided to go with the alternate. Currently, we now have the option for the user to personally create their own PC where the system requires certain types of subclasses chosen from the database (see implementation of factory class). This change was

beneficial towards our project as we were then able to implement the other design patterns into the system.

- Later on in the iterations, we included the concept of designing PCs and laptops. When purchasing the pieces to create these machines, the system requires certain subclasses in order create them. We thought of assigning each individual piece in the database their respected value, but this method would be too time consuming and complicated as the admin has the option of creating new products to be stored in the database. The Factory Design pattern was then implemented into the code to help of the assignment of the subclasses in the database. Using this design pattern demonstrated high cohesion and took away an unneeded tasks happening in the DataLayer class.

References

Logo Creation:

http://www.logomakr.com

Software Life Cycle Model Resources:

http://www.dummies.com/how-to/content/ten-benefits-of-agile-project-management.html
http://epmlive.com/5-benefits-of-agile/
https://cs.anu.edu.au/courses/comp3120/public_docs/WP_PM_AdvantagesofAgile.pdf
http://www.seguetech.com/blog/2013/04/12/8-benefits-of-agile-software-development
https://www.versionone.com/agile-101/agile-software-development-benefits/
http://www.allaboutagile.com/10-good-reasons-to-do-agile-development/
http://istqbexamcertification.com/what-is-agile-model-advantages-disadvantages-and-when-to-use-it/

Use Case Diagrams:

http://creately.com/diagram/example/hpczuu4u/Manufacturing%20company

Design Patterns:

https://github.com/bethrobson/Head-First-Design-Patterns
http://www.tutorialspoint.com/design_pattern/
http://www.journaldev.com/1540/decorator-pattern-in-java-example-tutorial

Creating our Diagrams:

http://creately.com/
https://www.gliffy.com/

Class Files

Not all classes have been appended due to the large amount of them. Important classes have been.

DataLayer Package

DatabaseInterface.java

```
package DataLayer;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;

import BusinessLayer.OrderClasses.GroupDiscount;
import BusinessLayer.OrderClasses.Order;
import BusinessLayer.ProductClasses.Product;

public interface DatabaseInterface {

	public int checkNextAvailableId(String filename) throws FileNotFoundException;
	public boolean checkProductStock(int productId) throws FileNotFoundException;
	public String [] checkAdminLogIn(String email, String password) throws FileNotFoundException;
	public String [] checkCustomerLogIn(String email, String password) throws FileNotFoundException;
	public void addCustomerToTextFile(int id, String firstName, String surname, String address, String email,
String password, String creditCardNumber) throws IOException;
	public void updateCustomerAddress(String newAddress, int customerId) throws IOException;
	public void updateCustomerCreditCardNumber(String newCustomerCreditCard, int customerId) throws
IOException;
	public boolean isProductNotAlreadyPresent(String productName) throws FileNotFoundException;
	public void writeNewProductToFile(String name, double cost, int stock, boolean isActive, boolean
isDiscount, String details) throws FileNotFoundException;
	public ArrayList<Product> factoryDesignPatternSearch() throws FileNotFoundException;
	public void reduceProductOrderStock(int [] orderProductIds) throws FileNotFoundException;
	public void rewriteProductFile(ArrayList<Product> products) throws IOException;
	public void writeNewCommentToFile(int customerId, String comment ,String customerName) throws
FileNotFoundException;
	public ArrayList<String> getComments(int productID) throws IOException;
	public double getDiscount(int productID) throws IOException;
	public void addDiscount(int productID , double discount) throws IOException;
	public ArrayList<Order> getAllOrdersFromFile() throws IOException;
	public void writeNewOrderToFile(int orderId, int customerId, String customerName, String orderProductIds)
throws FileNotFoundException;
	public void writeNewDiscountToFile(ArrayList<Integer> added , double discount) throws
FileNotFoundException;
	public ArrayList<GroupDiscount> getAllGroupDiscountsFromFile() throws FileNotFoundException;
}
```

DataControl.java

```
package DataLayer;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
```

```
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.ArrayList;
import java.util.Scanner;

import BusinessLayer.productFactoryDesign;
import BusinessLayer.OrderClasses.GroupDiscount;
import BusinessLayer.OrderClasses.Order;
import BusinessLayer.ProductClasses.CPU;
import BusinessLayer.ProductClasses.Desktop;
import BusinessLayer.ProductClasses.GPU;
import BusinessLayer.ProductClasses.Keyboard;
import BusinessLayer.ProductClasses.Laptop;
import BusinessLayer.ProductClasses.MemoryDrives;
import BusinessLayer.ProductClasses.Monitor;
import BusinessLayer.ProductClasses.Motherboard;
import BusinessLayer.ProductClasses.Mouse;
import BusinessLayer.ProductClasses.Product;
import BusinessLayer.ProductClasses.RAM;
import BusinessLayer.ProductClasses.Speaker;
import BusinessLayer.ProductClasses.Tablet;

public class DataControl implements DatabaseInterface {

	public final String customerFileName = "customerList.txt";
	public final String commentsFileName = "commentsList.txt";
	public final String adminFileName = "adminList.txt";
	public final String productFileName = "productList.txt";
	public final String ordersFileName = "ordersList.txt";
	public final String discountsFileName = "discountList.txt";
	public final String groupDiscountsFileName = "groupDiscountList.txt";

//
##########################################################################################################
##########################################
//		General Methods
//
##########################################################################################################
##########################################

	public 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 boolean checkProductStock(int productId) throws FileNotFoundException {
		boolean stockExists = true;

		File searchTextFile = new File(productFileName);
		Scanner lineIn = new Scanner(searchTextFile);
		while (lineIn.hasNextLine()) {
			String aLineFromFile = lineIn.nextLine();
			String [] splitLineFromFile = aLineFromFile.split(",");
			if (Integer.parseInt(splitLineFromFile[0]) == productId) {
				if (Integer.parseInt(splitLineFromFile[2]) < 1)
					stockExists = false;
			}
		}
		lineIn.close();
		return stockExists;
	}

//
##########################################################################################
##########################################
//	adminList.txt Methods
//
##########################################################################################
##########################################

	public String [] checkAdminLogIn(String email, String password) throws FileNotFoundException {
		String [] adminDetails = new String [5];

		File searchTextFile = new File(adminFileName);
		Scanner lineIn = new Scanner(searchTextFile);
		while (lineIn.hasNextLine()) {
			String aLineFromFile = lineIn.nextLine();
			adminDetails = aLineFromFile.split(",");
			if (adminDetails[3].equalsIgnoreCase(email) &&
adminDetails[4].equalsIgnoreCase(password)) {
				lineIn.close();
				return adminDetails;
			}
		}
		lineIn.close();
		for(int i = 0; i < adminDetails.length; i++)
			adminDetails[i] = "fail";

		return adminDetails;
	}

//
##########################################################################################
##########################################
//	customerList.txt Methods
//
##########################################################################################
##########################################

	public String [] checkCustomerLogIn(String email, String password) throws FileNotFoundException {
```

```
		String [] customerDetails = new String [7];

		File searchTextFile = new File(customerFileName);
		Scanner lineIn = new Scanner(searchTextFile);
		while (lineIn.hasNextLine()) {
			String aLineFromFile = lineIn.nextLine();
			customerDetails = aLineFromFile.split(",");
			if (customerDetails[4].equalsIgnoreCase(email) &&
customerDetails[5].equalsIgnoreCase(password)) {
				lineIn.close();
				return customerDetails;
			}
		}
		lineIn.close();
		for(int i = 0; i < customerDetails.length; i++)
			customerDetails[i] = "fail";

		return customerDetails;
	}

	public void addCustomerToTextFile(int id, String firstName, String surname, String address, String email,
String password, String creditCardNumber) throws IOException {
		String lineToAppend = "\n" + id + "," + firstName + "," + surname + "," + address + "," + email + ","
+ password + "," + creditCardNumber;
		try {
		   Files.write(Paths.get(customerFileName), lineToAppend.getBytes(),
StandardOpenOption.APPEND);
		}
		catch (IOException e) {
		   //exception handling left as an exercise for the reader
		}
	}

	public void updateCustomerAddress(String newAddress, int customerId) throws IOException {
		File searchTextFile = new File(customerFileName);
		Scanner lineIn = new Scanner(searchTextFile);
		while (lineIn.hasNextLine()) {
			String aLineFromFile = lineIn.nextLine();
			String [] splitLineFromFile = aLineFromFile.split(",");
			if (customerId == Integer.parseInt(splitLineFromFile[0])) {
				// Update text file
			}
		}
		lineIn.close();
	}

	public void updateCustomerCreditCardNumber(String newCustomerCreditCard, int customerId) throws
IOException {
		File searchTextFile = new File(customerFileName);
		Scanner lineIn = new Scanner(searchTextFile);
		while (lineIn.hasNextLine()) {
			String aLineFromFile = lineIn.nextLine();
			String [] splitLineFromFile = aLineFromFile.split(",");
			if (customerId == Integer.parseInt(splitLineFromFile[0])) {
				// Update text file
			}
		}
```

```
                lineIn.close();
        }

//
##########################################################################################
########################################
//          productList.txt Methods
//
##########################################################################################
########################################

        public boolean isProductNotAlreadyPresent(String productName) throws FileNotFoundException {
                boolean isProductNotPresent = true;

                File searchTextFile = new File(productFileName);
                Scanner lineIn = new Scanner(searchTextFile);
                while (lineIn.hasNextLine()) {
                        String aLineFromFile = lineIn.nextLine();
                        String [] splitLineFromFile = aLineFromFile.split(",");
                        if (productName.equals(splitLineFromFile[1])) {
                                isProductNotPresent = false;
                        }
                }
                lineIn.close();

                return isProductNotPresent;
        }

        public void writeNewProductToFile(String name, double cost, int stock, boolean isActive, boolean
isDiscount, String details) throws FileNotFoundException {
                int nextAvailableProductId = checkNextAvailableId(productFileName);
                String lineToAppend = "\n" + nextAvailableProductId + "," + name + "," + stock + "," + cost + "," +
isActive + "," + isDiscount + details;
                try {
                   Files.write(Paths.get(productFileName), lineToAppend.getBytes(),
StandardOpenOption.APPEND);
                }
                catch (IOException e) {
                   //exception handling left as an exercise for the reader
                }
        }

        public void reduceProductOrderStock(int [] orderProductIds) throws FileNotFoundException {
                File searchTextFile = new File(productFileName);
                Scanner lineIn = new Scanner(searchTextFile);
                for (int i = 0; i < orderProductIds.length; i++) {
                        while (lineIn.hasNextLine()) {
                                String aLineFromFile = lineIn.nextLine();
                                String [] splitLineFromFile = aLineFromFile.split(",");
                                if (orderProductIds[i] == Integer.parseInt(splitLineFromFile[0])) {
                                        if (Integer.parseInt(splitLineFromFile[0]) > 0) {
                                        // Update text file by reducing the stock for that product
                                        }
                                }
                        }
                }
                lineIn.close();
```

```
	}

	public void rewriteProductFile(ArrayList<Product> products) throws IOException{
		File file = new File(productFileName);
		FileWriter writer = new FileWriter(file);
		PrintWriter out = new PrintWriter(writer);
		Product p;
		for(int i =0; i < products.size();i++){
			System.out.println("[info]  : ------ Rewriting File (DataControl) ------");
			p= products.get(i);
			out.println(p.toString());
		}
		out.close();
	}

	public ArrayList<Product> factoryDesignPatternSearch() throws FileNotFoundException {
		File searchTextFile = new File(productFileName);
		ArrayList<Product> allFileProducts = new ArrayList<Product>();
		Scanner lineIn = new Scanner(searchTextFile);
		productFactoryDesign productFactory = new productFactoryDesign();
		while(lineIn.hasNextLine()){
			String aLineFromFile = lineIn.nextLine();
			if (aLineFromFile.length() > 10) {
				String[] splitLineFromFile = aLineFromFile.split(",");
				// Check there is stock
				if (Integer.parseInt(splitLineFromFile[2]) > 0) {
					switch(splitLineFromFile[1]){
					case "CPU":
						CPU cCPU =
productFactory.getCPU(Integer.parseInt(splitLineFromFile[0]), splitLineFromFile[1],
Integer.parseInt(splitLineFromFile[2]), Double.parseDouble(splitLineFromFile[3]),
Boolean.parseBoolean(splitLineFromFile[4]), Boolean.parseBoolean(splitLineFromFile[5]),
splitLineFromFile[6], splitLineFromFile[7],
Double.parseDouble(splitLineFromFile[8]));
						allFileProducts.add(cCPU);
						break;
					case "RAM":
						RAM cRAM =
productFactory.getRAM(Integer.parseInt(splitLineFromFile[0]), splitLineFromFile[1],
Integer.parseInt(splitLineFromFile[2]), Double.parseDouble(splitLineFromFile[3]),
Boolean.parseBoolean(splitLineFromFile[4]), Boolean.parseBoolean(splitLineFromFile[5])
,splitLineFromFile[6], splitLineFromFile[7],
Double.parseDouble(splitLineFromFile[8]));
						allFileProducts.add(cRAM);
						break;
					case "GPU":
						GPU cGPU =
productFactory.getGPU(Integer.parseInt(splitLineFromFile[0]), splitLineFromFile[1],
Integer.parseInt(splitLineFromFile[2]), Double.parseDouble(splitLineFromFile[3]),
Boolean.parseBoolean(splitLineFromFile[4]), Boolean.parseBoolean(splitLineFromFile[5])
,splitLineFromFile[6],
Boolean.parseBoolean(splitLineFromFile[7]), Integer.parseInt(splitLineFromFile[8]));
						allFileProducts.add(cGPU);
						break;
					case "Keyboard":
						Keyboard cKeyboard =
productFactory.getKeyboard(Integer.parseInt(splitLineFromFile[0]), splitLineFromFile[1],
```

```
Integer.parseInt(splitLineFromFile[2]), Double.parseDouble(splitLineFromFile[3]),
Boolean.parseBoolean(splitLineFromFile[4]), Boolean.parseBoolean(splitLineFromFile[5])
                                                                , splitLineFromFile[6],
Boolean.parseBoolean(splitLineFromFile[7]));
                                                        allFileProducts.add(cKeyboard);
                                                        break;
                                                case "MemoryDrives":
                                                        MemoryDrives cMemory =
productFactory.getMemory(Integer.parseInt(splitLineFromFile[0]), splitLineFromFile[1],
Integer.parseInt(splitLineFromFile[2]), Double.parseDouble(splitLineFromFile[3]),
Boolean.parseBoolean(splitLineFromFile[4]), Boolean.parseBoolean(splitLineFromFile[5])
                                                                ,
Boolean.parseBoolean(splitLineFromFile[6]), Integer.parseInt(splitLineFromFile[7]), splitLineFromFile[8]);
                                                        allFileProducts.add(cMemory);
                                                        break;
                                                case "Monitor":
                                                        Monitor cMonitor =
productFactory.getMonitor(Integer.parseInt(splitLineFromFile[0]), splitLineFromFile[1],
Integer.parseInt(splitLineFromFile[2]), Double.parseDouble(splitLineFromFile[3]),
Boolean.parseBoolean(splitLineFromFile[4]), Boolean.parseBoolean(splitLineFromFile[5])
                                                                ,splitLineFromFile[6],
Boolean.parseBoolean(splitLineFromFile[7]), Boolean.parseBoolean(splitLineFromFile[8]));
                                                        allFileProducts.add(cMonitor);
                                                        break;
                                                case "Mouse":
                                                        Mouse cMouse =
productFactory.getMouse(Integer.parseInt(splitLineFromFile[0]), splitLineFromFile[1],
Integer.parseInt(splitLineFromFile[2]), Double.parseDouble(splitLineFromFile[3]),
Boolean.parseBoolean(splitLineFromFile[4]), Boolean.parseBoolean(splitLineFromFile[5])
                                                                ,Integer.parseInt(splitLineFromFile[6]),
Boolean.parseBoolean(splitLineFromFile[7]), Boolean.parseBoolean(splitLineFromFile[8]));
                                                        allFileProducts.add(cMouse);
                                                        break;
                                                case "Speaker":
                                                        Speaker cSpeaker =
productFactory.getSpeaker(Integer.parseInt(splitLineFromFile[0]), splitLineFromFile[1],
Integer.parseInt(splitLineFromFile[2]), Double.parseDouble(splitLineFromFile[3]),
Boolean.parseBoolean(splitLineFromFile[4]), Boolean.parseBoolean(splitLineFromFile[5])
                                                                ,Integer.parseInt(splitLineFromFile[6]),
Boolean.parseBoolean(splitLineFromFile[7]), Boolean.parseBoolean(splitLineFromFile[8]),
Integer.parseInt(splitLineFromFile[9]));
                                                        allFileProducts.add(cSpeaker);
                                                        break;
                                                case "Laptop":
                                                        Laptop cLaptop =
productFactory.getLaptop(Integer.parseInt(splitLineFromFile[0]), splitLineFromFile[1],
Integer.parseInt(splitLineFromFile[2]), Double.parseDouble(splitLineFromFile[3]),
Boolean.parseBoolean(splitLineFromFile[4]), Boolean.parseBoolean(splitLineFromFile[5])
                                                                , splitLineFromFile[6],
Boolean.parseBoolean(splitLineFromFile[7]), Integer.parseInt(splitLineFromFile[8]),
Integer.parseInt(splitLineFromFile[9]), Double.parseDouble(splitLineFromFile[10]));
                                                        allFileProducts.add(cLaptop);
                                                        break;
                                                case "Tablet":
                                                        Tablet cTablet =
productFactory.getTablet(Integer.parseInt(splitLineFromFile[0]), splitLineFromFile[1],
```

```
Integer.parseInt(splitLineFromFile[2]), Double.parseDouble(splitLineFromFile[3]),
Boolean.parseBoolean(splitLineFromFile[4]), Boolean.parseBoolean(splitLineFromFile[5])
                                                                    , splitLineFromFile[6],
Boolean.parseBoolean(splitLineFromFile[7]), Integer.parseInt(splitLineFromFile[8]),
Integer.parseInt(splitLineFromFile[9]), Boolean.parseBoolean(splitLineFromFile[10]),
Double.parseDouble(splitLineFromFile[11]));
                                                    allFileProducts.add(cTablet);
                                                    break;
                                                case "Desktop":
                                                    Desktop cDesktop =
productFactory.getDesktop(Integer.parseInt(splitLineFromFile[0]), splitLineFromFile[1],
Integer.parseInt(splitLineFromFile[2]), Double.parseDouble(splitLineFromFile[3]),
Boolean.parseBoolean(splitLineFromFile[4]), Boolean.parseBoolean(splitLineFromFile[5])
                                                                    , splitLineFromFile[6],
Boolean.parseBoolean(splitLineFromFile[7]));
                                                    allFileProducts.add(cDesktop);
                                                    break;
                                                case "Motherboard":
                                                    Motherboard cMotherboard =
productFactory.getMotherboard(Integer.parseInt(splitLineFromFile[0]), splitLineFromFile[1],
Integer.parseInt(splitLineFromFile[2]), Double.parseDouble(splitLineFromFile[3]),
Boolean.parseBoolean(splitLineFromFile[4]), Boolean.parseBoolean(splitLineFromFile[5])
                                                                    , splitLineFromFile[6], splitLineFromFile[7],
splitLineFromFile[8], splitLineFromFile[9]);
                                                    allFileProducts.add(cMotherboard);
                                                    break;
                                                }
                                            }
                                        }
                                    }
                                    lineIn.close();
                                    return allFileProducts;
                                }

                                public  void writeNewCommentToFile(int customerId, String comment ,String customerName) throws
FileNotFoundException {
                                    int nextAvailableProductId = checkNextAvailableId(commentsFileName);
                                    String lineToAppend = "\n" + nextAvailableProductId + "," + customerId + "," + comment + "," +
customerName;
                                    try {
                                       Files.write(Paths.get(commentsFileName), lineToAppend.getBytes(),
StandardOpenOption.APPEND);
                                    }
                                    catch (IOException e) {
                                       //exception handling left as an exercise for the reader
                                    }
                                }

                                public  ArrayList<String> getComments(int productID) throws IOException{

                                    ArrayList<String> commentsInFile = new ArrayList<String>();
                                    File searchTextFile = new File(commentsFileName);
                                    Scanner lineIn = new Scanner(searchTextFile);
                                    String comment;
                                    while (lineIn.hasNextLine()) {
                                        String aLineFromFile = lineIn.nextLine();
                                        String [] splitLineFromFile = aLineFromFile.split(",");
```

```
                if (splitLineFromFile.length > 0) {
                        if(Integer.parseInt(splitLineFromFile[1]) == productID){
                                comment = splitLineFromFile[2];
                                comment += "\n-" + splitLineFromFile[3];
                                commentsInFile.add(comment);
                        }
                }
        }
        lineIn.close();

        return commentsInFile;
}

public double getDiscount(int productID) throws IOException{

        File searchTextFile = new File(discountsFileName);
        Scanner lineIn = new Scanner(searchTextFile);
        while (lineIn.hasNextLine()) {
                String aLineFromFile = lineIn.nextLine();
                String [] splitLineFromFile = aLineFromFile.split(",");
                if (splitLineFromFile.length > 0) {
                        if(Integer.parseInt(splitLineFromFile[0]) == productID){
                                return Double.parseDouble(splitLineFromFile[1]);
                        }
                }
        }
        lineIn.close();

        return 0;
}

public void addDiscount(int productID , double discount) throws IOException{

        File searchTextFile = new File(discountsFileName);
        Scanner lineIn = new Scanner(searchTextFile);
        boolean found = false;
        ArrayList<String> newFile = new ArrayList<String>();
        while (lineIn.hasNextLine()) {
                String aLineFromFile = lineIn.nextLine();
                String [] splitLineFromFile = aLineFromFile.split(",");
                if (splitLineFromFile.length > 0) {
                        if(Integer.parseInt(splitLineFromFile[0]) == productID){
                                newFile.add(productID + "," + discount);
                                found = true;
                        }
                        else
                                newFile.add(aLineFromFile);
                }
        }

        if(found){
                FileWriter writer = new FileWriter(searchTextFile);
                PrintWriter out = new PrintWriter(writer);
                for(int i =0; i < newFile.size();i++){
                        out.println(newFile.get(i));
                }
                out.close();
```

```
                }
                else{
                        String lineToAppend = "\n" + productID + "," + discount;
                        try {
                           Files.write(Paths.get(discountsFileName), lineToAppend.getBytes(),
StandardOpenOption.APPEND);
                        }
                        catch (IOException e) {
                           //exception handling left as an exercise for the reader
                        }
                }
                lineIn.close();
        }

//
##########################################################################################
########################################
//ordersList.txt Methods
//
##########################################################################################
########################################

        public void writeNewOrderToFile(int orderId, int customerId, String customerName, String orderProductIds)
throws FileNotFoundException {
                int nextAvailableProductId = checkNextAvailableId(ordersFileName);
                String lineToAppend = "\n" + nextAvailableProductId + "," + orderId + "," + customerId + "," +
customerName + "," + orderProductIds;
                try {
                   Files.write(Paths.get(ordersFileName), lineToAppend.getBytes(),
StandardOpenOption.APPEND);
                }
                catch (IOException e) {
                   //exception handling left as an exercise for the reader
                }
        }

        public void writeNewDiscountToFile(ArrayList<Integer> added , double discount) throws
FileNotFoundException {

                int nextAvailableDiscountId = checkNextAvailableId(groupDiscountsFileName);
                String IDs = "";
                for(int i =0;i < added.size();i++)
                        IDs += added.get(i) + ",";
                String lineToAppend = "\n" + nextAvailableDiscountId + "," + added.size() + "," + IDs + discount;
                try {
                   Files.write(Paths.get(groupDiscountsFileName), lineToAppend.getBytes(),
StandardOpenOption.APPEND);
                }
                catch (IOException e) {
                   //exception handling left as an exercise for the reader
                }
        }

        public ArrayList<GroupDiscount> getAllGroupDiscountsFromFile() throws FileNotFoundException {

                ArrayList<GroupDiscount> allDiscountsInFile = new ArrayList<GroupDiscount>();
                File searchTextFile = new File(groupDiscountsFileName);
```

```
		Scanner lineIn = new Scanner(searchTextFile);
		int prices;
		int i;
		while (lineIn.hasNextLine()) {
			String aLineFromFile = lineIn.nextLine();
			String [] splitLineFromFile = aLineFromFile.split(",");
			if (splitLineFromFile.length > 0) {
				ArrayList<Integer> productsInDiscount = new ArrayList<Integer>();
				prices = Integer.parseInt(splitLineFromFile[1]);
				for( i = 0; i< prices;i++){
					productsInDiscount.add(Integer.parseInt(splitLineFromFile[2 + i]));
				}

				GroupDiscount lineDiscount = new
GroupDiscount(Integer.parseInt(splitLineFromFile[0]), productsInDiscount, Double.parseDouble(splitLineFromFile[i
+2]));
				allDiscountsInFile.add(lineDiscount);
			}
		}
		lineIn.close();

		return allDiscountsInFile;
	}

	public ArrayList<Order> getAllOrdersFromFile() throws IOException {

		ArrayList<Order> allOrdersInFile = new ArrayList<Order>();
		File searchTextFile = new File(ordersFileName);
		Scanner lineIn = new Scanner(searchTextFile);
		int prices;
		int i;
		while (lineIn.hasNextLine()) {
			String aLineFromFile = lineIn.nextLine();
			String [] splitLineFromFile = aLineFromFile.split(",");
			if (splitLineFromFile.length > 0) {
				ArrayList<Product> productsInOrder = new ArrayList<Product>();
				int orderID = Integer.parseInt(splitLineFromFile[0]);
				int shippingID = Integer.parseInt(splitLineFromFile[1]);
				int customerID = Integer.parseInt(splitLineFromFile[2]);
				String customerName = splitLineFromFile[3];
				String[] orderProducts = splitLineFromFile[4].split(" ");
				for( i = 0; i< orderProducts.length;i++){
					productsInOrder.add(new Product(Integer.parseInt(orderProducts[i]) , ""
, 0 , 0 ,true , false));
				}

				Order lineOrder = new Order(orderID, customerID, customerName , shippingID ,
productsInOrder);
				allOrdersInFile.add(lineOrder);
			}
		}
		lineIn.close();

		return allOrdersInFile;
	}
}
```

BusinessLayer Customer Sub Package

User.java

```
package BusinessLayer.CustomerClasses;

public class User {

	private String firstName;
	private String surname;
	private String email;
	private String password;

	public User(String firstName, String surname , String email, String password){
		this.firstName = firstName;
		this.surname = surname;
		this.email = email;
		this.password = password;
	}

	public String getFirstName() {
		return firstName;
	}

	public void setFirstName(String name) {
		this.firstName = name;
	}

	public String getSurname() {
		return surname;
	}

	public void setSurname(String name) {
		this.surname = name;
	}

	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}

	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
		this.password = password;
	}
}
```

Admin.java

```
package BusinessLayer.CustomerClasses;

public class Admin extends User{

	int adminID;

	public Admin(int adminID , String firstName, String surname, String email , String password ){
		super(firstName, surname , email , password);
		this.adminID = adminID;
	}

	public int getAdminID() {
		return adminID;
	}
}
```

Customer.java

```
package BusinessLayer.CustomerClasses;

import java.util.ArrayList;

public class Customer extends User {

	private int customerId;
	private String creditCardNumber;
	private String customerAddress;
	private ArrayList<Integer> customerOrders;

	public Customer(int id, String firstName, String surname, String address, String email, String password, String
creditCardNumber){
		super(firstName, surname ,email , password);
		this.customerId = id;
		this.customerAddress = address;
		this.creditCardNumber = creditCardNumber;


	}

	public String getCustomerAddress() {
		return customerAddress;
	}

	public void setCustomerAddress(String customerAddress) {
		this.customerAddress = customerAddress;
	}

	public String getCreditCardNumber() {
		return creditCardNumber;
	}

	public void setCreditCardNumber(String creditCardNumber) {
		this.creditCardNumber = creditCardNumber;
	}

	public String getCustomersOrders(){
		String customerOrderIdList = "";
		for (int i = 0; i < customerOrders.size(); i++) {
			customerOrderIdList += customerOrders.get(i);
		}
		if (customerOrderIdList == "") return "No orders for this customer";
		return customerOrderIdList;
```

```
	}

	public void addToCustomersOrders(int orderId) {
		customerOrders.add(orderId);
	}

	public int getCustomerId() {
		return customerId;
	}

	public void setCustomerId(int customerId) {
		this.customerId = customerId;
	}
}
```

BusinessLayer Order Sub Package

Order.java

```
package BusinessLayer.OrderClasses;

import java.util.ArrayList;
import java.util.Date;
import java.io.IOException;

import BusinessLayer.ProductClasses.Product;

@SuppressWarnings("unused")
public class Order {

	private OrderDetail orderDetail;
	private ShippingInfo shippingInfo;

	private int orderId;
	private int customerId;
	private int shippingId;
	private String customerName;
	private String dateOrderCreated;
	private String dateOrderShipped;

	private boolean isNewOrder;
	private boolean isProcessedOrder;
	private boolean isCancelledOrder;

	public Order(int orderId, int customerId, String customerName, int shippingId, ArrayList<Product>
orderProducts) throws IOException {

		this.orderDetail = new OrderDetail(orderId, orderProducts);
		this.shippingInfo = new ShippingInfo(shippingId);

		this.orderId = orderId;
		this.customerId = customerId;
		this.shippingId = shippingId;
		this.customerName = customerName;
		Date date = new Date();
		this.dateOrderCreated = date.toString();
		this.isNewOrder = true;
	}

// ShippingInfo Methods
```

```
	public String getShippingInfo() {
		return shippingInfo.getShippingInfo();
	}

	public void setShippingCost(double shippingCost) {
		shippingInfo.setShippingCost(shippingCost);
	}

	public void setShippingRegionId(int shippingRegionId) {
		shippingInfo.setShippingRegionId(shippingRegionId);
	}

// OrderDetail Methods
	public double getTotalCost() {
		return orderDetail.getTotalCost();
	}

	public int [] getOrderProductIds() {
		return orderDetail.getOrderProductIds();
	}

	public String getOrderProductIdsToString() {
		return orderDetail.getOrderProductIdsToString();
	}

	public String getPreOrderDetails() {
		String orderInfo = orderDetail.getOrderDetails();
		return orderInfo;
	}

	public String getPostOrderDetails() {
		String orderInfo = orderDetail.getOrderDetails() +
				"<br>Customer: " + customerName +
				"<br>" + getOrderStatus() +
				"<br>Date Order Created: " + dateOrderCreated;
		if (dateOrderShipped != null) orderInfo += "<br>Date Order Shipped: " + dateOrderShipped;
		return orderInfo;
	}

	public String [] getOrderProductNames() {
		return orderDetail.getOrderProductNames();
	}

	public ArrayList<Product> getProducts() {
		return orderDetail.getProducts();
	}

	public void setProducts(ArrayList<Product> orderProducts) {
		orderDetail.setProducts(orderProducts);
	}

// Order Methods
	public int getOrderId() {
		return orderId;
	}

	public String getOrderCreateDate() {
```

```
		return dateOrderShipped;
	}

	public String getOrderShippingDate() {
		return dateOrderShipped;
	}

	public void setOrderShippingDate() {
		Date date = new Date();
		this.dateOrderShipped = date.toString();
	}

	public void setOrderShippingDate(String dateOrderShipped) {
		this.dateOrderShipped = dateOrderShipped;
	}

	public void setProcessOrder() {
		this.isNewOrder = false;
		this.isProcessedOrder = true;
		this.isCancelledOrder = false;
	}

	public void setCancelOrder() {
		this.isNewOrder = false;
		this.isProcessedOrder = false;
		this.isCancelledOrder = true;
	}

	public String getOrderStatus() {
		String orderStatusString = "";
		if (true == isNewOrder)		orderStatusString = "Order has been created!";
		if (true == isProcessedOrder)	orderStatusString = "Order is being processed!";
		if (true == isCancelledOrder)	orderStatusString = "Order has been cancelled!";
		return orderStatusString;
	}

	public int getCustomerID() {
		return this.customerId;
	}
}
```

OrderDetail.java

```
package BusinessLayer.OrderClasses;

import java.util.ArrayList;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.text.DecimalFormat;

import BusinessLayer.BusinessLayerDataControl;
import BusinessLayer.ProductClasses.Product;
import DataLayer.DataControl;

public class OrderDetail {

	private int orderId;
	private ArrayList<Product> orderProducts;
```

```
	private double totalCost;
	private boolean isGroupDiscount;
	private double maxDiscount;

	OrderDetail (int orderId, ArrayList<Product> orderProducts) throws IOException {
		this.orderId = orderId;
		this.orderProducts = orderProducts;
		this.isGroupDiscount = false;
		for (int i = 0; i < orderProducts.size(); i++)
			this.totalCost += orderProducts.get(i).getUnitCost();


	}

	public int [] getOrderProductIds() {
		int [] orderProductIds = new int [orderProducts.size()];
		for (int i = 0; i < orderProducts.size(); i++) {
			orderProductIds[i] = orderProducts.get(i).getProductId();
		}
		return orderProductIds;
	}

	public String getOrderProductIdsToString() {
		String productIdString = "";
		for (int i = 0; i < orderProducts.size(); i++) {
			productIdString += orderProducts.get(i).getProductId() + " ";
		}
		return productIdString;
	}

	public String [] getOrderProductNames() {
		String [] orderProductNames = new String [orderProducts.size()];
		for (int i = 0; i < orderProducts.size(); i++) {
			orderProductNames[i] = orderProducts.get(i).getProductName();
		}
		return orderProductNames;
	}

	public double getTotalCost() {
		return totalCost;
	}

	public String getOrderDetails() {
		String returnString = "Order ID: " + orderId;

		for (int i = 0; i < orderProducts.size(); i++) {
			returnString += "<br>Product ID:" + orderProducts.get(i).getProductId();
			returnString += "<br>Product Name:" + orderProducts.get(i).getProductName();
			try {
				if(orderProducts.get(i).getDiscount() == 0)
					returnString += "<br>Product Price:" +
orderProducts.get(i).getUnitCost() + "<br>";
				else{
					returnString += "<br>Discount Applied:" +
orderProducts.get(i).getDiscount() + "%";
					returnString += "<br>Product Price:" + new
DecimalFormat("##.##").format(orderProducts.get(i).getUnitCost()) + "<br>";
```

```
                        }
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }

                if(this.isGroupDiscount)
                    returnString += "<br>Group Discount of " + maxDiscount  + "% applied.<br>";

                returnString += "<br>TotalCost: " +  new DecimalFormat("##.##").format(totalCost) + "<br>";
                return returnString;
            }

            public ArrayList<Product> getProducts() {
                return orderProducts;
            }

            public void setProducts(ArrayList<Product> orderProducts) {
                this.orderProducts = orderProducts;
                this.totalCost = 0.0;
                for (int i = 0; i < orderProducts.size(); i++)
                    totalCost += orderProducts.get(i).getUnitCost();
                BusinessLayerDataControl dataControl = new BusinessLayerDataControl();
                ArrayList<GroupDiscount> groupDiscounts;
                try {
                    groupDiscounts = dataControl.getAllGroupDiscountsFromFile();
                    maxDiscount = 0;
                    GroupDiscount g;
                    for(int j = 0;j < groupDiscounts.size();j++){
                        g = groupDiscounts.get(j);
                        if(g.areProductsValid(orderProducts)){
                            if(g.getDiscount() > maxDiscount)
                                maxDiscount = g.getDiscount();
                                this.isGroupDiscount = true;
                            //System.out.println("Found Discount");
                        }
                     }
                     totalCost = totalCost * (1 - (maxDiscount / 100));

                    // Add shipping cost
                    this.totalCost += 5.0;
                } catch (FileNotFoundException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }

}
```

Receipt.java

```
package BusinessLayer.OrderClasses;

abstract public class Receipt {

    abstract public String printReceipt();
}
```

ReceiptDecorator.java

```
package BusinessLayer.OrderClasses;

abstract public 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;
	}
}
```

BasicReceipt.java

```
package BusinessLayer.OrderClasses;

public class BasicReceipt extends Receipt {

	@Override
	public String printReceipt() {
		return "<br>Note: ";
	}
}
```

HeaderReceipt.java

```
package BusinessLayer.OrderClasses;

public class HeaderReceipt extends ReceiptDecorator {

	public HeaderReceipt(Receipt receipt) {
		super(receipt);
	}

	@Override
	public String printReceipt() {
		return super.printReceipt() + "<br>DCM Computing<br>";
	}
}
```

BusinessLayer Product Sub Package

Component.java

```
package BusinessLayer.ProductClasses;

public interface Component {

    public int getProductId();
    public void setProductId(int newProductId);
    public String getProductName();
    public void setProductName(String newProductName);
    public int getStock();
    public void setStock(int newStock);
```

```
    public double getUnitCost();
    public void setUnitCost(double newUnitCost);
    public void setProductActive();
    public void setProductDiscount(double unitCost);
    public String getProductStatus();
    public String getProductDetails();
}
```

Product.java

```
package BusinessLayer.ProductClasses;

import java.io.IOException;
import java.text.DecimalFormat;
import java.util.ArrayList;

import BusinessLayer.BusinessLayerDataControl;
import BusinessLayer.Observer;
import BusinessLayer.Subject;

/* This class models a Product that can be sold in
 * the on-line e-commerce system example.
 */

public class Product implements Component, Subject {

    private int productId;
    private String productName;
    private boolean isActive;
    private boolean isProductDiscount;
    protected double unitCost;
    private int stock;
    private ArrayList<Observer> observers;

    /**
     * Construct a new product using the provided item
     * and price.
     *
     * @param productId the productId of the product.
     * @param stock the amount left to sell for the product
     * @param isActive states whether the product is active or not.
     * @param isProductDiscount states whether the product has a discount set for it
     * @param productName the productName of the product.
     * @param unitCostthe unitCostfor which this Product should be sold.
     */
    public Product(int productId, String productName, int stock, double unitCost, boolean isActive, boolean
isProductDiscount) {
        this.productId = productId; //Generate new Id function should be used here
        this.productName = productName;
        this.stock = stock;
        this.unitCost= unitCost;
        this.isActive = isActive;
        this.isProductDiscount = isProductDiscount;
        observers = new ArrayList<Observer>();
    }

    /**
     * Get the ID of this Product.
```

```
     *
     * @return the ID of this Product.
     */
    public int getProductId() {
        return productId;
    }
    public void setProductId(int newProductId) {
        productId =  newProductId;
    }

    /**
     * Get the productName of this Product.
     *
     * @return the productName of this Product.
     */
    public String getProductName() {
        return productName;
    }
    public void setProductName(String newProductName) {
        productName =  newProductName;
    }

    /**
     * Get the stock of this Product.
     *
     * @return the stock of this Product.
     */
    public int getStock() {
        return stock;
    }
    public void setStock(int newStock) {
        stock =  newStock;
    }

    /**
     * Get the unitCostof this Product.
     *
     * @return the inventory number of this Product.
     */
    public double getUnitCost() {
            if(isProductDiscount){
                    try {
                            return unitCost * (1 - (getDiscount() / 100));
                    } catch (IOException e) {
                            // TODO Auto-generated catch block
                            return unitCost;
                    }
            }
            else{
                    return unitCost;
            }
    }

    public void setUnitCost(double newUnitCost) {
        this.unitCost= newUnitCost;
    }
```

```
    public void setProductActive()
    {
        this.isActive = true;
    }

    public void setProductDiscount(double discount) {
        this.isProductDiscount =  true;
        try {
            BusinessLayerDataControl dataControl = new BusinessLayerDataControl();
                    dataControl.addDiscount(productId, discount);
            } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
            }
    }

    public void removeDiscount(){
            this.isProductDiscount = false;
    }

    public String getProductStatus()
    {
        String statusString = "";
        if(true == isActive)            statusString = "Product is available";
        else if(true == isProductDiscount)  statusString = "Product is on sale";
        return statusString;
    }

    public double getDiscount() throws IOException{
            BusinessLayerDataControl dataControl = new BusinessLayerDataControl();
            return dataControl.getDiscount(productId);
    }

    /**
     * Get a String that describes this Product. Note that
     * this method overrides the toString method inherited
     * from Object.
     *
     * @return a String describing this Product.
     */
    public String getProductDetails() {
        return "\nProduct ID: " + this.productId + "\nProduct Name: " + this.productName +"\nPrice: " + getUnitCost() +
"\nStock: " + this.stock + "\n";
    }

    public String getProductUIDetails(){
        return  this.productName + "  \u20ac" + new DecimalFormat("##.##").format(getUnitCost()) + "  " + this.stock;
    }
    public String toString(){
        return "" + this.productId + "," + this.productName +","  + this.stock + "," + this.unitCost + "," + this.isActive + ","
                + this.isProductDiscount;


    }

    @Override
    public void registerObserver(Observer o) {
```

```
        observers.add(o);

    }

    @Override
    public void notifyObservers() throws IOException {
        Observer o;
        try {
            for(int i = 0; i < observers.size();i++ )
            {
                o = observers.get(i);
                o.Update(this);
            }
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

System.java

```
package BusinessLayer.ProductClasses;

import java.util.ArrayList;

public interface System {

    public void addProduct(Product component);
    public ArrayList<Product> getComponents();
    public void removeProduct(Product component);
    public Product getChildatIndex(int i);
}
```

ComputerSystem.java

```
package BusinessLayer.ProductClasses;

import java.util.ArrayList;

public class ComputerSystem extends Product implements System{

    private String OS;
    private ArrayList<Product> components;

    public ComputerSystem(int productId, String productName, int stock, double unitCost, boolean isActive, boolean
isProductDiscount, String OS) {
        super(productId, productName, stock, unitCost, isActive, isProductDiscount);
        this.OS = OS;
        components = new ArrayList<Product>();
    }

    public void addProduct(Product component) {
            components.add(component);
    }

    public ArrayList<Product> getComponents() {
        return components;

    }

    public void removeProduct(Product component) {
            components.remove(component);
    }
```

```
    public Product getChildatIndex(int i) {
        return components.get(i);
    }

    public String toString(){
            return super.toString() + "," + OS;
    }

    public String getOS() {
        return OS;
    }

    public void setOS(String OS) {
        this.OS = OS;
    }
}
```

Laptop.java

```
package BusinessLayer.ProductClasses;

public class Laptop extends ComputerSystem {
    private boolean gamingLaptop;
    private int screenSize;
    private int screenRez;
    private double weight;


    public Laptop(int productId, String productName, int stock, double unitCost, boolean isActive, boolean
isProductDiscount, String OS, boolean gamingLaptop, int screenSize, int screenRez, double weight) {
        super(productId, productName, stock, unitCost, isActive, isProductDiscount, OS);
        this.gamingLaptop = gamingLaptop;
        this.screenSize = screenSize;
        this.screenRez = screenRez;
        this.weight = weight;
    }

    public String toString() {
            return super.toString() + "," + this.gamingLaptop + "," + this.screenSize + "," + this.screenRez + "," +
this.weight;
    }

    public boolean isGamingLaptop() {
        return gamingLaptop;
    }

    public void setGamingLaptop(boolean gamingLaptop) {
        this.gamingLaptop = gamingLaptop;
    }

    public double getWeight() {
        return weight;
    }

    public void setWeight(double weight) {
        this.weight = weight;
    }

    public int getScreenRez() {
```

```
        return screenRez;
    }

    public void setScreenRez(int screenRez) {
        this.screenRez = screenRez;
    }

    public int getScreenSize() {
        return screenSize;
    }

    public void setScreenSize(int screenSize) {
        this.screenSize = screenSize;
    }

    public double getUnitCost() {
            if(!this.getComponents().isEmpty()){
                    for(int i = 0; i<this.getComponents().size();i++) {
                            unitCost = unitCost + this.getComponents().get(i).unitCost;
                    }
            }
            return unitCost;
    }
}
```

CPU.java

```
package BusinessLayer.ProductClasses;

public class CPU extends Product {
    private String series;
    private String CPUSocketType;
    private double OperatingFrequency;

    public CPU(int productId, String productName, int stock, double unitCost, boolean isActive, boolean
isProductDiscount, String series, String CPUSocketType, double operatingFrequency) {
        super(productId, productName, stock, unitCost, isActive, isProductDiscount);
        this.series = series;
        this.CPUSocketType = CPUSocketType;
        this.OperatingFrequency = operatingFrequency;
    }

    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 double getOperatingFrequency() {
        return OperatingFrequency;
    }

    public void setOperatingFrequency(double operatingFrequency) {
        OperatingFrequency = operatingFrequency;
    }

    public String getSpecs() {
            return "Series: " + series + " Socket Type: " + CPUSocketType + " Operating Frequency: "  +
OperatingFrequency;
    }

    public String toString(){
             return super.toString() + "," + this.series + "," + this.CPUSocketType +","  + this.OperatingFrequency;
    }
}
```

BusinessLayer Package

runProject.java

```
package BusinessLayer;

import javax.swing.JOptionPane;
import UserInterfaceLayer.LoginUI;

@SuppressWarnings("unused")
public class runProject {

            public static void main(String[] args) {

                        try {
                                    LoginUI frame=new LoginUI();
                        }
                        catch(Exception e) {
                                    JOptionPane.showMessageDialog(null, e.getMessage());
                        }
            }

}
```

Login.java

```
package BusinessLayer;

import UserInterfaceLayer.*;
import DataLayer.DataControl;
import javax.swing.*;

import BusinessLayer.CustomerClasses.Admin;
import BusinessLayer.CustomerClasses.Customer;
import BusinessLayer.ProductClasses.Product;

import java.io.*;
import java.util.ArrayList;

// Control Class
public class Login {
```

```
	public Login(String email, String password) throws IOException {
		System.out.println("[debug] : ****** Entering Login Class ******");

		BusinessLayerDataControl dataControl = new BusinessLayerDataControl();
	    String [] customerDetails = dataControl.checkCustomerLogIn(email, password);
	    String [] adminDetails = dataControl.checkAdminLogIn(email, password);

	    if (!(customerDetails[6].equalsIgnoreCase("fail"))) {
		// Customer(int id, String firstName, String surname, String email, String password, String
creditCardNumber)
		Customer customer = new
Customer(Integer.parseInt(customerDetails[0]),customerDetails[1],customerDetails[2],customerDetails[3],customerD
etails[4], customerDetails[5], customerDetails[6]);
		@SuppressWarnings("unused")

		ArrayList<Product> productsInFile = dataControl.factoryDesignPatternSearch();

				ProductListUI frame = new ProductListUI(customer, productsInFile);
	    }
	    else if (!(adminDetails[4].equalsIgnoreCase("fail"))) {
		// Admin(int adminID , String firstName, String surname, String email , String password )
		Admin admin = new
Admin(Integer.parseInt(adminDetails[0]),adminDetails[1],adminDetails[2],adminDetails[3],adminDetails[4]);
		@SuppressWarnings("unused")
				ProductListUI frame = new ProductListUI(admin);
	    }
	    else
		JOptionPane.showMessageDialog(null,"User not found",
"Error",JOptionPane.ERROR_MESSAGE);
	    }
}
```

Register.java

```
package BusinessLayer;

import UserInterfaceLayer.LoginUI;
import UserInterfaceLayer.RegisterUI;
import DataLayer.DataControl;
import javax.swing.*;
import java.util.*;
import java.io.*;

@SuppressWarnings("unused")
//Control Class
public class Register
{
	public Register(String firstName, String surname, String address, String email, String pass1, String pass2,
String creditCardNumber) {
		System.out.println("[debug] : ****** Entering Register Class ******");

		 if(pass1.compareTo(pass2) != 0) {
	JOptionPane.showMessageDialog(null,"Passwords do not match","Error",JOptionPane.ERROR_MESSAGE);
	RegisterUI createNewRegisterUI = new RegisterUI();
		}

      else if(!isValidEmailAddress(email)) {
```

```
        JOptionPane.showMessageDialog(null,"Not a valid email address","Error",JOptionPane.ERROR_MESSAGE);
        RegisterUI createNewRegisterUI = new RegisterUI();
                        }

       else {
         try {
                          BusinessLayerDataControl dataControl = new BusinessLayerDataControl();
              int nextAvailableId = dataControl.checkNextAvailableId("customerList.txt");

               dataControl.addCustomerToTextFile(nextAvailableId, firstName, surname, address, email, pass1,
creditCardNumber);

            JOptionPane.showMessageDialog(null, "You have successfully registered. Please log in");
            LoginUI l = new LoginUI();
         }

         catch (IOException e)
       {
          e.printStackTrace();
       }
       }
            }

            public static boolean isValidEmailAddress(String email)
            {
                        String ePattern = "^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@((\\[[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]
{1,3}\\])|(([a-zA-Z\\-0-9]+\\.)+[a-zA-Z]{2,}))$";
       java.util.regex.Pattern p = java.util.regex.Pattern.compile(ePattern);
       java.util.regex.Matcher m = p.matcher(email);
       return m.matches();
   }
}
```

Observer.java

```
package BusinessLayer;

import java.io.IOException;

import BusinessLayer.ProductClasses.Product;

public interface Observer {

            public void Update(Product p) throws IOException;
}
```

Subject.java

```
package BusinessLayer;

import java.io.IOException;

public interface Subject {

            public void registerObserver(Observer O);
            public void notifyObservers() throws IOException;
}
```

ProductList.java

```
package BusinessLayer;

import java.util.ArrayList;
import java.io.IOException;

import BusinessLayer.CustomerClasses.Customer;
import BusinessLayer.OrderClasses.Order;
import BusinessLayer.ProductClasses.Product;
import DataLayer.DataControl;
import UserInterfaceLayer.CartUI;

public class ProductList implements Observer {

	public ProductList(ArrayList<Product> pickedProducts, Customer customer) throws IOException {

		// Order(int orderId, int customerId, String customerName, int shippingId, ArrayList<Product>
orderProducts)

		BusinessLayerDataControl dataControl = new BusinessLayerDataControl();
		int orderId = dataControl.checkNextAvailableId("ordersList.txt");
		int customerId = customer.getCustomerId();
		String customerName = customer.getFirstName() + customer.getSurname();
		int shippingId = orderId;

		Order customerOrder = new Order(orderId, customerId, customerName, shippingId,
pickedProducts);

		@SuppressWarnings("unused")
		CartUI createNewCartUi = new CartUI(customerOrder,customer);
	}

	public ProductList(){

	}
	@Override
	public void Update(Product p) throws IOException {
		BusinessLayerDataControl dataControl = new BusinessLayerDataControl();
		ArrayList<Product> x =dataControl.factoryDesignPatternSearch();
		for(int i =0;i < x.size();i++){
			if(x.get(i).getProductId() == p.getProductId()){
				x.remove(i);
				x.add(i , p);
			}
		}
		dataControl.rewriteProductFile(x);


	}
}
```

ApplyDiscount.java

```
package BusinessLayer;

import BusinessLayer.ProductClasses.Product;
import DataLayer.DataControl;
```

```
import java.io.*;
import java.util.ArrayList;


public class ApplyDiscount {

	public ApplyDiscount(Product p) throws IOException{
		ProductList r = new ProductList();
		p.registerObserver(r);
	}

	public void applyDiscount(Product p, double discount) throws IOException {
		p.setProductDiscount(discount);
		p.notifyObservers();
	}

	public void removeDiscount(Product p) throws IOException {
		p.removeDiscount();
		p.notifyObservers();
	}
}
```

OrderSummary.java

```
package BusinessLayer;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;

import BusinessLayer.OrderClasses.*;
import BusinessLayer.CustomerClasses.Customer;
import DataLayer.DataControl;
import BusinessLayer.OrderClasses.Order;
import BusinessLayer.ProductClasses.Product;
import UserInterfaceLayer.OrderSummaryUI;

public class OrderSummary {

	@SuppressWarnings("unused")
	public OrderSummary(Order userOrder, Customer currentCustomer) throws IOException {
		System.out.println("[debug] : ****** Entering OrderSummary Class ******");

			// Write the new order to ordersList.txt
			int orderId = userOrder.getOrderId();
			int customerId = currentCustomer.getCustomerId();
			String customerName = currentCustomer.getFirstName() + " " + currentCustomer.getSurname();
			String orderProductIds = userOrder.getOrderProductIdsToString();
			BusinessLayerDataControl dataControl = new BusinessLayerDataControl();
			dataControl.writeNewOrderToFile(orderId, customerId, customerName, orderProductIds);

			// Reduce the stock amount of the products the customer is buying
			dataControl.reduceProductOrderStock(userOrder.getOrderProductIds());
			int[] orderProductIDs = userOrder.getOrderProductIds();
			ArrayList<Product> myProducts = new ArrayList<Product>();
			ArrayList<Product> allProducts = dataControl.factoryDesignPatternSearch();
			for(int i = 0; i < orderProductIDs.length;i++){
				for(int j = 0;j < allProducts.size(); j++){
```

```
                                        if(orderProductIDs[i] == allProducts.get(j).getProductId()){
                                                myProducts.add(allProducts.get(j));

                                        }
                                }
                        }
                        ProductList listOfProducts = new ProductList();
                        for(int i =0; i < myProducts.size();i++){
                                myProducts.get(i).registerObserver(listOfProducts);
                                myProducts.get(i).setStock(myProducts.get(i).getStock() - 1);
                                myProducts.get(i).notifyObservers();
                        }

                        // Use Decorator Design pattern for printing receipt
                        Receipt headerReceipt = new ThankYouReceipt(new HeaderReceipt(new BasicReceipt()));
                        String postOrderDetails = headerReceipt.printReceipt();

                        postOrderDetails += userOrder.getPostOrderDetails();

                        Receipt shippingReceipt = new ShippingReceipt(new BasicReceipt());
                        postOrderDetails += shippingReceipt.printReceipt();

                        OrderSummaryUI createNewOrderSummaryUI = new OrderSummaryUI(postOrderDetails);

        }
}
```

productFactoryDesign.java

```
package BusinessLayer;

import BusinessLayer.ProductClasses.*;

public class productFactoryDesign {

        public CPU getCPU(int productId, String productName, int stock, double unitCost, boolean isActive, boolean
isProductDiscount, String series, String CPUSocketType, double operatingFrequency){
                return new CPU(productId, productName, stock, unitCost, isActive, isProductDiscount, series,
CPUSocketType, operatingFrequency);
        }

        public RAM getRAM(int productId, String productName, int stock, double unitCost, boolean isActive, boolean
isProductDiscount, String type, String multiChannel, double casLatency){
                return new RAM(productId, productName, stock, unitCost, isActive, isProductDiscount, type,
multiChannel, casLatency);
        }

        public GPU getGPU(int productId, String productName, int stock, double unitCost, boolean isActive, boolean
isProductDiscount, String memory, boolean virtualRealityReady, int displayPorts){
                return new GPU(productId, productName, stock, unitCost, isActive, isProductDiscount, memory,
virtualRealityReady, displayPorts);
        }

        public Keyboard getKeyboard(int productId, String productName, int stock, double unitCost, boolean isActive,
boolean isProductDiscount, String keySwitchType, boolean backlit){
                return new Keyboard(productId, productName, stock, unitCost, isActive, isProductDiscount,
keySwitchType, backlit);
        }

        public MemoryDrives getMemory(int productId, String productName, int stock, double unitCost, boolean
isActive, boolean isProductDiscount, boolean desktopType, int driveCapacity, String RPM){
```

```
		return new MemoryDrives(productId, productName, stock, unitCost, isActive, isProductDiscount,
desktopType, driveCapacity, RPM);
	}

	public Monitor getMonitor(int productId, String productName, int stock, double unitCost, boolean isActive,
boolean isProductDiscount, String screenResolution, boolean curved, boolean is3d){
		return new Monitor(productId, productName, stock, unitCost, isActive, isProductDiscount,
screenResolution, curved, is3d);
	}

	public Mouse getMouse(int productId, String productName, int stock, double unitCost, boolean isActive, boolean
isProductDiscount, int dpi, boolean programmableButtons, boolean dpiSwitching){
		return new Mouse(productId, productName, stock, unitCost, isActive, isProductDiscount, dpi,
programmableButtons, dpiSwitching);
	}

	public Speaker getSpeaker(int productId, String productName, int stock, double unitCost, boolean isActive,
boolean isProductDiscount, int watts, boolean desktopControls, boolean subwoofer, int satelliteSpeakers){
		return new Speaker(productId, productName, stock, unitCost, isActive, isProductDiscount, watts,
desktopControls, subwoofer, satelliteSpeakers);
	}

	public Tablet getTablet(int productId, String productName, int stock, double unitCost, boolean isActive, boolean
isProductDiscount, String OS, boolean stylus, int screenSize, int screenRez, boolean attachableKeyboard, double
weight){
		return new Tablet(productId, productName, stock, unitCost, isActive, isProductDiscount, OS, stylus,
screenSize, screenRez, attachableKeyboard, weight);
	}

	public Laptop getLaptop(int productId, String productName, int stock, double unitCost, boolean isActive,
boolean isProductDiscount, String OS, boolean gamingLaptop, int screenSize, int screenRez, double weight){
		return new Laptop(productId, productName, stock, unitCost, isActive, isProductDiscount, OS,
gamingLaptop, screenSize, screenRez, weight);
	}

	public Desktop getDesktop(int productId, String productName, int stock, double unitCost, boolean isActive,
boolean isProductDiscount, String OS, boolean gamingSystem){
		return new Desktop(productId, productName, stock, unitCost, isActive, isProductDiscount, OS,
gamingSystem);
	}

	public Motherboard getMotherboard(int productId, String productName, int stock, double unitCost, boolean
isActive, boolean isProductDiscount, String cpuSocketType, String series, String formFactor, String memoryStandard){
		return new Motherboard(productId, productName, stock, unitCost, isActive, isProductDiscount,
cpuSocketType, series, formFactor, memoryStandard);
	}
}
```

CreatePC.java

```
package BusinessLayer;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;

import javax.swing.DefaultListModel;

import BusinessLayer.CustomerClasses.Customer;
import BusinessLayer.ProductClasses.Component;
import BusinessLayer.ProductClasses.Desktop;
import BusinessLayer.ProductClasses.Product;
```

```
public class CreatePC {

	private ArrayList<Product> productsInFile;
	private DefaultListModel<String> model;
	private DefaultListModel<String> modelChosen;
	private Desktop currentDesk= new Desktop(1234, "Boaty McBoatFace", 1, 0.0, true, false, "Windows",
true);
	private Customer currentCustomer;
	public int indexThroughComponentsToChoose = 0;
	private boolean chooseArrayMade = false;
	public boolean chosenArrayMade = false;
	public boolean backButtonClicked = false;



	public CreatePC(Customer c) throws IOException{
		currentCustomer = c;
		ArrayList<Product> productsInFile;
		Desktop currentDesk= new Desktop(1234, "Boaty McBoatFace", 1, 0.0, true, false, "Windows",
true);
		int indexThroughComponentsToChoose = 0;
		boolean chooseArrayMade = false;
		boolean chosenArrayMade = false;
		boolean backButtonClicked = false;

	}

	public ArrayList<Product> getProductsInFile() {
		try {
			BusinessLayerDataControl dataControl = new BusinessLayerDataControl();
			productsInFile = dataControl.factoryDesignPatternSearch();
		}
		catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		return productsInFile;
	}

	public void buyButtonClicked() {
		System.out.println("[info]  : ------ Buy button clicked (CreatePCUI Customer) ------");
		try {
			@SuppressWarnings("unused")
			ProductList createNewProductList = new ProductList(currentDesk.getComponents(),
currentCustomer);
		}
		catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

	}

	public void addButtonClicked(int indexOfSelectedComponents) {
		System.out.println("[info]  : ------ Add button clicked (CreatePCUI Customer) ------");
		chosenArrayMade = true;
		System.out.println(indexThroughComponentsToChoose);
```

```
		String part ="";
		switch(indexThroughComponentsToChoose) {
		case 0 : part = "Motherboard"; break;
		case 1 : part = "CPU"; break;
		case 2 : part = "GPU"; break;
		case 3 : part = "RAM"; break;
		case 4 : part = "MemoryDrives"; break;
		case 5 : part = "Monitor"; break;
		case 6 : part = "Keyboard"; break;
		case 7 : part = "Mouse"; break;
		case 8 : part = "Speaker"; break;
		}
		int y =0;
		int u = 0;

		if (indexOfSelectedComponents == -1){
			indexOfSelectedComponents = 0;
		}
		System.out.println(indexOfSelectedComponents);

		while (y < getProductsInFile().size() && u <=indexOfSelectedComponents) {

			System.out.println("y"+y);
			Component someProduct = getProductsInFile().get(y);
			if(someProduct.getProductName().equals(part)) {
				u++;
			}
			y++;
		}
		currentDesk.addProduct(productsInFile.get(y-1));
		indexThroughComponentsToChoose++;

}

public void backBtnClicked() {
		if(chosenArrayMade){
			currentDesk.getComponents().remove(currentDesk.getComponents().size() - 1);
			indexThroughComponentsToChoose--;
			System.out.println("BACK " +indexThroughComponentsToChoose );
			backButtonClicked = true;
		}

}

public ArrayList<Product> getListOfComponents() {

		return currentDesk.getComponents();
}

public String getlabel(String part) {
		String labelPart = "";
		switch(part) {
		case "Motherboard" : labelPart = "Motherboard"; break;
		case "CPU" : labelPart = "CPU"; break;
		case "GPU" : labelPart = "GPU"; break;
		case "RAM" : labelPart = "RAM"; break;
		case "MemoryDrives" : labelPart = "Memory Drive"; break;
```

```
                case "Monitor" : labelPart = "Monitor"; break;
                case "Keyboard" : labelPart = "Keyboard"; break;
                case "Mouse" : labelPart = "Mouse"; break;
                case "Spreaker" : labelPart = "Speaker System"; break;
        }
                return labelPart;

        }
}
```

BusinessLayerDataControl.java

```
package BusinessLayer;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;

import BusinessLayer.OrderClasses.GroupDiscount;
import BusinessLayer.OrderClasses.Order;
import BusinessLayer.ProductClasses.Product;
import DataLayer.DataControl;
import DataLayer.DatabaseInterface;

public class BusinessLayerDataControl implements DatabaseInterface {

        private DataControl dataControl = new DataControl();

        public int checkNextAvailableId(String filename) throws FileNotFoundException {
                return dataControl.checkNextAvailableId(filename);
        }

        public boolean checkProductStock(int productId) throws FileNotFoundException {
                return dataControl.checkProductStock(productId);
        }

        public String [] checkAdminLogIn(String email, String password) throws FileNotFoundException {
                return dataControl.checkAdminLogIn(email, password);
        }

        public String[] checkCustomerLogIn(String email, String password) throws FileNotFoundException {
                return dataControl.checkCustomerLogIn(email, password);
        }

        public void addCustomerToTextFile(int id, String firstName, String surname, String address, String email,
String password, String creditCardNumber) throws IOException {
                dataControl.addCustomerToTextFile(id, firstName, surname, address, email, password,
creditCardNumber);
        }

        public void updateCustomerAddress(String newAddress, int customerId) throws IOException {
                dataControl.updateCustomerAddress(newAddress, customerId);
        }

        public void updateCustomerCreditCardNumber(String newCustomerCreditCard, int customerId) throws
IOException {
                dataControl.updateCustomerCreditCardNumber(newCustomerCreditCard, customerId);
        }
```

```
	public boolean isProductNotAlreadyPresent(String productName) throws FileNotFoundException {
		return dataControl.isProductNotAlreadyPresent(productName);
	}

	public void writeNewProductToFile(String name, double cost, int stock, boolean isActive, boolean
isDiscount, String details) throws FileNotFoundException {
		dataControl.writeNewProductToFile(name, cost, stock, isActive, isDiscount, details);
	}

	public void reduceProductOrderStock(int [] orderProductIds) throws FileNotFoundException {
		dataControl.reduceProductOrderStock(orderProductIds);
	}

	public void rewriteProductFile(ArrayList<Product> products) throws IOException {
		dataControl.rewriteProductFile(products);
	}

	public ArrayList<Product> factoryDesignPatternSearch() throws FileNotFoundException {
		return dataControl.factoryDesignPatternSearch();
	}

	public void writeNewCommentToFile(int customerId, String comment ,String customerName) throws
FileNotFoundException {
		dataControl.writeNewCommentToFile(customerId, comment, customerName);
	}

	public ArrayList<String> getComments(int productID) throws IOException {
		return dataControl.getComments(productID);
	}

	public void addDiscount(int productId, double discount) throws IOException {
		dataControl.addDiscount(productId, discount);
	}

	public double getDiscount(int productId) throws IOException {
		return dataControl.getDiscount(productId);
	}

	public ArrayList<Order> getAllOrdersFromFile() throws IOException {
		return dataControl.getAllOrdersFromFile();
	}

	public void writeNewOrderToFile(int orderId, int customerId, String customerName, String orderProductIds)
throws FileNotFoundException {
		dataControl.writeNewOrderToFile(orderId, customerId, customerName, orderProductIds);
	}

	public void writeNewDiscountToFile(ArrayList<Integer> added , double discount) throws
FileNotFoundException {
		dataControl.writeNewDiscountToFile(added, discount);
	}

	public ArrayList<GroupDiscount> getAllGroupDiscountsFromFile() throws FileNotFoundException {
		return dataControl.getAllGroupDiscountsFromFile();
	}
}
```

AddNewProduct.java

```
package BusinessLayer;

import DataLayer.DataControl;
import java.io.*;

public class AddNewProduct {

	private String name;
	private double cost;
	private int stock;


	public boolean validateProduct(String name) throws IOException{

		boolean valid = true;
		BusinessLayerDataControl dataControl = new BusinessLayerDataControl();
		valid = dataControl.isProductNotAlreadyPresent(name);
		//return valid; -> Overriding return boolean as we are allowing products with the same name
		return true;
	}

	public String process(String inName , double inCost , int inStock) throws IOException{
		name = inName;
		cost = inCost;
		stock = inStock;

		return this.name;

	}

	public void addProduct(String details) throws IOException{
		BusinessLayerDataControl dataControl = new BusinessLayerDataControl();
		dataControl.writeNewProductToFile(name, cost, stock, true, false, details);
	}
}
```

AddNewComment.java

```
package BusinessLayer;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;

import BusinessLayer.CustomerClasses.Customer;
import BusinessLayer.OrderClasses.Order;
import BusinessLayer.ProductClasses.Product;
import DataLayer.DataControl;

public class AddNewComment {

	public ArrayList<Product> getProducts(Customer c) throws IOException{
		BusinessLayerDataControl dataControl = new BusinessLayerDataControl();
		ArrayList<Product> allProducts = dataControl.factoryDesignPatternSearch();
		ArrayList<Order> allOrders = dataControl.getAllOrdersFromFile();
		ArrayList<Product> myProductsID = new ArrayList<Product>();
		ArrayList<Product> orderProducts = new ArrayList<Product>();
```

```
		for(int i = 0; i < allOrders.size();i++){
			if(allOrders.get(i).getCustomerID() == c.getCustomerId()){
				orderProducts = allOrders.get(i).getProducts();
				for(int j = 0;j < orderProducts.size();j++){
					myProductsID.add(orderProducts.get(j));
				}
			}
		}

		for(int i = 0; i < myProductsID.size() - 1;i++){
			for(int j = i + 1;j < myProductsID.size(); j++){
				if(myProductsID.get(i).getProductId() == myProductsID.get(j).getProductId()){
					myProductsID.remove(j);
					j--;
				}
			}
		}

		ArrayList<Product> myProducts = new ArrayList<Product>();
		for(int i = 0; i < myProductsID.size();i++){
			for(int j = 0;j < allProducts.size(); j++){
				if(myProductsID.get(i).getProductId() == allProducts.get(j).getProductId()){
					myProducts.add(allProducts.get(j));
				}
			}
		}

		return myProducts;

	}

	public void addComment(String text , int ID , String name) throws IOException{
		BusinessLayerDataControl dataControl = new BusinessLayerDataControl();
		dataControl.writeNewCommentToFile(ID, text, name);
	}
}
```

UserInterface Package

LoginUI.java

```
package UserInterfaceLayer;

import BusinessLayer.*;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
import java.util.*;
import java.io.*;

@SuppressWarnings({ "unused", "serial" })
public class LoginUI extends JFrame implements ActionListener
{
	JButton SUBMIT, REGISTER;
	JPanel panel;
	JLabel label1,label2;
```

```
        final JTextField  text1,text2;
        JPanel inputControls;
        JPanel labelPanel = new JPanel(new GridLayout(0,1,3,3));
JPanel fieldPanel = new JPanel(new GridLayout(0,1,3,3));
JFrame averageFrame;

        public LoginUI() {
                System.out.println("[debug] : ****** Entering LoginUI Class ******");

                inputControls = new JPanel(new BorderLayout(5,5));
                inputControls.add(labelPanel, BorderLayout.WEST);
        inputControls.add(fieldPanel, BorderLayout.CENTER);

                label1 = new JLabel();
                label1.setText("Email Address:");
                text1 = new JTextField(15);

                label2 = new JLabel();
                label2.setText("Password:");
                text2 = new JPasswordField(15);

                labelPanel.add(label1);
                fieldPanel.add(text1);
                labelPanel.add(label2);
                fieldPanel.add(text2);

                SUBMIT=new JButton("Submit");
                REGISTER=new JButton("Register");

                SUBMIT.setActionCommand("1");
                REGISTER.setActionCommand("2");

                JPanel controls = new JPanel(new FlowLayout(FlowLayout.CENTER,5,2));
                controls.add(SUBMIT);
   controls.add(REGISTER);

                SUBMIT.addActionListener(this);
                REGISTER.addActionListener(this);

                JPanel gui = new JPanel(new BorderLayout(10,10));
   gui.setBorder(new TitledBorder("Please Log In"));
   gui.add(inputControls, BorderLayout.CENTER);
   gui.add(controls, BorderLayout.SOUTH);

   averageFrame = new JFrame("Login Page");
   averageFrame.setContentPane(gui);
   averageFrame.pack();
   averageFrame.setLocationByPlatform(true);
   averageFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   averageFrame.setVisible(true);
        }

        public void actionPerformed(ActionEvent ae)
        {
                int action = Integer.parseInt(ae.getActionCommand());
                String enteredEmail=text1.getText();
                String enteredPassword=text2.getText();
```

```
    if(action == 1)
    {
      try {
                        Login l = new Login(enteredEmail,enteredPassword);
                }
      catch (FileNotFoundException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
      catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
    }
    else
    {
       text1.setText("");
       text2.setText("");
       averageFrame.setVisible(false);
       RegisterUI frame=new RegisterUI();
       frame.setSize(300,600);
       frame.setVisible(false);
     }
   }
}
```

ProductListUI.java

```
package UserInterfaceLayer;

import BusinessLayer.*;
import BusinessLayer.CustomerClasses.Admin;
import BusinessLayer.CustomerClasses.Customer;
import BusinessLayer.ProductClasses.Product;
import DataLayer.DataControl;

import java.util.ArrayList;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.TitledBorder;

import java.io.*;

@SuppressWarnings({ "serial" })
public class ProductListUI extends JFrame {
        public final static ArrayList<Product> pickedProducts = new ArrayList<Product>();

// ********************************************************************************************************************************
// ************        Admin ProductListUI
// ********************************************************************************************************************************

        public ProductListUI(final Admin admin) throws IOException {
                System.out.println("[debug] : ****** Entering ProductListUI Class as Admin ******");

                final JFrame ProductListUiFrame = new JFrame("Product List");
```

```
                JButton addNewProductButton = new JButton("Add New Product");
                addNewProductButton.addActionListener(new ActionListener() {
                    @Override
                    public void actionPerformed(ActionEvent evt) {
                            System.out.println("[info]  : ------ Add New Product button clicked (ProductListUI Admin)
------");

                        try {
                            AddNewProductUI add = new AddNewProductUI(admin);
                                    add.display();
                                    }
                        catch (IOException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                    }
                    }
                });

                JButton addNewProductDiscount = new JButton("Add New Product Discount");
                addNewProductDiscount.addActionListener(new ActionListener() {
                    @Override
                    public void actionPerformed(ActionEvent evt) {
                            System.out.println("[info]  : ------ Add New Product Discount button clicked (ProductListUI
Admin) ------");

                        try {
                            ApplyDiscountUI add = new ApplyDiscountUI(admin);
                                    }
                        catch (IOException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                    }
                    }
                });
                JButton quitButton = new JButton("Quit");
                quitButton.addActionListener(new ActionListener() {
                    @Override
                    public void actionPerformed(ActionEvent evt) {
                            System.out.println("[info]  : ------ Quit button clicked (ProductListUI Admin) ------");
                            System.exit(0);
                    }
                });
                JButton addNewGroupDiscount = new JButton("Add New Group Discount");
                addNewGroupDiscount.addActionListener(new ActionListener() {
                    @Override
                    public void actionPerformed(ActionEvent evt) {
                            System.out.println("[info]  : ------ Add New Group Discount button clicked (ProductListUI
Admin) ------");

                        try {
                            AddNewGroupDiscountUI add = new AddNewGroupDiscountUI(admin);
                                    }
                        catch (IOException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                        }
                            }
```

```
                });

        JPanel controls = new JPanel(new FlowLayout(FlowLayout.CENTER,5,2));
        controls.setBorder(new TitledBorder("You are logged in as an Admin user"));
        controls.add(addNewGroupDiscount);
        controls.add(addNewProductDiscount);
        controls.add(addNewProductButton);
    controls.add(quitButton);

    ProductListUiFrame.setContentPane(controls);
    ProductListUiFrame.pack();
    ProductListUiFrame.setLocationByPlatform(true);
    ProductListUiFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    ProductListUiFrame.setSize(800, 600);
    ProductListUiFrame.setResizable(false);
    ProductListUiFrame.setLocationRelativeTo(null);
    ProductListUiFrame.setVisible(true);
    }

// ***************************************************************************************************************************
// ************        Customer ProductListUI
// ***************************************************************************************************************************

    private JList choices;
    private ArrayList<Product> productsAddedToCart;
    private DefaultListModel<String> model;

    public ProductListUI(final Customer customer, ArrayList<Product> productsInFile) throws IOException {
        super("Products");

        System.out.println("[debug] : ****** Entering ProductListUI Class as Customer ******");

        Container pane = getContentPane();
        pane.setBackground(new Color(0,100,200));
        pane.setLayout(new BorderLayout(5,5));
        JLabel productsListJLabel = new JLabel("Products");
        pane.add(productsListJLabel, BorderLayout.NORTH);
        productsListJLabel.setFont(new Font("Dialog",Font.BOLD,20));
        JPanel m = new JPanel();
        model = new DefaultListModel<String>();

        for(int i = 0; i < productsInFile.size(); i++){
                Product someProduct = productsInFile.get(i);
                model.addElement(someProduct.getProductUIDetails());
                choices = new JList(model);
        }

        m.add(choices);
        pane.add(m, BorderLayout.WEST);
        m.setFont(new Font("Courier",Font.BOLD,14));

        JButton buyB = new JButton("Buy");
        buyB.addActionListener(new ActionListener() {
           @Override
           public void actionPerformed(ActionEvent evt) {
                System.out.println("[info]  : ------ Buy button clicked (ProductListUI Customer) ------");
```

```
                        try {
                                displayList();
                        }
                        catch (IOException e1) {
                                // TODO Auto-generated catch block
                                e1.printStackTrace();
                        }
                }
            });
            m.add(buyB);

            JButton createPcB = new JButton("Create Desktop");
            createPcB.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent evt) {
                        System.out.println("[info]  : ------ Create PC button clicked (ProductListUI Customer)
------");

                        try {
                                CreatePCUI createPCUI = new CreatePCUI(customer);
                        }
                        catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
            });
            m.add(createPcB);

            JButton createLaptopB = new JButton("Create Laptop");
            createLaptopB.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent evt) {
                        System.out.println("[info]  : ------ Create PC button clicked (ProductListUI Customer)
------");

                        try {
                                CreateLaptopUI createLaptopUI = new CreateLaptopUI(customer);
                        }
                        catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
            });
            m.add(createLaptopB);

            JButton createComment = new JButton("Add Comment");
            createComment.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent evt) {
                        System.out.println("[info]  : ------ Add Comment button clicked (ProductListUI Customer)
------");
                                try {
                                        AddNewCommentUI addComment = new
AddNewCommentUI(customer);
                                        addComment.display();
```

```
                                } catch (IOException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
                        }
                });
                m.add(createComment);

                JButton getComments = new JButton("Look at Comments");
                getComments.addActionListener(new ActionListener() {
                    @Override
                    public void actionPerformed(ActionEvent evt) {
                            System.out.println("[info]  : ------ Look Comments button clicked (ProductListUI Customer)
------");
                                try {
                                        GetCommentsUI getComment = new GetCommentsUI();
                                        int selected = choices.getSelectedIndex();

                                        if(choices.getSelectedIndex() >= 0){
                                                BusinessLayerDataControl dataControl = new
BusinessLayerDataControl();
                                                productsAddedToCart = new
ArrayList<Product>(dataControl.factoryDesignPatternSearch());
                                                Product p = productsAddedToCart.get(selected);
                                                getComment.display(p);
                                        }
                                }
                                catch (IOException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
                    }
                });
                m.add(getComments);

                JButton getGroupDiscounts = new JButton("Look at Group Discounts");
                 getGroupDiscounts.addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent evt) {
                                System.out.println("[info]  : ------ Get Discounts button clicked (ProductListUI
Customer) ------");
                                try {
                                        GetGroupDiscountsUI getDiscounts = new GetGroupDiscountsUI();
                                        getDiscounts.display();
                                }
                                catch (IOException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
                        }
                 });
                 m.add(getGroupDiscounts);

                JPanel p = new JPanel();

                JButton cartB = new JButton("Check Out");
                cartB.addActionListener(new ActionListener() {
```

```
                @Override
                public void actionPerformed(ActionEvent evt) {
                        System.out.println("[info]  : ------ Check Out button clicked (ProductListUI Customer)
------");

                                try {
                                        ProductList createNewProductList = new ProductList(pickedProducts,
customer);
                                }
                                catch (IOException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
                    }
                });

                JButton logOutB = new JButton("Log Out");
                logOutB.addActionListener(new ActionListener() {
                    @Override
                    public void actionPerformed(ActionEvent evt) {
                        System.out.println("[info]  : ------ Log Out button clicked (ProductListUI Customer) ------");

                        System.exit(0);
                    }
                });

                p.add(cartB);
                p.add(logOutB);
                add(p, BorderLayout.CENTER);

                setSize(1200, 700);
                setVisible(true);
                setDefaultCloseOperation(EXIT_ON_CLOSE);
        }

        private void displayList() throws IOException {

                int selected = choices.getSelectedIndex();

                BusinessLayerDataControl dataControl = new BusinessLayerDataControl();
                productsAddedToCart = new ArrayList<Product>(dataControl.factoryDesignPatternSearch());
                Product p = productsAddedToCart.get(selected);
                pickedProducts.add(p);
                for(int i = 0; i < pickedProducts.size(); i++)
                        System.out.println(pickedProducts.get(i).getProductDetails());
        }
}
```

AddNewCommentUI.java

```
package UserInterfaceLayer;

import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;

import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.border.TitledBorder;

import BusinessLayer.AddNewComment;
import BusinessLayer.CustomerClasses.Customer;
import BusinessLayer.ProductClasses.Product;
import DataLayer.DataControl;

public class AddNewCommentUI {

	private JList choices;
	private ArrayList<Product> customerProduct;
	private DefaultListModel<String> model;
	private Customer c;

	public AddNewCommentUI(Customer c){
		this.c = c;
	}

	public void display() throws IOException{
		JButton SUBMIT;
		JPanel panel;
		JLabel label0 , label1;
		final JLabel label2;
		final JTextField  text1;

		JPanel inputControls;

		JPanel labelPanel = new JPanel(new GridLayout(0,1,3,3));
	   JPanel fieldPanel = new JPanel(new GridLayout(0,1,3,3));
	   JPanel addedPanel = new JPanel(new GridLayout(0,1,3,3));

	   inputControls = new JPanel(new BorderLayout(5,5));
		inputControls.add(labelPanel, BorderLayout.WEST);
	inputControls.add(fieldPanel, BorderLayout.CENTER);
	inputControls.add(addedPanel, BorderLayout.SOUTH);

		label1 = new JLabel();
		label1.setText("Please enter your Comment");
		text1 = new JTextField(15);

		labelPanel.add(label1);

		final JFrame averageFrame = new JFrame("Comment Page");

		JPanel m = new JPanel();
		model = new DefaultListModel<String>();
```

```
                AddNewComment a = new AddNewComment();
                customerProduct = a.getProducts(c);
                choices = new JList(model);
                for(int i = 0; i < customerProduct.size(); i++){
                        Product someProduct = customerProduct.get(i);
                        model.addElement(someProduct.getProductUIDetails());
                        choices = new JList(model);
                }
                m.add(choices);

        SUBMIT=new JButton("Submit");

        SUBMIT.addActionListener(new ActionListener(){
                @Override
                public void actionPerformed(ActionEvent e){
                                int selected = choices.getSelectedIndex();
                                if(selected >= 0){
                                        AddNewComment a = new AddNewComment();
                                        Product p = customerProduct.get(selected);
                                        try {
                                                a.addComment(text1.getText(), p.getProductId() ,
c.getFirstName() + " " + c.getSurname());
                                                averageFrame.setVisible(false);
                                        }
                                        catch (IOException e1) {
                                                // TODO Auto-generated catch block
                                                e1.printStackTrace();
                                        }
                                }
                }
        });

                SUBMIT.setActionCommand("1");

        JPanel controls = new JPanel(new FlowLayout(FlowLayout.CENTER,5,2));
        if(customerProduct.size() > 0){
                 fieldPanel.add(text1);
                 controls.add(SUBMIT);
        }
        else
                 label1.setText("You cannot add a comments as you have bought no items");


                JPanel gui = new JPanel(new BorderLayout(10,10));
    gui.setBorder(new TitledBorder("Add Comment"));
    gui.add(inputControls, BorderLayout.CENTER);
    gui.add(controls, BorderLayout.SOUTH);

                gui.add(m, BorderLayout.WEST);
                m.setFont(new Font("Courier",Font.BOLD,14));

    averageFrame.setContentPane(gui);
    averageFrame.pack();
    averageFrame.setLocationByPlatform(true);
    averageFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    averageFrame.setVisible(true);
```

```
	}
}
```

AddNewProductUI.java (Partial)

```
package UserInterfaceLayer;

import java.io.IOException;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;

import BusinessLayer.AddNewProduct;
import BusinessLayer.CustomerClasses.Admin;


public class AddNewProductUI {

	private JButton SUBMIT;
	private JLabel label0 , label1 , label2 , label3;
	private JTextField  text1 , text2 , text0 , text3;
	private JPanel labelPanel, fieldPanel;
	private Admin adminUser;

	JPanel inputControls;

	public AddNewProductUI(Admin a){
		this.adminUser = a;
		labelPanel = new JPanel(new GridLayout(0,1,3,3));
	   fieldPanel = new JPanel(new GridLayout(0,1,3,3));

	   inputControls = new JPanel(new BorderLayout(5,5));
		inputControls.add(labelPanel, BorderLayout.WEST);
	inputControls.add(fieldPanel, BorderLayout.CENTER);

	}

	public void display() throws IOException{

		JPanel inputControls;

		JPanel labelPanel = new JPanel(new GridLayout(0,1,3,3));
	   JPanel fieldPanel = new JPanel(new GridLayout(0,1,3,3));

	   inputControls = new JPanel(new BorderLayout(5,5));
		inputControls.add(labelPanel, BorderLayout.WEST);
	inputControls.add(fieldPanel, BorderLayout.CENTER);


	label0 = new JLabel();
	label0.setText("Enter Product Name:");
		label1 = new JLabel();
		label1.setText("Enter Cost:");
		label2 = new JLabel();
	label2.setText("Stock");
	text0 = new JTextField(15);
		text1 = new JTextField(15);
```

```
                text2 = new JTextField(15);

JFrame averageFrameMain = new JFrame("New Product Page");

                labelPanel.add(label0);
                labelPanel.add(label1);
                labelPanel.add(label2);
                fieldPanel.add(text0);
                fieldPanel.add(text1);
                fieldPanel.add(text2);

        SUBMIT=new JButton("Submit");

        SUBMIT.addActionListener(new ActionListener(){
                @Override
                public void actionPerformed(ActionEvent e){
                        System.out.println("[info]  : ------ Submit button clicked (AddNewProductUI Admin) ------");
                        AddNewProduct a = new AddNewProduct();
                        String name;
                        double cost;
                        int stock;
                        name = text0.getText();
                        cost = Double.parseDouble(text1.getText());
                        stock = Integer.parseInt( text2.getText());
                        averageFrameMain.setVisible(false);
                        try{
                        if(a.validateProduct(name))
                                switch( a.process(name , cost , stock)){
                                case "CPU": displayAddCPU(a);break;
                                case "Monitor": displayAddMonitor(a);break;
                                case "RAM": displayAddRam(a);break;
                                case "Motherboard": displayAddMotherboard(a);break;
                                };
                        }
                        catch(IOException ex){

                        }

                }
        });

                SUBMIT.setActionCommand("1");

                JPanel controls = new JPanel(new FlowLayout(FlowLayout.CENTER,5,2));
                controls.add(SUBMIT);

                JPanel gui = new JPanel(new BorderLayout(10,10));
gui.setBorder(new TitledBorder("New Product"));
gui.add(inputControls, BorderLayout.CENTER);
gui.add(controls, BorderLayout.SOUTH);

averageFrameMain.setContentPane(gui);
averageFrameMain.pack();
averageFrameMain.setLocationByPlatform(true);
averageFrameMain.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
averageFrameMain.setVisible(true);
    }
```

```
        public void displayAddCPU(final AddNewProduct a){

        label0 = new JLabel();
        label0.setText("Enter CPU Series:");
                label1 = new JLabel();
                label1.setText("Enter CPU Socket Type");
                label2 = new JLabel();
        label2.setText("Enter Operating Frequency");
        text0 = new JTextField(15);
                text1 = new JTextField(15);
                text2 = new JTextField(15);

    JFrame averageFrame = new JFrame("New CPU");

                labelPanel.add(label0);
                labelPanel.add(label1);
                labelPanel.add(label2);
                fieldPanel.add(text0);
                fieldPanel.add(text1);
                fieldPanel.add(text2);

            SUBMIT=new JButton("Submit");

            SUBMIT.addActionListener(new ActionListener(){
                    @Override
                    public void actionPerformed(ActionEvent e){
                            String result;
                            result = "," +  text0.getText();
                            result += "," +  text1.getText();
                            result += "," +  text2.getText();
                            try{
                                    a.addProduct(result);
                                    averageFrame.setVisible(false);
                            }
                            catch(IOException ex){

                            }
                    }
            });

                    SUBMIT.setActionCommand("1");

                    JPanel controls = new JPanel(new FlowLayout(FlowLayout.CENTER,5,2));
                    controls.add(SUBMIT);

                    JPanel gui = new JPanel(new BorderLayout(10,10));
        gui.setBorder(new TitledBorder("New CPU"));
        gui.add(inputControls, BorderLayout.CENTER);
        gui.add(controls, BorderLayout.SOUTH);

        averageFrame.setContentPane(gui);
        averageFrame.pack();
        averageFrame.setLocationByPlatform(true);
        averageFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        averageFrame.setVisible(true);
}
```

CartUI.java

```
package UserInterfaceLayer;

import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.util.ArrayList;

import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JScrollPane;

import BusinessLayer.CustomerClasses.Customer;
import BusinessLayer.OrderClasses.Order;
import BusinessLayer.ProductClasses.Product;

@SuppressWarnings("serial")
public class CartUI extends JPanel{
	int counter = 0;

	public CartUI(Order userOrder,Customer currentCustomer) {
		System.out.println("[debug] : ****** Starting CartUI Class  ******");

		//
**********************************************************************************************************************
		// ************	Start JFrame stuff
		//
**********************************************************************************************************************

		JList list;
		ArrayList<Product> orderProducts = userOrder.getProducts();

		DefaultListModel model;

		JFrame cartUiFrame = new JFrame("Your Details - please review your details");
		cartUiFrame.setLayout(new BorderLayout());
		model = new DefaultListModel();
		list = new JList(model);
		JScrollPane pane = new JScrollPane(list);
		JButton addButton = new JButton("Add Element");
		JButton removeButton = new JButton("Remove Element");
		for (int i = 0; i < orderProducts.size(); i++)
			model.addElement("Product " + orderProducts.get(i).getProductDetails());

		addButton.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
			}
		});

		removeButton.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
```

```
                                        int selectedIndex =list.getSelectedIndex();
                                        if (selectedIndex != -1) {
                                                model.removeElementAt(selectedIndex);
                                                orderProducts.remove(selectedIndex);
                                                userOrder.setProducts(orderProducts);
                                        }
                                }
                        });

                        cartUiFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                        JLabel addShippingCostLabel = new JLabel();
                        addShippingCostLabel.setText("<html><br><b>There is a \u20ac5.00 shipping cost that will be
added to your order.</b><br><br></html>");

                        JButton purchaseButton = new JButton("Proceed to Details");
                        purchaseButton.addActionListener(new ActionListener() {
                                @Override
                                public void actionPerformed(ActionEvent evt) {
                                        System.out.println("[info]  : ------ Proceed to Details button clicked (CartUI)
------");

                                        cartUiFrame.setVisible(false);
                                        cartUiFrame.dispose();
                                        userOrder.setProducts(orderProducts);
                                        EditOrderDetailsUI createNewOrderSummaryUI = new
EditOrderDetailsUI(userOrder, currentCustomer);
                                }
                        });

                        JPanel controls = new JPanel(new FlowLayout(FlowLayout.CENTER,5,2));

                        controls.add(addShippingCostLabel);
                        controls.add(pane, BorderLayout.NORTH);
                        controls.add(addButton, BorderLayout.WEST);
                        controls.add(removeButton, BorderLayout.EAST);
                        controls.add(purchaseButton, BorderLayout.SOUTH);

                        cartUiFrame.pack();
                        cartUiFrame.setContentPane(controls);
                        cartUiFrame.setLocationByPlatform(true);
                        cartUiFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                        cartUiFrame.setSize(800, 600);
                        cartUiFrame.setResizable(false);
                        cartUiFrame.setLocationRelativeTo(null);
                        cartUiFrame.setVisible(true);
        }
}
```

CreatePCUI.java

```
package UserInterfaceLayer;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Font;
import java.awt.event.ActionEvent;
```

```
import java.awt.event.ActionListener;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;

import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;

import BusinessLayer.BusinessLayerDataControl;
import BusinessLayer.CreatePC;
import BusinessLayer.ProductList;
import BusinessLayer.CustomerClasses.Customer;
import BusinessLayer.ProductClasses.Product;
import BusinessLayer.ProductClasses.Desktop;
import BusinessLayer.ProductClasses.Component;

@SuppressWarnings("serial")
public class CreatePCUI extends JPanel{

	private JList listOfChoices;
	private JList listOfChosenProducts;
	private DefaultListModel<String> model;
	private DefaultListModel<String> modelChosen;

	JPanel topPanel = new JPanel();
	JPanel bottomPanel = new JPanel();
	JPanel m = new JPanel();
	JPanel p = new JPanel();
	JFrame CreatePCUiFrame = new JFrame("Create PC model:");
	JLabel currentLabel = new JLabel("Add a component to the list");
	private boolean chooseArrayMade = false;
	private boolean chosenArrayMade = false;
	private CreatePC viewController;

	public CreatePCUI(Customer currentCustomer) throws IOException {
		System.out.println("[debug] : ****** Starting CreatePCUI Class  ******");

		//
**********************************************************************************************************************
		// ************	Start JFrame stuff
		//
**********************************************************************************************************************

		JFrame CreatePCUiFrame = new JFrame("Create PC model:");
		CreatePCUiFrame.setBackground(new Color(0,100,200));
		CreatePCUiFrame.setLayout(new BorderLayout(200,200));
		JLabel productsListJLabel = new JLabel("Products");
		CreatePCUiFrame.add(productsListJLabel, BorderLayout.NORTH);
		productsListJLabel.setFont(new Font("Dialog",Font.BOLD,20));
		model = new DefaultListModel<String>();
		modelChosen = new DefaultListModel<String>();
		viewController = new CreatePC(currentCustomer);
```

```
		JButton addBtn = new JButton("Add");
		JButton backBtn = new JButton("Back");
		JButton buyBtn = new JButton("Buy");
		bottomPanel.setVisible(false);
		bottomPanel.add(buyBtn);
		buyBtn.addActionListener(new ActionListener() {
			@Override
			public void actionPerformed(ActionEvent evt) {
				viewController.buyButtonClicked();
			}
		});



		addBtn.addActionListener(new ActionListener() {
			@Override
			public void actionPerformed(ActionEvent evt) {
				viewController.addButtonClicked(listOfChoices.getSelectedIndex());
				populateArrayOfProducts();
			}
		});
		m.add(addBtn);
		populateArrayOfProducts();
		backBtn.addActionListener(new ActionListener() {
			@Override
			public void actionPerformed(ActionEvent evt) {
				System.out.println("[info]  : ------ Back button clicked (CreatePCUI Customer)
------");
				viewController.backBtnClicked();
				if(viewController.chosenArrayMade){
					if ( viewController.indexThroughComponentsToChoose >= 8) {
						m.add(addBtn);
						m.add(listOfChoices);
						currentLabel.setVisible(true);
					}
				}
				populateArrayOfProducts();
				viewController.backButtonClicked = false;
			}
		});

		p.add(backBtn);




	}
	public void populateArrayOfProducts() {
		if (listOfChoices != null ) {
			m.remove(listOfChoices);
			listOfChoices.removeAll();
			chooseArrayMade = false;

		}
		if(listOfChosenProducts != null) {
			listOfChosenProducts.removeAll();
```

```
		}
		if (modelChosen != null) {
			modelChosen.removeAllElements();
		}
		if (model != null) {
			model.removeAllElements();
		}
		String part ="";
		boolean endOfFileReached = false;
		while (model.isEmpty() && !endOfFileReached)
		{
			switch(viewController.indexThroughComponentsToChoose) {
			case 0 : part = "Motherboard"; break;
			case 1 : part = "CPU"; break;
			case 2 : part = "GPU"; break;
			case 3 : part = "RAM"; break;
			case 4 : part = "MemoryDrives"; break;
			case 5 : part = "Monitor"; break;
			case 6 : part = "Keyboard"; break;
			case 7 : part = "Mouse"; break;
			case 8 : part = "Speaker"; break;
			}
			for(int i = 0; i < viewController.getProductsInFile().size(); i++){
				Product someProduct = viewController.getProductsInFile().get(i);
				if(someProduct.getProductName().equals(part)) {
					model.addElement(someProduct.getProductUIDetails());
				}
				listOfChoices = new JList(model);
			}
			if (model.isEmpty()) {
				if (viewController.backButtonClicked == true)
				{
					viewController.indexThroughComponentsToChoose--;
					bottomPanel.setVisible(false);;

				}
				else {
					if(viewController.indexThroughComponentsToChoose > 8) {
						endOfFileReached = true;
						m.removeAll();
						currentLabel.setVisible(false);
						bottomPanel.setVisible(true);

					}

System.out.println(viewController.indexThroughComponentsToChoose);
					viewController.indexThroughComponentsToChoose++;
				}
			}

		}
		if (listOfChoices != null && !chooseArrayMade)
		{
			m.add(listOfChoices);
			chooseArrayMade = true;
		}
		String labelPart = viewController.getlabel(part);
```

```
                currentLabel.setText("Please pick a " + labelPart + " for your PC" );


                int i = 0;
                ArrayList<Product> chosenComps= viewController.getListOfComponents();
                Iterator<Product> it = chosenComps.iterator();
                while (it.hasNext()){
                        i++;
                        Product someComp = it.next();
                        System.out.println(someComp.getProductName());
                        modelChosen.addElement(someComp.getProductUIDetails());
                        listOfChosenProducts = new JList(modelChosen);

                }

                if (listOfChosenProducts != null && !chosenArrayMade){
                        p.add(listOfChosenProducts);
                        chosenArrayMade = true;
                }
                topPanel.add(currentLabel);
                CreatePCUiFrame.add(bottomPanel, BorderLayout.SOUTH);
                CreatePCUiFrame.add(topPanel, BorderLayout.NORTH);
                CreatePCUiFrame.add(m, BorderLayout.WEST);
                CreatePCUiFrame.add(p, BorderLayout.EAST);
                CreatePCUiFrame.setSize(800, 500);
                CreatePCUiFrame.setVisible(true);
        }
}
```