# Table of content

[Table of content 1](#_Toc26195590)

[Introduction 3](#_Toc26195591)

[Class Diagram 4](#_Toc26195592)

[Use Case Diagram 6](#_Toc26195593)

[Use Cases Specification 7](#_Toc26195594)

[Login 7](#_Toc26195595)

[Validation 7](#_Toc26195596)

[Manage Manager 8](#_Toc26195597)

[Edit Manager 8](#_Toc26195598)

[Manage Festival 9](#_Toc26195599)

[Manage Product 9](#_Toc26195600)

[Manage Supplier 10](#_Toc26195601)

[Manage Catalogue 11](#_Toc26195602)

[Object-Oriented Concept 12](#_Toc26195603)

[Inheritance 12](#_Toc26195604)

[Polymorphism 13](#_Toc26195605)

[Abstraction 14](#_Toc26195606)

[Encapsulation 14](#_Toc26195607)

[Association 15](#_Toc26195608)

[composition 15](#_Toc26195609)

[Usage of some Keywords 17](#_Toc26195610)

[clone 17](#_Toc26195611)

[Try & catch (exception) 17](#_Toc26195612)

[Interface 18](#_Toc26195613)

[ArrayList<> 19](#_Toc26195614)

[Conclusion 21](#_Toc26195615)

[Reference 22](#_Toc26195616)

WORK MARTIX BREAK DOWN

|  |  |  |  |
| --- | --- | --- | --- |
| Name | TP Number | Break Down | Signature |
| Zhang Zi Teng | TP052096 | 50% |  |
| Leong Soong Jun | TP050994 | 50% |  |

# Introduction

In Malaysia, we observe different special promotions or discount offers in different festivals and we often receive product catalogues either online or offline which are distributed from different channels. Upon those festivals, product management and catalogue design for business enterprises become the most concern throughout a year to promote their products to their customers. In this assignment, we will construct a software solution using object-oriented programming paradigm to support product information management and catalogue maker for different festivals.

# Class Diagram

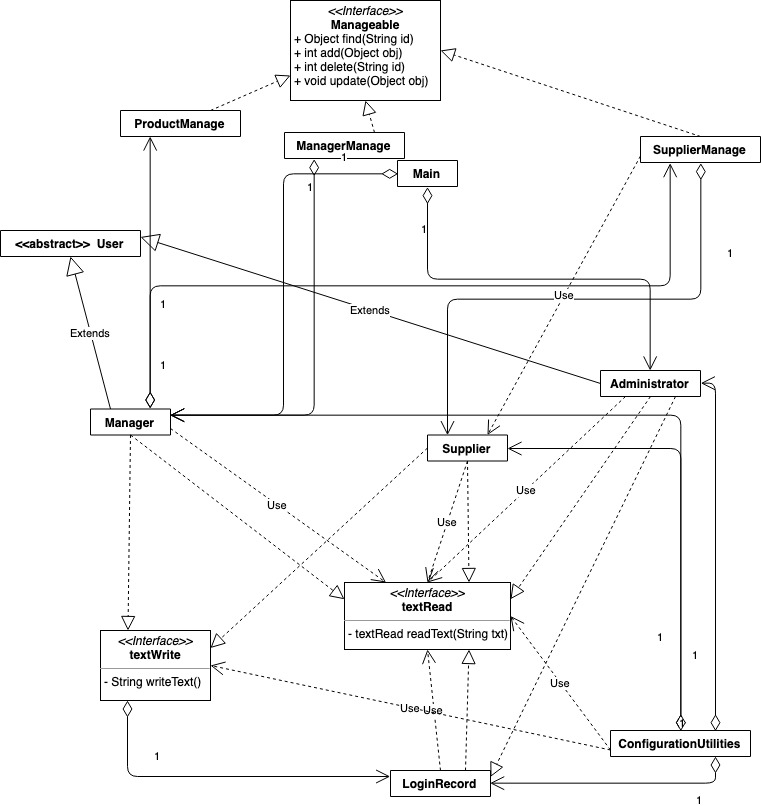


Figure 1 Class Diagram 1

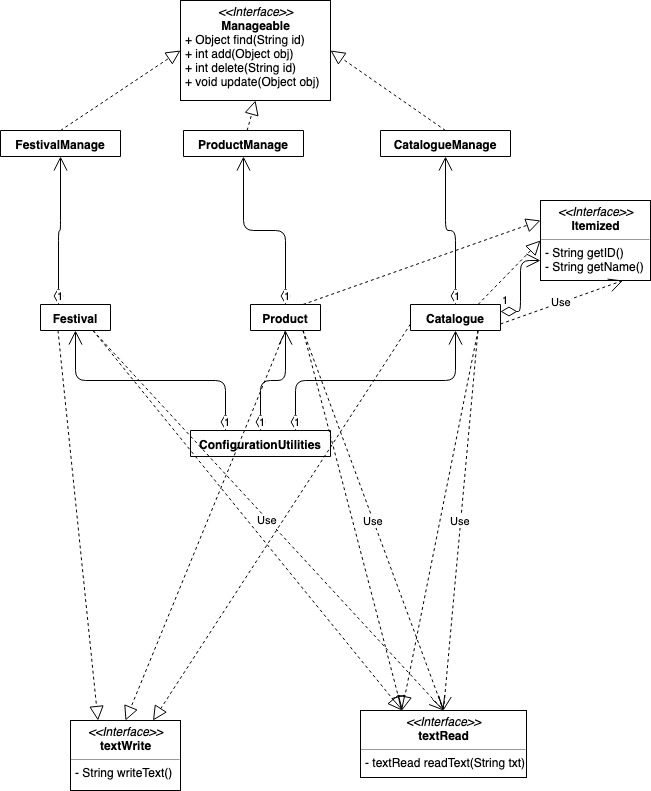


Figure 2 Class diagram 2

# Use Case Diagram

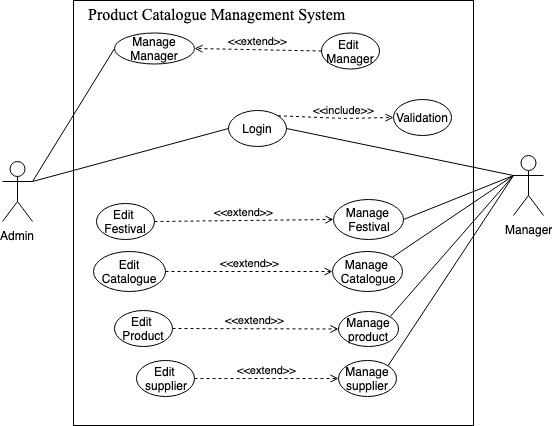


Figure 3 Use case Diagram

## Use Cases Specification

|  |  |
| --- | --- |
| Use Case | Login |
| Brief Description | This use case asking the ID and Password from user. |
| Actors | Manager, Admin |
| Preconditions | Manager or Admin provides id and password that is to login. |
| Main Flow | (a) The use case begins when a user activates the System.  (b) The system requests that the user enters ID and Password to login as admin or manager.  (c) The system searches id and password from the admin.txt file and manager.txt file to validate if they are correct. (ref. Alternative flow). |
| Alternative Flows | The actor enters id and password and select which kind of account it wants to login. Administrator or Manager. |

|  |  |
| --- | --- |
| Use Case | Validation |
| Brief Description | This use case Validation the ID and Password from user. |
| Actors | Admin, manager |
| Preconditions | Validation id and password from user. |
| Main Flow | The use case begins when a user activates enter id and password.  The system read id and password from the admin.txt file and manager.txt file to validate. (ref. Alternative flow). |
| Alternative Flows | Password and id can match to the txt data then give access to admin or manager. If can’t match, give warn. |

|  |  |
| --- | --- |
| Use Case | Manage Manager |
| Brief Description | This use case allows admin to find, add, update, and delete the Name and password of all the managers. |
| Actors | Admin |
| Preconditions | Admin set Name, password and department of each manager. |
| Main Flow | (a) The use case begins when an admin password can match the id.  (b) The system requests that the user enters Name, Password and department to add or update a new manager.  (c) The system reads and write the id, Name, department and password for the manager.txt file.(ref. Alternative flow). |
| Alternative Flows | * The admin can add, update, delete manager. * There is a warn message to tell admin when delete a manager. Such as yes, no, or cancel. * The admin can choose to go back to login page or exit the system. |

|  |  |
| --- | --- |
|  | Edit Manager |
| Brief Description | This use case allows admin to find add, delete, update. |
| Actors | Manager |
| Preconditions | Display the data from manager.txt file. |
| Main Flow | The use case begins when a manager click Login to administrator or click button add, update and delete in the manage manager interface. |
| Alternative Flows | Edit the data and write to text file. |

|  |  |
| --- | --- |
| Use Case | Manage Festival |
| Brief Description | This use case only allows manager to add, update and delete the festivals. And print the festival data from text file to PDF. |
| Actors | Manager |
| Preconditions | Manager click the festival management button. |
| Main Flow | 1. Display the data read from festival.txt file. 2. Manager can select festivals to check festival information 3. Manager can add, update, delete, print, go back to login page and exit the program. |
| Alternative Flows | * Manager can click add to add festivals. * Manager can click delete to delete festivals. * Manager can click update to edit festivals information. * Manager can click print to print festival information from txt file to a pdf file. * Manager can click go back to login to change account. * Manager can click exit to close the system. |

|  |  |
| --- | --- |
| Use Case | Manage Product |
| Brief Description | This use case only allows manager to add, update, search and delete the products. And print the product data from text file to PDF. |
| Actors | Manager |
| Preconditions | Manager click the product management button. |
| Main Flow | 1. Display the data read from product.txt file. 2. Manager can select product to check product information 3. Manager can add product and add supplier to product. 4. Manager can update, delete, print, go back to login page and exit the program. |
| Alternative Flows | * Manager can click add product to add product. * Manager can set product information (price, discount, quantity, weight) * Manager can select supplier to add to product * Manager can click delete to delete product. * Manager can click update to edit product information. * Manager can click print to print product information from txt file to a pdf file. * Manager can click go back to login to change account. * Manager can click exit to close the system. |

|  |  |
| --- | --- |
| Use Case | Manage Supplier |
| Brief Description | This use case only allows manager to add, update and delete the Supplier. And print the supplier data from text file to PDF. |
| Actors | Manager |
| Preconditions | Manager click the supplier management button. |
| Main Flow | 1. Display the data read from supplier.txt file. 2. Manager can select supplier to check festival information 3. Manager can add, update, delete, print, go back to login page and exit the program. |
| Alternative Flows | * Manager can click add to add supplier. * Manager can click delete to delete supplier. * Manager can click update to edit supplier information. * Manager can click print to print supplier information from txt file to a pdf file. * Manager can click go back to login to change account. * Manager can click exit to close the system. |

|  |  |
| --- | --- |
| Use Case | Manage Catalogue |
| Brief Description | This use case only allows manager to add, update and delete catalogue; add, delete product for each catalogue; add, delete festivals for each catalogue. And can print catalogue information to pdf file. |
| Actors | Manager |
| Preconditions | Manager click the catalogue management button. |
| Main Flow | 1. Display the data read from catalogue.txt file, product.txt file and festival.txt file. 2. Manager can select catalogue to check its product and festival information 3. Manager can add, update, delete catalogue, add delete product; add delete festival; print, go back to login page and exit the program. |
| Alternative Flows | Manager can click add catalogue to add catalogue.  Manager can click delete catalogue to delete catalogue.  Manager can click update catalogue to edit catalogue name.  Manager can click add or delete product to catalogue to add or delete one or more products to each catalogue.  Manager can click add or delete festival to catalogue to add or delete one or more festivals to each catalogue.  Manager can click print to print catalogue information from txt file to a pdf file.  Manager can click go back to login to change account.  Manager can click exit to close the system. |

# Object-Oriented Concept

An object is an entity with a well-defined boundary and identity that encapsulates state and behaviour. The state is represented by attributes and relationships, nonetheless, the behaviour is represented by operations, methods, and state machines. A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics.

## Inheritance

Inheritance is a subclass inherits its parent’s attributes, operations, and relationships. A subclass may add additional attributes, operations, relationships or redefine inherited operations. Upon the use of inheritance in an application, we would like to list out one of the codes as an example to prove its importance. We set “Administrator” class to “extends” “User” class. Hence, the “Administrator” class has inherited string inputs completely from the “User” class, for example, public declaration of ID, Name, Password etc.

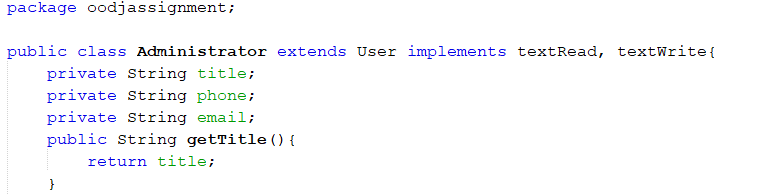


Figure 4: Public class **Administrator** *extends* **User**

We see some key benefits of inheritance. For instances, inheritance can minimize the number of duplicate codes in an application, it helps to organize the code. Inheritance is common with its reusability, it helps the developer to reduce unnecessary work. Besides, the developer can override the methods from the superclass then redesigned a new derived class.

## Polymorphism

Polymorphism is the ability of an object to take on many forms. It occurs where many classes are related to each other by inheritance. Unlike inheritance, polymorphism can perform different tasks. We see two types of polymorphism, thus, run-time polymorphism is implemented which fully deploy the methods through method overriding, for instances, we declare a method then execute the codes. In “public Object Clone ()”, we declare “a” as a variable and allowed “a” to accept ID, Password etc with the method of “a.setid(this.getid())”, “a.setPassword(this.getPassword())” etc.

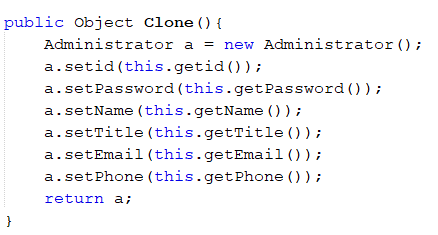


Figure 5: Public Object **Clone** () {}

However, we find out the benefit of polymorphism. Method override has allowed performing related functions by accessed through a common name. It also encouraged different ways to initialize objects of a class by implemented on constructors.

## Abstraction

Abstraction is the essential characteristics of an entity that distinguishes it from all other kinds of entities. It used to hide certain details and only show the essential features of the objects. We insert this concept in “User” due to the existing of personal information of the end-user. We also set the “protected” to the declaration in “User”. However, the other class can inherit its properties by using “extends” with method override.

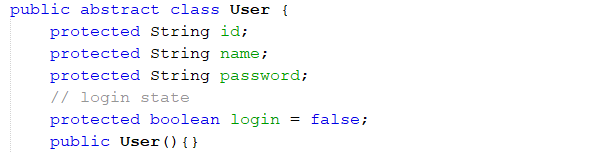


Figure 6: Public *abstract* class **User**

We see the benefits of implement abstraction, it allows the developers to group several related classes as siblings. Abstraction helps to reduce the complexity of the design and implementation process of software.

## Encapsulation

Encapsulation is a mechanism of wrapping the variables and code acting on the methods together as a single unit. It is the mechanism that binds together code and the data it manipulates. Besides, it prevents the data from being accessed by the code outside this shield. Upon this mechanism, we apply to our “Catalogue”. We set “private” to the declaration in “Catalogue” because we want to control the variables is not accessed by other classes. For example,

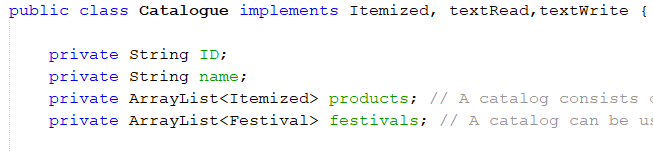


Figure 7: Public class **Catalogue**

Furthermore, we see some key benefit of encapsulation. Encapsulation is famous in hiding data; the end-user cannot understand the inner implementation of the class. We can easily increase its flexibility, by setting the variables into read-only and write-only formal. Encapsulation also improves the re-usability and easy to change with new requirements.

## Association

Association is a relation between two separate classes which establishes through their objects. Association can be one-to-one, one-to-many, many-to-one, many-to-many. Composition and aggregation are the two forms of association, nevertheless, we would like to implement and mention the use of the composition.

## composition

The composition represents the Has-A relationship. It is best achieved for code reuse. The composition is a restricted form of aggregation in which two entities are highly dependent on each other. We can see Figure 5 which apply the previous LoginRecord into the latest LoginRecord. We can say if the first LoginRecord will be destroyed, the second LoginRecord will be affected which leads to destroyed. Therefore, the second LoginRecord is highly dependent on the first LoginRecord.

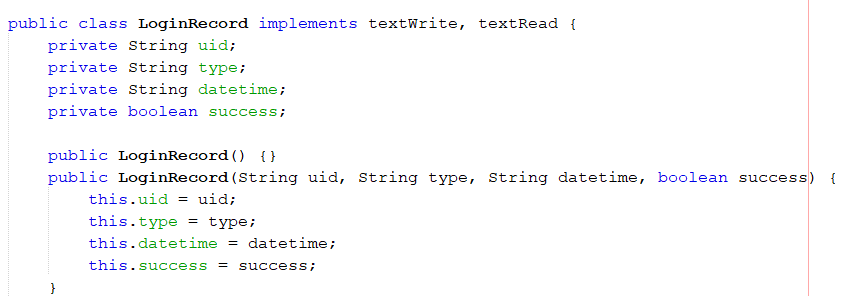


Figure 8: Public class **LoginRecord** & public **LoginRecord**

We see the benefit of composition. Developers can control the visibility of other objects to client classes and reuse only what developers need. Composition allows a more flexible and extensible mechanism to reuse code.

# Usage of some Keywords

## clone

Nevertheless, we also implement the clone method. Object cloning refers to the creation of exact copy of an object. It creates a new instance of the class of current object and initializes all its fields with exactly the contents of the corresponding fields of this object. Hence, it has two types of clone method which are deep copy and shallow copy, in our case, we apply clone method with deep copy. We have cloned the “addFestival” and “addProduct” functions into another class.

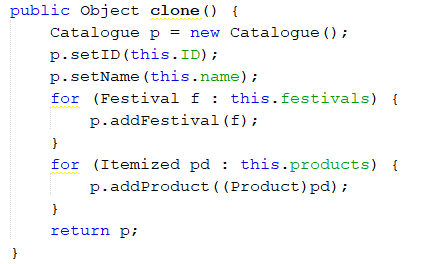


Figure 9: Public Object **clone**

The benefits of the clone method are to assign an object reference to another reference variable then it will point to the same address location of the old object and no new copy of the object will be created. Developers use copy constructor, then copy the data completely over explicitly. It can avoid extra processing if the developers use object cloning.

## Try & catch (exception)

An exception is an unwanted or unexpected event, which occurs during the execution of a program, that disrupts the normal flow of the program’s instructions. The exception handler can handle the occurred exception when the run-time system searches the call stack to find the method. We concern the inability of the application to occur incidentally, therefore, we execute an exception in our source code. We used it to “readText” and “writeText” to avoid unnecessary crash between the interpretation of text file and our application.

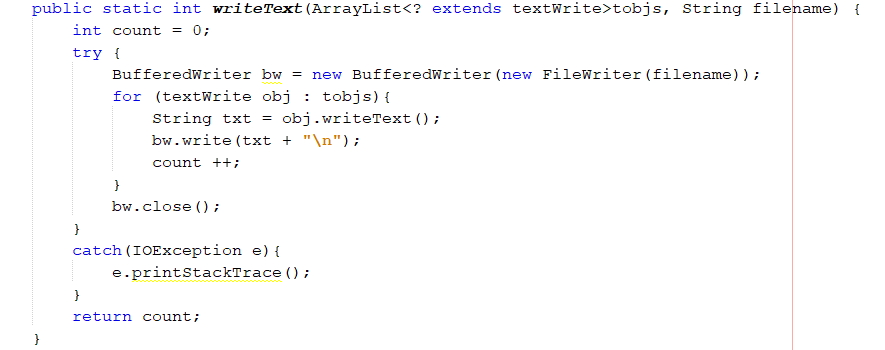


Figure 10: Exception Handler

We see huge benefits on the exception handler. First, it can separate error-handling code from the regular code. Second, it helps to propagate error up the call stack. Third, it will classify the error types and error differentiation.

## Interface

The interface is a mechanism to achieve abstraction. It can be only abstract methods in the Java interface, not the method body. It is used to achieve abstraction and multiple inheritances in Java. Hence, it provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. In our case, we used some interfaces to cooperate with other classes. For instances, public class “Administrator” “implements” “textRead” and “textWrite” to achieve multiple inheritances in the class. We can inherit the methods from “textRead” and “textWrite” and achieve more functionality in this class.

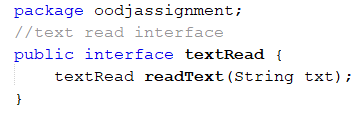
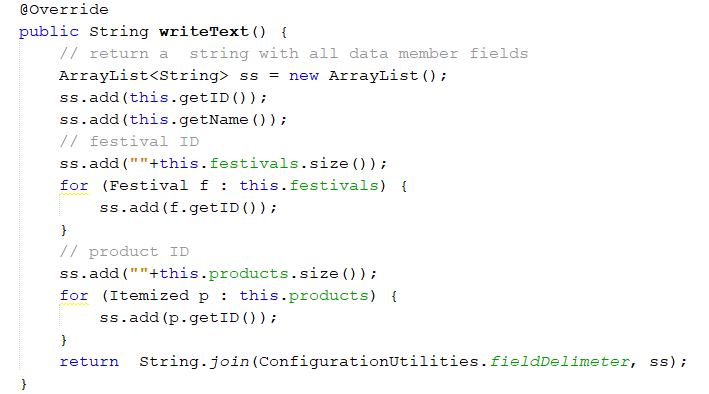


Figure 11: Public interface **textRead**

We discover some benefits of the interface. Interfaces allow standard sets of methods to be used across the class hierarchy. For example, developers define textRead interface to support read-string operations. Besides, interface types allow objects to be referenced by the methods they support without considering their location in the class hierarchy. Interfaces are declared independently of classes, they are unaffected by changes to specific classes or to the class hierarchy.

## ArrayList<>

ArrayList is a part of collection framework and is present in Java package. It provides developers with dynamic arrays in Java. On the other hand, it may be slower than the standard arrays but can be helpful in programs where lots of manipulation in the array is needed. We use ArrayList to perform add, delete, update etc functions.



Nonetheless, we see some key benefits of ArrayList in Java. ArrayList allows duplicate and null values. An ArrayList is a re-sizable array, also called a dynamic array. It grows its size to accommodate new elements and shrinks the size when the elements are removed. ArrayList internally uses an array to store the elements, meanwhile, retrieve the elements by their index.

# Conclusion

Upon the request of the assignment, we learned how to develop an object-oriented system. We have learnt how to design and develop a software solution using the object-oriented paradigm and translate it into software application that exploits the strength of object-oriented paradigm. Besides, we learned demonstrate object-oriented concepts and their functionalities in the existing system.

# Reference

GeeksforGeeks. (2019). *Clone() method in Java - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/clone-method-in-java-2/ [Accessed 2 Dec. 2019].

Török, P., Patel, J. and Khan, N. (2019). *Why do we use the clone() method in Java?*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/5802118/why-do-we-use-the-clone-method-in-java [Accessed 2 Dec. 2019].

GeeksforGeeks. (2019). *ArrayList in Java - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/arraylist-in-java/ [Accessed 2 Dec. 2019].

Stack Abuse. (2019). *Java's Object Methods: equals(Object)*. [online] Available at: https://stackabuse.com/javas-object-methods-equals-object/ [Accessed 2 Dec. 2019].

GeeksforGeeks. (2019). *Interfaces in Java - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/interfaces-in-java/ [Accessed 2 Dec. 2019].

GeeksforGeeks. (2019). *final keyword in java - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/final-keyword-java/ [Accessed 2 Dec. 2019].

GeeksforGeeks. (2019). *Access Modifiers in Java - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/access-modifiers-java/ [Accessed 2 Dec. 2019].

GeeksforGeeks. (2019). *Method Class | getName() Method in Java - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/method-class-getname-method-in-java/ [Accessed 2 Dec. 2019].

GeeksforGeeks. (2019). *Try, catch, throw and throws in Java - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/try-catch-throw-and-throws-in-java/ [Accessed 2 Dec. 2019].

Guru99.com. (2019). *Java Swing Tutorial: Examples to create GUI*. [online] Available at: https://www.guru99.com/java-swing-gui.html [Accessed 2 Dec. 2019].