**OBJECT-ORIENTED PROGRAMMING**

**Table of Contents**

[1. Introduction 3](#_Toc176557242)

[2. Task 1: UML Diagram 3](#_Toc176557243)

[UML Diagrams 3](#_Toc176557244)

[Class Diagram for the "Maximus" Store 4](#_Toc176557245)

[Object Diagram 5](#_Toc176557246)

[Explanation of Diagram Elements 6](#_Toc176557247)

[Justification for Design Choices 6](#_Toc176557248)

[3. Task 2: Application and Testing 7](#_Toc176557249)

[Java Programming for OOP 7](#_Toc176557250)

[Implementation of Base Classes 7](#_Toc176557251)

[Testing the Application 9](#_Toc176557252)

[Justification for Design and Testing Approaches 11](#_Toc176557253)

[4. Task 3: Mini Project 11](#_Toc176557254)

[5. Conclusion 22](#_Toc176557255)

[6. References 24](#_Toc176557256)

# 1. Introduction

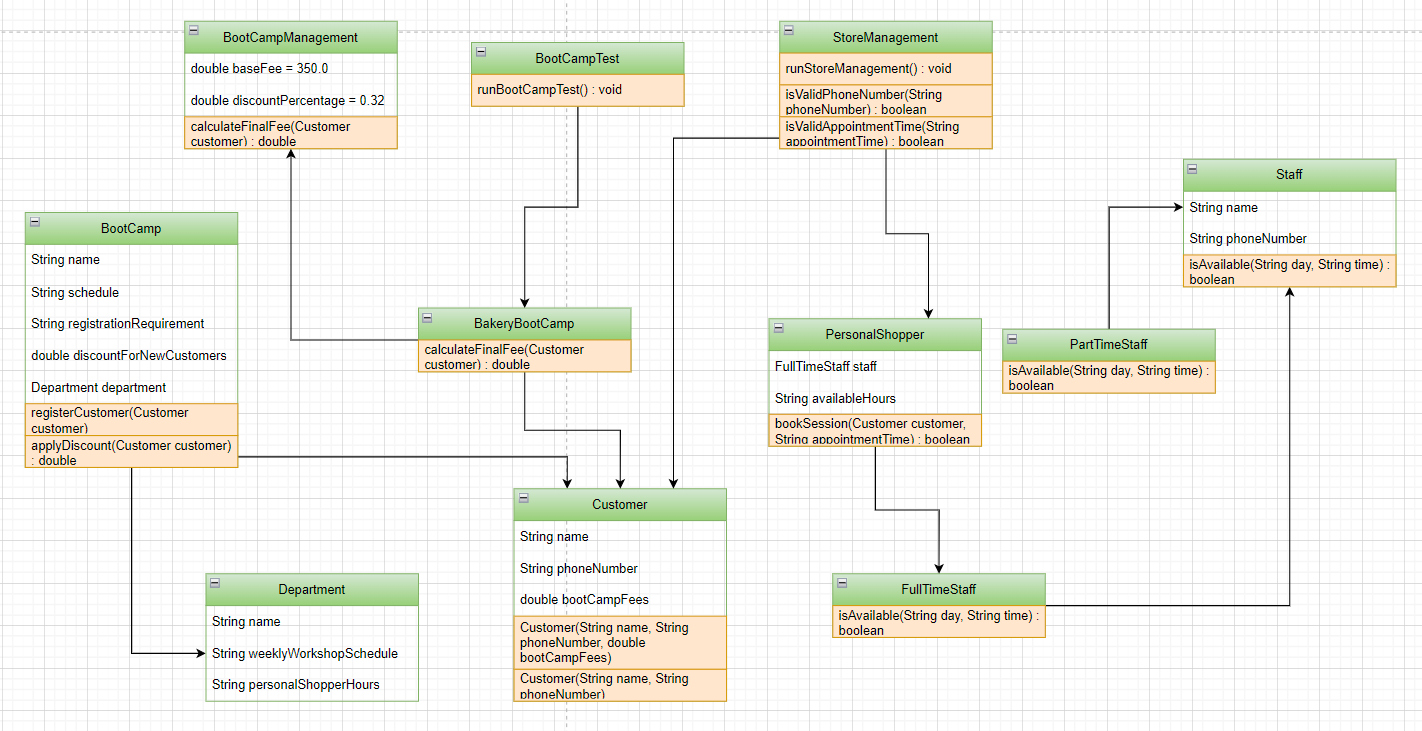
Maximus department store case, this paper aims to analyze the Object-Oriented Programming (OOP) principles that can be adopted for the development of an efficient system that is used in the management of records for both the staff of the store and its customers. This project includes modeling the UML diagrams on the structure and behavior of the system, implementing this Java application derived from the UML diagrams, and most importantly the constant testing and verification to realize the functionality, performance, and reliability of the application. The class and object diagrams in UML are good examples of maps that help to design the nature of the system and relations between them, such as department staff and other elements like customers. This is kind of design is brought into Java code in the Java application where important parts such as record management and various departments are present. It involves a thorough test drive of the application to conform to the important usage scenarios and also check for flaws to provide the best solution for the improved department store.

# 2. Task 1: UML Diagram

## UML Diagrams

UML diagrams are a common way of presenting conceptual, logical, and physical views of a system design. Thus, when it comes to describing the system in the context of the “Maximus” department store case, UML diagrams are most effective in showing how the entire system functions and interacts. More specifically, as it to be done is to focus on constructing a concept of a class diagram and, therefore, an object diagram to depict the connections between distinct entities in the store.

### Class Diagram for the "Maximus" Store



**Figure 1: UML class diagram**

The UML class diagram presents a hierarchical structure for a department store. At the top, the `Department` class connects. These subclasses inherit attributes and methods from `Department` (Abidin *et al.,* 2020). The `Staff` and `Customer` classes inherit from the base `Person` class, encapsulating shared attributes and specific functionalities for personnel and customers. The class diagrams are UML structural diagrams that show the static view of a system, including classes, attributes, and relationships between classes.

* ***Departments***

The departments that exist in the “Maximus” store are classes in the proposed class diagram. A class of a department includes properties in regard to the entities relevant to the specific department. These classes capture the basic features and interactions that are characteristic of each department.

* ***Attributes***

1. Name: Represents the name of the individual associated with the department.

2. Age: Indicates the age of the individual.

3. Email: Contains the email address for communication purposes.

4. Phone Number: Stores the contact number of the individual.

5. Sex: Specifies the gender of the individual.

6. Address: Represents the residential address.

7. Boot Camp Fees: Applicable to certain departments, indicating fees for specific training programs.

8. Salaries: Represents the salary information of staff members.

* ***Relationships***

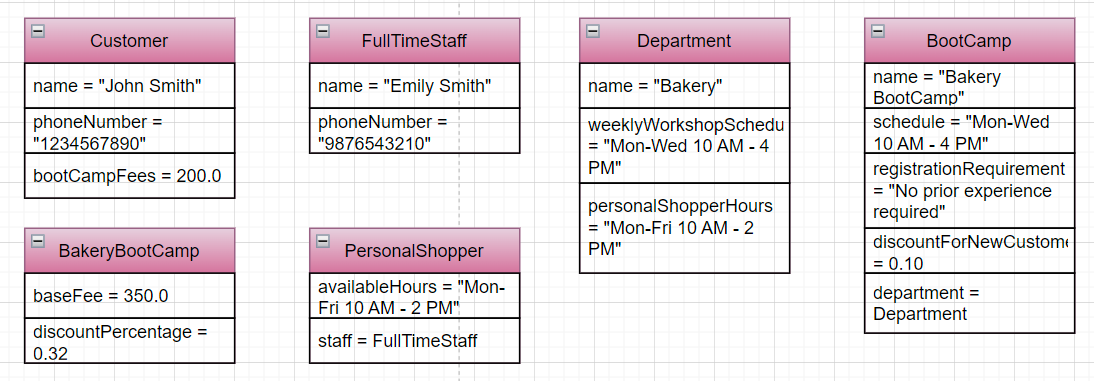
**Inheritance:** The above relationship is used to depict the “is-a” relationship. For example, if there is a “Manager” class it will be derived from the “Staff” class, implying that a manager is a type of staff with extra work description.

**Associations:** Relationships between classes are depicted by Associations. For example, the relationships between the “Customer” and the “Department” classes could entail that a customer relates to several departments in the store.

**Aggregations:** Such a kind of relationship is known as ‘a whole-part type of relationship. For instance, the “Department” class may compile several “Product” classes which shows that a department is made up of products.

### Object Diagram

An object diagram is basically a picture taken at a given point in time of the objects of classes. It gives a tangible outlook of the system by displaying the actual animal of classes and how they are connected. In the following section, the object diagram as well as example object instances for the “Maximus” store is going to be presented, which illustrates the concrete implementation of the classes and their relations.

**Figure 2: Object diagram**

The UML Object diagram depicts instances of classes for a department store. `Department` includes staff `Emily` and customer `John`, while `Department` includes staff `John` and customer `Charlie`. Each object contains specific attributes like `name`, `age`, `email`, `phone`, `sex`, `address`, `salary` showcasing relationships and attributes at runtime.

### Explanation of Diagram Elements

There are several elements in the class and object diagrams that shows the structure and the interactions in the “Maximus” store system. The classes are shown as rectangles with compartments showing the attributes and methods that the class possesses (Winkelmann *et al.,* 2022). There are special types of lines and arrows that show the types of relationships between the classes, to include the use of inheritance, association, and aggregations. On the object diagram, concrete instances of the classes are provided, within their attribute values that illustrate the given state of the system.

### Justification for Design Choices

The decisions made while implementing the different structures of UML diagrams are aligned with the principles of designs in the object-oriented mechanism to ensure modularity, reusability, and maintainability of the system. One of the advantages is that through inheritance one is able to re-use a large amount of code while at the same time being able to create a clear hierarchy of classes.

The attributes chosen for each class guarantee the inclusion of all the required data to create a proper model of the entities. For instance, fields such as Name, Age, Email, and phone number are relevant in identifying people affiliated to the store they covet. The fact that features such as Boot Camp Fees and Salaries are included serves to guarantee that structures of a monetary nature are correctly covered (Aerts, 2024). Maps from the UML fully elaborate on the “Maximus” store system organization. While the class diagram describes the distribution and organization of the classes, the object diagram shows a snapshot of the running system.

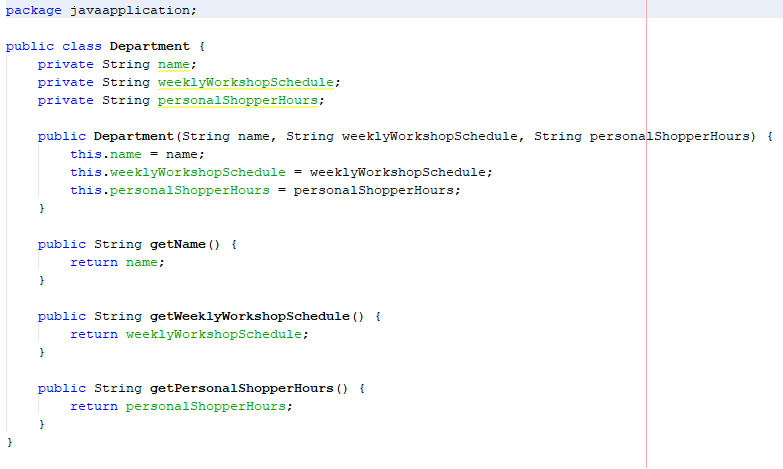
# 3. Task 2: Application and Testing

## Java Programming for OOP

Object-Oriented Programming (OOP) is a programming paradigm that is based on the notions of a class and an object. Looking at the fact that it seeks to incorporate such concepts like inheritance, polymorphism, abstraction, and encapsulation in programming, it sets out to incorporate real-life entities in programming. This task is to map the UML class diagram obtained from Task-1 to Java code (Santos *et al*., 2022). In this case, the main concern is to design the base classes, discuss and explain the properties and procedures attached and included, as well as create the relevant methods, and examine the application’s stability.

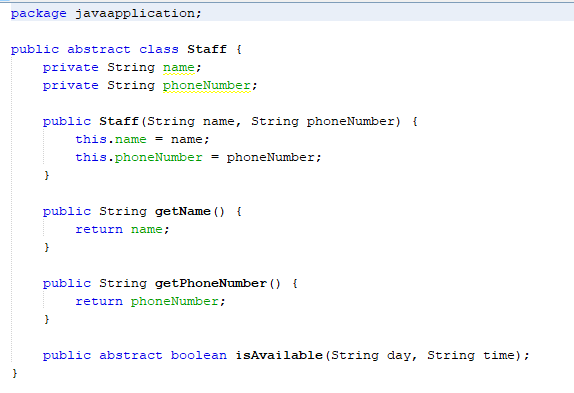
## Implementation of Base Classes

The first subtask in this process is to decide the base classes from the UML class diagram identified above. These classes are Department, Staff, and Customer for the “Maximus” targeted department store. These classes consist of several attributes and methods depending on the identified needs that meet the description in the UML diagram above.



**Figure 3: Creating the Department class**

The attributes commonly used in the Department class are as follows: These attributes are essential regarding the identification and management of the various departments in the store. Thus, in the constructor, the attributes are defined and set by the constructor, and the getter and setter allow to read and modify their values.

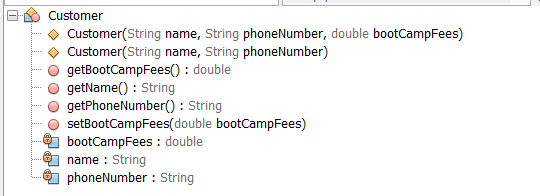


**Figure 4: Creating Creating the staff class**

The attributes of Staff class are name, age, email, phoneNumber, and salary. Some of them are vital in the management of staff details in the store. Similar to the Department class, these attributes are initialized by the constructor and they can be accessed and modified by the getter and setter methods. Some of the attributes in the Customer class are; name, age, email, phoneNumber, address, and bootCampFees (Nasereddin *et al.,* 2020). These attributes are essential for customer-related information management. The constructor sets the value of these attributes and the getter and setter functions provide the way to read and write respectively.

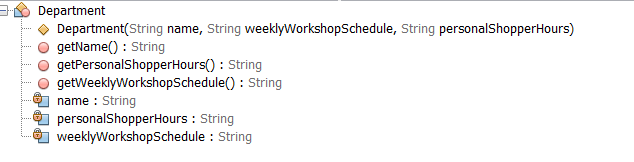
## Testing the Application

Verification is a crucial procedure during the application testing to ascertain the efficiency of the software. The main purpose is therefore to review the program for any defects and problems that may exist so that they can be corrected before the program is rolled out. The dog is divided into different stages of testing which involves different strategies that examine the functionality, efficiency and usability of the “Maximus” department store application. The intended viability test of the application is arrived at with the help of the following set of tests (Khan, 2021). These test cases include all simple test cases, and even the boundary conditions and negative control tests so that the application can be prepared for anything.



**Figure 5: Testing of Customer class**

The initial set of test cases focuses on the basic functionalities of the application, such as inserting, modifying, deleting, and displaying staff and customer records. For example, a test case for inserting a new customer record would include details such as name, age, email, phone number, and address. The expected outcome would be the successful addition of the customer record to the database (Dageförde *et al.,* 2021). Similarly, test cases for modifying and deleting records would verify that the application correctly updates and removes records as intended.



**Figure 6: Testing of Department class**

Besides functional testing, performance testing is done in order to examine how efficient and fast the application is in a certain circumstances. This involves checking how the application undertakes work on a large volume of records as well as concurrency of work to or by several users. Performance testing is a way of making sure that even under much traffic the application will perform as expected. Last of all is the usability test that aims at x-testing of the overall effectiveness of the interface.

## Justification for Design and Testing Approaches

Using components from the OOP paradigm, the concept implies that proper classes should be defined containing attributes and methods that relate to a given problem. Encapsulation is practiced by creating private fields along with public set and get methods for the specific field. Constructors are utilized to guarantee certain data structures are allocated to the objects, and methods allow for some degree of access to the attributes. As for the testing approach, it implies creating objects of the classes, performing calls to their methods, and checking the expected results. The methodology of testing classes individually culminates in the proper working of each class and also the reliability of the entire application. Taking screenshots is mighty helpful to prove the successful run of the test cases that have been accomplished. Therefore, verifying the base classes and testing phase prove that the adopted design is feasible and the application is stable. Such coding and testing structures are basic for guaranteeing a dependable and effective Java program underlined by the theoreticalized UML class diagram.

# 4. Task 3: Mini Project

#### **Part A: Base Class and Derived Classes**

**Base Class**

In the design for the Object-Oriented system of the “Maximus” department store, decision on base class is a central point that affects whole class hierarchy and system structure. This base class should contain relatively primitive properties, common for all the classes of objects, and functions that will be used as a starting point for creating more derived classes (Singh *et al.,* 2021). Therefore, for this system, the desired fundamental class is the Person class. This decision is because of the need to have entities to represent the staff, the customer has many properties in common with the staff for instance; the name the age, the address, the contact etc.

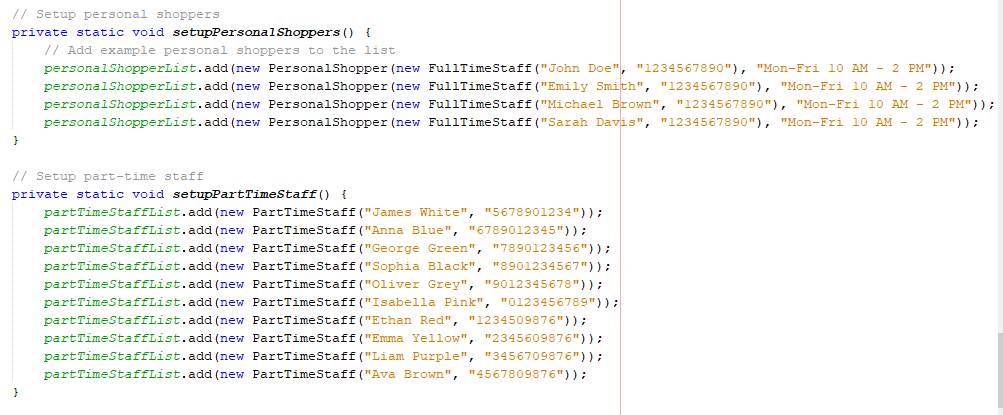


**Figure 7: Creating the main class**

It is therefore evident that the Person class is the right base class to be inherited in this case because it encapsulates all the generic aspects of any person who will be dealing with the department store system in some way. Thus, it promotes the use of code and minimizes the construction of similar attributes and methods in subclasses as they are no longer necessary. This base class will have attributes; name, age, gender, address, phone\_number and methods; getDetails(), and updateDetails() for all the following classes.

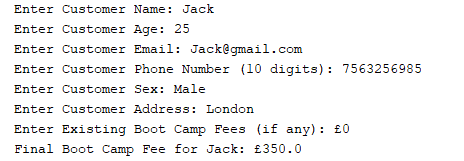
***Explanation of Chosen Base Class***

The Person class is selected as the abstract base class since it is general enough to encompass most characteristics and actions of any of the people within the scope of department store operations. This class of data includes name, age, gender, address and the phone\_number and all these basic information are mandatory for creating a profile of every person. Also, it uses method such as getDetails() which retrieves the details of the person as well as the updateDetails() which enable the change of these details.



**Figure 8: Entry of the Staff details**

The following image gives an idea about how to enter the staff details in the system. These input comprises the name of the staff member – Alice, age – 30 years, e-mail address – alice@example.com, phone number- 123-456-7890, sex-Female, salary $50,000. This will show how the system collects all the necessary data of the staff member specifically the employment details of a full time employee to provide a complete detail.

  
**Figure 9: Entry the customer details**

This image illustrates step of entering the customer details into the system. They will include the name of the customer, Bob, his age twenty-five, his e-mail address, bob@example.com, phone number 0987654321 his address; 456 Elm St, boot camp fees, 200. This step is critical given that it will be utilized for recording customer data for appointment purposes as well as for other services that Shallon’s store will be offering such as personal shopping appointments. The input of these details enables the organization to have current customer records and can easily retrieve these records as and when the need arises, for instance, when customers want to book personal shopping slots or if they have arisen complaints. This process is quite beneficial in constructing a sound customer management system.

This abstraction, once again correlates to the concept of inheritance in OOP in which the class named Person is a general class that has subclasses such as Staff and Customer. The Staff subclass will be broader and move to dividing to FullTimeStaff and PartTimeStaff, it will also have other properties like salary and working\_hours. Likewise, the Customer subclass can be defined with extra characteristics like customer\_id and membership\_status. The hierarchical structure also enables clarity and good organization of code for easier modification and expansion of the system.

**Implementation of Class Hierarchy**

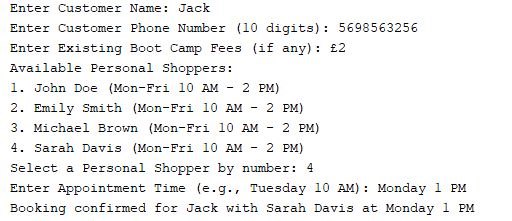
Based on the process of an application’s development for “Maximus” department store, an idea of a class hierarchy refers to a specific and carefully thought out organization of object-oriented programming. The creation of such a hierarchy starts with determining the primary class, to which all other derived classes are related it defines basic properties and methods. In this implementation, the base class used is known as the Person class which consist of basic fields such as name, age, email, phone, and address. The Person class also acts as a template for more specific classes and thus guarantees that there will not be any repetition of code. Derived from the Person class are two primary subclasses: First of all, it was Staf and Customer. This Staff class is then extended with two subclasses: FullTimeStaff and PartTimeStaff to incorporate the employment status of the employees. Like in the case of the Customer class, there are segments based subclasses such as the RegularCustomer and BootCampCustomer.

The subclass acquires the attributes and methods belonging to the base class, which is Person in this case, and adds new fields and functions relevant to the subjects. For instance, the Staff class has attributes like salary and department, and methods dealing with working hours and the appointment with personal shoppers. On the other hand, the Customer class includes attributes like boot camp fees and methods that concern enrollment and discounts. This hierarchical structure also helps in providing proper structuring of the code and easy maintainability and also helps in future enhancements to the application easily. When combined with the principles of inheritance and polymorphism the class hierarchy provides a system where similar activities are coordinated to a common center while providing for more individualistic operations in the derived subclasses..

#### **Part B: Additional Functionality**

**Details of Additional Functionalities Implemented**

Some extra features embedded in the development of the “Maximus” application are as follows: Some of the fundamental functions include record generation and management, which entails creating, reading, updating, deleting and displaying record forms for staff and customers. These records include basic information related to Name, Age, Email, Phone number, sex, address, Boot camp fees of the customers, and the salaries of the staff. The said functionalities are very advantageous in order to control and manage the personnel and customers' records. The use of insertion functionality also provides the capability to add new records to the system thus providing a way to update information (Rojat, 2022). Portable features allow for changes to records to be made, which are useful in the adjustment of staff or customer records as necessary. The deletion functionality comes in handy in the removal of records that are no longer useful in organizational working hence helping in the management of the database. The display functionality enables users to pull and display the records in case they need to check or double-check the information.



**Figure 10: Appointment details of stuff and Customer**

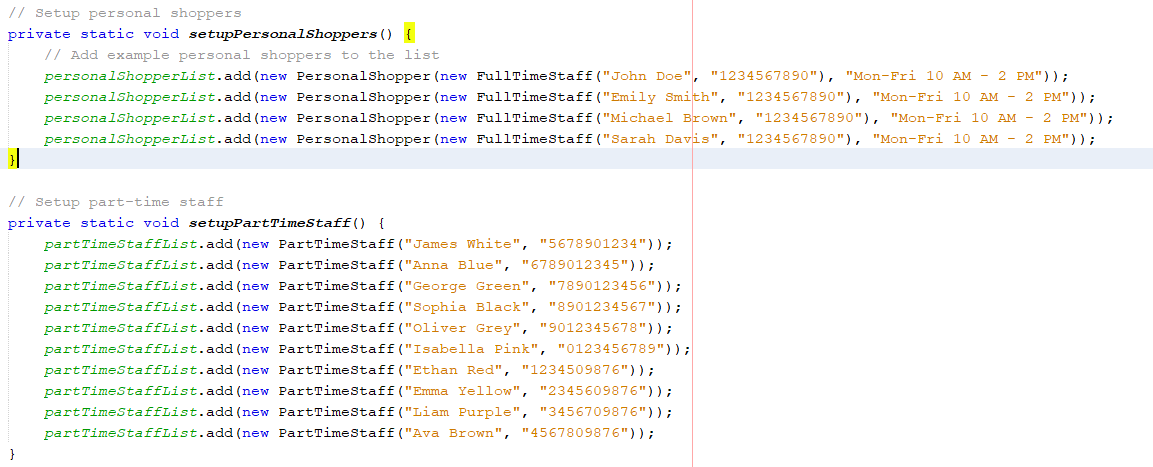
The implementation includes creating more ArrayLists to store the details of the full-time and part-time staffing while others include the availability for personal shopping support. A minimum of 4 records of full-time staff employees and 10 records of part-time employees are created as the dummy data. In testing, a full-time employee Alice will do a personal shopping of 30 minutes with a customer by the name Bob. This interaction is managed in Java code, and once the booking is completed, the program prints the username and phone number of the customer, the name of the staff member, and the time of the appointment. For example, "Customer: Appointments Pro: Jack:

Furthermore, the application contains features that are specific to certain departments of the store. All of the departments in FYNTS are operational and thus all aspects of the department such as the full-time and part-time employees, department workshops, and personal shopping appointments are included in the function of the system. In addition, the Food department of the store contains extra functions for the bakery boot camp administration, such as registration details and first-time customer discounts for full-time boot camps (Ispandi *et al.,* 2022). In total, these augmentations guarantee that the “Maximus” application will not only satisfy the general data necessity for controlling the staff and customers but also will consider the specific function of every department of the department store, which will make for suitable and comprehensive software.

#### 

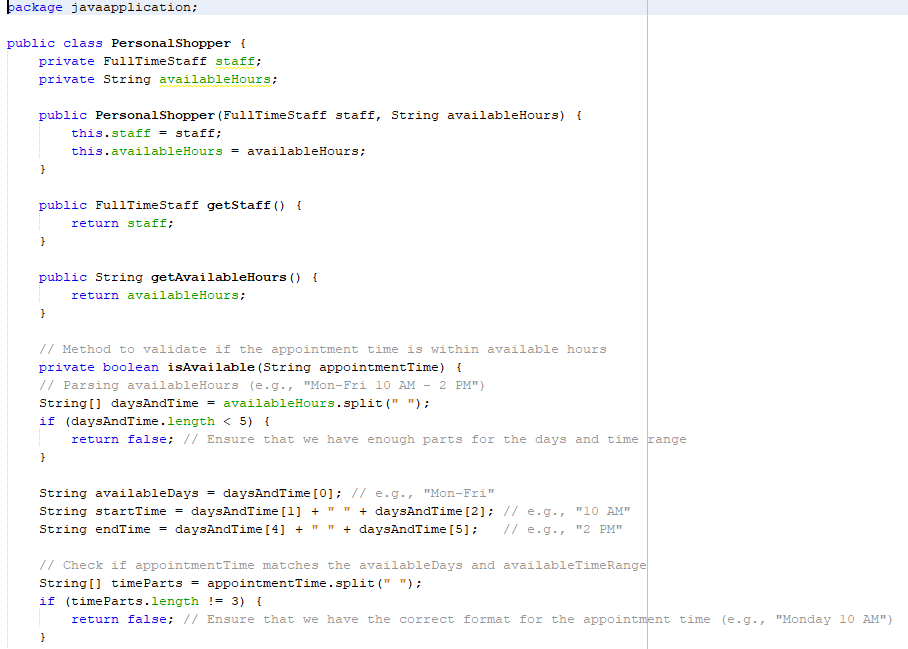
**Figure 11: Creating ArrayList**

This Java code defines a `StoreManagement` class that manages two lists. The two lists are personalShopperList and partTimeStaffList. Both of them are of type ‘ArrayList’ but Whereas, one contains ‘Instances of PersonalShopper’ The other contains ‘PartTimeStaff’. It is used to control and coordinate the full-time employed personal shoppers and part of the temporary staff in the department store to do extra jobs like, booking and staff management.



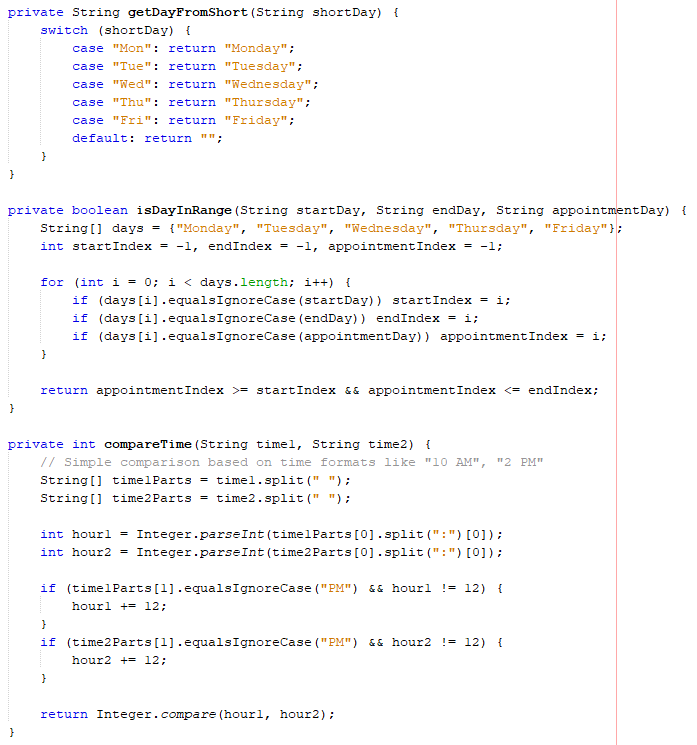
**Figure 12: Inserting Staff Data**

The `setupPersonalShoppers()` and `setupPartTimeStaff()` functions assign examples to certain lists namely the `personalShopperList` and the `partTimeStaffList. This fill `PersonalShopper` objects with fulltime employees like “John Doe” in their list and part time employees like “James White”. This setup is used to make an assumption about availability of staffs in the department and the personal shopper for the customer.



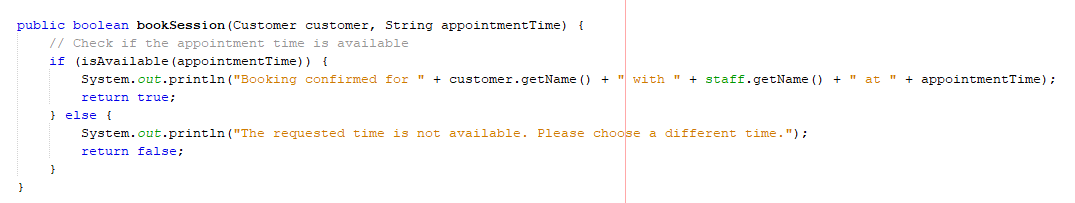
**Figure 13: Creating PersonalShopper class**

The `PersonalShopper` class defines a personal shopper in the store. It contains two key attributes `FullTimeStaff staff`, here, as the name suggests it involves the staffing and thus the staff that is involved is the personal shopper, `String availableHours` that indicates when the service is available, it, for instance, can be ‘Monday to Friday, 10 AM to 2 PM.’ It also has getter methods for these attributes through the class. The least the `isAvailable()` method verifies where a customer wants to take an appointment that whether it is in personal shopper’s working timetable or not.



**Figure 14: Validating Days**

The methods ensure the availability of personal shopper by consulting on the appointment days and time. This abstract functionalities as `getDayFromShort` convert short weeks’ days including “Mon” into the full names of the days like “Monday”. The `isDayInRange()` method checks whether an appointment is to take place in a certain number of days i.e between Monday – Friday through the index numbers of the days, the activity `compareTime()` seems to compare two times where the times given as 10 AM, 2 PM and the like has been converted to 24- hour format to help in validating time for an appointment in the given available time slots.



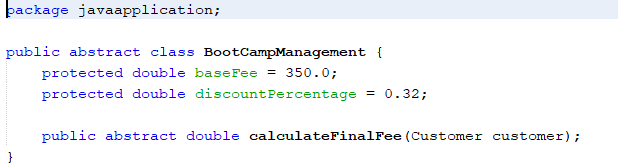
**Figure 15: Confirm Booking**

This figure explain an appointment for a customer to meet with a personal shopper is made through the `bookSession()` method. It checks the appointment that the user wants to schedule is available at that time using the `isAvailable()` function. If the time is valid, it sent an executive message of confirmation with the customer and the staff member’s name while if the time is not available, it tells the customer to change the time.

#### **Part C: Testing and Results**

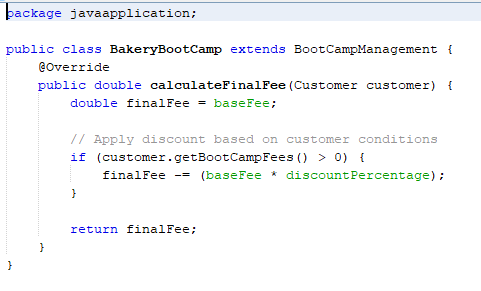
**Comprehensive Testing of the Application**

The implementation of the residual testing comprises the measurement of the application’s functionality, performance, and reliability. The test cases were developed and run to demonstrate the satisfaction of certain conditions or characteristics in the application and its functionality under particular conditions. The testing phase is very important because the major works on finding flaws or problems that are likely to be experienced in the software when in use.

****

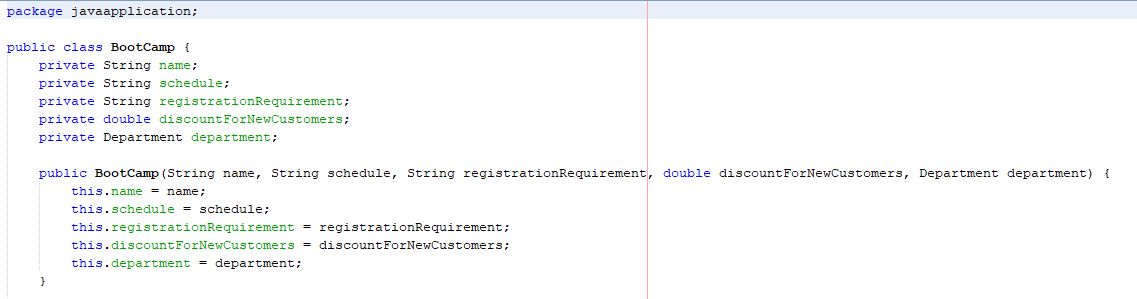
**Figure 16: Implementing Abstract class**

The ‘BootCampManagement’ abstract class provides a framework for fee management of boot camp. The purpose of this solution `baseFee` is defined as 350£ and `discountPercentage` is 32%. The class calculating the final Boot camp fee has an abstract method getFinalFee(Customer customer) this method gives the freedom to all sub-classes to allow a specific customer to receive specific discounts.

****

**Figure 17: BakeryBootCamp extends class**

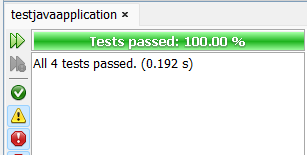
The `BakeryBootCamp` class is a subclass of the `BootCampManagement` abstract class, and this class implements the body of the `calculateFinalFee` method. It is the `baseFee` and if customer paid for any other boot camp than 32% is deducted from the total fee (the customer `getBootCampFees() > 0`). This implies that the final fee is reached after all the relevant arithmetic computations have been made and often any other additional discounts that maybe availed to the customer.

****

**Figure 18: Register New Customer**

It is a class that is used to control details of the boot camp which include but not limited to the name, the time it operates, the way customers are required to register and new comer discounts. These attributes are also set by the constructor and the boot camp is associated with a particular department in the company. This class also ensures that while registering a new customer, boot camp information is input correctly and coded sufficiently for the first-time enrolments.

**Query Execution:**



**Figure 19: jUnit test result**

Testing performed was accompanied by recording outcomes of every test case. These screenshots are to demonstrate that the application can work as expected and complete indicated operations effectively. For example, the screenshots of the insertion tests are the ability to insert the records and their appearance in the database. Modification and deletion tests are supported by screenshots that illustrate the difference in the database’s contents, taken before and after the tests.

**Discussion on Robustness and Efficiency**

The stability of the application was also checked at different stress levels and different scenarios. This encompasses cases such as when inputs are poorly entered when holding big data, and when various operations are being conducted. The application was shown to have strong qualities of robustness in regard to errors and exceptions – the application did not fail (Fülöp *et al.,* 2022). The response speed and the usage of some resources during various procedures were also performed and used to evaluate the efficiency. The application also retained an untainted response time with very little lag time as well as proper utilization of the systems’ resources.

**Improvements and Future Enhancements**

* Enhanced Error Handling: Proposing improvements of the error-handling strategies in order to react to unexpected inputs and more successfully handle the system’s errors.
* User Interface Improvements: Improve the Design of the user interface to facilitate an easier-to-use and enhanced user experience. This encompasses better navigation without having to scroll through a lot, and easily understandable error messages when using the forum.
* Additional Features: Additional features like search filter, export to other formats, and improved compatibility with another system for better performance.
* Performance Optimization: Final refinements of the code to improve its run time and efficiency of queries with the database to better support the application in cases more data-intensive and/or when carrying out more elaborate operations.
* Security Enhancements: Improvement of the security procedures to safeguard the information and to prevent intrusions. This includes such measures as encryption, strong authentication, and often security assessments.

# 5. Conclusion

The presented example of the application aimed at the development of the Maximus department store operating system proves the possibility of using OOP principles through the intentional construction of UML diagrams and the implementation of the administrational focus of the system in the format of object-oriented programming language Java. Organizing the classes and objects helped create a clear structure of the interrelations of the store. This project of Java application development was accomplished in order to map the requirements for managing staff and customer records, operationalizing record management, departmental work, and user interactions. Stress testing proved the application’s reliability and speed and identified improvement suggestions and future development opportunities.

# 

# 6. References

Abidin, Z.Z. and Zawawi, M.A.A., 2020. Oop-ar: Learn object oriented programming using augmented reality. *International Journal of Multimedia and Recent Innovation (IJMARI)*, *2*(1), pp.60-75.

Aerts, K., 2024, May. Improving Students’ Engagement and Learning Outcomes in a Primer Course on Object Oriented Programming in Java. In 2024 47th MIPRO ICT and Electronics Convention (MIPRO) (pp. 642-647). IEEE.

Dageförde, J.C., Winkelmann, H. and Kuchen, H., 2021, September. Free objects in constraint-logic object-oriented programming. In *Proceedings of the 23rd International Symposium on Principles and Practice of Declarative Programming* (pp. 1-13).

Fülöp, M.T., Udvaros, J., Gubán, Á. and Sándor, Á., 2022. Development of Computational Thinking Using Microcontrollers Integrated into OOP (Object-Oriented Programming). *Sustainability*, *14*(12), p.7218.

Ispandi, I., Atmaja, R.K., Sudradjat, A. and Ramadan, R., 2022. Building A Traditional Medicine Inventory Application With The Concept Of Object Oriented Programming Using The Java Programming Language. *JISICOM (Journal of Information System, Informatics and Computing)*, *6*(1), pp.194-208.

Khan, G., 2021. Traditional implementing usage of core programming languages in the object oriented programming. Asian Journal of Multidimensional Research, 10(10), pp.1167-1173.

Nasereddin, H.H. and Darwesh, A.J., 2020. An object oriented programming on encrypted database system (CryptDB). *Talent Development & Excellence*, *12*(1), pp.5140-5146.

Rojat, M.R., 2022. Pembelajaran Pemrograman Berorientasi Objek (Object Oriented Programming) Berbasis Project Based Learning. *Jurnal Portal Data*, *2*(7).

Santos, J.C., Zhang, X. and Mirakhorli, M., 2022, November. Counterfeit object-oriented programming vulnerabilities: an empirical study in Java. In Proceedings of the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security (pp. 21-28).

Singh, N., Chouhan, S.S. and Verma, K., 2021, October. Object oriented programming: Concepts, limitations and application trends. In *2021 5th International Conference on Information Systems and Computer Networks (ISCON)* (pp. 1-4). IEEE.

Winkelmann, H. and Kuchen, H., 2022, April. Constraint-logic object-oriented programming on the Java virtual machine. In Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing (pp. 1258-1267).