PROGRAMMING FOR SOFTWARE ENGINEERING
7COM1025

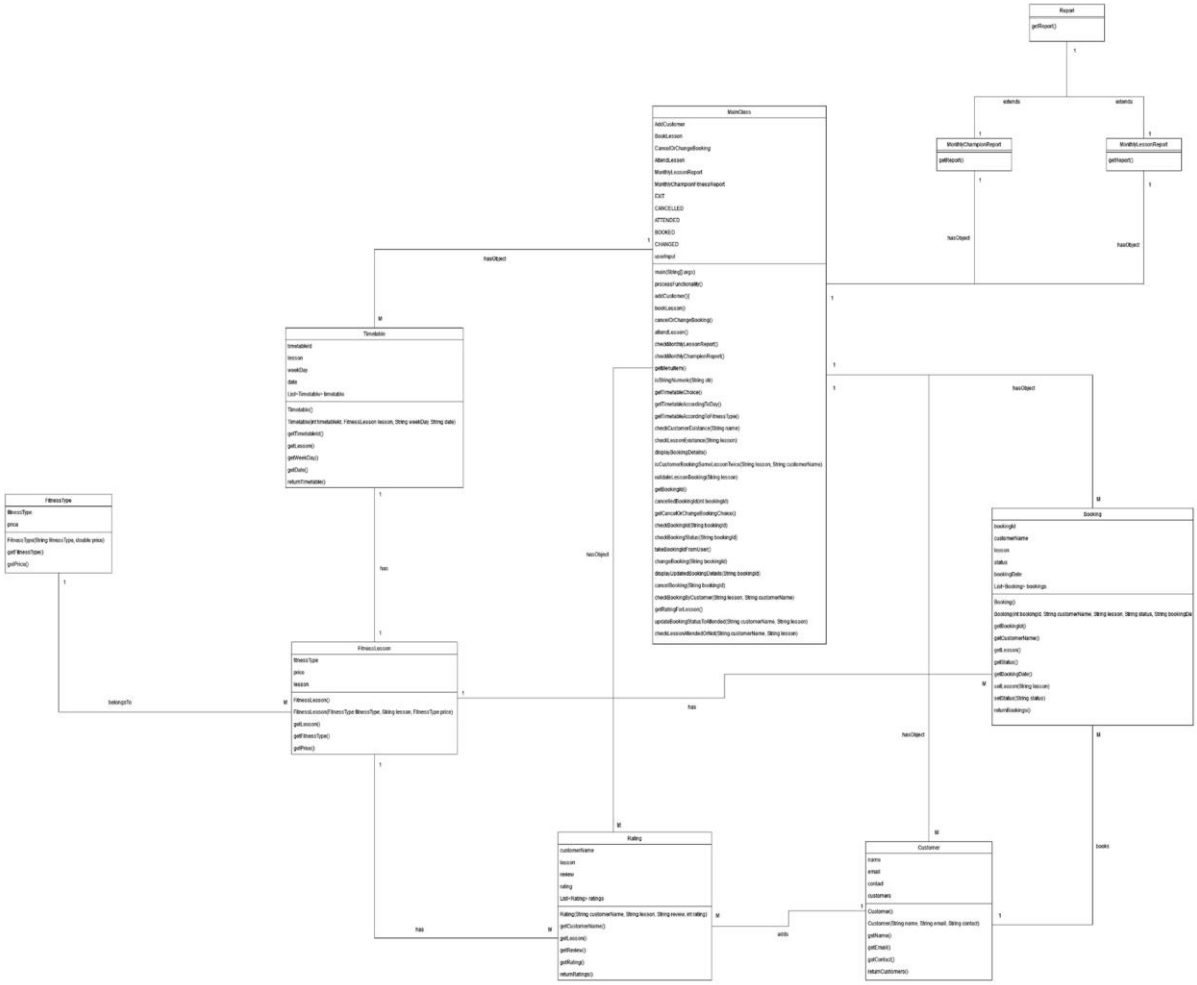
HARI KRISHNA ERAPARAJU
21089188

Any assumptions made about the system.

A few assumptions have been made in addition to the scenario description and these are mentioned below:

- ➔ The system will be designed using the CLI interface as it will offer easy interaction and access to the application by simply entering the option the user would like to choose.
- ➔ The options will be provided in a “menu-based” format making it easier for users to interact with the application.
- ➔ It has been also assumed that the staff will add the students in the database and students will not have an option to register themselves.

UML Class Diagram



The overall structure and design of your program

The program structure follows a ***hierarchical structure*** as it can be seen that each class has its own functions. In order to access the classes, they are accessed using Objects. In addition to this, documentation has been done to ensure the readability of the program.
As shown in the screenshot below, app object has been created which is used to call the function “processFunctionality” from MainClass

```
public static void main(String[] args) {
    MainClass app = new MainClass();
    app.processFunctionality();
}
```

Code conventions have been taken into consideration such as naming conventions for better readability and understanding. As shown in the screenshot below, naming conventions have been used for classes. For example, customer class is used to store customer information such as name, email, contact. Similarly, other classes also use naming conventions to make it easily readable.

```
public class Customer {

    private String name;
    private String email;
    private String contact;

    public static List<Customer> customers = new ArrayList<>();

    public Customer() {
    }

    public Customer(String name, String email, String contact) {
        this.name = name;
        this.email = email;
        this.contact = contact;
    }

    public String getName() {
        return name;
    }

    public String getEmail() {
        return email;
    }

    public String getContact() {
        return contact;
    }
```

All the classes and functions follow a ***Single Responsibility Principle(SRP)*** as it can be seen that each class and function only perform specific things.

Any design patterns/design principles used.

Design Pattern: Factory Method Pattern

Factory Method Design Pattern has been used which is one of the most common and widely used design patterns (Wedyan and Abufakher, 2020). This pattern allows the programmer to create objects without exposing the logic and refer to the new object through a common interface. The report.java class uses the factory design pattern as shown below.

Report.java

```
abstract class Report {

    abstract void getReport();
}
```

The "MonthlyChampionReport.java" and "MonthlyLessonReport" class also uses a factory design pattern as shown below.

MonthlyChampionReport.java

```
public class MonthlyChampionReport extends Report{

    @Override
    void getReport() {
        List<Booking> bookingList = Booking.returnBookings();
        List<Timetable> timetable = Timetable.returnTimetable();

        double totalIncome = 0.0;
        String lesson = "";
        System.out.println();
        System.out.printf( format: "%-10s %-50s %-50s %n",  args: "S.No.", args: "Lesson",  args: "Total Income Generated");
        System.out.println( x: "------------------------------------------------------------------------------------------------------");
        for(int i=0; i<timetable.size(); i++){
            for(int j=0; j<bookingList.size(); j++){

                if(timetable.get( index: i).lesson.getLesson().equalsIgnoreCase( anotherString: bookingList.get( index: j).getLesson())){
                    totalIncome = totalIncome +  timetable.get( index: i).lesson.getPrice().getPrice();
                    lesson = bookingList.get( index: j).getLesson();
                }
            }
            if(!timetable.get( index: i).lesson.getLesson().equalsIgnoreCase( anotherString: lesson)){
                totalIncome = 0;
            }
            System.out.printf( format: "%-10s %-50s $%-50s %n%n", (i+1), args: timetable.get( index: i).lesson.getLesson(), args: totalIncome);
        }
    }

}
```

MonthlyLessonReport.java

```java
public class MonthlyLessonReport extends Report {
    @Override
    void getReport() {
        List<Booking> bookingList = Booking.returnBookings();
        List<Timetable> timetable = Timetable.returnTimetable();
        List<Rating> rating = Rating.returnRatings();
        int noOfCust = 0;
        int noOfRatings = 0;
        int totalRating = 0;

        String lesson = "";
        String lesson1 = "";
        System.out.println();
        System.out.printf("%-10s %-50s %-50s %-50s %n", "S.No.", "Lesson", "Number Of Customers", "Average Ratings");
        System.out.println("----------------------------------------------------------------------------------------------------------------------------");
        for(int i=0; i<timetable.size(); i++){

            //Get Number of customer who booked a lesson
            for(int j=0; j<bookingList.size(); j++){

                if(timetable.get(i).lesson.getLesson().equalsIgnoreCase(bookingList.get(j).getLesson())){
                    noOfCust = noOfCust + 1;
                    lesson = bookingList.get(j).getLesson();
                }
            }

            //Average Rating
            for(int k=0; k<rating.size(); k++){

                if(timetable.get(i).lesson.getLesson().equalsIgnoreCase(rating.get(k).getLesson())){
                    noOfRatings = noOfRatings + 1;
                    totalRating = totalRating + rating.get(k).getRating();
                    lesson1 = rating.get(k).getLesson();
                }
```

JUnit testing

S.No.	Test Case	Test Data	Expected Output	Actual Output	Result
1	To check whether there are 32 lessons in the timetable		It should display a success message.	It displays a success message.	Pass

```java
@Test
public void checkNoOfLessons() {
    System.out.println("\n\nTest Case : To check whether there are 32 lessons in the timetable");

    List<Timetable> timetable = Timetable.returnTimetable();

    if(timetable.size() == 32){
        System.out.println("Result : Test Passed");
        assert true;
        return;
    }else{
        fail("Test Failed");
    }
}
```

Tests passed: 100.00 %

All 7 tests passed. (0.3 s)

MainTestClass passed

```
Test Case : To check whether there are 32 lessons in the timetable
Result : Test Passed
```

S.No.	Test Case	Test Data	Expected Output	Actual Output	Result
2	To search timetable by Fitness Type	Fitness Type = “Yoga”	It should display the timetable of “yoga” lessons.	It displays a timetable of “yoga” lessons.	Pass

```java
@Test
public void searchTimetableByFitnessType() {
    System.out.println( x: "\n\nTest Case : To search timetable by Fitness Type");

    String fitnessType = "Yoga";

    List<Timetable> timetable = Timetable.returnTimetable();
    System.out.println();
    System.out.printf( format: "%-30s %-30s %-30s %-30s %-30s %n",  args: "Fitness Type",  args: "Lesson",  args: "Price",  args: "Day",  args: "Date");
    System.out.println( x: "------------------------------------------------------------------------------------------------------------------------------------------------------------------------------");

    for(int i=0; i<timetable.size(); i++){
        if(timetable.get( index: i).lesson.getFitnessType().getFitnessType().equalsIgnoreCase( anotherString: fitnessType)){
            System.out.printf( format: "%-30s %-30s %-30s %-30s %-30s %n%n",  args: timetable.get( index: i).lesson.getFitnessType().getFitnessType(),
                    args: timetable.get( index: i).lesson.getLesson(),  args: timetable.get( index: i).lesson.getPrice().getPrice(),  args: timetable.get( index: i).getWeekDay(),
                    args: timetable.get( index: i).getDate());
        }
    }
    if(!timetable.isEmpty()){
        System.out.println( x: "Result: Test Passed");
        assert true;
        return;
    }else{
        fail( message: "Test Failed");
    }
}
```

S.No.	Test Case	Test Data	Expected Output	Actual Output	Result
3	To search timetable ofSunday	Day = “Sunday”	It should display the timetable of “sunday”.	It displays the timetable of “sunday”.	Pass

```java
@Test
public void searchTimetableOfSunday() {
    System.out.println("\n\nTest Case : To search timetable of Sunday");

    String day = "sunday";

    List<Timetable> timetable = Timetable.returnTimetable();
    System.out.println();
    System.out.printf("%-30s %-30s %-30s %-30s %-30s %n", "Fitness Type", "Lesson", "Price", "Day", "Date");
    System.out.println("-----------------------------------------------------------------------------------------------------------------------------------------------------------");

    for(int i=0; i<timetable.size(); i++){
        if(timetable.get(i).getWeekDay().equalsIgnoreCase(anotherString:day)){
            System.out.printf("%-30s %-30s $%-30s %-30s %-30s %n%n", timetable.get(i).lesson.getFitnessType().getFitnessType(),
                    timetable.get(i).lesson.getLesson(), timetable.get(i).lesson.getPrice().getPrice(), timetable.get(i).getWeekDay(),
                    timetable.get(i).getDate());
        }
    }
    if(!timetable.isEmpty()){
        System.out.println("Result: Test Passed");
        assert true;
        return;
    }else{
        fail("Test Failed");
    }
}
```

Tests passed: 100.00 %

All 7 tests passed. (0.3 s)

> MainTestClass passed

```
Test Case : To search timetable of Sunday

Fitness Type                   Lesson                         Price                          Day                            Date
-----------------------------------------------------------------------------------------------------------------------------------------------------------
ZUMBA                          Zumba Lesson 1                 $62.0                           Sunday                         12 March,2023

BODYSCULPT                     Bodysculpt Lesson 1            $59.0                           Sunday                         12 March,2023

Yoga                           Yoga Lesson 2                  $75.0                           Sunday                         19 March,2023

SPIN                           Spin Lesson 2                  $54.0                           Sunday                         19 March,2023

ZUMBA                          Zumba Lesson 3                 $62.0                           Sunday                         26 March,2023

BODYSCULPT                     Bodysculpt Lesson 3            $59.0                           Sunday                         26 March,2023

Yoga                           Yoga Lesson 4                  $75.0                           Sunday                         2 April,2023

SPIN                           Spin Lesson 4                  $54.0                           Sunday                         2 April,2023

ZUMBA                          Zumba Lesson 5                 $62.0                           Sunday                         9 April,2023

BODYSCULPT                     Bodysculpt Lesson 5            $59.0                           Sunday                         9 April,2023

Yoga                           Yoga Lesson 6                  $75.0                           Sunday                         16 April,2023

SPIN                           Spin Lesson 6                  $54.0                           Sunday                         16 April,2023

ZUMBA                          Zumba Lesson 7                 $62.0                           Sunday                         23 April,2023

BODYSCULPT                     Bodysculpt Lesson 7            $59.0                           Sunday                         23 April,2023

Yoga                           Yoga Lesson 8                  $75.0                           Sunday                         30 April,2023

SPIN                           Spin Lesson 8                  $54.0                           Sunday                         30 April,2023

Result: Test Passed
```

S.No.	Test Case	Test Data	Expected Output	Actual Output	Result
4	To search timetable of Saturday	Day = “saturday”	It should display the timetable of “saturday”.	It displays the timetable of “Saturday”.	Pass

```java
@Test
public void searchTimetableOfSaturday() {
    System.out.println("\n\nTest Case : To search timetable of Saturday");

    String day = "saturday";

    List<Timetable> timetable = Timetable.returnTimetable();
    System.out.println();
    System.out.printf("%-30s %-30s %-30s %-30s %-30s %n", "Fitness Type", "Lesson", "Price", "Day", "Date");
    System.out.println("-----------------------------------------------------------------------------------------------------------------------------------------------------------");

    for(int i=0; i<timetable.size(); i++){
        if(timetable.get(i).getWeekDay().equalsIgnoreCase(anotherString:day)){
            System.out.printf("%-30s %-30s $%-30s %-30s %-30s %n%n", timetable.get(i).lesson.getFitnessType().getFitnessType(),
                    timetable.get(i).lesson.getLesson(), timetable.get(i).lesson.getPrice().getPrice(), timetable.get(i).getWeekDay(),
                    timetable.get(i).getDate());
        }
    }
    if(!timetable.isEmpty()){
        System.out.println("Result: Test Passed");
        assert true;
        return;
    }else{
        fail("Test Failed");
    }
}
```

S.No.	Test Case	Test Data	Expected Output	Actual Output	Result
5	To book a lesson	customerName, lesson	It should book a lesson.	It books lessons for customers and displays booking details.	Pass

```
@Test
public void bookClass() {
    System.out.println( x: "\n\nTest Case : To book a lesson");
    String customerName = "John";
    String lesson  = "yoga lesson 3";
    Random rand = new Random();
    int low = 100;
    int high = 200;
    int bookingId = rand.nextInt(high-low) + low;

    String date = String.valueOf( obj: java.time.LocalDate.now());

    Booking booking = new Booking(bookingId,customerName, lesson,  status: "booked", bookingDate: date);
    Booking.bookings.add( e: booking);

    displayBookingDetails();
    System.out.println( x: "Result: Test Passed");
}
```

```java
private static void displayBookingDetails(){
    System.out.println("\nYour Booking Details are : \n");

    List<Booking> bookingList = Booking.returnBookings();
    List<Timetable> timetable = Timetable.returnTimetable();

    for(int i=bookingList.size(); i>0; i--){

        System.out.printf("%-20s %-30s %-30s %-30s %-30s %n", "Booking No.", "Customer Name", "Lesson", "Status", "Booking Date");
        System.out.println("-----------------------------------------------------------------------------------------------------------------------------------");
        System.out.printf("%-20s %-30s %-30s %-30s %-30s %n", bookingList.get(bookingList.size()-1).getBookingId(),
                bookingList.get(bookingList.size()-1).getCustomerName(), bookingList.get(bookingList.size()-1).getLesson(),
                bookingList.get(bookingList.size()-1).getStatus(), bookingList.get(bookingList.size()-1).getBookingDate());

        System.out.println("\nLesson Details are : \n");
        for(int j=0; i<timetable.size(); j++){
            if(timetable.get(j).lesson.getLesson().equalsIgnoreCase(bookingList.get(bookingList.size()-1).getLesson())){
                System.out.printf("%-30s %-30s %-30s %-30s %-30s %n", "Fitness Type", "Lesson", "Price", "Day", "Date");
                System.out.println("-----------------------------------------------------------------------------------------------------------------------------------");
                System.out.printf("%-30s %-30s $%-30s %-30s %-30s %n%n", timetable.get(j).lesson.getFitnessType().getFitnessType(),
                        timetable.get(j).lesson.getLesson(), timetable.get(j).lesson.getPrice().getPrice(), timetable.get(j).getWeekDay(),
                        timetable.get(j).getDate());
                break;
            }
        }
        break;
    }
}
```

Tests passed: 100.00 %

The test passed. (0.121 s)

- MainTestClass passed
  - bookClass passed (0.021 s)

```
Test Case : To book a lesson

Your Booking Details are :

Booking No.          Customer Name                  Lesson                         Status                         Booking Date
-----------------------------------------------------------------------------------------------------------------------------------
178                  John                           yoga lesson 3                  booked                         2023-03-17

Lesson Details are :

Fitness Type                   Lesson                         Price                          Day                            Date
-----------------------------------------------------------------------------------------------------------------------------------
Yoga                           Yoga Lesson 3                  $75.0                          Saturday                       25 March,2023

Result: Test Passed
```

S.No.	Test Case	Test Data	Expected Output	Actual Output	Result
6	To check lesson capacity before booking a lesson	Lesson, capacity=5	It should display an error message, if the lesson is booked by 5 customers.	It displays an error message, if the lesson is booked by 5 customers.	Pass

```java
@Test
public void checkLessonCapacity(){
    System.out.println("\n\nTest Case : To check lesson capacity before booking a lesson");
    String lesson = "yoga lesson 3";
    addFiveBookingForLesson(lesson);

    int capacity = 5;

    List<Booking> bookingList = Booking.returnBookings();
    int count = 0;
    String status = "booked";
    for(int i=0; i<bookingList.size(); i++){
        if(bookingList.get(i).getLesson().equalsIgnoreCase(lesson) && bookingList.get(i).getStatus().equalsIgnoreCase(status)){
            count = count + 1;
        }
    }

    if(count >= capacity){
        System.out.println("\nError : "+lesson+" is booked by 5 customers. Please change the lesson to book.");
        System.out.println("Result: Test Passed");
        assert true;
        return;
    }else{
        fail("Test Failed");
    }
}
```

Tests passed: 100.00 %

All 7 tests passed. (0.281 s)

MainTestClass passed

```
Test Case : To check lesson capacity before booking a lesson

Error : yoga lesson 3 is booked by 5 customers. Please change the lesson to book.
Result: Test Passed
```

S.No.	Test Case	Test Data	Expected Output	Actual Output	Result
7	To prove that, a customer cannot book the same lesson twice.	Lesson, customerName	It should display an error message, if the customer books the same lesson twice.	It displays an error message, if the customer books the same lesson twice.	Pass

```
@Test
public void validateCustomerToBookLessonTwice(){
    System.out.println( x: "\n\nTest Case : To prove that, a customer cannot book the same lesson twice.");
    String lesson = "yoga lesson 3";
    String customerName = "Allen";

    addFiveBookingForLesson(lesson);

    List<Booking> bookingList = Booking.returnBookings();
    String status = "booked";
    boolean value = false;

    for(int i=0; i<bookingList.size(); i++){
        boolean lessonValue = bookingList.stream().anyMatch(o -> lesson.equalsIgnoreCase( anotherString: o.getLesson()));
        boolean statusValue = bookingList.stream().anyMatch(o -> status.equalsIgnoreCase( anotherString: o.getStatus()));
        boolean custValue = bookingList.stream().anyMatch(o -> customerName.equalsIgnoreCase( anotherString: o.getCustomerName()));

        if(lessonValue && custValue && statusValue){
            value = true;
        }
    }

    if(value){
        System.out.println("\nError: '"+customerName+"' cannot book the same lesson twice");
        System.out.println( x: "Result: Test Passed");
        assert true;
        return;
    }else{
        fail( message: "Test Failed");
    }
}
```

Tests passed: 100.00 %

All 7 tests passed. (0.291 s)

MainTestClass passed

```
Test Case : To prove that, a customer cannot book the same lesson twice.

Error: 'Allen' cannot book the same lesson twice
Result: Test Passed
```

Any refactoring used during the development of the system.

Refactoring is a technique used by programmers in order to improve internal code by making many small changes without altering the code's external behaviour. While developing the program, refactoring has been used as show in the below instances:

```
private static final String CANCELLED = "cancelled";
private static final String ATTENDED = "attended";
private static final String BOOKED = "booked";
private static final String CHANGED = "changed";
```

```
//Save Booking
Booking booking = new Booking(bookingId, customerName: name, lesson,  status: BOOKED, bookingDate: date);
Booking.bookings.add( e: booking);
```

```
//Update Bookings Details
for(int i=0; i<bookingList.size(); i++){
    if(String.valueOf( i: bookingList.get( index: i).getBookingId()).equalsIgnoreCase( anotherString: bookingId)){
        bookingList.get( index: i).setLesson(lesson);
        bookingList.get( index: i).setStatus( status: CHANGED);
    }
}
```

```
// Cancel Booking
List<Booking> bookingList = Booking.returnBookings();
for(int i=0; i<bookingList.size(); i++){
    if(String.valueOf( i: bookingList.get( index: i).getBookingId()).equalsIgnoreCase( anotherString: bookingId)){
        bookingList.get( index: i).setStatus( status: CANCELLED);
    }
}
```

```
for(int i=0; i<bookingList.size(); i++){
    if(bookingList.get( index: i).getCustomerName().equalsIgnoreCase( anotherString: customerName) && bookingList.get( index: i).getLesson().equalsIgnoreCase( anotherString: lesson)){
        bookingList.get( index: i).setStatus( status: ATTENDED);
    }
}
```

As it can be seen, constants have been declared without changing the external behaviour of the program. These constants are used to access the classes such as Cancelled, Attended and other classes.