FACULTY OF SCIENCE, ENGINEERING, AND COMPUTING

School of Computer Science & Mathematics

BSc DEGREE

IN

SOFTWARE ENGINEERING

Programming 3 Data Structures & Design Patterns

Course Work

Name: U L T I Perera

ID Number: K2377403 / E216750

Content

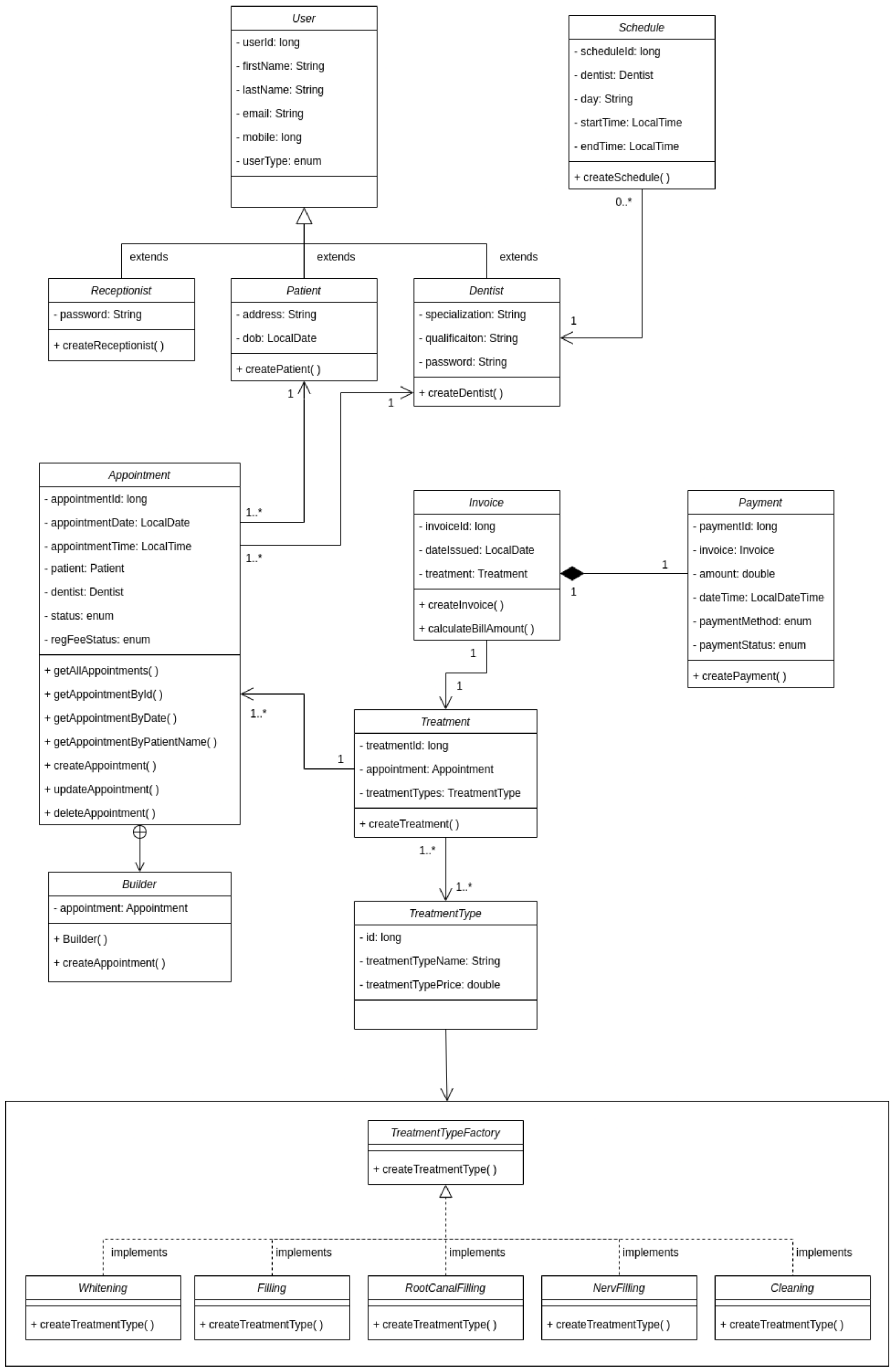
1. Introduction

ToothCare application is a web-based software application which is the course work for the Programming 3: Data Structures and Design Patterns. This full stack web application is developed using Java Spring boot framework for the Backend (Server Side) and React JavaScript library with TypeScript as the language for the Frontend (Client Side). The aim of this course work is to showcase the application and the usage of the Data Structures and Design Patterns with Object Oriented Programming (OOP) concepts in a software development environment,

2. Class Diagram

2.1. Class Diagram

Following is the Class diagram designed for the ToothCare application with the identified classes and the fields and methods.



2.2. Key decisions

Following are the key decisions made when designing the class diagram for the ToothCare web application.

- Identifying classes
  - In the ToothCare appointment booking application, the classes identified are,
    - User
      - As the superclass for every user type to generalize and represent common attributes and methods.
    - Dentist
      - To model all Dentists attributes and methods related to the dentists in the system.
    - Receptionist
      - To model Receptionists attributes and methods in the system.
    - Patient
      - To model Patients attributes and methods in the system.
    - Appointment
      - To model Appointment related attributes and methods in the system.
    - Schedule
      - To model Receptionists attributes and methods in the system.
    - Treatment
      - To model treatment related attributes and methods.
    - TreatmentType
      - To model all treatment types related attributes and methods.
    - Invoice
      - To model invoice related attributes and methods.
    - Payment
      - To model invoice related attributes and methods.
- Defining attributes
  - In User superclass, defined the common attributes and methods related to all user types (Dentist subclass, Receptionist subclass, Patient subclass).
- Defining visibility of class members
  - decided attributes in each class private to encapsulate them and use getters and setters to access them.
  - All made all methods publicly accessible through the application.
- Establishing relationships
  - Connected the Schedule class with Dentist class to represent the association “Schedule has a Dentist”.
  - Connected the Patient class with Appointment class to represent the association “Appointment should have a Patient”.
  - Connected the Dentist class with Appointment class to represent the association “Appointment should have Dentist assigned to it”.
  - Connected the Payment class with Invoice class to represent the aggregation that the “Payment should have an Invoice to exist”.
  - Connected the Invoice class with Treatment class to represent the association “Invoice should have a Treatment”.
  - Connected Treatment class with TreatmentType class to represent the association “Treatment should have objects of TreatmentType”
  - Connected Treatment class with the Appointment class to represent the association that the “Treatment has a Appointment assigned it to”.

  - Connected Whitening, Cleaning, Filling, Nerve Filling, Root Canal Therapy concrete classes with TreatmentTypeFactory interface to represenet the implementation of TreatmentTypeFactory on concrete classes.
- Defining cardinality
  - A Dentist can have 0 or more Schedules.
  - An Appointment should have 1 Patient while a Patient can have 1 or more Appointments.
  - An Appointment should have 1 Dentist assigned to it while a Dentist can have 1 or more Appointments.
  - An Appointment should have 1 Treatment while a Treatment can be there for 1 or more Appointments.
  - A Treatment should have 1 or more TreatmentTypes while a TreatmentType can be there for 1 or more Treatments.
  - One Treatment can have only 1 Invoice while an Invoice also can have only 1 Appointment.
  - One Payment can have only 1 Invoice while an Invoice can be assigned to 1 Payment.

3. Source Code of The Proposing System

3.1. Source Code

The source code for the application can be found on the below provided GitHub link. Please follow the below mentioned instructions (Readme.md file contains the same instructions) to get the application up and running.

GitHub: ToothCare-Programming-3-CW

I. Use the following command can be used to clone the git repository to the local computer.

```
git clone https://github.com/TharishaPerera/cw-programming-3.git
```

II. Open the 'toothcare' directory from your IDE and run the application.

III. Open a terminal window in the 'frontend' directory and run the following commands.

```
npm install && npm run dev
```

IV. Open web browser and go to 'http://localhost:5173" to access the application.

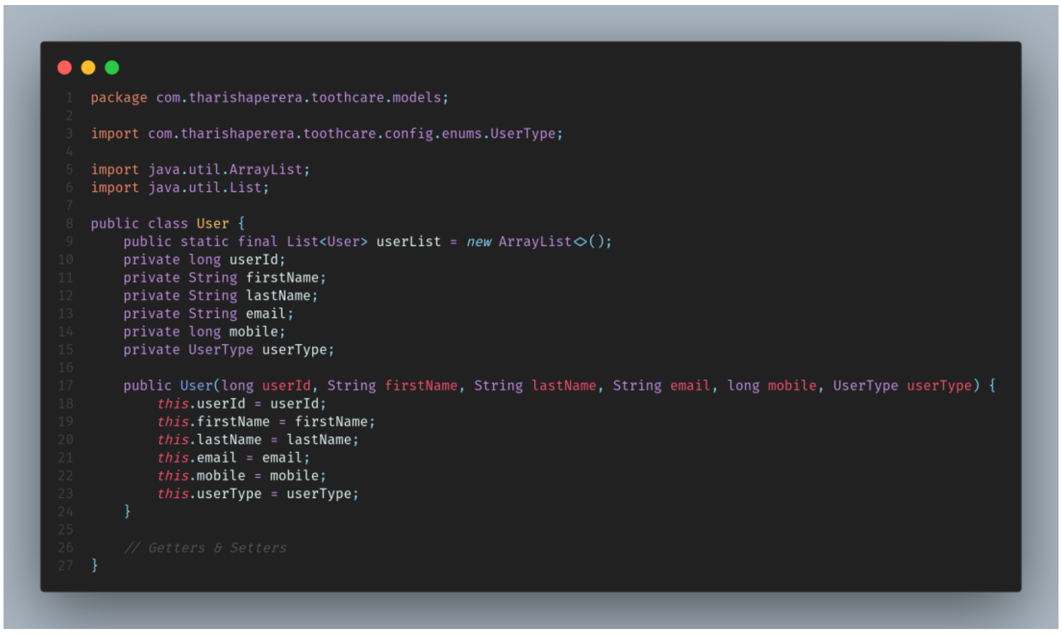
V. Use gayani@toothcare.com and "gayani@1234" as credentials to log into the system.

NOTE: *If the given credentials aren't working, please go back to the base URL and click on the "Get Started" button again to initialize the data.*

3.2. Classes and Objects

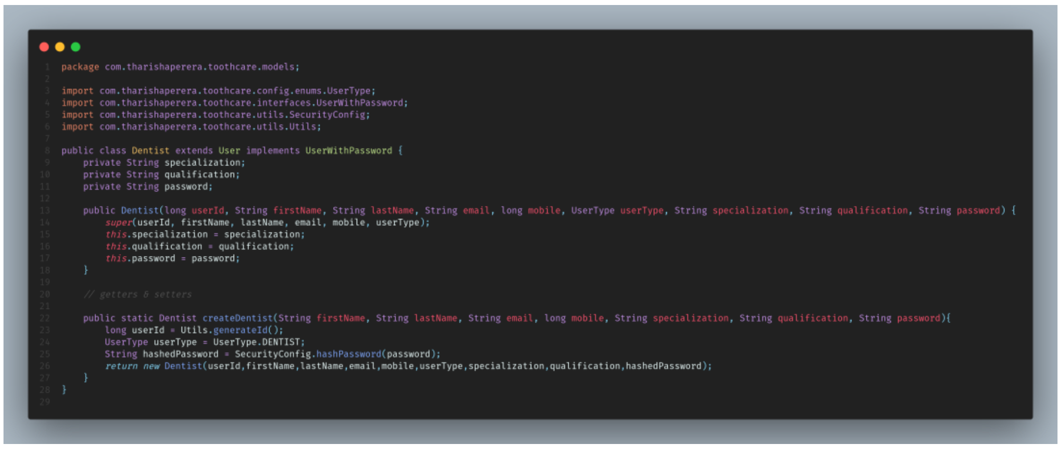
In this section, the classes that are defined in the application and the usage of their objects throughout the application will be discussed.

1. User



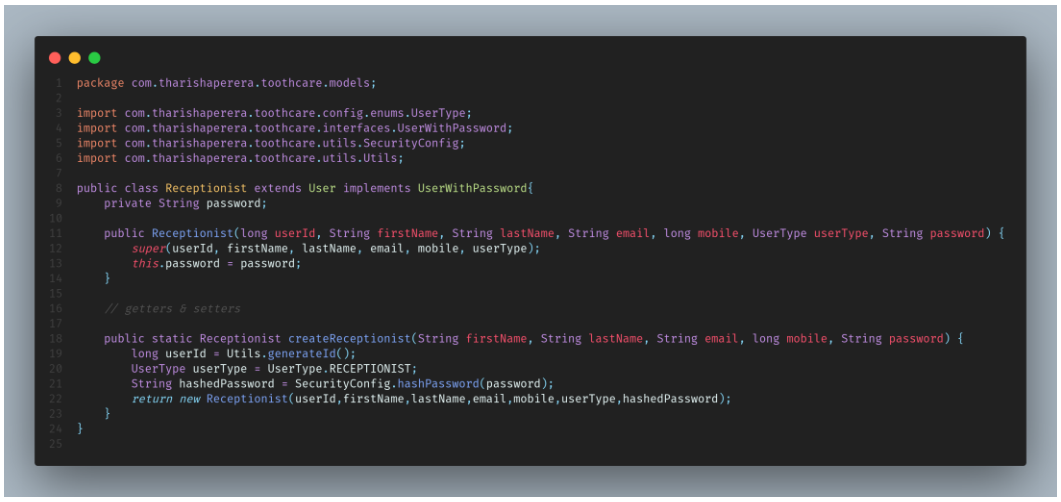
```java
package com.tharishaperera.toothcare.models;

import com.tharishaperera.toothcare.config.enums.UserType;

import java.util.ArrayList;
import java.util.List;

public class User {
    public static final List<User> userList = new ArrayList<>();
    private long userId;
    private String firstName;
    private String lastName;
    private String email;
    private long mobile;
    private UserType userType;

    public User(long userId, String firstName, String lastName, String email, long mobile, UserType userType) {
        this.userId = userId;
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.mobile = mobile;
        this.userType = userType;
    }

    // Getters & Setters
}
```

2. Dentist



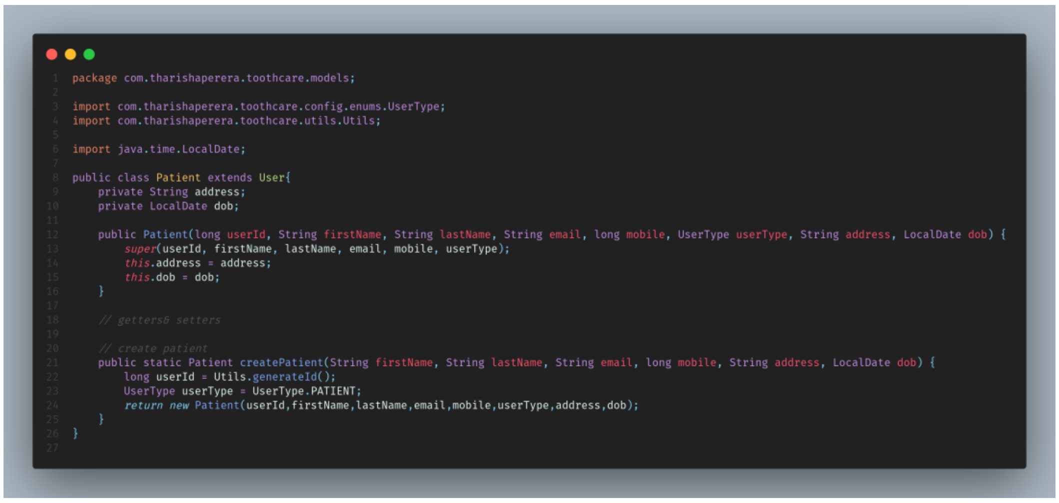
```java
package com.tharishaperera.toothcare.models;

import com.tharishaperera.toothcare.config.enums.UserType;
import com.tharishaperera.toothcare.interfaces.UserWithPassword;
import com.tharishaperera.toothcare.utils.SecurityConfig;
import com.tharishaperera.toothcare.utils.Utils;

public class Dentist extends User implements UserWithPassword {
    private String specialization;
    private String qualification;
    private String password;

    public Dentist(long userId, String firstName, String lastName, String email, long mobile, UserType userType, String specialization, String qualification, String password) {
        super(userId, firstName, lastName, email, mobile, userType);
        this.specialization = specialization;
        this.qualification = qualification;
        this.password = password;
    }

    // getters & setters

    public static Dentist createDentist(String firstName, String lastName, String email, long mobile, String specialization, String qualification, String password){
        long userId = Utils.generateId();
        UserType userType = UserType.DENTIST;
        String hashedPassword = SecurityConfig.hashPassword(password);
        return new Dentist(userId,firstName,lastName,email,mobile,userType,specialization,qualification,hashedPassword);
    }
}
```

3. Receptionist



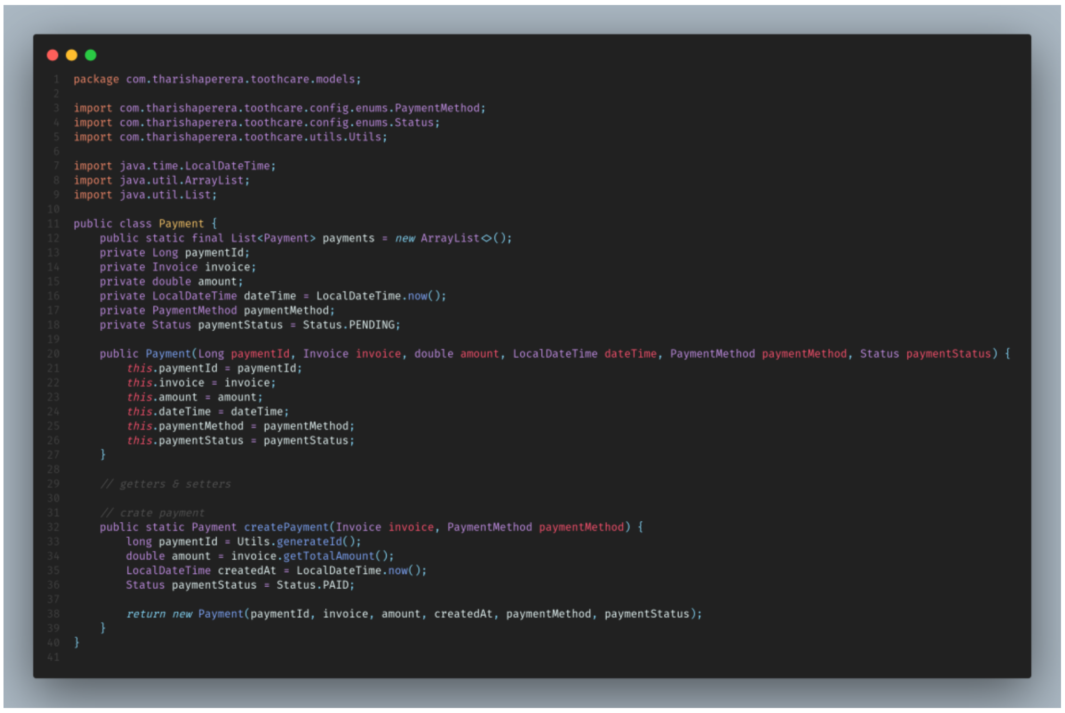
```java
package com.tharishaperera.toothcare.models;

import com.tharishaperera.toothcare.config.enums.UserType;
import com.tharishaperera.toothcare.interfaces.UserWithPassword;
import com.tharishaperera.toothcare.utils.SecurityConfig;
import com.tharishaperera.toothcare.utils.Utils;

public class Receptionist extends User implements UserWithPassword{
    private String password;

    public Receptionist(long userId, String firstName, String lastName, String email, long mobile, UserType userType, String password) {
        super(userId, firstName, lastName, email, mobile, userType);
        this.password = password;
    }

    // getters & setters

    public static Receptionist createReceptionist(String firstName, String lastName, String email, long mobile, String password) {
        long userId = Utils.generateId();
        UserType userType = UserType.RECEPTIONIST;
        String hashedPassword = SecurityConfig.hashPassword(password);
        return new Receptionist(userId,firstName,lastName,email,mobile,userType,hashedPassword);
    }
}
```

4. Patient



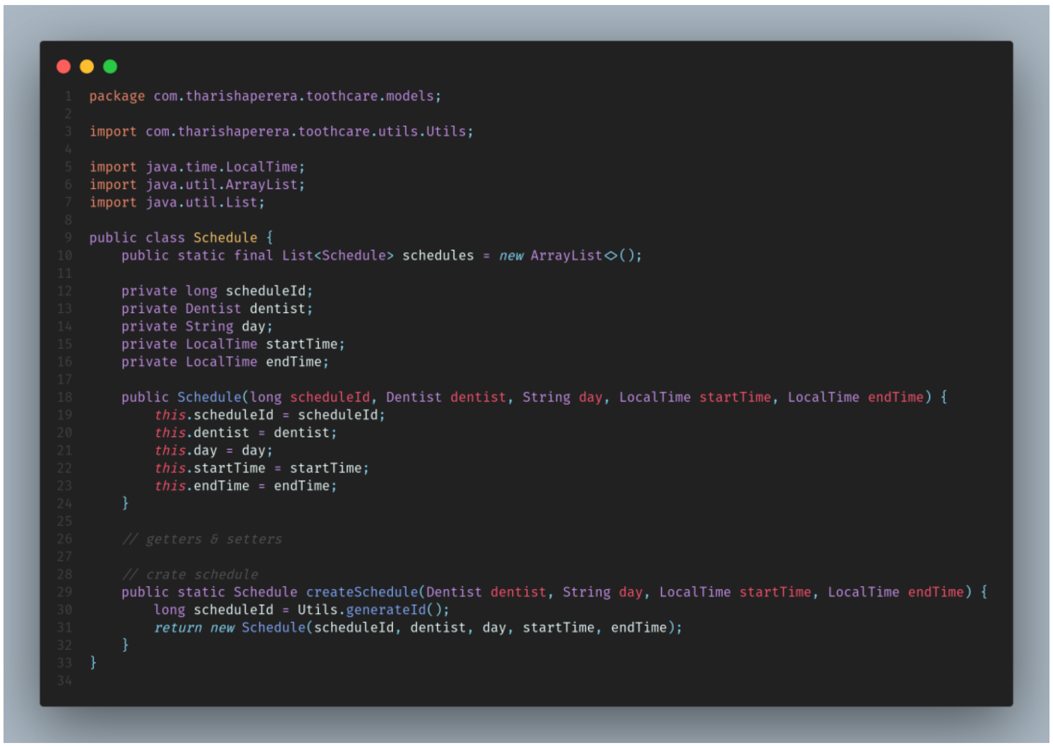
```java
package com.tharishaperera.toothcare.models;

import com.tharishaperera.toothcare.config.enums.UserType;
import com.tharishaperera.toothcare.utils.Utils;

import java.time.LocalDate;

public class Patient extends User{
    private String address;
    private LocalDate dob;

    public Patient(long userId, String firstName, String lastName, String email, long mobile, UserType userType, String address, LocalDate dob) {
        super(userId, firstName, lastName, email, mobile, userType);
        this.address = address;
        this.dob = dob;
    }

    // getters& setters

    // create patient
    public static Patient createPatient(String firstName, String lastName, String email, long mobile, String address, LocalDate dob) {
        long userId = Utils.generateId();
        UserType userType = UserType.PATIENT;
        return new Patient(userId,firstName,lastName,email,mobile,userType,address,dob);
    }
}

```

5. Payment



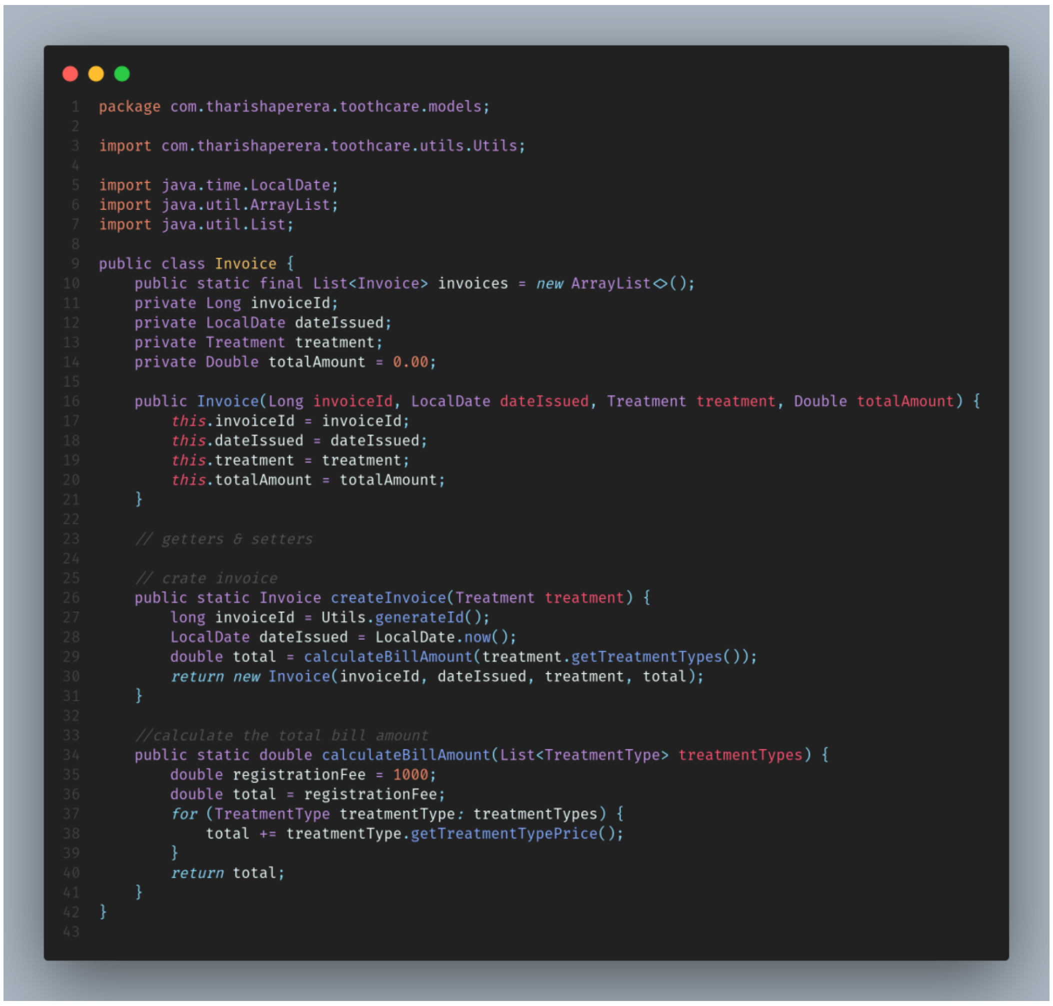
```java
package com.tharishaperera.toothcare.models;

import com.tharishaperera.toothcare.config.enums.PaymentMethod;
import com.tharishaperera.toothcare.config.enums.Status;
import com.tharishaperera.toothcare.utils.Utils;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

public class Payment {
    public static final List<Payment> payments = new ArrayList<>();
    private Long paymentId;
    private Invoice invoice;
    private double amount;
    private LocalDateTime dateTime = LocalDateTime.now();
    private PaymentMethod paymentMethod;
    private Status paymentStatus = Status.PENDING;

    public Payment(Long paymentId, Invoice invoice, double amount, LocalDateTime dateTime, PaymentMethod paymentMethod, Status paymentStatus) {
        this.paymentId = paymentId;
        this.invoice = invoice;
        this.amount = amount;
        this.dateTime = dateTime;
        this.paymentMethod = paymentMethod;
        this.paymentStatus = paymentStatus;
    }

    // getters & setters

    // crate payment
    public static Payment createPayment(Invoice invoice, PaymentMethod paymentMethod) {
        long paymentId = Utils.generateId();
        double amount = invoice.getTotalAmount();
        LocalDateTime createdAt = LocalDateTime.now();
        Status paymentStatus = Status.PAID;

        return new Payment(paymentId, invoice, amount, createdAt, paymentMethod, paymentStatus);
    }
}

```

6. Schedule



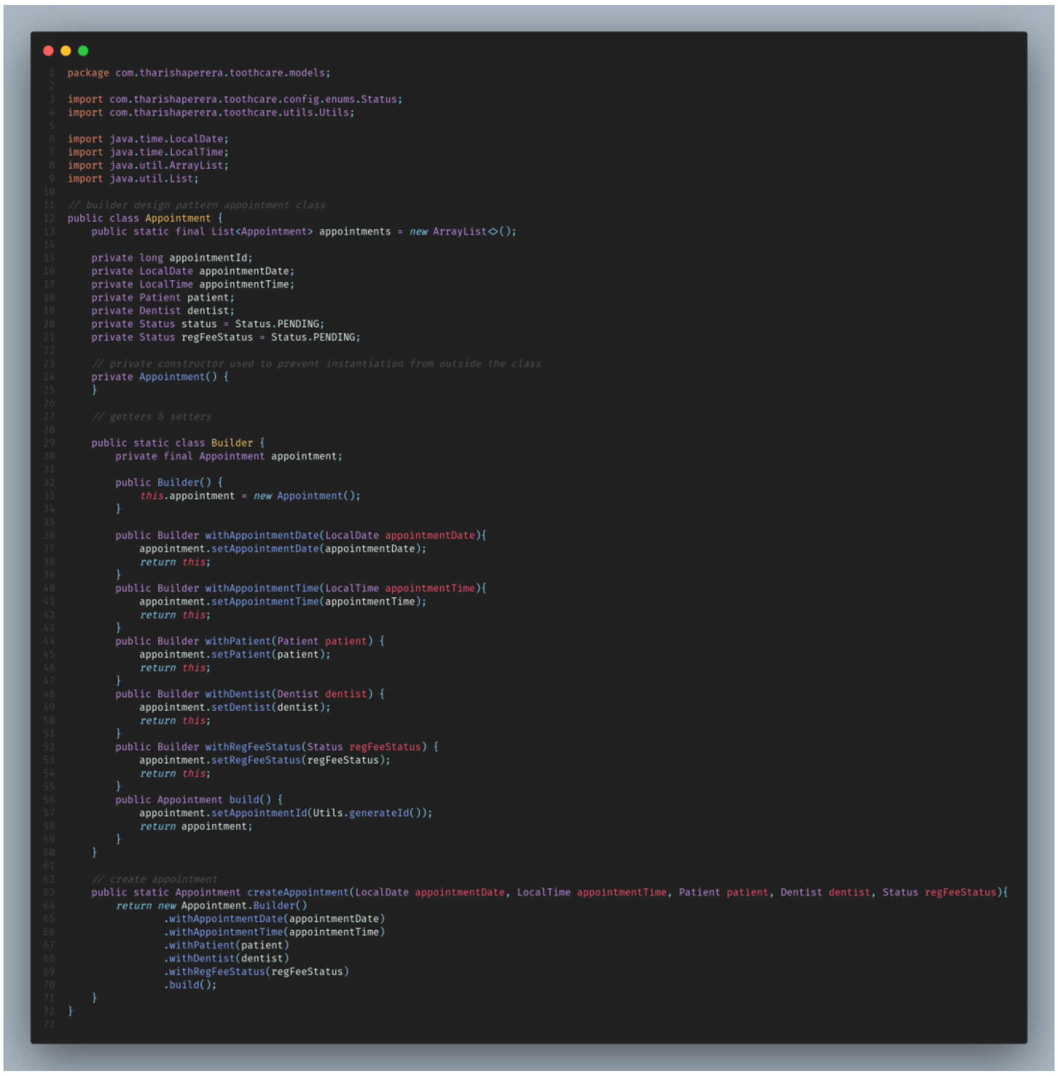
```java
package com.tharishaperera.toothcare.models;

import com.tharishaperera.toothcare.utils.Utils;

import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;

public class Schedule {
    public static final List<Schedule> schedules = new ArrayList<>();

    private long scheduleId;
    private Dentist dentist;
    private String day;
    private LocalTime startTime;
    private LocalTime endTime;

    public Schedule(long scheduleId, Dentist dentist, String day, LocalTime startTime, LocalTime endTime) {
        this.scheduleId = scheduleId;
        this.dentist = dentist;
        this.day = day;
        this.startTime = startTime;
        this.endTime = endTime;
    }

    // getters & setters

    // crate schedule
    public static Schedule createSchedule(Dentist dentist, String day, LocalTime startTime, LocalTime endTime) {
        long scheduleId = Utils.generateId();
        return new Schedule(scheduleId, dentist, day, startTime, endTime);
    }
}
```

7. Invoice



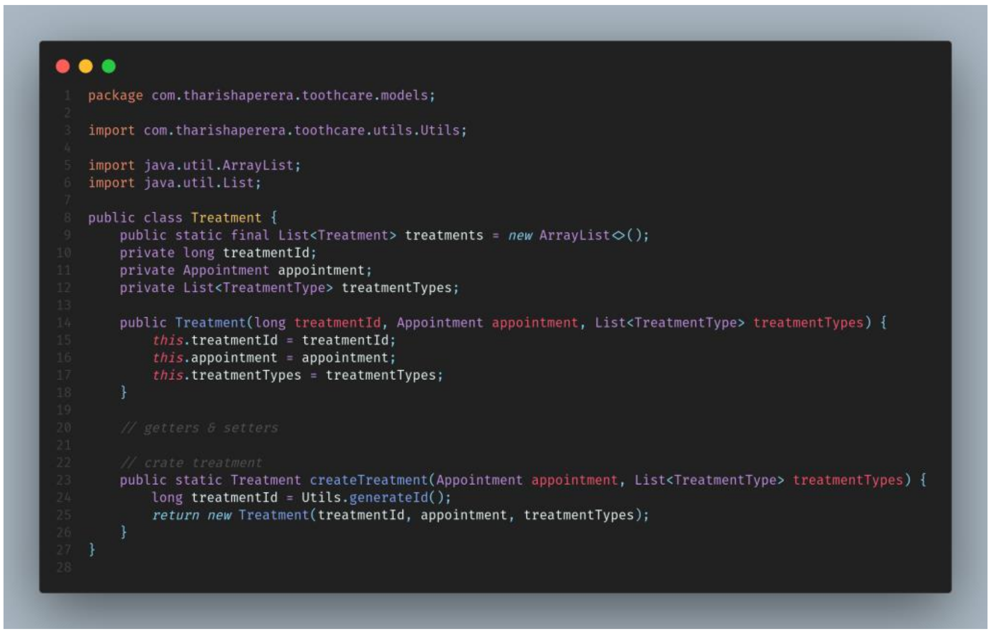
```java
package com.tharishaperera.toothcare.models;

import com.tharishaperera.toothcare.utils.Utils;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

public class Invoice {
    public static final List<Invoice> invoices = new ArrayList<>();
    private Long invoiceId;
    private LocalDate dateIssued;
    private Treatment treatment;
    private Double totalAmount = 0.00;

    public Invoice(Long invoiceId, LocalDate dateIssued, Treatment treatment, Double totalAmount) {
        this.invoiceId = invoiceId;
        this.dateIssued = dateIssued;
        this.treatment = treatment;
        this.totalAmount = totalAmount;
    }

    // getters & setters

    // crate invoice
    public static Invoice createInvoice(Treatment treatment) {
        long invoiceId = Utils.generateId();
        LocalDate dateIssued = LocalDate.now();
        double total = calculateBillAmount(treatment.getTreatmentTypes());
        return new Invoice(invoiceId, dateIssued, treatment, total);
    }

    //calculate the total bill amount
    public static double calculateBillAmount(List<TreatmentType> treatmentTypes) {
        double registrationFee = 1000;
        double total = registrationFee;
        for (TreatmentType treatmentType: treatmentTypes) {
            total += treatmentType.getTreatmentTypePrice();
        }
        return total;
    }
}
```

8. Appointment



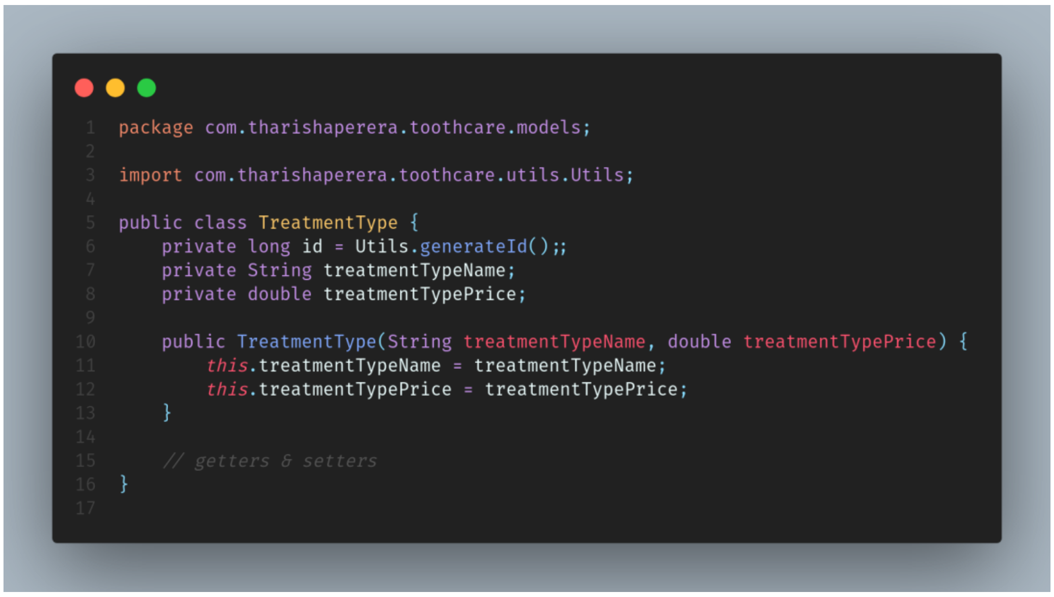
```java
package com.tharishaperera.toothcare.models;

import com.tharishaperera.toothcare.config.enums.Status;
import com.tharishaperera.toothcare.utils.Utils;

import java.time.LocalDate;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;

// builder design pattern appointment class
public class Appointment {
    public static final List<Appointment> appointments = new ArrayList<>();

    private long appointmentId;
    private LocalDate appointmentDate;
    private LocalTime appointmentTime;
    private Patient patient;
    private Dentist dentist;
    private Status status = Status.PENDING;
    private Status regFeeStatus = Status.PENDING;

    // private constructor used to prevent instantiation from outside the class
    private Appointment() {
    }

    // getters & setters

    public static class Builder {
        private final Appointment appointment;

        public Builder() {
            this.appointment = new Appointment();
        }

        public Builder withAppointmentDate(LocalDate appointmentDate){
            appointment.setAppointmentDate(appointmentDate);
            return this;
        }
        public Builder withAppointmentTime(LocalTime appointmentTime){
            appointment.setAppointmentTime(appointmentTime);
            return this;
        }
        public Builder withPatient(Patient patient) {
            appointment.setPatient(patient);
            return this;
        }
        public Builder withDentist(Dentist dentist) {
            appointment.setDentist(dentist);
            return this;
        }
        public Builder withRegFeeStatus(Status regFeeStatus) {
            appointment.setRegFeeStatus(regFeeStatus);
            return this;
        }
        public Appointment build() {
            appointment.setAppointmentId(Utils.generateId());
            return appointment;
        }
    }

    // create appointment
    public static Appointment createAppointment(LocalDate appointmentDate, LocalTime appointmentTime, Patient patient, Dentist dentist, Status regFeeStatus){
        return new Appointment.Builder()
                .withAppointmentDate(appointmentDate)
                .withAppointmentTime(appointmentTime)
                .withPatient(patient)
                .withDentist(dentist)
                .withRegFeeStatus(regFeeStatus)
                .build();
    }
}
```

9. Treatment



```java
package com.tharishaperera.toothcare.models;

import com.tharishaperera.toothcare.utils.Utils;

import java.util.ArrayList;
import java.util.List;

public class Treatment {
    public static final List<Treatment> treatments = new ArrayList<>();
    private long treatmentId;
    private Appointment appointment;
    private List<TreatmentType> treatmentTypes;

    public Treatment(long treatmentId, Appointment appointment, List<TreatmentType> treatmentTypes) {
        this.treatmentId = treatmentId;
        this.appointment = appointment;
        this.treatmentTypes = treatmentTypes;
    }

    // getters & setters

    // crate treatment
    public static Treatment createTreatment(Appointment appointment, List<TreatmentType> treatmentTypes) {
        long treatmentId = Utils.generateId();
        return new Treatment(treatmentId, appointment, treatmentTypes);
    }
}
```

10. TreatmentType

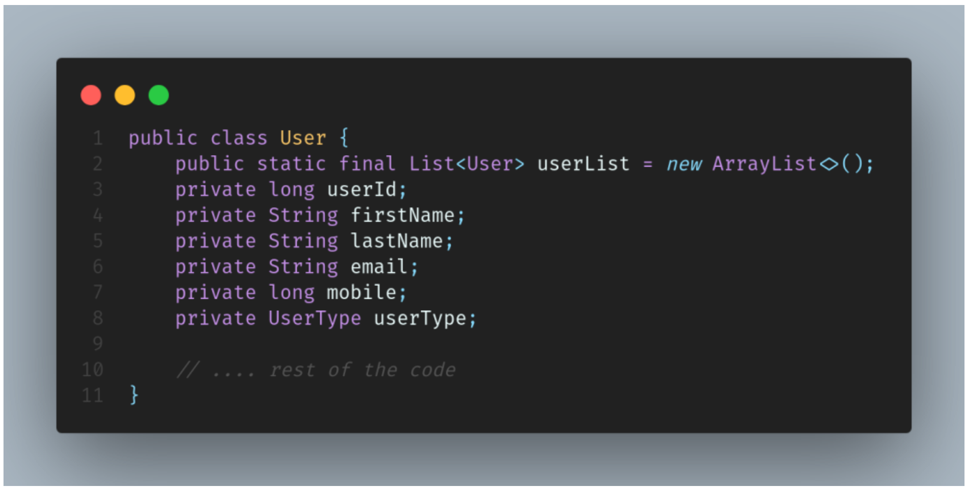


```java
package com.tharishaperera.toothcare.models;

import com.tharishaperera.toothcare.utils.Utils;

public class TreatmentType {
    private long id = Utils.generateId();;
    private String treatmentTypeName;
    private double treatmentTypePrice;

    public TreatmentType(String treatmentTypeName, double treatmentTypePrice) {
        this.treatmentTypeName = treatmentTypeName;
        this.treatmentTypePrice = treatmentTypePrice;
    }

    // getters & setters
}
```

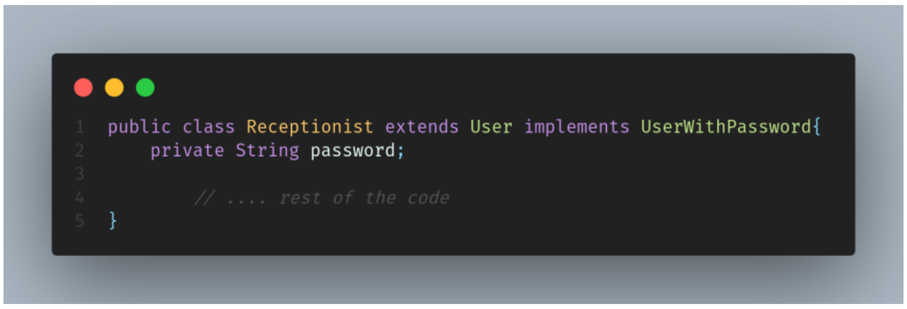
3.3. Object Oriented Programming Concepts

There are few Object-Oriented Programming concepts that were used to provide a maintainable program during the development of this ToothCare application.

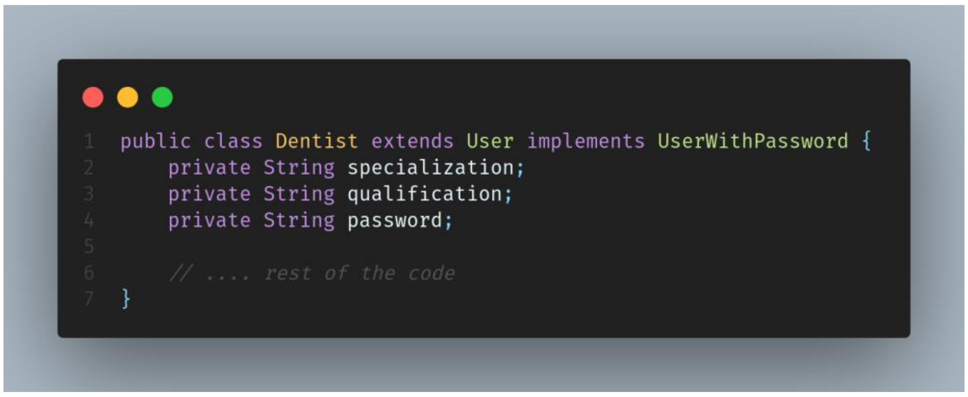
1. Inheritance
   – Super class User is extended by the Patient, Dentist, Receptionist classes and inherited attributes and methods from User class to their own classes



```java
public class User {
    public static final List<User> userList = new ArrayList<>();
    private long userId;
    private String firstName;
    private String lastName;
    private String email;
    private long mobile;
    private UserType userType;

    // .... rest of the code
}
```

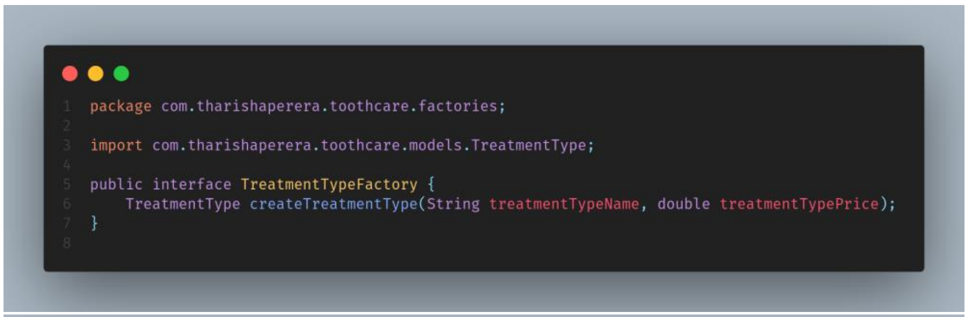


```java
public class Receptionist extends User implements UserWithPassword{
    private String password;

        // .... rest of the code
}
```

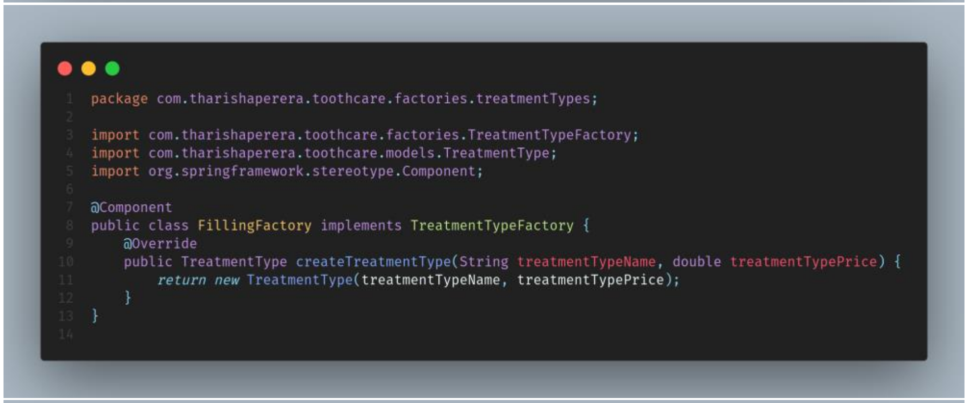


```java
public class Dentist extends User implements UserWithPassword {
    private String specialization;
    private String qualification;
    private String password;

    // .... rest of the code
}
```

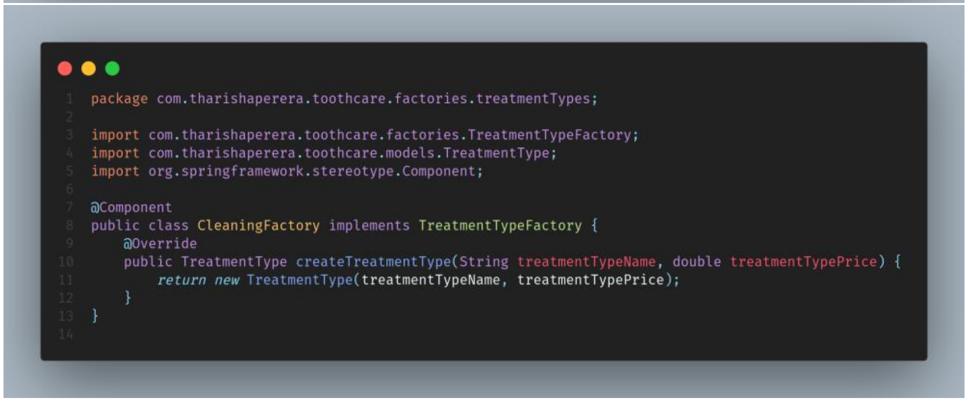
2. Polymorphism
   – Here the “createTreatmentType” method in “TreatmentTypeFactory” is overridden by the interface-based classes by showcasing the Method Overriding polymorphism form.



```java
package com.tharishaperera.toothcare.factories;

import com.tharishaperera.toothcare.models.TreatmentType;

public interface TreatmentTypeFactory {
    TreatmentType createTreatmentType(String treatmentTypeName, double treatmentTypePrice);
}
```



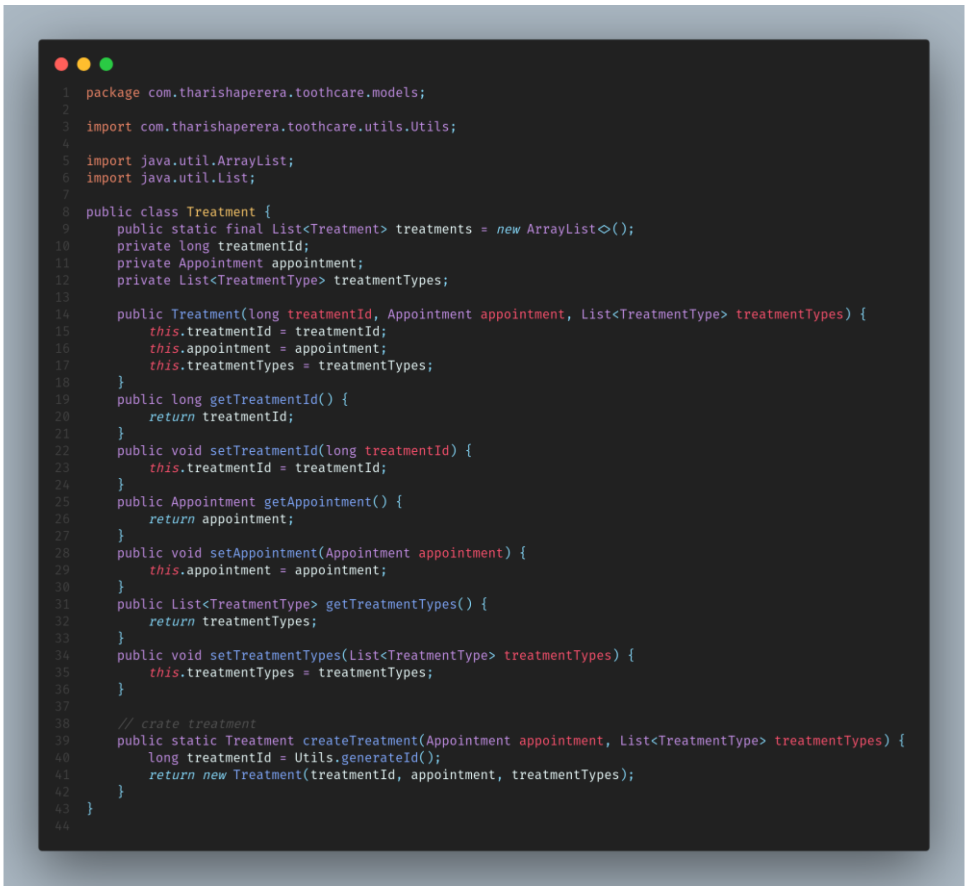
```java
package com.tharishaperera.toothcare.factories.treatmentTypes;

import com.tharishaperera.toothcare.factories.TreatmentTypeFactory;
import com.tharishaperera.toothcare.models.TreatmentType;
import org.springframework.stereotype.Component;

@Component
public class FillingFactory implements TreatmentTypeFactory {
    @Override
    public TreatmentType createTreatmentType(String treatmentTypeName, double treatmentTypePrice) {
        return new TreatmentType(treatmentTypeName, treatmentTypePrice);
    }
}
```



```java
package com.tharishaperera.toothcare.factories.treatmentTypes;

import com.tharishaperera.toothcare.factories.TreatmentTypeFactory;
import com.tharishaperera.toothcare.models.TreatmentType;
import org.springframework.stereotype.Component;

@Component
public class CleaningFactory implements TreatmentTypeFactory {
    @Override
    public TreatmentType createTreatmentType(String treatmentTypeName, double treatmentTypePrice) {
        return new TreatmentType(treatmentTypeName, treatmentTypePrice);
    }
}
```

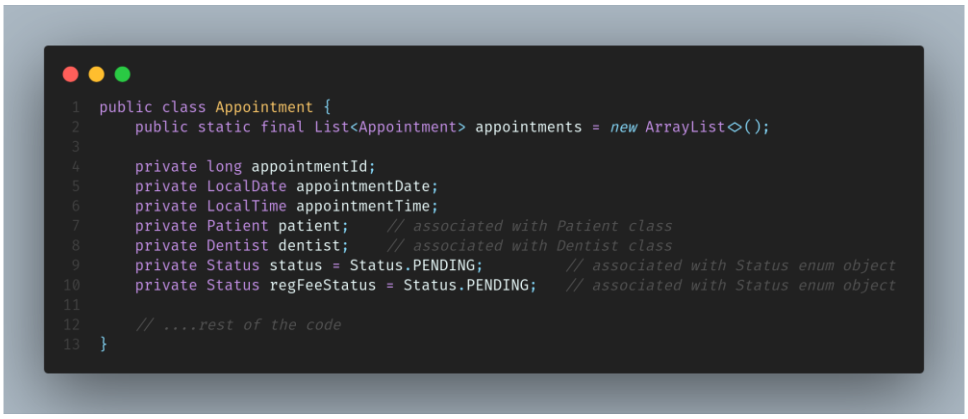
3. Encapsulation

- Here, all the attributes are made private to restrict the direct access and implemented methods to operate those attributes. By following Encapsulation concept, the internal class data is hidden, and it prevents accidental data modifications.



```java
package com.tharishaperera.toothcare.models;

import com.tharishaperera.toothcare.utils.Utils;

import java.util.ArrayList;
import java.util.List;

public class Treatment {
    public static final List<Treatment> treatments = new ArrayList<>();
    private long treatmentId;
    private Appointment appointment;
    private List<TreatmentType> treatmentTypes;

    public Treatment(long treatmentId, Appointment appointment, List<TreatmentType> treatmentTypes) {
        this.treatmentId = treatmentId;
        this.appointment = appointment;
        this.treatmentTypes = treatmentTypes;
    }
    public long getTreatmentId() {
        return treatmentId;
    }
    public void setTreatmentId(long treatmentId) {
        this.treatmentId = treatmentId;
    }
    public Appointment getAppointment() {
        return appointment;
    }
    public void setAppointment(Appointment appointment) {
        this.appointment = appointment;
    }
    public List<TreatmentType> getTreatmentTypes() {
        return treatmentTypes;
    }
    public void setTreatmentTypes(List<TreatmentType> treatmentTypes) {
        this.treatmentTypes = treatmentTypes;
    }

    // crate treatment
    public static Treatment createTreatment(Appointment appointment, List<TreatmentType> treatmentTypes) {
        long treatmentId = Utils.generateId();
        return new Treatment(treatmentId, appointment, treatmentTypes);
    }
}
```

4. Association
   - In the ToothCare application, the Association OOP concept is used to create relationships between two or more classes or objects.
   - In the following code snippet, the Appointment class has associations with Patient and Dentist classes as well as the Status enum object.

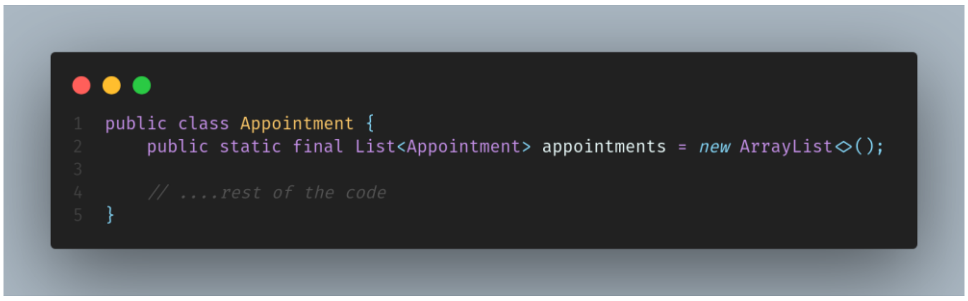


```java
public class Appointment {
    public static final List<Appointment> appointments = new ArrayList<>();

    private long appointmentId;
    private LocalDate appointmentDate;
    private LocalTime appointmentTime;
    private Patient patient;    // associated with Patient class
    private Dentist dentist;    // associated with Dentist class
    private Status status = Status.PENDING;          // associated with Status enum object
    private Status regFeeStatus = Status.PENDING;    // associated with Status enum object

    // ....rest of the code
}
```

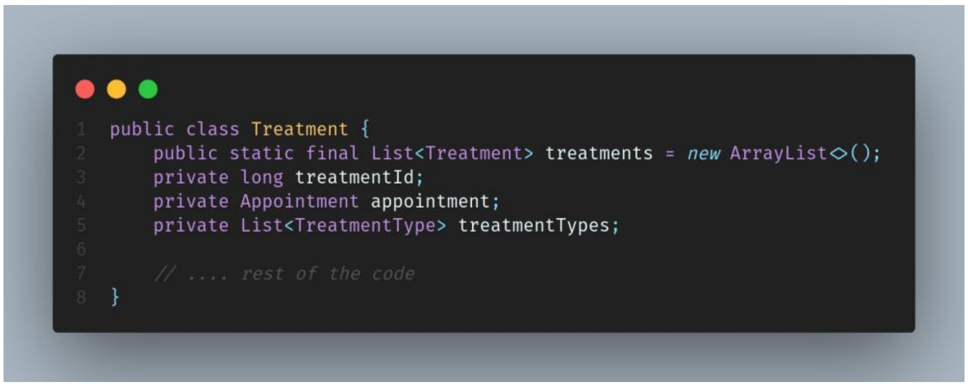
3.4. Data Structures

The ToothCare application uses ArrayList as the data structure to store the data throughout the application due to a few reasons.

- ArrayLists automatically adjust their size dynamically, providing flexibility during the run time. It is convenient since the number of elements that will be stored is not known in this application scenario.
- In addition, ArrayList is one of the easy-to-use data structures providing the easy ability to add, remove and access data randomly.



```java
public class Appointment {
    public static final List<Appointment> appointments = new ArrayList<>();

    // ....rest of the code
}
```

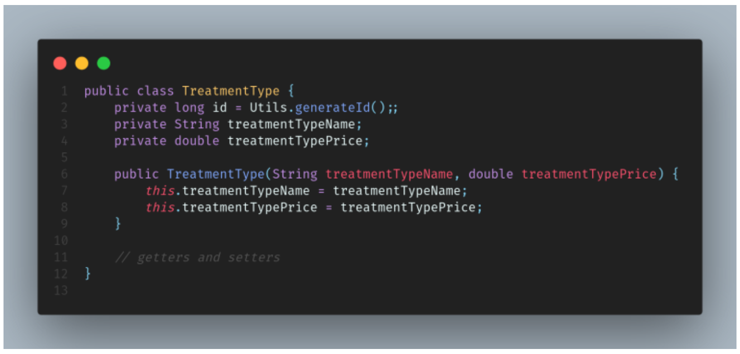


```java
public class Treatment {
    public static final List<Treatment> treatments = new ArrayList<>();
    private long treatmentId;
    private Appointment appointment;
    private List<TreatmentType> treatmentTypes;

    // .... rest of the code
}
```

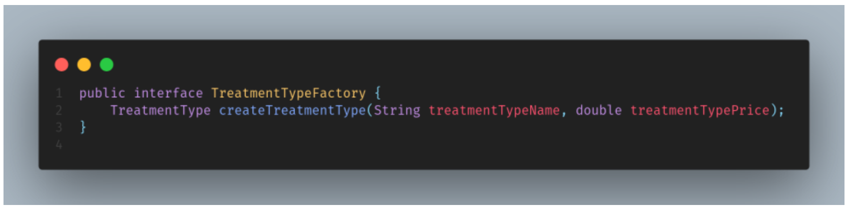
3.5. Relevant Algorithms and Patterns

The ToothCare application has used two design patterns to implement its classes and their objects, which are the Factory Design Pattern and the Builder Design Pattern. Addition to that and since the springboot is used for the backend, the singleton pattern is being used by default in the application.

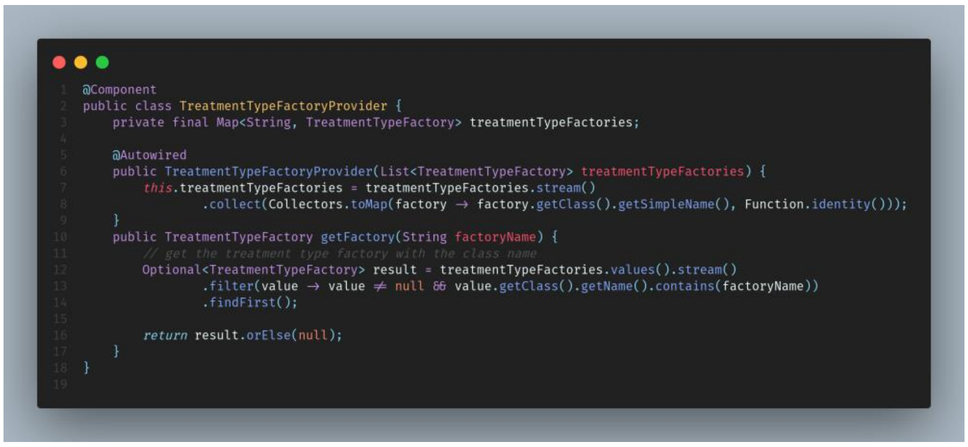
The reason to use factory pattern for the TreatmentType class and for its objects was, the factory design pattern provides a centralized place to create objects, which makes it easier to understand and maintain the process.



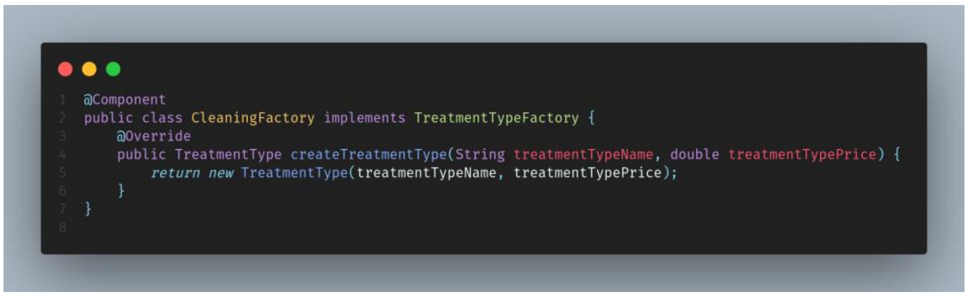
```java
public class TreatmentType {
    private long id = Utils.generateId();;
    private String treatmentTypeName;
    private double treatmentTypePrice;

    public TreatmentType(String treatmentTypeName, double treatmentTypePrice) {
        this.treatmentTypeName = treatmentTypeName;
        this.treatmentTypePrice = treatmentTypePrice;
    }

    // getters and setters
}
```



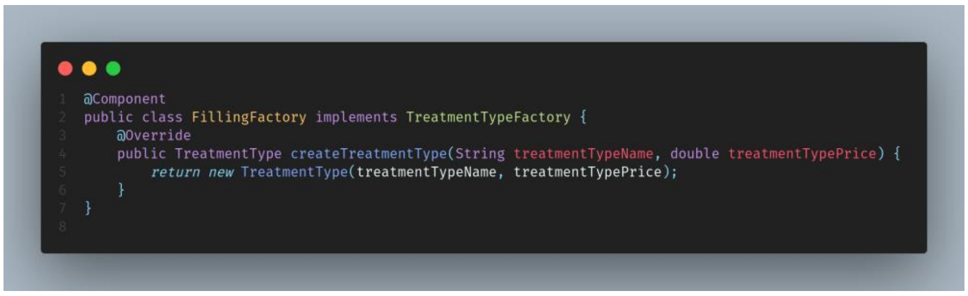
```java
public interface TreatmentTypeFactory {
    TreatmentType createTreatmentType(String treatmentTypeName, double treatmentTypePrice);
}
```



```java
@Component
public class TreatmentTypeFactoryProvider {
    private final Map<String, TreatmentTypeFactory> treatmentTypeFactories;

    @Autowired
    public TreatmentTypeFactoryProvider(List<TreatmentTypeFactory> treatmentTypeFactories) {
        this.treatmentTypeFactories = treatmentTypeFactories.stream()
                .collect(Collectors.toMap(factory -> factory.getClass().getSimpleName(), Function.identity()));
    }
    public TreatmentTypeFactory getFactory(String factoryName) {
        // get the treatment type factory with the class name
        Optional<TreatmentTypeFactory> result = treatmentTypeFactories.values().stream()
                .filter(value -> value != null && value.getClass().getName().contains(factoryName))
                .findFirst();

        return result.orElse(null);
    }
}
```

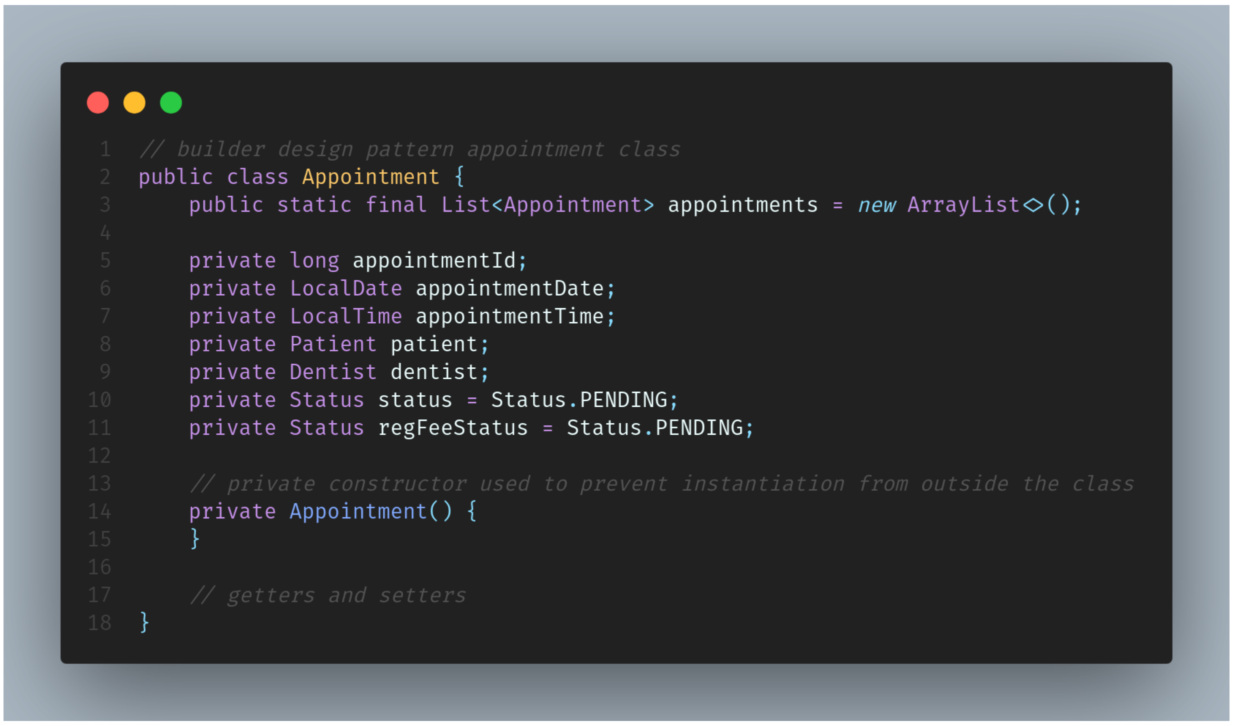


```java
@Component
public class CleaningFactory implements TreatmentTypeFactory {
    @Override
    public TreatmentType createTreatmentType(String treatmentTypeName, double treatmentTypePrice) {
        return new TreatmentType(treatmentTypeName, treatmentTypePrice);
    }
}
```

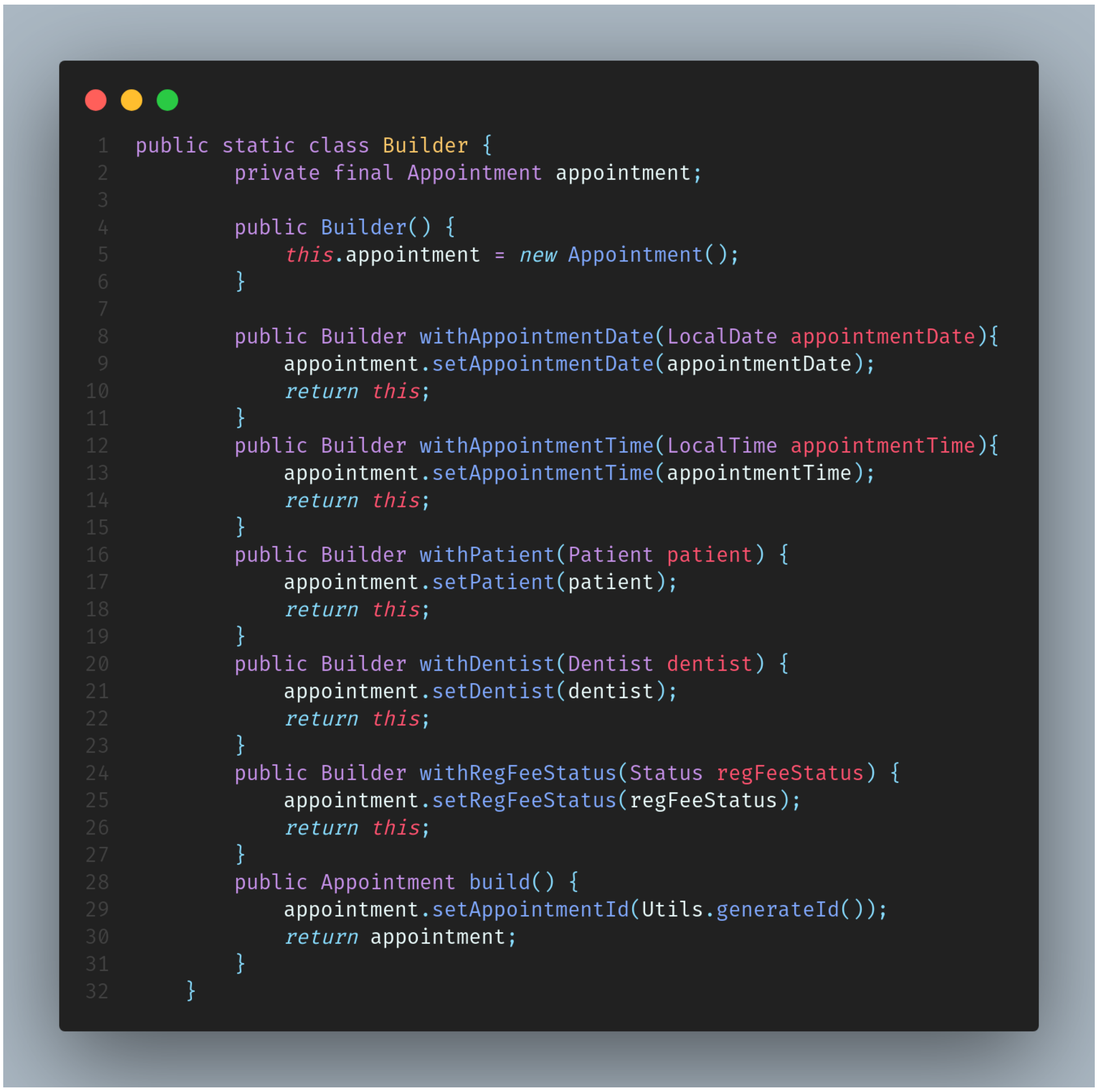


```java
@Component
public class FillingFactory implements TreatmentTypeFactory {
    @Override
    public TreatmentType createTreatmentType(String treatmentTypeName, double treatmentTypePrice) {
        return new TreatmentType(treatmentTypeName, treatmentTypePrice);
    }
}
```

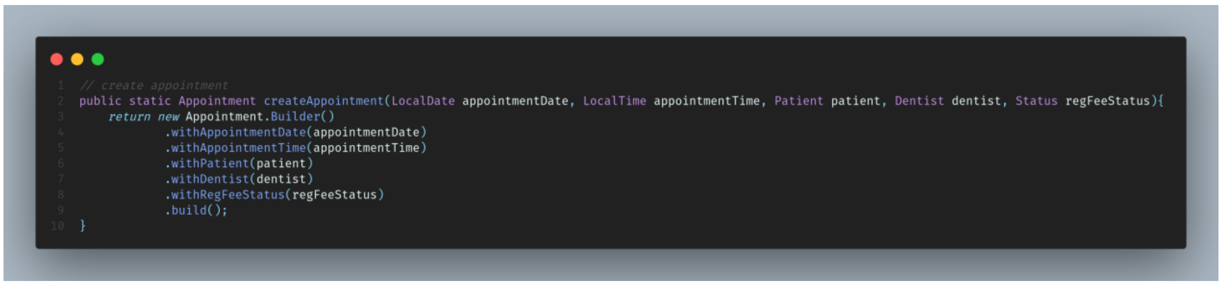
The reason to use Builder Pattern for the Appointment class to create its objects was, the builder pattern provides a readable and a flexible way to create objects with a large number of attributes. Also, the builder pattern is widely used to create complex objects step by step.



```java
// builder design pattern appointment class
public class Appointment {
    public static final List<Appointment> appointments = new ArrayList<>();

    private long appointmentId;
    private LocalDate appointmentDate;
    private LocalTime appointmentTime;
    private Patient patient;
    private Dentist dentist;
    private Status status = Status.PENDING;
    private Status regFeeStatus = Status.PENDING;

    // private constructor used to prevent instantiation from outside the class
    private Appointment() {
    }

    // getters and setters
}
```



```java
public static class Builder {
        private final Appointment appointment;

        public Builder() {
            this.appointment = new Appointment();
        }

        public Builder withAppointmentDate(LocalDate appointmentDate){
            appointment.setAppointmentDate(appointmentDate);
            return this;
        }
        public Builder withAppointmentTime(LocalTime appointmentTime){
            appointment.setAppointmentTime(appointmentTime);
            return this;
        }
        public Builder withPatient(Patient patient) {
            appointment.setPatient(patient);
            return this;
        }
        public Builder withDentist(Dentist dentist) {
            appointment.setDentist(dentist);
            return this;
        }
        public Builder withRegFeeStatus(Status regFeeStatus) {
            appointment.setRegFeeStatus(regFeeStatus);
            return this;
        }
        public Appointment build() {
            appointment.setAppointmentId(Utils.generateId());
            return appointment;
        }
    }
```



```java
// create appointment
public static Appointment createAppointment(LocalDate appointmentDate, LocalTime appointmentTime, Patient patient, Dentist dentist, Status regFeeStatus){
    return new Appointment.Builder()
            .withAppointmentDate(appointmentDate)
            .withAppointmentTime(appointmentTime)
            .withPatient(patient)
            .withDentist(dentist)
            .withRegFeeStatus(regFeeStatus)
            .build();
}
```

4. Test Cases

Test Case #	TC_01	Test case name	Log into the system
System	Toothcare	Subsystem	Login
Designed by	Tharisha Perera	Designed date	20/12/2023
Executed by	Tharisha Perera	Execution date	24/12/2023
Short description	Test case to check the functionality of logging into the system		

Pre-conditions: User should create the initial data by clicking on the Get Started button on the landing page.

Step	Action	Expected System Response	Pass / Fail	Comment
1	Navigate to the login page by clicking on the get started button or use the"/login" URL path	User should be redirected to the login page with login form	Pass	-
2	Click on the login button without providing the email and password	Required field validation messages should appear for both input fields	Pass	-
3	Enter an invalid email and click on the submit button	System should give an invalid email validation message for the email field and required validation message for the password field.	Pass	-
4	Enter correct credentials to the form and submit	The user should be redirected to the main menu page	Pass	-

Post-conditions: None

Test Case #	TC_02	Test case name	Create appointment
System	Toothcare	Subsystem	Appointments
Designed by	Tharisha Perera	Designed date	20/12/2023
Executed by	Tharisha Perera	Execution date	24/12/2023
Short description	Test case to check, the appointment creation is working well or not.		

Pre-conditions: User should be able to log into the system.

Step	Action	Expected System Response	Pass / Fail	Comment
1	Click on the Appointment link on the main menu.	User should be redirected to the Appointments page	Pass	-
2	Click on the Create button on the top right corner	User should be redirected to the Appointment creation form page	Pass	-
3	Select appointment date	User should not be able to select past dates	Pass	Past dates are disabled in the date picker element.
4	Select appointment time	Available Appointment times should be visible according to the selected date.	Fail	Time select element displays all the available appointment times.
5	Fill in all the required data.	The system should be able to provide validation messages based on the user inputs	Pass	-
6	Click on the submit button after filling in all the required data	User should get a success message and be redirected to all Appointments page	Pass	-

Post-conditions: None

Test Case #	TC_03	Test case name	Update appointment
System	Toothcare	Subsystem	Appointments
Designed by	Tharisha Perera	Designed date	20/12/2023
Executed by	Tharisha Perera	Execution date	24/12/2023
Short description	Test case to check, the appointment update functionality is working well or not.		

Pre-conditions: User should be able to log into the system and access all appointments page.

Step	Action	Expected System Response	Pass / Fail	Comment
1	Click on the Appointment link on the main menu.	User should be redirected to the Appointments page	Pass	-
2	Click on the Edit icon button on the appointment table right corner	User should be redirected to the Appointment update form page	Pass	-
3	Click on the Edit icon button on the appointment table right corner	The update form should be populated with the selected appointment data	Fail	The form is not populated with selected appointment data.

Post-conditions: None

Test Case #	TC_04	Test case name	Accept appointment registration fee
System	Toothcare	Subsystem	Appointments
Designed by	Tharisha Perera	Designed date	20/12/2023
Executed by	Tharisha Perera	Execution date	24/12/2023
Short description	Test case to check the accept appointment registration fee after creating the appointment.		

Pre-conditions: User should be able to log into the system and access all appointments page.

Step	Action	Expected System Response	Pass / Fail	Comment
1	Click on the Appointment link on the main menu.	User should be redirected to the Appointments page	Pass	-
2	Click on the toggle icon button which has the "*Registration Fee...*" tooltip on the appointment table right corner	User should get a message confirming the update appointment process status and change the toggle icon and the tooltip	Pass	-

Post-conditions: None				

Test Case #	TC_05	Test case name	Complete appointment
System	Toothcare	Subsystem	Appointments
Designed by	Tharisha Perera	Designed date	20/12/2023
Executed by	Tharisha Perera	Execution date	24/12/2023
Short description	Test case to check the complete appointment functionality after patient receives the treatments.		
Pre-conditions: User should be able to log into the system and access all appointments page.			

Step	Action	Expected System Response	Pass / Fail	Comment
1	Click on the Appointment link on the main menu.	User should be redirected to the Appointments page	Pass	-
2	Click on the toggle icon button which has the "*Appointment Status...*" tooltip on the appointment table right corner	User should get a popup window asking to select the treatments that the patient has received.	Pass	-
3	Select one or more treatments that the patient received.	User should be able to select single or multiple treatments,	Pass	-
4	Click on the complete button at bottom to complete the appointment after selecting the treatments.	User should be redirected to the Invoice page	Pass	-
5	Click on the complete button at bottom to complete the appointment after selecting the treatments.	User should be able to view the details of the appointment and patient, treatments received and the amount to pay.	Pass	-
Post-conditions: User should be able to receive the payment and update the invoice.				

Test Case #	TC_06	Test case name	Generate the invoice
System	Toothcare	Subsystem	Invoices
Designed by	Tharisha Perera	Designed date	20/12/2023
Executed by	Tharisha Perera	Execution date	24/12/2023
Short description	Test case to check the functionality of generating an invoice for the patient.		

Pre-conditions: User should be able to log into the system and access the invoice details page.

Step	Action	Expected System Response	Pass / Fail	Comment
1	Click on the Invoices link on the main menu.	User should be redirected to all Invoices page	Pass	-
2	Click on the eye icon button on the invoice table in the right corner to view the invoice.	User should be redirected to the invoice details page	Pass	-
3	Click on the print button on the top right corner	The system should give the browsers popup window to print the invoice and print or save the invoice.	Pass	-

Post-conditions: None

Test Case #	TC_07	Test case name	Receive the total payment
System	Toothcare	Subsystem	Invoices
Designed by	Tharisha Perera	Designed date	20/12/2023
Executed by	Tharisha Perera	Execution date	24/12/2023
Short description	Test case to check the functionality of accepting payment and close the invoice.		

Pre-conditions: User should be able to log into the system and access the invoice details page.

Step	Action	Expected System Response	Pass / Fail	Comment
1	Click on the Invoices link on the main menu.	User should be redirected to all Invoices page	Pass	-
2	Click on the eye icon button on the invoice table in the right corner to view the invoice.	User should be redirected to the invoice details page	Pass	-
3	Click on the receive payment button on the bottom right corner	The system should give a popup window to select the payment method	Pass	-

4	Click on the receive payment button on the bottom right corner	User should be able to select only one payment method to receive payment	Pass	-
5	Click on the continue button to close the invoice	Users should be redirected to the Payments page with updated details of the completed invoice.	Pass	-
Post-conditions: None				

Test Case #	TC_08	Test case name	View & Delete Patients
System	Toothcare	Subsystem	Patients
Designed by	Tharisha Perera	Designed date	20/12/2023
Executed by	Tharisha Perera	Execution date	24/12/2023
Short description	Test case to check the functionality of viewing and deleting patients created		
Pre-conditions: User should be able to log into the system and access the patient's page.			

Step	Action	Expected System Response	Pass / Fail	Comment
1	Click on the Patients link on the main menu.	User should be redirected to all Patients page	Pass	-
2	Click on the Patients link on the main menu.	User should be able to view all the patients registered in the system.	Pass	-
3	Click on the Trash can icon button on the patients' tables to delete the user	The system should delete the user and refresh the patient table to update the data.	Pass	-
Post-conditions: None				

Test Case #	TC_09	Test case name	View & Delete Dentists
System	Toothcare	Subsystem	Dentists
Designed by	Tharisha Perera	Designed date	20/12/2023
Executed by	Tharisha Perera	Execution date	24/12/2023
Short description	Test case to check the functionality of viewing and deleting dentists created		
Pre-conditions: User should be able to log into the system and access the dentist's page.			

Step	Action	Expected System Response	Pass / Fail	Comment

1	Click on the Dentists link on the main menu.	User should be redirected to all Dentists page	Pass	-
2	Click on the Dentists link on the main menu.	User should be able to view all the dentists registered in the system.	Pass	-
3	Click on the Trash can icon button on the dentists' tables to delete the user	The system should delete the user and refresh the dentist's table to update the data.	Pass	-
Post-conditions: None				

Test Case #	TC_10	Test case name	Filter Appointments
System	Toothcare	Subsystem	Appointment
Designed by	Tharisha Perera	Designed date	26/12/2023
Executed by	Tharisha Perera	Execution date	26/12/2023
Short description	Test case to check the functionality of Filtering Appointments using Appointment date.		

Pre-conditions: User should be able to log into the system and access the Appointments page.

Step	Action	Expected System Response	Pass / Fail	Comment
1	Click on the Appointments link on the main menu.	User should be redirected to all Appointments page	Pass	-
2	Click on the Filter button in the top right corner.	User should be able to view a dropdown menu with all filtering options.	Pass	-
3	Click on the Filter by date link	User should be redirected to the Filter Appointments by date page	Pass	-
4	Select the date to filter appointments	User should be able to view all the appointment with has the selected date on the Appointment table	Pass	-

Post-conditions: None

Appendix

- GitHub Repository
  - https://github.com/TharishaPerera/cw-programming-3.git
  - This link will take you to the GitHub repository containing the source code, documentation and Postman API collection files.
- Google Drive Folder

- https://drive.google.com/drive/folders/1U9XHpgjYrWke2lNMOOwO-kuRYyHhqz9b?usp=sharing
- Access the zipped version of the source code and the other project files, and the project report.