Islamic University of Technology
Board Bazar, Gazipur, Dhaka

BookATurf Console App Report

SWE 4302- Object Oriented Concepts II Lab

SOLID Application Based JAVA Project

Kashshaf Labib
ID: 210042151
BSc. in Software Engineering
Department of Computer Science and Engineering

January 20, 2024

Contents

Chapter 1

Overview of the Console App 'BookATurf'

1.1 Growing popularity of Turfs across Bangladesh

In the contemporary era, the popularity of turfs for sports and recreational activities has been on the rise across the country. To cater to the increasing demand and streamline the booking process, the "BookATurf" Management System has been developed. This project provides a comprehensive solution for managing turf bookings, ensuring a smooth and efficient experience for customers. However, for the sake of simplicity, the project is designed in the console to demonstrate the basic functionalities of a Turf booking app which can help designing an actual web app in future.

1.2 Scope of the Project

The scope of the "BookATurf Management System" revolves around facilitating the booking and management of different types of turfs, such as Artificial Grass Turfs, Natural Grass Turfs, and Cricket Turfs. The system allows users to register, log in, explore available turfs, book slots, and view booking details. Administrators have the capability to add new turfs, manage slot pricing, and monitor overall turf availability.

1.3 Key Functionalities

1. User Registration and Login:
    - Users can register with a unique username and an email and password.
    - Existing users can log in securely using their credentials.
2. Turf Exploration:
    - Users and administrators can explore details about available turfs.
    - Turf details include ID, name, location, player capacity, and specific details based on turf type.
3. Slot Booking:
    - Users can book available slots on their desired turfs.
    - The system checks and ensures slot availability during the booking process.
4. Slot Availability Checking:
    - Users can check the availability of turf slots providing their desired time

5. Exception Handling:

    - The system incorporates exception handling for scenarios like user validation, duration constraints, and payment issues.

The "BookATurf Management System" addresses the growing demand for turf facilities, providing an organized and efficient idea for the users. By adhering to SOLID principles, the project achieves a modular and extensible design, offering a robust solution to meet the evolving needs of turf enthusiasts across the country.

Chapter 2

SOLID Principles and their implementations on the project

The SOLID Principles are five principles of Object-Oriented class design. They are a set of rules and best practices to follow while designing a class structure.These five principles help us understand the need for certain design patterns and software architecture in general. Most of the developers spend 40% or 50% of their development time writing codes. The rest of the time, they spend reading the code and maintaining the system. Therefore, it is important to create a good system design. A good system needs a good code base that is easy to read, understand, maintain (add/change features, fix bugs), and extend in the future. This saves development time and resources while increasing work satisfaction.

2.1 Single Responsibility Principle (SRP) and its application on the project:

The Single Responsibility Principle states that a class should do one thing and therefore it should have only a single reason to change. To put this idea more precisely, the specification of the class should only be impacted by a single possible change in the software's specification.

SRP is closely related to the concepts of Coupling (low) and Cohesion (high). SRP does not necessarily mean that a class should only have one method or property, but rather that the functionality should be related to a single responsibility (and have only one reason for changing).

SRP in BookATurf:

- SlotAvailabilityService: This class solely handles the availability of turf slots for the users.

```
public class SlotAvailabilityService {

    public static int isSlotAvailable(IAvailability turf, LocalTime
        startingTime, LocalTime endingTime)
    {
        return turf.isSlotAvailable(startingTime, endingTime);
    }
}
```

- BookingService: This class has method that corresponds to only booking of a turf-slot for a user.

```
public class BookingService {

    public static void bookSlot(IBook turf,int slotID,int amount,int userID)
```

```
    {
        Book bookedSlot=turf.bookSlot(slotID,amount);
        if(bookedSlot!=null)
        {
            User user=Utility.getUserbyID(userID);
            if(user!=null)
            {
                user.bookingList.add(bookedSlot);
                System.out.println("Slot booked successfully");
            }
            else
            {
                System.out.println("User not found, booking unsuccessful");
            }
        }
        else
        {
            System.out.println("Booking Unsuccessful");
        }
    }


}
```

- SlotPricingService: This class has method that sets the price of a turfslot according to the duration of a booking.

```
public class SlotPricingService {

    public static void setSlotPrice(TurfSlot slot) throws DurationException {

        Duration duration=Duration.between(slot.startingTime,slot.endingTime);

        if(duration.toHours()<1)
        {
            throw new DurationException("Duration of a slot has to be at
                least 1 hour");
        }
        else if(duration.toHours()<2)
        {
            slot.setSlotPrice(1000);
        }
        else
        {
            slot.setSlotPrice(1500);
        }
    }
}
```

- SlotDetailService: This class is responsible to show the details to the users who want to book turfslots.

```
public class TurfDetailService {

    public void getTurfDetails(int turfID,IDetails turf)
    {
        turf.getDetails(turfID);
    }

}
```

- RandomNumberGenerator: This class provides random numbers to generate userID for the customers who register to the system.

```
public class RandomNumberGenerator {

    public static int generateRandomNumber(int min,int max)
    {
        return (int)(Math.random()*(max-min+1)+min);
    }
}
```

- UserRegistrationService: Dedicated class for the user registration service which ensures the uniqueness of the username and the userID.

```
public class UserRegistrationService {

    public static void registerUser(String userName,String userEmail,String
        password) throws UserValidationException
    {
        if(!UserValidator.checkDuplicateUserName(userName))
        {
            int
                userID=RandomNumberGenerator.generateRandomNumber(10000,99999);
            while(UserValidator.checkDuplicateUserID(userID))
            {
                userID=RandomNumberGenerator.generateRandomNumber(10000,99999);
            }
            User user=new User(userName,userEmail);
            user.userID=userID;
            user.setPassword(password);
            Utility.userList.add(user);
            System.out.println("User Registered Successfully");
            System.out.println("User ID: "+user.userID);
            System.out.println("User Name: "+user.userName);
            System.out.println("User Email: "+user.userEmail);

        }
        else
        {
            throw new UserValidationException("Username already exists,
                please try again with a different username");
        }

    }

}
```

- UserLoginService: This class solely handles the login of a user by checking the existence of the user in the system.

```
public class UserLoginService {

    public static boolean userLogin(int userID,String password){

        for(User user:Utility.userList)
        {
            if(user.userID==userID && user.getPassword().equals(password))
            {
                return true;
            }
        }

        return false;
```

```
14
15      }
16  }
```

- UserValidator: BookATurf ensures unique username and userID for the users who register to the system and this class handles the requirements for the task.

```
1  public class UserValidator {
2
3      public static boolean checkDuplicateUserName(String username)
4      {
5          if(Utility.userList.isEmpty())
6          {
7              return false;
8          }
9          return Utility.userList.stream()
10                 .anyMatch(user -> user.userName.equals(username));
11     }
12
13     public static boolean checkDuplicateUserID(int userID)
14     {
15         if(Utility.userList.isEmpty())
16         {
17             return false;
18         }
19         return Utility.userList.stream()
20                 .anyMatch(user -> user.userID==userID);
21     }
22 }
```

2.2 Open-Closed Principle (OCP) and its application on the project:

The Open-Closed Principle requires that classes should be open for extension and closed to modification. Modification means changing the code of an existing class, and extension means adding new functionality.

The gist of this idea is that we need to be able to extend the functionality of a class without modifying its current code. This is due to the fact that every time we alter the current code, we run the danger of perhaps introducing bugs. Thus if at all possible, we should refrain from modifying the production code that has been already tested. We can achieve this through polymorphism.

OCP in BookATurf: In the project, the Turf class is open for extension (by creating subclasses like ArtificialGrassTurf, NaturalGrassTurf, and CricketTurf), but closed for modification.

2.3 Liskov Substitution Principle (LSP) and its application on the project:

This principle states that subtypes must be substitutable by its parent type. That is, If for each object o1 of type S there is an object o2 of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when o1 is substituted for o2 then

S is a subtype of T.

LSP in BookATurf: The subclasses seem to inherit from the Turf class without violating the behavior expected from the base class. For example:

Base Class Turf:

```
public abstract class Turf implements IAvailability, IDetails, IBook{

    public int turfID;

    public String turfName;

    public String turfLocation;

    public int playerCapacity;

    public List<TurfSlot> turfSlotList;


}
```

Child class ArtificialGrassTurf:

```
public class ArtificialGrassTurf extends Turf implements
    IDetails,IAvailability,IBook
{
    public ArtificialGrassTurf(int turfID, String turfName, String turfLocation,
        int playerCapacity)
    {
        this.turfID=turfID;
        this.turfName=turfName;
        this.turfLocation=turfLocation;
        this.playerCapacity=playerCapacity;
        turfSlotList=new ArrayList<>();
    }
    @Override
    public void getDetails(int turfID) {

        System.out.println("Turf ID: "+turfID);
        System.out.println("Turf Name: "+turfName);
        System.out.println("Turf Location: "+turfLocation);
        System.out.println("Player Capacity: "+playerCapacity);
        System.out.println("Turf Type: Artificial Grass Turf");
        System.out.println("Turf Slot Details:");
        if(!turfSlotList.isEmpty())
        {
            for(TurfSlot turfSlot:turfSlotList) {
                System.out.println(turfSlot.getSlotDetails());
            }
        }
        else
        {
            System.out.println("No slots available currently");
        }

    }

    @Override
    public int isSlotAvailable(LocalTime startingTime, LocalTime endingTime) {
        for(TurfSlot turfSlot:turfSlotList)
        {
            if(turfSlot.startingTime.equals(startingTime) &&
```

```
                turfSlot.endingTime.equals(endingTime))
            {
                return turfID;
            }
        }
        return -1;
    }

    @Override
    public Book bookSlot(int slotID,int amount)
    {
        Book book=null;
        for(TurfSlot turfSlot:turfSlotList)
        {
            if(turfSlot.slotID==slotID)
            {
                if(!turfSlot.isBooked())
                {
                    if(amount>=turfSlot.getSlotPrice())
                    {
                        int
                            bookID=RandomNumberGenerator.generateRandomNumber(1000,2000);
                        turfSlot.setBooked(true);
                        book=new Book(slotID,turfID,bookID);
                        break;
                    }
                    else
                    {
                        System.out.println("Insufficient amount");
                        return null;
                    }

                }
                else
                {
                    System.out.println("Slot is already booked");
                    return null;
                }
            }
            else
            {
                System.out.println("Please enter a valid slot ID");
                return null;
            }
        }
        return book;
    }

}
```

2.4 Interface Segregation Principle (ISP) and its application on the project:

Segregation means keeping things separated, and the Interface Segregation Principle is about separating the interfaces. The ISP states that "Clients should not be forced to implement any methods they do not use. Rather than one fat interface, numerous little interfaces are preferred, based on groups of methods, with each interface serving one submodule."

ISP in BookATurf:

1. 'IAvailability' interface: This interface imposes a contract to check the availability of a turf slot in a certain time.

```
public interface IAvailability {

    public int isSlotAvailable(LocalTime startingTime, LocalTime endingTime);
}
```

2. 'IDetails' interface: This interface imposes a contract to provide the details of turfs and their slots to the users.

```
public interface IDetails {

    public void getDetails(int turfID);
}
```

3. 'IBook' interface: This interface provides methods of booking to be implemented by a class.

```
public interface IBook {

    public Book bookSlot(int slotID,int amount);

}
```

2.5 Dependency Inversion Principle (DIP) and its application on the project:

DIP, the fifth SOLID principle, states that high-level modules/classes should not depend on low-level modules/classes. Instead, both should depend upon abstractions.

Secondly, abstractions should not depend on implementation details; details should depend upon abstractions.

DIP in BookATurf: The 'ArtificialGrassTurf', 'NaturalGrassTurf' and 'CricketTurf' classes depend on abstractions (interfaces) like 'IAvailability', 'IBook', 'IDetails' and do not depend on concrete implementation.

Chapter 3

Avoided Code Smells in the Project

Code Smells: In computer programming, a code smell is any characteristic in the source code of a program that possibly indicates a deeper problem.

Some of the code smells that were avoided while designing the project were:

1. Inappropriate Name: The code smell "Inappropriate Name" refers to situations where variable, method, or class names are misleading, unclear, or do not accurately represent their purpose.

   In the provided project, there are instances of well-named variables, methods, and classes. Here are a few examples:

   Well-named Classes:

   TurfSlot, ArtificialGrassTurf, NaturalGrassTurf, CricketTurf, etc., are appropriately named and convey their purpose clearly.

   Descriptive Method Names:

   Methods like getDetails, isSlotAvailable, bookSlot, and others are named in a way that accurately reflects their functionality

   Clear Variable Names:

   Variable names such as userID, userEmail, password, startingTime, endingTime, etc., are clear and provide a good understanding of their roles.

2. Duplicate Code: For different type of Turfs like ArtificialGrassTurf, NaturalGrassTurf and CricektTurf the booking methods and slot availability checker are implemented in a reusable manner. Moreover, the project uses interfaces (IAvailability, IDetails, IBook) to define common behaviors that are implemented by different classes. This approach ensures consistency in method signatures and functionality while allowing flexibility in individual implementations.

3. Conditional Complexity: The project does not contain many switch cases or if-else block making the code more readable.

4. Large Class: The "Large Class" code smell occurs when a class has grown too large, often resulting in low cohesion and high coupling. This can make the class harder to understand, maintain, and extend. Large classes may handle multiple responsibilities, violating the Single Responsibility Principle (SRP) from the SOLID principles.

   In BookATurf the classes are responsible for specific things and has only single reason to change. This makes the classes small and precised for single responsibility.

5. Feature Envy: A method that seems more interested in some other class than the one it is in.

   In this project, the classes have distributed functionalities thus it prevents the classes from being overly complex.

6. Lazy Class: An intermediate base class that isn't really required or a container of another object that does not add any additional functionality.

   In the project there no intermediate base classes or container of another object without purpose.

7. Refused Bequest: Subclasses inherit code that they don't want which is a possible violation of LSP.

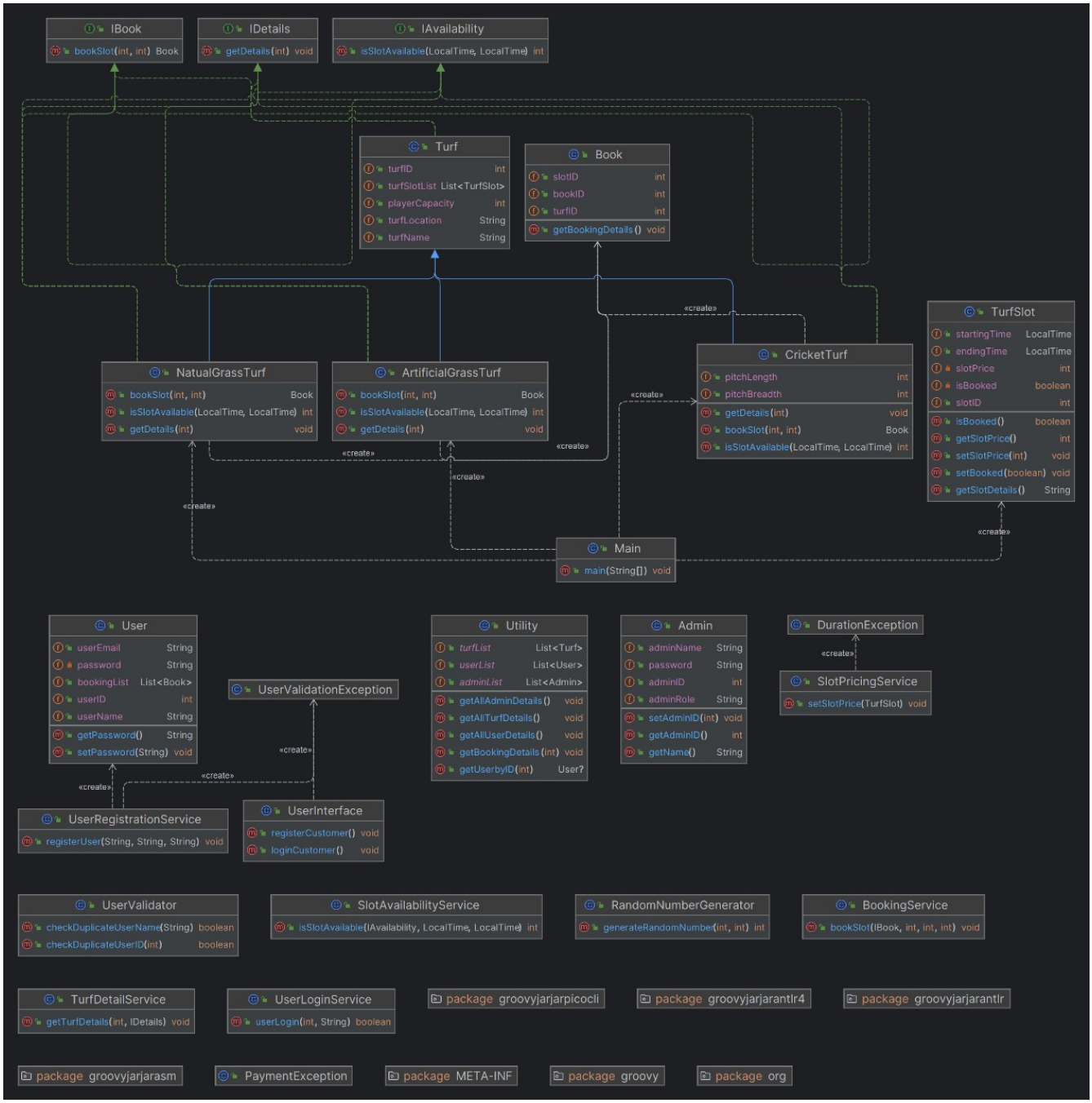
   The program is designed in a such a way that it does not viloate LSP so there are no space for refused bequest code smell.

8. Shotgun Surgery: To introduce a small new change, a developer has to change many classes and methods, and most of the time has to write duplicated code, which violates the "Don't Repeat Yourself" principle.

   The program has defined separation of concerns in the classes and the methods. Single Responsibility of the classes makes the program free from shotgun surgery.

Chapter 4

UML



Chapter 5

Github Link of the Project

https://github.com/Kashshaf-Labib/BookATurf-Console-App.git