UIT2301- PROGRAMMING AND DESIGN PATTERNS

Assignment 1

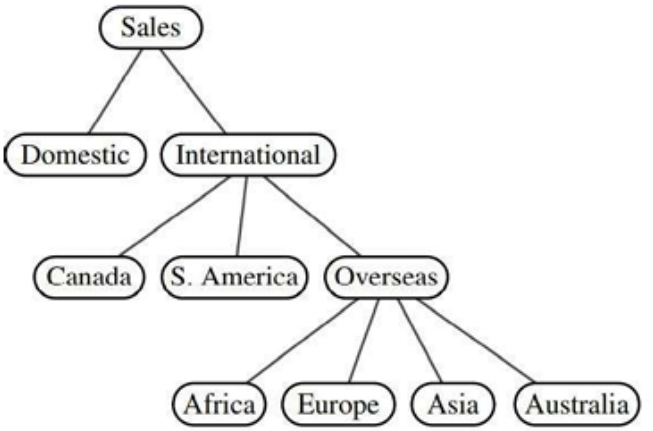
Name: Santhosh L

Reg No: 3122 23 5002 111

Class: IT C

Q1.

The indented parenthetic representation of a tree T is a variation of the parenthetic representation of T that uses indentation and line breaks (as illustrated below). Implement a program that prints this representation for an expression tree using appropriate Object-oriented methodology.



(a)

Sales (
  Domestic
  International (
    Canada
    S. America
    Overseas (
      Africa
      Europe
      Asia
      Australia
)
)
)

(b)

```
class TreeNode:
    def _init_(self, value):
        self.value = value
        self.children = []
```

```
    def add_child(self, child_node):
        self.children.append(child_node)

    def display(self, level=0):
        indent = '    ' * level
        print(f"{indent}{self.value}", end=" ")

        if self.children:
            print("(")
            for child in self.children:
                child.display(level + 1)
            print(f"{indent})")
        else:
            print()

root = TreeNode("Sales")
domestic = TreeNode("Domestic")
international = TreeNode("International")
canada = TreeNode("Canada")
s_america = TreeNode("S. America")
overseas = TreeNode("Overseas")
africa = TreeNode("Africa")
europe = TreeNode("Europe")
asia = TreeNode("Asia")
australia = TreeNode("Australia")

root.add_child(domestic)
root.add_child(international)
domestic.add_child(canada)
domestic.add_child(s_america)
```

```
international.add_child(overseas)
overseas.add_child(africa)
overseas.add_child(europe)
overseas.add_child(asia)
overseas.add_child(australia)

root.display()
```

Q2.

Design library management system (LMS). The system should have the following entities: Library, Book, Author, and Patron.

Requirements

1. A Library can have multiple Books, and each Book is associated with one Library.

2. Each Book can have one or more Authors, and each Author can write multiple Books.

3. Patrons can borrow Books from the Library, and each Book can be borrowed by multiple Patrons.

4. Books should have attributes such as Title, ISBN, and Publication Year.

5. Authors should have attributes such as Name and Birthdate.

6. Patrons should have attributes such as Name, Library Card Number, and Contact Information.

Draw the UML Diagram for the following:

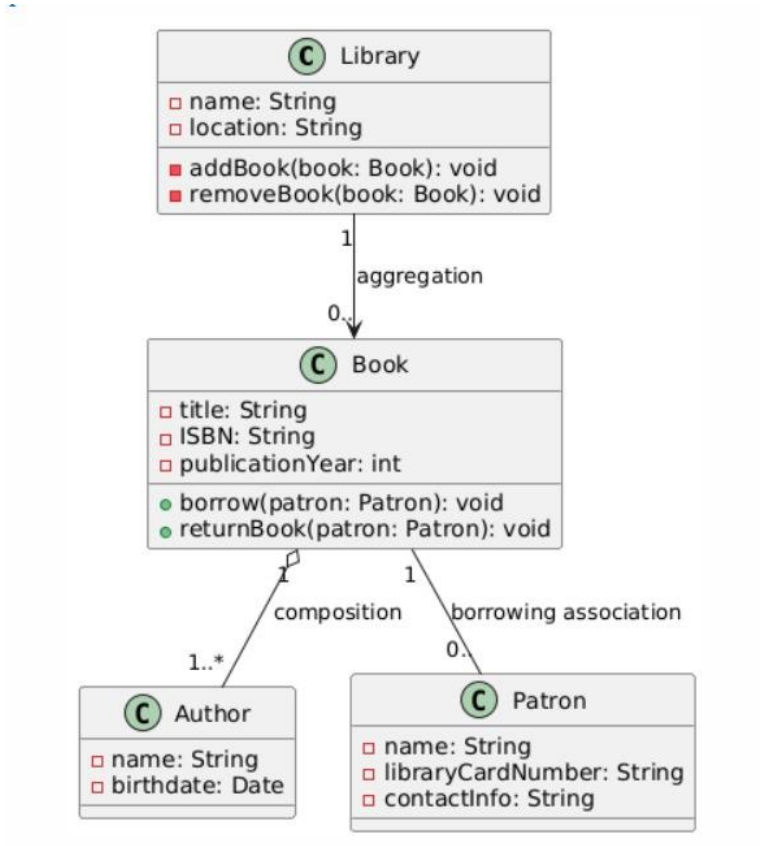
Class Diagram

Use Case Diagram

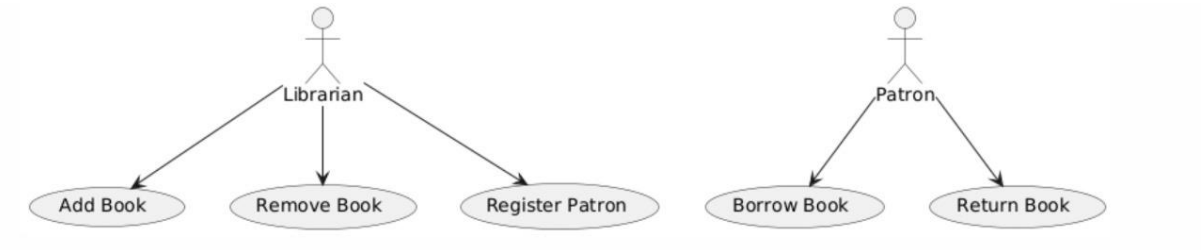
Logical view of LMS

a. Create a UML class diagram that accurately represents the relationships and attributes of the Library, Book, Author, and Patron classes.
b. Use appropriate notations to indicate the relationships between these classes, specifically distinguishing between Composition and Aggregation where necessary.
c. Provide a brief explanation of your choice of Composition or Aggregation for each relationship in your UML diagram and justify your decision.

a)



b)



c) We have used composition and aggregation in the UML diagram in various positions and the following are the reasons for their usage:

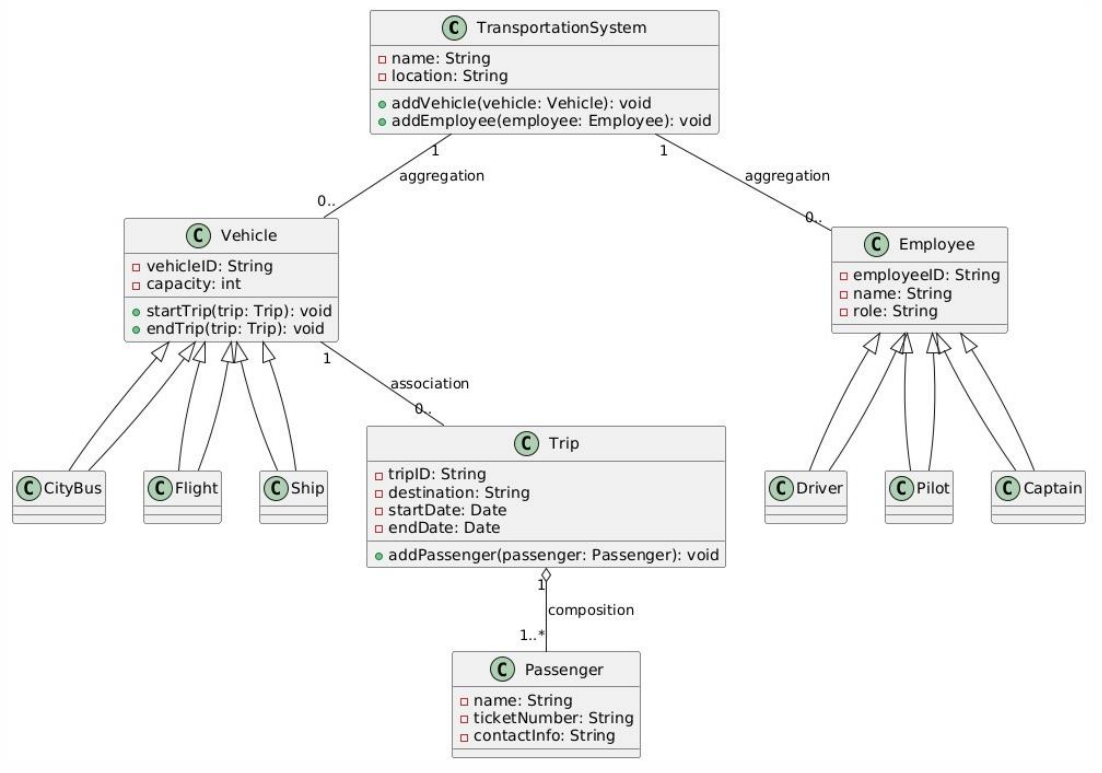
- Composition is used where there are multiple Books per Library, with each Book assigned to one Library.
- Aggregation is used where there are one or more Authors per Book with each Author possibly writing multiple Books.
- Association is used where there are multiple Patrons borrowing the same Book, with each Patron able to borrow multiple Books.

Q3.

Apply the object-oriented knowledge to the model any transportation system like roadways, waterways, airways etc in a transport system.

a. Illustrate the class design of the system demonstrating the various properties such as association, relationships, composition, aggregation, and inheritance.
b. Create a package to implement the model and clearly illustrate the abstraction levels of various departments in the department, ex City Bus, Mofussil bus, Employees, passengers, trip.
c. Create a package to implement the model and clearly illustrate the abstraction levels
d. Define the __init__.py module for all levels of abstraction
e. Write the command to install this package within other models.

a)



b)

roadways/city_bus.py

```
class CityBus:
    def _init_(self, bus_id, capacity):
        self.bus_id = bus_id
        self.capacity = capacity

    def start_trip(self):
        print("City bus trip started")

    def end_trip(self):
        print("City bus trip ended")
```

roadways/mofussil_bus.py

```
class MofussilBus:
    def _init_(self, bus_id, capacity):
```

```
        self.bus_id = bus_id
        self.capacity = capacity

    def start_trip(self):
        print("Mofussil bus trip started")

    def end_trip(self):
        print("Mofussil bus trip ended")
```

roadways/trip.py

```
class RoadwayTrip:
    def _init_(self, trip_id, route, bus):
        self.trip_id = trip_id
        self.route = route
        self.bus = bus
        self.passengers = []

    def add_passenger(self, passenger):
        self.passengers.append(passenger)
```

airways/flight.py

```
class Flight:
    def _init_(self, flight_number, capacity):
        self.flight_number = flight_number
        self.capacity = capacity

    def take_off(self):
        print("Flight has taken off")

    def land(self):
```

```
        print("Flight has landed")
```

airways/trip.py

```
class AirwayTrip:
    def _init_(self, trip_id, origin, destination, flight):
        self.trip_id = trip_id
        self.origin = origin
        self.destination = destination
        self.flight = flight
        self.passengers = []

    def add_passenger(self, passenger):
        self.passengers.append(passenger)
```

employees/driver.py

```
class Driver:
    def _init_(self, name, license_number):
        self.name = name
        self.license_number = license_number

    def drive(self):
        print("Driver is driving the vehicle")
```

employees/pilot.py

```
class Pilot:
    def _init_(self, name, license_number):
        self.name = name
        self.license_number = license_number

    def fly(self):
```

```
        print("Pilot is flying the aircraft")
```

passengers/passenger.py

```
class Passenger:
    def _init_(self, name, age, ticket_id):
        self.name = name
        self.age = age
        self.ticket_id = ticket_id



from abc import ABC, abstractmethod
from typing import List
class Transportation(ABC):
    @abstractmethod
    def get_capacity(self):
        pass

    @abstractmethod
    def get_speed(self):
        pass

    @abstractmethod
    def get_mode(self):
        pass

class RoadTransport(Transportation):
    def __init__(self, vehicle_type, license_plate):
        self.vehicle_type = vehicle_type
        self.license_plate = license_plate
```

```
    def get_capacity(self):
        return 40

    def get_speed(self):
        return 60

    def get_mode(self):
        return "Road"

class Bus(RoadTransport):
    def __init__(self, vehicle_type, license_plate, bus_number, route):
        super().__init__(vehicle_type, license_plate)
        self.bus_number = bus_number
        self.route = route

    def start_trip(self):
        print(f"Bus {self.bus_number} starting trip on route {self.route}")

    def end_trip(self):
        print(f"Bus {self.bus_number} ending trip on route {self.route}")

class Passenger:
    def __init__(self, name, ticket_number, seat_number):
        self.name = name
        self.ticket_number = ticket_number
        self.seat_number = seat_number

    def book_ticket(self):
        print(f"Passenger {self.name} booked a ticket with seat number {self.seat_number}")

class Trip:
```

```
    def __init__(self, transport, passengers: List[Passenger]):
        self.transport = transport
        self.passengers = passengers
        self.departure_time = None
        self.arrival_time = None

    def start_trip(self):
        self.transport.start_trip()

    def end_trip(self):
        self.transport.end_trip()

    def add_passenger(self, passenger: Passenger):
        self.passengers.append(passenger)
```

c)

1. Transportation (Abstract Base Class)

- Represents any form of transportation (road, water, air).
- **Methods:** get_capacity(), get_speed(), get_mode()

2. RoadTransport (Derived from Transportation)

- Represents road-based transport like buses, cars, trucks, etc.
- Properties: vehicle_type, license_plate
- **Methods:** start_trip(), end_trip()

3. WaterTransport (Derived from Transportation)

- Represents water-based transport like ships, ferries, etc.
- Properties: vessel_type, port
- **Methods:** start_trip(), end_trip()

4. AirTransport (Derived from Transportation)

- Represents air-based transport like airplanes, helicopters, etc.
- Properties: flight_number, airport
- **Methods:** start_trip(), end_trip()

5. Trip (Association with Transportation)

- Represents a specific trip undertaken by a vehicle.
- Properties: departure_time, arrival_time, route
- **Methods:** calculate_trip_duration(), check_trip_status()

6. Employee (Aggregation with Transportation)

- Represents an employee in the transportation system (driver, pilot, crew).
- Properties: name, role, id
- **Methods:** assign_to_trip()

7. Passenger(Aggregation with Trip)

- Represents a passenger traveling on a particular trip.
- Properties: name, ticket_number, seat_number
- **Methods:** book_ticket()

8. Bus (Composition with RoadTransport)

- A specific type of road transport.
- Properties: bus_number, route
- **Methods:** start_trip(), end_trip()

9. CargoShip (Composition with WaterTransport)

- A specific type of water transport for cargo.
- Properties: ship_id, cargo_capacity
- **Methods:** load_cargo(), unload_cargo()

10. Flight (Composition with AirTransport)

- A specific type of air transport.
- Properties: flight_number, capacity
- **Methods:** board_passengers(), land()

1. CityBus (Level 1 Abstraction)

- Provides functionality for local road transport.
- **Methods:** start_trip(), end_trip()

2. MofussilBus (Level 1 Abstraction)

- Represents intercity buses.
- **Methods:** start_trip(), end_trip()

3. Employees (Level 2 Abstraction)

- Aggregation of employees who work in the transportation system.
- **Methods:** assign_employee_to_vehicle()

4. Passengers (Level 2 Abstraction)

- Aggregation of passengers who book tickets for a trip.
- **Methods:** book_ticket(), get_seat_number()

5. Trip (Level 3 Abstraction)

- Represents a trip that a passenger takes from one point to another.
- **Methods:** track_trip(), calculate_fare()

1. Package TransportSystem

- Sub-package RoadTransport

- **Classes:** Bus, CityBus, MofussilBus
- **Responsibilities:** Managing buses, their trips, and passengers.

- Sub-package WaterTransport

- **Classes:** CargoShip, Ferry

- **Responsibilities:** Managing ships, loading/unloading cargo, water-based **trips.**

- Sub-package AirTransport

- **Classes:** Flight, Helicopter

- **Responsibilities:** Managing air-based trips, boarding passengers, landing.

2. Package Departments

- Sub-package EmployeeDepartment

- Classes: Driver, Pilot, CrewMember

- **Responsibilities:** Assign employees to trips, manage schedules.

- Sub-package PassengerDepartment

- Classes: Passenger, Ticketing

- Responsibilities: Book tickets, assign seats, track passenger details.

3. Package TripManagement

- Classes: Trip, BookingSystem, RouteManagement

- Responsibilities: Track trips, manage routes, calculate fares, handle booking.

d)

1. **Level 1 Abstraction:**
   This includes classes like CityBus, MofussilBus, and other transport types that represent specific vehicles or modes of transport.
2. **Level 2 Abstraction:**
   Includes the Employee and Passenger classes, which manage the roles and operations related to the employees and passengers in the system.
3. **Level 3 Abstraction:**
   Includes the Trip and BookingSystem classes, which represent higher-level operations like managing and tracking trips and bookings.

finite.py Module

```
from abc import ABC, abstractmethod

from typing import List

import datetime
```

```
class Transportation(ABC):
    @abstractmethod
    def get_capacity(self):
        pass

    @abstractmethod
    def get_speed(self):
        pass

    @abstractmethod
    def get_mode(self):
        pass

class RoadTransport(Transportation):
    def __init__(self, vehicle_type, license_plate):
        self.vehicle_type = vehicle_type
        self.license_plate = license_plate

    def get_capacity(self):
        return 40

    def get_speed(self):
        return 60

    def get_mode(self):
        return "Road"

class CityBus(RoadTransport):
    def __init__(self, vehicle_type, license_plate, bus_number, route):
        super().__init__(vehicle_type, license_plate)
        self.bus_number = bus_number
```

```
        self.route = route

    def start_trip(self):
        print(f"CityBus {self.bus_number} starting trip on route {self.route}")

    def end_trip(self):
        print(f"CityBus {self.bus_number} ending trip on route {self.route}")

class MofussilBus(RoadTransport):
    def __init__(self, vehicle_type, license_plate, bus_number, route):
        super().__init__(vehicle_type, license_plate)
        self.bus_number = bus_number
        self.route = route

    def start_trip(self):
        print(f"MofussilBus {self.bus_number} starting trip on route {self.route}")

    def end_trip(self):
        print(f"MofussilBus {self.bus_number} ending trip on route {self.route}")

class Employee:
    def __init__(self, name, role, employee_id):
        self.name = name
        self.role = role
        self.employee_id = employee_id

    def assign_to_vehicle(self, vehicle):
        print(f"Employee {self.name} assigned to {vehicle.vehicle_type} with license plate
{vehicle.license_plate}")

class Passenger:
```

```
    def __init__(self, name, ticket_number, seat_number):
        self.name = name
        self.ticket_number = ticket_number
        self.seat_number = seat_number

    def book_ticket(self):
        print(f"Passenger {self.name} booked a ticket with seat number {self.seat_number}")

class Trip:
    def __init__(self, transport: Transportation, passengers: List[Passenger]):
        self.transport = transport
        self.passengers = passengers
        self.departure_time = None
        self.arrival_time = None

    def start_trip(self):
        self.departure_time = datetime.datetime.now()
        self.transport.start_trip()
        print(f"Trip started at {self.departure_time}")

    def end_trip(self):
        self.arrival_time = datetime.datetime.now()
        self.transport.end_trip()
        print(f"Trip ended at {self.arrival_time}")

    def add_passenger(self, passenger: Passenger):
        self.passengers.append(passenger)
        print(f"Passenger {passenger.name} added to the trip")

class BookingSystem:
    def __init__(self):
```

```
        self.bookings = []

    def create_booking(self, passenger: Passenger, trip: Trip):
        self.bookings.append({"passenger": passenger, "trip": trip})
        print(f"Booking created for {passenger.name} on trip {trip.transport.license_plate}")

    def get_booking_info(self):
        for booking in self.bookings:
            passenger = booking["passenger"]
            trip = booking["trip"]
            print(f"Booking Info: {passenger.name} - Trip on {trip.transport.license_plate}")

if __name__ == "__main__":
    bus = CityBus("Bus", "XYZ-123", "CB001", "Route 1")
    mofussil_bus = MofussilBus("Bus", "ABC-789", "MB001", "Route 2")

    passenger1 = Passenger("John Travolta", "T123", "A1")
    passenger2 = Passenger("Jane Diana", "T124", "A2")

    trip = Trip(bus, [passenger1, passenger2])
    trip.start_trip()
    trip.end_trip()
    booking_system = BookingSystem()
    booking_system.create_booking(passenger1, trip)
    booking_system.get_booking_info()
```

e)

transport_system/

```
|
```

├── finite.py

├── __init__.py

├── setup.py

setup.py

```
from setuptools import setup, find_packages

setup(
    name="transport_system",      # The name of your package
    version="0.1",                # The version of your package
    packages=find_packages(),     # Automatically find all packages
    install_requires=[
        # 'some_package',
    ],
    author="Your Name",
    author_email="your.email@example.com",
    description="A package for managing transportation systems including road, water, and air transport",
    long_description="This package provides a model for a transportation system with vehicles, trips, employees, and passengers.",
    long_description_content_type="text/markdown",
    url="https://github.com/yourusername/transport_system",
    classifiers=[
        "Programming Language :: Python :: 3",
        "License :: OSI Approved :: MIT License",
        "Operating System :: OS Independent",
    ],
)
```

```
cd /path/to/transport_system

bash

pip install .

from transport_system import finite
```

```
bus = finite.CityBus("Bus", "XYZ-123", "CB001", "Route 1")
passenger1 = finite.Passenger("John Doe", "T123", "A1")
trip = finite.Trip(bus, [passenger1])

trip.start_trip()
trip.end_trip()
```

Q4.

Understand the concept of exceptions in real-time systems such as railway reservations. Handling these exceptions gracefully by providing informative error messages and options for users to take corrective actions is crucial for a smooth user experience.

1. Invalid Input Data:
   - `InvalidInputException`: Raised when the user provides invalid input data, such as non-numeric values for the number of tickets or incorrect station names.
2. Seat Availability:
   - `SeatUnavailableException`: Raised when a user tries to reserve a seat, but all seats in a particular class or on a specific train are already booked.
3. Payment Issues:
   - `PaymentFailureException`: Raised when a payment transaction fails, either due to insufficient funds or a technical issue with the payment gateway.
4. Booking Conflicts:
   - `BookingConflictException`: Raised when there is a conflict between two or more bookings, such as double-booking the same seat or overlapping reservation timings.
5. Train Cancellation:
   - `TrainCancelledException`: Raised when a user attempts to book a seat on a train that has been canceled or is not operational on a specific date.
6. Invalid Dates:
   - `InvalidDateException`: Raised when the user provides a date that is in the past or too far in the future.
7. Database Errors:
   - `DatabaseConnectionException`: Raised when there are issues connecting to the database or executing database queries for reservations.
8. Authentication and Authorization:
   - `AuthenticationException`: Raised when a user provides incorrect login credentials.
   - `AuthorizationException`: Raised when a user attempts to perform an action for which they do not have the necessary permissions.
9. Ticket Cancellation:
   - `TicketCancellationException`: Raised when there is an issue with canceling a ticket, such as attempting to cancel a ticket after the cancellation deadline has passed.
10. Network and System Errors:
   - `NetworkErrorException`: Raised when there are network-related issues, such as a loss of internet connectivity during a booking attempt.
   - `SystemErrorException`: Raised when there are system-level errors or unhandled exceptions in the reservation system.
11. Seat Upgrade Issues:
   - `SeatUpgradeException`: Raised when a user tries to upgrade their seat to a higher class, but no seats are available in that class.

```
class RailwayReservationException(Exception):
    pass

class InvalidInputException(RailwayReservationException):
    pass

class SeatUnavailableException(RailwayReservationException):
    pass

class PaymentFailureException(RailwayReservationException):
    pass

class BookingConflictException(RailwayReservationException):
```

```
    pass

class TrainCancelledException(RailwayReservationException):
    pass

class InvalidDateException(RailwayReservationException):
    pass

class DatabaseConnectionException(RailwayReservationException):
    pass

class AuthenticationException(RailwayReservationException):
    pass

class AuthorizationException(RailwayReservationException):
    pass

class TicketCancellationException(RailwayReservationException):
    pass

class NetworkErrorException(RailwayReservationException):
    pass

class SystemErrorException(RailwayReservationException):
    pass

class SeatUpgradeException(RailwayReservationException):
    pass

def reserve_seat(seat_id, user, payment_info):
    try:
```

```
    if not is_seat_available(seat_id):
        raise SeatUnavailableException("The seat is already booked.")

    if not user.is_authenticated:
        raise AuthenticationException("User not authenticated.")

    if not user.has_permission("book_seat"):
        raise AuthorizationException("User not authorized to book this seat.")

    if not process_payment(payment_info):
        raise PaymentFailureException("Payment transaction failed.")

    confirm_booking(seat_id, user)

except InvalidInputException as e:
    print(f"Invalid input provided: {e}")
except SeatUnavailableException as e:
    print(f"Seat unavailable: {e}")
except PaymentFailureException as e:
    print(f"Payment issue: {e}")
except BookingConflictException as e:
    print(f"Booking conflict: {e}")
except TrainCancelledException as e:
    print(f"Cannot book: {e}")
except InvalidDateException as e:
    print(f"Invalid date: {e}")
except DatabaseConnectionException as e:
    print(f"Database error: {e}")
except AuthenticationException as e:
    print(f"Authentication error: {e}")
except AuthorizationException as e:
```

```
    print(f"Authorization error: {e}")
except TicketCancellationException as e:
    print(f"Ticket cancellation issue: {e}")
except NetworkErrorException as e:
    print(f"Network error: {e}")
except SystemErrorException as e:
    print(f"System error: {e}")
except SeatUpgradeException as e:
    print(f"Seat upgrade error: {e}")
except RailwayReservationException as e:
    print(f"An unexpected error occurred: {e}")
finally:
    print("Booking process complete.")
```