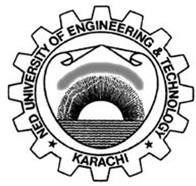
# NED UNIVERSITY OF ENDINEERING & TECHNOLOGY

**Department of Software Engineering**



ParkFlow VDM

# Formal Methods of Software Engineering (SE-313)

## Batch: 2021

**Instructor: Dr.Mustafa Lati**

## Group: B-08

**M. Talal Zafar (SE-21076) Ismail Amed Bajwa (SE-21080)**

# Scope of The Document:

The scope of the VDM document for the automated parking management system project is to create 4+1 architectural view models while applying the Vienna Development Methodology (VDM). This involves defining key functionalities and interactions in the logical view, exploring dynamic processes in the process view, depicting the physical architecture in the physical view, and focusing on software development aspects in the development view. The VDM application includes systematic requirements analysis, detailed specification for various views, structured design and implementation, com- prehensive testing, thorough documentation, and iterative development for continuous improvements. The document aims to facilitate effective stakeholder communication, establish a sustainable foundation for system maintenance, and ensure compliance with the principles of the Vienna Development Methodology.

# Project Information:

* 1. **Problem Statement**

In urban areas, parking congestion and inefficiencies are a significant challenge, leading to increased traffic congestion, wasted time, and environmental concerns. Traditional parking systems often lack optimization, real-time information, and streamlined processes, resulting in a suboptimal user experience. There is a need for an advanced automated parking management system that optimizes space utilization, reduces congestion, and enhances the overall parking experience for users.

# OverView:

ParkFlow is an innovative automated parking management system designed to address the challenges associated with traditional parking systems. The system aims to redefine and simplify parking operations by providing real-time space allocation, smart navigation, and streamlined payment processes. By leveraging technology, ParkFlow seeks to optimize space utilization, reduce congestion, and enhance the overall user experience in parking facilities.

# Scope

ParkFlow is an innovative automated parking management system designed to The scope of the automated parking management system encompasses a comprehensive set of features aimed at revolutionizing the traditional parking experience. The system will enable drivers to register seamlessly, providing name and vehicle number. Upon registration, each driver will be assigned a unique identifier, such as a ticket, QR code, or license plate recognition, linking them to their parking session. Real-time information about available parking spaces will be dynamically displayed, allowing drivers to choose spaces based on preferences like proximity to exits. The system will automate the allocation of selected parking spaces and record entry times as drivers enter.

For the exit process, the system will prompt drivers to provide their unique identifier for verification. The calculation of parking fees will be automated, considering the duration of the parking session, and the calculated fee will be transparently displayed on the system interface. The payment process will be streamlined, accommodating various electronic payment methods. Exit times will be accurately recorded to finalize parking sessions, and the system will maintain updated records for administrative purposes.

Overall, the scope aims to optimize space utilization, improve user experience, and introduce efficiency and transparency in parking operations..

# Objectives

The scope of the ParkFlow project includes the development and implementation of an automated parking management system with the following key features:

* + - **Optimize Space Utilization:** Maximize the utilization of parking spaces within the facility through dynamic allocation and optimization algorithms.
    - **Reduce Congestion:** Minimize traffic congestion in parking facilities by providing efficient navigation and space allocation.
    - **Enhance User Experience:** Improve the overall user experience by offering streamlined registration, real-time information, and user-friendly interfaces.
    - **Ensure Security and Reliability:** Implement robust security measures to safeguard user data, financial transactions, and overall system reliability.
    - **Provide Seamless Payments:** Offer a seamless and secure payment process for parking fees, enhancing user convenience.

# Project Information:

* 1. **Functional Requirements**

## User Registration:

The driver enters the parking facility and interacts with the system to register. This could involve entering personal details, vehicle information, and payment preferences.

After registration, the system may provide a unique identifier (e.g., ticket, QR code, or license plate recognition) to associate the driver with their parking session.

## Available Spaces Display:

The system displays information about available parking spaces to the driver, enabling them to choose a space based on preferences such as proximity to an exit or specific areas within the facility.

## Space Allocation and Entry:

The driver selects an available parking space, and the system allocates that space to the driver. The system records the entry time as the driver enters the parking space.

## Exit Process:

When the driver is ready to leave, they approach the exit barrier. The system prompts the driver to provide their unique identifier (ticket, QR code, or license plate) to verify their parking session.

## Payment Calculation:

The system calculates the parking fee based on the duration of the parking session. This can be done by subtracting the entry time from the exit time and applying the relevant parking rates. The driver sees the calculated fee on the system interface.

## Payment Process:

The driver makes the payment using the preferred payment method (e.g., credit card, mobile payment, or other electronic means). Once the payment is confirmed, the barrier is lifted, allowing the driver to exit the parking facility.

## Exit and Record Keeping:

The system records the exit time, finalizes the parking session, and updates relevant records. Optionally, the system can provide an electronic receipt or confirmation to the driver.

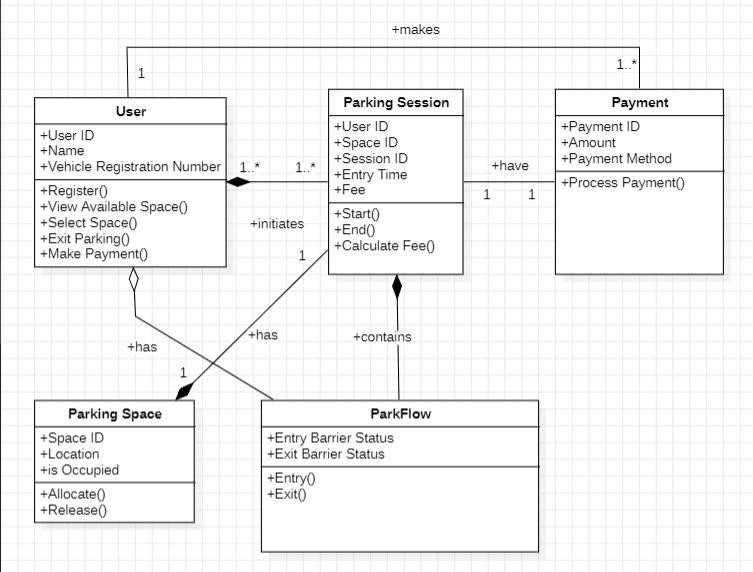
# 4+1 View Model:

The 4+1 architectural view model serves as a robust framework for comprehensively describing software systems. By leveraging five concurrent views, it addresses the diverse concerns of stakeholders such as end-users, developers, system engineers, and project managers. This model is particularly beneficial for the ParkFlow project, providing a holistic understanding of the automated parking system's architecture from different vantage points.

ParkFlow is devised to streamline and automate the parking process, enhancing user experience, optimizing space utilization, and ensuring efficient management. The 4+1 view model for ParkFlow delineates the system into logical, development, process, and physical perspectives. Through these perspectives, various stakeholders gain insights into different aspects of the system.

# Logical View:

* + 1. **Class Diagram:**



# Classes:

**User:** Represents a person using the parking system. Attributes include user ID, name, and vehicle registration number.

**Parking Session:** Represents a single instance of a vehicle parked in the lot. Attributes include session ID, entry time, and fee.

**Payment:** Represents a payment made for a parking session. Attributes include payment ID, amount, and payment method. **Parking Space:** Represents a designated area for parking a vehicle. Attributes include space ID, location, and whether it's occupied.

**ParkFlow:** Represents the core system managing the parking lot operations.

# Relationships:

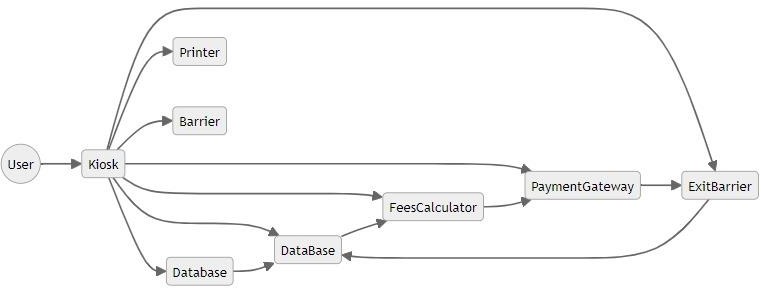
* + - * + User has a one-to-many relationship with Parking Session, meaning one user can have multiple parking sessions.
        + Parking Session has a one-to-one relationship with Parking Space, meaning a session is assigned to a specific space.
        + ParkFlow has a one-to-many relationship with Parking Space, meaning the system manages multiple spaces.
        + ParkFlow interacts with all other classes through various methods:
        + User: Register(), View Available Spaces(), Select Space(), Exit Parking(), Make Payment().
        + Parking Session: Start(), End(), Calculate Fee().
        + Payment: Process Payment().
        + Parking Space: Allocate(), Release().

This diagram suggests the system focuses on managing individual parking sessions rather than long-term parking arrangements.

The payment method for each session is stored separately, implying pay-per-session parking.The ParkFlow system takes an active role in managing parking spaces and payments.

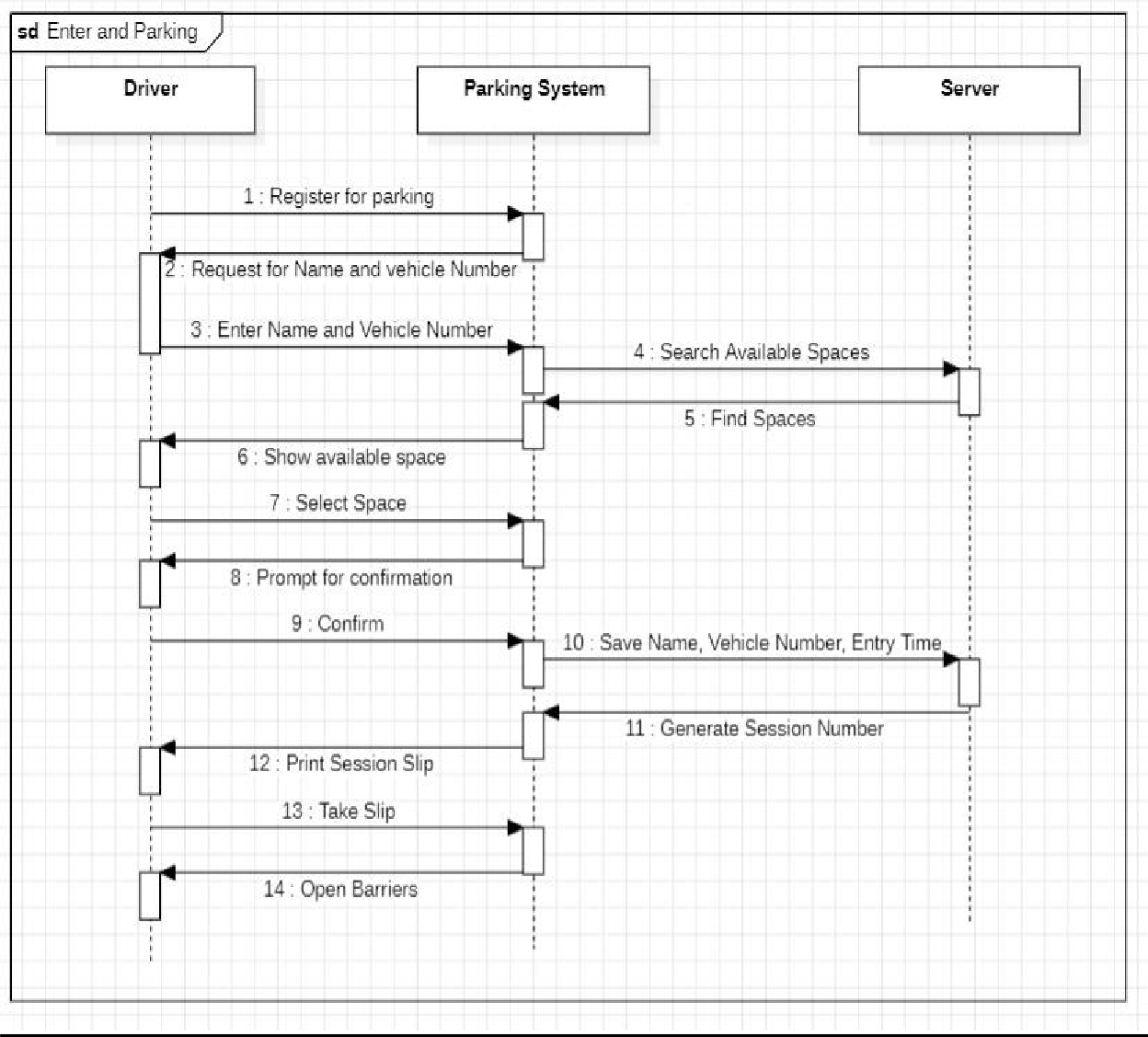
# Development View:

This view the sequence of steps involved in processing payments through a kiosk. The user initiates the process by entering their payment details into the kiosk, which securely relays the information to a payment gateway. The payment gateway meticulously verifies the provided payment information, ensuring its accuracy and matching it to the user's account details. If the payment information is deemed valid, the gateway forwards it to a fees calculator, which calculates the associated transaction fees. The computed fees are then returned to the gateway, which incorporates them into the finalized payment information. This comprehensive information is then stored in a secure database, and a receipt is printed for the user's reference. Upon successful processing and receipt issuance, the barrier opens, allowing the user to proceed.



# Process View:

## Enter & Parking:

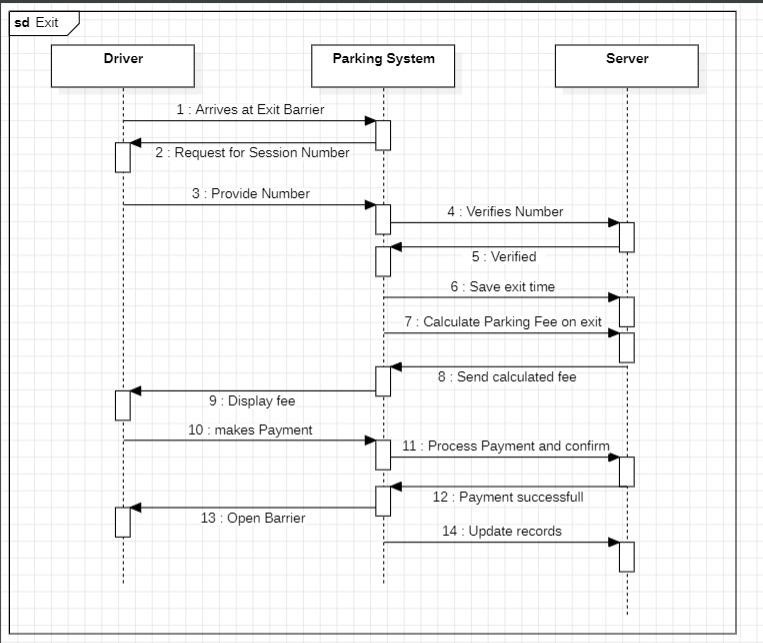


The main steps in the process are as follows:

* **Enter and parking:** The driver enters the parking lot.
* **Register for parking:** The driver registers for parking by entering their name and vehicle number.
* **Search Available Spaces:** The system searches for available parking spaces.
* **Find Spaces:** The system displays a list of available parking spaces to the driver.
* **Select Space:** The driver selects a parking space.
* **Print Season Slip:** The system prints a season slip for the driver, which they must display in their car.
* **Take Up Spaces:** The driver parks their car in the selected space.
* **Open Barriers:** The system opens the barrier so that the driver can exit the parking lot.

The flowchart also shows some of the decisions that the system makes at each step. For example, if there are no available parking spaces, the system will direct the driver to wait or go to another parking lot.

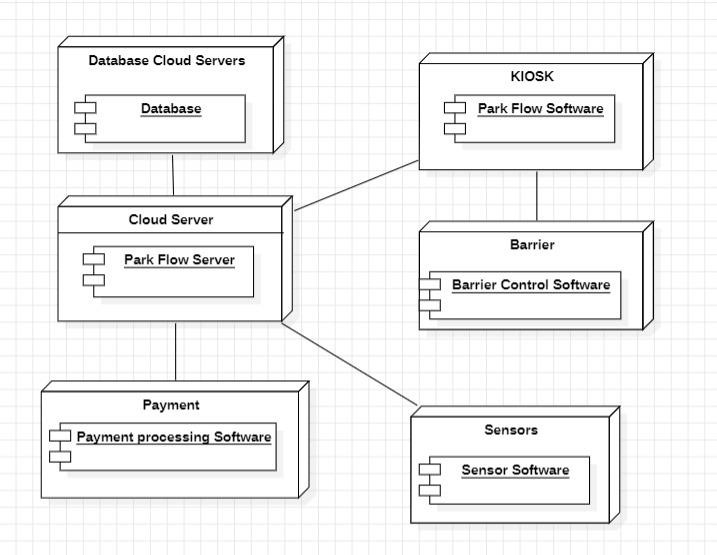
## Exit:



The flow chart you sent me depicts a parking system’s exit process. Here is a breakdown of the steps involved:

* **Arrives at Exit Barrier**: As the driver reaches the exit barrier, the system automatically detects their presence.
* **Request for Session Number:** The system prompts the driver to enter their session number, which they received upon entering the parking garage.
* **Provide Number**: The driver inputs the session number through a designated keypad or scanner.
* **Verifies Number:** The system checks the entered session number against its database to ensure its validity.
* **Verified:** If the number is valid, the system proceeds to the next step.
* **Save Exit Time:** The system records the exact time the driver attempts to exit the parking garage.
* **Calculate Parking Fee on Exit:** Based on the entry time and exit time, the system calculates the total parking fee.
* **Send Calculated Fee:** The system transmits the calculated fee to a display unit near the exit barrier.
* **Display Fee:** The fee is displayed for the driver to see.
* **Makes Payment:** The driver inserts cash or uses a payment card to pay the displayed fee.
* **Process Payment and Confirm:** The system processes the payment and verifies its success.
* **Payment Successful:** If the payment is successful, the system proceeds to the next step.
* **Open Barrier:** The system raises the exit barrier, allowing the driver to leave the parking garage.
* **Update Records:** Finally, the system updates its records to reflect the completed parking session and payment.

# 4.4 Physical View:



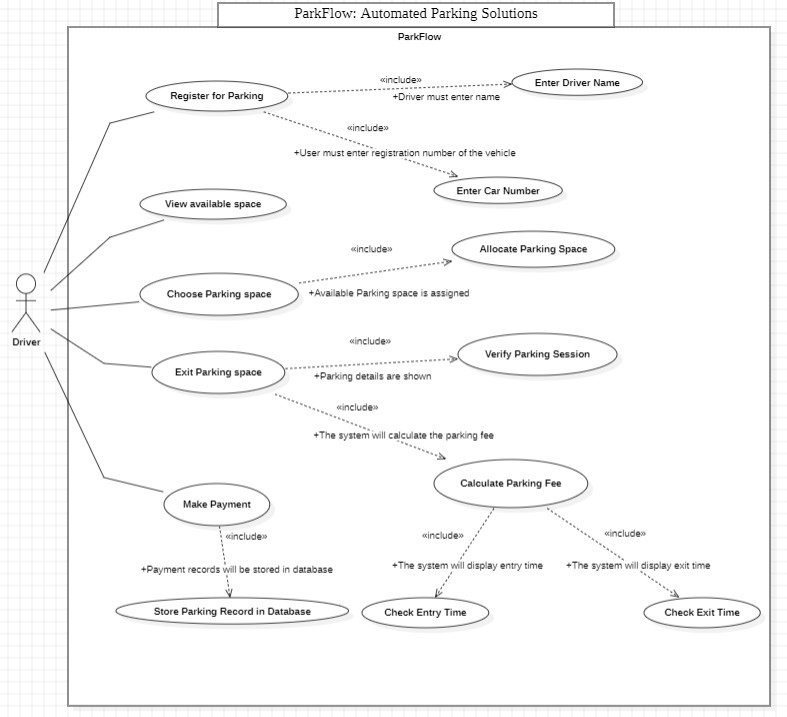
* **Sensors:** Located throughout the parking lot, these detect the presence and location of vehicles.
* **Sensor Software:** Processes the data from the sensors and transmits it to the ParkFlow Server.
* **ParkFlow Server:** The central component that controls the entire system. It receives data from the sensors, manages parking spaces, calculates fees, and communicates with other system components.
* **Database Cloud Servers:** Store information about parking spaces, users, and transactions.
* **Kiosk:** Provides users with a physical interface to interact with the system. They can register for parking, pay fees, and get information about parking availability.
* **Barrier Control Software:** Controls the opening and closing of the entry and exit barriers based on authorization from the ParkFlow Server.
* **Payment Processing Software:** Handles various payment methods, such as cash,

credit cards, and mobile payments.

## User Experience:

* **Entering the Parking Lot:** Upon entering, sensors detect the vehicle and send data to the ParkFlow Server. The barrier opens automatically if a prepaid parking session is detected or if the user pays at the kiosk.
* **Parking:** The ParkFlow Server assigns a parking space to the vehicle and tracks its location.
* **Exiting the Parking Lot:** As the vehicle approaches the exit barrier, sensors detect it and send data to the ParkFlow Server. The server calculates the parking fee based on the entry and exit times.
* **Payment:** The driver can pay the fee at the kiosk or through an automated payment system.
* Exiting the System: Once the payment is confirmed, the barrier opens, and the vehicle can leave the parking lot.

# 5. +1 Scenario(UseCase):



**Actors:**

* + **Driver**: The primary user of the system who interacts with it to park their vehicle.

**Use Cases:**

* + **Register for Parking:** This covers the driver's initial interaction with the system, providing their name and vehicle registration number.
  + **View Available Spaces:** The driver searches for and views a list of currently available parking spaces within the lot.
  + **Choose Parking Space:** The driver selects a specific vacant space from the available options.
  + **Allocate Parking Space:** The system assigns the chosen space to the driver's parking session.
  + **Enter Parking Lot:** The driver enters the parking lot, with the system potentially recognizing and authorizing their entry based on the allocated space.
  + **Park Vehicle:** The driver parks their vehicle in the assigned space.
  + **Exit Parking Space:** The driver prepares to leave their parked vehicle.
  + **Verify Parking Session:** The system identifies the driver's parking session and displays relevant information like duration and fee.
  + **Make Payment:** The driver pays the calculated parking fee through various methods like cash, credit card, or mobile payment.
  + **Exit Parking Lot:** Once the payment is confirmed, the system opens the exit barrier allowing the driver to leave the lot.
  + **Store Parking Record:** The system stores the completed parking session details in its database for future reference or reporting.

# VDM SL:

### types

ParkingSlot = map N to (N \* N \* [Char] \* [Char] \* time \* time \* Z);

### values

rate:N=20

InitialParkingSlots: ParkingSlot =

{ 1 |-> (1, 0, "ali", "7897", time`now, time`now, 0),

2 |-> (2, 0, "", "", time`now, time`now, 0),

3 |-> (3, 0, "", "", time`now, time`now, 0),

4 |-> (4, 0, "", "", time`now, time`now, 0) }

**state** ParkFlow **of**

classObj: ParkFlow

slots: ParkingSlot

slotNo: N

sessionNo: N

parkflowBothType: Z

name: [Char]

vn: [Char]

**inv mk**-ParkFlow(SlotNo, SessionNo, ParkflowBothType, name, Vn) ==

ParkflowBothType in set {0, 1, 2} and

Rate>=10 and

SlotNo >= 0 and SessionNo >= 0 and

name <> nil and Vn <> nil;

**init mk**-ParkFlow(name, Vn, SessionNo, , entTime, extTime ) ==

name:=" " and

Vn:=" " and

SessionNo:= 0

### end

**functions**

calculateFee(entryTime: Time, exitTime: Time) : N

**pre** entryTime <= exitTime

**post** RESULT >= 0;

### operations

CheckParkflowBothType()

**rd** ParkflowBothType:N

**pre** true

**post** ParkflowBothType >= 0 and ParkflowBothType <= 2

DisplaySlots()

**rd**  InitialParkingSlots

**PRE** true

**POST** true

DO

for slot in set {1, 2, 3, 4} do

status := if (slot, ClassObj.GetStatus(slot)) = (slot, 0)

then "Free"

else "Occupied";

Print("Space ", slot, " Is ", status);

printSlip()

**rd**  Name

**rd**  Vn

**pre** name <> "" and Vn <> ""

**post** true

printSlip()

**ext wr**  Name

**ext wr**  Vn

**pre** true

**post** name <> "" and Vn <> ""

displaySlots();

GetExitData()

**rd**  SessionNo

**pre** SessionNo >= 0

**post** true

AuthenticateSession();

CalculateFees(enterTime, exitTime)

**rd**  rate

**pre:** enterTime <= exitTime

**post**: RESULT >= 0

DO

RETURN (exitTime - enterTime) \* rate;

PayAmount(fees)

**rd** fees;

**pre** fees >= 0

**post** true

AuthenticateSession()

**rd** SessionNo;

**rd** exitTime;

**pre** SessionNo >= 0

**post** true

SessionNo := ext;

fees := CalculateFees(enterTime, exitTime);

PayAmount(fees);

allotSpaces()

**rd** SessionNo;

**rd** ParkflowBothType;

**rd** Vn;

**rd** name;

**pre** true

**post** SlotNo >= 0 and SlotNo <= 4

printSlip();

GetParkflowBothType()

**rd** SessionNo;

**rd** ParkflowBothType;

**pre** true

**post** ParkflowBothType>= 0 and ParkflowBothType<= 2

# VDM SL To C++ Code:

#include <iostream>

#include <map>

#include <vector>

#include <set>

#include <chrono>

#include <array>

#include <stdlib.h>

#include <random>

using namespace std;

using namespace std::chrono;

 time\_t current\_time = time(0);

std::array<std::tuple<int, int, std::string, std::string,time\_t,time\_t,int>, 4> arrayOfTuples = {

        std::make\_tuple(1, 1, "ali", "7897", current\_time, current\_time, 0),

        std::make\_tuple(2, 0, "", "", current\_time, current\_time, 0),

        std::make\_tuple(3, 0, "", "", current\_time, current\_time, 0),

        std::make\_tuple(4, 0, "", "", current\_time, current\_time, 0),

    };

class ParkFlow {

private:

    int SlotNo, SessionNo;

    int ParkflowBothType;

    string name;

    string Vn;

public:

  //Constructor

    ParkFlow(int SlotNo, int SessionNo, int ParkflowBothType,

    std::string name, std::string Vn): SlotNo(SlotNo), SessionNo(SessionNo),

    ParkflowBothType(ParkflowBothType), name(name), Vn(Vn){}

    void CheckParkflowBothType() {

        int type;

        cout << "Enter Booth Type" << endl;

        cout << "Enter 1 for entry" << endl;

        cout << "Enter 2 for exit" << endl;

        cout << "Enter 0 to Close" << endl;

        cin >> type;

        ParkflowBothType = type;

    }

    void displaySlots() {

        for (auto& [first, second, name, number,entryTime,exitTime,fees] : arrayOfTuples) {

            std::cout << "Space " << first;

            if (second == 0) {

                cout << " Is Free" << endl;

            }

            else {

                cout << " Is Occupied By Mr." << name << " with vehical number" << number << endl;

            }

        }

    }

void printSlip() {

        for (auto& [first, second, name, number,entryTime,exitTime,fees] : arrayOfTuples) {

            if (name==this->name) {

                cout <<"Dear Mr. "<<name<<endl<< "You have Booked parking slot number "<<first<<" for your car with number "<<number << endl;

                cout <<"And Your Session Number is "<<second << endl;

                cout <<"Entry Time :- "<<entryTime << endl;

                 cout << "You Space is Reserved Go And Park" << endl;

                break;

            }

        }

    }

    void GetEntryData() {

        cout << "Enter Details" << endl;

        cout << "Enter Name" << endl;

        cin >> name;

        cout << "Enter Vehicle Number" << endl;

        cin >> Vn;

        displaySlots();

    }

void GetExitData() {

        cout << "Enter Details" << endl;

        cout << "Enter Session Number" << endl;

        cin >> SessionNo;

        AuthenticateSession();

    }

  int CalculateFees(time\_t enterTime,time\_t exitTime){

     int rate=20;

         return  std::difftime(exitTime, enterTime)\*rate;

  }

 void PayAmount(int fees){

     int number;

     cout<<"Enter Card Number"<<endl;

     cin>>number;

     cout<<"Amount "<<fees<<"Rs has been deducted from your account "<<number<<endl;

     cout<<"Thank you for parking"<<endl;

 }

void AuthenticateSession(){

    for (auto& [first, second, name, number,entryTime,exitTime,fees] : arrayOfTuples) {

            if (second == SessionNo) {

exitTime = time(0);

  fees=CalculateFees(entryTime,time(0));

                cout<<"your amount is Rs"<<fees<<endl;

                PayAmount(fees);

                break;

            }

        }

}

    void allotSpaces() {

        std::random\_device rd;

        std::mt19937 gen(rd());

        std::uniform\_int\_distribution<> dist(1, 100);

        int ask;

        std::cout << "Select Avaliable Space" << endl;

        cin >> ask;

        for (auto& [first, second, name, number,entryTime,exitTime,fees] : arrayOfTuples) {

            if (ask == first && second != 0) {

                cout << " Is Occupied By " << name << endl;

                allotSpaces();

            }

            if (ask == first && second == 0) {

                second = dist(gen);

                name = this->name;

                number = this->Vn;

                entryTime = time(0);

    // entryTime=ctime(&currentTime);

                printSlip();

                break;

            }

        }

    }

    int GetParkflowBothType() {

        return ParkflowBothType;

    }

};

int main() {

    ParkFlow ClassObj;

    while(ClassObj.GetParkflowBothType()){

    ClassObj.CheckParkflowBothType();

    if (ClassObj.GetParkflowBothType() == 1) {

        ClassObj.GetEntryData();

        ClassObj.allotSpaces();

    }else if(ClassObj.GetParkflowBothType() == 2){

        ClassObj.GetExitData();

    }else if(ClassObj.GetParkflowBothType() == 0){

        break;

    }else{

        cout<<"Choose Correct Options"<<endl;

    }}

    return 0;

}

**OUTPUT:**

Enter Booth Type

Enter 1 for entry

Enter 2 for exit

Enter 0 to Close

1

Enter Details

Enter Name

Ismail

Enter Vehicle Number

BS4J7

Space 1 Is Occupied By Mr.ali with vehical number7897

Space 2 Is Free

Space 3 Is Free

Space 4 Is Free

Select Avaliable Space

3

Dear Mr. Ismail

You have Booked parking slot number 3 for your car with number BS4J7

And Your Session Number is 69

Entry Time :- 1705034041

You Space is Reserved Go And Park

Enter Booth Type

Enter 1 for entry

Enter 2 for exit

Enter 0 to Close

2

Enter Details

Enter Session Number

69

your amount is Rs300

Enter Card Number

4321567890121234

Amount 300Rs has been deducted from your account 4321567890121234

Thank you for parking

**COMPARISON OF VDM-SL WITH ITS C++ IMPLEMENTATION:**

|  |  |
| --- | --- |
| **VDM-SL** | **C++ Code** |
| **types**  ParkingSlot = map N to (N \* N \* [Char] \* [Char] \* time \* time \* Z);  **values**  rate:N=20  InitialParkingSlots: ParkingSlot =  { 1 |-> (1, 0, "ali", "7897", time`now, time`now, 0),  2 |-> (2, 0, "", "", time`now, time`now, 0),  3 |-> (3, 0, "", "", time`now, time`now, 0),  4 |-> (4, 0, "", "", time`now, time`now, 0) }  **Description:** VDM-SL: Defines types for ProductID, Product, UserID, CartItem, and ShoppingCart. | time\_t current\_time = time(0);    std::array<std::tuple<int, int, std::string, std::string,time\_t,time\_t,int>, 4> arrayOfTuples = {          std::make\_tuple(1, 1, "ali", "7897", current\_time, current\_time, 0),          std::make\_tuple(2, 0, "", "", current\_time, current\_time, 0),          std::make\_tuple(3, 0, "", "", current\_time, current\_time, 0),          std::make\_tuple(4, 0, "", "", current\_time, current\_time, 0),      }; |

|  |  |
| --- | --- |
| **VDM-SL** | **C++ Code** |
| **state** ParkFlow of  classObj: ParkFlow  slots: ParkingSlot  slotNo: N  sessionNo: N  parkflowBothType: Z  name: [Char]  vn: [Char]  **inv mk**-ParkFlow(SlotNo, SessionNo, ParkflowBothType, name, Vn) ==  ParkflowBothType in set {0, 1, 2} and  Rate>=10 and  SlotNo >= 0 and SessionNo >= 0 and  name <> nil and Vn <> nil;  **init mk**-ParkFlow(name, Vn, SessionNo, , entTime, extTime ) ==  name:=" " and  Vn:=" " and  SessionNo:= 0  **End**  **Description:** VDM-SL: Defines the state of the ParkFlow with slots, slotNo, sessionNo, parkflowBothType, name and Vn. | **class** ParkFlow {  **private**:  int SlotNo, SessionNo;  int ParkflowBothType;  string name;  string Vn;  **public**:  //Constructor  ParkFlow(int SlotNo, int SessionNo, int ParkflowBothType,  std::string name, std::string Vn): SlotNo(SlotNo), SessionNo(SessionNo),  ParkflowBothType(ParkflowBothType), name(name), Vn(Vn){}  };  The VDM-SL mk- ParkFlow is an initialization function; in C++, you might use a constructor for similar purpose |

| **VDM-SL** | **C++ Code** |
| --- | --- |
| **functions**  calculateFee(entryTime: Time, exitTime: Time) : N  **pre** entryTime <= exitTime  **post** RESULT >= 0;  **Description:**  Defines functions calculateFee, having entry and exit time as parameters. Entry time should not be more then exit time and final value should not be less then 0. | int **CalculateFees**(time\_t enterTime,time\_t exitTime){  int rate=20;  return std::difftime(exitTime, enterTime)\*rate;  }  Function CalculateFees, which takes in the entry and exit time, takes difference between them and then applies rate to calculate the Fee. |

| **VDM-SL** | **C++ Code** |
| --- | --- |
| **operations**  **CheckParkflowBothType()**  **rd** ParkflowBothType:N  **pre** true  **post** ParkflowBothType >= 0 and ParkflowBothType <= 2  **DisplaySlots()**  **rd** InitialParkingSlots  **pre** true  **post** true  DO  for slot in set {1, 2, 3, 4} do  status := if (slot, ClassObj.GetStatus(slot)) = (slot, 0)  then "Free"  else "Occupied";  Print("Space ", slot, " Is ", status);  **printSlip()**  **rd** Name  **rd** Vn  **pre** name <> "" and Vn <> ""  **post** true    **printSlip()**  **ext wr** Name  **ext wr** Vn  **pre** true  **post** name <> "" and Vn <> ""  displaySlots();  **GetExitData()**  **rd** SessionNo  **pre** SessionNo >= 0  **post** true  **AuthenticateSession()**;  **CalculateFees(enterTime, exitTime)**  **rd** rate  **pre**: enterTime <= exitTime  **post**: RESULT >= 0  RETURN (exitTime - enterTime) \* rate; | **void CheckParkflowBothType()** {          int type;          cout << "Enter Booth Type" << endl;          cout << "Enter 1 for entry" << endl;          cout << "Enter 2 for exit" << endl;          cout << "Enter 0 to Close" << endl;          cin >> type;          ParkflowBothType = type;      }  **void displaySlots()** {          for (auto& [first, second, name, number,entryTime,exitTime,fees] : arrayOfTuples) {              std::cout << "Space " << first;              if (second == 0) {                  cout << " Is Free" << endl;              }              else {                  cout << " Is Occupied By Mr." << name << " with vehical number" << number << endl;              }          }      }  **void printSlip()** {          for (auto& [first, second, name, number,entryTime,exitTime,fees] : arrayOfTuples) {              if (name==this->name) {                  cout <<"Dear Mr. "<<name<<endl<< "You have Booked parking slot number "<<first<<" for your car with number "<<number << endl;  cout <<"And Your Session Number is "<<second << endl;  cout <<"Entry Time :- "<<entryTime << endl;  cout << "You Space is Reserved Go And Park" << endl;  break;  }  }  } |

|  |  |
| --- | --- |
| **PayAmount(fees)**  **rd** fees;  **pre** fees >= 0  **post** true    **AuthenticateSession**()  **rd** SessionNo;  **rd** exitTime;  **pre** SessionNo >= 0  **post** true  SessionNo := ext;  fees := CalculateFees(enterTime, exitTime);  PayAmount(fees);  **allotSpaces()**  **rd** SessionNo;  **rd** ParkflowBothType;  **rd** Vn;  **rd** name;  **pre** true  **post** SlotNo >= 0 and SlotNo <= 4  printSlip();    **GetParkflowBothType()**  **rd** SessionNo;  **rd** ParkflowBothType;  **pre** true  **post** ParkflowBothType>= 0 and  ParkflowBothType<= 2  **Description:**  VDM-SL: These operations define  the core functionalities of the ParkFlow, such as  display slots, pay amount, allot slots, calculate fee and print slip. | **void GetEntryData()** {  cout << "Enter Details" << endl;  cout << "Enter Name" << endl;  cin >> name;  cout << "Enter Vehicle Number" << endl;  cin >> Vn;  displaySlots();  }  **void GetExitData()** {  cout << "Enter Details" << endl;  cout << "Enter Session Number" << endl;  cin >> SessionNo;  AuthenticateSession();  }  **int CalculateFees(time\_t enterTime,time\_t exitTime)**{  int rate=20;  return std::difftime(exitTime, enterTime)\*rate;  }  **void PayAmount(int fees**){       int number;       cout<<"Enter Card Number"<<endl;       cin>>number;       cout<<"Amount "<<fees<<"Rs has been deducted from your account "<<number<<endl;       cout<<"Thank you for parking"<<endl;   }  **void AuthenticateSession()**{      for (auto& [first, second, name, number,entryTime,exitTime,fees] : arrayOfTuples) {              if (second == SessionNo) {  exitTime = time(0);    fees=CalculateFees(entryTime,time(0));                  cout<<"your amount is Rs"<<fees<<endl;                  PayAmount(fees);                  break;              }          }  }  **void allotSpaces()** {          std::random\_device rd;          std::mt19937 gen(rd());          std::uniform\_int\_distribution<> dist(1, 100);          int ask;          std::cout << "Select Avaliable Space" << endl;          cin >> ask;          for (auto& [first, second, name, number,entryTime,exitTime,fees] : arrayOfTuples) {              if (ask == first && second != 0) {                  cout << " Is Occupied By " << name << endl;                  allotSpaces();              }              if (ask == first && second == 0) {                  second = dist(gen);                  name = this->name;                  number = this->Vn;                  entryTime = time(0);                  printSlip();                  break;              }          }      }  **int GetParkflowBothType()** {          return ParkflowBothType;      }  }; |

# 7. Testing Class:

class ParkFlowTest {

public:

    void testCalculateFees() {

        ParkFlow pf;

        time\_t enterTime = time(0);

        time\_t exitTime = enterTime + 600; // 10 minutes later

        int expectedFees = 20 \* std::difftime(exitTime, enterTime);

        int calculatedFees = pf.CalculateFees(enterTime, exitTime);

        if(calculatedFees == expectedFees){

            cout<<"Fee Test Case Passed"<<endl;

        }else {

            cout << "Fee Test Case  Failed" << endl;

            cout << "Expected: " << expectedFees << ", Actual: " << calculatedFees << endl;

        }

    }

    void testAllotSpaces() {

        ParkFlow pf;

        // booth type ==1

        pf.CheckParkflowBothType();

        // Wnterentry details

        pf.GetEntryData();

        // Run the allotSpaces function

        pf.allotSpaces();

        cout << "Test AllotSpaces Passed" << endl;

    }

    void testAuthenticateSession() {

        ParkFlow pf;

        // Set up an occupied slot

       // Simulate user input for booth type (1 for entry)

pf.CheckParkflowBothType();

// Simulate user input for entry details

pf.GetEntryData();

        // Simulate user input for booth type (2 for exit)

        pf.CheckParkflowBothType();

        // Run the AuthenticateSession function

        pf.AuthenticateSession();

        // Print test result

        cout << "Test AuthenticateSession Passed" << endl;

    }

    void runAllTests() {

        testCalculateFees();

         testAllotSpaces();

        testAuthenticateSession();

    }

};

 // Test Cases

    ParkFlowTest TestObj;

    TestObj.runAllTests();

**Testing Class Output:**

testAllotSpaces()

**OUTPUT**

Enter Booth Type

Enter 1 for entry

Enter 2 for exit

Enter 0 to Close

1

Enter Details

Enter Name

talal zafar

Enter Vehicle Number

Space 1 Is Occupied By Mr.ali with vehical number 7897

Space 2 Is Free

Space 3 Is Free

Space 4 Is Free

Select Available Space

2

Dear Mr. talal

You have Booked parking slot number 2 for your car with number zafar

And Your Session Number is 96

Entry Time :- 1705033031

You Space is Reserved Go And Park

**Test AllotSpaces Passed**

testCalculateFees()

**OUTPUT**

**Fee Test Case Passed**

testAuthenticateSession()

**OUTPUT**

**Fee Test Case Passed**

Enter Booth Type

Enter 1 for entry

Enter 2 for exit

Enter 0 to Close

1

Enter Details

Enter Name

talal

Enter Vehicle Number

fg425

Space 1 Is Occupied By Mr.ali with vehical number 7897

Space 2 Is Free

Space 3 Is Free

Space 4 Is Free

Select Available Space

3

Dear Mr. talal

You have Booked parking slot number 3 for your car with number fg425

And Your Session Number is 49

Entry Time :- 1705033648

You Space is Reserved Go And Park

**Test AllotSpaces Passed**

Enter Booth Type

Enter 1 for entry

Enter 2 for exit

Enter 0 to Close

2

Enter Details

Enter Name

talal

Enter Vehicle Number

fg425

Space 1 Is Occupied By Mr.ali with vehical number 7897

Space 2 Is Free

Space 3 Is Occupied By Mr.talal with vehical number fg425

Space 4 Is Free

Enter Booth Type

Enter 1 for entry

Enter 2 for exit

Enter 0 to Close

2

your amount is Rs 1040

Enter Card Number

4321567890121234

Amount 1040 Rs has been deducted from your account 4321567890121234

Thank you for parking

**Test AuthenticateSession Passed**