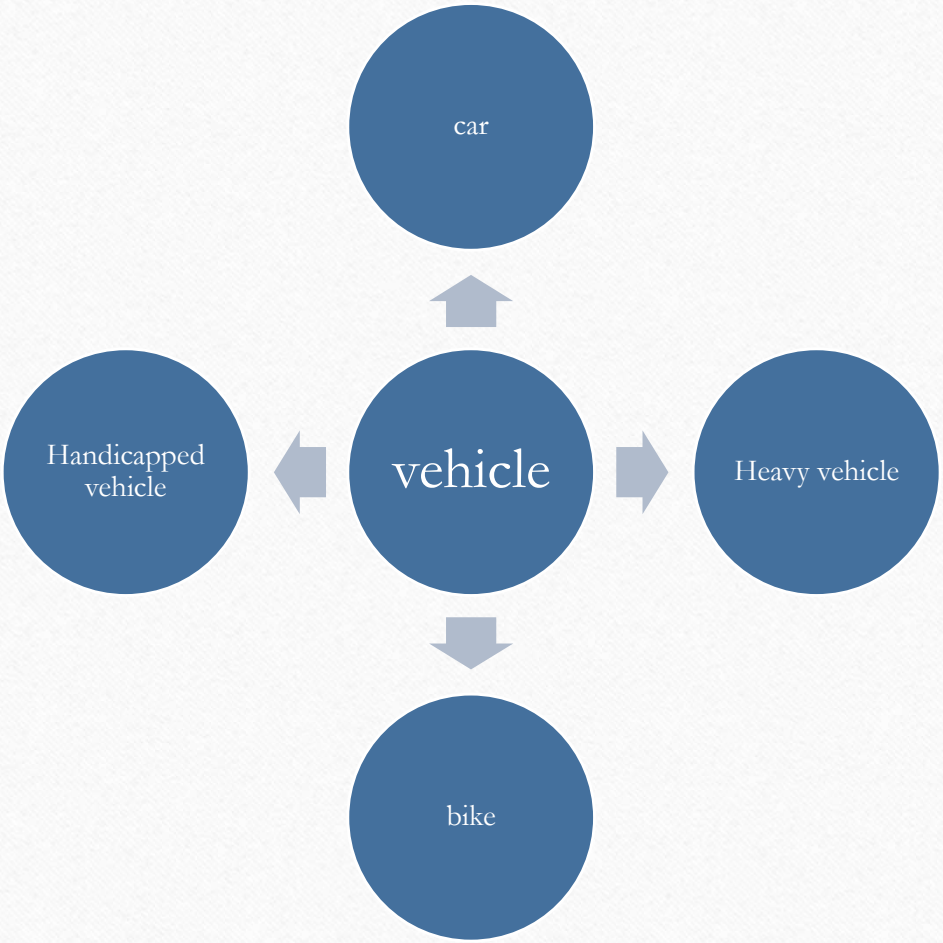
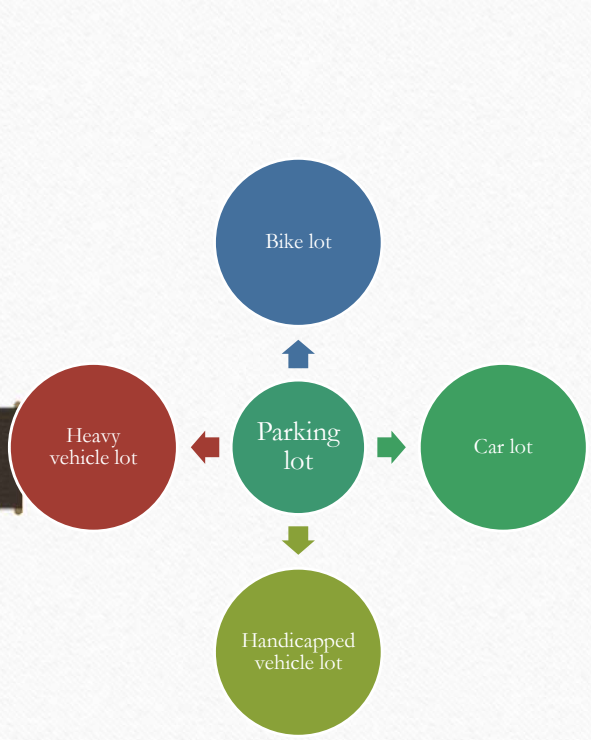
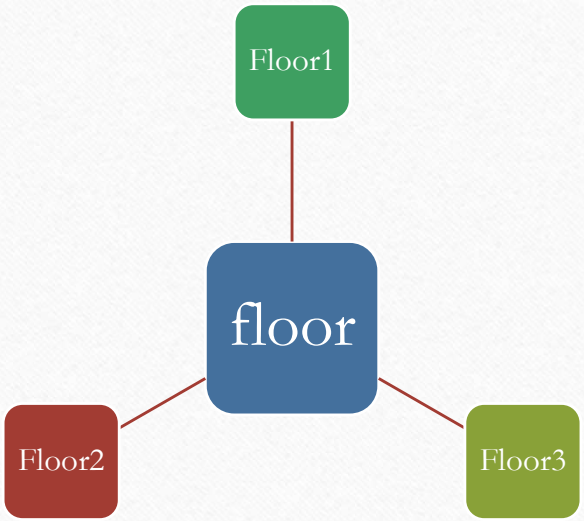
PARKING LOT

Case Study 1

-GROUP 3


car
Handicapped vehicle
vehicle
Heavy vehicle
bike


Bike lot
Heavy vehicle lot
Parking lot
Car lot
Handicapped vehicle lot


Floor1
floor
Floor2
Floor3


Entry terminal
Terminal
Exit terminal


checkout
Withdraw

Functioning of the Code

class:Parking Lot

Parking Lot

- Main function along with the user interface is defined in the parking lot class.
- As per the details the user enter ,different objects related to different classes are created from the main function.
- Several objects of different classes and their corresponding methods are called from this class.
- Parking fee is calculated in the main function and passed to the payment class using constructor.

DISPLAY METHOD

we have used several arrays in order to structure the slot system of the lot.

When the user enters the value of the empty slot allotted to user is initialised to 1.when he leaves the value of the allotted slot is modified to one using the store field

```
Please select from the following
 1)DISPLAY SLOTS
 2) NEW USER
 3) USER EXIT
 4)EXIT INTERFACE
1
SLOT DISPLAY
C0[0] C1[0] C2[0] C3[0] C4[0] C5[0] C6[0] C7[0] C8[0] C9[0]
B0[0] B1[0] B2[0] B3[0] B4[0] B5[0] B6[0] B7[0] B8[0] B9[0]
H0[0] H1[0] H2[0] H3[0] H4[0] H5[0] H6[0] H7[0] H8[0] H9[0]
H.V0[0] H.V1[0] H.V2[0] H.V3[0] H.V4[0] H.V5[0] H.V6[0] H.V7[0]
H.V8[0] H.V9[0]
```

Initially all the slots are initialized to zero value

```
1
SLOT DISPLAY
C0[1] C1[0] C2[0] C3[0] C4[0] C5[0] C6[0] C7[0] C8[0] C9[0]
B0[0] B1[0] B2[0] B3[0] B4[0] B5[0] B6[0] B7[0] B8[0] B9[0]
H0[0] H1[0] H2[0] H3[0] H4[0] H5[0] H6[0] H7[0] H8[0] H9[0]
H.V0[0] H.V1[0] H.V2[0] H.V3[0] H.V4[0] H.V5[0] H.V6[0] H.V7[0]
H.V8[0] H.V9[0]
```

When a car user enters the nearest slot is allotted to him and the its value is initialized to 1

```
C0[1] C1[0] C2[0] C3[0] C4[0] C5[0] C6[0] C7[0] C8[0] C9[0]
B0[1] B1[0] B2[0] B3[0] B4[0] B5[0] B6[0] B7[0] B8[0] B9[0]
H0[0] H1[0] H2[0] H3[0] H4[0] H5[0] H6[0] H7[0] H8[0] H9[0]
H.V0[0] H.V1[0] H.V2[0] H.V3[0] H.V4[0] H.V5[0] H.V6[0] H.V7[0]
H.V8[0] H.V9[0]
```

Slot display when a new bike user enters
(no user has left until now)

```
C0[1] C1[0] C2[0] C3[0] C4[0] C5[0] C6[0] C7[0] C8[0] C9[0]
B0[1] B1[1] B2[0] B3[0] B4[0] B5[0] B6[0] B7[0] B8[0] B9[0]
H0[0] H1[0] H2[0] H3[0] H4[0] H5[0] H6[0] H7[0] H8[0] H9[0]
H.V0[0] H.V1[0] H.V2[0] H.V3[0] H.V4[0] H.V5[0] H.V6[0] H.V7[0]
H.V8[0] H.V9[0]
```

A new bike user has entered(no one left until now)

```
C0[1] C1[0] C2[0] C3[0] C4[0] C5[0] C6[0] C7[0] C8[0] C9[0]
B0[0] B1[1] B2[0] B3[0] B4[0] B5[0] B6[0] B7[0] B8[0] B9[0]
H0[0] H1[0] H2[0] H3[0] H4[0] H5[0] H6[0] H7[0] H8[0] H9[0]
H.V0[0] H.V1[0] H.V2[0] H.V3[0] H.V4[0] H.V5[0] H.V6[0] H.V7[0]
H.V8[0] H.V9[0]
```

This is the display slot when a bike user who entered after the first user has left.(the array is modified to zero)

DIFFERENT SUBCLASSES OF PARKING LOT

Car lot

- car lot has an array of slots as fields
- it has a getlot() method to return the nearest slot to the user as it is the first slot that is empty.
- The Boolean value of the returned slot will be set to 1 in the main function to indicate that the slot is filled.

Bike lot

- bike lot has an array of slots as fields
- it has a getlot() method to return the nearest slot to the user as it is the first slot that is empty.
- The Boolean value of the returned slot will be set to 1 in the main function to indicate that the slot is filled

Heavy vehicle lot

- heavyvehiclelot has an array of slots as fields.
- it has a getlot() method to return the nearest slot to the user as it is the first slot that is empty.
- The Boolean value of the returned slot will be set to 1 in the main function to indicate that the slot is filled

Handicapped lot

It has the same functionality as that of the previous subclasses

We can add more methods or features to handicapped lot as per our convenience.

Class :Vehicle

Vehicle class has two private fields vehicle id,vehicle type.As these private fields cannot be initialized directly outside the vehicle class a constructor is used to intitialize these fields.so as soon as the object is created,the parameters are passed to the constructor.

It has two methods 'getvehicleid' and 'getvehicletype' to return the values of these fields.

This vehicle class is extended to four subclasses which inherit the functionality of the vehicle class.

CAR

- Class 'car'
- Class car has a constructor which forms constructor chaining with the constructor present in the super class to initialize the corresponding fields

BIKE

- Class 'bike'
- Class bike has a constructor which forms constructor chaining with the constructor present in the super class to initialize the corresponding fields

HANDICAPPED

- Class 'handicapped'
- Class handicapped has a constructor which forms constructor chaining with the constructor present in the super class to initialize the corresponding fields

- Class 'heavy vehicle'
- Its functionality is same as that of the previous subclasses.

CLASS: TERMINAL

The terminal class has two private strings hours and minutes. These fields represent entry and exit time in accordance with the parameters passed from the respective subclasses. these private fields are initialised using the constructor.
It has the gettime() method to display the time.

ENTRY TERMINAL

- This class is mostly used to inititalize the fields of terminal class through constructor chaining.

EXIT TERMINAL

- The fields of terminal represent the exit timings when the parameters are passed from the exit terminal.
- It has withdraw() method to display the choices available for the user to withdraw his vehicle.

Abstract class withdraw:

It has intime hr,min and outtime hr,min as fields and these fields are used in the subclass 'withdraw' to calculate the parking charges of the user.

We cannot create objects of this class so to initialize the fields we used the concept of constructor-chaining.

This class is extended to 'checkout' subclass.

Check out subclass:

This class is mostly used to display the difference in entry and exit time to the user.several options are displayed to the user for cash payment.

Class payment:

The functionality in this class mostly involves displaying different options to the user.
We further divided it on the basis of a different mode of payment. we have used a 'while' loop in Order to ensure that the user chooses a valid option, even if he chooses an option other than the options provided, he gets the chance to enter his choice again.
The while loop will run as long as the user enters valid input.

Why we have used classes, interfaces and abstract class ?

- In our parking lot ,classes are used as a template for similar kind of objects.
- We have used different classes and those classes are extended to their subclasses so that whenever we are interested in adding new features or functionalities we can add or extend so that the other classes and the links between the classes are undisturbed.
- We used 'withdraw' as an abstract class because it has no functionality but when the user chooses not to checkout his vehicle, then there is nothing that can be done,else if he wants to check out his vehicle then an object of its subclass 'checkout' is created and user was given choices to proceed to his payment.

As per the design of our parking lot ,using interfaces wasn't needed.

How oops concept supported our decision?
Have you used any other concepts from Java?

- For creating a parking lot , the object-oriented paradigm assisted us to be more closer to the real world.
- Various concepts of oops like inheritance, encapsulation, and abstraction supported us in creating a parking lot where there is a communication between the objects.
- We were in need of making message passing between different objects which was done through several concepts of oops.
- Using the concept of encapsulation ,we were able to integrate the methods and the fields of different classes.
- Using the concept of inheritance, we were able to connect different classes so that all the subclasses can carry the functionality of the parent class and different methods can be defined in the subclasses independent of each other.
- Though there is a complex code in the main function the user interface is still simple as we used the concept of abstraction in order to hide the implementation details.

By using the concept of oops ,we have improved the reusability of code.

Even if are interested in adding more features to the parking lot ,we can easily do it without disturbing the core structure of the code because of availability of more classes.

TEAM MEMBERS:

1.CS21B030

2.CS21B032

3.CS21B058

4.CS21B036

5.CS21B020