**NATIONAL ENGINEERING COLLEGE, K.R.NAGAR, KOVILPATTI - 628 503**

(An Autonomous Institution – Affiliated to Anna University, Chennai) **Department of Computer Science and Engineering**

23CS24C – OBJECT ORIENTED PROGRAMMING IN C++

Experiential Learning

A Report on

TOUR MANAGEMENT SYSTEM

Submitted By: Maha Manisha M   Course Instructor

Mrs A Lincy

Team Member:Anusuyadevi N Assistant Professor

Dharshini K (Department of CSE)

# SCENARIO

Design and implement a Tour Booking Management System application with the following features:

1. Create a class Tour that represents a tour package available for booking. It should have attributes like tourID, destination, duration, price, departureDate, returnDate, and availability. Implement constructors, accessors, and mutators as needed.
2. Create a class Customer to represent individuals booking tours. It should have attributes like customerID, name, age, gender, address, email, and phone Number. Implement constructors, accessors, and mutators as needed.
3. Implement a class Booking to manage tour bookings. It should include attributes like bookingID, customerID, tourID, bookingDate, numAdults, numChildren, and totalCost. Implement functions for creating new bookings, updating booking details, canceling bookings, and generating booking reports.
4. A "Package" class could represent a bundled tour package that includes multiple individual tours. It could inherit from the "Tour" class and add additional attributes and methods specific to a package, such as a list of included tours or special discounts.
5. Develop a class TourOperator to manage tour packages and bookings. Implement functions for adding new tours, updating tour details, handling booking requests, tracking booking history, and generating reports on available tours and booking trends.
6. Implement exception handling to manage runtime errors. For example, handle exceptions when a tour is fully booked, a booking request is invalid, or if the customer details are incomplete.
7. Develop a file handling system to store and retrieve tour, customer, and booking informationsecurely. Implement encryption and decryption mechanisms for data storage. Create functions for saving the data to a file and loading data from a file, ensuring data remains encrypted.

Ensure that your program demonstrates the effective use of C++ features, adheres to object- oriented programming principles, and meets the specified requirements. You can add your own additional class, methods, Templates wherever needed. Your code should be well- structured, maintainable, capable of handling real-world scenarios and error-tolerant.

# INTRODUCTION

The Tour Booking Management System is a software application designed to help tour operators manage tour packages and bookings easily. It allows operators to create and manage tour details, handle customer bookings, and keep track of booking history. Customers can book tours, and operators can update or cancel bookings as needed. The system includes features for managing customer information and generating reports on tours and bookings.It also ensures secure storage of data with encryption and handles errors gracefully. Built with object-oriented principles, the system is easy to maintain and can effectively manage real-world tour booking scenarios.

**PROBLEM STATEMENT**

• To create a tour management application using C++ that have

features to book tour, update tour or cancel tour.

• To save the tour details of both customer and reservation details in a

separate files.

• To create an error free application that is capable to handle real life

scenario.

# OBJECTIVES

1.**Easy Booking Process:** Simplify the process of creating, updating, and canceling tour booking s to ensure accuracy and user-friendliness.

**2.Customer Information Management**: Maintain detailed and organized records of customers, including their personal information and booking history.

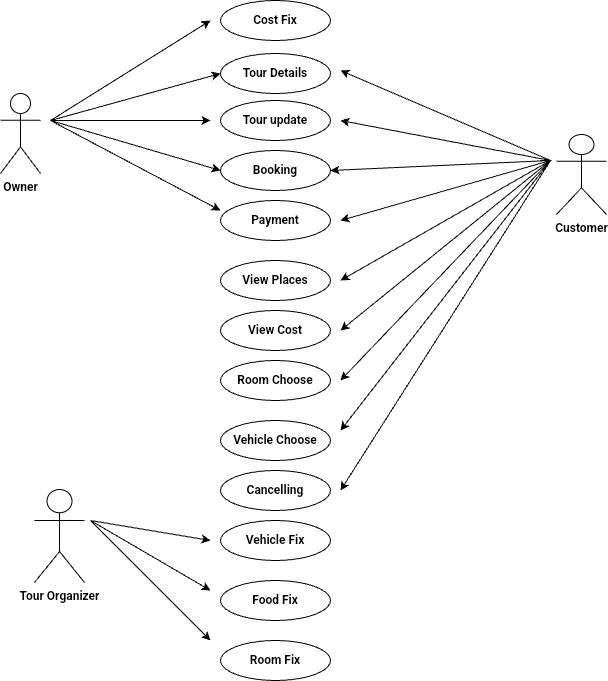
**3.Secure Data Handling:** Implement robust file handling with encryption to securely store and retrieve tour, customer, and booking information.

**4.Comprehensive Tour Management:** Allow for the addition, updating, and removal of tour packages with all necessary details.

**5.Advanced Reporting and Tracking**: Generate insightful reports on available tours, booking trends, and booking history to aid in strategic decision-making.

**6.User-Friendly Interface**: Develop an intuitive user interface for easy navigation and interaction with the system.

**USE CASE DIAGRAM:**



A Use Case Diagram is a type of Unified Modeling Language (UML) diagram that represents the interaction between actors and a system under consideration to accomplish specific goals. It provides a high-level view of the system’s functionality by illustrating the various ways users can interact with it.

**ACTORS:**

Actors are external entities that interact with the system. These can include users, other systems, or hardware devices. In the context of Use Case Diagram, actors initiate use cases and receive the outcomes.

**USE CASES:**

Use cases represent specific things your system do. Use cases are represented by ovals.

**SYSTEM BOUNDARY:**

The system boundary is a visual representation of the scope or limits of the system you are modeling. It defines what is inside the system and what is outside. The system boundary is typically represented by rectangular box that surrounds all the use cases of the system.

**ASSOCIATION RELATIONSHIP:**

It represents a communication or interaction between an actor and a use case. It is depicted by a line connecting the actor to the use case.

**INCLUDE RELATIONSHIP:**

It indicates that a use case includes the functionality of another use case. It is denoted by dashed arrow pointing from the including use case to the included use case. This relationship promotes modular and reusable design.

**EXTEND RELATIONSHIP:**

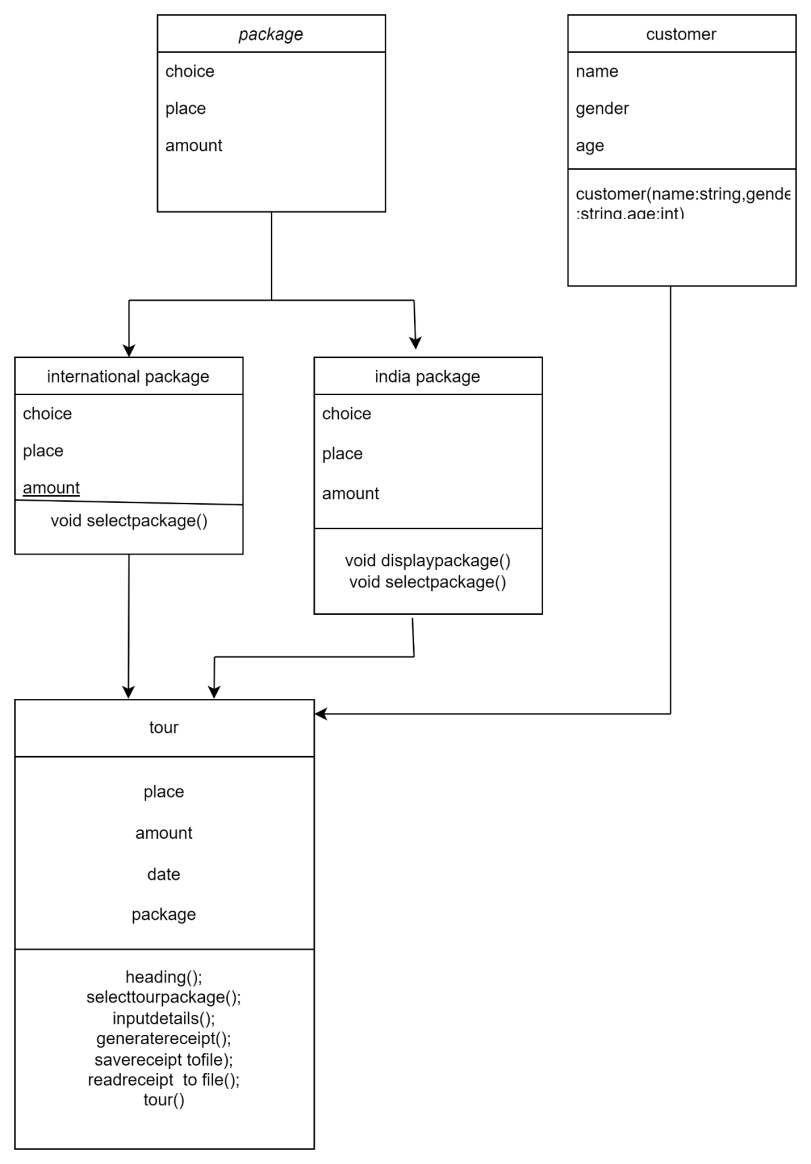
It illustrates that a use case can be extended by another use case under specific condition. It is represented by a dashed arrow with keyword “extend”. This relationship is useful for handling optional or exceptional behavior.

Owner - Manages tour details, updates, cost fixing, and oversees bookings.

Customer - Books tours, makes payments, views places and costs, chooses rooms and vehicles, and can cancel bookings

Tour Organizer - Manages logistical aspects like fixing vehicles, food, and rooms for the tours.

**CLASS DIAGRAM**



Class diagrams are a type of UML diagram used in software engineering to visually represent the structure and relationships of classes within a system. In these diagrams, classes are depicted as boxes, each containing three compartments for the class name, attributes, and methods. Lines connecting classes illustrate associations, showing relationships such as one-to-one or one-to-many.

**CLASS NAME:**

The name of the class is typically written in the top compartment of the class box and is centered and bold.

**CLASS ATTRIBUTES:**

Attributes are shown in the second partition. The attribute type is shown after the colon. Attributes map onto member variables (data members) in code.

**CLASS OPERATIONS (METHODS):**

Operations are shown in the third partition. They are services the class provides. The return type of a method is shown after the colon at the end of the method signature. The return type of method parameters is shown after the colon following the parameter name.Operations map onto class methods in code.

**1.Package Class:**

* Attributes: choice, place, amount
* Subclasses: international package and india package

● Methods:

void selectpackage()

1. **Customer Class:** 
   * Attributes: name, gender, age
   * Constructor: customer(name: string, gender: string, age: int)
2. **International Package Class:**

● Inherits from `package`

● Methods:

void selectpackage()

**4.India Package Class:**

● Inherits from package

● Methods:

void displaypackage() void selectpackage()`

**5.Tour Class:**

* + Attributes: place, amount, date, package

● Methods:

heading() selecttourpackage() inputdetails() generatereceipt() savereceipt to file() readreceipt to file() tour()

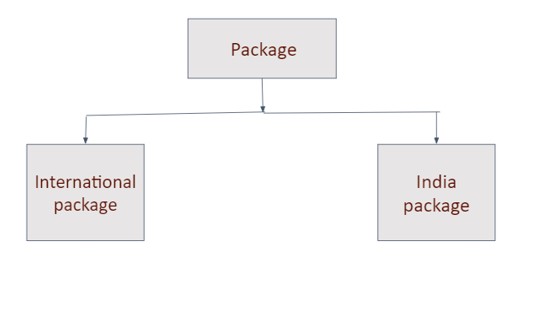
The customer class connects to the `tour` class, suggesting that a customer can book a tour. The international package and india package are specific types of packages that inherit common attributes and methods from the package class.

**OBJECT ORIENTED CONCEPTS**

**INHERITANCE:**

**HIERARCHICAL INHERITANCE:**

Hierarchical inheritance is when a single parent class is inherited by multiple child class .



Here in our code package is parent class and international package ,india package are child class

Hierarchical inheritance in the code helps manage different tour packages by using a common base class called `Package`. This base class has functions that all package types must implement.

The `InternationalPackage` and `IndiaPackage` classes inherit from `Package` and provide specific details for international and domestic tours. This makes the code more reusable, organized, and easier to manage, allowing the `Tour` class to handle different packages smoothly without needing to know their specific details.

**POLYMORPHISM**

Run-time Polymorphism occurs when a function call to a derived class object is resolved at run time. This is typically achieved using function overriding and pointers or references to the base class. In C++, run-time polymorphism is implemented using virtual functions.

The code uses \*\*runtime polymorphism\*\*. This is evident from the use of virtual functions in the base class `Package`, which are overridden in the derived classes `InternationalPackage` and

`IndiaPackage`. The specific implementation of `displayPackages` and `selectPackage` is determined at runtime based on the type of the object pointed to by the base class pointer `package`.

Customer Class:

● Constructor: Customer(string name, string gender, int age)

Package Class (Abstract Base Class):

Virtual Method: displayPackages(),selectPackage()

Override Method: displayPackages(),selectPackage()

IndiaPackage Class (Derived from Package)**:**

Override Method: displayPackages(),selectPackage()

Tour Class:

heading()

selectTourPackage() inputDetails() generateReceipt() saveReceiptToFile() readReceiptFromFile()

**FILES CONCEPT**

**INTRODUCTION**:

Files allows to read from and write to files. In our code, we used file handling to save and retrieve the tour receipt information. The operations are performed using the <fstream> library in C++, which provides file stream classes: ifstream for reading from files and ofstream for writing to files.

**INPUT FILE:**

We have create an input file named receipt.txt that stores the following details

1.Passangers details

2.Selected type

3.package

4.Departure date

5.Return Date

4.Total amount

**OUTPUT FILE**:

We have create an output file named receipt.txt that display the following details which are saved in the input file

1.Passangers details

2.Selected type

3.package

4.Departure date

5.Return Date

4.Total amount

**TEMPLATE**

In this scenario, the Tour Booking Management System involves handlingvarious types of tours, customers, and bookings. Implementing templates in C++can offer several benefits in such a system, particularly for improving codereusability, type safety, and flexibility.The system supports different types of tours: Indian, International, and Adventure. Each type of tour has specific options and details.Using a template class , we can handle different data types like Customer andTour in a generic manner. This reduces code duplication and increasesmaintainability.

**CODING:**

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

using namespace std;

class Customer

{

public:

string name;

string gender;

int age;

Customer(string name, string gender, int age) : name(name), gender(gender), age(age) {}

};

class Package

{

public:

virtual void displayPackages() = 0;

virtual void selectPackage(int choice, string &place, int &amount) = 0;

virtual ~Package() = default;

};

class InternationalPackage : public Package

{

public:

void displayPackages()

{

cout << "\t\t\t\t1. England Tour Packages 6 Days 7 Nights (28880/-)\n";

cout << "\t\t\t\t2. Thailand Tour Packages 5 Days 4 Nights (15500/-)\n";

cout << "\t\t\t\t3. New York Tour Packages 11 Days 10 Nights (567800/-)\n";

}

void selectPackage(int choice, string &place, int &amount)

{

switch (choice)

{

case 1:

place = "England Tour";

amount = 28880;

break;

case 2:

place = "Thailand Tour";

amount = 15500;

break;

case 3:

place = "New York Tour";

amount = 567800;

break;

default:

cout<<"Invalid choice for international package";

}

}

};

class IndiaPackage : public Package

{

public:

void displayPackages()

{

cout << "\t\t\t\t1. Shimla Tour Packages 6 Days 7 Nights (18880/-)\n";

cout << "\t\t\t\t2. Kashmir Tour Packages 5 Days 4 Nights (35500/-)\n";

cout << "\t\t\t\t3. Kolkata Tour Packages 11 Days 10 Nights (10000/-)\n";

}

void selectPackage(int choice, string &place, int &amount)

{

switch (choice)

{

case 1:

place = "Shimla Tour";

amount = 18880;

break;

case 2:

place = "Kashmir Tour";

amount = 35500;

break;

case 3:

place = "Kolkata Tour";

amount = 10000;

break;

default:

cout<<"Invalid choice for India package";

}

}

};

class Tour

{

private:

vector<Customer> customers;

string type;

string place;

int amount;

string departuredate,returndate;

Package \*package;

public:

~Tour() { delete package;

}

void heading()

{

cout << "\t\t\t\t\*Tour Management System\*\n";

}

void selectTourPackage()

{

int choice;

heading();

cout << "\t\t\t\t1. International Tour Packages\n";

cout << "\t\t\t\t2. India Tour Packages\n";

cout << "\t\t\t\tEnter Choice: ";

while (!(cin >> choice) || (choice != 1 && choice != 2))

{

cout << "Enter Right Choice: ";

}

switch (choice)

{

case 1:

type = "International Tour Package";

package = new InternationalPackage();

break;

case 2:

type = "India Tour Package";

package = new IndiaPackage();

break;

}

package->displayPackages();

cout << "\t\t\t\tEnter Choice: ";

while (!(cin >> choice) || (choice < 1 || choice > 3))

{

cout << "Enter Correct Choice: ";

}

package->selectPackage(choice, place, amount);

}

void inputDetails()

{

heading();

int numPassengers;

cout << "\t\t\t\tEnter Number Of Passengers: ";

while (!(cin >> numPassengers) || numPassengers <= 0)

{

cout << "Enter a valid number of passengers: ";

}

cout << "\t\t\t\tEnter departure Date (DD/MM/YY): ";

cin >> departuredate;

for (int i = 0; i < numPassengers; ++i)

{

string name, gender;

int age;

cout << "\t\t\t\tEnter The " << i + 1 << "th Passenger Name: ";

cin >> name;

cout << "\t\t\t\tEnter The " << i + 1 << "th Passenger Gender: ";

cin >> gender;

cout << "\t\t\t\tEnter The " << i + 1 << "th Passenger Age: ";

while (!(cin >> age) || age <= 0)

{

cout << "Enter a valid age: ";

}

customers.push\_back(Customer(name, gender, age));

}

}

void generateReceipt()

{

heading();

cout << "\n\t\t\t\t\*Take Screenshot For Further Use\*\n";

for (size\_t i = 0; i < customers.size(); ++i)

{

cout << "\t\t\t\t" << i + 1 << "st Passenger Name: " << customers[i].name << endl;

cout << "\t\t\t\t" << i + 1 << "st Passenger Gender: " << customers[i].gender << endl;

cout << "\t\t\t\t" << i + 1 << "st Passenger Age: " << customers[i].age << endl << endl;

}

cout << "\t\t\t\tSelected Type: " << type << endl;

cout << "\t\t\t\tPackage: " << place << endl;

cout << "\t\t\t\tDepartureDate: " << departuredate << endl;

int totalAmount = amount \* customers.size();

cout << "\t\t\t\tTotal Amount: " << totalAmount << endl;

cout << "\n\t\t\t\t\*Thank You For Registration\*";

saveReceiptToFile();

}

void saveReceiptToFile()

{

ofstream outFile("receipt.txt");

outFile << "\t\t\t\t\*Tour Management System\*\n";

outFile << "\n\t\t\t\t\*Take Screenshot For Further Use\*\n";

for (size\_t i = 0; i < customers.size(); ++i)

{

outFile << "\t\t\t\t" << i + 1 << "st Passenger Name: " << customers[i].name << endl;

outFile << "\t\t\t\t" << i + 1 << "st Passenger Gender: " << customers[i].gender << endl;

outFile << "\t\t\t\t" << i + 1 << "st Passenger Age: " << customers[i].age << endl << endl;

}

outFile << "\t\t\t\tSelected Type: " << type << endl;

outFile << "\t\t\t\tPackage: " << place << endl;

outFile << "\t\t\t\tdepartureDate: " << departuredate << endl;

int totalAmount = amount \* customers.size();

outFile << "\t\t\t\tTotal Amount: " << totalAmount << endl;

outFile << "\n\t\t\t\t\*Thank You For Registration\*";

outFile.close();

}

void readReceiptFromFile()

{

ifstream inFile("receipt.txt");

if (!inFile)

{

cerr << "Error: File could not be opened" << endl;

return;

}

string line;

while (getline(inFile, line))

{

cout << line << endl;

}

inFile.close();

}

};

int main()

{

Tour tour;

tour.selectTourPackage();

tour.inputDetails();

tour.generateReceipt();

cout << "\nReading Receipt from file...\n";

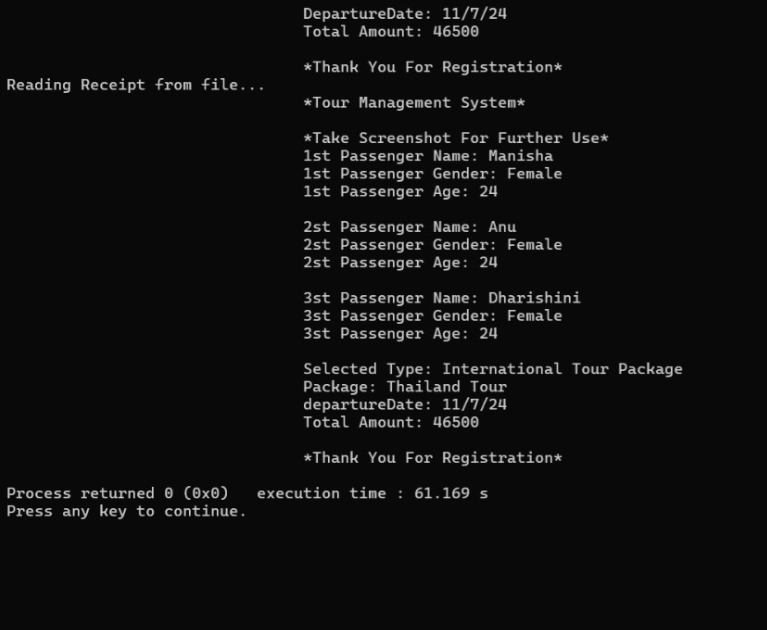
tour.readReceiptFromFile();

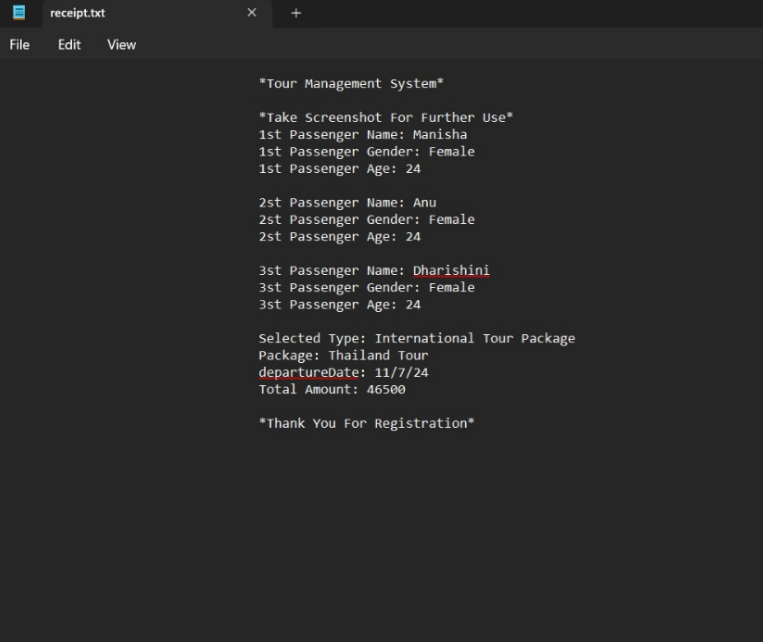
return 0;

}

**OUTPUT:**

****

****

****

**CONCLUSION**

The use of polymorphism allows the system to handle various tour types in a consistent manner. This System’s design ensures efficient management of different tour offerings, streamlines the booking process and allows for easy expansion. This design ensures that the system can grow and adapt over time, providing a strong foundations for future enhancements.

**REFERENCES**

1) Use Case Diagram - https://www.geeksforgeeks.org/use-case-diagram/

2) Class Diagram -https://www.geeksforgeeks.org/unified-modeling-language-uml-classdiagrams/

3) OOPS Concepts - <https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/>

4) File Handling - https://www.javatpoint.com/cpp-files-and-streams

5) Exception - https://www.w3schools.com/cpp/cpp\_exceptions.asp

6) A Tour of C++ - https://www.goodreads.com/book/show/18197265-a-tour-of-c