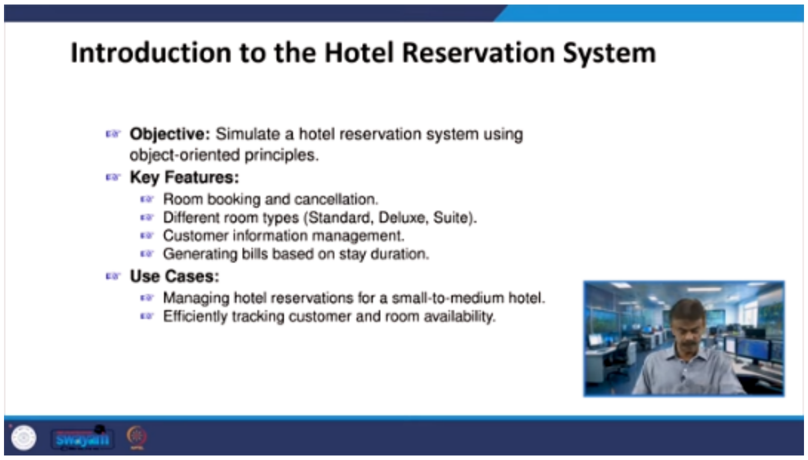
FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

Lecture57

Lecture 57: Case Study - Hotel reservation System C++

Welcome to lecture number 57. We are going to see case studies in object-oriented programming. So, in this case study, we are going to see a hotel reservation system. The main objective of this study is to simulate a hotel reservation system using object-oriented principles, right. So, whatever we have studied, some of the concepts we are going to use.



Introduction to the Hotel Reservation System

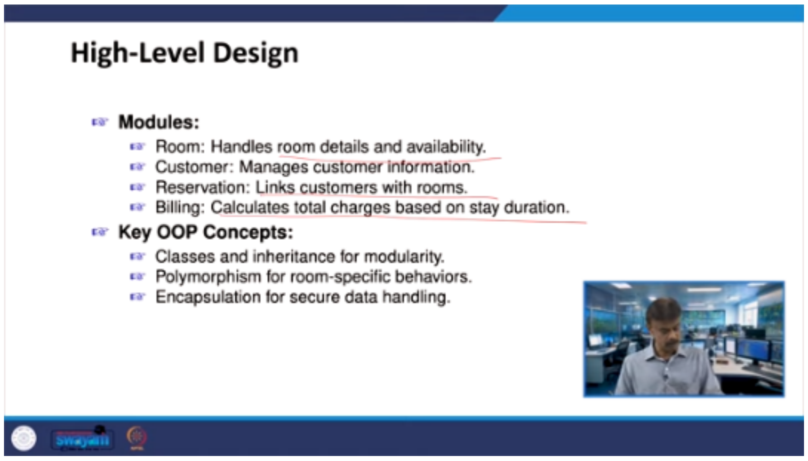
- **Objective:** Simulate a hotel reservation system using object-oriented principles.
- **Key Features:**
  - Room booking and cancellation.
  - Different room types (Standard, Deluxe, Suite).
  - Customer information management.
  - Generating bills based on stay duration.
- **Use Cases:**
  - Managing hotel reservations for a small-to-medium hotel.
  - Efficiently tracking customer and room availability.

The key features are room booking and cancellation. So, if you have a hotel room, we have to write code for booking and cancellation. And there will be different types of rooms, right? You may have standard, deluxe, and suite, so accordingly, the rate will change. Maybe the tax will change, right? So, we have to do those calculations and customer So, regarding the customer, we have to maintain the database and the information management system for the customers. Then, generating bills based on the stay duration—how many nights they are staying—based on that, we have to generate the bills.

So, use cases here: we have to manage the hotel reservation for a small to medium hotel, right? So, we have to do the hotel reservation for a small to medium hotel. Another use case is efficiently tracking customers and room availability, right. So, you have to track the customer, and also we have to think about room availability. So, these are all the main objectives, key features, and use cases for the hotel reservation system.

So, the modules that we are going to use are as follows. So, we will have a room. The first module is the room, which handles room details and the availability of the rooms. Then, customer—we have to manage the customer information. So, this module will manage the customer information.

Then, reservation—so this will link customers with rooms—and then billing. So, this will calculate total charges based on the stay duration. So, now we have key OOP concepts. Right. So, here to do this, what are all the OOP or object-oriented programming concepts that we are going to use, right?



High-Level Design

- **Modules:**
  - Room: Handles room details and availability.
  - Customer: Manages customer information.
  - Reservation: Links customers with rooms.
  - Billing: Calculates total charges based on stay duration.
- **Key OOP Concepts:**
  - Classes and inheritance for modularity.
  - Polymorphism for room-specific behaviors.
  - Encapsulation for secure data handling.

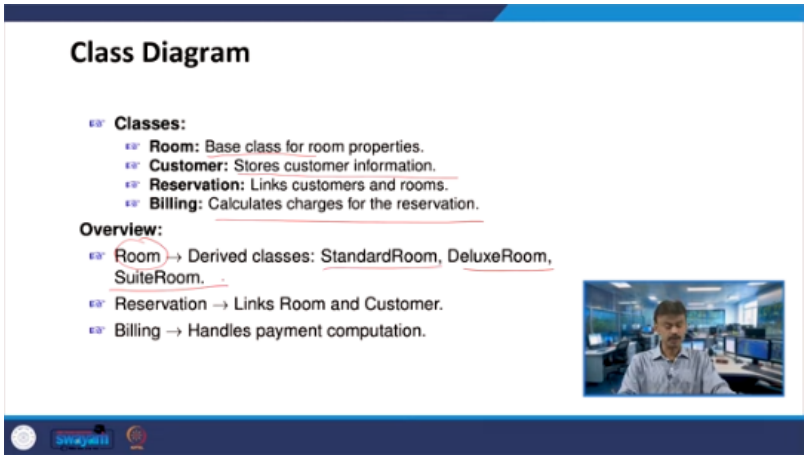
So, we use classes and inheritance for modularity, right? So, polymorphism for room-specific behaviors, then encapsulation for secure data handling. So, classes and inheritance, polymorphism, and encapsulation—we are going to use these in this particular case study. So, now what about the class diagram, like what we have done in the previous problem?

So, the classes are room right. So, we keep it as a base class right all the room properties. So, whatever member functions or whatever the member data that it is being used or in the case of C++ member data and the methods right. So, we will create a base class the class is room and then you will have customer Right.

So which stores all the information about the customer. Then we have a reservation class that links customers and rooms. Then we have the billings. Right. That means that is calculating the charges for the reservation.

Right. So these are all the classes that we are going to use for this particular problem. And we said room. Right. That is the base class.

So, if the base class is the room, you may ask the question: what about the derived classes? So, the derived classes we are going to have are standard room, deluxe room, and suite room, right? So, these are all the derived classes. Then the reservation, which is going to link the room and the customer, right? So, the reservation is what links the room and the customer.



Class Diagram

- **Classes:**
  - **Room:** Base class for room properties.
  - **Customer:** Stores customer information.
  - **Reservation:** Links customers and rooms.
  - **Billing:** Calculates charges for the reservation.

Overview:

- Room → Derived classes: StandardRoom, DeluxeRoom, SuiteRoom.
- Reservation → Links Room and Customer.
- Billing → Handles payment computation.

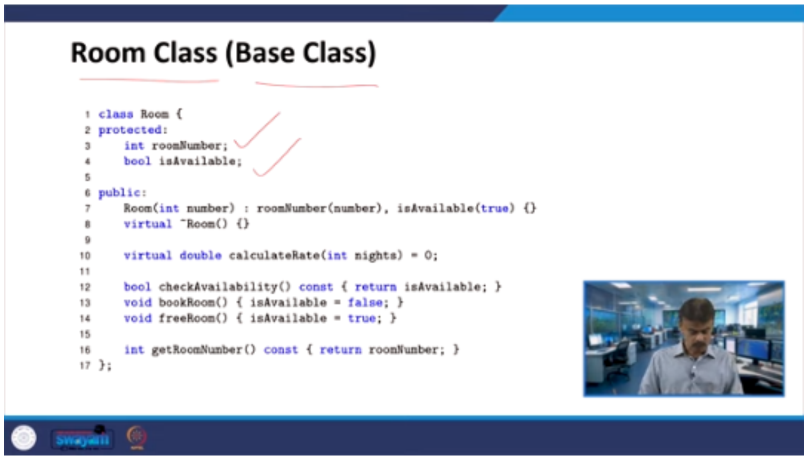
So, when the customer may prefer a standard room, or the customer may prefer the deluxe room or the suite room, right? So, in that case, the reservation is what links the customers and the rooms. So, then you have billing, right? So, it handles payment computation, right? So, billing is

So, it handles payment computation. Okay. So, let us go to the program. Right. As I said.

Right. So, these are all the classes that we are going to use. So, the first class is a room, which is a base class. Right. The room class is the base class.

So, the room has the protected member data: room number and is available. Right. So, room and is available. Right. So, here we are going to see with the help of C++.

Right. So, that means I have to talk about the member data and member functions, and this is your constructor, right. So, under this constructor, this is a one-argument constructor where, right. So, you have the parameter number, right, whose data type is integer, and the number is assigned to room number, right, in this constructor, and is available is true, right, is available is true. So, this is what you are doing in the constructor.



Room Class (Base Class)

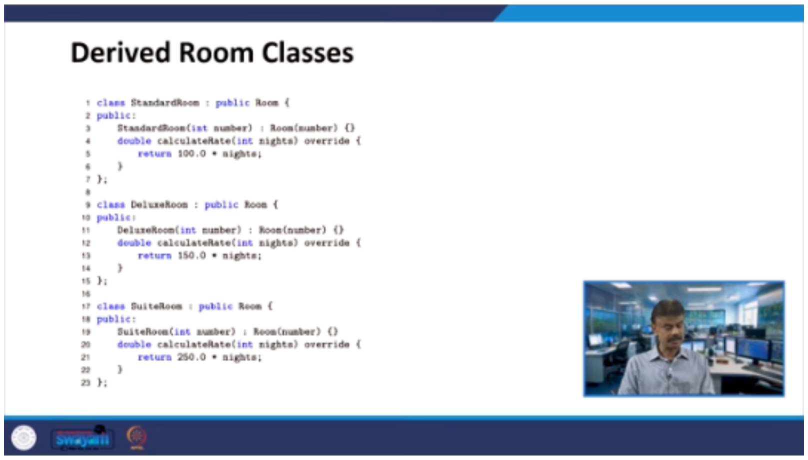
```
1  class Room {
2  protected:
3      int roomNumber;
4      bool isAvailable;
5
6  public:
7      Room(int number) : roomNumber(number), isAvailable(true) {}
8      virtual ~Room() {}
9
10     virtual double calculateRate(int nights) = 0;
11
12     bool checkAvailability() const { return isAvailable; }
13     void bookRoom() { isAvailable = false; }
14     void freeRoom() { isAvailable = true; }
15
16     int getRoomNumber() const { return roomNumber; }
17 };
```

And in the virtual destructor here, you have it right. So, that will deallocate the memory for whatever objects you are creating. Once the job is over, it will be deallocated, right? So, here you have calculateRate, right, which is a pure virtual function. So, that means in the inheritance, right, or the class that is going to inherit Room, we should have that calculateRate. Rate, and here you have the number of nights that the customer is staying, which is an integer variable.

So, initially, I set this function when I set it equal to 0. We know that it is a virtual function. So, the usage of a virtual function, right? Anyway, line number 8 itself we have seen. So, these are all the virtual destructor, and here you have the virtual function, right? So, that means it will be inherited further, right? Or in the further derived class, it will be inherited. Now, there are a few member functions: checkAvailability, right? It is a boolean data type, or I mean to say, boolean return type. So, this is returning isAvailable, right? If isAvailable is true, return 1; if it is not, return false, OK? And bookRoom, another member function called bookRoom. So, bookRoom by default sets isAvailable equal to false.

Right, by default, isAvailable equal to false. Yes, if somebody is booking a room, isAvailable will go to false, right? And freeRoom: suppose somebody is vacating the room, isAvailable equal to true. So, once they are vacating the room, the room is available, so isAvailable equal to true. So, this is a boolean, right? So, it will change accordingly. So, by default, it is true. And then, if somebody is booking the room and they are staying in the room, the room is not free, right? So, that means isAvailable will be equal to false, right? If it is freeRoom, isAvailable equal to true. And then, there is one more function, getRoomNumber, right?

That is a function whose return type is integer and this will return room number. So this is your class room. Up to line number 17, you have class room, right? That is a base class. Now, you will have set of derived classes, all right.



Derived Room Classes

```cpp
class StandardRoom : public Room {
public:
    StandardRoom(int number) : Room(number) {}
    double calculateRate(int nights) override {
        return 100.0 * nights;
    }
};

class DeluxeRoom : public Room {
public:
    DeluxeRoom(int number) : Room(number) {}
    double calculateRate(int nights) override {
        return 150.0 * nights;
    }
};

class SuiteRoom : public Room {
public:
    SuiteRoom(int number) : Room(number) {}
    double calculateRate(int nights) override {
        return 250.0 * nights;
    }
};
```

So, class standard room which is publicly inheriting room, right. So, room is a base class and in fact, it has a pure virtual function called it as abstract, not abstract class because we are defining some of the, right, member functions. It contains the abstract function, I mean to say pure virtual function, right. So, standard room is inheriting. I will just put sr, it is understood, right.

standard room right it is iterating room and you have a constructor standard room passing number right and number is assigned to room number is assigned to room you have the calculate rate right. So, this calculate rate here you go right this is in the pure virtual function this we have to define and it is define in the derived class right calculate rate in nights all right nights is a variable whose data type is integer in fact argument all right so nights parameter whose data type is integer so it will be overridden all right. So, that means it will override this particular virtual function in line number 10 whenever the object is being created and whenever you call this that particular object is invoking calculate rate this will be called under standard room.

So, this will return 100 into nights right 100 dollars are right that the price is 100 which will be multiplied by nights so standard room so this is your class standard room and the next one is class deluxe room all right so the deluxe room it is also publicly inheriting so i will put dr I will put DR, the deluxe room, which is publicly inheriting room, right. You have the constructor deluxe room, correct. So, you have a constructor deluxe room.

You are passing the number. That means this is a constructor, one argument constructor, same like the previous one. And then number is assigned to room. And again here also you have calculate rate member function, all right. So, which will override the base class room.

right and it will return 115 tonight that means DR is 150. So, this is 100 this is 150 right. So, next one is suit room suit room. So, which is also publicly inheriting all are publicly inheriting right. So, I will put SUR suit room right.

So, this is SUR right. So, which is publicly inheriting room So, all are public if I put public here. So, which is meant for all all right. So, again here you have constructor passing number number is assigned to room and you have calculate rate.

So, that will overwrite the base class room and return 250 into nights ok. So, these are all the derived classes. So, we had seen base classes and these rooms are all the derived classes. So, next we have customer class right. So, it is a separate class.

So customer class, you have a private member data name and phone under string. Here public customer, which is a constructor. So the constructor has customer name, reference customer name under string. And then you have reference customer phone under string. Fine.

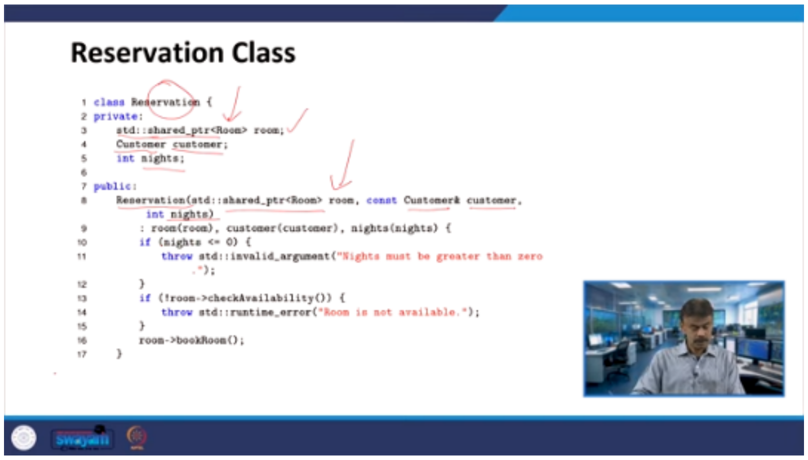
And then the constructor customer name will be copied into name and customer phone will be copied into phone. Fine. So now you have two member functions. One is get name, right? So, get name is returning name and another member function is get phone whose return type is, right, string and it will return phone, right?

Get name whose return type is also string and get phone whose return type is also string. So, they are returning name and phone respectively. So, this is customer class up to here line number 12, you have customer class, right? So, I will write, okay, anyway it is here. fine then the third one reservation class right so now you have to do reservation right so in that case you have class reservation you have private member data room room is a object right under shad underscore ptr std scope resolution operator that means a inbuilt right so shad pointer of room capital r right so what is capital r capital room is a class right so this we had seen already

So, there you have shad underscore ptr of room. It is like a vector of integer kind of right. And then you have customer, customer class we had seen previously. Customer is an object here and here you have a customer class. And you have one member data integer nights right.

So, nights is a member data whose data type is integer. Now, you have reservation, which is the constructor, right? So, under the constructor, you have room, which comes under share_PTR, and then you have customer reference under customer, and then you have nights. So, room, customer, and nights. So, this you have to pass whenever you are calling the reservation or any object under reservation, right? The capital R. So, we have to pass, right.

So, three parameters: room, customer, and nights. So, the room is assigned to room, and the customer is assigned to customer. Okay. So now, line number 10: if nights is less than or equal to 0, right, which is not possible. I mean, somebody cannot stay for 0 nights, right, or negative.



Reservation Class

```
1  class Reservation {
2  private:
3      std::shared_ptr<Room> room;
4      Customer customer;
5      int nights;
6
7  public:
8      Reservation(std::shared_ptr<Room> room, const Customer& customer,
           int nights)
9          : room(room), customer(customer), nights(nights) {
10         if (nights <= 0) {
11             throw std::invalid_argument("Nights must be greater than zero
                   .");
12         }
13         if (!room->checkAvailability()) {
14             throw std::runtime_error("Room is not available.");
15         }
16         room->bookRoom();
17     }
```

So, it has to throw the exception, right? So, what is it throwing? It is throwing an invalid argument exception. So, it is sending the message: nights must be greater than 0, right? So, if the user is passing or you are entering less than or equal to 0, So, nights must be greater than 0 to be displayed. Right. So, let us go to line number 13. Okay. Room is invoking check availability.

Right. If the room is invoking check availability. Right. So, suppose it is returning false. Right.

Room's check availability is returning false. So, now, not false. We know not false is what? True. Right.

If this statement is true. Right. So, initially, check room availability. If it is returning false, that means the room is not available. So, not false will be true.

If this statement is true, then line number 14 will be executed, right? So, it is throwing the exception under runtime error. So, that means this will be displayed. 'Room is not available' will be displayed, okay? So, I hope it is clear.

If the room is not available, yes, it will throw. Or it will print 'Room is not available.' Line number 16, room is invoking book room. Right. So, we have seen the book room function, a member function, right over here.

Line number 13. Right. So, 'book room now' is available equal to false. So, when it is calling this function, yes, somebody is booking the room, meaning the availability of that particular room will be false. So, 'is available' will be false now when it is invoking.

So, this is coming under the reservation class, reservation. So, now we will go to line number 18, right? So, here you have 'distractor,' all right, reservation distractor. So, the room is invoking the object 'room,' which is invoking 'free room,' right? So, 'free room,' yeah, it is like deallocating the memory, kind of equivalent, right?

So, 'free room' is making 'is available' true, right? So, 'free room' is making its availability true. That is what it is doing in the case of 'distractor.' All right. So, now here you are having one function, 'calculate total cost.'

All right. The member function return type is double. So room is invoking calculate rate and you are passing nights the number of nights. Right. So the calculate right.

So room is invoking. So then we know that. All right. So, whichever object that you are creating under this. So, this is inherited by SR, DR and SUR that means standard room, deluxe room and SUR.

So, whatever objects that you are creating accordingly this will be calculated. Anyway this we can see that. So, where you are right. So, which room the customer is selecting accordingly right that will calculate. the cost and then here you have a get customer name whose return type is string so that will return customer is invoking get name right so the get name function which is available here right under customer then get room number whose return type is integer so this will return room is invoking get room number so get room number is available get room number okay so we had seen somewhere yes here

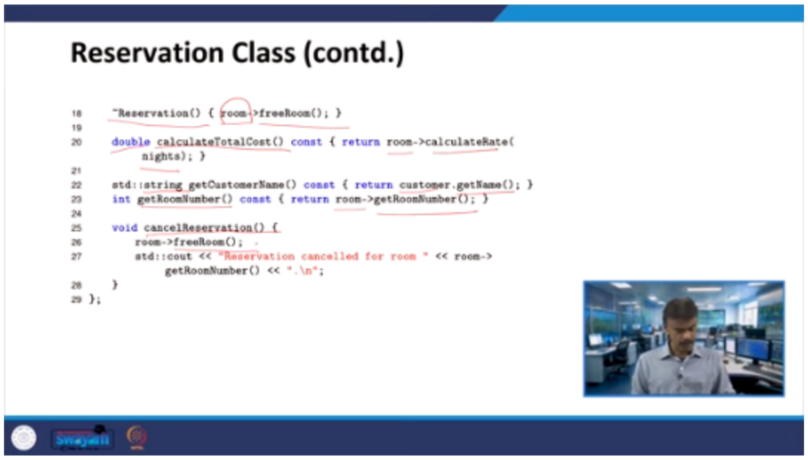
In the room class. Get the room number. That will return the room number. Line number 16. Then you have one more member function called cancel reservation.

Right. So it is possible. Right. So you have to think of all the cases. So what we are studying in this particular case study.

Suppose. Right. We are using this hotel information system. Right. Or hotel booking system.

We have to think about all the possible cases. Right. So you call it a blueprint. Right. So make a blueprint and work accordingly.

So now you have one more member function called cancel reservation. So the room which is invoking free room. Right. The room which is invoking free room. All right.



Reservation Class (contd.)

```
    ~Reservation() { room->freeRoom(); }

    double calculateTotalCost() const { return room->calculateRate(
        nights); }

    std::string getCustomerName() const { return customer.getName(); }
    int getRoomNumber() const { return room->getRoomNumber(); }

    void cancelReservation() {
        room->freeRoom();
        std::cout << "Reservation cancelled for room " << room->
            getRoomNumber() << ".\n";
    }
};
```

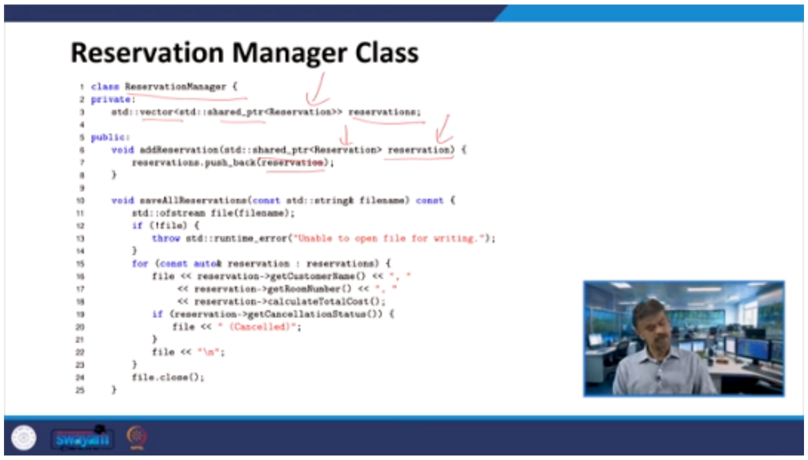
So the room which is invoking free room. That means is available equals true. Right. So that is what will happen over here. The cancel.

All right. So, free room. True. And then see the reservation canceled for the room. Room is invoking get room number.

So, get room number will print which is the room number. Right. So, this also we have seen that will return room number. That is for the cancellation. So now.

We are seeing the reservation manager class. So, under the reservation manager class, you have the class reservation manager, right. So, here you have reservation vector. So, vector of shared PTR that has reservation, correct. And you have public member function add reservation.

Here you are passing reservation under shared PTR reservation. So, now you are putting the reservation this particular object in the stack all right. So, anyway we can see this when you are creating an object. So, right suppose the object is pushing right the customer let us say reservation 1 right. So, then again one more right let us say reservation to push back only right.



Reservation Manager Class

```cpp
class ReservationManager {
private:
    std::vector<std::shared_ptr<Reservation>> reservations;

public:
    void addReservation(std::shared_ptr<Reservation> reservation) {
        reservations.push_back(reservation);
    }

    void saveAllReservations(const std::string& filename) const {
        std::ofstream file(filename);
        if (!file) {
            throw std::runtime_error("Unable to open file for writing.");
        }
        for (const auto& reservation : reservations) {
            file << reservation->getCustomerName() << ", "
                 << reservation->getRoomNumber() << ", "
                 << reservation->calculateTotalCost();
            if (reservation->getCancellationStatus()) {
                file << " (Cancelled)";
            }
            file << "\n";
        }
        file.close();
    }
```

So, I without loss of generality I use linked list. Or if you use a simple array kind of R1 and then push back R2 something like this all right. So, that is add reservation. Save all reservations that is another function. So, here we are handling the file.

So, file name under string all right. So, here we are first time right in this program OF stream output file stream right. So, we use file and you have some file name right. So, whichever you are passing some dot txt or something. if file is true right that means not file is false right that means no file can be opened so in that case unable to open the file writing so this we had seen in the last program itself right so then you have range for look for right so the reservations will under reservation now right so whatever reservation here i put r1 r2 r3 let us say up to rn all right so you keep on going up to rn right so for all the cases

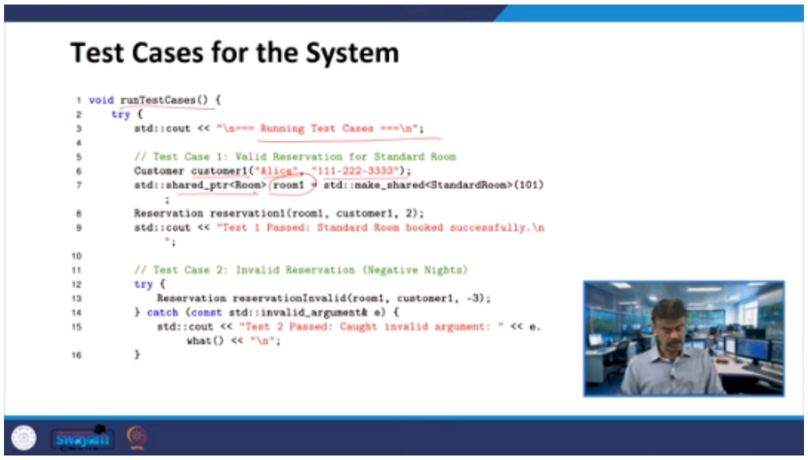
right we are saving right r1 let us say it is replaced by c1 customer 1 customer 2 customer 3 etc customer n all right so here you can have the name all right so reservation will invoke get customer name you are writing in the file get room number writing in the file reservation is invoking calculate total cost writing in the file right So, then if reservation is calling get cancellation status, right. So, we can have the get cancellation function if it is called. So, this is true. So, then in the file it will be written cancel, right.

So, you can see we have seen all the cases and you can see what exactly the function is doing. So, here in this case when it is calling this particular case right get cancellation status if this is true it will be cancel will be return in the file. Then you have a new line and then file is getting closed. So, this is reservation manager class right and yeah in fact we have one more function member function generate all bills. So, this is what you are doing once you check out.

So, this will happen right generate all the bills all right. So, here you have again the for loop the complete reservations. R1, R2, R3 etc. If reservation get cancellation status is false then this will become true not of this will become true. Then you are getting reservation is invoking get customer name.

So, printing customer, then room. Reservation is invoking get room number. You have a new line: total cost in dollars. Reservation is invoking calculate total cost, and then you are printing this. So, this is all about your reservation manager class. So now, we will have the test cases for the system, right? Suppose you are the customer, right? So, the run test cases: we are trying to see out run testing cases. Customer 1, you are passing

the name and phone number. This is what we had seen, right, in the customer class? And here, you have room 1 object under shad ptr of room, right? So, the right-hand side, you are passing right with the make underscore shad, right? You are passing this as a std. Therefore, this is a constructor under this because this will be a class under this. You are passing 101, right? So, now when I am passing this, this will be stored in room, right? This is what we had seen several times. When you are a room 1, right?



Test Cases for the System

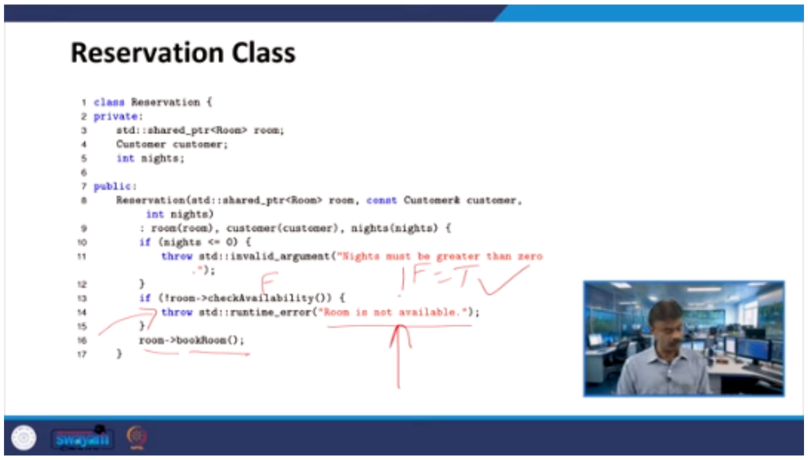
```cpp
void runTestCases() {
    try {
        std::cout << "\n=== Running Test Cases ===\n";

        // Test Case 1: Valid Reservation for Standard Room
        Customer customer1("Alice", "111-222-3333");
        std::shared_ptr<Room> room1 = std::make_shared<StandardRoom>(101)
            ;
        Reservation reservation1(room1, customer1, 2);
        std::cout << "Test 1 Passed: Standard Room booked successfully.\n
            ";

        // Test Case 2: Invalid Reservation (Negative Nights)
        try {
            Reservation reservationInvalid(room1, customer1, -3);
        } catch (const std::invalid_argument& e) {
            std::cout << "Test 2 Passed: Caught invalid argument: " << e.
                what() << "\n";
        }
```

So, this will be shared in room. So, now you have reservation class. You have reservation object passing room 1. So, that is 101, customer 1 allies, right? And the number of days is 2. Right, the number of days is correct.

So, see out test 1 pass standard room book successfully right. So, here you have suppose you are passing let us say minus 3 reservation invalid right this is the object under reservation. So, here you have minus 3 you are passing right. So, when you are passing the negative number You know the exception will be thrown.

This we had already seen. If it is a negative number. So this will be caught here. So test 2 passed. And this will throw the message caught invalid argument.

So whatever the message that we are throwing. If you look for the negative number. If it is less than 0. Yes. If nights less than or equal to 0.



Reservation Class

```cpp
class Reservation {
private:
    std::shared_ptr<Room> room;
    Customer customer;
    int nights;

public:
    Reservation(std::shared_ptr<Room> room, const Customer& customer,
         int nights)
        : room(room), customer(customer), nights(nights) {
        if (nights <= 0) {
            throw std::invalid_argument("Nights must be greater than zero
                .");
        }
        if (!room->checkAvailability()) {
            throw std::runtime_error("Room is not available.");
        }
        room->bookRoom();
    }
```

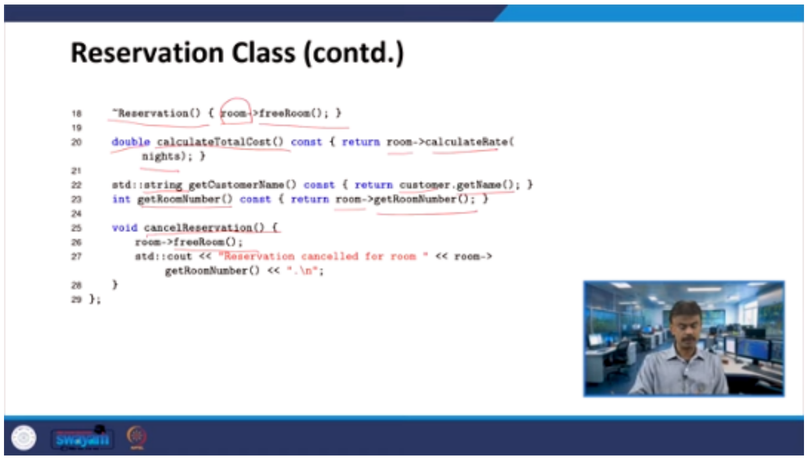
it will print night must be greater than 0. So, this is what will be printed e dot what right. So, caught invalid argument. So, the statement will be printed and here you have room 1 customer 1 comma 1 right. If it is unavailable right if the room is unavailable

So then we had, right? So we had, right? So room 1 is already booked, right? For the customer 1 allies. So that means it is false now.

So if it is false, so again, right? So it is catching the runtime, right? And then whatever be the statement for e.what will be printed, right? So this is the unavailable room. And then bill, correct billing, right?

Customer 2 is an object under the class customer. So you have the name Bob. The phone number, next is room 2, the same, right? Room number is 201, and you are passing room 2, customer 2, 3, right? So the expected cost will be 3 into 250, right? And then if it is equal, right, that means it is calling the calculate total cost. The reservation 2 is invoking calculate total cost, so the calculate total cost here, room 2. Right here under shad underscore ptr room, right? So here, right-hand side, you are upcasting with shoot room, right? So suit room calculate total cost, suit room. Here you have calculate total cost.

So, in fact, when I am calling calculate total cost, that particular function, it will be called over here, right. So, yes, here it is. Calculate total cost. Upcasting with suit room. So, therefore, when the room is invoking the calculate rate by passing the nights, right.



Reservation Class (contd.)

```
    ~Reservation() { room->freeRoom(); }

    double calculateTotalCost() const { return room->calculateRate(
        nights); }

    std::string getCustomerName() const { return customer.getName(); }
    int getRoomNumber() const { return room->getRoomNumber(); }

    void cancelReservation() {
        room->freeRoom();
        std::cout << "Reservation cancelled for room " << room->
            getRoomNumber() << ".\n";
    }
};
```

So, this particular, right, will be called. This particular will be returned 250 into nights, right. So, both are equal. 250 into nights, 3 into 250, right? Both are same.

So, therefore, billing is correct, will be printed. And case 5 is reservation manager, right? Here you have file handling. Manager is an object under reservation manager, right? So, here manager is invoking add reservation, correct?

And here you are pausing room 2, customer 2 and 3, number of nights is 3. So, this you are calling that manager, the object manager is invoking save all reservations. This we had seen by passing this particular file, reservations underscore test dot txt. So, this we had seen. So, save all reservation, you are passing the file.

So, when you are passing the file, so here you go, generate bill, save all reservation. You are passing that particular file. So, then all these will be executed. So, this you know. Right.

So one by one, you can go through that, get customer name, get room number, calculate total cost. Right. So all these will be computed. Right. And it will be returned.

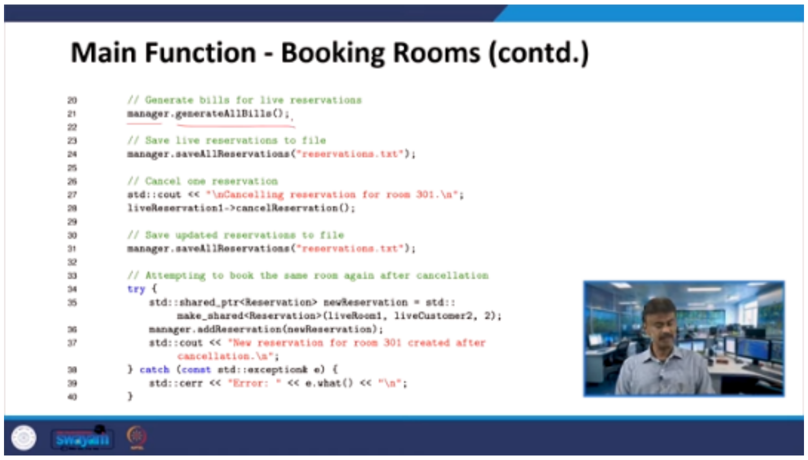
Right. So then you have a see out return in reservation underscore test. So test five pass file handling works correctly. That will be printed. Otherwise, if there is any problem, if I am not able to open this file, this file does not exist, then test fail and whatever write the E dot what that we have written will be printed, exception will be printed, all right.

So, now come to the main program, all right. So, here you are calling run test cases. So, that means you are calling this, right. So, you are having everything. This is already I explained.

So, these things will be done right up to here will be done. Then manager you have reservation manager under reservation manager you have an object manager. So, now we can create live customers and reservations right. So, here so live customer one you have John Doe and his phone number and here you have the object live room one right under shared PTR room. upcasting with shoot room now 301 101 201 and 301 all right and here you have live reservation one under shared ptr so this is a reservation you are pausing room live room one live room one is this 301 and live customer one john do and the number of nights is four right again you are calling add reservation by pausing all the information of live reservation one right so here you go the live reservation one

It will be called and you know that, right? So all these details will be saved. Similarly, for the second case, right? Jane Smith 987-654-3210 is a phone number, right? And then here you have room number 302.

So then you are doing number of days is 2, right? So like this, you can write. And then here generate all. Manager is invoking generate all bills, right? So it will generate bills for the live reservation and save all reservations.



Main Function - Booking Rooms (contd.)

```
    // Generate bills for live reservations
    manager.generateAllBills();

    // Save live reservations to file
    manager.saveAllReservations("reservations.txt");

    // Cancel one reservation
    std::cout << "\nCancelling reservation for room 301.\n";
    liveReservation1->cancelReservation();

    // Save updated reservations to file
    manager.saveAllReservations("reservations.txt");

    // Attempting to book the same room again after cancellation
    try {
        std::shared_ptr<Reservation> newReservation = std::
            make_shared<Reservation>(liveRoom1, liveCustomer2, 2);
        manager.addReservation(newReservation);
        std::cout << "New reservation for room 301 created after
            cancellation.\n";
    } catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << "\n";
    }
```

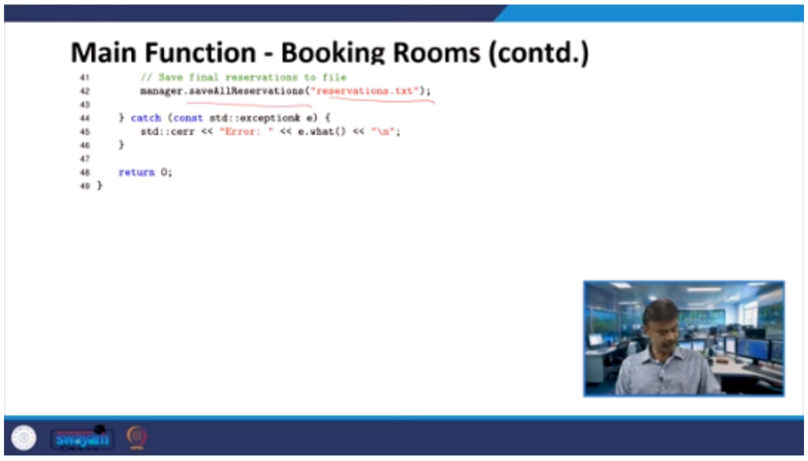
And here you are passing the file reservations.txt. Just go to that function, and all the functionality will be executed. And cancel. See how canceling the reservation for room number 301 works. So room number 301 is getting vacated.

That means the availability will be true. And you can see the function and what exactly is happening. Live reservation 1 is invoking cancel reservation. And then the manager is invoking save all reservations. And here the save updated reservations to 5.

And then attempting to book the same room again after cancellation. Yes, it is possible, right? If it is canceled, the same room can be booked, right? So, here you go. New reservation is an object under shared PTR reservation.

So, the constructor is also a reservation. That means under make shared, you have room customer and the number of nights. So this is a new reservation, right? So you are adding over here. That means this function will be called, and then the rest of the things will happen, right?

So in case you are getting any exception, that will be caught over here, right? And then the manager will save all reservations. Again and again, we are doing the same. So that means the final reservations to the file, right? So this will be saved, the final reservation to the file.



Main Function - Booking Rooms (contd.)

```
        // Save final reservations to file
        manager.saveAllReservations("reservations.txt");

    } catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << "\n";
    }

    return 0;
}
```

If there is any problem, the exception will be caught here. So, what I am asking you—I mean, you can write—I mean, you can work out the complete code, even with some modifications. Right, so that you can practice when you are writing this big code. Right? So when I am running the code, I have to get the output like this. Okay? So whatever I have explained, this is the output, right? So you can try this. Fine. So in this case study, what are all the problems we may face, right? And what are all the remedies? Right.

Room availability. So you have to use the flag. Right. True or false to track the availability. Right.

So this is very important in the hotel management system. The room availability is very important. How you are making sure. right and then dynamic pricing so sometimes you may feel that sometimes it may be that is the indian context the same room you get for 7500 and sometime you may get for 6500 so that you call it as a dynamic pricing or when you are booking one day before the rate is completely changed or one month before the rate is completely changed right so extend room classes to support seasonal pricing and this is a dynamic pricing data persistence So here we have in fact done use file handling for storing the reservations.

So which is cumbersome job and extensibility. So you have to design classes to support additional room types. So the data persistence in the case, it is advisable to use file handling and storing reservations. and extensibility is another challenge the solution is design classes to support additional room types so if you have let us say four or five types we have seen three types suppose it goes five to six types so we have to do that inheritance for extensibility so this is the summary so we have a simulated real world hotel reservation system in this problem we have used several oop principles and we have implemented room booking cancellation billing right so different room types in fact with the dynamic rates in the sense the room rates are different all right and customer and reservation management

So, customer reservation management. So, we have also implemented this. Additionally, we have incorporated error handling, exceptions, and data persistence. So, with this, I am concluding this case study. I hope you have understood and enjoyed it.

Thank you very much.