public class Hotel

public void addRoom(RoomType roomType, int id)

throws a RuntimException if id is a duplicate

otherwise adds a room of type roomType to the hotel

public boolean isRegistered(int phoneNumber)

returns true if guest is registered,

otherwise false

public Guest registerGuest(String name, String address, int phoneNumber)

throws a RuntimeException if guest is already registered

after calling this method;

isRegistered(phoneNumber\_ should return true

findGruestByPhoneNumber should return a valid Guest

public Guest findGuestByPhoneNumber(int phoneNumber)

returns a valid Guest identified by phoneNumber

otherwise returns null

public Room findAvailableRoom(RoomType selectedRoomType, Date arrivalDate, int stayLength) {

returns a room of type roomType that is available from arrivalDate for stayLength

otherwise returns null

public Booking findBookingByConfirmationNumber(long confirmationNumber)

returns a booking identified by confirmation number

if no such booking exists, returns null

public Booking findActiveBookingByRoomId(int roomId)

returns the checked in booking associated with the room identified by roomId

otherwise returns null

\*public long book(Room room, Guest guest, Date arrivalDate, int stayLength, int occupantNumber,

CreditCard creditCard);

Returns a unique confirmation number for a booking.

The confirmation number has this format: ddMMYYYYrrr, where ddMMYYYY is the date of the booking and rrr is the room Id.

After calling this method:

A booking should exist for the room (this method should call room.book())

The room should not be available for the specified arrivalDate and staylength

The booking should be returned from findBookingByConfirmationNumber()

\*public void checkin(long confirmationNumber)

throws a RuntimeException if no booking for confirmation number exists

After calling this method:

The Booking referenced by confirmationNumber should be returned by getActiveBookingByRoomId()

The Booking referenced by confirmationNumber should have a state of CHECKED\_IN

\*public void addServiceCharge(int roomId, ServiceType serviceType, double cost)

throws a RuntimeException if no active booking associated with the room identified by roomID can be found

After this method is called:

A ServiceCharge should have been added to the active booking.

\*public void checkout(int roomId)

throws a RuntimeException if no active booking associated with the room identified by roomID can be found

The Booking referenced by confirmationNumber should have a state of CHECKED\_OUT

public class Room

public Room(int id, RoomType roomType)

returns a properly instantiated Room

following this method:

getId should return id

getType should return roomType

room state should be READY

public int getId()

returns the id of the Room

public String getDescription()

returns the RoomType description

public RoomType getType()

returns the RoomType of the room

public boolean isAvailable(Date arrivalDate, int stayLength)

returns true if no existing bookings conflict with the stated parameters

otherwise returns false

public boolean isReady()

returns true if the rooms state is READY

otherwise false

\*public Booking book(Guest guest, Date arrivalDate, int stayLength, int numberOfOccupants, CreditCard creditCard)

returns a new Booking

After calling this method:

isAvailable() should return false for any arrivalDates and stayLengths that clash with the new booking

the booking state should be PENDING

\*public void checkin()

throws a RuntimeException if the rooms state is not READY

After calling this method

The rooms state should be OCCUPIED

the booking state should be CHECKED\_IN

\*public void checkout(Booking booking)

throws a RuntimeException if the rooms state is not OCCUPIED

After calling this method

The rooms state should be READY

the booking state should be CHECKED\_OUT

public class Booking

public Booking(Guest guest, Room room, Date arrivalDate, int stayLength, numberOfOccupants,

CreditCard creditCard)

Returns a properly instantiated Booking

After calling this method:

All getters should return the values supplied in the constructor

The Booking state should be PERNDING

public boolean doTimesConflict(Date requestedArrival, int stayLength)

returns false if requestedArrival and staylength overlap with booked arrivalTimes and staylength

otherwise returns true

public long getConfirmationNumber()

returns the unique confirmation number associated with the booking

The confirmation number has this format: ddMMYYYYrrr, where ddMMYYYY is the date of the booking and rrr is the room Id.

public int getRoomId()

returns the id of the room associated with the booking

public Room getRoom()

returns the room associated with the booking

public Date getArrivalDate()

returns the arrival date associated with the booking

public int getStayLength()

returns the stay length associated with the booking

public Guest getGuest()

returns the guest associated with the booking

public CreditCard getCreditCard()

returns the credit card associated with the booking

public boolean isPending()

returns true if booking state is PENDING

otherwise returns false

public boolean isCheckedIn()

returns true if booking state is CHECKED\_IN

otherwise returns false

public boolean isCheckedOut()

returns true if booking state is CHECKED\_OUT

otherwise returns false

public List<ServiceCharge> getCharges()

returns a List containing all ServiceCharges associated with this Booking

\*public void checkIn()

throws a RuntimeException if booking state is not PENDING

After calling this method:

Room associated with booking state should be OCCUPIED

Booking state should be CHECKED\_IN

\*public void addServiceCharge(ServiceType serviceType, double cost)

creates a new ServiceCharge

After calling this method:

The List returned by getCharges should include an entry for this ServiceCharge

\*public void checkOut() {

throws a RuntimeException if booking state is not CHECKED\_IN

After calling this method:

Room associated with booking state should be READY

Booking state should be CHECKED\_OUT

public class BookingCTL {

public BookingCTL(Hotel hotel)

creates a new Booking UI

sets state to PHONE

returns a properly instantiated BookingCTL

public void run() {

calls UI.run()

public void phoneNumberEntered(int phoneNumber)

throws a RuntimeException if state is not PHONE

calls hotel.findGuestByPhoneNumber()

if guest identified by phoneNUmber is registered

calls UI.displayGuestDetails()

sets state to ROOM

sets UI state to ROOM

else

sets state to REGISTER

sets UI state to REGISTER

public void guestDetailsEntered(String name, String address)

throws a RuntimeException if state is not REGISTER

calls hotel.registerGuest()

calls UI.displayGuestDetails()

sets state to ROOM

sets UI state to ROOM

public void roomTypeAndOccupantsEntered(RoomType selectedRoomType, int occupantNumber)

throws a RuntimeException if state is not ROOM

calls selectedRoomType.isSuitable()

if selectedRoomType is suitable

sets state to TIMES

sets UI state to TIMES

else

calls UI.displayMessage() with room not suitable message

public void bookingTimesEntered(Date arrivalDate, int stayLength)

throws a RuntimeException if state is not ROOM

calls hotel.findAvailableRoom()

if room is available

calls selectedRoomType.calculateCost()

calls UI.displayBookingDetails()

sets state to CREDIT

sets UI state to CREDIT

else

calls UI.displayMessage() with room unavailable message

public void creditDetailsEntered(CreditCardType type, int number, int ccv)

throws a RuntimeException if state is not CREDIT

creates a new CreditCard

calls CreditAuthorizer.authorise()

if approved

calls hotel.book()

calls UI.displayConfirmedBooking()

sets state to COMPLETED

sets UI state to COMPLETED

else

calls UI.displayMessage with credit not authorised message

public class CheckinCTL

public CheckinCTL (Hotel hotel)

creates a new Booking UI

sets state to CHECKING

returns a properly instantiated CheckinCTL

public void run() {

calls UI.run()

\*public void confirmationNumberEntered(long confirmationNumber)

throws a RuntimeException if state is not CHECKING

calls hotel.findBookingByConfirmationNumber()

if no booking

calls UI.displayMessage() with No booking found message

calls booking.isPending

if booking is not pending

calls booking.isCheckedIn()

if booking is checked in

calls UI.displayMessage() with Already checked in message

else

calls UI.displayMessage() with Already checked out message

else

calls booking.getRoom()

calls room.isReady()

if room is not ready

calls UI.displayMessage() with Room not ready message

else

calls UI.displayCheckInMessage

sets state to CONFIRMING

sets UI state to CONFIRMING

public void checkInConfirmed(boolean confirmed)

throws a RuntimeException if state is not CONFIRMING

if confirmed

calls hotel.checkin()

calls UI.displayMessage() with Checkin confirmed message

sets state to COMPLETED

sets UI state to COMPLETED

else

calls UI.displayMessage() with Checkin cancelled

sets state to CANCELLED

sets UI state to CANCELLED

public class RecordServiceCTL {

public RecordServiceCTL(Hotel hotel)

creates a new RecordServiceUI

sets state to ROOM

public void run()

calls UI.run()

public void roomNumberEntered(int roomNumber)

throws a RuntimeException if state is not CONFIRMING

calls hotel.findActiveBookingByRoomId()

if no booking found

calls UI.displayMessage() with No Active Booking message

else

sets state to SERVICE

sets UI state to SERVICE

public void serviceDetailsEntered(ServiceType serviceType, double cost) {

throws a RuntimeException if state is not SERVICE

calls hotel.addServiceCharge()

calls UI.displayServiceChargeMessage()

sets state to COMPLETED

sets UI state to COMPLETED

public class CheckoutCTL

public CheckoutCTL(Hotel hotel) {

creates a new CheckoutUI

sets state to ROOM

public void run()

calls UI.run()

public void roomIdEntered(int roomId) {

throws a RuntimeException if state is not ROOM

calls hotel.findActiveBookingByRoomId()

if no booking found

calls UI.displayMessage() with No Active Booking message

else

starts to build a checkout invoice (calls multiple getters for guest, room, and booking details)

calls booking.getCharges()

for each charge

calls charge.getDescription()

calls charge.getCost()

adds description and cost to invoice

calls UI.displayMessage with checkout invoice message

sets state to ACCEPT

sets UI state to ACCEPT

public void chargesAccepted(boolean accepted) {

throws a RuntimeException if state is not ACCEPT

if charges accepted

calls UI.displayMessage() with charges accepted message

sets state to CREDIT

sets UI state to CREDIT

else

calls UI.displayMessage() with charges not accepted message

calls UI.displayMessage() with Checkout cancelled

sets state to CANCELLED

sets UI state to CANCELLED

public void creditDetailsEntered(CreditCardType type, int number, int ccv) {

throws a RuntimeException if state is not CREDIT

creates a new CreditCard

calls CreditAuthorizer.authorize()

if approved

calls hotel.checkout()

calls UI.displayMessage() with Credit card debited message

sets state to COMPLETED

sets UI state to COMPLETED

else

calls UI.displayMessage() with Credit not approved message