# Система записи к врачу

@startuml

' Define classes with fields and methods

class Hospital {

+name: String

}

class Office {

+officeId: Integer

+name: String

}

class User {

+userId: String

+email: String

}

class Doctor {

+specialization: String

applyToOffice(office: Office)

createSchedule(office: Office, timeslot: TimePeriod)

}

class Patient {

+registeredDate: Date

bookAppointment(doctor: Doctor, schedule: Schedule, appointmentType: String): Boolean

}

class Schedule {

+date: Date

+startTime: Time

+endTime: Time

isTimeAvailable(time: Time): Boolean

}

class Appointment {

+appointmentId: Integer

+type: String

+duration: Integer

}

class BloodTest {

+duration = 15

}

class Consultation {

+duration = 30

}

class Surgery {

+duration = 60

}

' Define relationships

Hospital "1" -- "\*" Office

Doctor "1" -- "\*" Schedule

Schedule "\*" -- "1" Office

Schedule "1" -- "\*" Appointment

Appointment "\*" -- "1" Doctor

Appointment "\*" -- "1" Patient

User <|-- Doctor

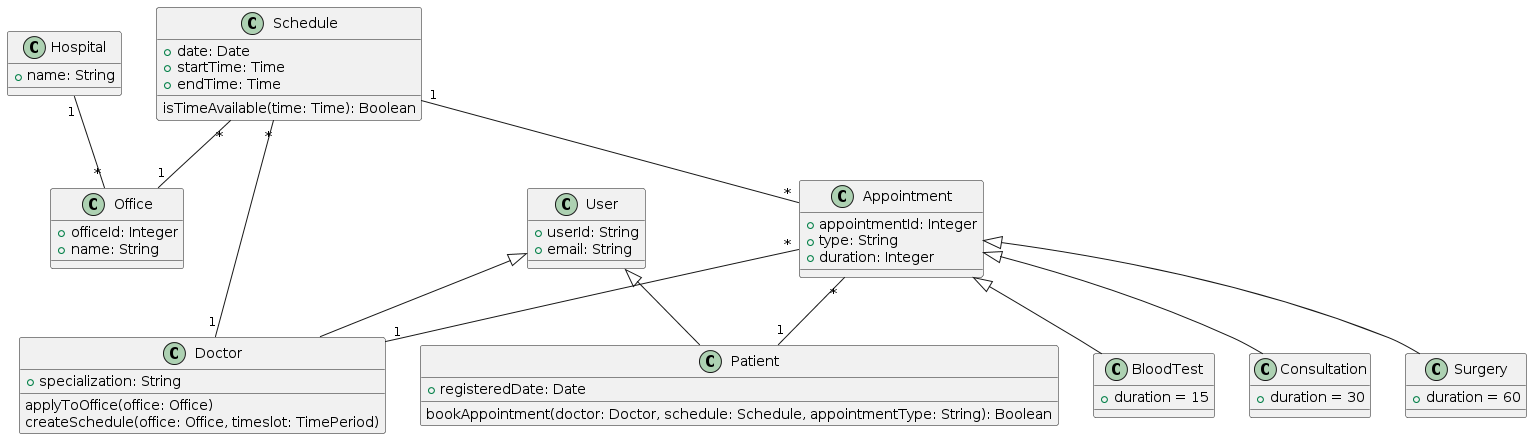
User <|-- Patient

Appointment <|-- BloodTest

Appointment <|-- Consultation

Appointment <|-- Surgery

@enduml



@startuml

left to right direction

skinparam packageStyle rectangle

actor Doctor

actor Patient

rectangle MyDoctor {

usecase "Apply to Practice in Offices" as Apply

usecase "Create Schedules" as CreateSch

usecase "Manage Appointments" as ManageApp

usecase "View Doctors" as ViewDoc

usecase "View Office Schedules" as ViewSch

usecase "Book Appointments" as BookApp

usecase "Receive Confirmation Email" as ReceiveEmail

Doctor --> Apply

Doctor --> CreateSch

Doctor --> ManageApp

Patient --> ViewDoc

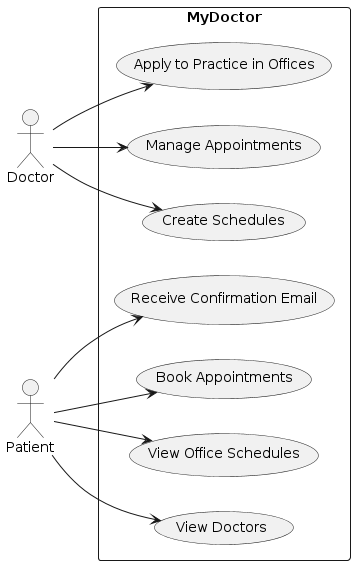
Patient --> ViewSch

Patient --> BookApp

Patient --> ReceiveEmail

}

@enduml



@startuml

actor Patient

entity "System" as System

entity "Doctor" as Doctor

entity "Email Service" as Email

Patient -> System : Request to book appointment

System -> Doctor : Check availability

Doctor -> System : Confirm availability

alt If available

System -> Patient : Show available times

Patient -> System : Select time

System -> Doctor : Book time

Doctor -> System : Confirm booking

System -> Email : Send confirmation email

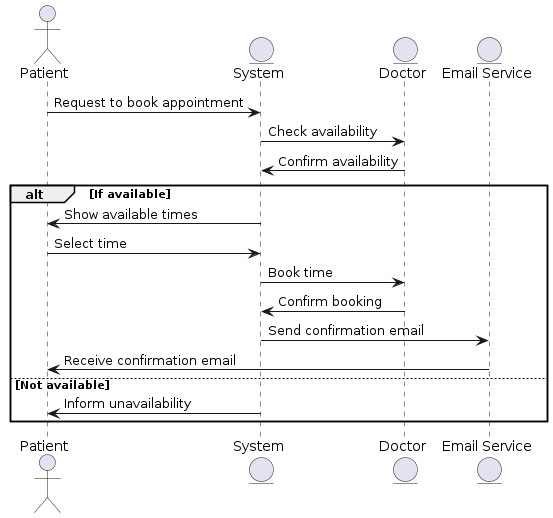
Email -> Patient : Receive confirmation email

else Not available

System -> Patient : Inform unavailability

end

@enduml



# Система управления доставками

@startuml

' Define classes with more detailed attributes and methods

class Customer {

- vatNumber: String

- name: String

- phoneNumber: String

- address: String

--

+ login(): void

+ selectRecipient(recipientVAT: String): Customer

+ enterPackageDetails(weight: double, type: String): Package

}

class Package {

- identifier: String

- weight: Double

- deliveryType: String

--

+ generateIdentifier(): String

+ getDeliveryStatus(): String

}

class DeliveryCenter {

- name: String

- address: String

--

+ routePackage(package: Package): void

+ receivePackage(package: Package): void

}

class Courier {

- vatNumber: String

- name: String

- phoneNumber: String

--

+ assignPackage(package: Package): void

+ deliverPackage(package: Package): void

}

' Relationships

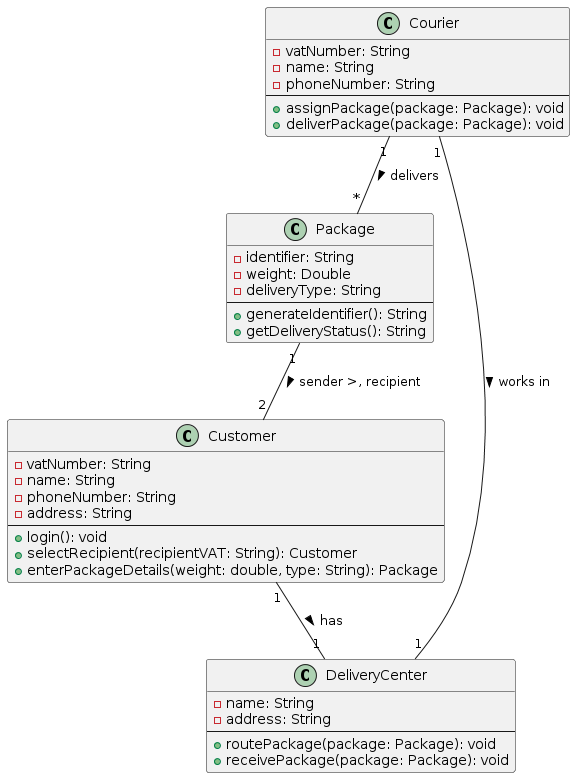
Customer "1" -- "1" DeliveryCenter : has >

Package "1" -- "2" Customer : sender >, recipient >

Courier "1" -- "\*" Package : delivers >

Courier "1" -- "1" DeliveryCenter : works in >

@enduml



@startuml

left to right direction

skinparam packageStyle rectangle

actor Customer

actor Courier

actor "System Administrator" as Admin

rectangle "Delivery Company System" {

usecase "Login" as UC1

usecase "Send Package" as UC2

usecase "Choose Delivery Center" as UC3

usecase "Deliver Package" as UC4

usecase "Track Package" as UC5

usecase "Manage Couriers" as UC6

usecase "Manage Delivery Centers" as UC7

Customer --> UC1

Customer --> UC2

Customer --> UC3

Customer --> UC5

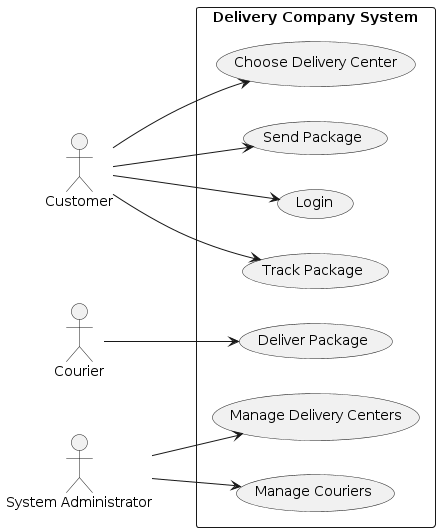
Courier --> UC4

Admin --> UC6

Admin --> UC7

}

@enduml



@startuml

actor Customer

entity "System Interface" as System

entity "Delivery Center" as DC

entity Courier

Customer -> System : Log in

activate System

Customer -> System : Select recipient and enter package details

System -> System : Generate unique identifier

System --> Customer : Display identifier

deactivate System

Customer -> System : Choose Delivery Center

activate System

System -> DC : Process package

activate DC

DC -> Courier : Assign Courier

activate Courier

DC --> System : Confirmation of processing

deactivate DC

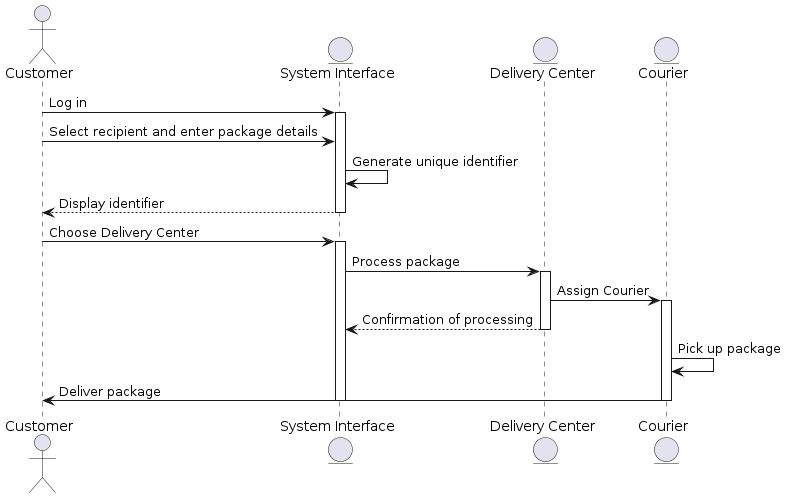
Courier -> Courier : Pick up package

Courier -> Customer : Deliver package

deactivate Courier

deactivate System

@enduml



# Cистема управления полетами и пилотами

@startuml

' This defines the diagram type and style

skinparam classAttributeIconSize 0

class Airline {

+id : String

+aircrafts : Set<Aircraft>

+flights : Set<Flight>

+pilots : Set<Pilot>

}

class Flight {

+id : String

+departureTime : DateTime

+arrivalTime : DateTime

+aircraft : Aircraft

+pilot : Pilot

+coPilot : Pilot

+departureAirport : Airport

+arrivalAirport : Airport

}

class Airport {

+identifier : String

}

class Aircraft {

+type : AircraftType

+state : String {Working, UnderRepair}

+status : String {Landed, Airborne}

}

class Pilot {

+experienceLevel : Integer {1..3}

}

class AircraftType {

+name : String

+requiredPilots : Integer

+roles : Map<String, Integer> {Captain:1, CoPilot:1, Navigator:optional}

}

' Relationships

Airline "1" -- "\*" Flight : operates >

Airline "1" -- "\*" Aircraft : owns >

Airline "1" -- "\*" Pilot : employs >

Flight "1" -- "1" Pilot : pilot >

Flight "1" -- "1" Pilot : coPilot >

Flight "1" -- "1" Airport

Flight "1" -- "1" Aircraft

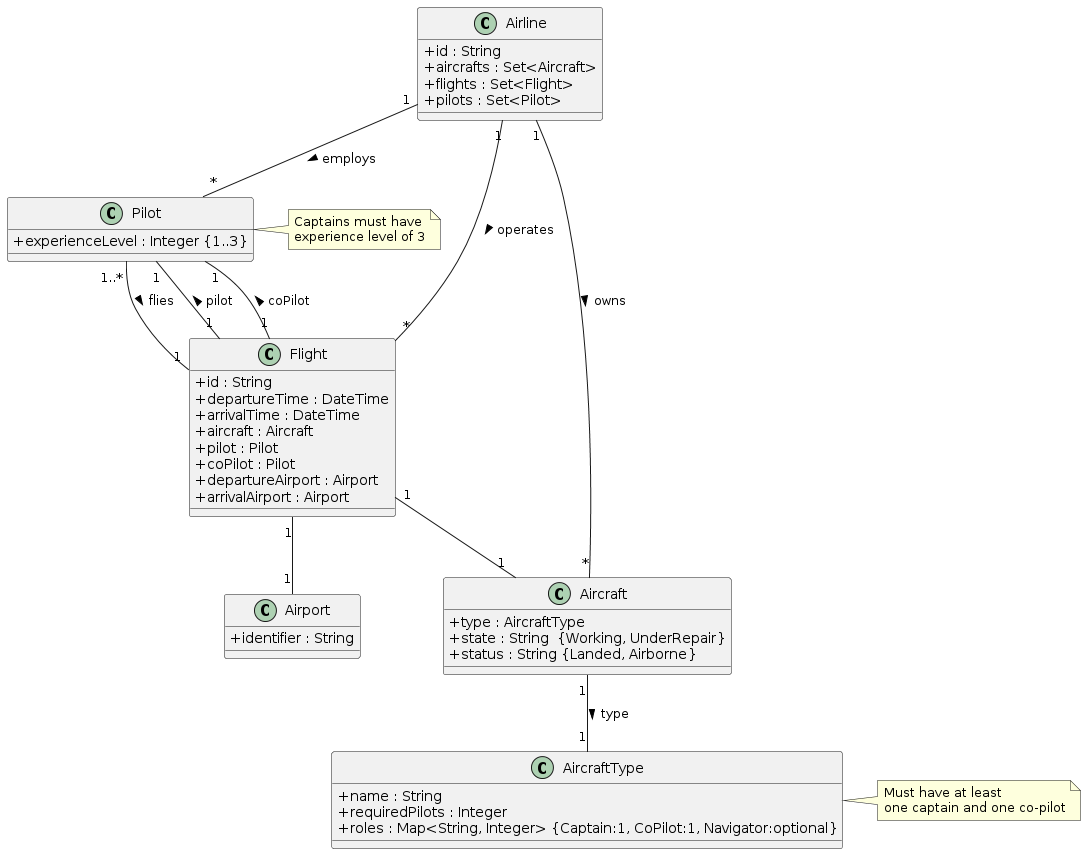
Aircraft "1" -- "1" AircraftType : type >

Pilot "1..\*" -- "1" Flight : flies >

' Specific constraints

note right of Pilot : Captains must have\nexperience level of 3

note right of AircraftType : Must have at least\none captain and one co-pilot

@enduml 

@startuml

left to right direction

skinparam packageStyle rectangle

actor AirlineAdmin as "Airline Admin"

actor Pilot

actor MaintenanceCrew as "Maintenance Crew"

rectangle "Flight and Pilot Management System" {

usecase "Schedule Flight" as UC1

usecase "Assign Pilot" as UC2

usecase "Manage Aircraft" as UC3

usecase "Log Flight Hours" as UC4

usecase "Manage Maintenance" as UC5

usecase "Check Flight Status" as UC6

usecase "Update Pilot Experience" as UC7

AirlineAdmin --> UC1

AirlineAdmin --> UC2

AirlineAdmin --> UC3

AirlineAdmin --> UC7

Pilot --> UC4

Pilot --> UC6

MaintenanceCrew --> UC5

MaintenanceCrew --> UC3

}

' Additional descriptions

note right of UC1 : "Create and manage flight schedules including departure and arrival times."

note right of UC2 : "Assign pilots and co-pilots to flights based on experience and aircraft type."

note right of UC3 : "Manage aircraft assignments and status updates (landed, airborne, under repair)."

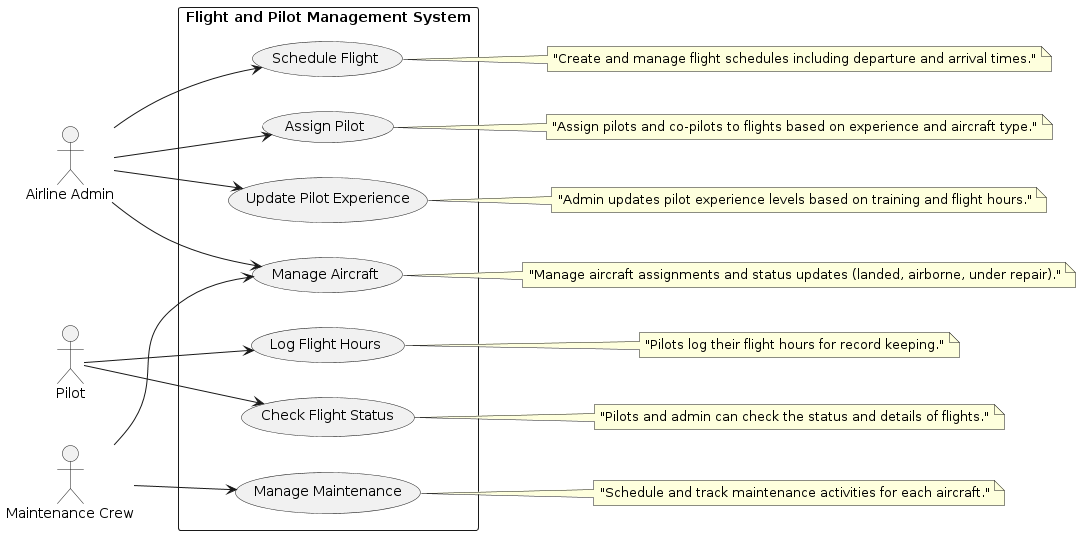
note right of UC4 : "Pilots log their flight hours for record keeping."

note right of UC5 : "Schedule and track maintenance activities for each aircraft."

note right of UC6 : "Pilots and admin can check the status and details of flights."

note right of UC7 : "Admin updates pilot experience levels based on training and flight hours."

@enduml



@startuml

actor AirlineAdmin as "Airline Admin"

participant "Flight Management System" as FMS

participant "Aircraft Management" as AM

participant "Pilot Management" as PM

database "Flight Database" as DB

AirlineAdmin -> FMS : Request to schedule flight

activate FMS

FMS -> AM : Check available aircraft

activate AM

AM -> DB : Query aircraft status

activate DB

DB --> AM : Aircraft details

deactivate DB

AM --> FMS : Aircraft assigned

deactivate AM

FMS -> PM : Assign pilots

activate PM

PM -> DB : Query pilot availability

activate DB

DB --> PM : Pilot details

deactivate DB

PM --> FMS : Pilots assigned

deactivate PM

FMS -> DB : Save flight details

activate DB

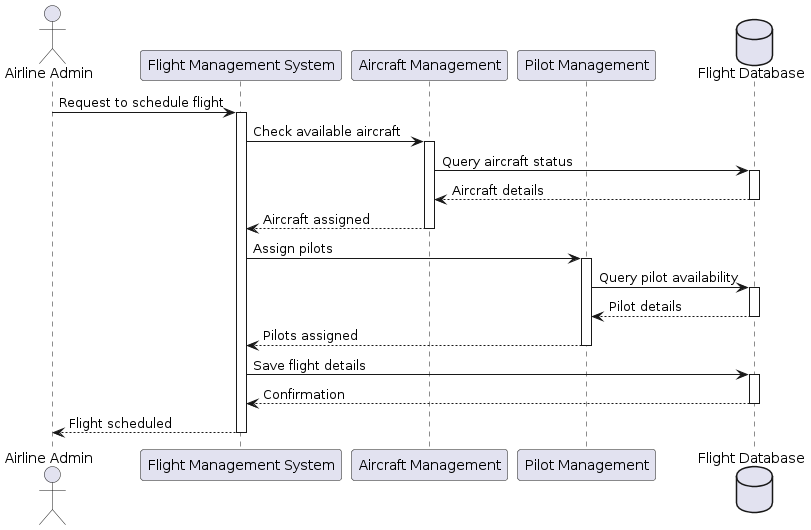
DB --> FMS : Confirmation

deactivate DB

FMS --> AirlineAdmin : Flight scheduled

deactivate FMS

@enduml



# Система управления проектами

@startuml

class ProjectManager {

- manageProject()

}

class Project {

- startDate : Date

- endDate : Date

- initiateProject()

- terminateProject()

}

class Team {

}

class WorkProduct {

- description : String

- percentComplete : int

- validate() : Boolean

}

class Requirement extends WorkProduct {

- validateWithUsers()

- publishMedia() : String

}

class System extends WorkProduct {

- testAgainstRequirements() : Boolean

- deploy(platform : String)

}

ProjectManager "1" -- "1" Project : manages >

Project "1" -- "1" Team : leads >

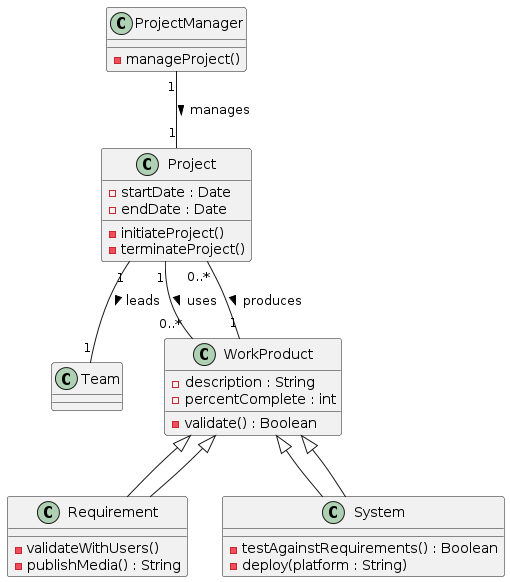
Project "1" -- "0..\*" WorkProduct : uses >

Project "0..\*" -- "1" WorkProduct : produces >

WorkProduct <|-- Requirement

WorkProduct <|-- System

@enduml



@startuml

left to right direction

actor ProjectManager

rectangle "Project Management System" {

usecase "Create Project" as UC1

usecase "Initiate Project" as UC2

usecase "Terminate Project" as UC3

usecase "Manage Team" as UC4

usecase "Update Project Status" as UC5

usecase "Validate Requirements" as UC6

usecase "Publish Requirements" as UC7

usecase "Test System" as UC8

usecase "Deploy System" as UC9

}

ProjectManager --> UC1 : initiates

ProjectManager --> UC2 : manages

ProjectManager --> UC3 : decides

ProjectManager --> UC4 : leads

ProjectManager --> UC5 : updates

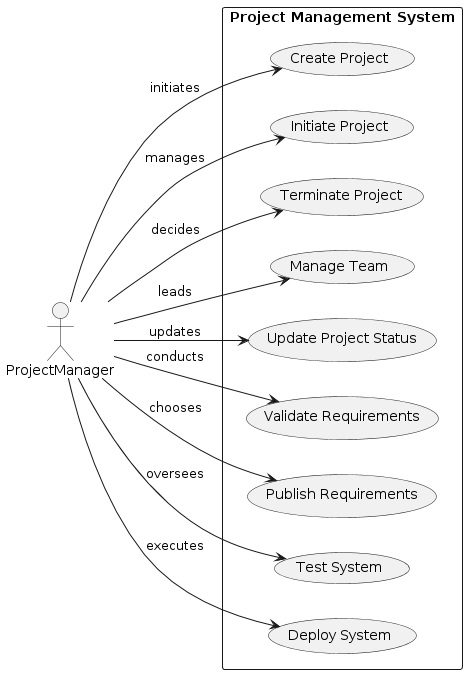
ProjectManager --> UC6 : conducts

ProjectManager --> UC7 : chooses

ProjectManager --> UC8 : oversees

ProjectManager --> UC9 : executes

@enduml



@startuml

actor ProjectManager

participant "Project Management System" as PMS

participant "Team" as Team

participant "WorkProduct" as WP

ProjectManager -> PMS : CreateProject(startDate, endDate)

activate PMS

PMS -> Team : FormTeam()

activate Team

deactivate Team

ProjectManager -> PMS : InitiateProject()

PMS -> WP : AssignRequirements()

activate WP

WP --> PMS : RequirementsAssigned()

deactivate WP

loop Each Milestone

ProjectManager -> PMS : UpdateProjectStatus()

PMS -> Team : AssignTasks()

Team --> PMS : TasksCompleted()

end

ProjectManager -> PMS : TerminateProject()

PMS -> WP : ValidateRequirements()

WP --> PMS : RequirementsValid()

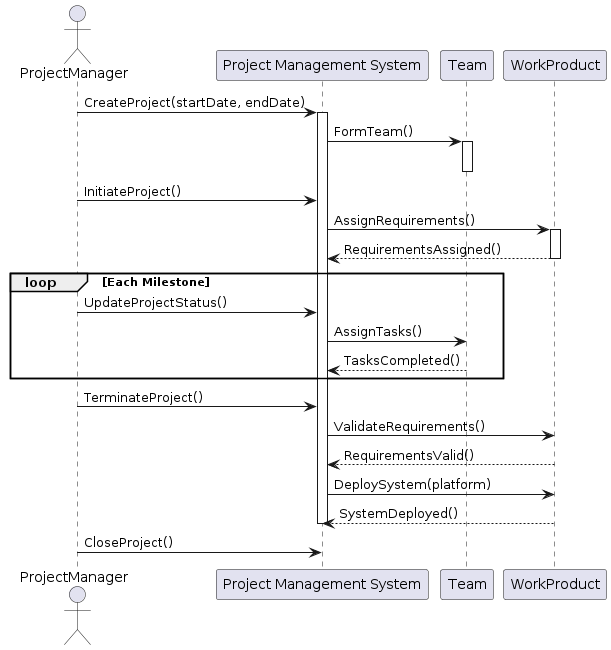
PMS -> WP : DeploySystem(platform)

WP --> PMS : SystemDeployed()

deactivate PMS

ProjectManager -> PMS : CloseProject()

@enduml



# Система проката фильмов

@startuml

class User {

+String name

+String email

}

class Subscriber {

+RechargeableCard card

}

Subscriber --|> User : inherits

class RechargeableCard {

+double credit

+void updateCredit(double amount)

}

class Movie {

+String title

+String director

+int stock

+boolean checkAvailability()

+void orderMovie()

}

class Order {

+String orderId

+Date orderDate

+double amount

}

class Catalogue {

+List<Movie> movies

+Movie findMovie(String title)

}

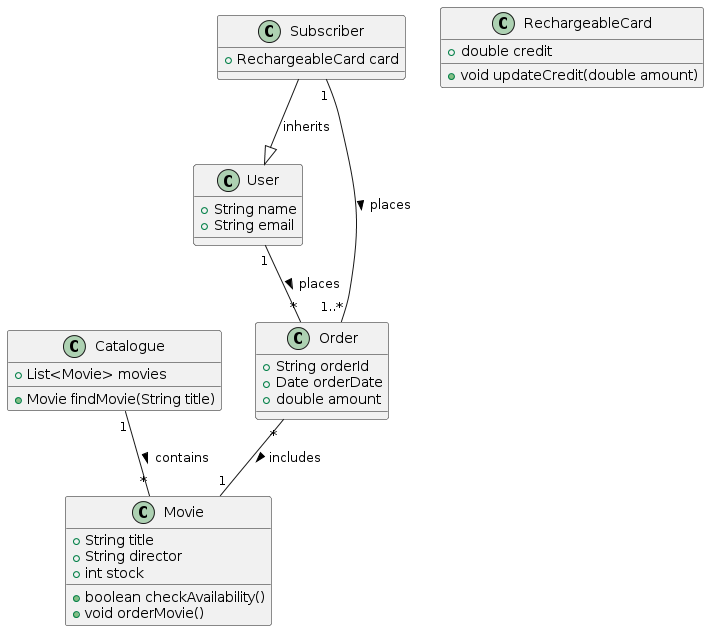
User "1" -- "\*" Order : places >

Order "\*" -- "1" Movie : includes >

Subscriber "1" -- "1..\*" Order : places >

Catalogue "1" -- "\*" Movie : contains >

@enduml



@startuml

left to right direction

actor User

actor Subscriber

usecase "Browse Catalogue" as Browse

usecase "Buy Movie" as Buy

usecase "Rent Movie" as Rent

usecase "Recharge Card" as Recharge

usecase "Check Availability" as Check

usecase "Order Movie" as Order

User --> Browse

User --> Buy

Subscriber --> Browse

Subscriber --> Buy

Subscriber --> Rent

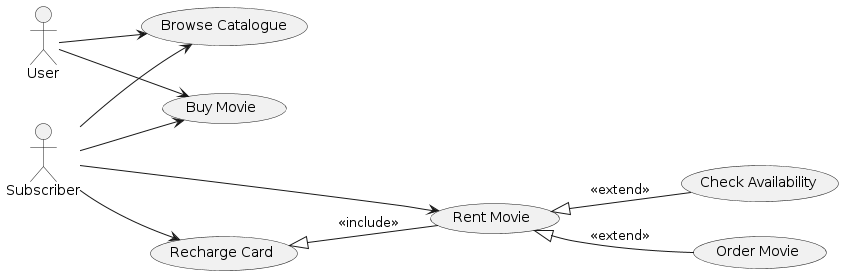
Subscriber --> Recharge

Rent <|-- Check : <<extend>>

Rent <|-- Order : <<extend>>

Recharge <|-- Rent : <<include>>

@enduml



@startuml

actor Subscriber

entity "System" as System

entity "Catalogue" as Catalogue

entity "Movie" as Movie

entity "RechargeableCard" as Card

Subscriber -> System : requestRentMovie(title)

System -> Catalogue : findMovie(title)

Catalogue -> Movie : getAvailability()

Movie --> Catalogue : availabilityStatus

alt if movie is available

Catalogue --> System : movieAvailable

System -> Card : checkCredit()

Card --> System : creditStatus

alt if credit is sufficient

System -> Movie : rentMovie()

Movie --> System : updateStock()

System -> Card : deductCredit(amount)

Card --> System : updateCredit()

System -> Subscriber : rentSuccess()

else

System -> Subscriber : insufficientCredit()

end

else

Catalogue --> System : movieNotAvailable

System -> Subscriber : offerToOrder()

Subscriber -> System : confirmOrder()

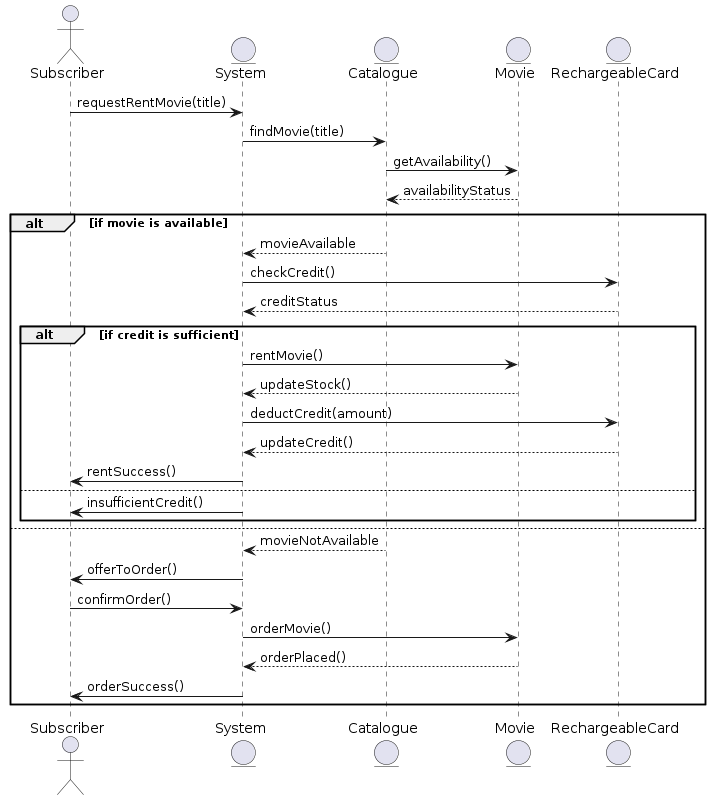
System -> Movie : orderMovie()

Movie --> System : orderPlaced()

System -> Subscriber : orderSuccess()

end

@enduml



# Частная парковка

@startuml

class Employee {

+name: String

+department: String

+cardNumber: String

}

class Guest {

+cardNumber: String

+dateIssued: Date

}

class IdentityCard {

+cardNumber: String

{abstract}

+verifyCard(): Boolean

}

class CardReader {

+readCard(cardNumber: String): Boolean

+sendSignal()

}

class Barrier {

+status: String

+raiseBarrier()

+lowerBarrier()

}

class CarPark {

+capacity: int

+numberOfCarsParked: int

+isFull(): Boolean

+addCar()

+removeCar()

}

class Sign {

+message: String

+updateMessage()

}

Employee -down-|> IdentityCard

Guest -down-|> IdentityCard

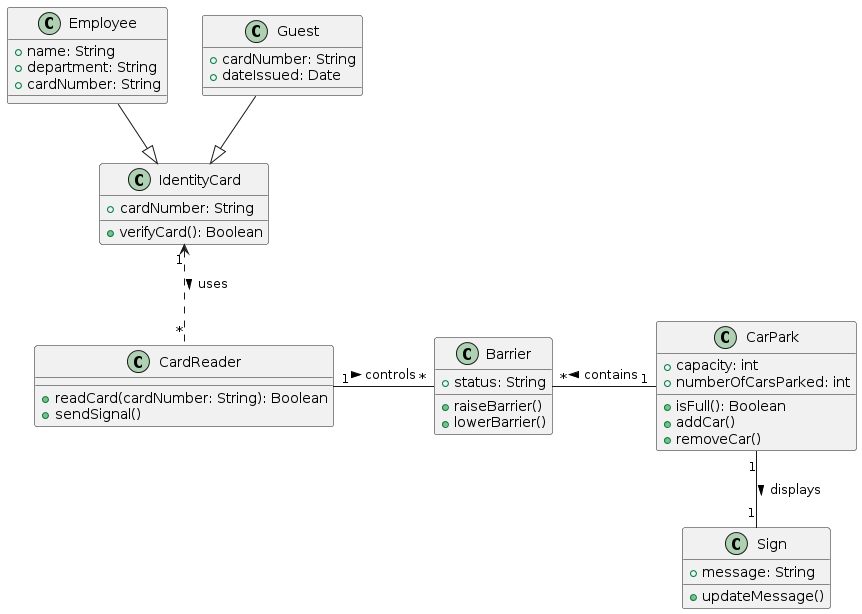
CardReader "1" -right- "\*" Barrier : controls >

CarPark "1" -left- "\*" Barrier : contains >

CarPark "1" -- "1" Sign : displays >

IdentityCard "1" <.. "\*" CardReader : uses >

@enduml



@startuml

left to right direction

skinparam packageStyle rectangle

actor Employee

actor Guest

actor SecurityStaff

rectangle CarParkAccess {

usecase "Access Car Park" as AccessCarPark

usecase "Exit Car Park" as ExitCarPark

usecase "Issue Guest Card" as IssueGuestCard

usecase "Verify Card" as VerifyCard

usecase "Update Sign" as UpdateSign

Employee --> AccessCarPark

Employee --> ExitCarPark

Guest --> AccessCarPark

Guest --> ExitCarPark

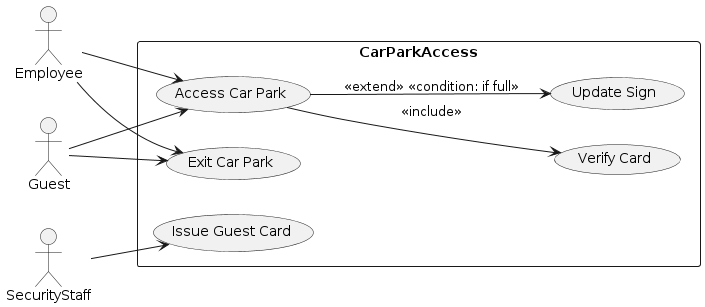
SecurityStaff --> IssueGuestCard

AccessCarPark --> VerifyCard : <<include>>

AccessCarPark --> UpdateSign : <<extend>> <<condition: if full>>

}

@enduml



@startuml

participant "Driver" as Driver

participant "Card Reader" as Reader

participant "System" as System

participant "Entrance Barrier" as EntranceBarrier

participant "Car Park Sign" as Sign

participant "Exit Barrier" as ExitBarrier

Driver -> Reader : Insert Card

activate Reader

Reader -> System : Verify Card Number

activate System

alt Card Valid

System -> Sign : Check if Full

activate Sign

alt Not Full

Sign --> System : Not Full

System -> EntranceBarrier : Send Signal to Open

activate EntranceBarrier

EntranceBarrier -> EntranceBarrier : Open

deactivate EntranceBarrier

Driver -> Driver : Drive In

else Full

Sign --> System : Full

System -> Driver : Display "Full"

end

deactivate Sign

else Card Invalid

System -> Driver : Access Denied

end

deactivate System

deactivate Reader

Driver -> ExitBarrier : Approach Exit

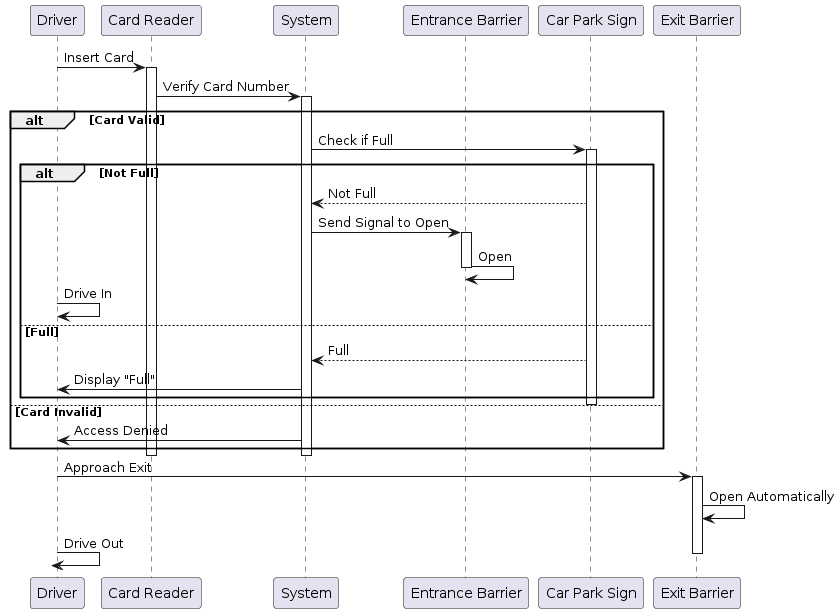
activate ExitBarrier

ExitBarrier -> ExitBarrier : Open Automatically

Driver -> Driver : Drive Out

deactivate ExitBarrier

@enduml



# Система управления фабрикой

@startuml

' Define Classes

class Factory {

}

class Machine {

- serialNumber: String

- make: String

- model: String

- purchaseDate: Date

}

class Worker {

- firstName: String

- lastName: String

- birthDate: Date

- address: String

- skills: List<String>

}

class Product {

- quantity: Int

}

class ProductType {

- name: String

- weight: Double

}

class Material {

- name: String

}

class Client {

- name: String

- address: String

- phoneNumber: String

- contactPerson: String

}

class PurchaseOrder {

- orderNumber: String

- orderDate: Date

- expectedDeliveryDate: Date

- actualDeliveryDate: Date

- products: List<Product>

}

' Define Relationships

Factory "1" -- "\*" Machine

Machine "\*" -- "\*" Worker : operates

Machine "1" -- "\*" ProductType : produces

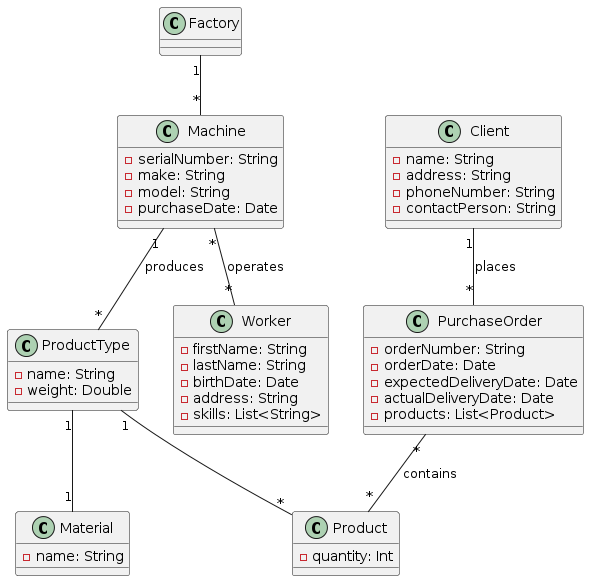
ProductType "1" -- "1" Material

ProductType "1" -- "\*" Product

Client "1" -- "\*" PurchaseOrder : places

PurchaseOrder "\*" -- "\*" Product : contains

@enduml



@startuml

left to right direction

skinparam packageStyle rectangle

actor Worker

actor Client

actor "System Administrator" as Admin

rectangle "Factory Management System" {

usecase "Operate Machine" as UC1

usecase "Assign Worker to Machine" as UC2

usecase "Produce Product" as UC3

usecase "Place Order" as UC4

usecase "Update Worker Info" as UC5

usecase "Update Machine Info" as UC6

usecase "Update Client Info" as UC7

usecase "Track Order" as UC8

Worker --> UC1

Worker --> UC3

Admin --> UC2

Client --> UC4

Client --> UC8

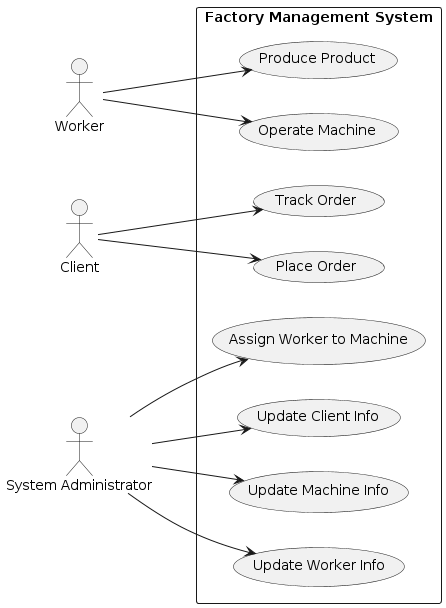
Admin --> UC5

Admin --> UC6

Admin --> UC7

}

@enduml



@startuml

actor Client

entity "Order System" as OrderSystem

database "Database" as DB

entity "Worker" as Worker

entity "Machine" as Machine

== Order Placement ==

Client -> OrderSystem : placeOrder(productId, quantity)

activate OrderSystem

OrderSystem -> DB : getProductDetails(productId)

activate DB

DB --> OrderSystem : productDetails

deactivate DB

OrderSystem -> DB : checkMachineForProduct(productId)

activate DB

DB --> OrderSystem : machineDetails

deactivate DB

OrderSystem -> DB : assignWorkerToMachine(machineId)

activate DB

DB --> OrderSystem : workerDetails

deactivate DB

== Product Production ==

OrderSystem -> Worker : operateMachine(machineId)

activate Worker

Worker -> Machine : produce(productType)

activate Machine

Machine --> Worker : productionComplete

deactivate Machine

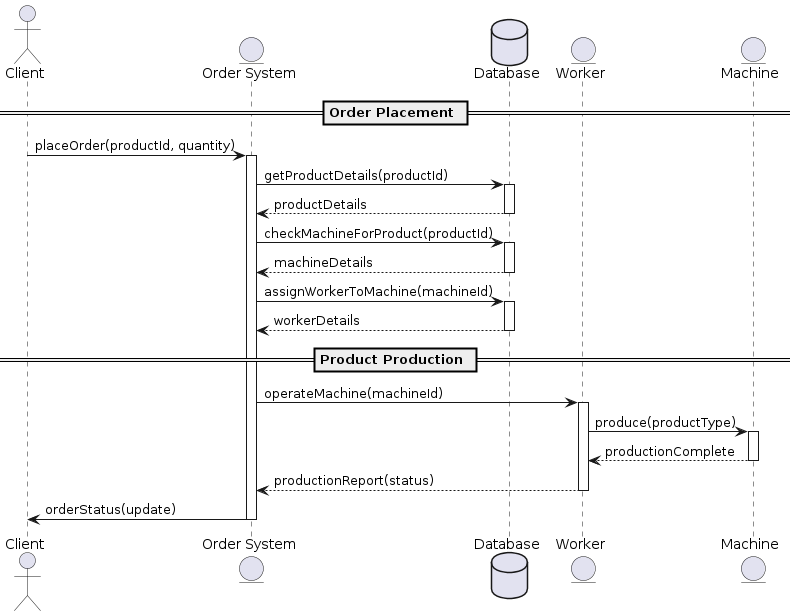
Worker --> OrderSystem : productionReport(status)

deactivate Worker

OrderSystem -> Client : orderStatus(update)

deactivate OrderSystem

@enduml



# Система управления рестораном

@startuml

' ----------- Definitions of the classes -----------

class PointOfSaleSystem {

+recordPayment()

+inputOrder()

+recordVoids()

+applyPromoDeals()

}

class PaymentProcessing {

+acceptCreditCard()

+acceptDebitCard()

+acceptMobilePayment()

}

class InventoryManagement {

+monitorStock()

+calculateCost()

+showOrderDetails()

}

class EmployeeScheduling {

+checkAttendance()

+manageShifts()

+recordTimeOff()

}

class PayrollAndAccounting {

+managePayroll()

+monitorCashFlow()

}

class Employee {

-name: String

-age: int

-gender: String

}

class Section {

-name: String

}

class Order {

-details: String

-tableNumber: int

}

class Transaction {

-amount: float

-type: String

}

' ----------- Inheritances -----------

class Cook extends Employee {

+prepareOrder()

}

class Waiter extends Employee {

+serveCustomer()

}

class Cashier extends Employee {

+prepareCheck()

}

' ----------- Associations -----------

PointOfSaleSystem "1" -- "1..\*" Order : captures>

PointOfSaleSystem "1" -- "1..\*" Transaction : records>

PaymentProcessing "1" -- "1..\*" Transaction : processes>

InventoryManagement "1" -- "1..\*" Order : uses>

Employee "1" -- "1..\*" Section : works in>

Employee "1" -- "0..1" EmployeeScheduling : scheduled by>

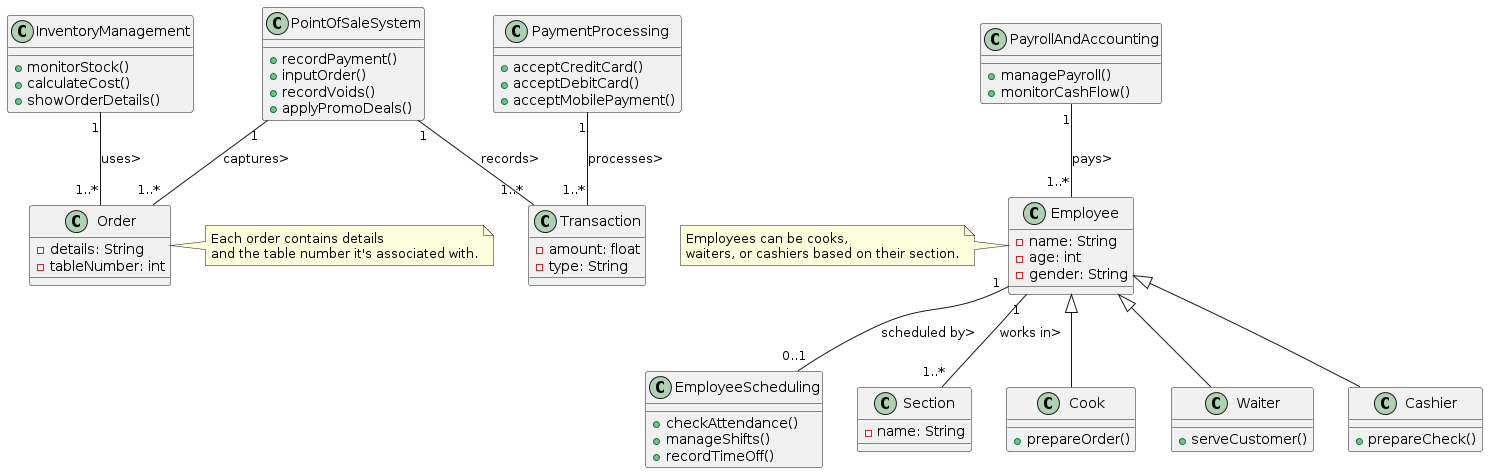
PayrollAndAccounting "1" -- "1..\*" Employee : pays>

' ----------- Comments on associations -----------

note right of Order : Each order contains details\nand the table number it's associated with.

note left of Employee : Employees can be cooks,\nwaiters, or cashiers based on their section.

@enduml



@startuml

left to right direction

actor Cook

actor Waiter

actor Cashier

actor Manager

rectangle "Restaurant Management System" {

usecase "Input Orders" as UC1

usecase "Record Payments" as UC2

usecase "Process Payment Options" as UC3

usecase "Monitor Inventory" as UC4

usecase "Prepare Orders" as UC5

usecase "Serve Customers" as UC6

usecase "Prepare Checks" as UC7

usecase "Check Employee Attendance" as UC8

usecase "Manage Shifts" as UC9

usecase "Manage Payroll" as UC10

usecase "Monitor Cash Flow" as UC11

' Connections

Cook --> UC1 : inputs >

Cook --> UC5 : prepares >

Waiter --> UC1 : inputs >

Waiter --> UC6 : serves >

Cashier --> UC2 : records >

Cashier --> UC3 : processes >

Cashier --> UC7 : prepares >

Manager --> UC8 : checks >

Manager --> UC9 : manages >

Manager --> UC10 : manages >

Manager --> UC11 : monitors >

Manager --> UC4 : monitors >

' Additional relations

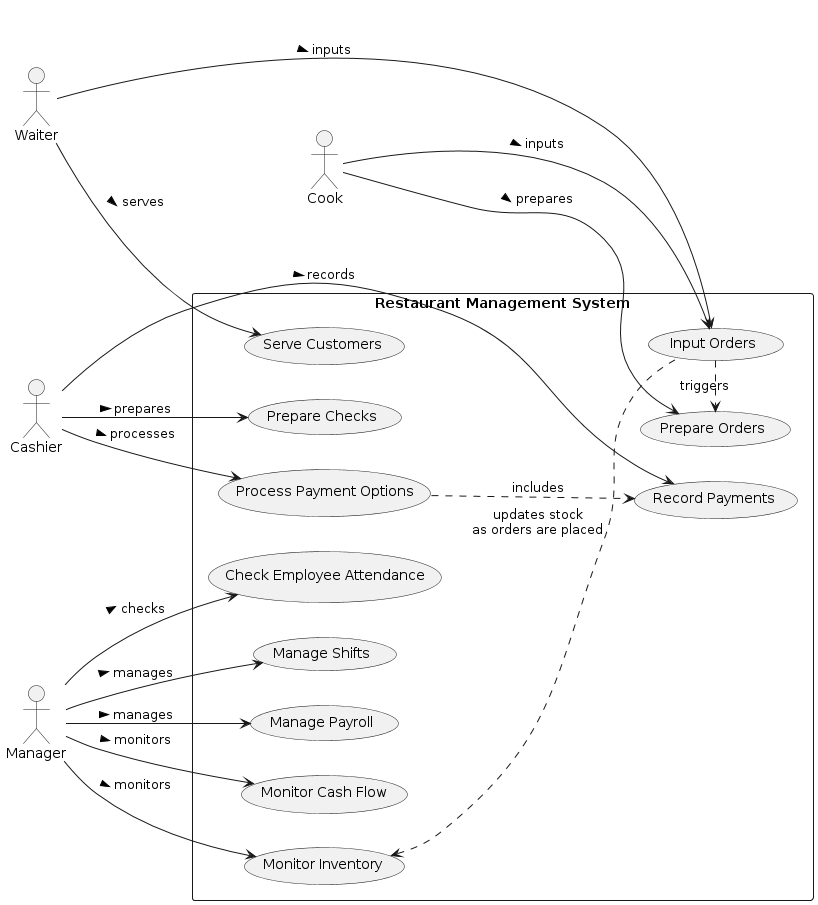
UC1 .up.> UC4 : "updates stock\nas orders are placed"

UC1 .right.> UC5 : "triggers"

UC3 .down.> UC2 : "includes"

}

@enduml



@startuml RestaurantManagementSequence

actor Waiter

actor Cook

actor Cashier

entity "PointOfSale System" as POS

entity "Payment Processing" as PaymentProc

entity "Inventory Management" as Inventory

entity "Payroll and Accounting" as Payroll

== Order and Payment Processing ==

Waiter -> POS : enterOrder(details, tableNumber)

activate POS

POS -> Inventory : checkStock(itemID)

activate Inventory

Inventory --> POS : stockStatus

deactivate Inventory

POS -> Cook : sendOrder(orderID)

activate Cook

Cook -> Inventory : updateStockUsage(itemID, quantity)

Inventory --> Cook : confirmation

Cook --> POS : orderPrepared(orderID)

deactivate Cook

POS -> Waiter : alertOrderReady(tableNumber)

deactivate POS

Waiter -> Customer : serveOrder()

Waiter -> POS : initiatePayment(tableNumber)

activate POS

Waiter -> PaymentProc : processPayment(method, amount)

activate PaymentProc

PaymentProc --> Waiter : paymentStatus

deactivate PaymentProc

== Payroll Management ==

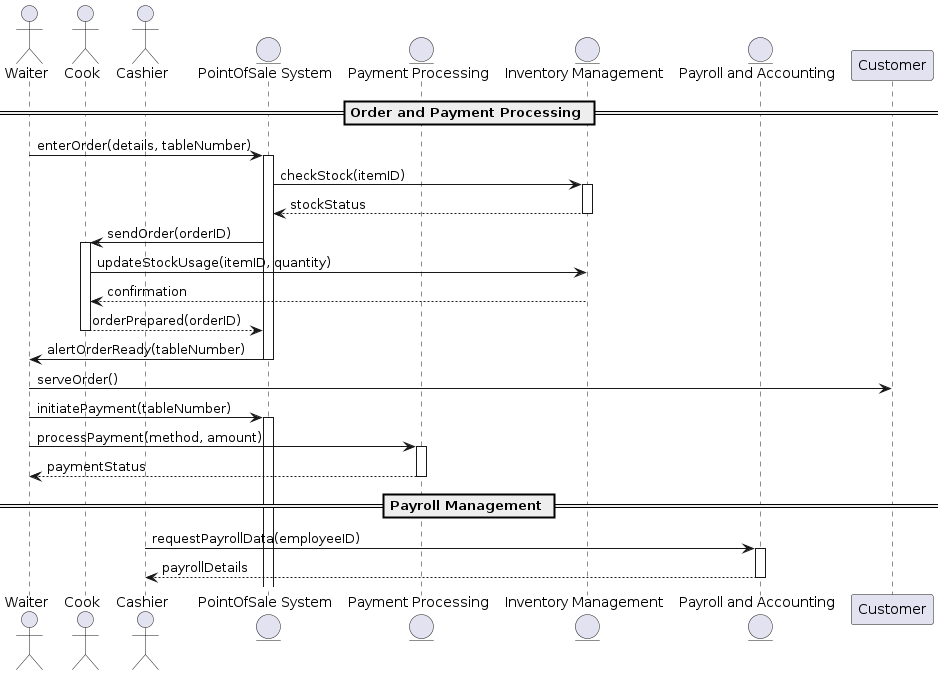
Cashier -> Payroll : requestPayrollData(employeeID)

activate Payroll

Payroll --> Cashier : payrollDetails

deactivate Payroll

@enduml



# Система бронирования в отеле

@startuml

class Room {

+roomNumber: int

+roomType: String

+capacity: int

+seasonalRates: Map<String, Double>

+calculatePrice(numberOfGuests: int, season: String): double

}

class Reservation {

+reservationDate: Date

+room: Room

+customer: Customer

+numberOfGuests: int

+reservationType: String

+calculateDiscount(): double

+applyAdditionalCharges(): void

}

class Customer {

+customerId: int

+name: String

+registered: boolean

+register(): void

}

class Payment {

+paymentType: String

+amount: double

+processPayment(): void

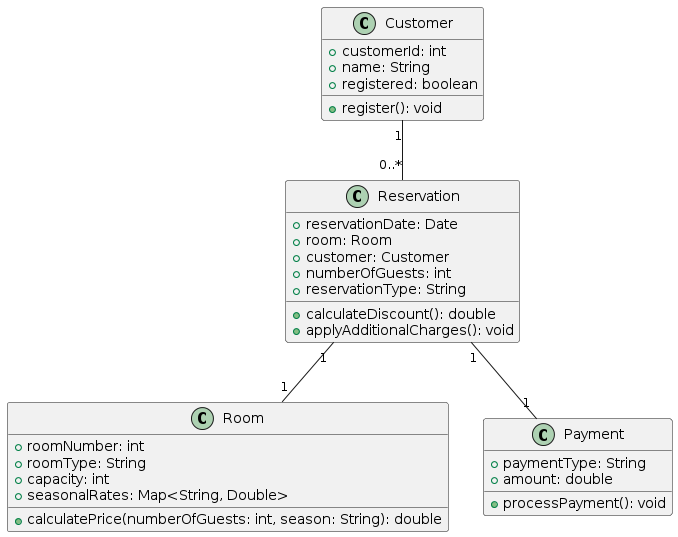
}

Customer "1" -- "0..\*" Reservation

Reservation "1" -- "1" Room

Reservation "1" -- "1" Payment

@enduml



@startuml

left to right direction

actor Customer

usecase "Register" as UC1

usecase "Make Reservation" as UC2

usecase "Make Payment" as UC3

usecase "Choose Payment Type" as UC4

usecase "Calculate Final Price" as UC5

Customer --> UC1

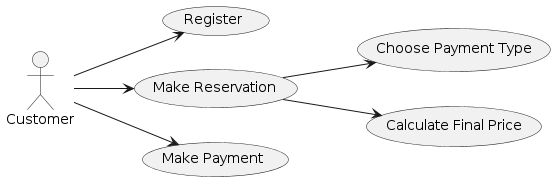
Customer --> UC2

Customer --> UC3

UC2 --> UC4

UC2 --> UC5

@enduml



@startuml

actor Customer

participant "Reservation System" as RS

participant "Payment System" as PS

Customer -> RS: requestReservation(data)

RS -> RS: validateAvailability()

alt room is available

RS -> Customer: showRoomOptions()

Customer -> RS: selectRoom(roomType, numberOfGuests, season)

RS -> RS: calculatePrice()

RS -> Customer: showPrice(price)

Customer -> RS: choosePaymentMethod(method)

RS -> PS: initiatePayment(price, method)

PS -> PS: calculateFinalAmount(discounts)

PS -> Customer: processPayment()

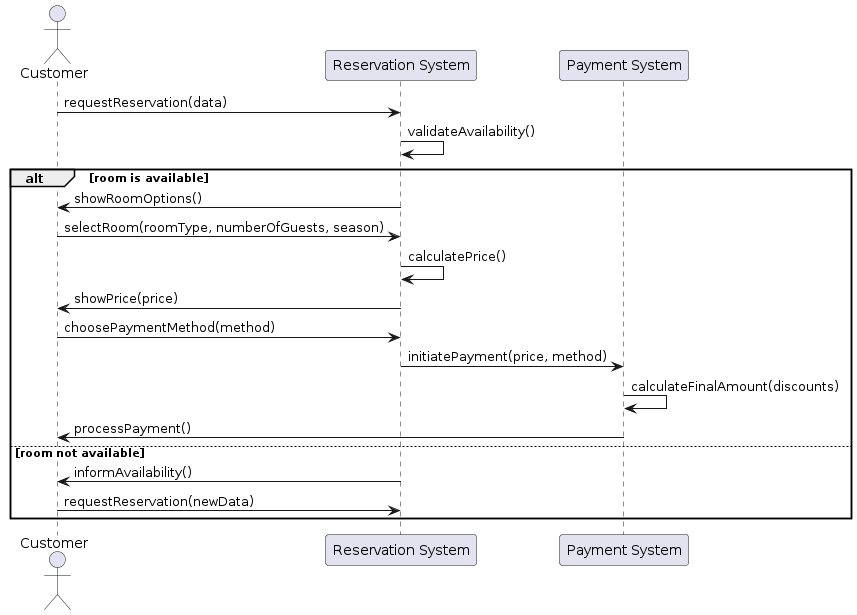
else room not available

RS -> Customer: informAvailability()

Customer -> RS: requestReservation(newData)

end

@enduml



# Система для благотворительных пожертвований

@startuml

class Customer {

+String username

+String password

+checkEligibility(): boolean

}

class Account {

+String accountID

+double balance

+AccountType type

+isActive(): boolean

}

class Donation {

+Account account

+CharityOrganization organization

+double amount

+confirmDonation(): boolean

+sendOTP(): String

+verifyOTP(String): boolean

+createEReceipt(): EReceipt

}

class CharityOrganization {

+String organizationID

+String name

+List<Account> donationAccounts

}

class BankStaff {

+String employeeID

+String name

+modifyCharityList(CharityOrganization, boolean): void

}

class EReceipt {

+String receiptID

+Date date

+String accountDetails

+String donationDetails

}

enum AccountType {

CHECKING

SAVINGS

}

class BankingSystem {

+List<CharityOrganization> charityOrganizations

+transferMoney(Account, CharityOrganization, double): boolean

}

class MobileApp {

+Customer loggedInCustomer

+showOrganizations(): List<CharityOrganization>

+donate(Donation): void

}

class WebApp {

+Customer loggedInCustomer

+showOrganizations(): List<CharityOrganization>

+donate(Donation): void

}

BankingSystem "1" -- "\*" CharityOrganization : manages

Customer "1" -- "\*" Donation : makes >

Donation "1" -- "1" Account

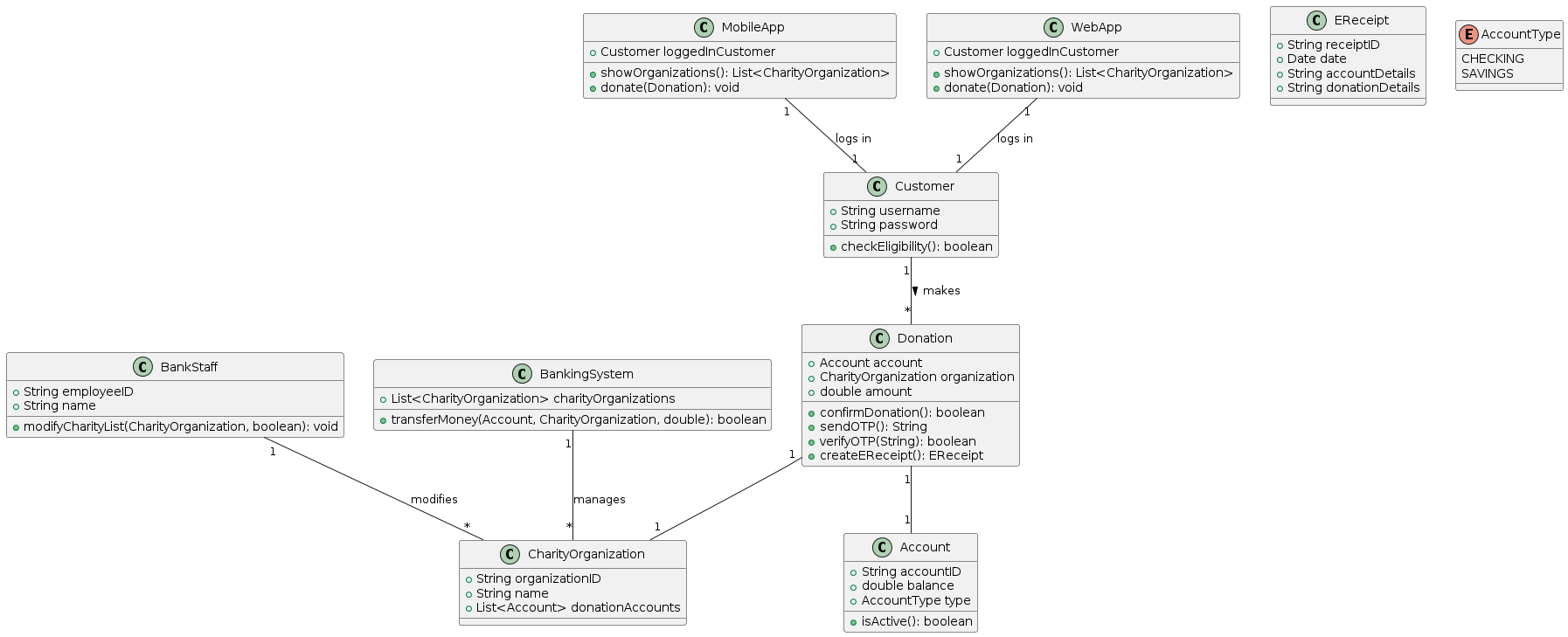
Donation "1" -- "1" CharityOrganization

BankStaff "1" -- "\*" CharityOrganization : modifies

MobileApp "1" -- "1" Customer : logs in

WebApp "1" -- "1" Customer : logs in

@enduml



@startuml

left to right direction

skinparam packageStyle rectangle

actor Customer

actor BankStaff

rectangle "Donation System" {

usecase "View Charity Organizations" as UC1

usecase "Make Donation" as UC2

usecase "Confirm Donation" as UC3

usecase "Receive OTP" as UC4

usecase "Verify OTP" as UC5

usecase "View E-Receipt" as UC6

usecase "Manage Charity Organizations" as UC7

Customer --> UC1 : selects

Customer --> UC2 : initiates

Customer --> UC3 : confirms

Customer --> UC4 : receives

Customer --> UC5 : enters

Customer --> UC6 : views

BankStaff --> UC7 : updates

UC2 .> UC3 : <<extends>>

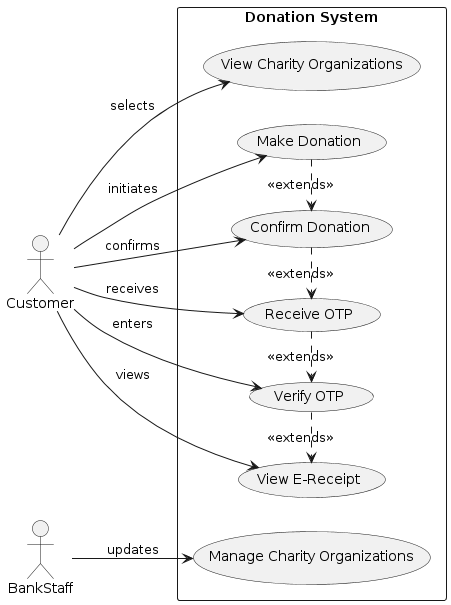
UC3 .> UC4 : <<extends>>

UC4 .> UC5 : <<extends>>

UC5 .> UC6 : <<extends>>

}

@enduml



@startuml

actor Customer

entity "Mobile App" as MobileApp

entity "Banking System" as BankingSystem

entity "SMS Gateway" as SMSGateway

Customer -> MobileApp : Selects "Make Donation"

MobileApp -> Customer : Displays charity organizations

Customer -> MobileApp : Chooses charity and amount

MobileApp -> BankingSystem : Requests donation setup(account, charity, amount)

BankingSystem -> MobileApp : Confirms transaction details

MobileApp -> Customer : Displays confirmation screen

Customer -> MobileApp : Confirms donation

alt successful confirmation

MobileApp -> SMSGateway : Request OTP

SMSGateway -> Customer : Sends OTP

Customer -> MobileApp : Enters OTP

MobileApp -> BankingSystem : Verifies OTP

BankingSystem -> MobileApp : OTP verified

BankingSystem -> BankingSystem : Process donation

BankingSystem -> MobileApp : Donation successful

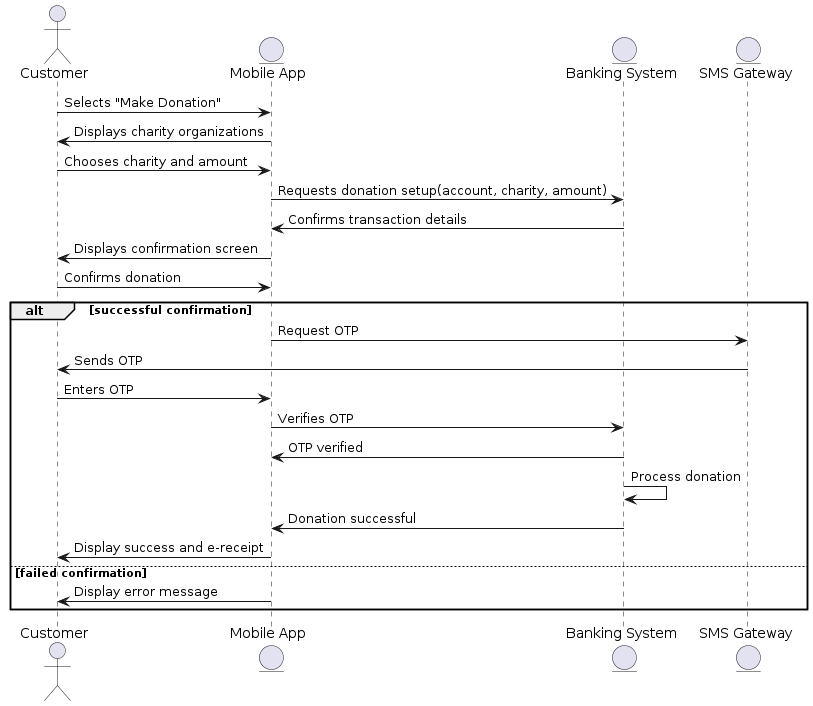
MobileApp -> Customer : Display success and e-receipt

else failed confirmation

MobileApp -> Customer : Display error message

end

@enduml



# Система управления сотрудниками

@startuml

' Abstract class Employee

class Employee {

- name: String

- dateOfBirth: Date

- socialInsuranceNumber: String

- department: Department

+ printInformation(): void

+ checkRetirement(): boolean

}

' Specific Employee types

class Worker extends Employee {

- weeklyWorkingHours: double

- hourlyRate: double

}

class Salesperson extends Employee {

- fixedSalary: double

- salesAmount: double

- commissionPercentage: double

}

class Manager extends Employee {

- position: String

- fixedSalary: double

+ printInformation(): void

+ checkRetirement(): boolean

}

' Subtypes of Worker

class ShiftWorker extends Worker {

- holidayPremium: double

}

class NonShiftWorker extends Worker {

- hasWeekendPermits: boolean

- hasHolidayPermits: boolean

}

' Department class

class Department {

- name: String

- type: String

+ managedBy: Manager

}

' Manager to Department association

Manager "1" --> "\*" Department : manages

' Employee to Department association

Employee "1" --> "1" Department

' Test class

class TestSystem {

+ initializeData(): void

+ outputRetirementEligibleEmployees(): void

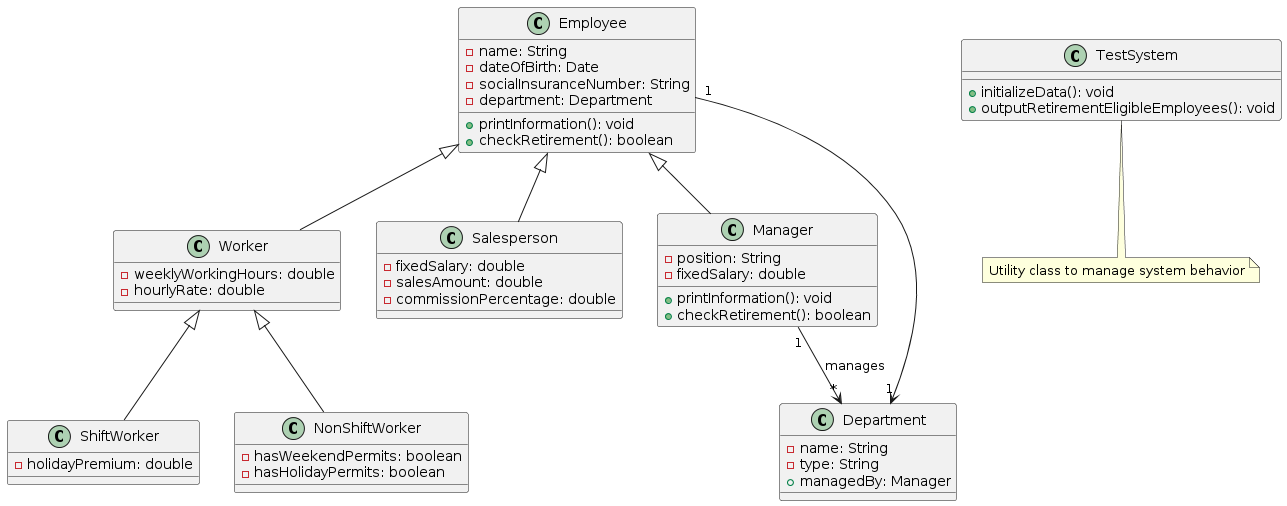
}

' Utility and Relationships

note "Utility class to manage system behavior" as Note1

TestSystem .. Note1

@enduml



@startuml

left to right direction

skinparam packageStyle rectangle

' Define actors

actor Employee

actor Manager

actor SystemAdministrator

' Define system boundary

rectangle "Employee Management System" {

' Use cases

usecase "Log In" as UC1

usecase "Check Retirement Status" as UC2

usecase "Print Employee Information" as UC3

usecase "Manage Departments" as UC4

usecase "Initialize System Data" as UC5

usecase "View Worker Schedules" as UC6

usecase "Update Employee Data" as UC7

' Actor interactions

Employee --> UC1 : Uses

Employee --> UC2 : Requests

Employee --> UC3 : Requests

Manager --> UC1 : Uses

Manager --> UC2 : Requests

Manager --> UC3 : Requests

Manager --> UC4 : Manages

Manager --> UC6 : Views

SystemAdministrator --> UC5 : Initializes

SystemAdministrator --> UC7 : Updates

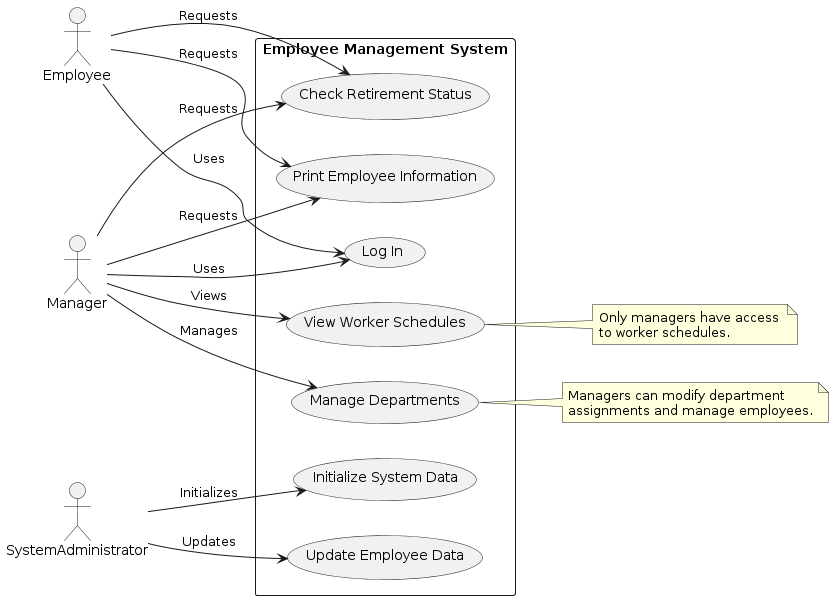
}

' Notes

note right of UC4 : Managers can modify department \nassignments and manage employees.

note right of UC6 : Only managers have access \nto worker schedules.

@enduml



@startuml

' Define participants

actor Employee

entity "Employee Management System" as EMS

entity Manager

' Start of the sequence

Employee -> EMS : Log In

activate EMS

EMS -> Employee : Authentication Success

Employee -> EMS : Request Retirement Status

activate Employee

EMS -> Manager : Verify Employee Age

activate Manager

Manager -> EMS : Return Age Verification

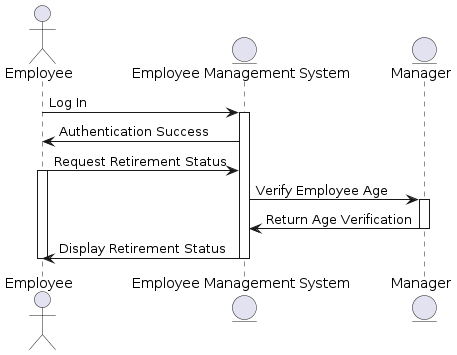
deactivate Manager

EMS -> Employee : Display Retirement Status

deactivate Employee

deactivate EMS

@enduml



# Система автопилот

@startuml

class RollAutopilotEngageControl {

+engageAutopilot()

+disengageAutopilot()

+sendCommandToActuator()

}

class PilotControl {

+autopilotSwitch: boolean

+toggleAutopilot()

}

class RollActuator {

+currentCommand: int

+setCommand(command : int)

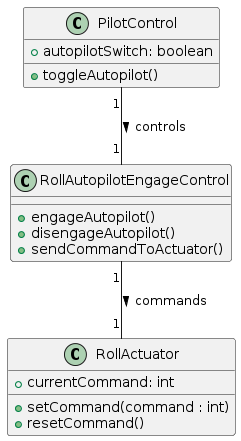
+resetCommand()

}

PilotControl "1" -- "1" RollAutopilotEngageControl : controls >

RollAutopilotEngageControl "1" -- "1" RollActuator : commands >

@enduml



@startuml

left to right direction

actor Pilot as "Pilot"

package "Roll Autopilot System" {

usecase "Engage Autopilot" as Engage

usecase "Disengage Autopilot" as Disengage

usecase "Zero Command to Roll Actuator" as Zero

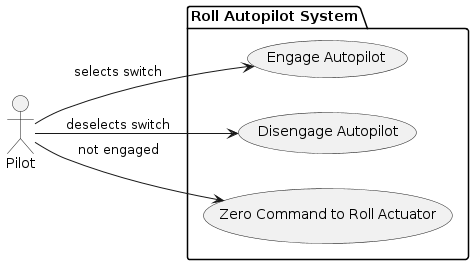
Pilot --> Engage : selects switch

Pilot --> Disengage : deselects switch

Pilot --> Zero : not engaged

}

@enduml



@startuml

actor Pilot as "Pilot"

entity "Control System" as ControlSystem

entity "Roll Actuator" as RollActuator

Pilot -> ControlSystem : Toggle Autopilot Engage Switch ON

activate ControlSystem

ControlSystem -> RollActuator : sendCommand(engage)

activate RollActuator

RollActuator --> ControlSystem : Command Acknowledged

deactivate RollActuator

ControlSystem --> Pilot : Autopilot Engaged

deactivate ControlSystem

Pilot -> ControlSystem : Toggle Autopilot Engage Switch OFF

activate ControlSystem

ControlSystem -> RollActuator : sendCommand(disengage)

activate RollActuator

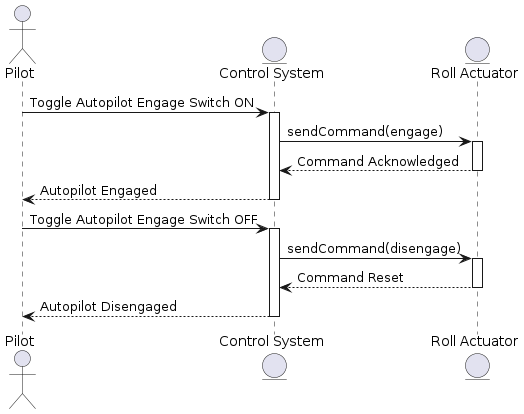
RollActuator --> ControlSystem : Command Reset

deactivate RollActuator

ControlSystem --> Pilot : Autopilot Disengaged

deactivate ControlSystem

@enduml



# Система моделирования городского траффика

@startuml

class User {

+userID: int

+username: String

+password: String

+login(): void

+logout(): void

}

class TrafficSystem {

+displayTrafficFlows(): void

+showEventImpact(): void

+updateTrafficFlows(): void

+generateReport(): Report

}

class TrafficFlow {

+flowID: int

+status: String

+color: String

+getColorBasedOnStatus(): String

}

class Event {

+eventID: int

+location: String

+expectedVisitors: int

+createEvent(): void

+editEvent(): void

}

class TrafficManagement {

+solutionID: int

+strategy: String

+applySolution(): void

+modifyTrafficFlow(flow: TrafficFlow): void

}

class Report {

+reportID: int

+content: String

+generateContent(): void

+retrieveReport(): String

}

User --> TrafficSystem

TrafficSystem --> TrafficFlow

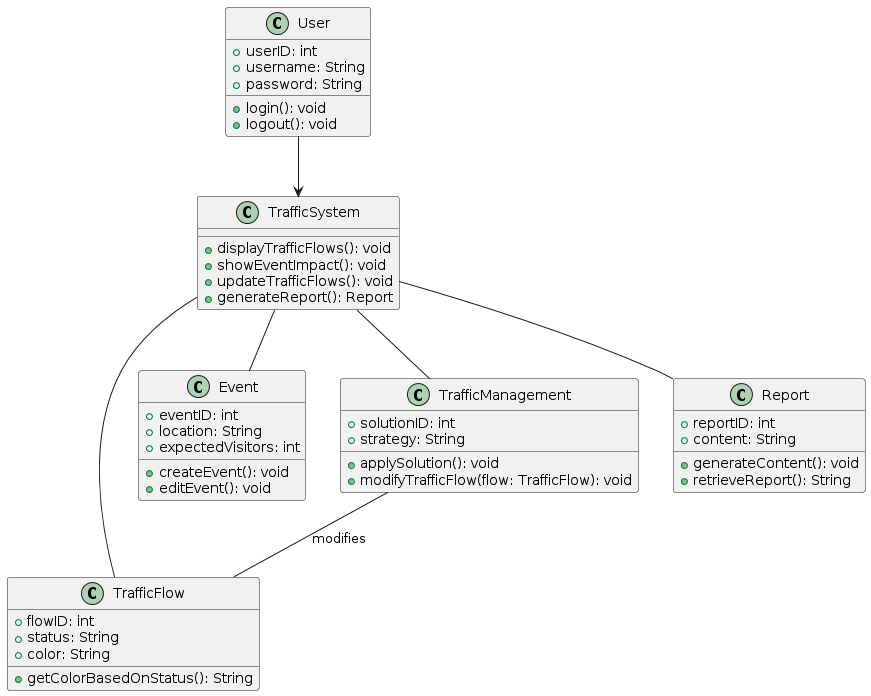
TrafficSystem --> Event

TrafficSystem --> TrafficManagement

TrafficSystem --> Report

TrafficManagement --> TrafficFlow : modifies

@enduml



@startuml

left to right direction

skinparam packageStyle rectangle

actor User

rectangle "Traffic Management System" {

usecase "Authenticate User" as UC1

usecase "View Traffic Flows" as UC2

usecase "Highlight Congested Flows" as UC2\_1

usecase "Specify Event" as UC3

usecase "View Event Impact on Traffic" as UC4

usecase "Manage Traffic Flows" as UC5

usecase "View Updated Traffic Flows" as UC6

usecase "Retrieve Traffic Report" as UC7

UC2\_1 .u.> UC2 : extends

User --> UC1 : logs in

User --> UC2 : views

User --> UC3 : specifies

User --> UC4 : views impact

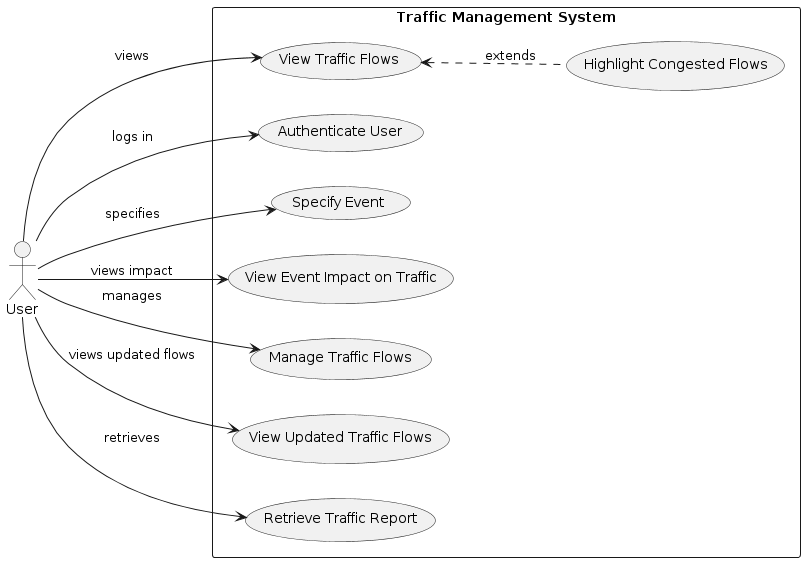
User --> UC5 : manages

User --> UC6 : views updated flows

User --> UC7 : retrieves

}

@enduml



@startuml

actor User

participant "Login System" as Login

participant "Traffic System" as Traffic

participant "Event System" as Event

participant "Management System" as Management

participant "Report System" as Report

User -> Login : provide credentials

activate Login

Login -> User : grant access

deactivate Login

User -> Traffic : request traffic data

activate Traffic

Traffic -> Traffic : retrieve historic data

Traffic -> Traffic : apply color coding for congestion

Traffic --> User : display traffic flows

deactivate Traffic

User -> Event : specify event details

activate Event

Event -> Event : record event

Event --> Traffic : trigger traffic analysis

activate Traffic

Traffic -> Traffic : calculate pre-event traffic

Traffic -> Traffic : calculate post-event traffic

Traffic --> User : show event impact

deactivate Traffic

deactivate Event

User -> Management : adjust traffic flow

activate Management

Management -> Traffic : update traffic settings

activate Traffic

Traffic -> Traffic : process changes

Traffic --> User : update traffic display

deactivate Traffic

deactivate Management

User -> Report : request traffic report

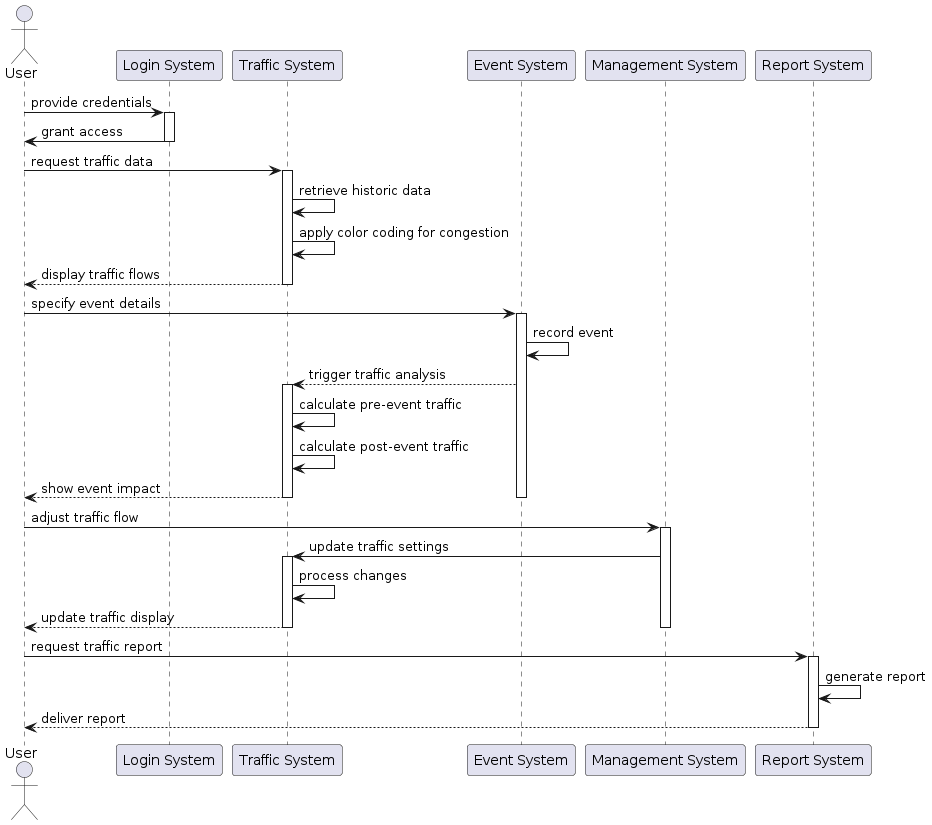
activate Report

Report -> Report : generate report

Report --> User : deliver report

deactivate Report

@enduml



# Система отказоустойчивости воздушного судна

@startuml

class Computer {

- softwareApplications: List<Application>

- receiveInput(data: InputData): void

}

class InputSensor {

- sensorData: List<InputData>

+ readData(): InputData

}

class RedundancyManager {

- inputSet: List<InputData>

- failureDetector: FailureDetector

- failureReporter: FailureReporter

- signalSelector: SignalSelector

+ monitorInputs(): void

+ manageRedundancy(): void

}

class FailureDetector {

- tripLevel: double

- persistenceLimit: int

+ detectFailure(inputs: List<InputData>): FailureStatus

}

class FailureReporter {

+ reportFailure(status: FailureStatus): void

}

class SignalSelector {

+ selectSignal(inputs: List<InputData>): InputData

}

Computer "1" \*-- "3" InputSensor : gets data from >

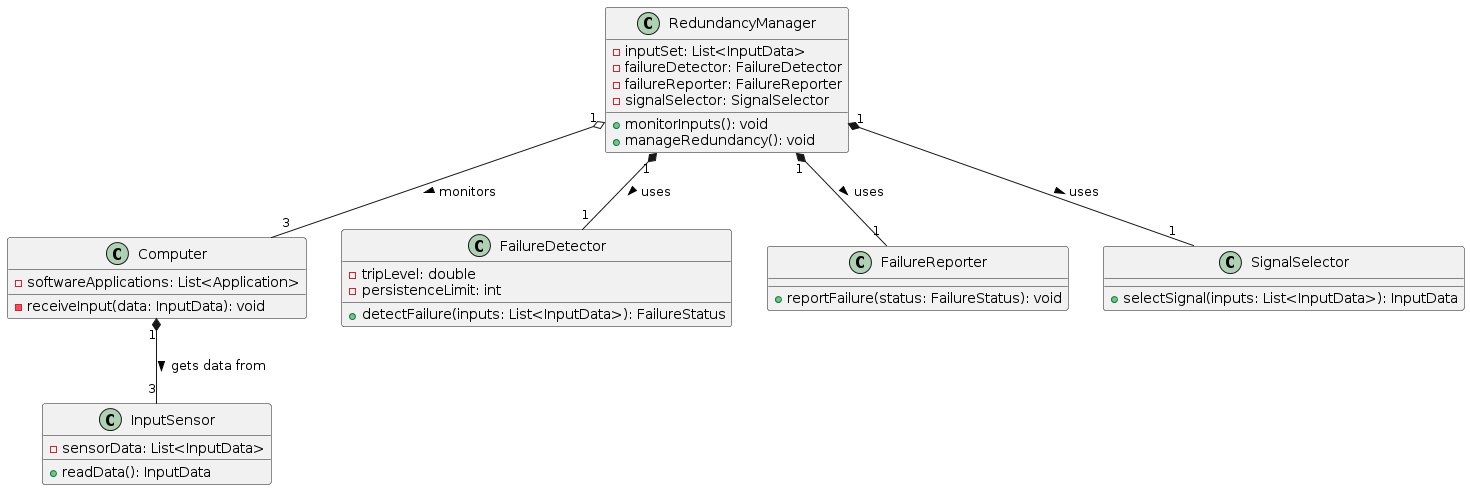
RedundancyManager "1" \*-- "1" FailureDetector : uses >

RedundancyManager "1" \*-- "1" FailureReporter : uses >

RedundancyManager "1" \*-- "1" SignalSelector : uses >

RedundancyManager "1" o-- "3" Computer : monitors >

@enduml



@startuml

left to right direction

actor Sensor as "Sensor"

actor Computer as "Computer"

actor "Failure Management System" as FailureManagement

rectangle "Redundancy Management System" {

usecase "Monitor Inputs" as UC1

usecase "Report Failure" as UC2

usecase "Select Signal" as UC3

}

Sensor --> UC1 : provides data

Computer --> UC1 : executes monitoring

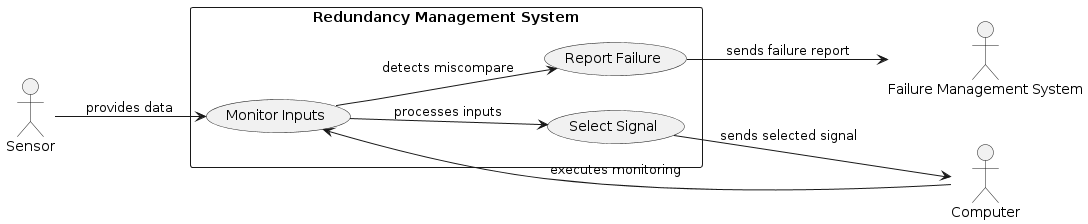
UC1 --> UC2 : detects miscompare

UC2 --> FailureManagement : sends failure report

UC1 --> UC3 : processes inputs

UC3 --> Computer : sends selected signal

@enduml



@startuml

participant InputSensor as "Sensor"

participant Computer

participant RedundancyManager as "Redundancy Manager"

participant FailureDetector as "Failure Detector"

participant FailureReporter as "Failure Reporter"

participant SignalSelector as "Signal Selector"

participant FailureManagementSystem as "Failure Management System"

== Input Processing ==

Sensor -> Computer : send(triplexInput)

Computer -> RedundancyManager : manageInput(triplexInput)

== Failure Detection ==

RedundancyManager -> FailureDetector : checkFailure(triplexInput)

activate FailureDetector

FailureDetector -> FailureDetector : analyzeInputs()

alt Failure Detected

FailureDetector -> FailureReporter : reportFailure()

FailureReporter -> FailureManagementSystem : notifyFailure()

end

deactivate FailureDetector

== Signal Selection ==

alt No Failure

RedundancyManager -> SignalSelector : selectSignal(triplexInput)

SignalSelector -> Computer : return(midValue)

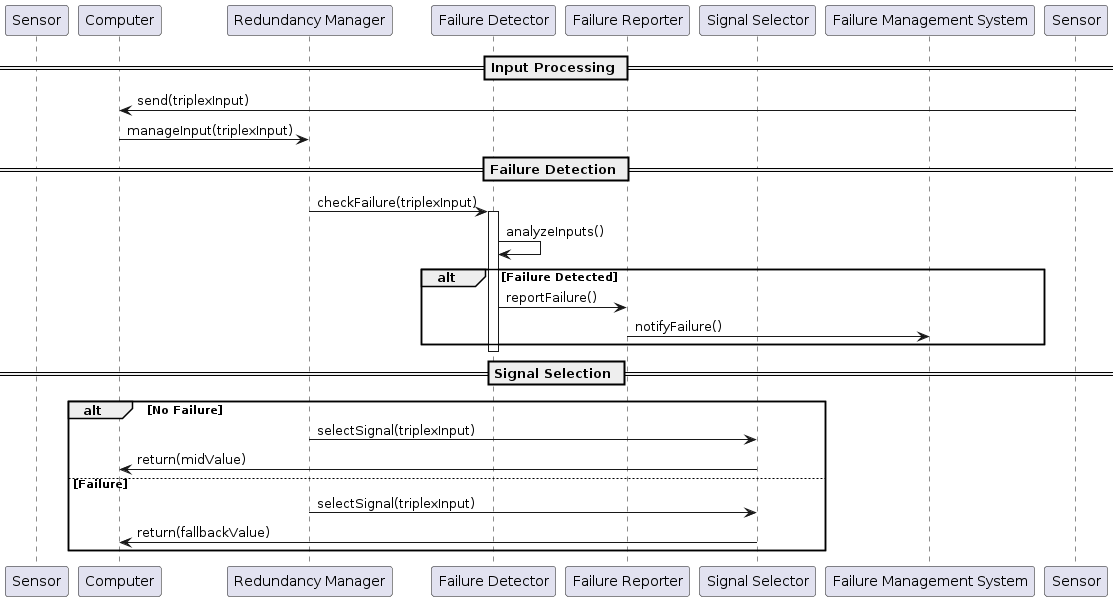
else Failure

RedundancyManager -> SignalSelector : selectSignal(triplexInput)

SignalSelector -> Computer : return(fallbackValue)

end

@enduml



# Портал футбольной ассоциации

@startuml

class User {

-userId: int

-username: String

-password: String

}

class TeamManager {

-teamId: int

}

class IFAAdmin {

}

class IFAAuditor {

}

class IFABudgetManager {

}

class Referee {

-refereeId: int

}

class SubReferee {

-subRefereeId: int

}

class IFASchedulingCommitteeMember {

}

class Fan {

-follows: List<Team>

}

class Team {

-teamId: int

-players: List<Player>

}

class Player {

-playerId: int

}

class Budget {

-amount: double

}

class Report {

-content: String

-status: String

}

class Schedule {

-date: Date

-matches: List<Match>

}

class Match {

-matchId: int

-result: String

}

class SystemInterface {

-languages: List<String>

-currencies: List<String>

}

class Alert {

-message: String

}

User <|-- TeamManager

User <|-- IFAAdmin

User <|-- IFAAuditor

User <|-- IFABudgetManager

User <|-- Referee

User <|-- SubReferee

User <|-- IFASchedulingCommitteeMember

User <|-- Fan

TeamManager "1" -- "\*" Team : manages

TeamManager "1" -- "\*" Budget : allocates

TeamManager "1" -- "\*" Report : uploads

IFABudgetManager "1" -- "\*" Budget : assigns

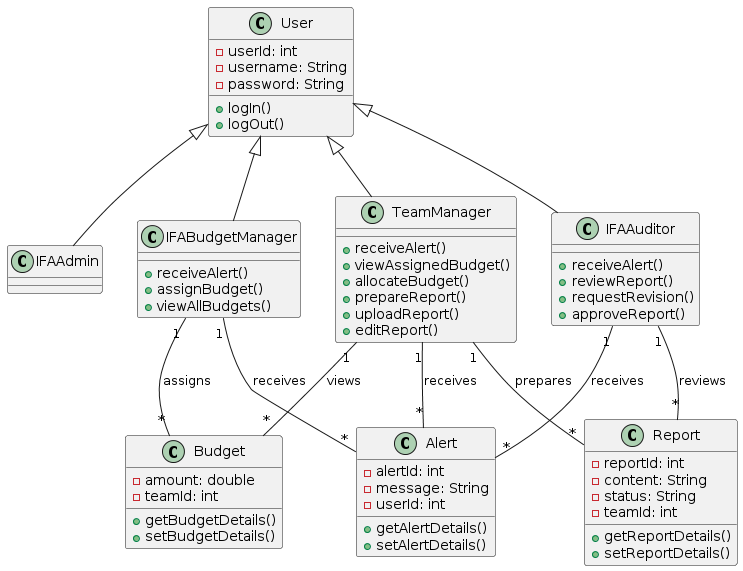
IFAAuditor "1" -- "\*" Report : reviews

Referee "1" -- "\*" Match : officiates

SubReferee "1" -- "\*" Match : assists

Match "\*" -- "1" Schedule

@enduml



@startuml

left to right direction

actor "IFA Budget Manager" as IFABudgetManager

actor "Team Manager" as TeamManager

actor "IFA Auditor" as IFAAuditor

rectangle "Budget and Reporting" {

usecase "Receive Budget Alert" as UC1

usecase "Assign Budget to Teams" as UC2

usecase "View Assigned Budget" as UC3

usecase "Allocate Budget to Activities" as UC4

usecase "Prepare Budget Report" as UC5

usecase "Upload Budget Report" as UC6

usecase "Receive Report Alert" as UC7

usecase "Review Budget Report" as UC8

usecase "Request Report Revision" as UC9

usecase "Edit and Re-upload Budget Report" as UC10

usecase "Approve Budget Report" as UC11

IFABudgetManager -right-> UC1 : receives

IFABudgetManager -down-> UC2 : assigns

TeamManager -down-> UC3 : views

TeamManager -down-> UC4 : allocates

TeamManager -down-> UC5 : prepares

TeamManager -down-> UC6 : uploads

IFABudgetManager -left-> UC7 : receives

IFABudgetManager -left-> UC8 : reviews

IFABudgetManager -left-> UC9 : requests

TeamManager -left-> UC10 : edits

IFABudgetManager -left-> UC11 : approves

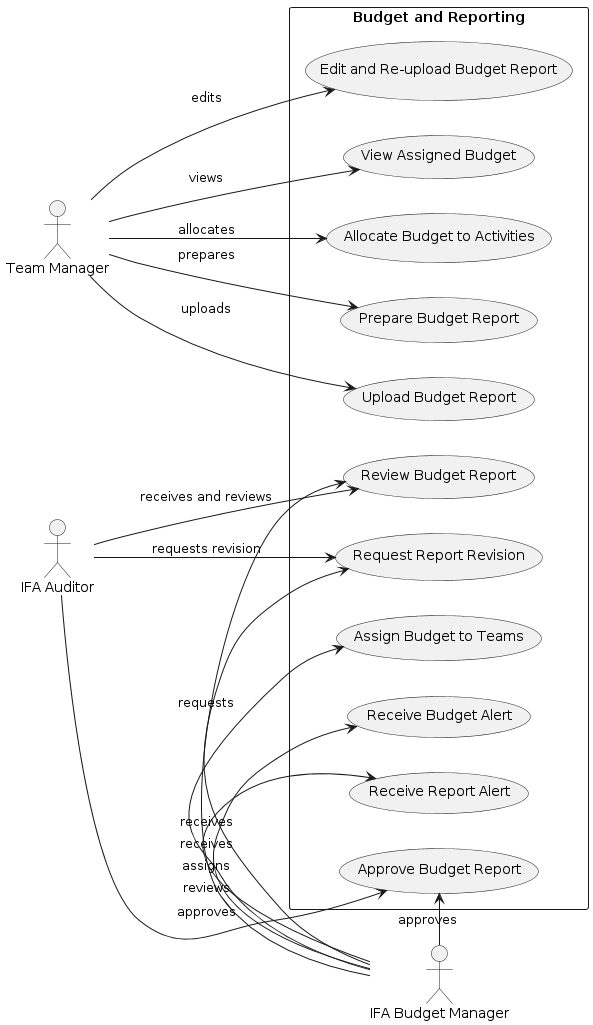
}

IFAAuditor -down-> UC8 : receives and reviews

IFAAuditor -down-> UC9 : requests revision

IFAAuditor -down-> UC11 : approves

@enduml



@startuml

actor "IFA Budget Manager" as IFABudgetManager

actor "Team Manager" as TeamManager

actor "IFA Auditor" as IFAAuditor

entity "Budget System" as BudgetSystem

== Budget Assignment Process ==

IFABudgetManager -> BudgetSystem : Log in

activate BudgetSystem

BudgetSystem --> IFABudgetManager : Authentication Success

IFABudgetManager -> BudgetSystem : Receive Alert

activate BudgetSystem

BudgetSystem --> IFABudgetManager : Show Alert Details

IFABudgetManager -> BudgetSystem : Assign Budget to Teams

activate BudgetSystem

BudgetSystem -> TeamManager : Send Budget Alert

deactivate BudgetSystem

== Team Manager Budget Allocation ==

TeamManager -> BudgetSystem : Log in

activate BudgetSystem

BudgetSystem --> TeamManager : Authentication Success

TeamManager -> BudgetSystem : View Assigned Budget

activate BudgetSystem

BudgetSystem --> TeamManager : Display Budget

TeamManager -> BudgetSystem : Allocate Budget to Activities

BudgetSystem --> TeamManager : Confirmation

TeamManager -> BudgetSystem : Prepare and Upload Report

BudgetSystem -> IFAAuditor : Send Report Alert

deactivate BudgetSystem

== Audit and Approval Process ==

IFAAuditor -> BudgetSystem : Log in

activate BudgetSystem

BudgetSystem --> IFAAuditor : Authentication Success

IFAAuditor -> BudgetSystem : Review Budget Report

BudgetSystem --> IFAAuditor : Display Report

IFAAuditor -> BudgetSystem : Request Report Revision

BudgetSystem -> TeamManager : Send Revision Alert

TeamManager -> BudgetSystem : Edit and Re-upload Report

activate BudgetSystem

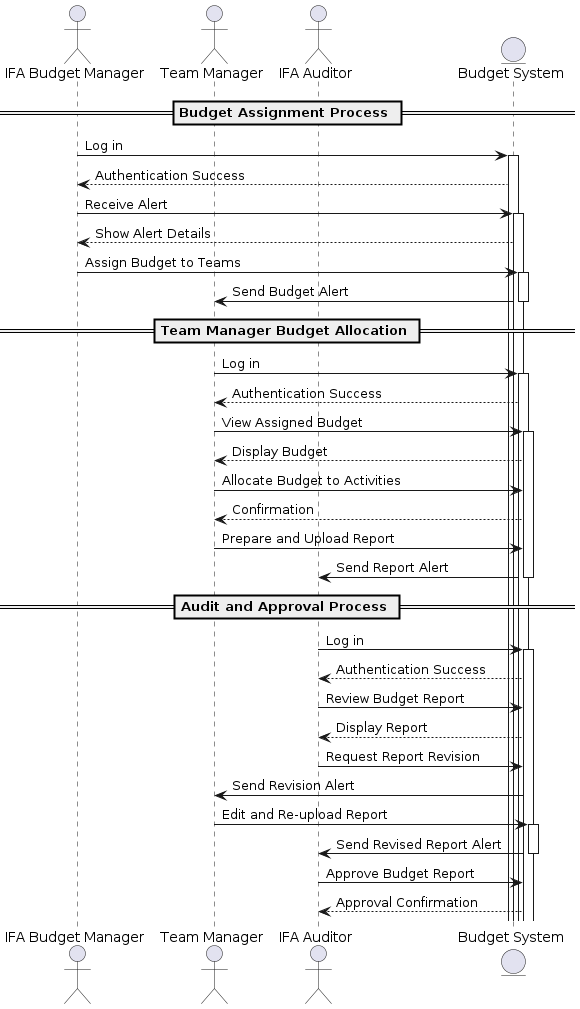
BudgetSystem -> IFAAuditor : Send Revised Report Alert

deactivate BudgetSystem

IFAAuditor -> BudgetSystem : Approve Budget Report

BudgetSystem --> IFAAuditor : Approval Confirmation

@enduml



Generate PlantUML code for a class diagram from requirements. Diagrams should be detailed, adherent to UML syntax and consistent with system requirements. Requirements:

Generate PlantUML code for a use case diagram from requirements. The diagram should be detailed, adherent to UML syntax and consistent with system requirements. Requirements:

Generate PlantUML code for a sequence diagram from requirements. The diagram should be detailed, adherent to UML syntax and consistent with system requirements. Requirements: