```
-- TaskManager OCL Constraints

-- Class: User
context User
inv validEmail: self.email.matches('[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,4}')
inv validPassword: self.password.size() >= 8
inv uniqueUsername: User.allInstances()->forAll(u1, u2 | u1 <> u2 implies u1.username <> u2.username)
inv uniqueEmail: User.allInstances()->forAll(u1, u2 | u1 <> u2 implies u1.email <> u2.email)
inv validRole: Set{'MANAGER', 'EMPLOYEE'}->includes(self.role)
inv validStatus: Set{'ACTIVE', 'INACTIVE', 'ON_LEAVE'}->includes(self.status)

-- Class: Employee (inherits from User)
context Employee
inv validDepartment: self.department <> null and self.department.size() > 0
inv validPosition: self.position <> null and self.position.size() > 0
inv taskLimit: self.assignedTasks->select(t | t.status <> 'Completed' and t.status <> 'Canceled')->size() <= 10

-- Class: Manager (inherits from User)
context Manager
inv managesEmployees: self.managedEmployees->notEmpty()
inv departmentConsistency: self.managedEmployees->forAll(e | e.department = self.department)
inv cannotManageSelf: not self.managedEmployees->includes(self)

-- Class: Task
context Task
inv validTaskTitle: self.title <> null and self.title.size() > 0
inv validDescription: self.description <> null
inv validDueDate: self.dueDate >= self.createdDate
inv assigneeNotNull: self.assignee <> null
inv assignedToNotNull: self.assignedTo <> null

inv statusNotNull: self.status <> null

inv priorityCheck: self.priority <> null and Set{'Low', 'Medium', 'High', 'Urgent'}-
>includes(self.priority)

inv validStatusValues: Set{'Not Started', 'In Progress', 'Under Review', 'Completed', 'Canceled'}-
>includes(self.status)


-- Class: TaskStatus

context TaskStatus

inv validStatusValues: Set{'Not Started', 'In Progress', 'Under Review', 'Completed', 'Canceled'}-
>includes(self.value)

inv statusHistory: self.previousStatus <> null implies self.dateChanged <> null

inv statusSequence: (self.value = 'Completed' or self.value = 'Canceled') implies

        self.task.statusHistory->exists(sh | sh.value = 'Under Review')


-- Class: Department

context Department

inv uniqueName: Department.allInstances()->forAll(d1, d2 | d1 <> d2 implies d1.name <>
d2.name)

inv hasManager: self.managers->notEmpty()


-- Class: Calendar

context Calendar

inv belongsToUser: self.owner <> null

inv validEvents: self.events->forAll(e | e.startTime < e.endTime)

inv taskIntegration: self.events->select(e | e.isTaskEvent)->forAll(e |

         Task.allInstances()->exists(t | t.name = e.title and t.dueDate = e.endDate))


-- Class: Dashboard

context Dashboard

inv userSpecific: self.user <> null

inv taskStatistics: self.taskStats <> null
inv upcomingTasksCount: self.upcomingTasks->size() <= 5


-- Operations constraints

context User::login(username: String, password: String): Boolean

pre validCredentials: username.size() > 0 and password.size() > 0

post result: result = true implies User.allInstances()->exists(u | u.username = username and u.password = password and u.status = 'ACTIVE')

post loginTracked: self.lastLoginDate = Date::today()


context User::signup(userData: UserData): User

pre validData: userData.email.matches('[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,4}') and userData.password.size() >= 8

pre uniqueData: User.allInstances()->forAll(u | u.email <> userData.email and u.username <> userData.username)

post userCreated: User.allInstances()->exists(u | u.email = userData.email and u.status = 'ACTIVE')


context User::viewProfile(): UserProfile

post result: result.user = self and result.lastUpdated <= Date::today()


context User::viewTasks(): Set(Task)

post result: result = self.assignedTasks->sortedBy(dueDate)


context User::viewCalendar(): Calendar

post result: result.owner = self


context Employee::updateTaskStatus(task: Task, newStatus: String): Boolean

pre validTask: self.assignedTasks->includes(task)

pre validTransition: (task.status = 'Not Started' and newStatus = 'In Progress') or

                (task.status = 'In Progress' and (newStatus = 'Under Review' or newStatus = 'Canceled')) or

                (task.status = 'Under Review' and (newStatus = 'Completed' or newStatus = 'In Progress'))
```

post statusUpdated: task.status = newStatus

post completionCheck: newStatus = 'Completed' implies task.completionDate = Date::today()

context Employee::viewEmployee(): EmployeeProfile

post result: result.employee = self

context Employee::viewEmployeeTasks(): Set(Task)

post result: result = self.assignedTasks->sortedBy(priority)

context Manager::assignTask(task: Task, employee: Employee): Boolean

pre validAssignment: self.managedEmployees->includes(employee)

pre taskLimitCheck: employee.assignedTasks->select(t | t.status <> 'Completed' and t.status <> 'Canceled')->size() < 10

post taskAssigned: employee.assignedTasks->includes(task)

post assigneeSet: task.assignee = self.username

post assignedToSet: task.assignedTo = employee.username

context Manager::manageEmployees(): Set(Employee)

post result: result = self.managedEmployees

context Manager::addTask(taskData: TaskData): Task

pre validData: taskData.title <> null and taskData.title.size() > 0 and taskData.dueDate >= Date::today()

post taskCreated: Task.allInstances()->exists(t | t.title = taskData.title and t.assignee = self.username)

context Manager::editTask(task: Task, taskData: TaskData): Boolean

pre canEdit: task.assignee = self.username or self.managedEmployees->exists(e | e.username = task.assignedTo)

post taskUpdated: task.title = taskData.title and task.dueDate = taskData.dueDate

context Manager::removeTask(task: Task): Boolean

```
pre canRemove: task.assignee = self.username or self.managedEmployees->exists(e |
e.username = task.assignedTo)

post taskRemoved: not Task.allInstances()->includes(task)


context Manager::filterTasks(criteria: FilterCriteria): Set(Task)

post result: Task.allInstances()->select(t |

        (criteria.status = null or t.status = criteria.status) and

        (criteria.priority = null or t.priority = criteria.priority) and

        (criteria.assignedTo = null or t.assignedTo = criteria.assignedTo))


context Manager::setStatuses(department: Department, statuses: Set(String)): Boolean

pre managesDepartment: self.department = department or self.managedDepartments-
>includes(department)

pre validStatuses: statuses->forAll(s | Set{'Not Started', 'In Progress', 'Under Review',
'Completed', 'Canceled', 'On Hold', 'Delegated'}->includes(s))

post statusesSet: department.availableStatuses = statuses


context Manager::setDepartment(employee: Employee, department: Department): Boolean

pre managesEmployee: self.managedEmployees->includes(employee)

pre validDepartment: Department.allInstances()->includes(department)

post departmentSet: employee.department = department


-- Association constraints

context Task

inv assignmentConsistency: User.allInstances()->exists(u | u.username = self.assignee and
u.role = 'MANAGER') and

                User.allInstances()->exists(u | u.username = self.assignedTo and u.role =
'EMPLOYEE')
```