Task 1: Create model for "automated smart warehouse"

```
sig Node {
  id: Int,
  zone: one Zone -- Zones within the warehouse (e.g., storage, staging, shipping)
}

sig Path {
  id: Int,
  start: one Node,
  end: one Node,
  capacity: Int -- Maximum number of robots that can use the path simultaneously
}

sig Package {
  id: Int,
  weight: Int,
  location: lone Node -- Either a specific node or being carried by a robot
}

sig Robot {
  id: Int,
  capacity: Int, -- How much weight the robot can carry
  currentNode: one Node,
  destination: one Node,
  status: one Status,
  carrying: set Package, -- The packages the robot is currently carrying
  energy: Int -- Energy level of the robot
}

abstract sig Status {}
one sig Idle, Moving, Loading, Unloading, Charging extends Status {}

sig Employee {
  id: Int,
  role: one Role,
  location: one Node
}

abstract sig Role {}
one sig Supervisor, Operator extends Role {}

sig Task {
  robot: one Robot,
  packages: set Package,
  destination: one Node
}

abstract sig Zone {}
one sig Storage, Staging, Shipping extends Zone {}
```

```
-- Facts to enforce meaningful configurations
fact NonNegativeIDs {
  -- Ensure all IDs are non-negative
  all n: Node, p: Path, pkg: Package, r: Robot, e: Employee |
    n.id >= 0 and p.id >= 0 and pkg.id >= 0 and r.id >= 0 and e.id >= 0
}

fact PathsAreValid {
  -- Ensure paths connect different nodes
  all p: Path | p.start != p.end
}

fact PackageAssignment {
  -- Packages must always exist at a node or be carried by a robot
  all p: Package | some r: Robot | p in r.carrying or some n: Node | p.location =
n
}

fact RobotsHaveTasks {
  -- Ensure all robots have a task assigned and carry at least one package
  all r: Robot | some t: Task | t.robot = r and t.packages != none
}

fact RobotEnergyCheck {
  -- Robots must have energy to perform tasks
  all r: Robot | r.energy > 0
}

fact UniquePackageTask {
  -- A package can only be part of one task at a time
  all p: Package | lone t: Task | p in t.packages
}

fact RobotsCarryAssignedPackages {
  -- Robots assigned a task must carry the corresponding packages
  all t: Task | t.packages in t.robot.carrying
}

fact SupervisorsAssignTasks {
  all t: Task, e: Employee |
    e.role = Supervisor and e.location = t.robot.currentNode =>
    t.robot.status = Idle
}

fact OperatorAssistsRobots {
  all r: Robot, e: Employee |
    e.role = Operator and e.location = r.currentNode =>
    r.energy > 0
}

-- Predicates
pred LoadPackages[r: Robot, ps: set Package] {
  -- Packages must be at the robot's current node
  all p: ps | p.location = r.currentNode
```

```
  -- Packages must not exceed the robot's capacity
  (sum p: ps | p.weight) <= r.capacity

  -- Packages are picked up
  all p: ps | p.location = none and p in r.carrying
}

pred PlanRoute[r: Robot, t: Task] {
  -- Robot moves to a valid destination using paths
  r.status = Moving
  some p: Path | r.currentNode = p.start and r.destination = p.end
}

pred DropPackages[r: Robot, ps: set Package] {
  -- Robot drops the packages at its destination
  all p: ps | p in r.carrying and p.location = r.currentNode
}

pred AssignTaskBySupervisor[e: Employee, t: Task] {
  e.role = Supervisor -- Only Supervisors can assign tasks
  t.robot.status = Idle -- Robot must be idle to receive a task
  t.robot.energy > 0 -- Robot must have enough energy
  t.robot.currentNode = e.location -- Robot must be at the same node as the
supervisor
}

pred OperatorHandlesPackage[e: Employee, ps: set Package, n: Node] {
  e.role = Operator -- Employee must be an operator
  e.location = n -- Operator must be at the same node as the packages
  all p: ps | p.location = n -- All packages must be at the operator's location
}

-- Commands to simulate the workflow
run WarehouseSimulation {
  some r: Robot, ps: set Package, t: Task |
    -- Load packages
    LoadPackages[r, ps] and
    -- Plan the route to destination
    PlanRoute[r, t] and
    -- Drop packages at the destination
    DropPackages[r, ps]

}

-- Command: Simulates the workflow where a supervisor assigns a task,
-- and an operator handles packages related to that task at the appropriate node.
run EmployeeAssignTask {
  some e1, e2: Employee, t: Task, ps: set Package, n: Node | -- There must exist
employees, tasks, packages, and nodes

    AssignTaskBySupervisor[e1, t] and -- A supervisor (e1) assigns a task (t) to a
robot
```

```
    OperatorHandlesPackage[e2, ps, n] and -- An operator (e2) handles packages
(ps) at node (n)

    t.robot.currentNode = n and -- The robot assigned to the task (t.robot) must
currently be at the node (n)

    all p: ps | p in t.packages -- All packages (ps) handled by the operator must
belong to the task (t)
}
```

Task 2: Document the solution

This Alloy model represents an **automated warehouse system**, focusing on robots, tasks, packages, employees, and infrastructure (nodes and paths).
The model is modular, consistent, and extensible, effectively simulating warehouse operations while ensuring system constraints.

Key Design Decisions

1. **Entities**:
    - **Nodes & Zones**: Represent warehouse areas (e.g., Storage, Shipping).
    - **Paths**: Define connections with capacity constraints.
    - **Robots & Packages**: Model transport processes, constrained by capacity and energy.
    - **Employees**: Supervisors assign tasks; operators handle packages.
2. **Dynamic Behavior**:
    - **Predicates**:
        - `LoadPackages`, `PlanRoute`, `DropPackages`: Simulate package workflows.
        - `AssignTaskBySupervisor`, `OperatorHandlesPackage`: Define employee interactions.
    - **Commands**:
        - `WarehouseSimulation`: Models package movement.
        - `EmployeeAssignTask`: Simulates task assignment and package handling.
3. **Constraints**:
    - Packages must have valid locations.
    - Tasks must be unique and logically assigned to robots.
    - Employees operate only within their roles.

Advanced Usage

- **Dynamic Workflows**: Predicates model real-time operations, ensuring logical task progression.
- **Conflict Avoidance**: Facts enforce consistency (e.g., unique tasks, valid package states).
- **Validation**: Alloy's instance visualization checks for errors and ensures correctness.

Task 3: Evaluation of the Model

1. Level of Abstraction

- **Strengths**:
  - Focused on core elements (nodes, paths, packages, robots, employees, and tasks).
  - Abstracts physical properties like time, distance, and robot speed, keeping the model simple.
- **Ignored Details**:
  - Robot mechanics, task execution time, and error recovery mechanisms.
  - Dynamic node creation and task priority management.

2. Level of Approximation

- **Over-Specification**:
  - Tasks are tightly coupled to specific robots, limiting flexibility for task reassignment.
  - All robots must carry packages and have tasks, making the model less adaptable to real-world idle states or downtime.
- **Under-Specification**:
  - The model under-specifies real-time factors, like robot energy depletion mid-task or package delivery delays.
  - Operator and supervisor interactions are simplified and do not include complex behaviors like handling task failures or prioritizing critical packages.

3. Ambiguity vs. Precision

- **Precision**:
  - The model is precise in defining relationships and constraints between entities, ensuring logical workflows.
  - Predicates like `LoadPackages` and `PlanRoute` clearly define robot and package behaviors.
- **Ambiguity**:
  - Ambiguities exist in handling edge cases, such as:
    - What happens when a robot fails mid-task?
    - How does the system prioritize tasks during resource contention?

4. Completeness

- **Strengths**:
  - Covers essential workflows, including task assignment, robot operations, and employee interactions.
  - Includes constraints to avoid conflicts (e.g., path capacity, unique task assignments).
- **Limitations**:
  - Does not model advanced scenarios like:
    - Task reassignment for failed robots.
    - Robots charging mid-task.
    - Handling concurrent tasks with varying priorities.

Evaluation conclusion:

- **Abstraction**: Balanced, but ignores low-level physical and real-time dynamics.
- **Approximation**: Slightly over-constrained, limiting flexibility, and under-specified for error handling and priorities.
- **Precision**: High precision for defined workflows, but lacks clarity in edge cases.

- **Completeness**: Core scenarios are well-covered, but advanced operations and exceptions remain unaddressed.

Task 4: Opinion on Alloy

1. Modeling Language and Tool

- **Strengths**: Alloy's declarative approach simplifies modeling complex systems with modular facts, predicates, and commands. Built-in visualization aids validation.
- **Limitations**: Bounded analysis limits scalability, and lack of native time-handling makes real-time dynamics harder to model.

2. Practical Use

- Highly effective for:
    - Validating complex systems (e.g., task workflows, path usage).
    - Capturing and verifying requirements as formal constraints.
    - Creating precise documentation for better communication.

3. Usage Scenarios

- Ideal for **automated warehouses**, **distributed systems**, and **cyber-physical systems**, as well as for **regulatory compliance**.

Alloy is excellent for system design and early-stage validation, with powerful visualization and automated checks. It could be even more versatile with improved scalability and real-time support.