Astro Robots

The Space station has a fleet of Astro robots. You need to design and implement an application to manage and control this fleet of robots.

Each robot type is designed to execute a single duty. Some types perform sentry duties while others clean solar panels.

Robot Details

Robot properties:

- Name
  A string between 2 and 32 characters
- Callsign
  a string of alphanumeric characters
- Model
  We have the following models:
  HAL9000, Tachikomas, Johnny5, Maschinenmensch
- Tools
  Each robot model have a group of tools.
  Each tool has the following properties:
  - Name
  - state which can be either *READY* or *MALFUNCTION.*
  - Custm work function, here the tools will be used randomly.
  - Tools are created in *READY* state.
  - Tools are used when performing the robot work. While using the tool it has a 0.2 chance of *MALFUNCTION*.
  - A robot with any of its tools in a *MALFUNCTION* state becomes itself a *FAILING* robot. Self-diagnosis can help heal it.
- Different models have different complement of tools

Tool	*HAL9000*	*Tachikomas*	*Johnny5*	*Maschinenmensch*
Laser cutter	✓	✓	✓	
Replicator	✓			✓
Disruptor	✓	✓		✓
Static Brush			✓	

Robot states

Robot can be in any of the following states:

- *ACTIVE* – the robot is ready to work
- *REBOOTING* – the robot is in rebooting
- *WORKING* – the robot is working on solar panel cleaning
- *FAILING* – the robot has an error

Application

The application will enable astronauts to provision and control the Astro robots. The application Will use a configuration file that specifies the initial set of robots to create in the fleet.

Using the application an engineer can execute various commands define bellow.

User Commands

1. List fleet robots.
   List all robots in the fleet:
   - Call sign
   - Model
   - Status
   - List of tools with status
2. Provision new robot
   a. User will create and add new robot to the fleet.
   b. Provide the following information
      i. Robot name
      ii. Robot model
         - Choose model from list of currently supported models.
      iii. Callsign id (how its identified when sending commands)
   c. The newly created robot will have all properties defined as described in section "Robot details".
   d. Newly created robot has a 0.9 probability of being created successfully and then its state will be *ACTIVE.* Otherwise, it will be *FAILING.*
   e. The result of the operation will be reflected to the screen as new robot properties and status.
3. Issue commands to robot
   User will select a robot and then issue commands to it. Application will display callsign property of all robots in the fleet that either are ACTIVE or FAILING and their status. User can choose a robot from the list whereupon the application will display the robot details (all properties) and present menu of applicable actions he can request the robot to perform:
   a. Dispatch
      Robot will start executing its "speciality work" algorithm.
   b. Reboot
      Issue a reboot command
   c. Self-Diagnostics
      Issue command to start self-diagnostics and repair
   d. Delete
      Remove robot from inventory and return to the main menu
   e. Back to main menu

   Actions a,b,c will take some time – random time for each action - but the UI is free to continue. Delete should be immediate.

4. Quit

Robot Action Details

1. Dispatch
    a. If the state of the robot is not *ACTIVE* print "Robot <robot name> is not active" and exist the operation.
    b. Otherwise change the state of the robot to *WORKING*.
    c. Print “Robot <robot name> is in active duty <robot specialized work>”.
    d. Duty shift cycle will take a random time between 30-180 seconds.
    e. When work cycle is finished change state back to *ACTIVE* automatically.
2. Reboot
    a. Change the state of the robot to *REBOOTING.*
    b. Print “Robot <robot name> is rebooting”.
    c. Rebooting will take a random time between 1-5 seconds.
    d. When booting is finished change state back to *ACTIVE* automatically.
3. Self-Diagnostics
    a. This operation is valid only if the robot is in a *FAILING* state.
    b. Scans the robot toolset. For any tool with *FAILURE* state start executing a self-healing operation. Each tool type has its own unique self-healing procedure. These procedures can take a random time between 10 to 20 seconds.
    c. You don't need to implement the procedure, just print "Starting self-healing on tool <tool name>".
    d. Self-healing procedure has a 0.9 probability of success.

Universe cosmic effects

Simulate cosmic rays’ impact on the robots. Asynchronously simulate cosmic particles hitting the robots and causing a malfunction with a probability of 0.1 in each tool.

Deliverables

- Provide a class diagram for the design.
- Think about seprating into different projects.
- Provide one project as a console UI application to managwe the robot fleet.
- Make sure you have unit tests.
- Use maven for project structure.
- Minimal Java version to use 17

Advice

- Provide clean code
- Follow SOLID principles
- Make use of relevant design patterns you know
- Design for flexibility

- Do take into account possible future change requests (CR)
  - Support new robot models
  - Support new tools
  - Support new commands