CS 350 Pre-Task 3 Feedback

The answers reflect my interpretation of the questions. In some cases, it wasn't entirely clearly what the question was. Ask for clarification is you're still unsure.

1. Would the ID be a unique id for that specific object, such that no other object has the same ID, or would it be like a part ID where all linear actuators have the same ID, but any other type of actuator does not have that ID?

The identifier is completely arbitrary. Except for never being empty, there are no rules.

2. Should the user be able to change stateStart, stateEnd, or step after object construction?

No. This is how the component is physically built.

3. Do we need to include the Javadoc html with our submission, or can we assume that the grader will generate the documentation while grading?

Submit only the source code.

4. Would an appropriate toString() method be just the getState() and getID() methods concatenated together, should the toString() report whether the object is dead/not dead as well?

toString() conventionally returns the internal state of an object. Unless specified, the format is normally your choice. Return the data stored at the class level in some meaningful form.

5. Would cancel() set the step size back to the original value before it was affected by acceleration?

No, cancel stops immediately. We have no capability to restart.

6. What is the purpose behind the underscores in the method signatures?

They do nothing, but they denote contractual methods defined by an interface.

7. Should we be able to call cancel() on a dying actuator?

Yes — a bullet to the head works anywhere.

8. Should we be able to revive the actuator if it dies?

We have no capability to restart.

9. Should we provide a Default Value Constructor for the actuators initializing some default values? If so, which values should those be?

No DVC. There's no context for how this actuator would be used, so there's no reasonable default. DVC is overemphasized in CS 210 and 211. In real code, it's rare unless there's a canonical default.

10. In updateState_() you mention that if the next step takes us past the end state, that the former is clamped to the latter. By this do you mean that we set the current state to the end state?

Yes.

11. In terminate_() you mention that for the linear actuators, we need to reverse the step for three calls. Does this mean the actuator moves backwards in state for those three?

Yes. See below.

12. When you say basic javadoc, do you mean just give a description of each method, or do you mean follow all the rules of javadoc but be breif?

Class, class-level data, and method headers. The English contents are already in the task document. You don't need anything profound here. It's an exercise in using Javadoc.

13. What is an appropriate toString in these cases?

See above.

14. In what format do we turn in the package? Just a folder, or zip, or something else?

Either individual source files or a zip of them.

15. How would this actuator class can later fit into the big picture?

We did this example on the board. The actuator drives something (anything, actually) that has a start and end position, a state in between them, and a step that goes from the current one to the next one.

The big picture would need to figure out how to use the actuator, not the other way around. Low-level components rarely know why they exist or are being told to do something.

16. I don't get the description of the getStep() method. It is to get the different value between the ending state and the starting state? Or to get the value of each time the state changing?

The step value is the current change in state per call to update_().

17. What is the format for actuator's ID

Arbitrary means anything except an empty string.

18. Do we need to include a main() to test all the method of 2 classes

No, any tests belong to you. I'm not asking for them or specifying them.

19. Does interval state represent the position of the piston or its speed?

It technically doesn't represent anything related to a piston. It's a point between start and end points on an interval. The fact that this position may later be connected to a piston (among lots of other things) is convenient (by design), but it's not anything the actuator needs to know.

20. Could you elaborate on what a step constitutes?

The step is the value by which the state updates per call to update_().

21. How is a transition step different from a normal step?

It's the same thing. A transition is from the current state to the next.

22. What do you mean when you refer to the "transition step" in the getStep_() method?

See above.

23. What's the difference between a step and a state?

Step is the distance to move from the current state to the next.

24. Does the progression of the state from start to end happen according to a predefined pattern, or does the state machine run until told to terminate or cancel?

Whatever calls update_() does so according to its own plan. Your code needs to react accordingly; i.e., advance or ignore request (if it's at the limit already or dead, etc.)

25. Any specific unit measures or decimal places? Especially with consideration of the output for the toString() methods.

The internal representation is defined as a double. The toString() can display it in raw form, even if this is occasionally ugly.

26. Any specific info needed in the toString? Just all the attributes for the class?

See above.

27. Should the toString() in any specific format? Prepare output for CSV, XML?

See above.

28. Is the attribute "double stepAcceleration" just a ratio to model the acceleration for the nonlinear actuator?

In the linear variant, the step never changes. In the nonlinear variant, it changes by stepAcceleration once per update.

29. If the process that is calling updateState_() stops calling while the actuator is terminating(dying) is there a alloted amount of time to wait before considering the actuator dead? Should the actuator's update be called by the actuator itself until it is dead(exactly 3 steps) or just die? Will this never happen with the calling process?

Time isn't part of the system. We measure events by updates. update_() is always called by the client, not by your code, because it's the client that decides when to cause an update. In other words, it holds the drum stick.

30. Will the cancel command ever be called while terminate is occurring?

It could.

31. Can an actuator be restarted after termination/cancellation? Which method would restart the actuator?

In the definition of this proof of concept, no. Obviously this limits its usefulness, but it isn't the goal to make a completely working variant.

32. Will we be making any other type of actuators in the future? Possibly one that uses a different model for representing movement?

There could be, which is the basis of the interface, but there won't be as a task.

33. Can the starting step state, for the actuator, begin at 0 or must it be >0 for its "positive" value.

Any starting and ending value (within the limits of a double) is valid.

34. Does there need to be a way to allow the actuator to reverse in steps, ie: go from end to start?

Yes, end can be less than start.

35. Does there need to be a way to tell an actuator to begin servicing calls to updateState_() again?

See above.

36. Terminate_() should decrease a linear component by 1/4th for the three calls, yes?

No, a linear component never changes its step size.

37. The difference between the Linear and NonLinear Actuator is the ending and starting up. Where the Linear just goes full speed right away and the NonLinear builds up.

Correct.

38. The NonLinear Actuator on the other hand builds up to full speed linearly since acceleration will add on x per second, but then will lose speed in a nonlinear fashion where it loses half it's current speed each call for 3 calls. Is that correct?

Sort of. First, seconds aren't part of the system.

Second, both gaining and losing speed are nonlinear, just with different curves. Acceleration (at the start) increases by stepAcceleration per update. There's no deceleration to the end state in our solution: we just slam into it and stop instantaneously. But if there's a terminate in progress, the step decreases by a half per update.

39. What information is needed to be given in the toString() method?

See above.

40. Is the state representative of a velocity or position (or another real-world value)?

See above.

41. Is there a way to reset the state to the start state?

No.

42. Is the actuator considered dead after it reaches its end state?

No, it's only dead if it gets canceled (immediately) or terminated (after dying).

43. Are empty strings permissible for the id?

No. This is actually specified in getID_() already.

44. Can the actuator work in a negative direction (i.e. stateStart > stateEnd)?

See above.

45. Should a step larger than the interval between stateStart and stateEnd (e.g. step=8, stateStart=4, stateEnd=6) be allowed?

It's not very practical, but yes, this is valid. Clamping to the limit (see above) will deal with this case.

46. Should calls to getState, getStartState, or getStartEnd throw exceptions if actuator is dead?

No, because the start, end, and current state are still valid.

47. Should calls to updateState throw exceptions if actuator is dead?

See above.

48. Should calls to cancel or terminate throw exceptions if actuator is dead or dying?

See above.

49. Please elaborate on the terminate procedure (i.e. what specifically does "reverse the step" mean).

Reverse the step means to go in the opposite direction by the current step per update. When we terminate (for reasons beyond the scope of the task), the direction of motion changes.

50. What do you consider basic Javadoc? What you consider basic might be different than what others have been taught.

See above.

51. boolean updateState_() Updates the state from its current state to its next state based on the current step. If the next state exceeds the end state, then the former is clamped to the latter. This returns whether the end state has been reached. What do you mean by "Clamped"?

Clamped means it never exceeds the limit.

52. MyActuatorLinear(String id, double stateStart, double stateEnd, double step) The step parameter is the size of the steps, correct?

Yes.

53. Can we get an example of the nonlinear actuator advancing with actual numbers?

We can do some on the board in lab and lecture.

54. Can we get an example of how the terminate_() function works for the linear actuator with real numbers?

See above.

55. For both of the above, can the examples show all the numbers involved in the process of advancing/slowing down such as the start and end states and the step and walk through each iteration of the change of state?

See above.

56. Should we validate every value passed into the constructors?

Partially: only the identifier and step have constraints.

57. When updateState_() reaches the end, does it set the component to be dead?

No, it's done its job. Dying/dead apply only when its job is interrupted.

58. After an actuator has been stopped (made dead), can it be restarted?

See above.

59. Will the step be positive/negative to denote the direction to step in?

The step value passed into the constructor will always be positive, even though direction of movement is negative when the end state is less than the start state.

60. What constitutes a proper toString() method?

See above.

61. Is the implementation of a singleton based salted id generator more work than necessary for this component?

See above.

62. Is there anything other than the Interface and two concrete classes that we should be submitting ie. readme.txt, output.txt?

Just the source code.

63. While the specification calls for concrete implementations of either actuator, should we consider an abstract super class for additional state based mechanisms we may be creating?

You may include other classes as you think appropriate.

64. This may or may not be important to the project, but in the task, what is meant by simplistic acceleration in response to how the non--linear actuator will operate?

Simplistic means it's not based on any defined system. In other words, it slows to a stop either linearly or nonlinearly as specified, not as it would necessarily do in real life.

65. What are you exactly looking for when you ask us to create a java framework, in comparison to standard java classes?

Framework in this case means we have a contract (the interface) that allows us to make these two specified variants and future variants (which we won't).

66. What is meant by considering the extensibility of our solution?

We're doing this as a proof of concept. How it's done may not exactly fit with how it'll eventually be used. Try to design it in a reusable way that wouldn't require much effort to change. This isn't a requirement, but keep it in mind.

67. In the method String_getID_() , the identifier of the component would be a name? such as "LinearActuator" and "nonLinearActuator"?

The identifier is whatever was provided in the constructor.

68. In class you said the cancel() method immediately stops the system? And terminate() method slowly stops the system over time?

Yes.