# CS 350 Task 3.2 Questions

**I'm ok with answering implementation questions here, but keep in mind that these questions are supposed to be about the problem domain and go to the customer to resolve. You wouldn't ask the customer how to write Java code. But in our classroom environment, it's hard to separate customer/client/instructor/student.**

I have question with calulate position, I was given was a message that reads "Your solution hangs in calculatePosition()". Is my solution going into an infinite loop when it gets into my calulate position? if so is there any tips you have to help solve this problem.

This is tough to diagnose here. The nature of the tester doesn't allow for one-stepping, but I'll take a look. An infinite loop can only occur in a loop, either a for/while or in recursion. Put a print statement in those places and see if the same output keeps appearing.

Usually an infinite loop in recursion will fail at some point when the stack overflows, but this isn't happening.

1. In trackmanager.compile(), what would invalidate a join?

Nothing can invalidate a join. This means to convert it from being valid to not valid. I think the question means what can be invalid? This is answered in the 3.1 questions.

2. How many runs are we going to be allowed?

The final number of iterations depends on the latest results. But this shouldn't go on much longer. After all these questions, there's nothing left undefined to confuse the story. Errors now are at the code level in implementing the story. They've also been answered.

3. Is IsNear() used to join segments or to make them unjoinable?

It's used to determine whether two tips are close enough together to be considered overlapping, which leads to them being joined with Joins in compile().

There's no concept of unjoinable. This implies preventing something legal from joining. We prevent illegal joins (more than one call to setJoin_()), but this is different.

Got full points so I don't have any questions.

Good job.

1. what is the result when we run the program

This question is too vague to answer.

2. what is the bearing method is representing

If I understand the question correctly, the answer is that it's the compass angle from one Coordinates object to another. It's a standard trig problem, but on a compass definition of degrees, not a math one.

i have no questions, i accidentally didn't finish fully fixing my known problem before i turned it in because i figured it out late right before midnight.

ok

Questions: no questions atm

ok

When joinTipC or joinTipD are called for a second time should the value passed in just be ignored or should we throw an exception

Throw a RuntimeException.

Can I add helper methods to clean up some of my code?

If they're private, yes. But public methods change the API, which changes the specs.

Should I throw runtime exceptions if a null is passed through rather than another type of exception?

The tester promises not to provide illegal arguments, except when it's testing specifically for this in the compile() validation.

Is the least amount of segments for a valid track 3?1.

The minimum meaningful track in the real world (ignoring the sharp turns) would be three to form a triangle, but actually two is valid. It seems weird to have two segments on top of each other, but technically this is a valid loop.

Why do we need multiple arguments in the addSegments(A_Segment...segments) method.

It's a convenience. The caller can provide all the segments here in one call, or all in separate calls one at a time, or any combination. But only until compile() is called (this is for our information only, not anything you need to enforce in the code, although it would ordinarily be highly advisable).

2. Would replacing addSegments(A_Segment...segments) with (A_Segment segments) cause it not to compile

Yes. It's a very different method signature. The Java compiler doesn't see this form.

No questions at this point. I lost three points on the last run and I know the reason. I have no questions.

ok

I have no questions at this time.

ok

If you have no questions, submit this statement

ok

I have no further questions about the assignment at this time.

ok

n/a

I guess this means no questions, but n/a can mean a lot of things.

Would having the isNear method using a radius of 0.01 affect the results of testTrackLayout1 in the tester?

Yes and no. The complete definition of isNear() is in the 3.1 questions.

When isNear() is tested early as individual low-level tests, the wrong implemention will fail. This costs one or two points, depending on how it fails.

For the high-level join tests that rely on isNear(), the track is configured in such a way that either implementation would still qualify as near.

I already know that I need to adjust my isNear method to use a square, but I need to know if the fails I am getting in testTrackLayout1 are from something else.

See previous.

What conditions/results are expected by test 1-3 in testTrackLayout1 of the tester?

These are the same test, one for each tip:

test 1: multiple join C {doesn't allow joinTipC more than once}
test 2: multiple join D {doesn't allow joinTipD more than once}

This verifies that you throw an exception if the caller submits a join when there's already one there. It would cause the previous one to disappear.

orphaned tips {doesn't allow nonloop track}

This verifies that all tips are joined. The track has to be a loop.

Do we need to check for the error where multiple track loops are created in compile() instead of one?

No. Technically it shouldn't matter because the track manager can actually work with multiple independent tracks as specified, but I don't test for this.

No questions at this time.

ok

No questions

ok

In the A_Segment.getTipDJoin() and A_Segment.getTipDJoin() methods, if we return â©new Join(targetSegment, isFromTipCorD)â© should â©thisâ© be returned as the targetSegment? Iâ©m a little confused because a new segment canâ©t be instantiated in this class.

The only thing that instantiates a segment is the creational code that builds them. The join is a structural process that connects created things. It doesn't create anything itself.

Which of the other methods is calculateBearing() supposed to be used in as a helper method?

It's not supposed to do anything. The specs don't dictate this, so the implementation is your choice. However, it's a basic trig problem (atan2() with some conversions to our compass coordinate system). There should be a need to call anything else in the solution.

How many more iterations of task 3 are we going to do?

See above.

Is there any specific order that we must take into account for segments being compiled?

No. There's no order implied in adding them.

Does the grader use our compiler for the test track or does it use a confirmed working one?

Be careful with the wording here. TrackManager has a compile() method, but we don't refer to it as a compiler.

The tester uses yours to test yours.

Can we come in during office hour and have you look at our results?

Yes, and also by email. This has always been the case. The only time I stated not to ask questions was in class when we had this pretask to due that night.

Can we come in during office hours and get help with our code?

See previous.

Regarding the multiple join check, I've tried a couple different ways and the tester always fails them. I've tried skipping the extra joins. I've tried throwing an exception on the extra joins. From the tester results, I can see that the multiple joins are indeed being detected in compile() but throwing a RuntimeException seems to crash the tester. What else could I try?

Throwing a RuntimeException for a multiple join (see above) is correct. It's not clear what's happening in your case. Send me the latest code.

I have no questions

ok

No Questions

ok

I don't have any questions

ok

At any point, does the tester test the track compiler seperate from calculate position?

testTrackLayout1 tests the validation of multiple joins and orphaned tips. It doesn't use calculatePosition. testTrackLayout2 does this.

In the test results, sometimes a stack trace is shown if our program threw an error, but the program doesn't say whether that test was passed or failed. What should we assume?

If it doesn't report pass, it's considered a fail. There are so many ways an exception can interrupt the normal flow of the tester, so reporting is inconsistent.

Should the joinTipC and joinTipD methods allow two tips that aren't near each other to join?

No. They have to be within isNear().

Are we to traverse through the segments a second time when validating the joins in compile() ?

This is your call. There are several ways to do the validation. But in general yes, you can't determine if there are any orphaned tips until you've joined all the tips that should.

Multiple joins can be detected in compile() or setJoin_()

Currently: this.segments.addAll(Arrays.asList(newSegments)); is the statement im using to add the segments to the list of segments. What other step/check am I missing or is this fine?

This is fine. You can check immediately after adding that they're there.

I think I'm pretty close to getting it, there's just some small things like the question above that my code is providing nullpointer errors about... or is it just that i'm not checking enought for null pointers with if statements all around?

The question is too vague. There should be no null pointer errors. If you're getting this, something is wrong.