CS 350 Task 3 Questions Round 1

Is there a reason why we would make the track segments different lengths?

To form the shape of the track, especially because there are no curve segments.

Regarding the linked list part for the tracks, is it going to be a singly or doubly? From your example it looked like a doubly.

Doubly.

Can the linked list be a circular or are we only sticking to the line?

This question is unclear.

In the coordinate class, when a method is passed in coordinates and it requires two pairs of coordinates, would the "other" coordinates be the ones passed in and "these" coordinates be the ones of the current coordinate object?

The Javadoc specifies this, but yes.

Since in the join class, the isTargetSegmentTipCorD method returns a boolean, are C or D supposed to be either true or false?

All the “or” boolean identifiers refer to true as first part and false as the second.

What do you mean where it says the in the TrackManager class in the compile method that it validates the joins?---

See below.

Coordinates – isNear what does "near" mean? Does it mean on the same segment as, or something else like some arbitrary set distance?

Within 0.01.

Coordinates – calculateBearing. What does bearing mean in the context of our solution?

The compass angle from one coordinates object to another.

Are we doing something with true north?

Yes, true north according to the compass.

A_Segment – calculatePosition. what does the parameter boolean isFromTipCorD mean? Isn't a boolean just true or false? Does true = c and false = d? How is that defined?

Join. Same question as above for isTargetSegmentTipCorD

See above.

Also, when joining points, are the created joins contained as part of the segment's state, as in, are joins a field in A_Segment? Or are they stored in the trackmanager?

Joins are stored in segments.

I understand the recursive solution of calculatePosition but I don't understand the recursive element of Track Manager

as was mentioned in the lecture. Is addSegments the recurisve part? Or is it compile?---

calculatePosition is the only recursive method. It’s not clear how TrackManager was mentioned as recursive in the lecture.

"public Coordinates add(Coordinates coordinates) Adds coordinates to these coordinates. [4]" Should this add the incoming set of coordinates to the current coordinates? As in this.x += x, this.y += y? Or something different?

It takes a second coordinates object, adds those contents to itself, and returns a new coordinates object.

"public boolean isNear(Coordinates coordinates) Determines whether coordinates are near these coordinates. [3]" How is "near" defined? Within how many units is considered "near"

See above.

Are we allowed to use the Java LinkedList class, or are we expected to impliment our own?

It’s conceptually similar to a linked list, but it’s not the same as what’s taught in 211 or provided by Java.

How will we test our code?

This is your call. It’s your code and not part of the specs.

Will we write our own main?

Yes. It’s not part of the specs.

ï»¿You mentioned the track being like a linked list. Am I right to assume that the nodes would be instances of the A_Segment class?

A_Segment is abstract and cannot have instances. SegmentStraight can and serves the role of a node.

If so, wouldnâ®œt the class require pointers to other C and D tips? So for the fields in A_Segment you would have: Coordinates, ID, Length, TipC, and TipD?

These are class member fields.

The Compile() method in TrackManager class, will it throw errors if two segments do not line up or will we have add() check for this?

Orphaned tips are not valid. Throw a RuntimeException.

Should joins be setup for both segments? (if SegA C is joined on segB D should we need to sed SegB D to join with target SegA C).

Yes, two joins are necessary for the bidirectional connection.

Why is `Join` an object when its function could be achieved otherwise?

There are always other ways to do anything. These are the specs.

Why does `A_Segment` have the `A_` prefix?

For clarity in the description that it’s an abstract class.

Is `Cor` in `isFromTipCorD` (and others) supposed to mean `Coordinates`?

See above.

Why are the tips named C and D?

It’s arbitrary, but A and B are already taken by the segments.

What's the utility/use case of `Coordinates.add` and `Coordinates.subtract`?

They’re helpers.

Why are there no definitions for `object.toString`?

You don’t have to implement these. It’s ok to so do, but there are no specs on what the output shall look like.

How is the coordinate plane defined?

Standard math.

Is there/should there be any limit to tracks?

It’s not clear what limit means here.

ï»¿1. Are we writing pseudocode or real code for this assignment?

Java code.

2. Will we be using information from the previous tasks?

Knowledge and understanding, but there’s actually nothing in this task that’s specific to trains. It’s a geometry problem.

3. What is the template for the final output for Task 3?

It’s not clear what this means.

You use the terminology "track is correct" in the task overview, meaning the track makes sense with what we have defined? So no segment A with the exact same tipC and tipD coordinates as segment B (tracks stacked on top of each other) or things like that? What do you mean by "track is correct"?

See below.

Is there a min/max length a track segment should be?

Minimum > 0, no maximum.

Difference between "bearing" and "distance" from one set of coordinates to another (Coordinates.calculateBearing() and Coordinates.calculateDistance())?

Bearing is an angle from one coordinates object to another. Distance is a distance between them.

What is considered "near" in the Coordinates.isNear() method? Do we get to define this?

See above.

Do we need to ensure each segment has a unique id, meaning a segment is not created with the same id as one already included in the track?

No

Do the (x,y) coordinates need to fall within a certain range? Like greater than 0 for instance?

See above.

In TrackManager.compile() it "validates the joins". What conditions need to be met to be considered "valid joins"?

No orphaned tips. No multiple joins.

1. What is the upper and lower bounds of the coordinate system?

See above.

2. Will the point of origin be in the "field", allowing both negative and positive coordinate values?

See above.

3. What measure will be used for bearing - units, origin, and direction of positive increase?

Compass definition.

4. Coordinates are provided in decimal numbers; is there a unit of minimum granularity/precision specified?

Whatever Java provides.

5. Where/how is "near" defined (as in Coordinates.isNear)?

See above.

6. When adding pairs of coordinates are we to add the raw X and Y values as with vectors, or is there some other operation intended?

See above.

7. A_Segment.calculatePosition and Join.isTargetSegmentTipCorD reference booleans to choose between tips C and D of a segment. What do true/false values represent?

See above.

8. Are the coordinates of a segment mutable?

No

9. When TrackManager is determining joined tips, will this be determined by shared coordinates? Will they be exactly the same or only within a certain tolerance?

See above.

10. How should TrackManager respond if given input segments where more than two tips appear that they should be joined?

See above.

11. How should TrackManager represent a tip that is not yet joined to another segment? Will we have a Join with an empty component or some special value to indicate an "open" track segment?

See above.

12. How should range errors in inputs be indicated to the caller? Should an exception be thrown?

It's not clear what a range error is.

13. Is addSegments designed to be called once or possibly multiple times, adding more segments to an already existing system?

Any number of times until compile() is called.

14. Should there be a restriction within this subsystem that guarantees "compile" will be called after all addSegments calls?

No

15. No provision seems to have been made for deleting a track segment once placed. Is this a capability that is anticipated in the future?

Not in this task.

16. CalculatePosition is a part of the segment object hierarchy, without access to Joins or the TrackManager. When passing distances longer than the segment length, what is it the responsibility of this method to determine?

calculatePostion() resides in A_Segment, which has access to its joins.

This was covered in Lecture 22.

If it is simply to pass back invalid coordinates or represent an exception, does the progression from one segment to another lie within an (undocumented at this time) method within these classes or will it be handled by a user of these classes?

There cannot be invalid coordinates on a closed track.

Where are the Join objects stored?

In the segments it joins.

What would an example Join class passed in on the joinTipC/joinTipD methods look like?

AC→BD:

```
Join joinToSegmentB = new Join(segmentB, false);

segmentA.joinTipC(joinToSegmentB);
```

How far is considered "near" for the isNear method?

See above.

Is the 0,0 coordinate considered the middle of the plane, or the bottom left of the plane?

See above.

For the output from the compile method, would println messages along the lines of "Segment A joined with segment B, join is valid" for each join be adequate or does there need to be something more visual?

Throw a RuntimeException if invalid.

Coordinates. calculateBearing: Bearing makes me think compass/lat/long but as the task stands this seems as simple as a slop calculation.

See above.

isNear what is the margin to qualify as "near"?

See above.

TrackManager.addSegments(A_Segment... segments) this is the first time I have seen this feature. I want to make sure I am reasearching the correct thing and this appears to be Variable Arguments (Varargs).

It is.

In the class SegmentStraight there's the calculatePosition method, how do we handle it if the distance input is longer than the length of the segment?

This was covered extensively in Lecture 22.

Where do we keep track of the joins? Also, how does the join know which segment it is joining from?

TrackManager tells it.

What is the scale of the coordinates?

See above.

Where is the origin (0,0)?

See above.

I'm trying to figure out how the coordinate scale works. Is it based off a general (0,0) which would be in the middle of the "Screen" or in one of the top corners?

See above.

Just so I can get a more basic understanding, for the Compile Method ->Is it taking each segment's coordinates, the targeted joins, and the straight segments and linking them all together pretty much?

This was covered in Lecture 22.

In A_Segment, can tip C and tip D have the same join?

A tip can only have one join.

How does the isNear method in Coordinates define "near?"

See above.

For clarification, are two segments, A and B, joined by each segment having a join that targets the other segment? A has a join that targets B and B has a join that targets A?

Yes

Is a straight segment allowed to have both the same tip C and D coordinates, essentially forming a point?

No, the length must be > 0.

Are segments joined to the segments added before and after it?

This question isn't clear.

Can compile in TrackManager be called more than once?

No.

Is there a definition of a "valid" track (for the TrackManager's validation)?

See above.

How does addSegment in TrackManager behave after the track has been complied?

No further segments can be added.

How does getSegments in TrackManager behave before and after compilation?

In both case it returns the segments in an unordered list.

Does getSegments in TrackManager return TrackManager's internal list of segments (assuming there is one)? If so, how does TrackManager behave if the list is changed?

We're assuming there's no manipulation.