Swinburne University of Technology
Faculty of Information & Communication Technologies

HIT3087 / HIT8087 Advanced Java

Sem2, 2011

Assignment 1: Model Railroad

<u>Due: in lecture Sept 28, 2011</u>

Introduction

This assignment is about displaying model trains moving on a layout. It will be an interesting application of the user interface technologies of Swing and Graphics 2D, threads, and collection classes. Unfortunately the data set may not be large enough to make performance and memory management significant issues.

The specification is broken down into many small steps and some of the laboratory exercise give you code to include in your program. Within the overall design there are some choices in what is implemented and in precisely what controls are used, for example, whether radio buttons, or JComboBoxes or menu items are used. Marks will also be given for overall coding and design style, that is, the quality of the solution.

Logistics

The assignment can be done in groups of 1 or 2 people. Whether the size of your group is 1 or 2, it will be marked in the same way so (theoretically), there will be more work if you do the assignment on your own. If two, you are both expected to contribute equal effort.

The assignment is worth a total of 25% of the total assessment. Regard 15/25 on the assignment as a good result and 20/25 as an excellent result. The assignment will be a lot of work to complete within the time allowed, but will help you learn. **Your code must be hand-written -- you are not allowed to use drag'n'drop or other code-generation facilities in Netbeans or Eclipse.**

The assignment is due on Wednesday, September 28, 2011 at 6:30pm. Hand it to me or place it in the Faculty assignment box before 4:30pm. The standard Swinburne penalty for lateness will apply: 10% of the possible marks per day or part-day. Other details about submission appear later in this document.

The Problem

- The supplied data consists of a text file defining the track segments (straight lines and circular arcs). (Connectivity is implicit in this file because xy points appear more than once, but possibly another file may be needed later.) These segments define the middle of the tracks. The data is in mm. Overall the layout fits within a rectangle approximately 3m x 2m -- see Figure 1. The application must read this file and display the whole layout with rails either side of the segments. (No sleepers.)
- The user must be able to control the display of (mostly textual) debugging information on the drawing, eg, segment names and boundaries.
- The user must be able to resize the window, zoom into and pan within the layout.
- The program must support animation of at least two trains of one or more cars. Each car should be displayed as a filled rectangle whose wheels (centre of bogeys, referred to later a "reference points") follow the track (centre line).

- Trains are controlled by the user: speed (up to 150 mm/sec, maybe using a JSlider), travel direction (forward/back, only allowed to change when the speed is zero) and turn direction (left, right or, possibly, auto).
- “Physics” such as inertia, collision detection and slow turnout operation, is not required.
- Collision avoidance is not required either. (It is complicated and the data file does not support it – possibly in Assignment 2.) This means trains might pass through each other.
- “Running off the end” must be prevented, eg at the end of L21. The train must stop with the relevant reference point at the end. The length (following the curves) of each train car must be preserved, ie other reference points of that train must not move too far. The current direction of travel of the train might be automatically inverted.
- The application must use suitable controls such as menus (including About and Help, Open and Quit), buttons, checkboxes, simple dialogs for errors.

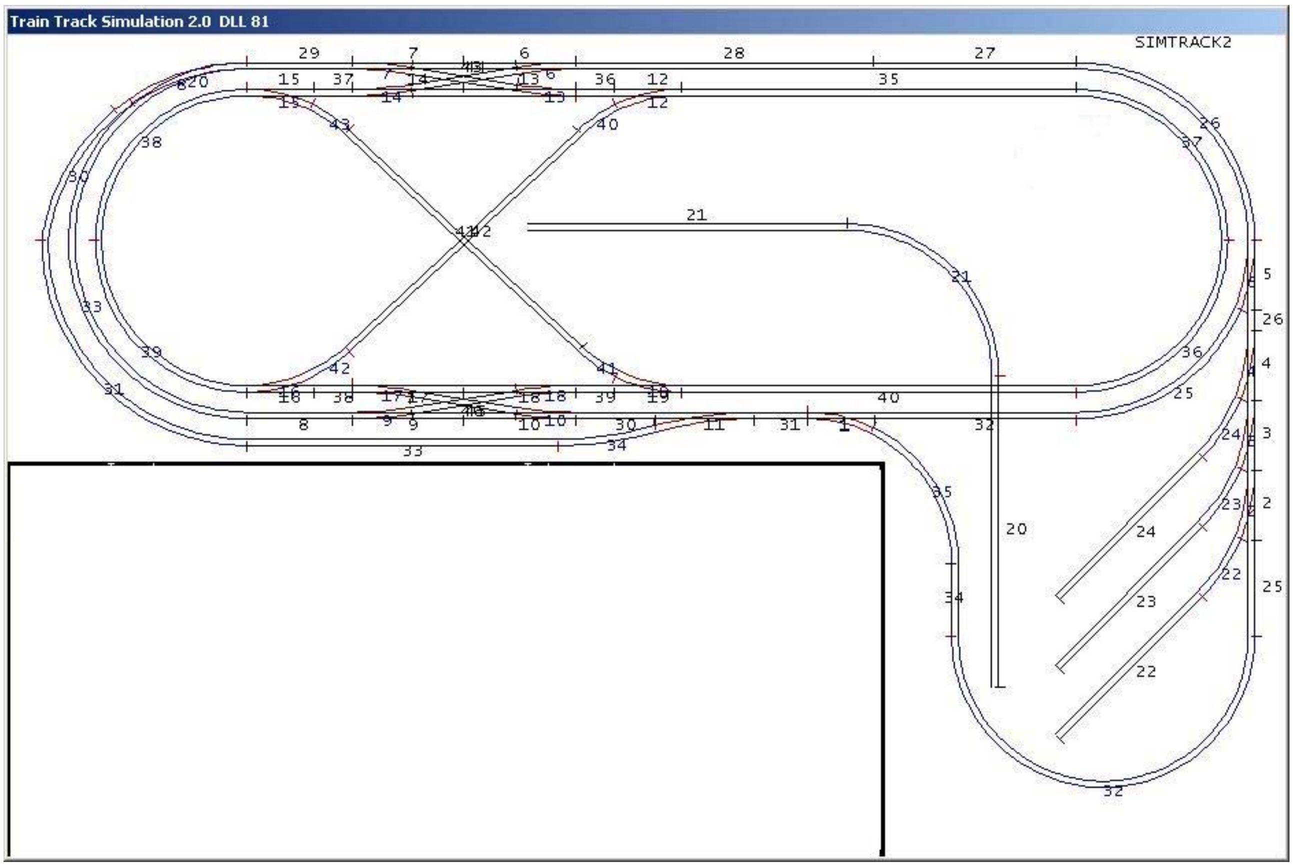


Figure 1: Swinburne Model Railroad Layout

The data file defines the centre-line between the rails. Segment numbers shown here do not differentiate between arcs and lines. Some are obscured, eg turnout 1 consists of arc A1 and line L1. Eighteen turnouts follow this pattern, but turnout 8 consists of A8 and A20.
(The actual model railroad is in ATC620 and is used in HIT3047 Real-Time Programming.)

A feature list follows. The description of each feature may not be precise; you have some freedom in exactly what you implement and how. Describe how you have interpreted that feature in the readme.txt file.

The order of the feature list is also designed as a suggestion for the order of development. Ideally, you should have all the theory before you start the assignment, but this is, of course, not possible. Some of the laboratory exercises are very relevant to the assignment. It is important to keep up in the labs: code should be polished and made more elegant during the weeks. Please use Blackboard for clarification.

Basic Stage (17 marks)

1. Ensure all code has the "right" amount of Javadoc comments. Document public features of classes and especially the overall purpose and functionality of the class. Write comments as you go on – do NOT put them off to the end. Writing comments during coding helps you understand what you are trying to do and is certainly useful the next week when you have forgotten!
   Submit a subdirectory full of html files generated by javadoc.exe. Your ant script may be used to recreate this during marking. (4 marks)
2. Create a Java program which runs as an application creating an empty window with menus File:Exit and Help:About. Basic use of Swing is needed. (1 mark)
3. Implement an "About" dialog activated from the help menu. You can use JOptionPane for this purpose. Include your name(s), date and version and of course the name of the application. (1 marks)
4. Write code to read all the text data (an alternative filename might be supplied on the command line). Build the data structure for the layout. Figure 5 shows the recommended class design. During file reading a HashMap<String, TrackPoint> will be useful for identifying points seen before – the string should be the point text as read from the file. (5 marks)
5. Display the layout in a JPanel subclass, named TrackPanel. Ensure that the layout is automatically scaled to fit in the panel, with the same scale (pixels per mm) used for vertical and horizontal, and the layout is the right way up. This will use an affine transform and Graphics2D. [Hint: start by showing the centre lines not the rails.] (3 marks)
6. Allow the user to choose to see arc segment numbers or line segment numbers. You do not have to duplicate Figure 1. (2 marks)
7. Provide an ant script to compile, run, clean, create javadocs and summary (-p). (1 marks)

Stage 2 (10 marks)

8. Set up a class, Train, to represent a model train, at this stage a single car. This is complicated by the fact that it may occupy several segments – we need a double-ended queue. As a train moves forward or backward segments will join or leave the data structure at one end or another. (The reason for this approach relates to turnouts: the decision of which branch of a turnout to take, left or right, is made once, on entry to it. The rest of the train follows the same route even if it reverses.)

   - See Appendix B for some details of how a car can occupy a segment and how to specify and modify the wheel positions of a car.

   Drawing a car involves finding the x,y position of the two wheel points of the car and drawing a rotated rectangle on top of the layout. (Hint: rotate the shape in user coordinates before drawing it, not as part of a g2 operation.) JLayeredPane might be useful(?)

Allow the user to single step a car forward by eg 10mm (for debugging). A car might start at a fixed position and go straight by default.

Zoom Stage (7 marks)

9. Add scroll bars and operations to zoom-in, zoom-out and reset (to 100% size). Zoom by factors of 2 (optionally 1.4 if shift-clicked). In a zoom-in operation try to keep the centre of the viewport fixed. Scrolling is done by the user dragging (or clicking) a scrollbar. (5 marks)
10. Write code for the user to initialize the front of a car to a chosen segment and distance s1 within it. Possibly by clicking on/near a segment. (2 marks)

Animation Stage (7 marks)

11. Animate trains with user control: speed (up to 150 mm/sec, maybe using a JSlider), travel direction (forward/back, only allowed to change when the speed is zero) and turnout direction (left, right or straight). (7 marks)

Advanced Stage (4 marks)

12. Implement right-click popup display of car properties. (2 marks)
13. Implement 3-car trains. (2 marks)

Quality and Style (5 marks)

14. Coding style, design, GUI appearance, as assessed by the marker. This mark for style and for item 1 will be less if you have completed less of items 2 to 18.

Total = 50 marks

The total will be scaled to 25%. The total mark will be reduced by subtracting marks for too much acknowledged code (eg > 20% of code) and any penalties for lateness, for being called in to demonstrate, or (large penalties) for the inclusion of unacknowledged code.

Where to get help

The laboratory work will help you complete this assignment as well as learn some other lecture topics. Use the Discussion Board. In past years this has been very, very helpful to students. Google for airport names.

Note that the way you can receive feedback on your progress in the subject is through your progress in the assignment; you should be able to have a fair idea of the mark you will get by what you have completed. You can also receive feedback and comments from your tutor in the lab sessions BUT do not expect the tutor to tell you what to do!

Submission and Marking

The assessment of the assignment will be based on the presence of working features and on submission according to the rules and deadlines. Assignment marks may be reduced by the presence of acknowledged external software comprising more than 20% of your assignment (based on lines of code). If your assignment is found to contain software the same or similar to another source and this has **not** been acknowledged, the marks penalty will be extremely high. **Important**: the portions of imported code must be acknowledged as comments in the code by clearly delimiting the boundaries of the imported code and providing a reference to the source of the code. A **readme.txt** file must be provided (see below) and this should state exactly what code is imported.

Your assignment must be submitted in an A4 envelope with a complete Swinburne Assignment Cover Sheet attached to the outside. The envelope must be handed directly to the lecturer or placed in the FICT assignment box at EN153 (details on Blackboard). Include:

(a) hard-copy of your code, preferably double-sided, two pages each side to save trees.

(b) If you did the assignment in a pair, both must sign the cover sheet.

(c) A simple UML-style class diagram showing the structure of your program. This will be considered in marking for style. Do not show methods or data members. Show associations but, if the only association between classes is that one constructs the other, then preferably don't show the association. The diagram can be drawn in pencil.

(d) A CDROM, containing duplicate copies of a SINGLE zip file, which contains your **readme.txt, build.xml** and subdirectories. Do not forget to retain the subdirectory information when you create the zip. Subdirectories:
- classes: class files
- src: java files
- data: data files – whatever is needed
- doc: html documentation of your program generated by javadoc.exe. Include all classes and non-private features of them. Do not include links to the standard API. Preferably include version, author (classes only)

Please check that it is possible to open the zip file, **clean, compile** and **run** using Ant; also that the readme.txt contains accurate and adequate "driving instructions".

(e) Your "**readme.txt**" should be created with a text editor such as notepad, not a word processor. (The marker will view this with Notepad++.) This file must contain
- identification information (name, id, and subject id for each group member);
- instructions on how to run the assignment (if the marker cannot run your assignment then he may contact you and ask you to come to the campus to demonstrate your work – this will cost you a penalty of 10%, but that is less than loss of many marks because the code did not run);
- instructions on how to recompile (perhaps using a BAT file) if Ant not used;
- a list of what items you have implemented and where in the source code to look. (I do not want to waste time trying to marks things that are not there!);
- a list of known bugs,
- identification of help you have received from outside your group especially code that you have obtained from other sources (you also have to identify this imported code in comments in your .java files),
- an estimate of what you think you will get for the assignment,
- (if working in a pair) who did what, and
- any other discussion that is needed.

Occasionally there will be problems with the submitted zip files. In such cases you may be asked to email replacement files. All students therefore need to retain a complete electronic copy of their software. Do not be tempted to modify/improve the work you have previously submitted.

Your assignment will be marked using j2sdk 1.6.0 (patch 21 or later) under Windows XP.

This assignment will be returned in lab classes and verbal feedback given.

Appendices:

A. Sample data lines (file endpoints_080206T2.csv):

```
ID,Kind,Turnout,x1,y1,x2,y2,len,xc,yc,radius
1,arc,1,1984,951,2146.3,988.4,168,1984,1322,371
2,arc,2,3069.2,1270.2,3097,1088,185,2487,1088,610
3,arc,3,3069.2,1095.2,3097,913,185,2487,913,610
...
43,arc,0,1094.3,354.7,795.7,174.7,352.1,580,870,728
1,line,1,1984,951,2152,951,168,,,
2,line,2,3097,1273,3097,1088,185,,,
3,line,3,3097,1088,3097,913,175,,,
...
45,line,0,1115.1,939.2,1379.9,895.8,268.4,,,
46,line,0,1115.1,895.9,1379.9,939.1,268.4,,,
```

Note: all measurements in mm and (0,0) is top left of Figure 1. Turnout=0 means the segment is not part of a turnout.

B. Geometry:

- Graphics2D can draw the rails easily. The rails are a fixed distance either side of the segments defined in the data file. For each arc of the layout set up two Arc2D objects, one with radius r+h and the other r – h where h is half the rail spacing. For each line set up two Line2D objects, one displaced h to the left and the other h to the right. (See Blackboard discussion forum for follow up information.)
  - There will in fact be small gaps and overlaps but they ought not be visible, ie less than line thickness.
  - You can set up one Path2D or GeneralPath object for all the rails.

- Whatever constructor for Arc2D you use, you will need the starting angle and angular extent (= endAngle – startAngle)

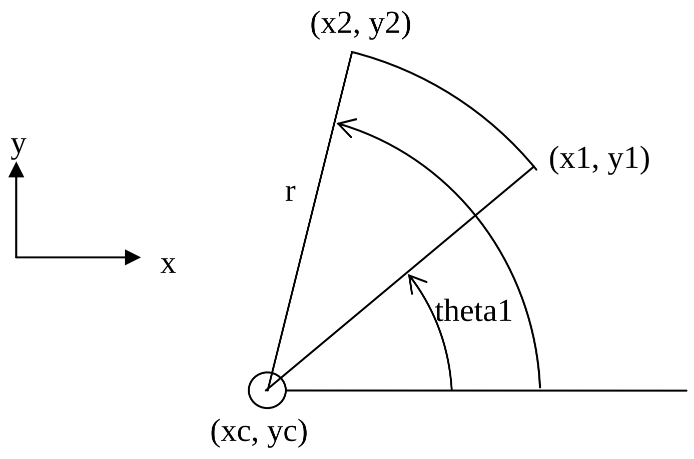


Figure 2: Calculating angles of an arc (coordinate system for java.lang.Math)

theta1 = Math.atan2(y1 – yc, x1 – xc); // radian

theta2 = …. similar

To convert to degrees multiply by 180.0/Math.PI

Note that coordinate system used by java.awt (and the data file) has y inverted and uses degrees for angles. Angles are still positive in the anticlockwise direction.

The path length along a circular arc is r*dTheta where dTheta is in radians.

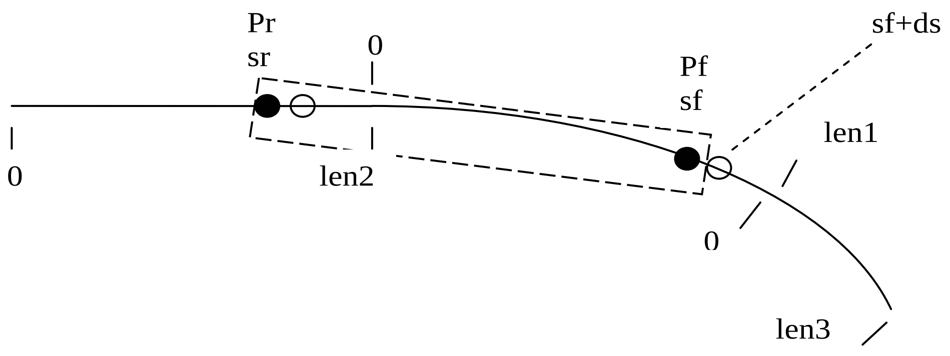


Figure 3: Specifying the position of a car on two segments (and the next segment shown)

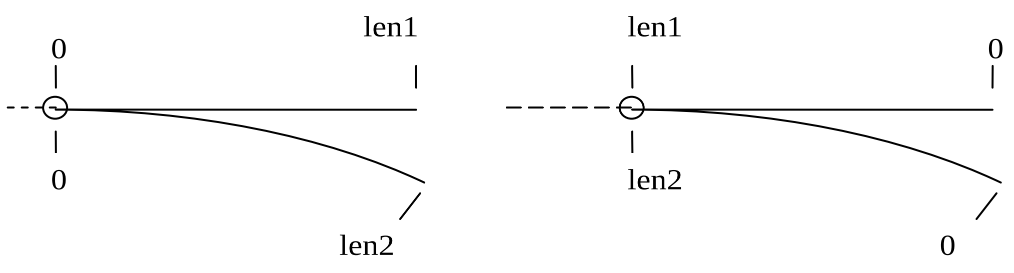
Occupation of a segment may be complete (ends of the car are outside the segment), or partial – the car entirely within one segment or the front might occupy one end of a segment or the other end, similarly for the rear. We will use mm along the segment to specify position on both kinds of segment. Each segment will have its own 1-D coordinate direction (zero is at one end – at the point provided first in the data file). So you specify the position of a reference point (ie the front or rear of a car) by specifying a distance, s, within the relevant segment. In Figure 3:

- Pf is the front-most part of the car within that segment. That corresponds to distance sf along that segment. Similarly Pr, sr is the rear. In the diagram above sr is in the coordinate system of the rear line segment and $0 < sr < len2$
- Here the positive direction within a segment is to the right (so this could correspond to L31 and A1 in Figure 1).
- When a car moves we adjust sf and sr by the same amount, ds, along their respective segments. Numerically this depends on the orientation of the train/car relative to the segments.
    - For convenience the attribute localOrientation will be NORM if the direction from rear to front (Pr to Pf) is the positive coordinate direction of the particular segment. The alternative is BACK. [Hint: represent these by +1 and -1 respectively.]
    - If localOrientation == NORM and the front of the car moves forward by ds mm then
      sf = sf + ds (ds could be negative.) Assuming ds is positive then if sf > len1 then another segment (an arc with length len3 in the diagram) becomes occupied by the car and a new object needs to be added to the front of the double-ended queue. The new sf will be the residual sf = sf –len1. We assume ds is small so that $0 < sf < len3$.
    - Usually localOrientation is the same for all segments occupied by a train so a new segment that is added will be the same but due to the topology there are two places where there is a flip, between segments A16 and A42 and between A19 and A42 (implicit in the data file).

  (Note this method of moving a train by following curves is an approximation and not physical. It can result in cars being squashed lengthwise by a small fraction. Real train cars are rigid and form chords of arcs, ie "short cuts".)

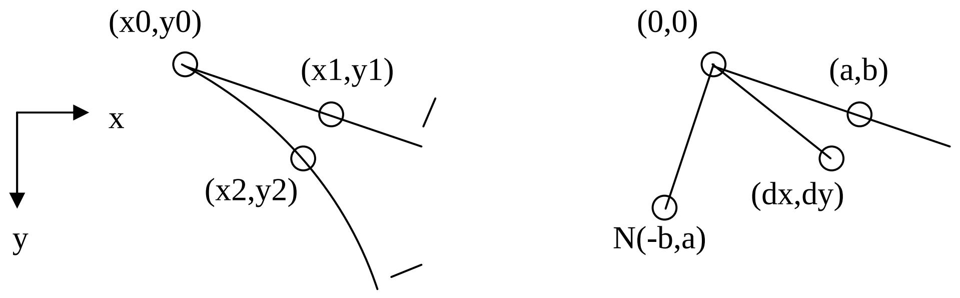
- To calculate the xy point that corresponds to a distance s within a segment:
    - For a straight line segment from point P1(x1,y1) to point P2(x2,y2) of length len
      x = x1 + (x2 – x1)*s / len;
      y = y1 + (y2 – y1)*s / len;
    - For an arc with centre Pc(xc,yc) and radius r from angle theta1 to theta 2 (radians) of length len
      theta = theta1 + (theta2 – theta1)*s / len;
      x = xc + r*cos(theta);
      y = yc + r*sin(theta); // assuming y is upward on the screen
      (Note: there is an alternative formula that is more efficient as it avoids sin() and cos(). It needs some more pre-computed values. Watch Blackboard.)

- Turnouts consist of a pair of segments, referred to as left or right or equivalently straight or turned. (This is convenient for train control, eg "always go straight".)



(a) (b)

Figure 4: Turnouts – (a) diverging (b) converging when considered in the NORM orientation.
In both these the 'left' segment ("arm") is straight. The 'choicePoints' are shown as circles. The 3rd segment (sometimes called "central") that joins each choice point is shown dashed; it is not considered part of the turnout.



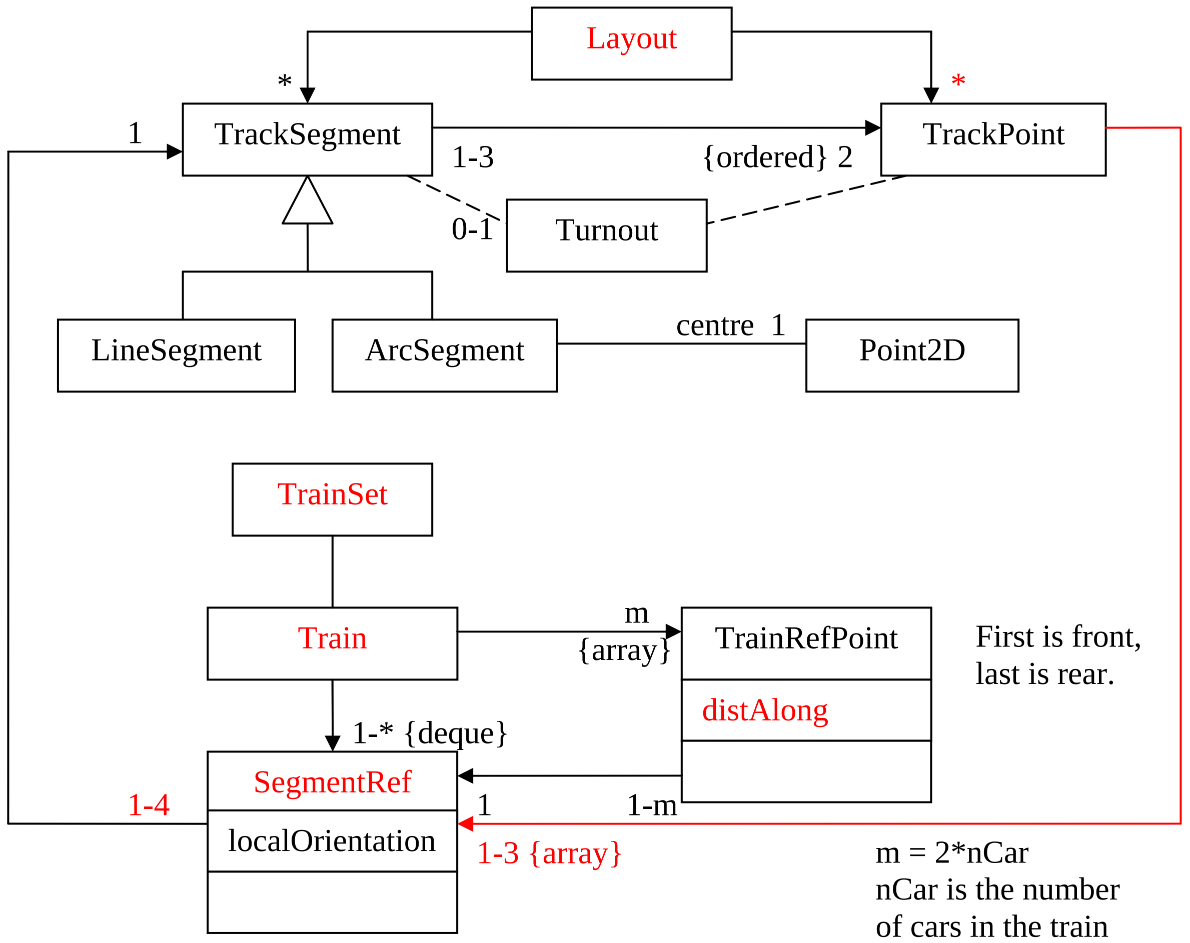
(a) (b)

Figure 5: Calculating which arm is "right"
(a) The problem is which of the two points (x1,y1) and (x2,y2) is to the right. These must be equal distances along their segments. They are in user coordinates (mm).
(b) Equivalent problem with the origin at the "choice point".
Here a = x1 – x0, b = y1 – y0, dx = x2 – x0, dy = y2 – y0. (All positive in this example.)
The line to N is at 90 degrees to the line to (a,b) towards the right. The line to (dx,dy) is also to the right if the "dot product" is positive, that is, if

$$dx*(-b) + dy*a > 0$$

(You could also solve this problem by comparing angles found using atan2 but that is complicated by discontinuities in the values returned by atan2.)

C. Suggested Partial Class Design – Figure 6



The set of SegmentRef objects owned by Train refer to exactly those segments occupied by the train. It is organized as a double-ended queue because the train can reverse travel direction (ds negative). Choosing the next segment at the current front is interesting; the rest of the TrainRefPoints just follow along.

Class Turnout may not be needed at all – having 'turnoutId' as fields of TrackSegment and TrackPoint may be enough. TrackPoint is associated with 1-3 TrackSegment objects. When there are 2 (very common) a trackPoint can easily tell what segment is next given what segment the carriage is coming from. This is more complicated at choicePoints where three segments meet:

- Which segment are we coming from?
- What is the turnout direction setting?

Hence these should be the parameters of a **next** method of class TrackPoint.
The red arrow above is for pre-calculated SegmentRef objects within each TrackPoint.
Effectively a TrackPoint needs to know

- Which are the three segments and if we enter one from which end are we starting at (localOrientation)?
- Which are the two arms?
- Which is the other segment ("central")?
- Which one of the two arms is (most) straight?
- Which one of these is right? Left?

The first point can be recorded (in the form of SegmentRef objects) when the data is read line by line. The other facts can be calculated when needed but it is more efficient, and easier to debug, to do it once only after the data has been read, but before trains start to move. Hence TrackPoint also needs an analyze or preprocess method. After that has been called on all TrackPoints, all the layout data will be fixed.