Com S 227
Spring 2022
Assignment 4
300 points

Due Date: Friday, May 5, 11:59 pm (midnight)

5% bonus for submitting 1 day early (by 11:59 pm May 4)

NO LATE SUBMISSIONS! No extensions. All work must be in Friday night.

General Information

This assignment is to be done on your own. See the Academic Integrity policy in the syllabus, for details. You will not be able to submit your work unless you have completed the *Academic Dishonesty policy questionnaire* on the Assignments page on Canvas. Please do this right away.

If you need help, see your instructor or one of the TAs. Lots of help is also available through the Piazza discussions.

Please start the assignment as soon as possible and get your questions answered right away. It is physically impossible for the staff to provide individual help to everyone the afternoon that the assignment is due!

Read this specification carefully. It is not quite like reading a story from beginning to end, and you will probably have to reread some parts many times. See the section "If you have questions" if you find something unclear in this document.

Introduction

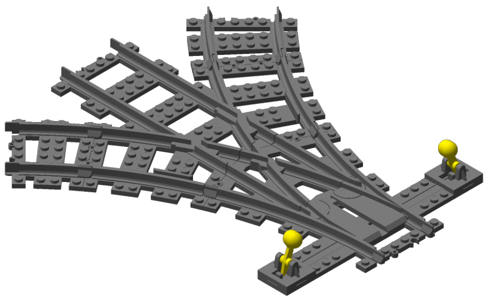
The purpose of this assignment is to give you some experience:

- using interfaces
- reusing code through inheritance ("is-a")
- reusing code through composition ("has-a")

A portion of your grade on this assignment (roughly 15% to 20%) will be determined by your logical organization of classes in a class hierarchy and by how effectively you have been able to use inheritance and composition to minimize duplicated code. Be sure you have done and understood Lab 8 checkpoint 2!

Summary of Tasks

For this project you are implementing a model train simulation. The train tracks the train rides on are modeled as paths connected by links. Links may join together multiple paths and the train can transfer from one path to another when crossing a link.



You will implement the following concrete classes, all of which directly or indirectly implement the two interfaces **`Crossable`** and **`Traversable`**.

```
AbstractLink
CouplingLink
DeadEndLink
MultiFixedLink
MultiSwitchLink
StraightLink
SwitchLink
TurnLink
```

The **`Crossable`** interface already extends **`Traversable`**, so you only need for the concrete classes to implement **`Crossable.`** That looks like a big list of classes, but you will find that if you do things right there isn't a whole lot of code to write! All your code should be in the hw4 package. The details of what these classes do is discussed further below.

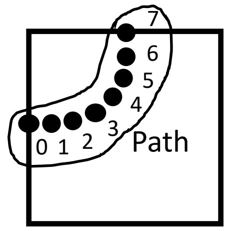
The model is defined as follows.

The Model

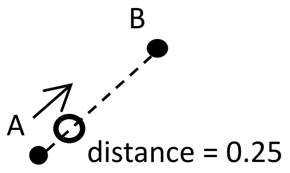
Point

A point consists of an x and y coordinate. Points are placed on paths and a point knows its index in the array of points on its path.

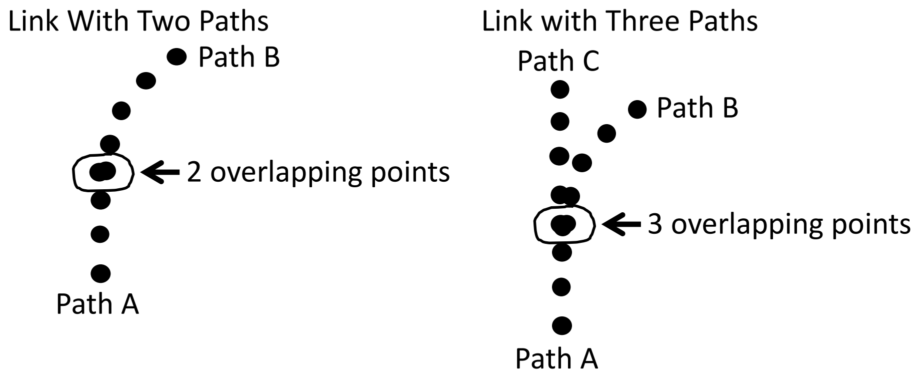
Path
A path consists of multiple points consecutively indexed starting at 0. Paths have two ends, which we will call the low endpoint (at index 0) and the high endpoint (at the highest index).



Position Vector
A position vector consists of two points label A and B and a relative linear distance between A and B. The direction of the vector is points from A to B. The distance is a value between 0 and 1, for example, 0 means at point A, 0.5 means halfway between points A and B, and 1 means at point B.



Link
A link connects two or more paths at their endpoints. The endpoints that form the link must be at the same coordinates.

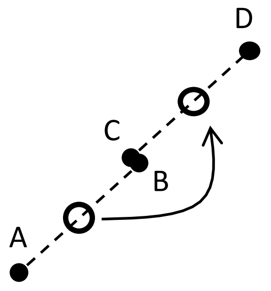


The Interfaces

The **`Traversable`** interface has the method:

```
public void shiftPoints(PositionVector positionVector);
```

Each link class must implement or inherit an implementation of this method. The purpose of this method is to move the `PositionVector` (i.e., the train) from one pair of points to the next. For example, in the figure below the train moves from between points A and B on the first path to between points C and D on the second path. It does this by modifying the given `PositionVector`, specifically calling `setPointA()` and `setPointB()`.



The `Crossable` interface has the methods:

```
public Point getConnectedPoint(Point point);
```

Suppose two paths are linked together at a point A on path A to a point B on path B. When the method is given point A is should return point B.

```
public void trainEnteredCrossing();
```

The simulation may call this method to indicate that a train has entered the crossing. This is relevant because some links cannot switch connections when a train is currently crossing.

```
public void trainExitedCrossing();
```

The simulation may call this method to indicate that a train has exited the crossing. This is relevant because some links switch connections after each train crossing.

AbstractLink

You are required to implement

Other classes in the API

The `api` package in the sample code includes the `Crossable` and `Traversable` interface and a few other classes that model the simulation. You should examine these classes to see how they can be used. *However, you must not modify any of these classes/interfaces*.

The `ui` package contains a class `SimMain` that contains a main method which can be run to visualize the simulation. It is still usually best to test and debug code with simple tests on the console, not the GUI.

Testing and the SpecChecker

We will provide a basic SpecChecker, but **it will not perform any functional tests of your code**. At this point in the course, you are expected to be able to read the specifications, ask questions when things require clarification, and write your own unit tests.

Since the test code is not a required part of this assignment and does not need to be turned in, **you are welcome to post your test code on Piazza for others to check, use and discuss.**

The SpecChecker will verify the class names and packages, the public method names and return types, and the types of the parameters. If your class structure conforms to the spec, you should see a message similar to this in the console output:

```
x out of x tests pass.
```

This SpecChecker will also offer to create a zip file for you that will package up everything in your hw4 package. **Please check this carefully. In this assignment you may be including one or more abstract classes of your own, in addition to the 7 required classes, so make sure they have been included in the zip file.**

Importing the starter code

The starter code includes a complete skeleton of the four classes you are writing. It is distributed as a complete Eclipse project that you can import. It should compile without errors out of the box.

1. Download the zip file. You don't need to unzip it.
2. In Eclipse, go to File -> Import -> General -> Existing Projects into Workspace, click Next.
3. Click the radio button for "Select archive file".
4. Browse to the zip file you downloaded and click Finish.

If you have an older version of Java or if for some reason you have problems with this process, or if the project does not build correctly, you can construct the project manually as follows:

1. Extract the zip file containing the sample code (*Note: on Windows, you have to right click and select "Extract All" – it will not work to just double-click it*)
2. In Windows Explorer or Finder, browse to the `src` directory of the zip file contents
3. Create a new empty project in Eclipse
4. In the Eclipse Package Explorer, navigate to the src folder of the new project.

5. Drag the `api`, `hw4`, `simulation`, and `ui` folders from Explorer/Finder into the `src` folder in Eclipse.

Suggestions for getting started

1. You can make significant progress without using inheritance, so don't let that stop you.

2. Start with `DeadEndLink` and directly implement the `Crossable` interface. (That is, temporarily forget about the requirement to extend `AbstractLink`.) This link class is the simplest to implement, its methods either return null or do nothing.

3. Next implement `CouplingLink` and directly implement the `Crossable` interface. (That is, temporarily forget about the requirement to extend `AbstractLink`.) The two most difficult to implement methods are `shiftPoint()` and `getConnectedPoint()`. For `shiftPoint` think about using `setPointA()` and `setpointB()`. It is useful to get information about the path from a point by calling `getPath()`.

4. At this point test track 1 when running `SimMain` should work.

5. Now make `DeadEndLink` and `CouplingLink` extend an abstract class `AbstractLink`.

6. Next implement `StraightLink`. At this point test track 2 should work.

7. Continue implementing the other required classes. Move as much common code possible into inherited classes. It is allowed to create new classes and use protected methods.

Requirements and guidelines regarding inheritance

A generous portion of your grade (maybe 20%) will be based on how well you have used inheritance, and possibly abstract classes, to create a clean design with a minimum of duplicated code. Please note that there is no one, absolute, correct answer – you have design choices to make.

You will explain your design choices in the class Javadoc for `AbstractLink`.

Specific requirements

You are not allowed to use non-private variables. Call the superclass constructor to initialize its attributes, and use superclass accessor methods to access them. If your subclass needs some kind of access that isn't provided by public methods, you are allowed to define your own `protected` methods or constructors.

Other general guidelines

You should not use `instanceof` or `getClass()` in your code, or do any other type of runtime type-checking, to implement correct behavior. Rely on polymorphism.

- No class should contain extra attributes or methods it doesn't need to satisfy its own specification.

- Do not ever write a method like this (it is redundant):

  ```
  public void foo()
  {
    super.foo()
  }
  ```

  There is almost never any reason to declare and implement a method that is already implemented in the superclass unless you need to override it to change its behavior. *That is the whole point of inheritance!*

Style and documentation

Special documentation requirement: you must add a comment to the top of the `AbstractLink` class with a couple of sentences explaining how you decided to organize the class hierarchy for the evaluators.

Roughly 10 to 15% of the points will be for documentation and code style. Some restrictions and guidelines for using inheritance are described above.

When you are overriding a superclass or interface method, **it is usually NOT necessary to re-write the Javadoc**, unless you are really, really changing its behavior. Just include the @Override annotation. (The Javadoc tool automatically copies the superclass Javadoc where needed.)

The other general guidelines are the same as in homework 3. Remember the following:

- You must add an @author tag with your name to the javadoc at the top of each of the classes you write.
- You must javadoc each instance variable and helper method that you add. Anything you add must be `private` or `protected`.
- Keep your formatting clean and consistent.
- Avoid redundant instance variables
- Accessor methods should not modify instance variables

If you have questions

It is extremely likely that the specification will not be 100% clear to you. Part of your job as a developer is to determine what still needs to be clarified and to formulate appropriate questions.

In particular, since you are not required to turn in any test code for this assignment, your tests and test cases can be freely shared and discussed on Piazza. *It is your responsibility to make sure that you correctly understand the specification and get clarifications when needed.*
For questions, please see the Piazza Q & A pages and click on the folder **`hw4`**. If you don't find your question answered already, then create a new post with your question. Try to state the question or topic clearly in the title of your post, and attach the tag **`hw4`**. *But remember, do not post any source code for the classes that are to be turned in.* It is fine to post source code for general Java examples that are not being turned in. (In the Piazza editor, use the buttons labeled "code" or "tt" to have Java code formatted the way you typed it.)

If you have a question that absolutely cannot be asked without showing part of your source code, make the post "private" so that only the instructors and TAs can see it. Be sure you have stated a specific question; vague requests of the form "read all my code and tell me what's wrong with it" will generally be ignored.

Of course, the instructors and TAs are always available to help you. See the Office Hours section of the syllabus to find a time that is convenient for you. We do our best to answer every question carefully, short of actually writing your code for you, but it would be unfair for the staff to fully review your assignment in detail before it is turned in.

Any posts from the instructors on Piazza that are labeled "Official Clarification" are considered to be part of the spec, and you may lose points if you ignore them. Such posts will always be placed in the Announcements section of the course page in addition to the Q&A page. (We promise that no official clarifications will be posted within 24 hours of the due date.)

What to turn in

Note: You will need to complete the "Academic Dishonesty policy questionnaire," found on the Assignments page on Canvas, before the submission link will be visible to you.

Please submit, on Canvas, the zip file that is created by the SpecChecker. The file will be named **`SUBMIT_THIS_hw4.zip`** and it will be located in whatever directory you selected when you ran the SpecChecker. It should contain one directory, **`hw4`**, which in turn contains the eight required classes along with any others you defined in the **`hw4`** package. **Please LOOK at the file you upload and make sure it is the right one and contains everything needed!**

Submit the zip file to Canvas using the Assignment 4 submission link and verify that your submission was successful.

We recommend that you submit the zip file as created by the specchecker. If necessary for some reason, you can create a zip file yourself. The zip file must contain the directory **hw4**, which in turn should contain everything in your hw4 directory. You can accomplish this by zipping up the **hw4** directory of your project. **Do not zip up the entire project**. The file must be a zip file, so be sure you are using the Windows or Mac zip utility, and NOT a third-party installation of WinRAR, 7-zip, or Winzip.