P06 City Route Planner

Overview

Imagine you are at the Camp Randall Arch, and need to walk to the Mosse Humanities Building. To do so, you need to travel about five blocks east, and two blocks north. How many possible paths could you take between the two (there are 21 not including side streets), and precisely what do these paths look like? In this assignment you will use object-oriented design patterns and recursion in order to solve the two problems of counting and enumerating possible walking paths on a city block grid.

Grading Rubric

5 points	**Pre-Assignment Quiz:** The P6 pre-assignment quiz is accessible through Canvas before having access to this specification by **11:59PM on Sunday 10/30/2022**. You CANNOT take the pre-assignment quiz for credit passing its deadline. But you can still access it. The pre-assignment quiz contains hints which can help you in the development of this assignment.
15 points	**Immediate Automated Tests:** Upon submission of your assignment to Gradescope, you will receive feedback from automated grading tests about whether specific parts of your submission conform to this write-up specification. If these tests detect problems in your code, they will attempt to give you some feedback about the kind of defect that they noticed. Note that passing all of these tests does NOT mean your program is otherwise correct. To become more confident of this, you should write and run additional tests of your own.
20 points	**Additional Automated Tests:** When your manual grading feedback appears on Gradescope, you will also see the feedback from these additional automated grading tests. These tests are similar to the Immediate Automated Tests, but may test different parts of your submission in different ways.
10 points	**Manual Grading Feedback:** After the deadline for an assignment has passed, the course staff will begin manually grading your submission. We will focus on looking at your algorithms, use of programming constructs, and the style and readability of your code. This grading usually takes about a week from the hard deadline, after which you will find feedback on Gradescope.

Learning Objectives

The goals of this assignment include:

- Reinforcing object-oriented design and encapsulation of data and related methods
- Exploring recursive thinking and problem decomposition
- Additional practice designing tester methods and scenarios

Additional Assignment Requirements and Notes

(Please read carefully!)

- Pair programming is **ALLOWED** but not required for this assignment. If you decide to work with a partner on this assignment, REGISTER your partnership NO LATER than **11:59PM on Sunday 10/30/2022** and MAKE SURE that you have read and understood the CS300 Pair Programming Policy.
- **The ONLY external libraries** you may use in your program are these libraries:

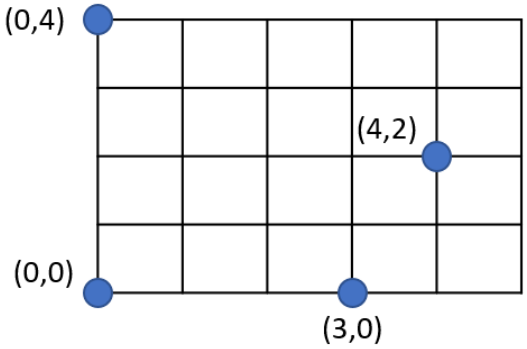
```
java.util.ArrayList
java.util.NoSuchElementException
java.util.Scanner
```

- Only your submitted `PathUtilsTester` class can contain a **main** method.
- **Javadocs** of the classes to be implemented in this assignments are available here.
- You are NOT allowed to add any constant or variable not defined in this write-up **outside of any method**.
- You CAN define local variables (inside a method) or **private** helper methods that you may need to implement the methods defined in this program.
- You CAN NOT add any **public methods** to your classes other than those defined in this write-up.
- Feel free to **reuse** any of the provided source code or javadoc comments in this write-up verbatim in your own submission.
- Be sure that your code follows the CS300 Course Style Guide.
- You MUST adhere to the Academic Conduct Expectations and Advice.

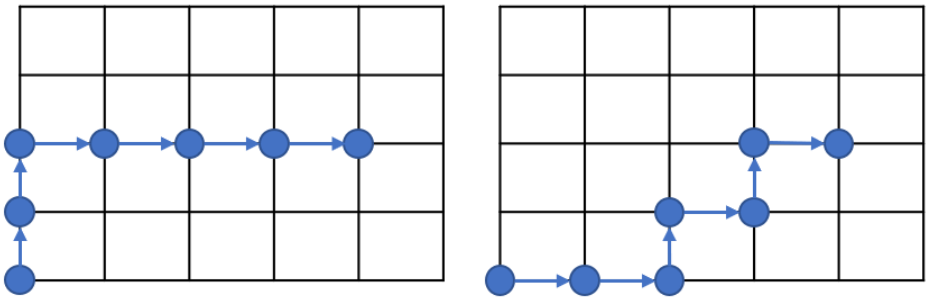
1 Getting Started

Start by creating a new Java Project in eclipse called **P06 City Route Planner**, or something similar. As always make sure that you are using **Java 17**, don't add a module, and that you use the default package.

Now create a new class called Intersection. This class represents a single intersection point where two streets cross at specified x and y coordinate positions. Implement this class following the provided javadocs. You will not be required to write tester methods for this class, but you are free to implement your own. You can picture Intersections as crossroad points on a 2D grid of streets. As an example, the following image represents four instances of the Intersection class located at `(0, 0)`, `(3, 0)`, `(0, 4)`, and `(4, 2)`.



Create a new class called Path. This class represents a path through a city grid which ONLY **moves exactly one step up, or one step to the right at each step**. Moving both up and right (diagonally) in a single step is NOT allowed. Implement this class following the provided javadocs. You will not be required to write tester methods for this class, but you are free to implement your own. As an example, the following images represent two distinct paths starting at `(0, 0)` and ending at `(4, 2)`.

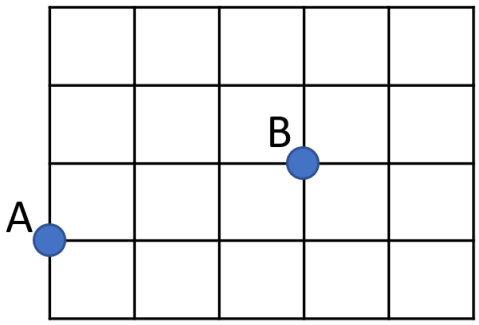


2 City Path Planning

Now that we have a way to represent a grid street plan, it's time to use *recursion* to find our way through the city! Recall that our city consists of blocks of buildings surrounded by streets.

Streets run at right angles to each other, forming a grid of intersections.

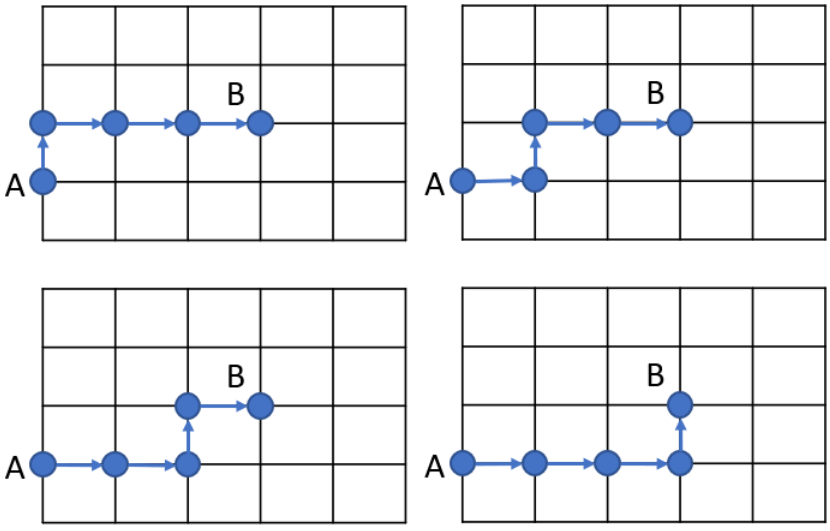
Create a new class called PathUtils with two methods `countPaths` and `findAllPaths` as specified by the javadocs. This is a utility class which solves the following problems. Suppose we have two Intersections on a city grid, call them A and B, which we want to travel between by moving only north or east at each step as in the Path class. Recall that in our Paths we can only move up and right, never backtracking left or down, and that we cannot cut diagonally through a block of buildings.



Two possible questions we can ask are

1. How many possible paths are there from A to B?
2. What exactly are these paths?

In this case, there are four possible paths, which look as follows:



It will be your job to answer these two questions in the `countPaths` and `findAllPaths` methods.

- In `countPaths`, you will count the number of paths going between a chosen starting and ending Intersection. As an example, for the above scenario, `countPaths(new Intersection(0, 1), new Intersection(3, 2))` should return 4. Think of the base cases, recursive cases, and how to decompose the problem to smaller subproblems before you start coding. You will not receive credit if you implement this method iteratively or combinatorially. You will only receive credit if you implement it in a directly recursive manner, or utilize a private static helper method which is recursive. You also may not use your implementation of `findAllPaths`.

- In `findAllPaths`, you will create an `ArrayList` containing all of the Path objects going between a chosen starting an ending Intersection. For example, for the above scenario, `findAllPaths(new Intersection(0, 1), new Intersection(3, 2))` should return an `ArrayList` with four Paths which look as follows:

  ```
  (0,1)->(1,1)->(2,1)->(3,1)->(3,2)
  (0,1)->(1,1)->(2,1)->(2,2)->(3,2)
  (0,1)->(1,1)->(1,2)->(2,2)->(3,2)
  (0,1)->(0,2)->(1,2)->(2,2)->(3,2)
  ```

  These Paths can be in any order in the returned ArrayList, but the result should contain exactly these Paths. Think of the base cases, recursive cases, and how to decompose the problem to smaller subproblems before you start coding. This method generalizes `countPaths`, so we recommend you complete that method first. You will not receive credit if you implement this method iteratively. You will only receive credit if you implement it in a directly recursive manner, or utilize a private static helper method which is recursive.

- Hint: To get from `A` to `B`, there are at most two possible directions we can travel in the first step, namely north and east. For each of these new Intersections `X`, if we can move from `A` directly to `X` in one step, and we know how many paths there are from `X` to `B`, how many paths are there from `A` to `B` which go through `X`? Then how many total paths are there from `A` to `B`?

- Hint: When is there nothing left to do (i.e. there are no smaller subproblems)? If I'm trying to get to a specified destination from my current location, when is there nothing else left to do? Remember that a Path containing a single Intersection `X` is a valid Path which takes you from `X` to `X`.

For each of these methods you may use the provided Javadocs verbatim, but remember to include the appropriate file headers, and to write inline comments to explain the overall design of your methods and helper methods as necessary.

3 Test Methods

Create a new class called PathUtilsTester with the six required `public static boolean` test methods: `testCountPathsNoPath`, `testCountPathsOnePath`, `testCountPathsRecursive`, `testFindAllPathsNoPath`, `testFindAllPathsOnePath`, and `testFindAllPathsRecursive`.

Implement these test methods as specified in the javadocs. Create your own test cases which are different from the provided examples. For the `findAllPaths` tests, make sure that there is both the correct number of Paths, and that the returned Paths exactly match what you expect to see. Remember that the order the Paths appear in the output will not necessarily be exactly the same as in your implementation.

For each of these methods you may use the provided Javadocs verbatim, but remember to include the appropriate file headers, and to write inline comments to explain your test cases. You can also use the following provided driver method to test your implementation.

```
public static void main(String[] args) {
  try (Scanner keyboard = new Scanner(System.in)) {
    int startX, startY, endX, endY;
    String input = "Y";
    while (input.equalsIgnoreCase("Y")) {
      System.out.print("Enter starting X coordinate: ");
      startX = keyboard.nextInt();
      System.out.print("Enter starting Y coordinate: ");
      startY = keyboard.nextInt();
      System.out.print("Enter ending X coordinate: ");
      endX = keyboard.nextInt();
      System.out.print("Enter ending Y coordinate: ");
      endY = keyboard.nextInt();
      Intersection start = new Intersection(startX, startY);
      Intersection end = new Intersection(endX, endY);
      System.out.println("Number of paths from start to end: "
             + PathUtils.countPaths(start, end));
      System.out.println("List of possible paths:");
      for (Path p : PathUtils.findAllPaths(start, end)) {
        System.out.println(p);
      }
      do {
        System.out.print("Try another route? (Y/N): ");
        input = keyboard.next();
      } while (!input.equalsIgnoreCase("Y")
             && !input.equalsIgnoreCase("N"));
    }
  }
}
```

4 Assignment Submission

Congratulations on finishing this CS300 assignment! After verifying that your work is correct, and written clearly in a style that is consistent with the CS300 Course Style Guide, you should submit your final work through Gradescope. The only FOUR files that you must submit include: `Intersection.java`, `Path.java`, `PathUtils.java`, and `PathUtilsTester.java`. Your score for this assignment will be based on your **"active"** submission made prior to the hard deadline. The second portion of your grade for this assignment will be determined by running that same submission against additional offline automated grading tests after the submission deadline. Finally, the third portion of your grade for your submission will be determined by humans looking for organization, clarity, commenting, and adherence to the CS300 Course Style Guide.