|  |
| --- |
|  |
| [CS](https://www.plymouth.edu/department/computer-science/) 2381: Data Structures  (Fall 2020) |
| [**Syllabus**](https://turing.plymouth.edu/~kgb1013/?course=2381)  [**Moodle**](http://www.plymouth.edu/courses/course/view.php?id=51851)  **Teachers**  [Kyle](https://turing.plymouth.edu/~kgb1013/)  **Assignments**  [Project 0](https://turing.plymouth.edu/~kgb1013/?course=2381&project=0)  [Project 1](https://turing.plymouth.edu/~kgb1013/?course=2381&project=1)  [Project 2](https://turing.plymouth.edu/~kgb1013/?course=2381&project=2)  [Project 3](https://turing.plymouth.edu/~kgb1013/?course=2381&project=3)  [Project 4](https://turing.plymouth.edu/~kgb1013/?course=2381&project=4)  [Project 5](https://turing.plymouth.edu/~kgb1013/?course=2381&project=5)  [Project 6](https://turing.plymouth.edu/~kgb1013/?course=2381&project=6)  [Project 7](https://turing.plymouth.edu/~kgb1013/?course=2381&project=7)  [Project 8](https://turing.plymouth.edu/~kgb1013/?course=2381&project=8)  **Other Pages**  [Kyle's Teaching](https://turing.plymouth.edu/~kgb1013/?main=teaching) [Kyle's Schedule](https://turing.plymouth.edu/~kgb1013/?main=schedule) [Kyle's Resources](https://turing.plymouth.edu/~kgb1013/?main=resources) | **Project 4: Your Queue to Start Coding**  **Assigned: Tue Sep 29 2020 Due: 11:59:00 PM on Mon Oct 12 2020 Team Size: 1 Language: Java Out of: 100 points**  In this project, you will implement the Queue data structure and use it to create a player for Rotisserie Nim. Since we are only going to implement very basic Queue operations, we'll name the class PureQueue.  **Part 0, 5 points:**Create a new folder for this project. Create a new file, PureQueue.java, where you will implement your version of a Queue. (Here again is [EmptyClass.java](https://turing.plymouth.edu/~kgb1013/DB/code/EmptyClass.java) if you'd like it.) First, write or update the Javadoc file header to include your own description of the class and an @author tag with your names.  **Part 1, 10 points:**Consider a no-parameter constructor. Write it (Javadoc first, naturally). While doing this (if you haven't already) you'll probably decide that it's time to add a field; a different data structure to use as your underlying data. You can't use one of Java's premade Queues, but you could try using something you've already seen... If you do use an ArrayList, which ends will represent the front and back?  **Part 2, 0 points:**Write a toString method. (This is the last time I'll remind you to Javadoc, but keep at it!) Afterwards, write a main method to be a unit test for your class. Test your constructor and toString.  **Part 3, 15 points:**Implement add. Some types of Queues have restricted size, so there's a chance that the element won't be added. This is why the return type is listed as boolean. Yours won't have this restriction, so you don't need to throw any exception(s) here. What does that mean your method should always return? Once you get it working, add tests to your unit test! (I'll stop reminding you to do this now, but be sure to always unit test new methods after you write them.)  **Part 4, 15 points:**Implement remove, which removes (and returns) the next element. (It returns the same element it removes.) There are times when this should throw a NoSuchElementException. When should it throw this exception? There are two ways to test for throwing: either use a conditional or catch another possible exception!  **Part 5, 15 points:**Implement element, which returns the next element, but doesn't remove it. When should this throw an exception?  **Part 6, 10 points:**Implement isEmpty, which returns whether the queue is empty.  **Part 7, 10 points:**Implement equals. Just like last time, the signature of this method should look like:  public boolean equals(Object other)  Implement this method.  **Part 8, 0 points:**Let's test your code out during actual game play. You'll need some things:   * Abstract CG class: download and compile [CombinatorialGame.java](https://turing.plymouth.edu/~kgb1013/DB/2381/code/AllProjects/CombinatorialGame.java). * Player classes: [Player.java](https://turing.plymouth.edu/~kgb1013/DB/2381/code/AllProjects/Player.java) and [RandomPlayer.class](https://turing.plymouth.edu/~kgb1013/DB/2381/code/AllProjects/RandomPlayer.class). * Abstract class for generation of game states: [PositionFactory.java](https://turing.plymouth.edu/~kgb1013/DB/2381/code/AllProjects/PositionFactory.java). * Referee: [Referee.java](https://raw.githubusercontent.com/paithan/CombinatorialGameCode/master/java/Referee.java).   **Part 9, 0 points:**Download the [RotisserieNim.java](https://turing.plymouth.edu/~kgb1013/DB/2381/code/QueueProject/RotisserieNim.java) source code and put it in your folder. Try to compile it. If it doesn't work, there's probably an issue with your PureQueue class. (Perhaps not, my code isn't perfect! If you really think it's a problem with my code, I recommend talking to me first.) Fix any problems before continuing.  **Part 10, 0 points:**Create your own class, TestingMonkey (or something similar) that has no constructors, only a main method. In that method, create a random Rotisserie Nim player (new RandomPlayer<RotisserieNim>()) and pit it against itself. You can instantiate a PositionBuilder with:  PositionFactory<RotisserieNim> factory = new RotisserieNim.RotisserieFactory(numPiles, maxPileSize);  Make sure everything is in working order before continuing.  **Part 11, 10 points:**Create a new class for your own Rotisserie Nim player, RotisserieNimPlayer.java. Before considering good strategies, focus on creating a player that makes any legal move. Your player should never make any forfeits. Remember:   * Your player should only directly invoke the PureQueue methods assigned here. I'll be testing your player with my own copy of PureQueue.java, so if you call other methods, I won't be able to test your player. * Don't use randomness in your player. (Randomness is a really powerful tool. If you're interested in writing a player that uses randomness, we should definitely talk after this course is finished!) * Don't call the getOptions method.   **Part 12, 0 points:**Change the code in TestingMonkey to replace one of the random players with your player. Run it to be even more certain your player works.  **Part 13, 0 points:**Spend some time playing Rotisserie Nim in your group. What is hard about this game? Do you see any instances where you can write a winning player?  **Part 14, 5 points:**Tweak your player so that it consistently defeats my random player more than 60% of the time.  **Part 15, 5 points:**Download the [RotisserieNimMediumPlayer](https://turing.plymouth.edu/~kgb1013/DB/2381/code/QueueProject/RotisserieNimMediumPlayer.class) class file and put it in the same folder. Tweak your player so that it consistently defeats this mediocre player more than 60% of the time.  **Part 16, 10 points (Bonus):**Download the [RotisserieNimStrongPlayer](https://turing.plymouth.edu/~kgb1013/DB/2381/code/QueueProject/RotisserieNimStrongPlayer.class) class file. Tweak your player so that it consistently defeats my strong player more than 45% of the time.  **Submitting your Project:**  Be careful to follow all these directions precisely in order to ensure I can grade your assignment in a timely manner. If you don't, I may give you zero points for the assignment.  Make sure your code all compiles from the command line:  javac \*.java  Make sure your player class doesn't use any methods of the data structure except for the ones I asked you to write. (I will be testing your player with my own version of the data structure. That way if there are any issues with your data structure you didn't notice, your player can still run correctly.)  Please check that your code only prints to the screen in the tests and not when being tested by the Referee. If it does, please comment those print statements out.  Create a *new* folder to submit your files in. Name this folder *YourLastName*Project4, all in PascalCase. (For example, my folder name would be: BurkeProject4.)  Copy your source code (.java files) into that directory. Your code should be at the top level of that directory (not in any subdirectory). You do not need to submit any extra classes you used to test your structure or player. (I've got really tough tests to do that myself!) I'll be looking for the following files in your folder:   * PureQueue.java * RotisserieNimPlayer.java   Finally, compress your file into a .zip file. (Don't use WinRar!) Upload the zipped file to the project page on Moodle. If you're submitting late, please also send me an email (without the code attached) so I know there's something waiting for me on Moodle. |