|  |  |
| --- | --- |
| Plymouth State Seal | [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 5: Priority Level: Q**  **Assigned: Tue Oct 06 2020 Due: 11:59:00 PM on Mon Oct 19 2020 Team Size: 1 or 2 Language: Java Out of: 105 points**  In this project, you will implement a Priority Queue and use it to create a player for Greedy Nim. Your version will be called PurePriorityQueue.  **Part 0, 0 points:**Create a new folder for this project. Create a new class called PurePriorityQueue.java. Set up the class Javadoc, as always.  **Part 1, 0 points:**This time, your constructor will take only one argument, a [Comparator](http://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html), so be sure to add an import statement before the class JavaDoc. A comparator is an object that can be used to compare two objects of the specified type using a total order. It has a compare method that returns an integer:   * negative, if the first parameter has higher priority than the second. * zero, if the priorities of the parameters is equal. * positive, if the first parameter has lower priority than the second.   For an example, download [GreedyNim.java](https://turing.plymouth.edu/~kgb1013/DB/2381/code/PriorityQueueProject/GreedyNim.java) and check out the anonymous Comparator class defined as a constant (MAX) near the top. Which of the two integers does this give priority to? Alternatively, we could have a min comparator: (Where does the code differ between these two?)  Comparator<Integer> min = new Comparator<Integer>() {  public int compare(Integer left, Integer right) {  return left.intValue() - right.intValue();  }  };  You can declare and use this comparator in your main method like this: (I added another one for good measure.)  public static void main(String[] args) {  //min gives higher priority to the lower number.  Comparator<Integer> min = new Comparator<Integer>() {  public int compare(Integer left, Integer right) {  return left.intValue() - right.intValue();  }  };  //longest gives higher priority to the longer String  Comparator<String> longest = new Comparator<String>() {  public int compare(String a, String b) {  return b.length() - a.length();  }  };  //lexicographical gives higher priority to strings that come first alphabetically  Comparator<String> lexicographical = new Comparator<String>() {  public int compare(String a, String b) {  int minLength = Math.min(a.length(), b.length());  for (int i = 0; i < minLength; i++) {  if (a.charAt(i) < b.charAt(i)) return -1;  if (a.charAt(i) > b.charAt(i)) return 1;  }  //one string is a prefix of the other  return a.length() - b.length();  }  };  PurePriorityQueue<Integer> minQueue = new PurePriorityQueue<Integer>(min);  PurePriorityQueue<String> longerQueue = new PurePriorityQueue<String>(longest);  PurePriorityQueue<String> alphabeticalQueue = new PurePriorityQueue<String>(lexicographical);  }  **Part 2, 5 points:**Write the Javadoc for the constructor (don't forget to tag the comparator parameter). Then write the (one-parameter) constructor itself. Your class should have two fields: the comparator and some kind of data structure that will hold the elements in this priority queue. You may choose to use any data structure we've used or programmed already. (You can always change it later, too.)  **Part 3, 0 points:**Write the toString method. It should at least print out all the elements inside the priority queue though I won't grade based on the order. (This is necessary for me to correctly grade the other parts.)  **Part 4, 0 points:**Start your unit test. Create empty priority queues for both Strings and Integers. You can either create your own separate classes for Comparators, or you can create inner classes like I did for the game position generators, or you can create them anonymously, like this:  Comparator<String> shorterLength = new Comparator<String>() {  public int compare(String left, String right) {  return left.length() - right.length();  }  };  PurePriorityQueue<String> strings = new PurePriorityQueue<String>(shorterLength);  **Part 5, 15 points:**We only have a few methods to implement for priority queues: add, remove, and element. Implement add first. Once you have it written, don't forget to thoroughly unit test it! (This one can be void unlike the Queue project.)  **Part 6, 10 points:**Implement element, which returns the item with the highest priority. In your code, this should throw a NoSuchElementException when there are no elements to choose from.  **Part 7, 10 points:**Implement remove. This should also throw an exception when it's empty.  **Part 8, 0 points:**Optional: implement isEmpty.  **Part 9, 10 points:**It's difficult to test whether two priority queues are equivalent, mostly because it's hard to test whether two comparators are the same (unless they are references to the same object). Thus, instead of an equals method, we'll write a boolean hasSameElementsAs method that takes another PurePriorityQueue as a parameter. Implement this method. Since there is no implicit order on the elements, this may not be as easy as it sounds. After you've gotten this written, I highly recommend testing it on priority queues with these elements (added in the order shown): [5, 1, 5], [1, 5, 5], [1, 5], and [5, 1, 1]. (Only the first two should be equal to each other.) If you can't figure out how to do this one, please come ask me! UPDATE: I've decided to add more about this method, because it's a bit tricky! Some cases I keep forgetting about:   * Even if the lists are sorted, they may not have the same Comparator and might be in different orders. * Sorting the underlying list of a priority queue might cause trouble if it's kept sorted.   Thus, I recommend the following approach:   1. Create two new ArrayLists. 2. Copy the elements in this into one of those lists and the elements of the other priority queue into the other list. 3. Sort those two lists using the sort method in the Collections class. (It doesn't matter which priority queue's comparator you use.) 4. Iterate through both lists and make sure all the elements are the same. (Or, just use the ArrayList's equals method.)   **Part 10, 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 11, 0 points:**Once your class is fully unit-tested, you should compile with [GreedyNim.java](https://turing.plymouth.edu/~kgb1013/DB/2381/code/PriorityQueueProject/GreedyNim.java). Try running that unit test. If you encounter errors, then it's probably the case that your unit test in PurePriorityQueue isn't strong enough. Buff it up and run it again to figure out where your code has gone wrong.  **Part 12, 0 points:**Create a new class to simulate games. Don't include a constructor, because we won't instantiate this class. Instead, include a main method. You can create and test games by doing something like the following:  Player<GreedyNim> random = new RandomPlayer<GreedyNim>();  PositionFactory<GreedyNim> positionGenerator = new GreedyNim.PositionBuilder(5, 8);  Referee<GreedyNim> ref = new Referee<GreedyNim>(random, random, positionGenerator);  ref.call();  **Part 13, 20 points:**Create your own player for Greedy Nim, GreedyNimPlayer.java. Before considering good strategies, focus on creating a player that compiles. The GreedyNim constructor does not take a priority queue as an argument; it takes a List (which could be an ArrayList) instead. It does this because it uses its own priority (choosing the maximum number) and doesn't want to allow a constructor to specify something different. You'll need to write some code to convert your priority queue to an ArrayList before you can create the position your player moves to. Remember:   * Your player should only directly invoke the PurePriorityQueue methods assigned here. I'll be testing your player with my own copy of PurePriorityQueue.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 14, 0 points:**Spend some time playing Greedy Nim in your group. This is another game that can be solved efficiently. See if you can figure out a winning strategy!  **Part 15, 5 points:**Tweak your player so that it consistently defeats the random player more than 60% of the time.  **Part 16, 20 points:**There's a Medium-Difficulty player awaiting on Mimir. (Normally it's something you have to communicate with over the server. Defeat this player   * 40+% of the time: 10 points * 60%+ of the time: 20 points   **Part 17, 10 points:**A perfect Greedy Nim player is also waiting on Mimir. Play against it! Tweak your player to consistently defeat my perfect player:   * 30%+ of the time: 10 points * 47%+ of the time: 20 points (10 is bonus) If you do this, you probably implemented a perfect player! Great job!   **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.   * If you are working alone, Name this folder *YourLastName*Project5, all in PascalCase. (For example, my folder name would be: BurkeProject5.) * If you are working in a group with two people, create a new folder labelled with your two last names, separated by "And", and followed by Project5 all in PascalCase. (For example, if I worked with Kathleen Stock, our folder would be named BurkeAndStockProject5.)   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:   * PurePriorityQueue.java * GreedyNimPlayer.java * GreedyNimRemotePlayer.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. |