|  |
| --- |
| [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 2: The Strongest Link**  **Assigned: Tue Sep 15 2020 Due: 11:59:00 PM on Mon Sep 28 2020 Team Size: 1 Language: Java Out of: 100 points**  In this project, you will implement a Linked List (without using the Java built-in Linked List class). Your class will be very versatile, so we'll use it to play a new game: Myopic Col.  **Part 0, 0 points:**Create a new folder for this project. Create a new file, PureLinkedList.java, where you will implement your version of a linked list. (Here's my [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, 0 points:**Other programmers should be able to create PureLinkedLists like the following:  PureLinkedList<Mouse> meeses = new PureLinkedList<Mouse>(new Mouse("Jerry"));  What will the class signature look like? Will you need to include a generic type?  You: Kyle, are you giving me this line of code so I can copy-paste it directly into my project somewhere?  Me: Um... no. This is for how the class will be used, not any internal code that will make it work.  You: Then why are you telling me this?  Me: There are a bunch of things you should learn from this, including:   * The number of Generic Types your code should have * The name of the class you're creating (though you probably already got that from the filename). * Those first two things should tell you what the signature line of the class will look like, e.g. public class .... * The number of parameters the constructor will take, and * The types of the parameters the constructor will take.   If you're unsure about any of those parts, please ask about them!  **Part 2, 0 points:**Your class should have only two fields, like so:  //the value at the head of this list  private YourGenericVariable value;  //the tail of this list  private PureLinkedList<YourGenericVariable> tail;  Add these fields to your class. (Naturally, you should use a different name than YourGenericVariable.)  **Part 3, 5 points:**Create a one-parameter constructor. This will create a LinkedList with a single node. There are two fields in your class, so you'll want to assign values to both of them. Those two lines will probably be your only two lines of code inside the constructor. Definitely assign something (non-null) to the field this.value. There is no parameter for the this.tail field, so do assign this one to null.  **Part 4, 0 points:**As always, after implementing a constructor, you should implement toString. I've provided this recursive version for you:  public String toString() {  if (this.tail == null) {  return "[" + this.value + "]";  } else {  String tailString = this.tail.toString();  String tailMinusLeftBracket = tailString.substring(1);  return "[" + this.value + ", " + tailMinusLeftBracket;  }  }  I recommend reading through this so that it makes sense to you. Which part is the base case?  **Part 5, 0 points:**Create your main method and start your unit test. In your first test, create a PureLinkedList and then print out the result of its toString method. My code to do that might look like something like this:  public static void main(String[] args) {  //testing the constructor and toString  PureLinkedList<Integer> integers = new PureLinkedList<Integer>(5);  System.out.println(integers.toString());// should print: [5]  }  Run your unit test to make sure things work correctly. After they do, I recommend rewriting your test to follow the framework used in project 0. Then update it again to test that everything works with a linked list of strings (instead of integers).  **Part 6, 10 points:**Check out the code for the toString method again. Do you understand what's happening here? If you're not sure about any part of this, I recommend asking for help.  **Part 7, 10 points:**Write the add method. This takes a single parameter of your generic type and adds it to the end of the linked list in a new node (thus increasing the length by 1). Hint: should this method be recursive?  **Part 8, 0 points:**Update your main method to test linked lists with more than one element. (Whenever you write or change a new method, always update your class test to check that everything's working as it should.)  **Part 9, 5 points:**Write the isLast method. This should take no parameters and return whether this is the last element in the list.  **Part 10, 10 points:**Write the length method, which returns an int, the number of elements in this list. Fun fact: Recursion is Awesome!  **Part 11, 5 points:**Let's do some really simple getters and setter. Write the getFirst method, which returns the first element in the list.  **Part 12, 5 points:**Write the setFirst method, which takes a single parameter and changes the first item in the list to be that value (instead of whatever it previously was). This operation does not change the length of the list.  **Part 13, 5 points:**Write the getTail method. This should return a linked list with one less element, which contains everything except for the first element. (This is probably easier than you think. Like, it's super easy. One line.)  **Part 14, 5 points:**Write a setTail method. This void method should take one PureLinkedList<Something> parameter, and change the subject by resetting it's tail to be the argument. For example, if I have two linked lists of Integer objects, narwhal, [1, 4, -45], and beluga, [7, 1, 32], then after executing:  narwhal.setTail(beluga);  narwhal would be [1, 7, 1, 32].  **Part 15, 5 points:**Write the get method. This takes a single int parameter and returns the element at that index. There are four things to look out for in the method call:   * If the index is negative, you want to throw an IndexOutOfBoundsException. You can throw one of these with a statement like this:   throw new IndexOutOfBoundsException("Your index is a skunky monkey!");  (Don't actually use that message.)   * If the index is zero, then you know you're looking at the correct node. * If your index is too big, you want to throw an index exception * If the index is greater than zero (but not too big) then you want to make the recursive call, which is slightly tricky.   This method should not modify the list in any way. (Seriously, make sure it doesn't modify the list by testing it!) Hint: the base case here is not going to be when the tail is null, but instead when the parameter is zero.  **Part 16, 5 points:**Write the set method. The signature for this should look something like:  public void set(int index, E element)  It should change the indexeth element in the list to equal the element parameter. Just like before, this should throw an IndexOutOfBoundsException if the index is too low or too high.  **Part 17, 10 points:**The next one is a bit tough: the equals method. As always, it should take a single Object parameter and return a boolean. There are ways to write this recursively and non-recursively, but I find the recursive version far easier.  **Part 18, 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 19, 0 points:**Time to test out your code with the game rules! Download the [PathMyopicCol.java](https://turing.plymouth.edu/~kgb1013/DB/2381/code/LinkedListProject/PathMyopicCol.java) source code and compile it. If it doesn't compile, there's probably a problem with your PureLinkedList implementation. Then try to run it. Go over the output very carefully. Again, if the output isn't correct, there are probably some errors that need to be corrected. Make sure you fix all errors before continuing.  **Part 20, 0 points:**Create a separate class to simulate games with a main method. You can create a game of path myopic col and run it by doing something like the following:  public static void main(String[] args) {  Player<PathMyopicCol> random = new RandomPlayer<PathMyopicCol>();  int minSize = 10;  int maxSize = 15;  double density = .2;  PositionFactory<PathMyopicCol> positionGenerator = new PathMyopicCol.PositionBuilder(minSize, maxSize, density);  Referee ref = new Referee(random, random, positionGenerator);  ref.call();  }  What are the rules of this game? Where are the random players making poor moves?  **Part 21, 0 points:**Download my [Easy Player](https://turing.plymouth.edu/~kgb1013/DB/2381/code/LinkedListProject/PathMyopicColEasyPlayer.class). You can create instances of this player by doing something like:  Player<PathMyopicCol> easy = new PathMyopicColEasyPlayer();  Test this out against a random opponent. It should fare worse than random!  **Part 22, 10 points:**Create a new class for your own player, PathMyopicColPlayer.java. Implement the getMove method:  public PathMyopicCol getMove(PathMyopicCol position, int playerId)  You'll again need to use the playerId parameter, which should have value either CombinatorialGame.LEFT or CombinatorialGame.RIGHT. Left colors spaces Blue; Right colors them Red. The "colors" are encoded using these same values. So a blue space actually has integer value CombinatorialGame.LEFT and CombinatorialGame.RIGHT for right. If a node is uncolored, it will have value PathMyopicCol.UNCOLORED .) A Path Myopic Col state includes a bunch of path-graphs, so when you get the state of the game, you actually get a collection of linked lists. You can use position.getPath() to get the PureLinkedList. At this point it's not necessary to make a strategically good move, but your method should always make a legal one from here on out. Remember:   * Your player should only directly invoke the PureLinkedList methods assigned here. I'll be testing your player with my own copy of PureLinkedList.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 23, 0 points:**Change your testing class to use your player. How does it fare against my easy player? Remember that you can add this line to your testing class to run 100000 games:  ref.gauntlet(100000);  I will probably be testing your games on very long boards, so you might want to increase the size of the games that are being generated.  **Part 24, 10 points:**Improve your player so that it consistently beats my:   * Easy player 65% of the time or more: 5 points * Random player 60% or more: 10 points * Random player 75+%: 15 points (5 is bonus)   Your algorithm should work equally well for both the Left and Right player. Make sure to test it against my player both ways!  **Part 25, 10 points (Bonus):**Download my [Perfect Player](https://turing.plymouth.edu/~kgb1013/DB/2381/code/LinkedListProject/PathMyopicColPerfectPlayer.class). You can create instances of this player by doing something like:  Player<PathMyopicCol> perfect = new PathMyopicColPerfectPlayer();  Upgrade your player to consistently beat my perfect player at least 47% of the time. If you do this, congratulations! You've probably written your own perfect player!  **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*Project2, all in PascalCase. (For example, my folder name would be: BurkeProject2.)  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:   * PureLinkedList.java * PathMyopicColPlayer.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. |