|  |
| --- |
| [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 7: Two Paths Diverged in a Tree**  **Assigned: Tue Oct 13 2020 Due: 11:59:00 PM on Mon Nov 02 2020 Team Size: 1 or 2 Language: Java Out of: 180 points**  In this project, you will implement a Binary Tree and then write a player for Myopic Col on trees. Your version of a Binary Tree will be called PureBinaryTree. This will have a recursive structure, similar to our Linked List implementation.  **Part 0, 0 points:**Create a new folder for this project, then create a new file, PureBinaryTree.java. Set up the class Javadoc, as always.  **Part 1, 0 points:**Each binary tree will have exactly three fields: a value at this node (of whatever your generic type is), and two PureBinaryTree<Type>s, which I named leftChild and rightChild. Be sure to make all three private.  **Part 2, 10 points:**Write a constructor that takes only one parameter (of your generic type). Similar to the linked lists, this should initialize both the recursive fields to null.  **Part 3, 0 points:**Let's write the toString in parts. I have two versions, one which takes one parameter, and the other (standard) version which has none. Let's do the one-parameter version first. The parameter will indicate how much this tree is indented when printed out. Add the following method: (you'll need to import java.lang.\*)  public String toString(String indent) {  StringBuilder builder = new StringBuilder();  builder.append(indent + this.value + "\n");  return builder.toString();  }  Later on, we'll modify this method later to make it print out the left and right branches of the tree as well.  **Part 4, 5 points:**We still need a no-parameter toString. We can write this in one line by calling the single-parameter version with the empty string as the argument.  **Part 5, 0 points:**Get a unit test started and create a couple of trees with different generic types. Call the toString method and make sure it's working.  **Part 6, 20 points:**Now let's add some getter and setter methods to retrieve and add values to the tree. First, write getValue, which takes no parameters and returns the value at this node.  **Part 7, 20 points:**Now write setValue, a void method which takes a value as the parameter and sets this node's value to that.  **Part 8, 20 points:**We're going to set up our trees so that we can build them like this:  PureBinaryTree<String> pokemon = new PureBinaryTree<String>("Bulbasaur");  PureBinaryTree<String> psychics = new PureBinaryTree<String>("Mewtwo");  pokemon.setLeftChild(psychics);  psychics.setLeftChild(new PureBinaryTree<String>("Kadabra"));  psychics.setRightChild(new PureBinaryTree<String>("Alakazam"));  PureBinaryTree Alakazam = psychics.getRightChild();  Alakazam.setRightChild(new PureBinaryTree<String>("Abra"));  This code creates a binary tree that looks like this:  Binary Tree of Pokemon Strings  Time to build on to the tree! Implement setLeftChild, a void method which takes a PureBinaryTree<Type> as a parameter and sets the left subtree to that. You can't really test this out a bunch yet, since toString doesn't yet handle the child trees. You'll fix that in the next part.  **Part 9, 0 points:**It's a bit hard to get a tree to print out top-to-bottom on the console, so we'll be printing it out right-to-left. Our root will be on the far left, with it's left subtree above it and the right subtree below, each indented two spaces. (So it's a rotated and flipped version of itself.) The tree pictured above will print out like this:  Kadabra  Mewtwo  Alakazam  Abra  Bulbasaur  Let's get to updating toString so that it prints out the left-hand parts! (We'll add in the right-hand children later.) Change your method to include the new lines in this:  StringBuilder builder = new StringBuilder();  if (this.leftChild != null) {  builder.append(this.leftChild.toString(indent + " "));  }  builder.append(indent + this.value + "\n");  //printing the right subtree will go here  return builder.toString();  Now you should be able to see what happens after you call setLeftChild. Test this out a bunch! You should be able to build very deep trees (even with only left children).  **Part 10, 20 points:**Implement setRightChild, which does nearly the same thing, but for the right-hand subtree.  **Part 11, 0 points:**Finish your toString method by including a part that prints contents of the right child. Add this section after it prints out the value at this vertex.  **Part 12, 15 points:**Implement getLeftChild, a no-parameter method that returns the left subtree. (You don't need to worry about making a copy.)  **Part 13, 15 points:**Implement getRightChild, which does the analagous thing to the right subtree.  **Part 14, 15 points:**I know you've been looking forward to implementing a recursive method. Sorry to keep you waiting; now's the time! Write getHeight, which takes no parameters and returns the number of levels in the tree. (If there are no children, the height should be 1.) How can you determine this using two recursive calls to the two subtrees? Definitely test this thoroughly to make sure you've got it right.  **Part 15, 15 points:**In order to check that trees are equivalent, we need an equals method. There are different ways to write this, but I strongly recommend a recursive solution. Since you'll have to test whether the subtrees are each non-null, I also recommend creating some local boolean variables such as leftSubtreesEqual to help organize your solution. Thoroughly test this to make sure it's working correctly!  **Part 16, 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 17, 0 points:**At this point, you should be able to try compiling with [TreeMyopicCol.java](https://turing.plymouth.edu/~kgb1013/DB/2381/code/BinaryTreeProject/TreeMyopicCol.java), [SwingDisplayable.java](https://turing.plymouth.edu/~kgb1013/DB/2381/code/SwingProjects/SwingDisplayable.java), and [TreeMyopicColPanel.java](https://turing.plymouth.edu/~kgb1013/DB/2381/code/BinaryTreeProject/TreeMyopicColPanel.java).  **Part 18, 0 points:**Create a separate file with no constructor to test that the game works correctly when used with a Referee. To create a factory of game positions for TreeMyopicCol, do something like the following:  PositionFactory<TreeMyopicCol> factory = new TreeMyopicCol.PositionBuilder(3, 6, .3, .2);  Check out the code for the game to see what each of the parameters do. Play around with the parameters to generate different sorts of game states. Run a few games with the referee and two random players and make sure everything is working smoothly.  **Part 19, 15 points:**Create your own player, TreeMyopicColPlayer.java player. Start off by getting getMove to always return a legal move. I highly recommend trying to write this so that it's recursive. Please ask for help if you're having a hard time organizing that! Remember:   * Your player should only directly invoke the PureBinaryTree methods assigned here. I'll be testing your player with my own copy of PureBinaryTree.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 20, 10 points:**Now that you have graphics to make it easier to watch the games, spend some time improving your player. Tweak it so that it consistently defeats my random player:   * 20+% of the time: 5 points * 30%+ of the time: 10 points * 40+% of the time: 15 points (5 is bonus) * 50+% of the time: 20 points * 60+% of the time: 25 points * 70+% of the time: 30 points * 80+% of the time: 35 points * 90+% of the time: 40 points * 95+% of the time: 45 points   **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*Project7, all in PascalCase. (For example, my folder name would be: BurkeProject7.) * 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 Project7 all in PascalCase. (For example, if I worked with Kathleen Stock, our folder would be named BurkeAndStockProject7.)   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:   * PureBinaryTree.java * TreeMyopicColPlayer.java * TreeMyopicColPanel.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. |