|  |
| --- |
|  |
| [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 8: Graphing Out Loud**  **Assigned: Tue Oct 27 2020 Due: 11:59:00 PM on Thu Nov 19 2020 Team Size: 1 or 2 Language: Java Out of: 115 points**  In this project, you will implement a Graph with values on the vertices and then write a player for Col on grid-shaped graphs. You will also use Java Swing graphics to display the state of the game and events to advance the game board.  **Part 0, 0 points:**Download my version of [Vertex.java](https://turing.plymouth.edu/~kgb1013/DB/2381/code/GraphProject/Vertex.java). This models all the necessary parts of a single vertex, including the neighbors. In order to distinguish it from all other vertices in the graph, each vertex has it's own unique integer ID. You'll need to verify this uniqueness in later parts. Generate the Javadoc for Vertex and check it out. I highly recommend writing some code that creates some Vertex objects and calls the methods on them.  **Part 1, 0 points:**Now we're ready to create a new class for a graph object. Create another new file, Graph.java. (As always, feel free to use my [EmptyClass.java](https://turing.plymouth.edu/~kgb1013/DB/code/EmptyClass.java) as a template.) This class will act as a specialized container for the vertices. It should use a generic type indicating the type of data that each vertex contains. That means the signature should look something like this:  public class Graph<Mouse extends Object> {  (You should use a different type name.)  **Part 2, 0 points:**You need your graph class to have a field that holds a bunch of vertices. I recommend using a collection of them:  private Collection<Vertex<WhateverYourGenericTypeIs>> vertices;  Add a field to your class.  **Part 3, 20 points:**The constructor should take an Iterable<Vertex<T>> (or whatever you want to name your generic type) and add it to it's own collection of vertices. That means we should be able to create a graph with some code like this:  ArrayList<Vertex<String>> stringVertices = new ArrayList<Vertex<String>>();  stringVertices.add(new Vertex("hi", 0));  stringVertices.add(new Vertex("bananarama", 35));  Graph<String> strings = new Graph<String>(stringVertices);  Before it adds each one, it should check to make sure it doesn't already have a vertex with that ID. I did this by first making a set of the IDs and making sure there were no duplicates. If there is a duplicate, I throw an IllegalArgumentException. For example, the following code will throw and catch the exception:  ArrayList<Vertex<String>> stringVertices = new ArrayList<Vertex<String>>();  stringVertices.add(new Vertex("hi", 52));  stringVertices.add(new Vertex("bananarama", 52));  try {  Graph<String> strings = new Graph<String>(stringVertices);  System.out.println("This line should never print!");  } catch (IllegalArgumentException e) {  System.out.println("Threw and caught the IllegalArgumentException. Good!");  }  Write the constructor.  **Part 4, 0 points:**Add a meaningful toString method. My version includes the value and ID of each node (in a separate line) as well as the IDs of each neighbor.  **Part 5, 10 points:**Add a getter for the vertices: public Collection<Vertex<T>> getVertices(). We want this to be a Collection and not just an Iterable because we will want to be able to invoke the size method later on.  **Part 6, 20 points:**Graphs don't necessarily keep track of the vertices in an ordered list (though you might be doing this under-the-hood). Thus, it's not appropriate to have a get(int index) method. Instead we'll include a getVertexById(int id). Write this method. If no such ID is included by a vertex, your method should throw a NoSuchElementException.  **Part 7, 20 points:**Time for equals! The things to check here are whether the two graphs have the same number of vertices and whether those vertices are equal. The two lists of vertices may not be in the same order, so use getVertexById to "index" into the other graph (instead of just traversing the other graph's data structure of vertices).  **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:**Now that you've got the data structure(s) worked out, download [Col.java](https://turing.plymouth.edu/~kgb1013/DB/2381/code/GraphProject/Col.java), and try compiling again. The main method for Col should run happily. (It generates the board randomly, so you may want to test it multiple times.) If there are any errors (exceptions, compile-time errors, and/or semantic errors) there are probably problems with your classes. Check your unit tests carefully to find and correct those problems.  **Part 10, 0 points:**As we've done before, create a separate class to use for testing. In the main method of that class, create a random Col player, a regular (non-graphical) referee and test out a few games to make sure everything is hunky-dory.  **Part 11, 0 points:**Visually displaying a general graph is a tough problem (and its own research field). Thus, we're going to restrict our attention to Col on grid-graphs. Download [GridCol.java](https://turing.plymouth.edu/~kgb1013/DB/2381/code/GraphProject/GridCol.java), [SwingDisplayable.java](https://turing.plymouth.edu/~kgb1013/DB/2381/code/SwingProjects/SwingDisplayable.java) and [GridColPanel.java](https://turing.plymouth.edu/~kgb1013/DB/2381/code/GraphProject/GridColPanel.java). Try compiling and then run the unit test in GridCol.java. Ensure that everything is working happily before continuing.  **Part 12, 0 points:**Switch your testing class's main method to use GridCol instead of Col and run a few games.  **Part 13, 0 points:**The paintComponent method in GridColPanel.java is incomplete. It draws a grid of circles, but neither draws the connections between vertices nor colors the circles. Nevertheless, let's test it to make sure it's drawing correctly! Download [RefereeWithSwingDisplay](https://turing.plymouth.edu/~kgb1013/DB/2381/code/SwingProjects/RefereeWithSwingDisplay.java), and replace the referee in your testing class with a referee that uses the graphical display:  new RefereeWithSwingDisplay<GridCol>(random, random, factory);  I highly recommend against running a gauntlet with this referee; it will slow things down and could even crash the code. Run a few games to see what the graphics do, even though it won't be very illuminating.  **Part 14, 30 points:**Now (finally) we're ready to start thinking about writing our own player for Grid Col. Create another new file to house your player, GridColPlayer.java, and be sure to implement Player<GridCol>. (As always, feel free to use my [EmptyClass.java](https://turing.plymouth.edu/~kgb1013/DB/code/EmptyClass.java) as a template.) As with every single other project we've done, first concentrate on making a legal move. As in previous projects, there are three color constants you can use:   * Col.BLUE * Col.RED * Col.UNCOLORED   Remember:   * Your player should only directly invoke the Graph methods assigned here. I'll be testing your player with my own copy of Graph.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 15, 15 points:**Tweak your player so that it consistently defeats my random player:   * 25+% of the time: 10 points * 35+% of the time: 15 points * 45+% of the time: 20 points (5 is bonus) * 55+% of the time: 25 points * 65+% of the time: 30 points * 75+% of the time: 35 points * 85+% 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*Project8, all in PascalCase. (For example, my folder name would be: BurkeProject8.) * 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 Project8 all in PascalCase. (For example, if I worked with Nancy Borgin, our folder would be named BorginAndBurkeProject8.)   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:   * Vertex.java * Graph.java * GridColPlayer.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. |