Perfect TicTacToe

1. Problem Description

You are to develop a program that plays the perfect game of TicTacToe. *It should never lose*. Your project can be done in three steps of increasing complexity.

- **Basic Level (for a maximum grade of 75%):** Develop the Tree ADT as an array-based concrete implementation.
- **Intermediate Level (for a maximum grade of 88%):** Your program will then create every possible TicTacToe board configuration organized as a tree.
- **Advanced Level (for a maximum grade of 100%):** Once the configurations have been calculated, you will employ a *minimax* algorithm to evaluate each position. When playing the game, the user will go first (X) and the computer (O) must respond with the move that maximizes its chances of victory. If done correctly, the computer will never lose. However, most games played with a competent human will end in a draw (a "Cat's Game").

For this assignment, there is no code to start with; you are to develop everything. However, the UML design and test drivers for the Tree ADT have been supplied so you can test your implementation. Your final Java program (*Adv. Level*) will prompt the user for moves until the game is over. You can review the sample output below to get an idea of the interaction.

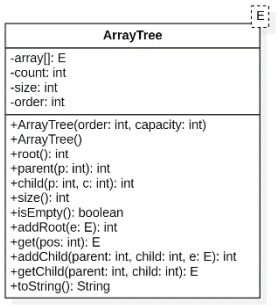
Depending on the level you choose, you are to turn in the following:

- Basic Level: Create the Array-based Tree ADT. Run it with `TestTreeADT.java` to insure it works. Turn in `ArrayTree.java`.
- Intermediate Level: Generate a Game Tree of all the possible moves but only print out the first 100. The code for this is shown below in section 2.2. The root of the tree is the empty board. Turn in `ArrayTree.java` and `GenTree.java` (which will include the `main` method which prints the first 100 board positions).
- Advanced Level: Write the program that interacts with the user and walks the Game Tree. Turn in `ArrayTree.java` (with the `main` method), `GenTree.java` and `TicTacToe.java`.

2. Additional Instructions

2.1. Basic Level: Array Tree ADT

Develop an `ArrayTree` class to meet the design shown here. It *should not* implement the `Tree<E>` interface we discussed in class.

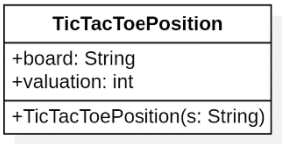


The behavior of each method should be clear from our discussion in class. While the class is generic (element is type E), the positions are all primitive integers. The default order is 2 and the default capacity is 1000. `toString()` should print a summary of the data elements as shown in the sample output below.

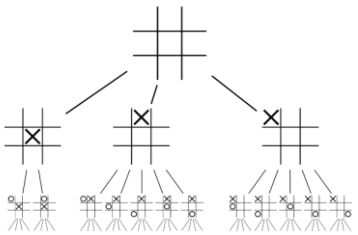
Running your ArrayTree with `TestTreeADT.java` should result in the following output. Note: it's ok that you get one **`Fail`** in testing a2. You should understand why that is expected as you implement the TreeADT.

```
*** Test ArrayTree ADT ***

Pass: a1.isEmpty()
Pass: a1.root()
Pass: a1.parent()
Pass: a1.child()
Pass: a1.size()
Pass: a1.isEmpty()
Pass: a1.get()
Pass: a1.getChild()
Small Tree: [ArrayTree: order=2, count=21, size=25, array={1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
17 18 19 20 21 - - - - }]

Pass: a2.isEmpty()
Pass: a2.root()
Pass: a2.parent()
Fail: a2.child(), expected 75, got -1
Pass: a2.size()
Pass: a2.isEmpty()
Pass: a2.get()
Pass: a2.getChild()
Large Tree: [ArrayTree: order=5, count=51, size=55, array={22 23 24 25 26 27 28 29 30 31 32 33 34
35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67
68 69 70 71 72 - - - - }]
```

2.2. Intermediate Level: Generating the Game Tree



Develop the element type that will represent each board position. In addition to representing the state of the TicTacToe board, you will also need an integer to store the evaluation of the position in Step 2.3 below. I recommend something as simple as the `TicTacToePosition` shown here. I chose to represent the board as a String of nine characters, each position either a *blank*, an 'X' or an 'O'.



The root of your tree will be the blank board. There are 9 children from here. That is, the human player can select to place their X in any position 0 through 8. Create these children. From there, each of these positions has at most 8 counter moves for O. This continues down until there are no moves left. We will ignore symmetries.

There are at most 9! + … 1! (986,409) board configurations. Our tree will need to be order 9 and this will result in many empty positions. You should start your tree out with the capacity set at a staggering 500,000,000 places!

To perform this step, you may want to study and adapt the breadth-first traversal algorithm. Remember not to generate moves for positions that are already occupied.

If this is as far as you get, you will want to prove that your solution works. Print out the first 100 positions in the array. Perhaps you could use code like the following (adapted to any data types/classes you have used). Put this code in the `main` method of a class called `GenTree.java`.

```
for (int i = 0; i < 100; i++) {
   TicTacToePosition t = gameTree.get(i);
   if (t != null)
      System.out.printf("%2d) %s\n", i, gameTree.get(i).board);
   else
      System.out.printf("%2d) Empty\n", i);
}
```

Your result should be as follows (I'm showing only the first 30 spots; you will display 100):

```
 0)
 1) X
 2)  X
 3)   X
 4)    X
 5)     X
 6)      X
 7)       X
 8)        X
 9)         X
10) Empty
11) XO
12) X O
13) X  O
14) X   O
15) X    O
16) X     O
17) X      O
18) X       O
19) OX
20) Empty
21)  XO
22)  X O
23)  X  O
24)  X   O
25)  X    O
26)  X     O
27)  X      O
28) O X
29)  OX
```

2.3. Advanced Level: Evaluating the Positions – Minimax

For each position in the tree, evaluate whether X (the human) will win, or O (the computer) will win by using the minimax algorithm. For each level in which you are choosing a move for the computer, you want to maximize the computer's outcome. Conversely, for each level when the human will be picking, you assume the human picks the move that maximizes their outcome.

For more information, you can review:

- Problem P-8.67 in the book
- https://en.wikipedia.org/wiki/Minimax
- https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-3-tic-tac-toe-ai-finding-optimal-move/

Also, I have provided a high-level algorithm below. Notice that human wins result in a negative value while computer wins result in a positive value. We will use that when playing the game in Step 2.4.

Algorithm evaluatePosition(Position P, int level):

If P is a winning position for X then return -1
else if P is a winning position for O then return +1
else if P is a draw position then return 0;

if this is an odd then level result ← +∞
else result ← -∞

for each child C of P:
evaluatePosition(C, level + 1);
if this is an odd level then result ← minimum of the current result and C's result
else result ← maximum of the current result and C's result

2.4. Advanced Level: Playing the Game

Play the game with the human opponent, allowing them to go first. Internally, keep track of where you are in the tree starting with the root: the empty game. When the user selects a spot, move down the tree to the appropriate child. The computer should then take the best child (value of +1) from that position. Continue until the game has been won or a draw is reached.

3. Notes

- Do not create these classes in a package. For Eclipse, this means using the *default* package.
- Turn in only your Java files as appropriate for your level: ArrayTree.java, GenTree.java and TicTacToe.java.

4. Required Main Classes

Each of the following classes has a main:
TestTreeADT (written for you), GenTree, TicTacToe

5. Required Input for Advanced Level

A series of board positions (0-8). Do not allow the selection of a board position which is already occupied (see Example 2).

6. Required Output for Advanced Level

Your output should look like the following examples. It must include your name. User input is shown in GREEN.

Example 1:

```
Perfect TicTacToe – Your Name

 0 | 1 | 2
-----------
 3 | 4 | 5
-----------
 6 | 7 | 8

Your Move (X):
What position (0-8)? 1
   | X |
-----------
   |   |
-----------
   |   |

My Move (O):
 O | X |
-----------
   |   |
-----------
   |   |

Your Move (X):
What position (0-8)? 4
 O | X |
-----------
   | X |
-----------
   |   |

My Move (O):
 O | X |
-----------
   | X |
-----------
   | O |
```

```
Your Move (X):
What position (0-8)? 3
 O | X |
-----------
 X | X |
-----------
   | O |

My Move (O):
 O | X |
-----------
 X | X | O
-----------
   | O |

Your Move (X):
What position (0-8)? 2
 O | X | X
-----------
 X | X | O
-----------
   | O |

My Move (O):
 O | X | X
-----------
 X | X | O
-----------
 O | O |

Your Move (X):
What position (0-8)? 8
 O | X | X
-----------
 X | X | O
-----------
 O | O | X

Winner is: Draw
```

Example 2:

```
Perfect TicTacToe – Your Name

 0 | 1 | 2
-----------
 3 | 4 | 5
-----------
 6 | 7 | 8

Your Move (X):
What position (0-8)? 1
   | X |
-----------
   |   |
-----------
   |   |

My Move (O):
 O | X |
-----------
   |   |
-----------
   |   |

Your Move (X):
What position (0-8)? 2
 O | X | X
-----------
   |   |
-----------
   |   |
```

```
My Move (O):
 O | X | X
-----------
 O |   |
-----------
   |   |

Your Move (X):
What position (0-8)? 0
That position is already taken.
What position (0-8)? 4
 O | X | X
-----------
 O | X |
-----------
   |   |

My Move (O):
 O | X | X
-----------
 O | X |
-----------
 O |   |

Winner is: O
```