

CMPT225, Spring 2021
Homework Assignment 3
Due date: Tuesday, March 9, 2021, 23:59

You need to implement the following classes:

Part 1:

- *arithmetic.ArithmeticExpressions.java*

Part 2:

- *binarytree.BinaryTree.java*
- *binarytree.BTNode.java*

Note that all files must be under the correct packages (folder).

You may add more classes to your solution if necessary.

Submit all your files in **assignment3.zip** to CourSys. Make sure your zip file can be unzipped using the command “*unzip assignment3.zip*” in CSIL. All your solutions in the zip file must be under the **src** folder, same as in the provided *hw3-cmpt225-spring21.zip*.

That is, the zip file needs to contain the following files (and additional files if needed)

- *src/arithmetic/ArithmeticExpressions.java*
- *src/binarytree/BinaryTree.java*
- *src/binarytree/BTNode.java*

Discussion with others: You may discuss the assignment with your classmates/tutors (or anyone else), but coding must be entirely your own.

References: You may use textbooks, wiki, stack overflow, geeksforgeeks, etc. If you do, specify the references in comments. Posting questions online asking for solutions (e.g. using chegg.com) is prohibited.

Readability: Your code should be readable using the standard Java conventions. Add comments wherever is necessary. If needed, write helper functions or add classes to improve readability.

Compilation: Your code MUST compile in CSIL using javac.

Make sure that your code compiles without warnings/errors.

If the code does not compile in CSIL the grade on that part will be 0 (zero).

Even if you can't solve a problem completely, make sure it compiles.

The assignment will be graded mostly **automatically**, with some exceptions.

Do not add main() to your solutions. The main() method will be in the test files.

Warnings: Warnings during compilation will reduce points.

More importantly, they indicate that something is probably wrong with the code.

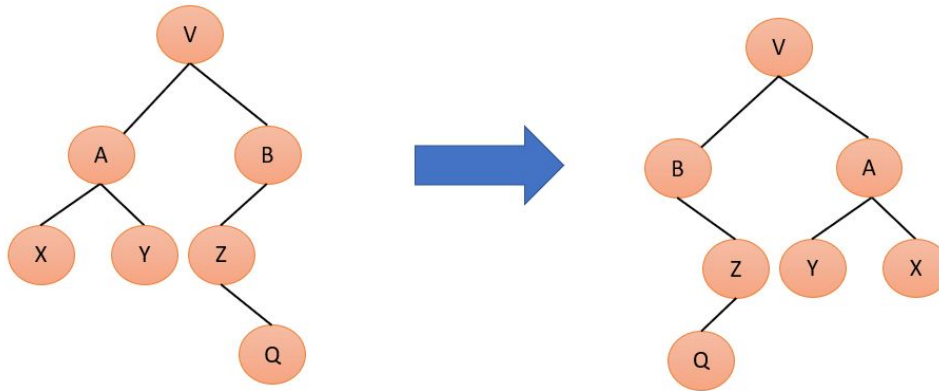
Testing: Test your code. Examples of tests are included. Your code will be tested using the provided tests as well as additional tests. You should create more tests to check your solution.

Good luck!

Part 1 [60 points] - Binary Trees

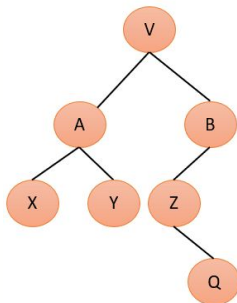
- a) [20 points] In the class `BinaryTree` write the method
public void `mirrorInverse()`

It transforms the tree as in a mirror image. For example:



- b) [20 points] In the class `BinaryTree` write the method
public `<T> BinaryTree<T> createFromPreorderInorder(List<T> preorder,`
`List<T> inorder)`

It gets two lists representing preorder and inorder of a binary tree, and reconstructs the tree. For example: if preorder is [V,A,X,Y,B,Z,Q] and inorder is [X,A,Y,V,Z,Q,B], then the function returns the following tree:



- c) [20 points] Write an iterator for the class `BinaryTree` that performs inorder traversal on the tree. The iterator is called using the method
public `Iterator<T> inorderIterator()`

The iterator must be dynamic in the following sense: if after the iterator is created, and the tree changes in some part that has not been processed by the iterator, then the iterator will see these changes and output the values in the updated tree. However, if you change the nodes that have already been processed, then it will have no effect on the iterator.

**** Be careful, do not change the nodes that are currently being processed, as this may have unexpected effects**

In particular, if you simply create a list with the inorder traversal of the tree in the constructor of the iterator, then it will not work in the dynamic sense.

Part 2 [40 points] - Evaluating arithmetic expressions

- a) In the class ArithmeticExpressions write the method
public static String infix2Prefix(String infixExpression)

The method gets an arithmetic expression in the infix notation, and returns it in prefix notation. All tokens must be separated by a single space.
See examples in the test files.

- b) In the class ArithmeticExpressions write the method
public static BTNode<Double> postfix2BinaryTree(String postfixExpression)

The method gets an arithmetic expression in the postfix notation, and returns a binary tree representing the expression. All tokens are separated by a single space.
See examples in the test files.