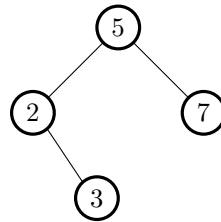


## Lab 6: binary trees

For the lab this week, your task will be to complete the file `BTreeExercises.java`. Use the main function to test your functions and show your examples if you want to sign off.

### 1. Some examples

- (a) In the `main` function, two examples `example1` and `example2` are defined. Draw the corresponding structures. Do these structures have sharing? Cycles?
- (b) Write some code that represent the following tree in a variable `example3`:



- (c) Run the `main` function and try to understand the output of `toString`. Try to define a variable `example4` such that `example4.toString()` is `"(2 (4)) 5 (2)"`.
  - (d) Define a further `example5` with a cycle. For testing, don't try to call `toString`; but it should be the case that `example5.hasCycle()` is `true`, so you can test that.
2. **Basic methods** In this question we look at starting to flesh out the class `BTree`

- (a) Write the body of the method  

```
public int depth()
```

that returns the depth/height of the tree, i.e. the longest path from the root of the tree to a leaf.
- (b) Write the body of the method  

```
public boolean equals(BTree<T> otherTree)
```

that checks whether the calling object represents the same tree as `otherTree` (it should *not* be the same as `==`; think of what you know about `String`)
- (c) Write the body of the method  

```
public ArrayList<T> prefixTraversal()
```

that converts a list to an array using a prefix traversal.

- (d) Write the body of the method

```
static public <T> BTree<T> of(ArrayList<T> arr)
```

that converts an array to a tree using the scheme presented in the lecture/the scheme in the last lab sheet presented in the last lab sheet to represents full binary trees using arrays (a `null` value in the array indicates the absence of any subtree).

3. **Challenge: more advanced functions** Let us look at some more functions of the class `BTree`

- (a) Write the body of the method

```
public <U> BTree<U> map(Function<T, U> f)
```

that returns the tree obtained by relabelling uniformly all the nodes of the calling tree using `f` (recall you can use `f.apply(bla)` to apply `f`).

- (b) Write the body of the method

```
static public BTree<Integer> of(String st)
```

that should convert a `String` to a tree; this should be the inverse of the function `toString` that is provided

- (c) Write the body of the method

```
public void unshareSubtrees()
```

that removes sharing from a structure with no cycles, but keeps the same underlying unfolding (so the output of the provided `toString` method should remain the same)

- (d) Change the code of the `toString` function so that it prints out useful informations in case there are cycles. You could for instance write the result as some sort of system of recursive equations, as

$$(x) 5 ((2) 4 (x)) \quad \text{where} \quad \begin{cases} x &= ((5) x) 7 y \\ y &= y 4 (x 5 y) \end{cases}$$