

This lab has **five (5)** checkpoints.

Learning Outcomes

Upon completing this lab, students should be able to

- Implement a BST of simple objects.
- Use a variable in a higher scope to count objects created and provide serial numbers.
- Filter results in BST methods based on object/node predicates.

Introduction

1. Given a *binary search tree*, if the node contains a `String` and an `int` and the `int` is set to the number of nodes inserted into the tree **before** this node, *draw* th structure of tree resulting after inserting elements in the given order.

- (a) dog elephant flamingo cat aardvark eagle

Example:

```
aardvark: 4
cat: 3
dog: 0
eagle: 5
elephant: 1
flamingo: 2
```

- (b) lego doll erectorSet mask squirtGun ball

Solution:

```
ball: 5
doll: 1
erectorSet: 2
lego: 0
mask: 3
squirtGun: 4
```

- (c) orange red green blue yellow indigo violet

Solution:

```
blue: 3
green: 2
indigo: 5
orange: 0
```

```
red: 1
  violet: 6
  yellow: 4
```

(d) red orange yellow green blue indigo violet

Solution:

```
blue: 4
green: 3
  indigo: 5
orange: 1
red: 0
  violet: 6
  yellow: 2
```

(e) blue green indigo orange red violet yellow

Solution:

```
blue: 0
green: 1
  indigo: 2
    orange: 3
      red: 4
        violet: 5
          yellow: 6
```

✓ Show the lab instructor your pretty BSTs.

2. Initialize a Gradle application project in a new directory. Also initialize git in the root of the project.

The name of the project: countries. Starting package: countries.

Change the name of countries.App to countries.TreeClient.

Delete everything in src/test/java so that your newly renamed application can be built with gradle build.

Add the run block to build.gradle that connects gradle's standard input to that of your program when using gradle run.

Confirm that you can build your application with gradle build.

Confirm that you can run your program with gradle run -q --console=plain.

Commit your working initial version to your local git repository.

Modify countries.TreeClient.main to

- check that there is exactly one command-line argument. Terminate the program with an error code if not.
- create a `TreeClient` object with the one command-line argument passed as the filename from which input will be read.
- Call `run` on the new object.

Compile and run your code.

✓ Show the lab instructor your working code run with `gradle run -q --console=plain --args=.././data/`

3. Add a new class, `countries.Country`:

```
class Coutry
    implements Comparable<Country> {
    public final String name;
    public final String continent;
    public Country(String theName, String theContinent) ...
    // compare by country name
    public int compareTo(Country that) ...
    // name | continent
    public String toString() ...
}
```

Implement the missing functions.

✓ Show the lab instructor your code. Nothing new runs yet.

4. Add the method `countries.TreeClient.readFile` that opens the file name passed into the constructor and reads it line by line.

```
while (next line)
    if line is blank
        continue
    // line is of the form
    // continent | country
    split line on "|"
    construct Country(country, continent)
    print the Country
```

If the line is not empty, use the `String.split` method to split the string on the *pipe* character (the vertical bar): |¹

Be sure to handle the file not being found.

Call `readFile` from `run` and see the code print out the countries data file contents.

✓ Show the lab instructor your code. Compile and run and show it working with `gradle run -q --console=plain --args=.././data/c1.txt`.

¹This may not have been the best character for Dr Ladd to have used: to use it in `String.split`, it has to be escaped: `"\\|"`.

5. Add the final class to this program: `Tree`. The `Tree` is a BST containing `Country` objects. You should implement the *constructor*, *add*, and *toString* first.

Modify `countries.TreeClient.readFile` so that instead of printing each new `Country`, it is added to a `Tree` (you need a `Tree` field in the class).

In run, after `readFile`, print the tree.

✓ Show the lab instructor your code. Compile and run and show it working with `gradle run -q --console=plain --args=../../data/c1.txt`.

6. Copy `Tree.toString` into another method, `String Tree.onContinent(String continent)`. It should return a string just like `toString` **but** only with countries that have the given continent value.

Add a line to the bottom of `TreeClient.run` that prints all the countries in Europe.

✓ Show the lab instructor your code. Compile and run and show it working with `gradle run -q --console=plain`.

7. [Not a separate checkpoint but necessary] Add and commit your full gradle project (if you `git add` just the root directory, `.`, `git` will add all subdirectories recursively).

Create a repository in your Gitea turn-in organization, associate it with your lab work, and push the repo. This will be done at the end of every lab.