Problem statement

Write a command line program that implements Unix command `tree` like functionality.

The program should have following features.

Story 1

Print tree structure recursively with proper formatting for a given directory. Also print a summary at the end.

```
➜  student-grading-java git:(main) tree src
src
├── main
│   ├── java
│   │   └── in
│   │       └── one2n
│   │           └── exercise
│   │               ├── Grade.java
│   │               ├── Grader.java
│   │               └── Student.java
│   └── resources
└── test
    ├── java
    │   └── in
    │       └── one2n
    │           └── exercise
    │               └── GraderTest.java
    └── resources
        └── grades.csv

12 directories, 5 files
```

Assumptions: - Your program behaviour should be same as `tree` command behaviour. - The output is printed on STDOUT

Expectations: - Write test cases for empty directories, nested empty directories, directories with multiple files, etc. - Your code should handle errors when there’s no directory reading (or execute) permission for nested directories.

Story 2 (listing option)

Print relative path to the directory being searched.

```
➜  student-grading-java git:(main) tree -f src
src
├── src/main
│   ├── src/main/java
│   │   └── src/main/java/in
│   │       └── src/main/java/in/one2n
│   │           └── src/main/java/in/one2n/exercise
│   │               ├── src/main/java/in/one2n/exercise/Grade.java
│   │               ├── src/main/java/in/one2n/exercise/Grader.java
│   │               └── src/main/java/in/one2n/exercise/Student.java
│   └── src/main/resources
└── src/test
    ├── src/test/java
    │   └── src/test/java/in
    │       └── src/test/java/in/one2n
    │           └── src/test/java/in/one2n/exercise
    │               └── src/test/java/in/one2n/exercise/GraderTest.java
    └── src/test/resources
        └── src/test/resources/grades.csv

12 directories, 5 files
```

Assumptions: - TODO

Expectations: - Handle argument parsing in the code - How will you write test cases for this code? - Can you reuse code from previous story if possible? How will you refactor your existing code to make this code-reuse possible?

Story 3 (listing option)

Only print directories, not files.

```
➜  student-grading-java git:(main) tree -d src
src
├── main
|   ├── java
|   |   └── in
|   |       └── one2n
|   |           └── exercise
|   └── resources
└── test
    ├── java
    |   └── in
    |       └── one2n
    |           └── exercise
    └── resources

12 directories
```

Assumptions: - TODO

Expectations: - Reuse code from previous stories as much as possible. Make your code modular and extensible. - Write test case for this story.

Story 4 (listing option)

Allow travesing specified nested levels only.

```
➜  student-grading-java git:(main) tree -L 3 src
src
├── main
|   ├── java
|   |   └── in
|   └── resources
└── test
    ├── java
    |   └── in
    └── resources
        └── grades.csv

8 directories, 1 file
```

Assumptions: - TODO

Expectations: - As mentioned in previous stories, we need to reuse code as much as possible. We should also write modular and extensible code.

Story 5 (file option)

Print file permissions for all files.

```
➜  student-grading-java git:(main) tree -p src
src
├── [drwxr-xr-x]  main
│   ├── [drwxr-xr-x]  java
│   │   └── [drwxr-xr-x]  in
│   │       └── [drwxr-xr-x]  one2n
│   │           └── [drwxr-xr-x]  exercise
│   │               ├── [-rw-r--r--]  Grade.java
│   │               ├── [-rw-r--r--]  Grader.java
│   │               └── [-rw-r--r--]  Student.java
│   └── [drwxr-xr-x]  resources
└── [drwxr-xr-x]  test
    ├── [drwxr-xr-x]  java
    │   └── [drwxr-xr-x]  in
    │       └── [drwxr-xr-x]  one2n
    │           └── [drwxr-xr-x]  exercise
    │               └── [-rw-r--r--]  GraderTest.java
    └── [drwxr-xr-x]  resources
        └── [-rw-r--r--]  grades.csv

12 directories, 5 files
```

Assumptions: - TODO

Expectations: - TODO

Story 6 (sorting option)

Sort the output by last modification time instead of alphabetically (the default). Note that, the actual output is just for indication only.

```
➜  student-grading-java git:(main) ✗ tree -t src
src
├── main
│   ├── java
│   │   └── in
│   │       └── one2n
│   │           └── exercise
│   │               ├── Grade.java
│   │               ├── Grader.java
│   │               └── Student.java
│   └── resources
└── test
    ├── java
    │   └── in
    │       └── one2n
    │           └── exercise
    │               └── GraderTest.java
    └── resources
        └── grades.csv

12 directories, 5 files
```

Assumptions: - TODO

Expectations: - TODO

Story 7 (XML/JSON option)

Print output in xml format. - `-X` Turn on XML output. Outputs the directory tree as an XML formatted file. - `-J` Turn on JSON output. Outputs the directory tree as an JSON formatted array.

```
➜  student-grading-java git:(main) ✗ tree -X -L 4 src
<?xml version="1.0" encoding="UTF-8"?>
<tree>
  <directory name="src">
    <directory name="main">
      <directory name="java">
        <directory name="in">
          <directory name="one2n">
          </directory>
        </directory>
      </directory>
      <directory name="resources">
      </directory>
    </directory>
    <directory name="test">
      <directory name="java">
        <directory name="in">
          <directory name="one2n">
          </directory>
        </directory>
      </directory>
      <directory name="resources">
        <file name="grades.csv"></file>
      </directory>
    </directory>
  </directory>
  <report>
    <directories>10</directories>
    <files>1</files>
  </report>
</tree>
```

Assumptions: - TODO

Expectations: - TODO

Story 8 (graphics option)

Do not print the indentation lines, typically used in conjunction with the -f option. Also removes as much whitespace as possible when used with the -J or -x options.

```
➜  student-grading-java git:(main) ✗ tree -if src
src
src/main
src/main/java
src/main/java/in
src/main/java/in/one2n
src/main/java/in/one2n/exercise
src/main/java/in/one2n/exercise/Grade.java
src/main/java/in/one2n/exercise/Grader.java
src/main/java/in/one2n/exercise/Student.java
src/main/resources
src/test
src/test/java
src/test/java/in
src/test/java/in/one2n
src/test/java/in/one2n/exercise
src/test/java/in/one2n/exercise/GraderTest.java
src/test/resources
src/test/resources/grades.csv

12 directories, 5 files
```

Assumptions: - TODO

Expectations: - TODO

Feel free to make suitable assumptions if needed, ensure to document them in README.md

Overall criteria for evaluation:

- Use any of Go, Java, Ruby, Python to write your code.
- Add unit tests for all stories and functions
- Write clean and readable code
- Adherance to coding standards and guidelines
- Treat it as production code, so use best practises what you can think of.