CS 245 Homework 1

Treemap Part 2

due before October 11

1 Overview

In this two-part homework, you will explore the concepts of inheritance and recursion by constructing a treemap, which can be used to show the relative sizes of files in a directory.

What is a treemap? Consider the following tree structure, which consists of a parent with just three immediate children:

```
9
  1
  2
  6
```

(a) Sizes

```
FertileTree tree = new FertileTree();
tree.addChild(new SterileTree(1));
tree.addChild(new SterileTree(2));
tree.addChild(new SterileTree(6));
```

(b) Code

(c) Treemap

The first child has size 1, and it is given $\frac{1}{9}$ of the treemap to plot itself into. The second has size 2; it is given $\frac{2}{9}$. The last size 6; it is given $\frac{2}{3}$. Each sterile tree, a tree without children, is depicted in the treemap as a solid rectangle of color sized according to its proportion within its parent.

What happens when child trees themselves have children? Suppose the third child were itself a tree with four children, of sizes 1, 1, 1, and 3. Instead of the third child being plotted as a solid block of color, it is partitioned into four cells, the sizes of which reflect the relative sizes of its children:

```
9
  1
  2
  6
    1
    1
    1
    3
```

(a) Sizes

```
FertileTree child3 = new FertileTree();
child3.addChild(new SterileTree(1));
child3.addChild(new SterileTree(1));
child3.addChild(new SterileTree(1));
child3.addChild(new SterileTree(3));

FertileTree tree = new FertileTree();
tree.addChild(new SterileTree(1));
tree.addChild(new SterileTree(2));
tree.addChild(child3);
```

(b) Code

(c) Treemap

Note that the third child consumes the same amount of space within its parent as before. Its size has not changed—it is still 6. It just happens to be subdivided amongst its children. Note also that the subdivision happened vertically instead of horizontally. This switch of orientation lets us see that we have gone down a level in the tree. Without the switch, we might think the root of the tree had six children.

Let's add another level. The first child of the third child formerly had size 1. Let's replace that with a tree that has two children of size 5, a tree of size 10. This change will change the parent sizes as well, causing a change in subdivisions all the way up the tree:

```
                FertileTree grandkid1 = new FertileTree();
                grandkid1.addChild(new SterileTree(5));
                grandkid1.addChild(new SterileTree(5));

17              FertileTree kid3 = new FertileTree();
  1             kid3.addChild(grandkid1);
  2             kid3.addChild(new SterileTree(1));
  15            kid3.addChild(new SterileTree(1));
    10          kid3.addChild(new SterileTree(3));
      5
      5         FertileTree tree = new FertileTree();
    1           tree.addChild(new SterileTree(1));
    1           tree.addChild(new SterileTree(2));
    3           tree.addChild(kid3);
```

(a) Sizes (b) Code (c) Treemap

The upper right rectangle was subdivided evenly in two, reflecting the new children we added. These children are bigger than any existing sterile tree, so these are the biggest swaths of color in the treemap. We also switched subdivision orientation again since we descended a level in tree.

To show a treemap of a file hierarchy, we can treat plain files as sterile trees, whose sizes are their lengths in bytes. Directories are fertile trees, whose sizes are the sums of their children's sizes. One can inspect a treemap of a directory to see how disk space is being utilized:

Figure 4: Treemap of `/home/johnch/checkouts/universe`

Static treemaps of this complexity are more art than information. Many programs exist that let users interact with the treemap dynamically and see what the files are. We'll stick with art in this assignment.

2 Requirements

1. Write all code in package `hw1`.
2. Retain your `RandomPlus` and `Utilities` classes from part 1. Ensure that they pass the SpecChecker from part 1.
3. Write a class `Tree` that encapsulates the abstract notion of a thing that has a size and children that are also trees. It has the following specification:
   (a) It is `abstract`.

(b) It has a `public` method `getSize` that is `abstract` and returns a `long`. At the abstract level, you don't have enough information to implement this method. Subclasses will implement it to return the tree's size.

(c) It has a `public` method `plot` that is used to plot the tree into an image as a treemap. This method is `abstract` and accepts the following parameters: a `BufferedImage` to plot into, a `Rectangle` specifying the bounds of the image that this tree will be plotted into, and a `boolean` stating whether this tree should plot its children horizontally (if false, they are plotted vertically). At the abstract level, we don't have enough information to implement this method. How we plot this tree depends on whether it has children or not. Also, as this method takes a number of parameters that we don't expect the general public to pass sane values for, hide this method to all but its subclasses by making it `protected`.

(d) It has a `public` method `plot` that accepts a `BufferedImage` as its sole parameter. The general public calls this version of `plot`, which triggers the `protected` helper and passes in reasonable parameters. Use the entire image's bounds and start by plotting the children horizontally.

4. Write a class `SterileTree` that encapsulates a tree without children. You can think of it as a plain old file, though it can represent other things—like dead end roads, TV shows without spinoffs, quests that don't have subquests, and other base cases. It has the following specification:

   (a) It is a subclass of `Tree`.

   (b) It has a constructor taking a `long` size as its sole parameter.

   (c) It implements the `getSize` method to return its size as given at construction time.

   (d) It implements the many-parameter version of `plot` to plot a solid random color in the bounds (inclusive) of the image. (Who could provide a random color?)

5. Write a class `FertileTree` that encapsulates a tree with child trees. You can think of it as a directory, though it can represent other things—like roads that lead to other roads, TV shows with spinoffs, islands that have lakes, and other general cases. It has a size that is the sum of the sizes of its children. It has the following specification:

   (a) It is a subclass of `Tree`.

   (b) It has an `addChild` method that accepts a `Tree` parameter. It appends the parameter tree to its list of children.

   (c) It implements the `getSize` method to return its size as the sum of the sizes of its children.

   (d) It implements the many-parameter version of `plot` by dividing up the bounds it is given to its children. If a tree has two children, the first of size 6 and the second of size 3, and plotting is horizontal, then the left 2/3 of the bounds is plotted into by the first child and the right 1/3 is plotted into by the second child. If the plotting is vertical, then the first plots into the top 2/3 and the second plots into the bottom 1/3. Children should be plotted in the layout opposite to this one. That is, if this level was distributed horizontally, the next level should be distributed vertically, and vice versa. Your `proportionalize` and `distribute` methods simplify this task.

6. Write a class `Main` that has the following specification:

   (a) It may have a `main` method that you can use to test your code. This is not specified.

   (b) It has a `static` method `buildFileTree` that accepts a `File` parameter. If the file is not a directory, it returns a `SterileTree` whose size is the file's size. If it's a directory, it returns a `FertileTree` whose children are trees starting at each item contained in this directory. (To facilitate comparison, please sort the results of your `listFiles` call using `Arrays.sort`.) Think recursively. The reference solution is only 10 lines of code.

3 Submission

This homework is part of a regular assignment and is graded by hand and with help from the SpecChecker. Your assignment is expected to fully meet the requirements above and the following:

1. Variable names should be meaningful and accurate.
2. Non-obvious parts of your code should be commented.
3. Code should be cleanly formatted and indented.

Your work will also be inspected for plagiarism. Please do your own work. Talk about code with your classmates. Ask questions of your instructor or TA. Do not look at others' code. Do not ask questions specific to your homework anywhere online but Piazza. (If you find violators of this rule, please let me know.) Write your own code.

Put the SpecChecker in your Build Path. Run it as a Java Application (not a JUnit Test) and fix problems until all tests pass. Upload the resulting ZIP file to the W drive.

4 Files

- `http://www.twodee.org/teaching/cs245/2013C/homework/treemap/speccheck_hw1-2.jar`