Assignment Quadtrees

Object Orientation

Spring 2025

1 Quad Trees

A quad tree is a data structure that represents trees where each inner node has exactly four subtrees. Trees are *self-referential* data structures, which means that a node can have nodes as attributes.

2 Learning Goals

For this assignment, you will be asked to design and implement a quad tree data structure in Java. After completing this exercise you should be able to:

- Create recursive data structures with self-referential classes
- Implement recursive functions over such data structures
- Introduce an interface or abstract class as a base for the representation of different kinds of nodes
- Define each concrete node as an extension of the base class

3 Problem Sketch

Quad trees can be used to store two-dimensional images efficiently. In this assignment we consider images of $N \times N$ black and white pixels where N is a power of two. Images where each pixel can have two states, either white or colored, are called *monochrome*. Storing images in quad trees is based on the following idea. First, the $N \times N$ image is split into 4 quadrants of size $\frac{N}{2} \times \frac{N}{2}$. These sub-images are then themselves recursively packed into four smaller quad trees. This process continues until either an individual pixel is reached, or all pixels of a whole quadrant are of the same color.

Images where the size is not a power of two require more work, and are not covered in this assignment.

There are two kinds of leaf nodes. One for the white and one for the black pixels. The actual compression happens when all subtree nodes have the same color. In this case the entire subtree is represented as a single color pixel. The following example illustrates this idea.

In Figure 1 on the left we see a monochrome bitmap of size $N = 8$, and on the right its quad tree representation. The bitmap is first split up into four sub-images, each of size 4. These quadrants are represented in the tree, clockwise starting from the upper left quadrant. The upper left and lower left quadrants are completely black, which is why there are two black nodes on the second level in the quad tree. The upper right and

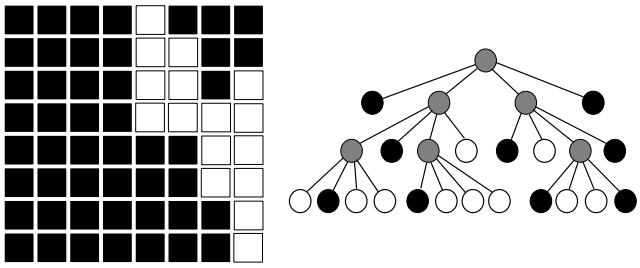

Figure 1: A monochrome bitmap and its quad tree representation.

lower right quadrants are further subdivided. This is represented by the grey nodes on the second level in the quad tree. See for yourself if you can understand how further subdivision and the quad tree correspond to each other.

Note that the same tree can represent images of different size. For example, a tree that consists of a single black node can represent an image of a single black pixel, or an image of four black pixels, and so on.

4 Implementation

Your program should have two representations of monochrome images: bitmaps and quad trees. You should write functions that can convert between them.

4.1 Bitmaps

A bitmap stores every pixel of an image. The class `Bitmap` represents images.

```
public class Bitmap {
    // each bit is stored in a two dimensional array named
       raster
    private final boolean[][] raster;
    private final int bmWidth, bmHeight;

    /**
     * Creates an empty bitmap of size width * height
     * @param width
     * @param height
     */
    public Bitmap( int width, int height ) {
        raster   = new boolean[width][height];
        bmWidth  = width;
        bmHeight = height;
    }
}
```

This class also contains methods to get and set bits, a method called `fillArea` that fills a given section with a given color and a method to convert the `Bitmap` to a `String`. These methods are not displayed in the example above.

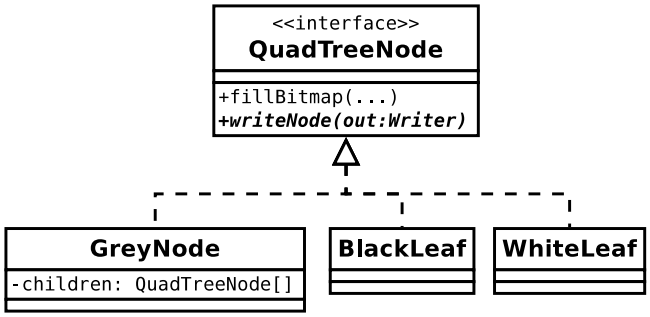

Figure 2: Hint for a class hierarchy for this assignment.

4.2 Quad Trees

We use the following interface to represent nodes of quad trees.

```
public interface QuadTreeNode {
    public void fillBitmap( int x, int y, int width, Bitmap
        bitmap );
    public void writeNode( Writer out );
}
```

For the concrete nodes, you should create three implementations of this interface. `GreyNode` for internal nodes, `BlackLeaf` for black leaves and `WhiteLeaf` for white leaves. An abstract method `fillBitmap` fills a given bitmap according to the tree structure. Figure 2 shows the class hierarchy with some of the methods.

There should also be a wrapper class called `QTree` to handle trees. This class keeps the internal representation of quad trees hidden.

```
public class QTree {
    private QuadTreeNode root;

    public QTree( Reader input ) {
        root = readQTree( input );
    }

    public QTree( Bitmap bitmap ) {
        root = bitmap2QTree(0, 0,  bitmap.getWidth(), bitmap);
    }

    public void writeQTree( Writer out ) {
        root.writeNode( out );
    }

    public void fillBitmap ( Bitmap bitmap ) {
        root.fillBitmap(0, 0, bitmap.getWidth(), bitmap);
    }

}
```

`QTree` has two constructors. The first one builds a tree from a serialization provided by a `Reader`. The second constructor builds a tree from a bitmap. The method `writeQTree` serializes the tree to a `Writer`. The method `fillBitmap` fills a given bitmap according to the tree structure. `Reader` and `Writer` are predefined Java classes used to read and write character streams. Look into the standard Java API for further information.

4.3 Serialization

The method `writeNode` writes the node structure as a sequence of bits. This is done by *depth-first pre-order traversal*. Each internal node is rendered as a 1, followed by the data of the four subtrees. Each leaf is rendered as a 0 followed by a bit that indicates the color: 0 for black and 1 for white.

The tree in Figure 1 is represented by the following series of bits.

100110100010100100010101011000110001010000000

Reading a tree follows the same pattern. A 1 is followed by the content of the four subtrees. A 0 stands for a leaf, followed by a bit that indicates the color.

Hints

- Writing a quad tree happens in `writeNode` of the node classes.
- A grey node has exactly four subtrees. A grey node's `writeNode` should output a 1 and then call `writeNode` of the subtrees. Nothing more.
- A white node's `writeNode` should just output 01.
- Reading a quad tree is entirely done in `QTree.readQTree`. When `readQTree` sees a 1 it calls itself recursively four times and constructs a grey node from the results. When it sees a 0 it reads the next bit and constructs a white or black node accordingly.

5 Your Tasks

Altogether there are three representations of images.

1. Bitmaps
2. Quad trees
3. Series of ones and zeros

You should implement four conversion functions.

- Bitmap to quad tree
- Quad tree to bitmap
- Quad tree to serialization
- Serialization to quad tree

Bitmap to quad tree and serialization to quad tree should happen in recursive functions in QTree. Quad tree to bitmap and quad tree to serialization should happen in `fillBitmap` and `writeNode` in the node classes.

6 Supplementary Files

On Brightspace you can find a project template to start this assignment.

6.1 Submit Your Project

To submit your project, follow these steps.

1. Find the folder that contains your assignment. In Visual Studio Code, you can find this by going to: File → Open Folder. Add the correct folder to a zip file. At the root level, your zip file should contain only one folder, e.g. `assignment-student-start`, which contain the entire project, i.e. the `app` folder and the Gradle files. *Do not submit only the `.java` files or the `src` folder!*
2. **Submit this zip file on Brightspace**. Do not submit individual Java files. Only one person in your group has to submit it. Submit your project before the deadline, which can be found on Brightspace.
3. **Do not submit any other format.** Do not submit .rar or .tar.gz or .7z files. Only zip files are allowed.