Ex11

s1260119 Yuta Tomiyama

Task 11.1

```
import java.util.concurrent.atomic.AtomicInteger;

class ex03_money {
    static AtomicInteger Account = new AtomicInteger(0);

    static class Spouse implements Runnable {
        private int Sum;

        public Spouse(int sum) {
            Sum = sum;
        }

        public void run() {
            for (int i = 0; i < Sum; i++) {

                Account.incrementAndGet();

            }

        }
    }

    static public void main(String args[]) {
        Spouse husband = new Spouse(500000);
        Spouse wife = new Spouse(500000);

        Thread t1 = new Thread(husband);
        Thread t2 = new Thread(wife);
        t1.start();
        t2.start();
        try {
            t1.join();
            t2.join();
        } catch (Exception e) {
            e.printStackTrace();
        }

        System.out.println(Account);
    }
}
```

Task 11.2

```
import java.util.ArrayList;
import java.util.Vector;
import java.util.stream.IntStream;

public class CompareCollection {
    public static void main(String[] args) {
        Vector<Integer> vector = new Vector<>();
        ArrayList<Integer> list = new ArrayList<>();
        IntStream.range(0, 100000).forEach(i -> {
            vector.add(0);
            list.add(0);
        });
        long time1 = System.nanoTime();
        for (int i = 0; i < list.size(); i++) {
            vector.set(i, i);
        }
        long time2 = System.nanoTime();
        for (int i = 0; i < vector.size(); i++) {
            list.set(i, i);
        }
        long time3 = System.nanoTime();
        System.out.println("Vector: " + (time2 - time1) + "ns");
        System.out.println("ArrayList: " + (time3 - time2) + "ns");
    }
}
```

Vector: 7722231ns
ArrayList: 2530067ns

ArrayList is 3x faster than Vector.

Task 11.3

```
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.awt.image.WritableRaster;
import java.io.File;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

class ex11_grayscale {
    public static void main(String[] args) throws Exception {
        BufferedImage img = ImageIO.read(new File(args[0]));
```

```
        BufferedImage new_img = new BufferedImage(img.getWidth(),
img.getHeight(), img.getType());

        WritableRaster raster = img.getRaster();
        WritableRaster newRaster = new_img.getRaster();

        int width = img.getWidth();
        int height = img.getHeight();

        int processors = Runtime.getRuntime().availableProcessors();
        List<GrayScaler> tasks = new ArrayList<>();
        for (int i = 0; i < processors - 1; i++) {
            GrayScaler task = new GrayScaler(
                    raster,
                    newRaster,
                    width,
                    height,
                    i * (width / processors),
                    (i + 1) * (width / processors)
            );
            tasks.add(task);
        }
        GrayScaler task = new GrayScaler(
                raster,
                newRaster,
                width,
                height,
                (processors - 1) * (width / processors),
                width
        );
        tasks.add(task);

        ExecutorService executor =
Executors.newFixedThreadPool(processors);
        executor.invokeAll(tasks);
        executor.shutdown();

        ImageIO.write(new_img, "png", new File(args[1]));
    }
}

class GrayScaler implements Callable<Void> {

    private final WritableRaster raster;
    private final WritableRaster newRaster;
    private final int height;
```

```
    private final int startX;
    private final int endX;

    public GrayScaler(
            WritableRaster raster,
            WritableRaster newRaster,
            int width,
            int height,
            int startX,
            int endX
    ) {
        this.raster = raster;
        this.newRaster = newRaster;
        this.height = height;
        this.startX = startX;
        this.endX = endX;
    }

    @Override
    public Void call() throws Exception {
        for (int x = startX; x < endX; x++) {
            for (int y = 0; y < height; y++) {
                double R = raster.getSample(x, y, 0);
                double G = raster.getSample(x, y, 1);
                double B = raster.getSample(x, y, 2);
                double level = 0.3 * R + 0.59 * G + 0.11 * B;

                newRaster.setSample(x, y, 0, level);
                newRaster.setSample(x, y, 1, level);
                newRaster.setSample(x, y, 2, level);
            }
        }
        return null;
    }
}
```

Task 11.4

```
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.awt.image.WritableRaster;
import java.io.File;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveAction;
```

```
public class FadeToGray extends RecursiveAction {

    private static final int MAX_PIXEL = 100;

    private final WritableRaster raster;
    private final WritableRaster newRaster;
    private final int startX;
    private final int endX;
    private final int startY;
    private final int endY;

    public FadeToGray(WritableRaster raster, WritableRaster newRaster,
int startX, int endX, int startY, int endY) {
        this.raster = raster;
        this.newRaster = newRaster;
        this.startX = startX;
        this.endX = endX;
        this.startY = startY;
        this.endY = endY;
    }

    @Override
    protected void compute() {
        if (endX - startX < MAX_PIXEL && endY - startY < MAX_PIXEL) {
            for (int x = startX; x < endX; x++) {
                for (int y = startY; y < endY; y++) {
                    double R = raster.getSample(x, y, 0);
                    double G = raster.getSample(x, y, 1);
                    double B = raster.getSample(x, y, 2);
                    double level = 0.3 * R + 0.59 * G + 0.11 * B;

                    newRaster.setSample(x, y, 0, level);
                    newRaster.setSample(x, y, 1, level);
                    newRaster.setSample(x, y, 2, level);
                }
            }
        }
        List<FadeToGray> tasks = new ArrayList<>();
        if (endX - startX >= MAX_PIXEL) {
            FadeToGray task1 = new FadeToGray(
                    raster,
                    newRaster,
                    startX,
                    (startX + endX) / 2,
                    startY,
```

```
                endY
            );
            FadeToGray task2 = new FadeToGray(
                    raster,
                    newRaster,
                    (startX + endX) / 2,
                    endX,
                    startY,
                    endY
            );
            tasks.add(task1);
            tasks.add(task2);
        }
        if (endY - startY >= MAX_PIXEL) {
            FadeToGray task1 = new FadeToGray(
                    raster,
                    newRaster,
                    startX,
                    endX,
                    startY,
                    (startY + endY) / 2
            );
            FadeToGray task2 = new FadeToGray(
                    raster,
                    newRaster,
                    startX,
                    endX,
                    (startY + endY) / 2,
                    endY
            );
            tasks.add(task1);
            tasks.add(task2);
        }
        invokeAll(tasks);
    }

    public static void main(String[] args) throws Exception {
        BufferedImage img = ImageIO.read(new File(args[0]));
        BufferedImage new_img = new BufferedImage(img.getWidth(),
img.getHeight(), img.getType());

        WritableRaster raster = img.getRaster();
        WritableRaster newRaster = new_img.getRaster();

        int width = img.getWidth();
        int height = img.getHeight();
```

```
        FadeToGray task = new FadeToGray(
                raster,
                newRaster,
                0,
                width,
                0,
                height
        );
        ForkJoinPool pool = new ForkJoinPool();
        pool.invoke(task);

        ImageIO.write(new_img, "png", new File(args[1]));
    }
}
```