Ex12

s1260119 Yuta Tomiyama

Task 12.1
non-parallel: 78ms
parallel: 77ms

```
import java.util.Arrays;
import java.util.Random;

public class QuickSort {

    public static void main(String[] args) {
        Random random = new Random();
        int[] array = new int[1000000];
        for (int i = 0; i < array.length; i++) {
            array[i] = random.nextInt(100000);
        }
        long start = System.currentTimeMillis();
        quickSort(array, 0, array.length - 1);
        long end = System.currentTimeMillis();
        System.out.println((end - start) + "ms");
    }

    static void quickSort(int[] array, int left, int right) {
        if (left < right) {
            int p = partition(array, left, right);
            if (p - left >= array.length / 30 && right - p >=
array.length) {
                // omp sections
                {
                    // omp section
                    {
                        quickSort(array, left, p - 1);
                    }
                    // omp section
                    {
                        quickSort(array, p + 1, right);
                    }
                }
            } else {
                quickSort(array, left, p - 1);
                quickSort(array, p + 1, right);
            }
        }
    }
```

```
    static int partition(int[] array, int left, int right) {
        int pivot = array[right];
        int i = left;
        for (int j = left; j < right; j++) {
            if (array[j] < pivot) {
                int tmp = array[i];
                array[i] = array[j];
                array[j] = tmp;
                i++;
            }
        }
        int tmp = array[i];
        array[i] = array[right];
        array[right] = tmp;
        return i;
    }
}
```

Task 12.2

```
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import java.nio.file.*;
import java.util.*;

class ex02_fileCrypto {
    static public void Encrypt(String inFileName, String outFileName,
String password) {
        try {
            byte[] file = Files.readAllBytes(Paths.get(inFileName));
            byte[] checkedFile = Arrays.copyOf(checker.getBytes(),
checker.length() + file.length);

            System.arraycopy(file, 0, checkedFile, checker.length(),
file.length);
            MessageDigest digest = MessageDigest.getInstance("SHA");
            digest.update(password.getBytes());
            SecretKeySpec key = new SecretKeySpec(digest.digest(), 0,
16, "AES");
            Cipher aes = Cipher.getInstance("AES/ECB/PKCS5Padding");
            aes.init(Cipher.ENCRYPT_MODE, key);
            Files.write(Paths.get(outFileName),
aes.doFinal(checkedFile));
        } catch (Exception e) {
            e.printStackTrace();
```

```
        }
    }

    static public boolean Decrypt(String inFileName, String outFileName,
String password) {
        try {
            byte[] checkedFile =
Files.readAllBytes(Paths.get(inFileName));

            MessageDigest digest = MessageDigest.getInstance("SHA");
            digest.update(password.getBytes());
            SecretKeySpec key = new SecretKeySpec(digest.digest(), 0,
16, "AES");
            Cipher aes = Cipher.getInstance("AES/ECB/PKCS5Padding");
            aes.init(Cipher.DECRYPT_MODE, key);
            String cleartext = new String(aes.doFinal(checkedFile));

            if (!cleartext.substring(0,
checker.length()).equals(checker))
                throw new Exception();

            Files.write(Paths.get(outFileName),
cleartext.substring(checker.length()).getBytes());

        } catch (Exception e) {
            return false;
        }

        return true;
    }

    static public void main(String[] args) {
        String inFileName = "ex02_mobydick.enc";
        String outFileName = "ex02_mobydick";
        List<Character> candidateChars = new ArrayList<>();
        for (char c = 'a'; c <= 'z'; c++) {
            candidateChars.add(c);
        }
        for (int i = 0; i <= 9; i++) {
            candidateChars.add(Character.forDigit(i, 10));
        }
        List<String> candidatePasswords = new ArrayList<>();
        // omp parallel
        for (char c1 : candidateChars) {
            for (char c2 : candidateChars) {
                for (char c3 : candidateChars) {
```

```
                    for (char c4 : candidateChars) {
                        candidatePasswords.add("" + c1 + c2 + c3 + c4);
                    }
                }
            }
        }
        for (int i = 0; i < 4; i++) {
            final int start = candidatePasswords.size() / 4 * i;
            final int end = Math.min(candidatePasswords.size() / 4 * (i
+ 1), candidatePasswords.size());
            new Thread(() -> {
                for (int j = start; j < end; j++) {
                    String password = candidatePasswords.get(j);
                    boolean valid = Decrypt(inFileName, outFileName,
password);
                    if (valid) {
                        System.out.println(password);
                    }
                }
            }).start();
        }
    }

    private static final String checker = "correct header";
}
```

Task 12.3

```
import java.math.BigInteger;

public class FindPrime {
    public static void main(String[] args) {
        int start = Integer.parseInt(args[0]);
        int end = Integer.parseInt(args[1]);
        // omp parallel for
        for (int i = start; i < end; i++) {
            if (BigInteger.valueOf(i).isProbablePrime(10)) {
                System.out.println(i);
            }
        }
    }
}
```

Task 12.4

```
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) {
            // omp parallel for
            for (int x = startX; x < endX; x++) {
                // omp parallel for
                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]));
    }
}
```