Ex3

s1260119 Yuta Tomiyama

Task 3.1

```
import java.util.*;
import java.util.concurrent.CountDownLatch;

class ex03_sort2Proc {
    static final int N = 100000;
    static int A[] = new int[N];
    static CountDownLatch C = new CountDownLatch(2);

    static class SortThread implements Runnable {
        int Left, Right;

        public SortThread(int left, int right) {
            Left = left;
            Right = right;
        }

        public void run() {
            Arrays.sort(A, Left, Right);
            C.countDown();
        }
    }

    static public void main(String args[]) {
        Random rand = new Random();
        for (int i = 0; i < N; ++i)
            A[i] = rand.nextInt(1000);

        new Thread(new SortThread(0, N / 2)).start();
        new Thread(new SortThread(N / 2, N)).start();

        try {
            C.await();
        } catch (Exception e) {
        }

        // here add merging code
        int l = 0;
        int r = N / 2;
        int a = 0;
        int[] sorted = new int[N];
        while (a < N) {
            if (l >= N / 2) {
```

```
                while (r < N) {
                    sorted[a] = A[r];
                    a++;
                    r++;
                }
                break;
            } else if (r >= N) {
                while (l < N / 2) {
                    sorted[a] = A[l];
                    a++;
                    l++;
                }
                break;
            }
            int leftHead = A[l];
            int rightHead = A[r];
            if (leftHead < rightHead) {
                sorted[a] = leftHead;
                l++;
            } else {
                sorted[a] = rightHead;
                r++;
            }
            a++;
        }
        for (int value : sorted) {
            System.out.print(value + " ");
        }
    }
}
```

Task 3.2

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.concurrent.CountDownLatch;

public class FindMaximum {

    static class FindThread implements Runnable {
        public int result;
        private final int[] array;
        private final int startInclusive;
        private final int endExclusive;
        private final CountDownLatch latch;
```

```
        public FindThread(int[] array, int startInclusive, int
endExclusive, CountDownLatch latch) {
            this.array = array;
            this.startInclusive = startInclusive;
            this.endExclusive = endExclusive;
            this.latch = latch;
        }

        @Override
        public void run() {
            int max = array[startInclusive];
            for (int i = startInclusive; i < endExclusive; i++) {
                if (array[i] > max) {
                    max = array[i];
                }
            }
            result = max;
            latch.countDown();
        }
    }

    public static void main(String[] args) throws Exception {
        int n = 100;
        int[] array = new int[n];
        CountDownLatch latch = new CountDownLatch(4);
        Random random = new Random();
        for (int i = 0; i < n; i++) {
            array[i] = random.nextInt(400);
        }
        List<FindThread> threads = new ArrayList<>();
        for (int i = 0; i < 4; i++) {
            threads.add(new FindThread(array, i * n / 4, i * n / 4 + 1,
latch));
        }
        threads.forEach(t -> new Thread(t).start());
        latch.await();
        int max = max(
                threads.get(0).result,
                threads.get(1).result,
                threads.get(2).result,
                threads.get(3).result
        );
        System.out.println(max);
    }
```

```
    private static int max(int... args) {
        int max = args[0];
        for (int a : args) {
            if (a > max) {
                max = a;
            }
        }
        return max;
    }
}
```

Task 3.3

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.concurrent.Semaphore;
import java.util.stream.Collectors;

public class HeadsOrTails {

    static class Process implements Runnable {

        private final Random random = new Random();
        public final int id;
        private final Semaphore s;
        boolean result; // ture is heads, false is tails

        public Process(int id, Semaphore s) {
            this.id = id;
            this.s = s;
        }

        @Override
        public void run() {
            result = random.nextBoolean();
            s.release();
        }
    }

    public static void main(String[] args) {
        Semaphore s = new Semaphore(0);
        int p = 4;
        List<Process> processes = new ArrayList<>();
        for (int i = 1; i <= p; i++) {
            processes.add(new Process(i, s));
```

```
        }
        int round = 1;
        while (true) {
            processes.forEach(t -> new Thread(t).start());
            processes.forEach((t) -> {
                try {
                    s.acquire();
                } catch (Exception e) {
                }
            });
            processes.removeIf(t -> !t.result);
            long winners = processes.stream().filter(t ->
t.result).count();
            System.out.println("round " + round + " winners");
            if (winners >= 1) {
                System.out.println(
                        processes.stream().map(t ->
String.valueOf(t.id)).collect(Collectors.joining(" "))
                );
            } else {
                System.out.println("none");
            }
            if (winners <= 1) {
                break;
            }
            round++;
        }
        String winner = processes.stream().findFirst().map(w ->
String.valueOf(w.id)).orElse("none");
        System.out.println("winner: " + winner);
    }
}
```

Task 3.4

```
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.Semaphore;

class ex03_money {
    static AccountType Account = new AccountType();
    static Semaphore M = new Semaphore(1);
    static CountDownLatch C = new CountDownLatch(2);

    static class AccountType {

        int account = 0;
```

```
    public int getValue() {
        return account;
    }

    synchronized void addOneUnit() {
        account++;
    }
}

static class Spouse implements Runnable {
    private int Sum;

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

    public void run() {
        for (int i = 0; i < Sum; i++) {
            try {
                M.acquire();
            } catch (Exception e) {
            }

            Account.addOneUnit();

            M.release();
        }

        C.countDown();
    }
}

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

    new Thread(husband).start();
    new Thread(wife).start();

    try {
        C.await();
    } catch (Exception e) {
    }

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

```
    }
}
```