

```

1 package tsa_sim;
2
3 import tsa_sim.person.*;
4 import tsa_sim.Checker.*;
5 import java.io.FileNotFoundException;
6 import java.text.DateFormat;
7 import java.text.SimpleDateFormat;
8 import java.util.Date;
9 import java.util.Random;
10 import java.util.TreeSet;
11 import java.util.concurrent.TimeUnit;
12 import java.util.logging.Level;
13 import java.util.logging.Logger;
14
15 //This could implement Runnable, but not necessary.
16 public class TSASimulator {
17     private static final int PASSENGER_COUNT = 50;
18     private static final int MAX_INITIAL_TIME = 100;
19     private static final int BASE_TICK_VALUE = 1000;
20     private final static Logger LOGGER = Logger.getLogger(TSASimulator.class.getName(
21 ));
22     private PersonBuilder personBuilder;
23     //Length of a tick in milliseconds.
24     private int tickValue;
25     private Checker checkerA;
26     private Checker checkerB;
27     private Checker checkerC;
28     private OrderedChecker initialChecker;
29     private PersonQueue passengerPool;
30     private PersonQueue completedPool;
31     private PersonQueue queueA;
32     private PersonQueue queueB;
33     private PersonQueue queueC;
34
35     public TSASimulator(int passengerCount, int initialTime, int tickValue) throws
36     FileNotFoundException {
37         this.tickValue = tickValue;
38         personBuilder = new PersonBuilder();
39         queueA = new PersonQueue();
40         queueB = new PersonQueue();
41         queueC = new PersonQueue();
42         completedPool = new PersonQueue();
43         passengerPool = new PersonQueue();
44         initialChecker = new OrderedChecker(
45             passengerPool,
46             new PersonQueue[] {queueA, queueB},
47             generateTimes(passengerCount, initialTime),
48             "Checker I"
49 );
50     checkerA = new Checker(queueA, new PersonQueue[] {queueC}, this.tickValue, "
51 Checker A");
52     checkerB = new Checker(queueB, new PersonQueue[] {queueC}, this.tickValue, "
53 Checker B");
54     checkerC = new Checker(queueC, new PersonQueue[] {completedPool}, this.
55 tickValue, "Checker C");
56
57     for(int i = 0; i < passengerCount; i++) {
58         passengerPool.add(personBuilder.buildPerson(i+1));
59     }
60
61     private void printPassengers() {
62         for(Person person : this.passengerPool) {
63             LOGGER.log(Level.INFO, String.format(
64                 "Id: %d, Name: %s, createdAt: %s%n",
65                 person.getId(),
66                 person.getFullName(),
67                 person.getCreatedAt().toString()));
68         }
69     }
70
71     private void run() {
72         Thread a = new Thread(checkerA);
73         Thread b = new Thread(checkerB);
74         Thread c = new Thread(checkerC);

```

```

72     Thread d = new Thread(initialChecker);
73
74     DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss.SSS");
75     Date start = new Date();
76     d.start();
77     a.start();
78     b.start();
79     c.start();
80
81     //Wait for initial checker to end
82     while (d.isAlive()) {
83         try {
84             d.join();
85         } catch (InterruptedException e) {
86             e.printStackTrace();
87             //try and recover
88             System.out.println("Restarting: " + d.getName());
89             d.start();
90         }
91     }
92     while (a.isAlive() || b.isAlive()) {
93         if(a.isAlive() && queueA.isEmpty()) {
94             a.interrupt();
95         }
96         if(b.isAlive() && queueB.isEmpty()) {
97             b.interrupt();
98         }
99     }
100    while (c.isAlive()) {
101        if (queueC.isEmpty()) {
102            c.interrupt();
103        }
104    }
105
106    Date end = new Date();
107    System.out.println("\n*** TSA SIMULATOR ***");
108    System.out.println("Execution completed!");
109    System.out.println("\nSTART: " + dateFormat.format(start));
110    System.out.println("END: " + dateFormat.format(end));
111    //TODO: Add time elapsed in ticks
112    System.out.print("TIME ELAPSED: ");
113    System.out.print(getDateDiff(start, end, TimeUnit.SECONDS) + " seconds");
114 }
115
116 private TreeSet<Date> generateTimes(int count, int length) {
117     //TODO: throw error if there are not count milliseconds in range.
118     TreeSet<Date> times = new TreeSet<>();
119     long floor = System.currentTimeMillis();
120     while(times.size() < count) {
121         times.add(generateTime(new Date(floor), new Date(floor + tickValue *
length)));
122     }
123
124     return times;
125 }
126
127 private static long getDateDiff(Date date1, Date date2, TimeUnit timeUnit) {
128     long diffInMillis = date2.getTime() - date1.getTime();
129     return timeUnit.convert(diffInMillis, TimeUnit.MILLISECONDS);
130 }
131
132 private static Date generateTime(Date floor, Date ceiling) {
133     Random generator = new Random();
134     return new Date(generator.nextInt((int)(ceiling.getTime() - floor.getTime())
) + floor.getTime());
135 }
136
137 public static void main(String[] args) {
138     int tickValue = BASE_TICK_VALUE;
139     int maxInitTime = MAX_INITIAL_TIME;
140     int passCount = PASSENGER_COUNT;
141     if (args.length >= 3) {
142         try {
143             tickValue = Integer.parseInt(args[2]);
144         } catch (NumberFormatException e) {
145             LOGGER.log(Level.WARNING, "Cannot parse tick value, using default");

```

```
146         tickValue = BASE_TICK_VALUE;
147     }
148     try {
149         maxInitTime = Integer.parseInt(args[1]);
150     } catch (NumberFormatException e) {
151         LOGGER.log(Level.WARNING, "Cannot parse initial time value, using
152         default");
153         maxInitTime = MAX_INITIAL_TIME;
154     }
155     try {
156         passCount = Integer.parseInt(args[0]);
157     } catch (NumberFormatException e) {
158         LOGGER.log(Level.WARNING, "Cannot parse passenger count, using
159         default");
160         passCount = PASSENGER_COUNT;
161     }
162     try {
163         TSASimulator simulator = new TSASimulator(passCount, maxInitTime,
164         tickValue);
165         simulator.run();
166     } catch (FileNotFoundException e) {
167         LOGGER.log(Level.SEVERE, "Missing name file: ", e);
168     }
169 }
170 }
```

```

1 package tsa_sim.person;
2
3 import java.util.Date;
4
5 public class Person {
6     private final int id;
7     private final String firstName;
8     private final String lastName;
9     private final Date createdAt;
10    //TODO: These should not be in this class for portability, an event listener
    should really record this.
11    //Extend person.
12    //First Queue entered (A or B)
13    private Date queuedAt = null;
14    //Second Queue entered
15    private Date finalQueuedAt = null;
16    //Queueing complete
17    private Date completedAt = null;
18
19    public Person(int id, Date createdAt, String firstName, String lastName) {
20        this.createdAt = createdAt;
21        this.id = id;
22        this.firstName = firstName;
23        this.lastName = lastName;
24    }
25
26    /*
27     * GETTERS
28     */
29    public Date getCreatedAt() {
30        return createdAt;
31    }
32
33    public int getId() {
34        return id;
35    }
36
37    public String getFirstName() {
38        return firstName;
39    }
40
41    public String getLastName() {
42        return lastName;
43    }
44
45    public String getFullName() {
46        return firstName + ' ' + lastName;
47    }
48
49    public Date getCompletedAt() {
50        return completedAt;
51    }
52
53    public Date getQueuedAt() {
54        return queuedAt;
55    }
56
57    public Date getFinalQueuedAt() {
58        return finalQueuedAt;
59    }
60
61    /*
62     * SETTERS
63     */
64    public void setCompletedAt(Date completedAt) {
65        this.completedAt = completedAt;
66    }
67
68    public void setQueuedAt(Date queuedAt) {
69        this.queuedAt = queuedAt;
70    }
71
72    public void setFinalQueuedAt(Date finalQueuedAt) {
73        this.finalQueuedAt = finalQueuedAt;
74    }
75 }

```

```
1 package tsa_sim.person;  
2  
3 import java.util.concurrent.LinkedBlockingQueue;  
4  
5 public class PersonQueue extends LinkedBlockingQueue<Person> {  
6  
7     public PersonQueue() {  
8         super();  
9     }  
10 }
```

```
1 package tsa_sim.person;
2
3 import java.io.File;
4 import java.io.FileNotFoundException;
5 import java.util.ArrayList;
6 import java.util.Date;
7 import java.util.Random;
8 import java.util.Scanner;
9
10 public class PersonBuilder {
11
12     private ArrayList<String> firstNames;
13     private ArrayList<String> lastNames;
14
15     public PersonBuilder() throws FileNotFoundException {
16         this.seedLists();
17     }
18
19     private void seedLists() throws FileNotFoundException {
20         //build name databases
21         firstNames = new ArrayList<>();
22         lastNames = new ArrayList<>();
23
24         File firstNameFile = new File("resources/first_names.txt");
25         File lastNameFile = new File("resources/last_names.txt");
26
27         Scanner input = new Scanner(firstNameFile);
28         while(input.hasNextLine()) {
29             firstNames.add(input.nextLine());
30         }
31         input = new Scanner(lastNameFile);
32         while(input.hasNextLine()) {
33             lastNames.add(input.nextLine());
34         }
35     }
36
37     public Person buildPerson(int id) {
38         Random generator = new Random();
39         return new Person(
40             id,
41             new Date(),
42             firstNames.get(generator.nextInt(firstNames.size())),
43             lastNames.get(generator.nextInt(lastNames.size()))
44         );
45     }
46 }
47 }
48
```

```

1 package tsa_sim.Checker;
2
3 import tsa_sim.person.*;
4
5 import java.text.DateFormat;
6 import java.text.SimpleDateFormat;
7 import java.util.concurrent.ThreadLocalRandom;
8 import java.util.logging.FileHandler;
9 import java.util.logging.Level;
10 import java.util.logging.Logger;
11
12 import static java.lang.Math.max;
13
14 public class Checker implements CheckerInterface {
15     private final static Logger LOGGER = Logger.getLogger(Checker.class.getName());
16     private final PersonQueue queue;
17     private final PersonQueue[] destination;
18     //How many ticks should the queue be expedited by.
19     private int timeModifier;
20     private int previousLength;
21     private int tick;
22     private String name;
23     private DateFormat dateFormat;
24     private FileHandler fh;
25
26     public Checker(PersonQueue origin, PersonQueue[] destination, int tick, String
name) {
27         this.queue = origin;
28         this.destination = destination;
29         this.name = name;
30         timeModifier = 0;
31         previousLength = 0;
32         this.tick = tick;
33         dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");
34     }
35
36     public void run() {
37         Thread.currentThread().setName(name);
38         while (true) {
39             try {
40                 if (queue.isEmpty()) {
41                     Thread.sleep(tick);
42                 } else {
43                     Thread.sleep(tick * max((ThreadLocalRandom.current().nextInt(15)
+ 1 - timeModifier), 1));
44                     process(queue.take());
45                 }
46
47                 if (previousLength < queue.size()) {
48                     timeModifier++;
49                 } else if (previousLength > queue.size()) {
50                     if (timeModifier > 0) {
51                         timeModifier--;
52                     }
53                 }
54                 previousLength = queue.size();
55             } catch (InterruptedException e) {
56                 LOGGER.log(Level.INFO, String.format("%s: Ending by interrupt",
Thread.currentThread().getName()));
57                 return;
58             }
59         }
60     }
61
62     public void process(Person person) {
63         //Set the earliest null timestamp
64         CheckerInterface.stamp(person);
65         //TODO: align these logs, format the name or something
66         LOGGER.log(Level.INFO, String.format(
67             "%s processed: Id: %7d, Name: %25s, createdAt: %s, queuedAt: %s,
finalQueuedAt: %s, completedAt: %s",
68             Thread.currentThread().getName(),
69             person.getId(),
70             person.getFullName(),
71             dateFormat.format(person.getCreatedAt()),
72             person.getQueuedAt() == null ? null : dateFormat.format(person.

```

```
72 getQueuedAt()),
73     person.getFinalQueuedAt() == null ? null : dateFormat.format(person.
    getFinalQueuedAt()),
74     person.getCompletedAt() == null ? null : dateFormat.format(person.
    getCompletedAt())
75 );
76     if (destination.length > 1) {
77         destination[ThreadLocalRandom.current().nextInt(destination.length)].add
    (person);
78     } else {
79         destination[0].add(person);
80     }
81 }
82 }
83
```



```

1 package tsa_sim.Checker;
2
3 import tsa_sim.person.Person;
4 import tsa_sim.person.PersonQueue;
5
6 import java.text.DateFormat;
7 import java.text.SimpleDateFormat;
8 import java.util.Date;
9 import java.util.TreeSet;
10 import java.util.concurrent.ThreadLocalRandom;
11 import java.util.logging.Level;
12 import java.util.logging.Logger;
13
14 public class OrderedChecker implements CheckerInterface {
15     private final static Logger LOGGER = Logger.getLogger(OrderedChecker.class.
16     getName());
17     private final PersonQueue queue;
18     private final PersonQueue[] destination;
19     private TreeSet<Date> popTimes;
20     private String name;
21     private DateFormat dateFormat;
22
23     public OrderedChecker(PersonQueue origin, PersonQueue[] destination, TreeSet<Date>
24     > popTimes, String name) {
25         this.queue = origin;
26         this.destination = destination;
27         this.name = name;
28         this.popTimes = popTimes;
29         dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");
30     }
31     /*
32     * If times are in the past objects will pop as soon as execution starts
33     */
34     public void setProcessTimes(TreeSet<Date> times) {
35         this.popTimes = times;
36     }
37
38     public void run() {
39         Thread.currentThread().setName(name);
40         while (!popTimes.isEmpty()) {
41             if( System.currentTimeMillis() >= popTimes.first().getTime()) {
42                 popTimes.remove(popTimes.first());
43                 try {
44                     process(queue.take());
45                 } catch (InterruptedException e) {
46                     LOGGER.log(Level.INFO, String.format("%s: Ending by interrupt",
47                     Thread.currentThread().getName()));
48                     return;
49                 }
50             }
51         }
52
53         LOGGER.log(Level.INFO, String.format("%s: No more times, ending execution.",
54         Thread.currentThread().getName()));
55     }
56
57     public void process(Person person) {
58         //Set the earliest null timestamp
59         CheckerInterface.stamp(person);
60         //TODO: align these logs, format the name or something
61         LOGGER.log(Level.INFO, String.format(
62         "%s processed: Id: %7d, Name: %25s, createdAt: %s, queuedAt: %s",
63         Thread.currentThread().getName(),
64         person.getId(),
65         person.getFullName(),
66         dateFormat.format(person.getCreatedAt()),
67         person.getQueuedAt() == null ? null : dateFormat.format(person.
68         getQueuedAt())
69         ));
70         if (destination.length > 1) {
71             destination[ThreadLocalRandom.current().nextInt(destination.length)].add(
72             person);
73         } else {
74             destination[0].add(person);
75         }
76     }
77 }

```

```
71     }  
72 }  
73
```

```
1 package tsa_sim.Checker;
2
3 import tsa_sim.person.Person;
4
5 import java.util.Date;
6
7 public interface CheckerInterface extends Runnable {
8
9     void run();
10
11     void process(Person person);
12
13     //TODO: Perform this in observer
14     static void stamp(Person person) {
15         if (person.getQueuedAt() == null) {
16             person.setQueuedAt(new Date());
17         } else if (person.getFinalQueuedAt() == null) {
18             person.setFinalQueuedAt(new Date());
19         } else {
20             person.setCompletedAt(new Date());
21         }
22     }
23 }
24
```