```java
package tsa_sim;

import tsa_sim.person.*;
import tsa_sim.Checker.*;
import java.io.FileNotFoundException;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Random;
import java.util.TreeSet;
import java.util.concurrent.TimeUnit;
import java.util.logging.Level;
import java.util.logging.Logger;

//This could implement Runnable, but not necessary.
public class TSASimulator {
    private static final int PASSENGER_COUNT = 50;
    private static final int MAX_INITIAL_TIME = 100;
    private static final int BASE_TICK_VALUE = 1000;
    private final static Logger LOGGER = Logger.getLogger(TSASimulator.class.getName());
    private PersonBuilder personBuilder;
    //Length of a tick in milliseconds.
    private int tickValue;
    private Checker checkerA;
    private Checker checkerB;
    private Checker checkerC;
    private OrderedChecker initialChecker;
    private PersonQueue passengerPool;
    private PersonQueue completedPool;
    private PersonQueue queueA;
    private PersonQueue queueB;
    private PersonQueue queueC;

    public TSASimulator(int passengerCount, int initialTime, int tickValue) throws FileNotFoundException {
        this.tickValue = tickValue;
        personBuilder = new PersonBuilder();
        queueA = new PersonQueue();
        queueB = new PersonQueue();
        queueC = new PersonQueue();
        completedPool = new PersonQueue();
        passengerPool = new PersonQueue();
        initialChecker = new OrderedChecker(
                passengerPool,
                new PersonQueue[] {queueA, queueB},
                generateTimes(passengerCount, initialTime),
                "Checker I"
        );
        checkerA = new Checker(queueA, new PersonQueue[] {queueC}, this.tickValue, "Checker A");
        checkerB = new Checker(queueB, new PersonQueue[] {queueC}, this.tickValue, "Checker B");
        checkerC = new Checker(queueC, new PersonQueue[] {completedPool}, this.tickValue, "Checker C");

        for(int i = 0; i < passengerCount; i++) {
            passengerPool.add(personBuilder.buildPerson(i+1));
        }
    }

    private void printPassengers() {
        for(Person person : this.passengerPool) {
            LOGGER.log(Level.INFO, String.format(
                    "Id: %d, Name: %s, createdAt: %s%n",
                    person.getId(),
                    person.getFullName(),
                    person.getCreatedAt().toString()));
        }
    }

    private void run() {

        Thread a = new Thread(checkerA);
        Thread b = new Thread(checkerB);
        Thread c = new Thread(checkerC);
        Thread d = new Thread(initialChecker);

        DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss.SSS");
        Date start = new Date();
        d.start();
        a.start();
        b.start();
        c.start();

        //Wait for initial checker to end
        while (d.isAlive()) {
            try {
                d.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
                //try and recover
                System.out.println("Restarting: " + d.getName());
                d.start();
            }
        }
        while ( a.isAlive() || b.isAlive()) {
            if(a.isAlive() && queueA.isEmpty()) {
                a.interrupt();
            }
            if(b.isAlive() && queueB.isEmpty()) {
                b.interrupt();
            }
        }
        while (c.isAlive()) {
            if (queueC.isEmpty()) {
                c.interrupt();
            }
        }

        Date end = new Date();
        System.out.println("\n*** TSA SIMULATOR ***");
        System.out.println("Execution completed!");
        System.out.println("\nSTART: " + dateFormat.format(start));
        System.out.println("END: " + dateFormat.format(end));
        //TODO: Add time elapsed in ticks
        System.out.print("TIME ELAPSED: ");
        System.out.print(getDateDiff(start, end, TimeUnit.SECONDS) + " seconds");
    }

    private TreeSet<Date> generateTimes(int count, int length) {
        //TODO: throw error if there are not count milliseconds in range.
        TreeSet<Date> times = new TreeSet<>();
        long floor = System.currentTimeMillis();
        while(times.size() < count) {
            times.add(generateTime(new Date(floor), new Date(floor + tickValue * length)));
        }

        return times;
    }

    private static long getDateDiff(Date date1, Date date2, TimeUnit timeUnit) {
        long diffInMillis = date2.getTime() - date1.getTime();
        return timeUnit.convert(diffInMillis,TimeUnit.MILLISECONDS);
    }

    private static Date generateTime(Date floor, Date ceiling) {
        Random generator = new Random();
        return new Date(generator.nextInt((int)(ceiling.getTime() - floor.getTime())) + floor.getTime());
    }

    public static void main(String[] args) {
        int tickValue = BASE_TICK_VALUE;
        int maxInitTime = MAX_INITIAL_TIME;
        int passCount = PASSENGER_COUNT;
        if (args.length >= 3) {
            try {
                tickValue = Integer.parseInt(args[2]);
            } catch (NumberFormatException e) {
                LOGGER.log(Level.WARNING, "Cannot parse tick value, using default");
        tickValue = BASE_TICK_VALUE;
      }
      try {
        maxInitTime = Integer.parseInt(args[1]);
      } catch (NumberFormatException e) {
        LOGGER.log(Level.WARNING, "Cannot parse initial time value, using default");
        maxInitTime = MAX_INITIAL_TIME;
      }
      try {
        passCount = Integer.parseInt(args[0]);
      } catch (NumberFormatException e) {
        LOGGER.log(Level.WARNING, "Cannot parse passenger count, using default");
        passCount = PASSENGER_COUNT;
      }
    }

    try {
      TSASimulator simulator = new TSASimulator(passCount, maxInitTime, tickValue);
      simulator.run();
    } catch (FileNotFoundException e) {
      LOGGER.log(Level.SEVERE, "Missing name file: ", e);
    }
  }
}

```

```java
package tsa_sim.person;

import java.util.Date;

public class Person {
    private final int id;
    private final String firstName;
    private final String lastName;
    private final Date createdAt;
    //TODO: These should not be in this class for portability, an event listener should really record this.
    //Extend person.
    //First Queue entered (A or B)
    private Date queuedAt = null;
    //Second Queue entered
    private Date finalQueuedAt = null;
    //Queueing complete
    private Date completedAt = null;

    public Person(int id, Date createdAt, String firstName, String lastName) {
        this.createdAt = createdAt;
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    /*
     * GETTERS
     */
    public Date getCreatedAt() {
        return createdAt;
    }

    public int getId() {
        return id;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public String getFullName() {
        return firstName + ' ' + lastName;
    }

    public Date getCompletedAt() {
        return completedAt;
    }

    public Date getQueuedAt() {
        return queuedAt;
    }

    public Date getFinalQueuedAt() {
        return finalQueuedAt;
    }

    /*
     * SETTERS
     */
    public void setCompletedAt(Date completedAt) {
        this.completedAt = completedAt;
    }

    public void setQueuedAt(Date queuedAt) {
        this.queuedAt = queuedAt;
    }

    public void setFinalQueuedAt(Date finalQueuedAt) {
        this.finalQueuedAt = finalQueuedAt;
    }
}
```

```java
package tsa_sim.person;

import java.util.concurrent.LinkedBlockingQueue;

public class PersonQueue extends LinkedBlockingQueue<Person> {

    public PersonQueue() {
        super();
    }
}
```

```java
package tsa_sim.person;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Date;
import java.util.Random;
import java.util.Scanner;

public class PersonBuilder {

    private ArrayList<String> firstNames;
    private ArrayList<String> lastNames;

    public PersonBuilder() throws FileNotFoundException {
        this.seedLists();
    }

    private void seedLists() throws FileNotFoundException {
        //build name databases
        firstNames = new ArrayList<>();
        lastNames = new ArrayList<>();

        File firstNameFile = new File("resources/first_names.txt");
        File lastNameFile = new File("resources/last_names.txt");

        Scanner input = new Scanner(firstNameFile);
        while(input.hasNextLine()) {
            firstNames.add(input.nextLine());
        }
        input = new Scanner(lastNameFile);
        while(input.hasNextLine()) {
            lastNames.add(input.nextLine());
        }
    }

    public Person buildPerson(int id) {
        Random generator = new Random();
        return new Person(
                id,
                new Date(),
                firstNames.get(generator.nextInt(firstNames.size())),
                lastNames.get(generator.nextInt(lastNames.size()))
        );
    }

}

```

```java
package tsa_sim.Checker;

import tsa_sim.person.*;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.concurrent.ThreadLocalRandom;
import java.util.logging.FileHandler;
import java.util.logging.Level;
import java.util.logging.Logger;

import static java.lang.Math.max;

public class Checker implements CheckerInterface {
    private final static Logger LOGGER = Logger.getLogger(Checker.class.getName());
    private final PersonQueue queue;
    private final PersonQueue[] destination;
    //How many ticks should the queue be expedited by.
    private int timeModifier;
    private int previousLength;
    private int tick;
    private String name;
    private DateFormat dateFormat;
    private FileHandler fh;

    public Checker(PersonQueue origin, PersonQueue[] destination, int tick, String 
name) {
        this.queue = origin;
        this.destination = destination;
        this.name = name;
        timeModifier = 0;
        previousLength = 0;
        this.tick = tick;
        dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");
    }

    public void run() {
        Thread.currentThread().setName(name);
        while (true) {
            try {
                if (queue.isEmpty()) {
                    Thread.sleep(tick);
                } else {
                    Thread.sleep(tick * max((ThreadLocalRandom.current().nextInt(15)
+ 1 - timeModifier), 1));
                    process(queue.take());
                }

                if (previousLength < queue.size()) {
                    timeModifier++;
                } else if (previousLength > queue.size()) {
                    if (timeModifier > 0) {
                        timeModifier--;
                    }
                }
                previousLength = queue.size();
            } catch (InterruptedException e) {
                LOGGER.log(Level.INFO, String.format("%s: Ending by interrupt", 
Thread.currentThread().getName()));
                return;
            }
        }
    }

    public void process(Person person) {
        //Set the earliest null timestamp
        CheckerInterface.stamp(person);
        //TODO: align these logs, format the name or something
        LOGGER.log(Level.INFO, String.format(
                "%s processed: Id: %7d, Name: %25s, createdAt: %s, queuedAt: %s, 
finalQueuedAt: %s, completedAt: %s",
                Thread.currentThread().getName(),
                person.getId(),
                person.getFullName(),
                dateFormat.format(person.getCreatedAt()),
                person.getQueuedAt() == null ? null : dateFormat.format(person.
getQueuedAt()),
                person.getFinalQueuedAt() == null ? null : dateFormat.format(person.getFinalQueuedAt()),
                person.getCompletedAt() == null ? null : dateFormat.format(person.getCompletedAt())
                ));
        if (destination.length > 1) {
            destination[ThreadLocalRandom.current().nextInt(destination.length)].add(person);
        } else {
            destination[0].add(person);
        }
    }
}

```

```java
package tsa_sim.Checker;

import tsa_sim.person.Person;
import tsa_sim.person.PersonQueue;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TreeSet;
import java.util.concurrent.ThreadLocalRandom;
import java.util.logging.Level;
import java.util.logging.Logger;

public class OrderedChecker implements CheckerInterface {
    private final static Logger LOGGER = Logger.getLogger(OrderedChecker.class.getName());
    private final PersonQueue queue;
    private final PersonQueue[] destination;
    private TreeSet<Date> popTimes;
    private String name;
    private DateFormat dateFormat;

    public OrderedChecker(PersonQueue origin, PersonQueue[] destination, TreeSet<Date> popTimes, String name) {
        this.queue = origin;
        this.destination = destination;
        this.name = name;
        this.popTimes = popTimes;
        dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");
    }
    /*
     * If times are in the past objects will pop as soon as execution starts
     */
    public void setProcessTimes(TreeSet<Date> times) {
        this.popTimes = times;
    }

    public void run() {
        Thread.currentThread().setName(name);
        while (!popTimes.isEmpty()) {
            if( System.currentTimeMillis() >= popTimes.first().getTime()) {
                popTimes.remove(popTimes.first());
                try {
                    process(queue.take());
                } catch (InterruptedException e) {
                    LOGGER.log(Level.INFO, String.format("%s: Ending by interrupt", Thread.currentThread().getName()));
                    return;
                }
            }
        }

        LOGGER.log(Level.INFO, String.format("%s: No more times, ending execution.", Thread.currentThread().getName()));
    }

    public void process(Person person) {
        //Set the earliest null timestamp
        CheckerInterface.stamp(person);
        //TODO: align these logs, format the name or something
        LOGGER.log(Level.INFO, String.format(
                "%s processed: Id: %7d, Name: %25s, createdAt: %s, queuedAt: %s",
                Thread.currentThread().getName(),
                person.getId(),
                person.getFullName(),
                dateFormat.format(person.getCreatedAt()),
                person.getQueuedAt() == null ? null : dateFormat.format(person.getQueuedAt())
        ));
        if (destination.length > 1) {
            destination[ThreadLocalRandom.current().nextInt(destination.length)].add(person);
        } else {
            destination[0].add(person);
        }

    }
}

```

```java
package tsa_sim.Checker;

import tsa_sim.person.Person;

import java.util.Date;

public interface CheckerInterface extends Runnable {

    void run();

    void process(Person person);

    //TODO: Perform this in observer
    static void stamp(Person person) {
        if (person.getQueuedAt() == null) {
            person.setQueuedAt(new Date());
        } else if (person.getFinalQueuedAt() == null) {
            person.setFinalQueuedAt(new Date());
        } else {
            person.setCompletedAt(new Date());
        }
    }
}

```