НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. ІГОРЯ СІКОРСЬКОГО”

КАФЕДРА АВТОМАТИЗОВАНИХ СИСТЕМ ОБРОБКИ ІНФОРМАЦІЇ І УПРАВЛІННЯ

Комп’ютерний практикум № 3

з дисципліни

“Моделювання систем”

Виконала:

студентка групи ІС-71

Алпаєва Ю.С

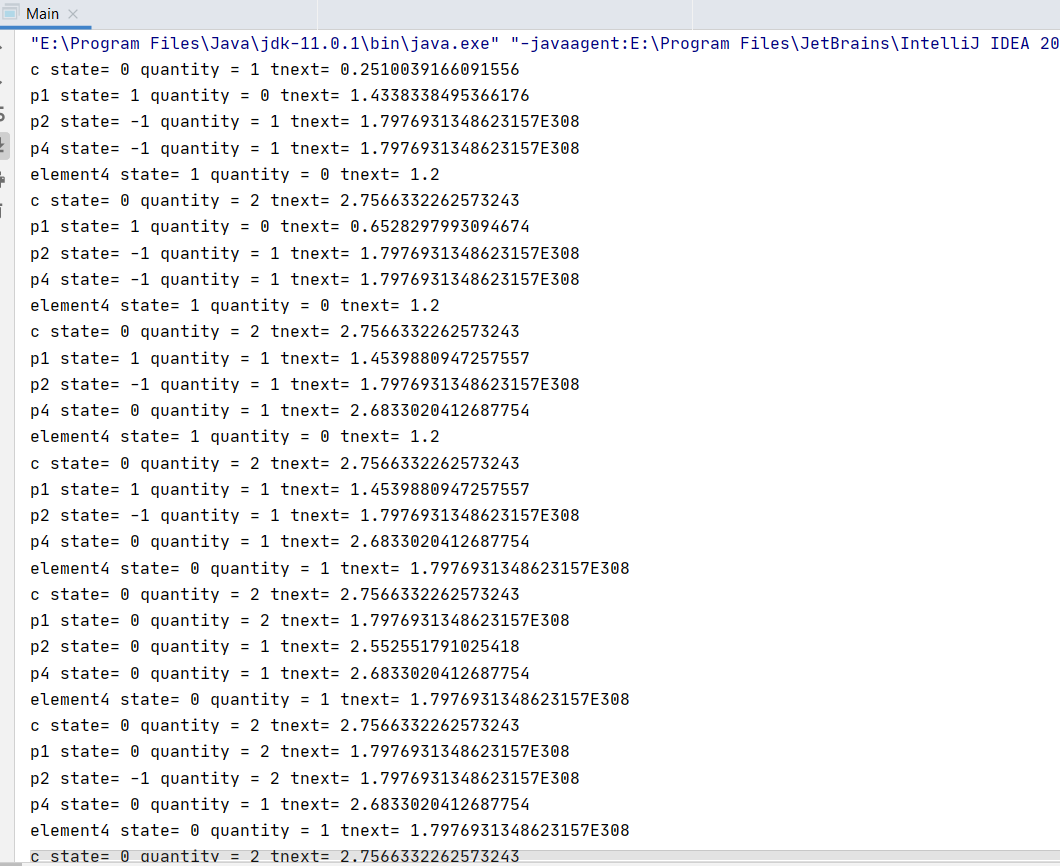
Перевірила:

старший викладач

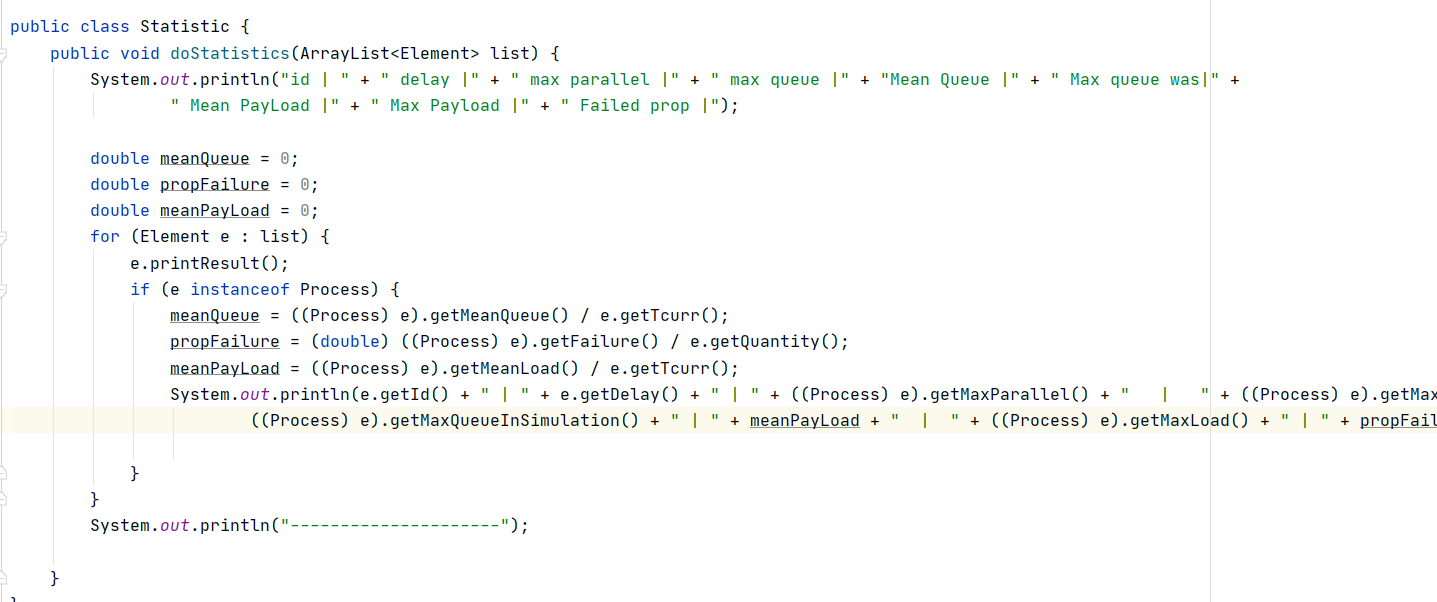
Новікова П.А.

1. Реалізувати алгоритм імітації моделі обслуговування з використанням об’єктно-орієнтованого підходу. **10 балів.**

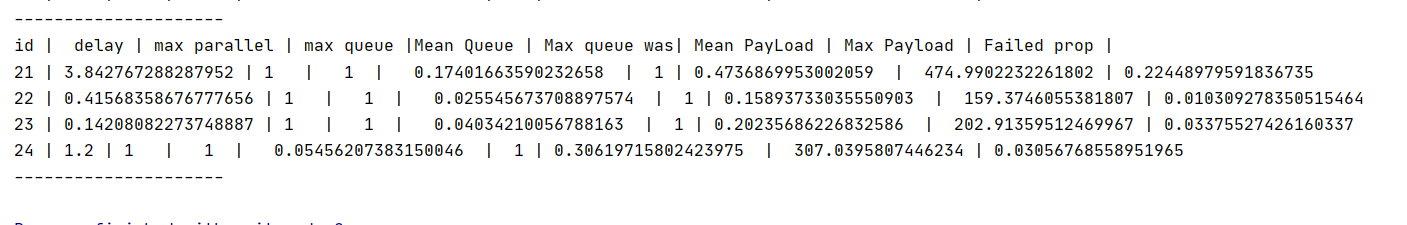
**Вивід програми:**



1. Модифікувати алгоритм, додавши обчислення статистичних характеристик функціонування моделі: ймовірність відмови в обслуговуванні, максимальне спостережуване та середнє спостережуване значення черг, максимальне спостережуване та середнє спостережуване значення завантаження пристроїв. **30 балів.**

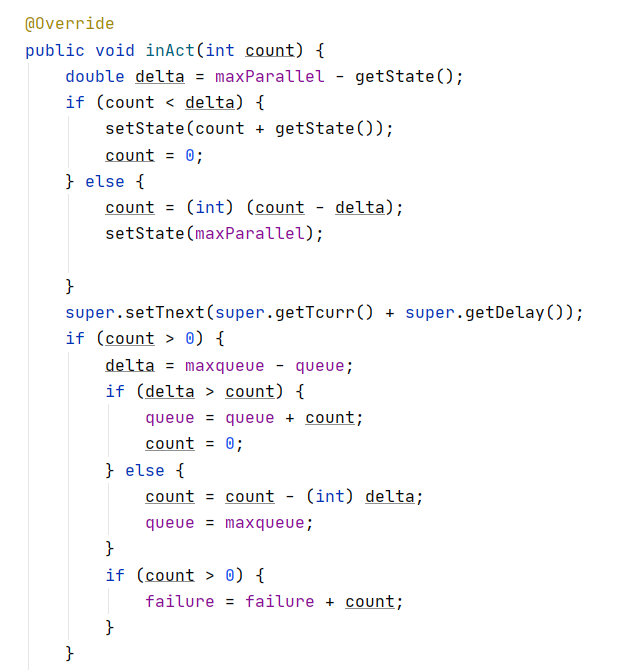


Вивід



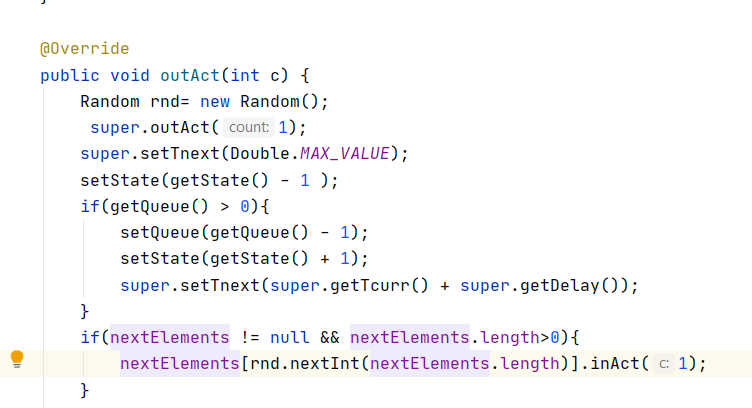
1. Модифікувати клас PROCESS, щоб можна було його використовувати для моделювання процесу обслуговування кількома ідентичними пристроями. **20 балів.**

Для модифікації класу PROCESS було введено змінну maxParallel – кількість пристроїв, що можуть обслуговуватись одночасно і в залежності.



1. Модифікувати клас PROCESS, щоб можна було організовувати вихід в два і більше наступних блоків, в тому числі з поверненням у попередні блоки.

Для цього в PROCESS було додано список, який зберігає список елементів в які можна перейти з PROCESS.



1. Створити модель за схемою, представленою на рисунку 2.1.

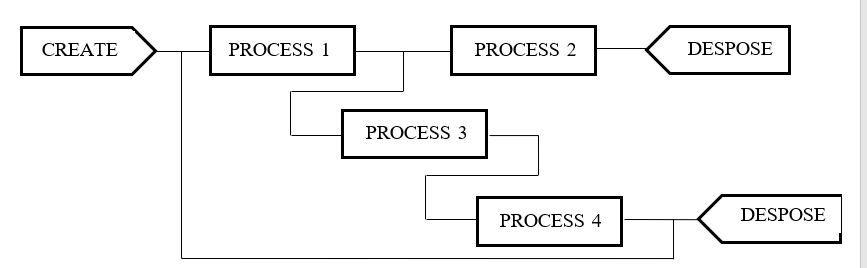
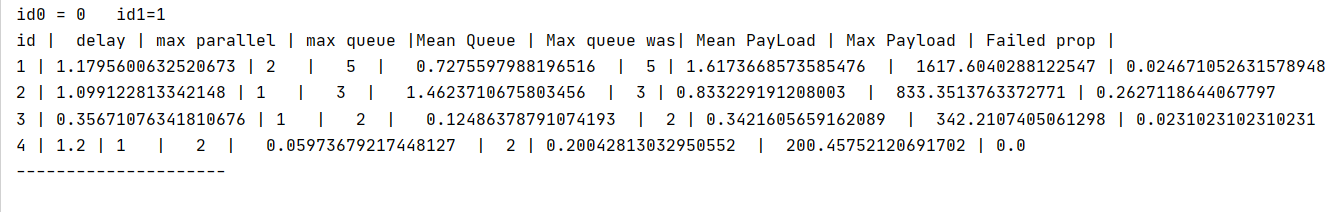


Рисунок 2.1 – Схема моделі.

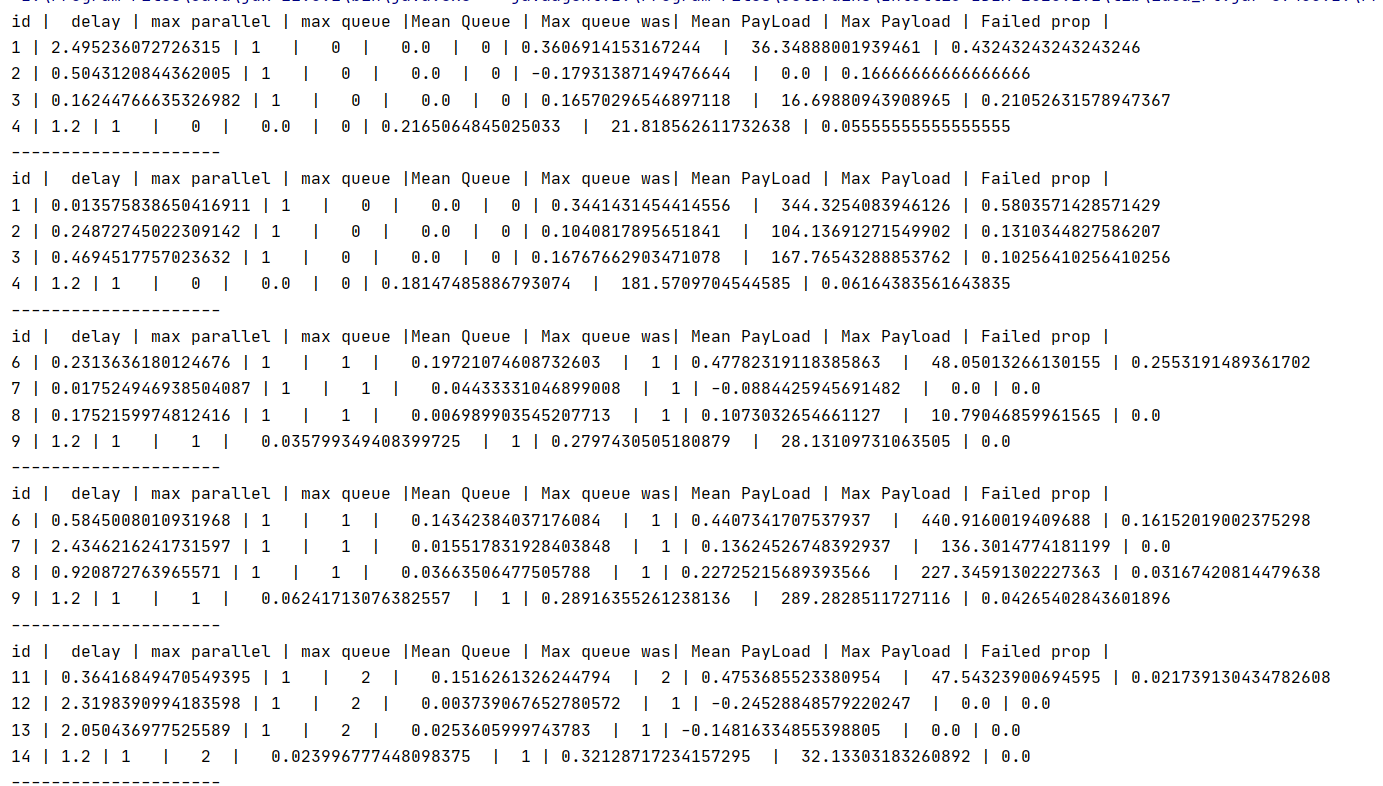
Реалізація в коді:

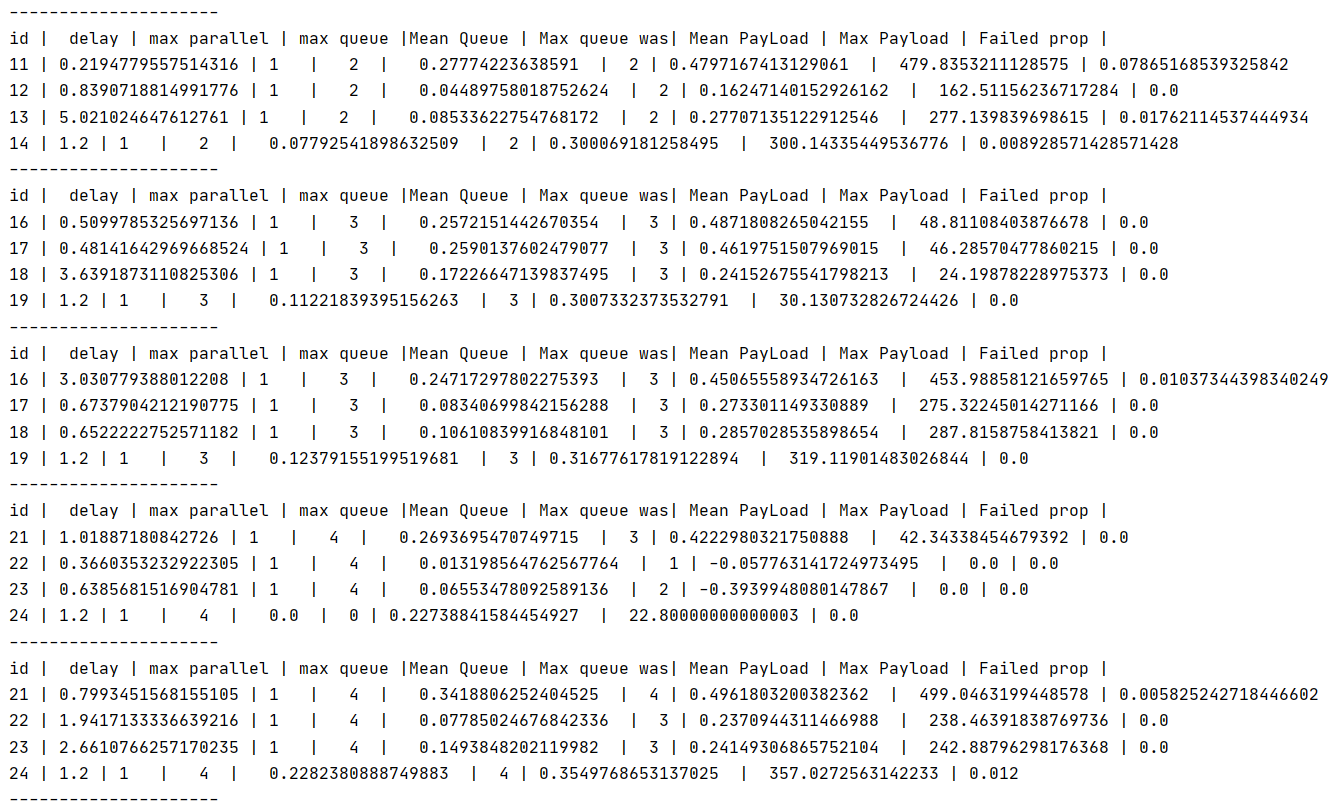




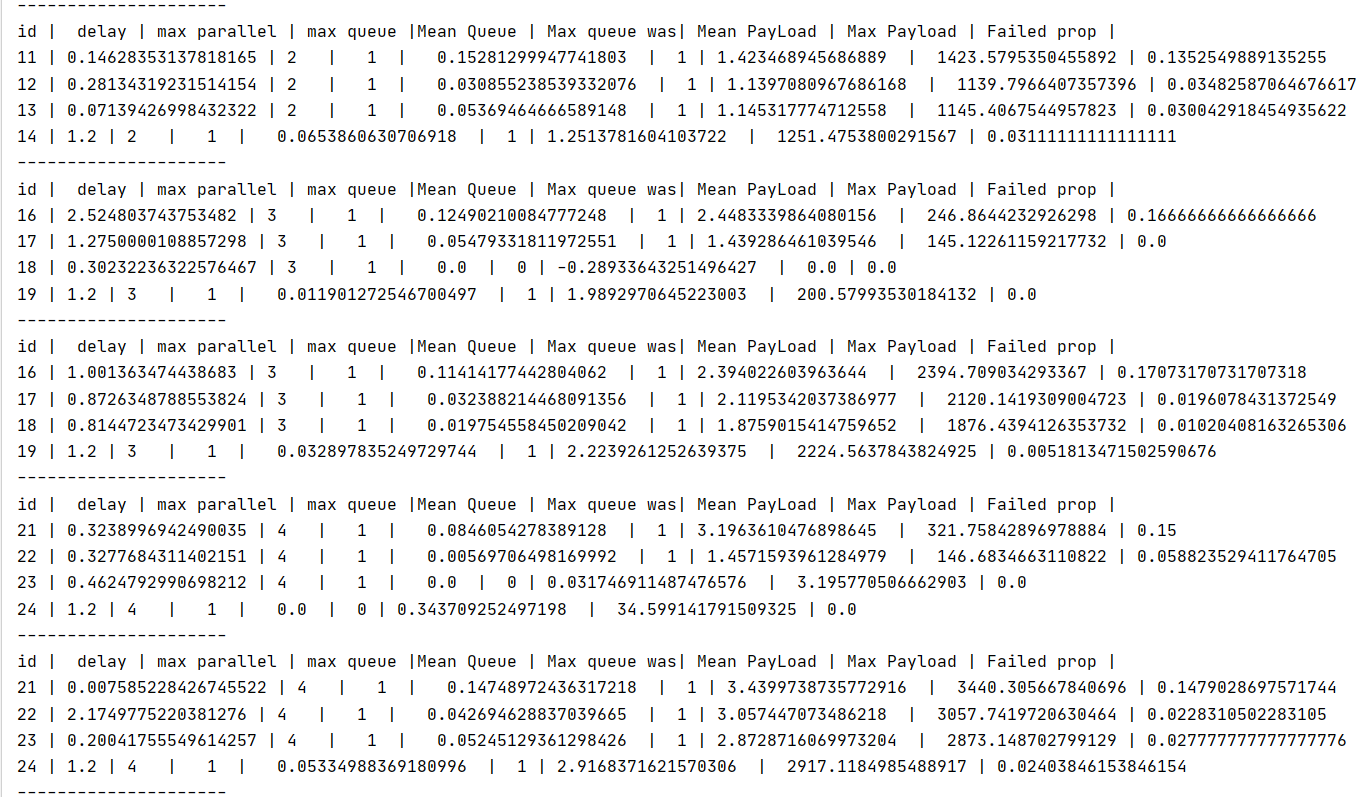
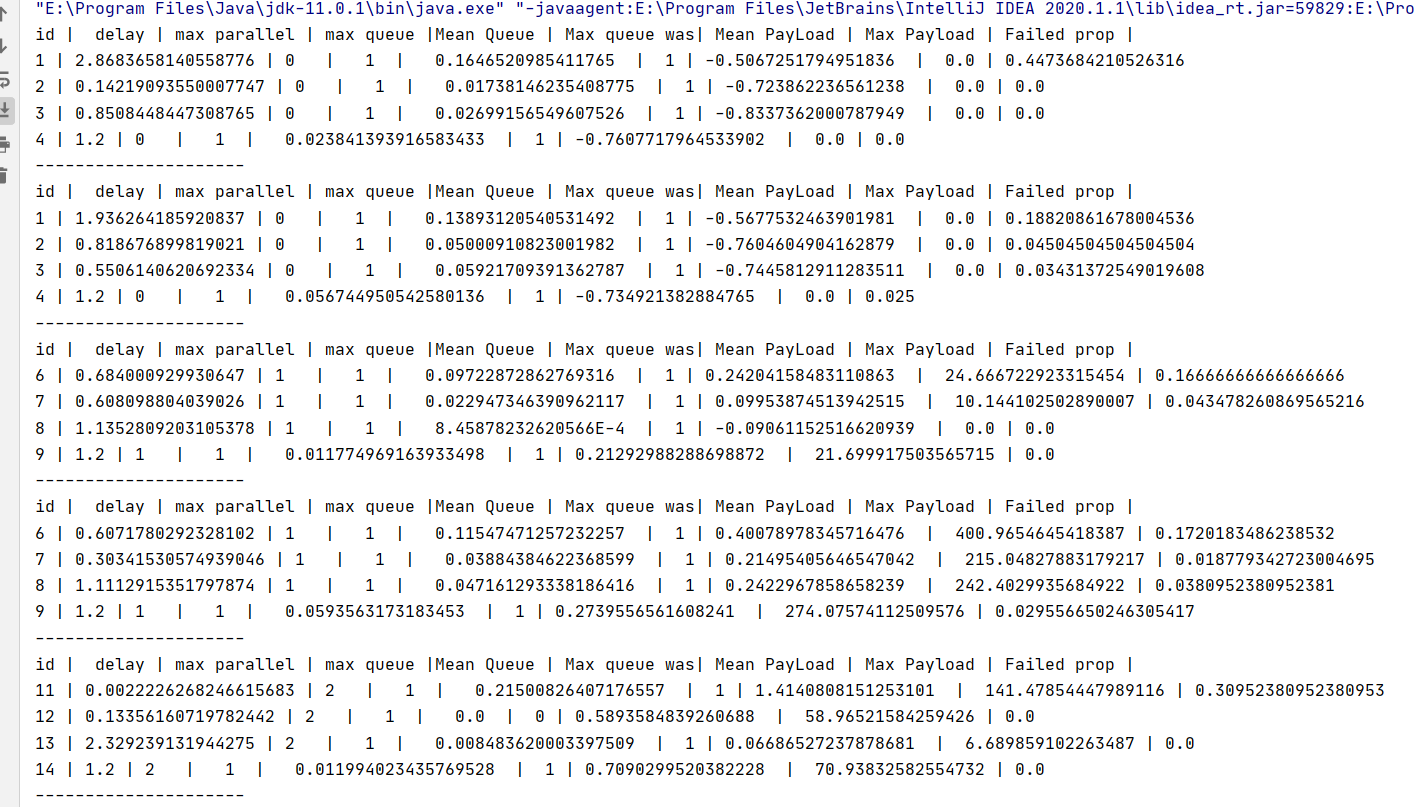
1. Виконати верифікацію моделі, змінюючи значення вхідних змінних та параметрів моделі. Навести результати верифікації у таблиці.

Збільшення параметру максимальної черги:

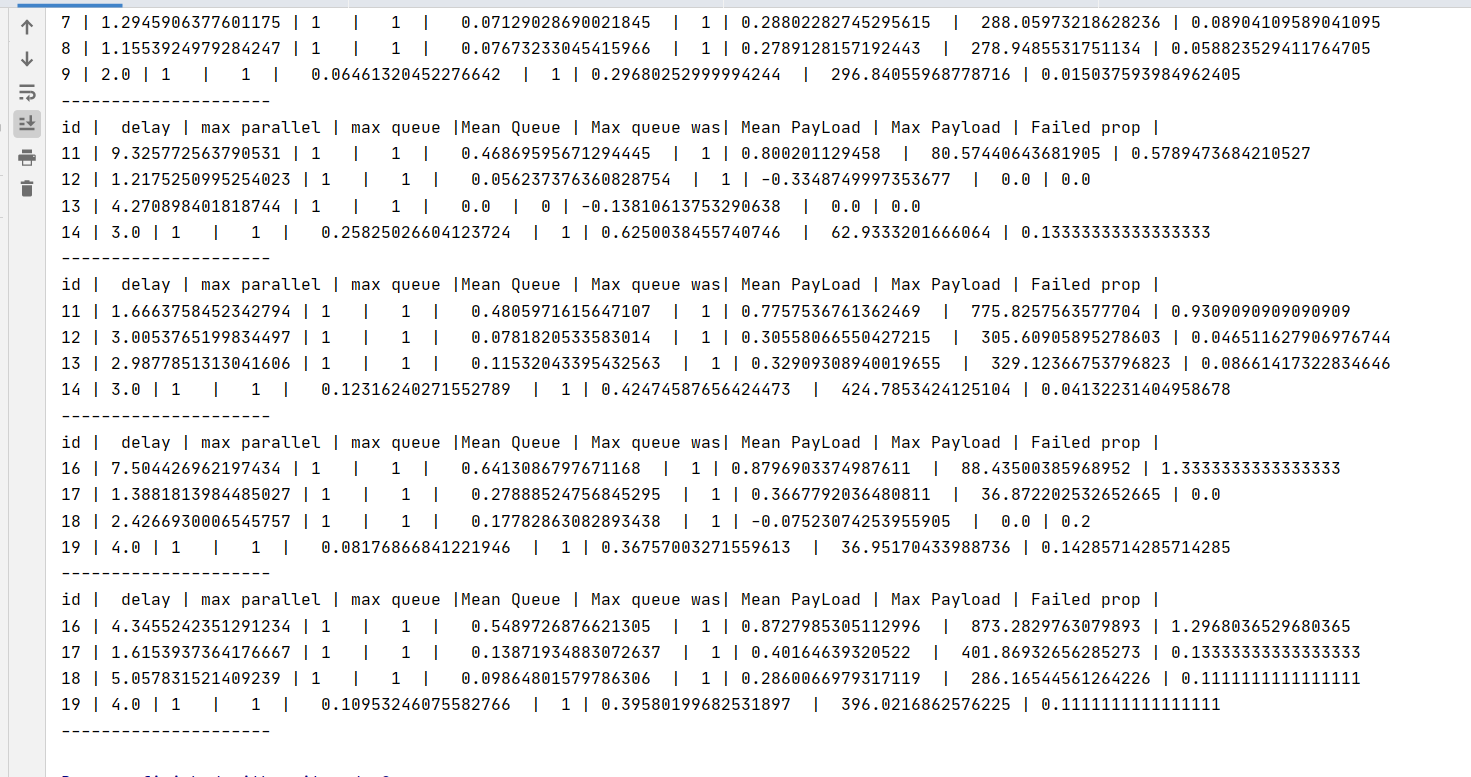
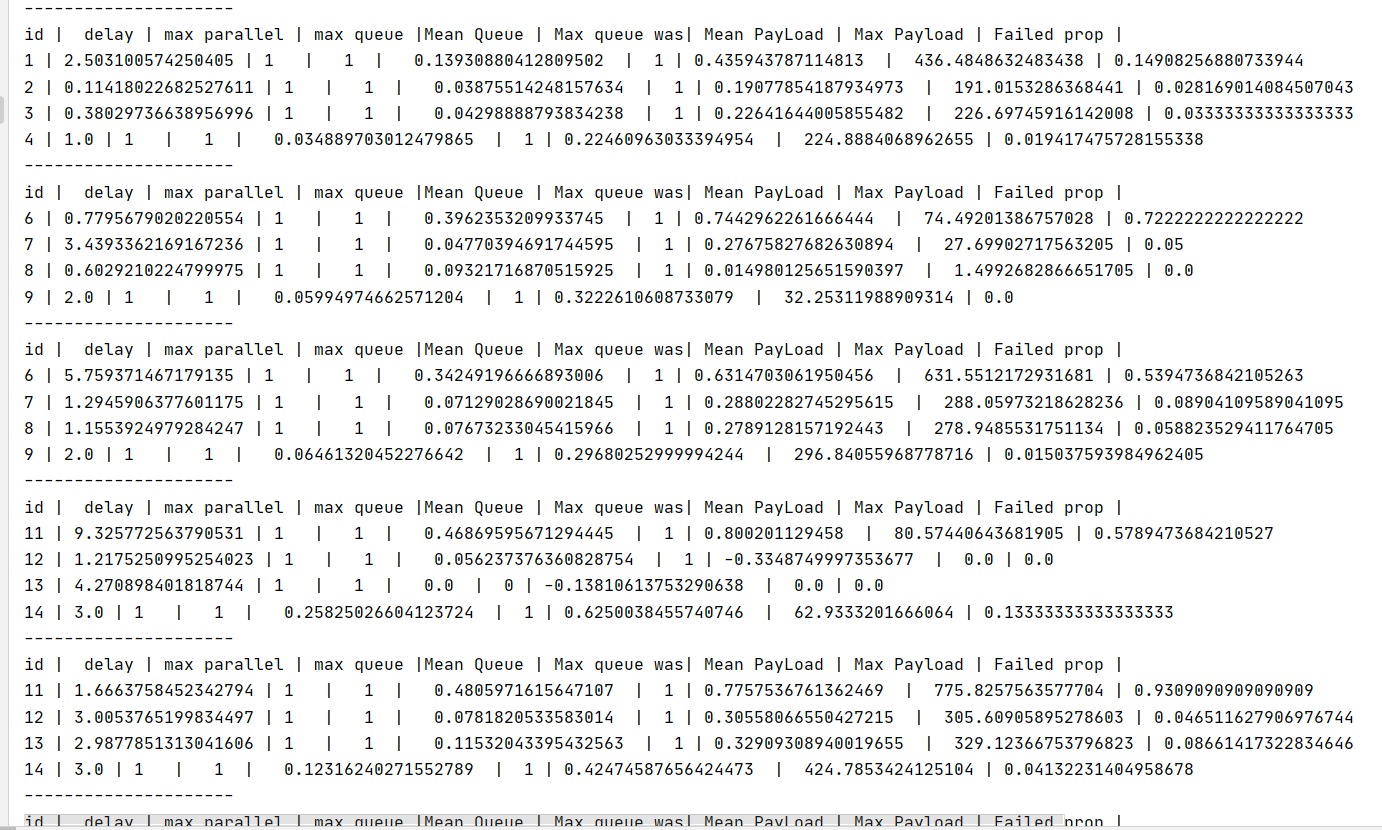




Зміна параметру MaxParallel



Збільшення часу затримки



**Висновок щодо верифікаціі**

Верифікація проводилась таким способом: по черзі запускалась програма з зміною одного параметра, на один.

Висновки після проведення Верифікації системи

|  |  |  |
| --- | --- | --- |
| Параметр,що був змінений | Поведінка системи | Висновок |
| Max queue | При збільшенні параметра поступово зменшувалась кількість відмов, збільшилось навантаження. Навантаження майже не змінилось | Поведінка системи є очікуваною  Чим більше значення максимальної черги тим менша кількість відмов |
| Max Parallel | При збільшенні параметра зменшилась кількість відмов та зросло сререднє навантаження. Зменшилось кількість заяв в черзі | Поведінка системи  є очікуваною.  Збільшилось кількість заяв які можно обробити одночасно на пристрої, отже зменшилась кількість відмов, зросло навантаження. |
| Delay | Зі збільшенням параметру зменшується завантаження системи | Поведінка системи  є очікуваною  Збільшення часу очікування пояснюється тим що на обробку однієї заявки витрачається більше часу, отже затримка стає більшою, збільшується час очікування, зменшується кількість заяв які може обробити система за час моделючання. |

**Висновок**

В ході данної лабораторної роботи було розроблено алгоритм симуляції та зроблениа верифікація моделі. Алгоритм написаний за допомогою об’єктно-орієнтованого підходу, що дає нам змогу гучко налаштовувати систему і модифікувати ії.

При зміні параметрів в усіх випадках поведінка системи є очікованою, отже робота алгоритму є коректною .

**Лістинг**

package com.company;  
  
import java.util.ArrayList;  
  
public class Main {  
  
 public static void main(String[] args) {  
 *// Task 5  
// Create c = new Create(2.0);  
// Process p1 = new Process(1.0,2);  
// Process p2 = new Process(4.0,1);  
// Process p3 = new Process(1.2,1);  
// Process p4 = new Process(1.2,1);  
//  
// System.out.println("id0 = " + c.getId() + " id1=" + p1.getId());  
// c.setNextElement(p1);  
// p1.setMaxqueue(5);  
// p2.setMaxqueue(3);  
// p3.setMaxqueue(2);  
// p4.setMaxqueue(2);  
//  
// c.setName("c");  
// p1.setName("p1");  
// p2.setName("p2");  
// p3.setName("p3");  
//  
// p3.setName("p4");  
//  
// c.setDistribution("exp");  
// p1.setDistribution("exp");  
// p2.setDistribution("exp");  
// p3.setDistribution("exp");  
// p1.setNextElements(new Process[]{p2, p3});  
// p3.setNextElements(new Process[]{p4});  
//  
// ArrayList<Element> list = new ArrayList<>();  
// list.add(c);  
// list.add(p1);  
// list.add(p2);  
// list.add(p3);  
// list.add(p4)  
  
 //Task 6* ModelCreate modelCreate = new ModelCreate();  
 for (int i = 0; i < 5; i++){  
 try {  
 Model m = modelCreate.createModel(i, 1);  
 m.simulate(1000);  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
  
 }  
*// Create c2 = new Create(2.0);  
// Process p12 = new Process(1.0, 1);  
// Process p22 = new Process(1.0, 1);  
// Process p32 = new Process(1.2, 1);  
// Process p42 = new Process(1.2, 1);  
//  
// // System.out.println("id0 = " + c.getId() + " id1=" + p1.getId());  
// c2.setNextElement(p1);  
// p12.setMaxqueue(4);  
// p22.setMaxqueue(4);  
// p32.setMaxqueue(4);  
// p42.setMaxqueue(4);  
//  
// c2.setName("c");  
// p12.setName("p1");  
// p22.setName("p2");  
// p32.setName("p3");  
//  
// p32.setName("p4");  
//  
// c2.setDistribution("exp");  
// p12.setDistribution("exp");  
// p22.setDistribution("exp");  
// p32.setDistribution("exp");  
// p12.setNextElements(new Process[]{p2, p3});  
// p32.setNextElements(new Process[]{p4});  
//  
// ArrayList<Element> list2 = new ArrayList<>();  
// list2.add(c2);  
// list2.add(p12);  
// list2.add(p22);  
// list2.add(p32);  
// list2.add(p42);  
//  
//  
// Model model2 = new Model(list2);  
// model2.simulate(100.0);  
//  
// Create c3 = new Create(2.0);  
// Process p13 = new Process(1.0, 1);  
// Process p23 = new Process(1.0, 1);  
// Process p33 = new Process(1.2, 1);  
// Process p43 = new Process(1.2, 1);  
//  
// // System.out.println("id0 = " + c.getId() + " id1=" + p1.getId());  
// c3.setNextElement(p1);  
// p13.setMaxqueue(4);  
// p23.setMaxqueue(4);  
// p33.setMaxqueue(4);  
// p43.setMaxqueue(4);  
//  
// c2.setName("c");  
// p13.setName("p1");  
// p23.setName("p2");  
// p33.setName("p3");  
//  
// p33.setName("p4");  
//  
// c3.setDistribution("exp");  
// p13.setDistribution("exp");  
// p23.setDistribution("exp");  
// p33.setDistribution("exp");  
// p13.setNextElements(new Process[]{p2, p3});  
// p33.setNextElements(new Process[]{p4});  
//  
// ArrayList<Element> list3 = new ArrayList<>();  
// list3.add(c2);  
// list3.add(p12);  
// list3.add(p22);  
// list3.add(p32);  
// list3.add(p42);  
//  
//  
// Model model3 = new Model(list3);  
// model3.simulate(100.0);* }  
}

package com.company;  
  
public class Create extends Element {  
  
 public Create(double delay) {  
 super(delay);  
 }  
  
 @Override  
 public void outAct(int c) {  
 super.outAct(1);  
 super.setTnext(super.getTcurr() + super.getDelay());  
 super.getNextElement().inAct(1);  
 }  
}

package com.company;  
  
public class Element {  
 private String name;  
 private double tnext;  
 private double delayMean, delayDev;  
 private String distribution;  
 private int quantity;  
 private double tcurr;  
 private int state;  
 private Element nextElement;  
 private static int *nextId* = 0;  
 private int id;  
  
  
 public Element() {  
  
 tnext = 0.0;  
 delayMean = 1.0;  
 distribution = "exp";  
 tcurr = tnext;  
 state = 0;  
 nextElement = null;  
 id = *nextId*;  
 *nextId*++;  
 name = "element" + id;  
 }  
  
 public Element(double delay) {  
 name = "anonymus";  
 tnext = 0.0;  
 delayMean = delay;  
 distribution = "";  
 tcurr = tnext;  
 state = 0;  
 nextElement = null;  
 id = *nextId*;  
 *nextId*++;  
 name = "element" + id;  
 }  
  
 public Element(String nameOfElement, double delay) {  
 name = nameOfElement;  
 tnext = 0.0;  
 delayMean = delay;  
 distribution = "exp";  
 tcurr = tnext;  
 state = 0;  
 nextElement = null;  
 id = *nextId*;  
 *nextId*++;  
 name = "element" + id;  
 }  
  
 public double getDelay() {  
 double delay = getDelayMean();  
 if ("exp".equalsIgnoreCase(getDistribution())) {  
 delay = FunRand.*Exp*(getDelayMean());  
 } else {  
 if ("norm".equalsIgnoreCase(getDistribution())) {  
 delay = FunRand.*Norm*(getDelayMean(),  
 getDelayDev());  
 } else {  
 if ("unif".equalsIgnoreCase(getDistribution())) {  
 delay = FunRand.*Unif*(getDelayMean(),  
 getDelayDev());  
 } else {  
 if ("".equalsIgnoreCase(getDistribution()))  
 delay = getDelayMean();  
 }  
 }  
 }  
  
 return delay;  
 }  
  
  
 public double getDelayDev() {  
 return delayDev;  
 }  
  
 public void setDelayDev(double delayDev) {  
 this.delayDev = delayDev;  
 }  
  
 public String getDistribution() {  
 return distribution;  
 }  
  
 public void setDistribution(String distribution) {  
 this.distribution = distribution;  
 }  
  
  
 public int getQuantity() {  
 return quantity;  
 }  
  
 public double getTcurr() {  
 return tcurr;  
 }  
  
 public void setTcurr(double tcurr) {  
 this.tcurr = tcurr;  
 }  
  
 public int getState() {  
 return state;  
 }  
  
 public void setState(int state) {  
 this.state = state;  
 }  
  
 public Element getNextElement() {  
 return nextElement;  
 }  
  
 public void setNextElement(Element nextElement) {  
 this.nextElement = nextElement;  
 }  
  
 public void inAct(int c) {  
  
 }  
  
 public void outAct(int count) {  
 quantity = quantity + count;  
 }  
  
 public double getTnext() {  
 return tnext;  
 }  
  
  
 public void setTnext(double tnext) {  
 this.tnext = tnext;  
 }  
  
 public double getDelayMean() {  
 return delayMean;  
 }  
  
 public void setDelayMean(double delayMean) {  
 this.delayMean = delayMean;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public void printResult() {  
 *//System.out.println(getName() + " quantity = " + quantity);* }  
  
 public void printInfo() {  
 System.*out*.println(getName() + " state= " + state +  
 " quantity = " + quantity +  
 " tnext= " + tnext);  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public void doStatistics(double delta) {  
  
 }  
}

package com.company;  
  
import java.util.Random;  
  
public class FunRand {  
  
 public static double Exp(double timeMean) {  
 double a = 0;  
 while (a == 0) {  
 a = Math.*random*();package com.company;  
  
import java.util.ArrayList;  
  
public class Model {  
  
 private ArrayList<Element> list = new ArrayList<>();  
 double tnext, tcurr;  
 int event;  
  
 public Model(ArrayList<Element> elements) {  
 list = elements;  
 tnext = 0.0;  
 event = 0;  
 tcurr = tnext;  
 }  
  
  
 public void simulate(double time) {  
 Statistic statistic = new Statistic();  
 while (tcurr < time) {  
 tnext = Double.*MAX\_VALUE*;  
 for (Element e : list) {  
 if (e.getTnext() < tnext) {  
 tnext = e.getTnext();  
 event = e.getId();  
  
 }  
 }  
*// System.out.println("\n event in " +  
// list.get(event).getName() +  
// ", time = " + tnext);  
//* for (Element e : list) {  
 e.doStatistics(tnext - tcurr);  
 }  
 tcurr = tnext;  
 for (Element e : list) {  
 e.setTcurr(tcurr);  
 }  
 if(event < list.size()) {  
 list.get(event).outAct(1);  
 }  
 for (Element e : list) {  
 if (e.getTnext() == tcurr) {  
 e.outAct(1);  
 }  
 }  
 *// printInfo();* }  
 statistic.doStatistics(list);  
 }  
  
 public void printInfo() {  
 for (Element e : list) {  
 *// e.printInfo();* }  
 }  
  
*// public void printResult() {  
//  
// System.out.println("\n-------------RESULTS-------------");  
//  
// for (Element e : list) {  
// e.printResult();  
// if (e instanceof Process) {  
// Process p = (Process) e;  
// p.doResultsStatistics();  
// System.out.println("mean length of queue = " +  
// p.getMeanQueue() / tcurr  
// + "\nfailure probability = " +  
// p.getProbabilityFailure() + " \nMax Queue " + p.getMaxQueueInSimulation()  
// + " Mean Queue " + p.getMeanQueue());  
// }  
// }  
// }*}

}  
 a = -timeMean \* Math.*log*(a);  
  
 return a;  
 }  
  
  
 public static double Unif(double timeMin, double timeMax) {  
 double a = 0;  
 while (a == 0) {  
 a = Math.*random*();  
 }  
 a = timeMin + a \* (timeMax - timeMin);  
  
 return a;  
 }  
 public static double Norm(double timeMean, double timeDeviation) {  
 double a;  
 Random r = new Random();  
 a = timeMean + timeDeviation \* r.nextGaussian();  
  
 return a;  
 }  
  
}

package com.company;  
  
import java.util.ArrayList;  
  
public class ModelCreate {  
 public Model createModel(int q,int parallel){  
 Create c = new Create(2.0);  
 Process p1 = new Process(1.0, parallel);  
 Process p2 = new Process(1.0, parallel);  
 Process p3 = new Process(1.2, parallel);  
 Process p4 = new Process(1.2, parallel);  
 c.setNextElement(p1);  
 p1.setMaxqueue(q);  
 p2.setMaxqueue(q);  
 p3.setMaxqueue(q);  
 p4.setMaxqueue(q);  
 c.setName("c");  
 p1.setName("p1");  
 p2.setName("p2");  
 p3.setName("p3");  
 p3.setName("p4");  
 c.setDistribution("exp");  
 p1.setDistribution("exp");  
 p2.setDistribution("exp");  
 p3.setDistribution("exp");  
 p1.setNextElements(new Process[]{p2, p3});  
 p3.setNextElements(new Process[]{p4});  
 ArrayList<Element> list = new ArrayList<>();  
 list.add(c);  
 list.add(p1);  
 list.add(p2);  
 list.add(p3);  
 list.add(p4);  
 Model model1 = new Model(list);  
 model1.simulate(100.0);  
 return model1;  
 }  
}

package com.company;  
  
import java.util.Collections;  
import java.util.Random;  
  
public class Process extends Element {  
  
 private int queue, maxqueue, failure;  
 private int maxParallel;  
  
 private double meanQueue;  
 private double meanLoad;  
 private double probabilityFailure;  
 private int maxQueueInSimulation = 0;  
 private double maxLoad = 0;  
  
 private Element[] nextElements;  
  
 public Element[] getNextElements() {  
 return nextElements;  
 }  
  
 public void setNextElements(Element[] nextElements) {  
 this.nextElements = nextElements;  
 }  
  
 public Process(double delay, int maxParallel) {  
 super(delay);  
 queue = 0;  
 maxqueue = Integer.*MAX\_VALUE*;  
 meanQueue = 0.0;  
 this.maxParallel = maxParallel;  
 }  
  
 @Override  
 public void inAct(int count) {  
 double delta = maxParallel - getState();  
 if (count < delta) {  
 setState(count + getState());  
 count = 0;  
 } else {  
 count = (int) (count - delta);  
 setState(maxParallel);  
  
 }  
 super.setTnext(super.getTcurr() + super.getDelay());  
 if (count > 0) {  
 delta = maxqueue - queue;  
 if (delta > count) {  
 queue = queue + count;  
 count = 0;  
 } else {  
 count = count - (int) delta;  
 queue = maxqueue;  
 }  
 if (count > 0) {  
 failure = failure + count;  
 }  
 }  
*// if (super.getState() == 0) {  
// super.setState(1);  
// super.setTnext(super.getTcurr() + super.getDelay());  
// } else {  
// if (getQueue() < getMaxqueue()) {  
// setQueue(getQueue() + 1);  
// } else {  
// failure++;  
// }  
// }* }  
  
 @Override  
 public void outAct(int c) {  
 Random rnd= new Random();  
 super.outAct(1);  
 super.setTnext(Double.*MAX\_VALUE*);  
 setState(getState() - 1 );  
 if(getQueue() > 0){  
 setQueue(getQueue() - 1);  
 setState(getState() + 1);  
 super.setTnext(super.getTcurr() + super.getDelay());  
 }  
 if(nextElements != null && nextElements.length>0){  
 nextElements[rnd.nextInt(nextElements.length)].inAct(1);  
 }  
  
*// super.outAct();  
// super.setTnext(Double.MAX\_VALUE);  
// super.setState(0);  
//  
// if (getQueue() > 0) {  
// setQueue(getQueue() - 1);  
// super.setState(1);  
// super.setTnext(super.getTcurr() + super.getDelay());  
// }* }  
  
 public double getMeanLoad() {  
 return meanLoad;  
 }  
  
 public void setMeanLoad(double meanLoad) {  
 this.meanLoad = meanLoad;  
 }  
  
 public int getFailure() {  
 return failure;  
 }  
  
 public int getQueue() {  
 return queue;  
 }  
  
  
 public void setQueue(int queue) {  
 this.queue = queue;  
 }  
  
  
 public int getMaxqueue() {  
 return maxqueue;  
 }  
  
 public double getProbabilityFailure() {  
 return probabilityFailure;  
 }  
  
 public void setProbabilityFailure(double probabilityFailure) {  
 this.probabilityFailure = probabilityFailure;  
 }  
  
 public int getMaxQueueInSimulation() {  
 return maxQueueInSimulation;  
 }  
  
 public void setMeanQueue(double meanQueue) {  
 this.meanQueue = meanQueue;  
 }  
  
 public void setMaxQueueInSimulation(int maxQueueInSimulation) {  
 this.maxQueueInSimulation = maxQueueInSimulation;  
 }  
  
 public void setMaxqueue(int maxqueue) {  
 this.maxqueue = maxqueue;  
 }  
*//  
// @Override  
// public void printInfo() {  
// super.printInfo();  
// System.out.println("failure = " + this.getFailure());  
// }* @Override  
 public void doStatistics(double delta) {  
 meanQueue = getMeanQueue() + queue \* delta;  
 meanLoad = meanLoad + (getState() \* delta);  
 if (maxQueueInSimulation < queue) {  
 maxQueueInSimulation = queue;  
 }  
 if(maxLoad < meanLoad){  
 maxLoad = meanLoad;  
 }  
  
  
 }  
  
 public double getMaxLoad() {  
 return maxLoad;  
 }  
  
 public void setMaxLoad(double maxLoad) {  
 this.maxLoad = maxLoad;  
 }  
  
 public void doResultsStatistics() {  
*// if (failure != 0 && getQuantity() != 0) {  
//  
// probabilityFailure = (double) failure / getQuantity();  
// meanQueue = meanQueue / getTcurr();  
// }* }  
  
 public int getMaxParallel() {  
 return maxParallel;  
 }  
  
 public void setMaxParallel(int maxParallel) {  
 this.maxParallel = maxParallel;  
 }  
  
 public double getMeanQueue() {  
 return meanQueue;  
 }  
}

package com.company;  
  
import java.util.ArrayList;  
  
public class Statistic {  
 public void doStatistics(ArrayList<Element> list) {  
 System.*out*.println("id | " + " delay |" + " max parallel |" + " max queue |" + "Mean Queue |" + " Max queue was|" +  
 " Mean PayLoad |" + " Max Payload |" + " Failed prop |");  
  
 double meanQueue = 0;  
 double propFailure = 0;  
 double meanPayLoad = 0;  
 for (Element e : list) {  
 e.printResult();  
 if (e instanceof Process) {  
 meanQueue = ((Process) e).getMeanQueue() / e.getTcurr();  
 propFailure = (double) ((Process) e).getFailure() / e.getQuantity();  
 meanPayLoad = ((Process) e).getMeanLoad() / e.getTcurr();  
 System.*out*.println(e.getId() + " | " + e.getDelay() + " | " + ((Process) e).getMaxParallel() + " | " + ((Process) e).getMaxqueue() + " | " + meanQueue + " | " +  
 ((Process) e).getMaxQueueInSimulation() + " | " + meanPayLoad + " | " + ((Process) e).getMaxLoad() + " | " + propFailure);  
  
 }  
 }  
 System.*out*.println("---------------------");  
  
 }  
}