**CS115 Computer Simulation,Assignment#2**

Write-up

First, my code is based on eclipse and perfectly works. I used the external jar package and of course, process drive approach. There is 5 process, sim process, arrival process, enterdock process, hoggout process and replacement process. First generate arrival process and it calls other processes to run the hole simulation. For example, arrive calls enterdock or hogged out,once enter dock, calculate departure time and if it is before hogged out, hogged out will not be executed. There are too much judgement calling other processes, so I can’t write them all, but based on the output, you can see my program is correct.

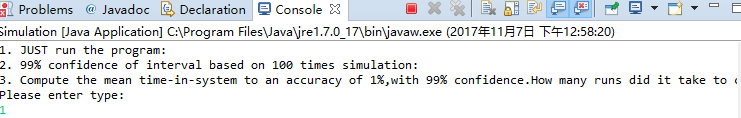
Second, my output is almost perfect. But there is a flaw in the dock-hogged out percentage.The added percentage is pretty nearly to 100 but not exactly. Maybe I counted some of other hog time into the dock-hogged out time. But I didn’t have time to correct it. While other percentage or out-put is no problem I believe. Furthermore, in assignment2, there are two new questions, first is compute 99% confidence of interval based on 100 times simulation and my output is 6.42-6.49, the difference between two numbers is 0.07, this is small enough and I believe it is a correct number. Second is compute the mean time-in-system to an accuracy of 1%, with 99% confidence.How many runs did it take to compute this? And my program answer is 108. This out-put is fairly make sense because I did check out each run and 108 is a correct number of this question.

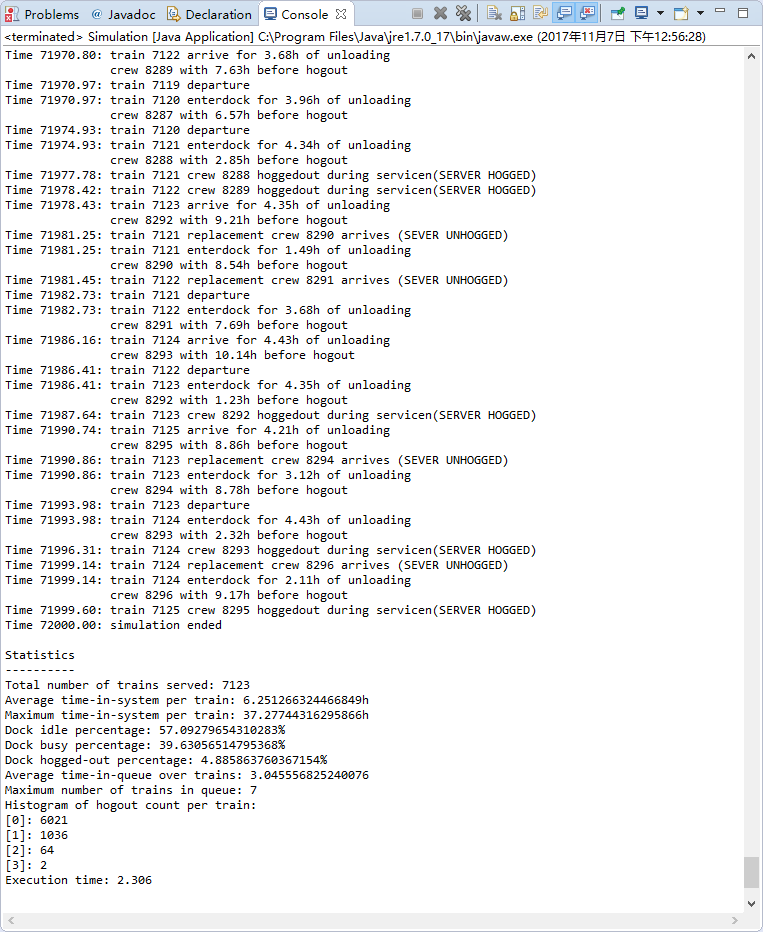
Third, there is a problem about random numbers. I don’t know why I can’t generate random numbers by using rand.uniform(), so I set seeds before that and the problem is that the seeds always start from 1 and determined by run times. For example, if you run the code one time and then run another time, trains arrive time is the same between two runs because the time seed is the same at the same time. Except that, I didn’t find any bugs in my code.

**INTRODUTION**

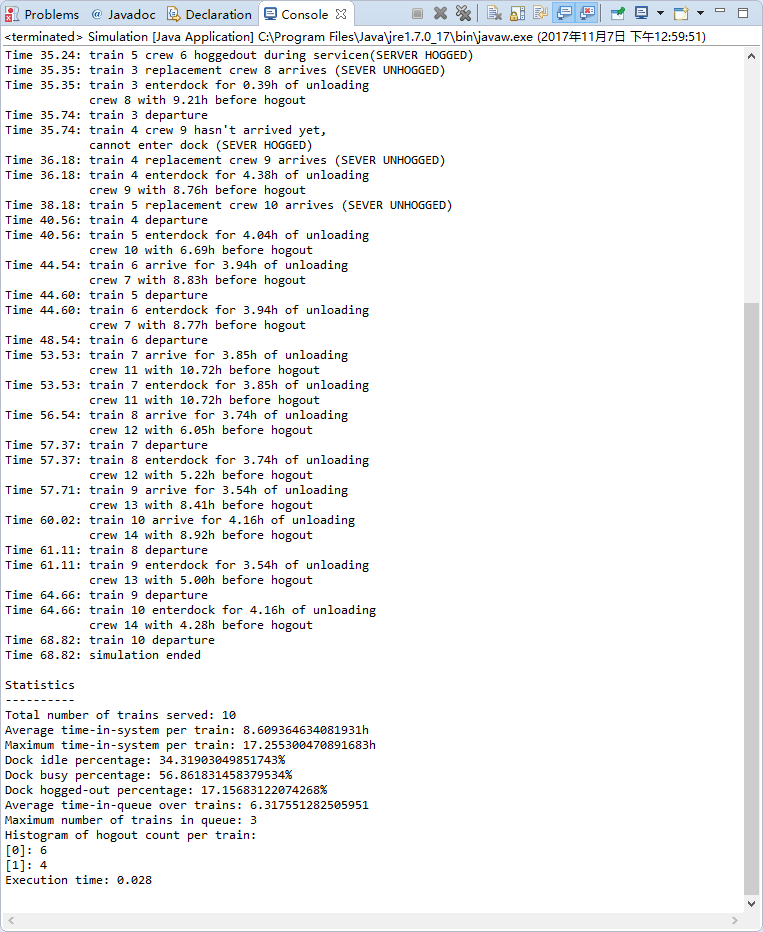
This program is based on eclipse.And it realizes the function blow:

1: insert 10 72000 simulation and select type1

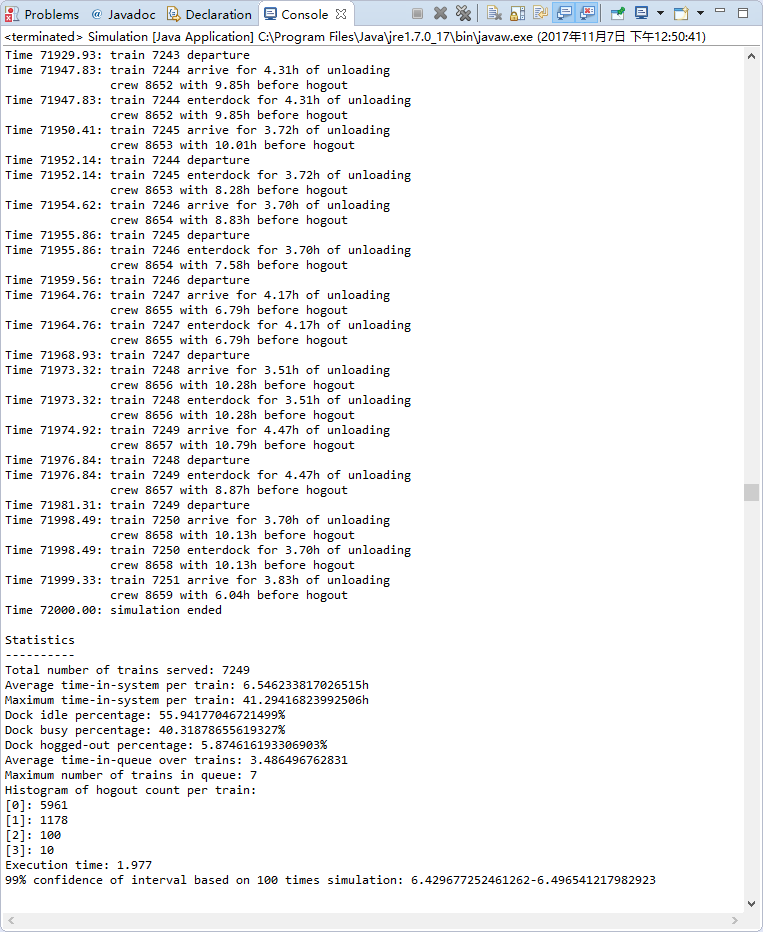




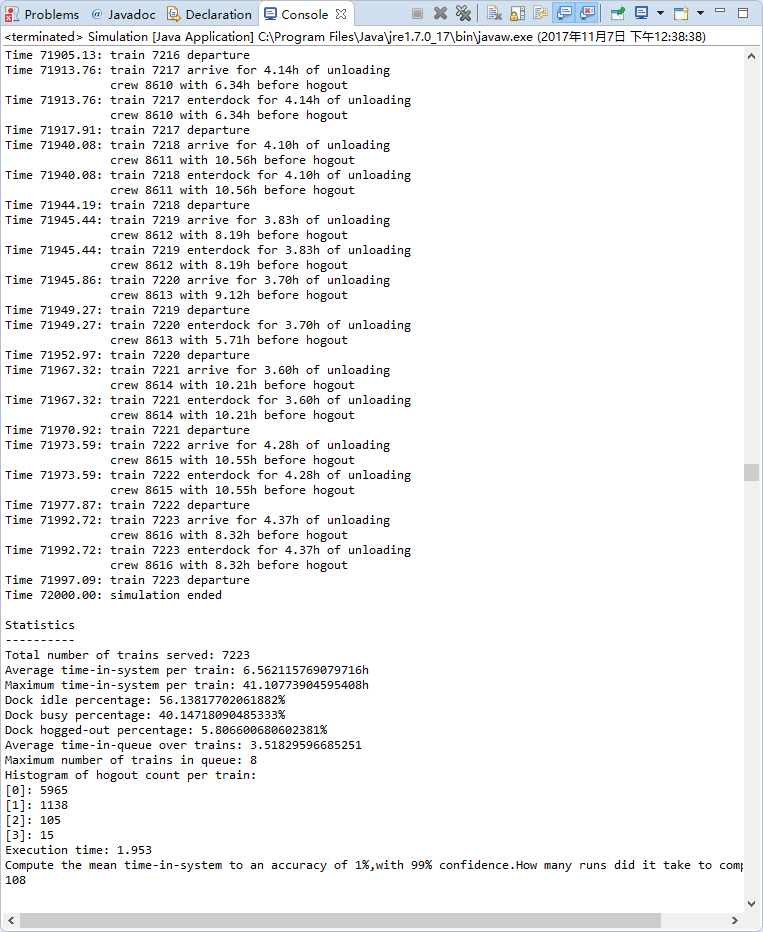
2.insert -s -d schedule.txt traveltimes.txt



3.insert “10 72000 simulation” and select type 2 to compute 99% confidence of interval based on 100 times simulation:



4.Insert “10 72000 simulation” and select type 3 to compute Compute the mean time-in-system to an accuracy of 1%,with 99% confidence.How many runs did it take to compute this?And my program answer is 108.



**SOURCE CODE**

import java.util.LinkedList;

import com.mesquite.csim.\*;

import com.mesquite.csim.Process;

import java.lang.Math;

import java.io.File;

import java.io.InputStreamReader;

import java.io.BufferedReader;

import java.io.FileInputStream;

import java.io.IOException;

import java.util.Scanner;

public class Simulation extends Model

{

public static void main(String[] args)

{

if(args.length==3)

{

@SuppressWarnings("resource")

Scanner sc = new Scanner(System.in);

System.out.println("1. JUST run the program:"+"\n"+"2. 99% confidence of interval based on 100 times simulation:"+"\n"+"3. Compute the mean time-in-system to an accuracy of 1%,with 99% confidence.How many runs did it take to compute this?"+"\n"+"Please enter type:");

Type = sc.nextLine();

poissontime=Double.parseDouble(args[0]);

T=Double.parseDouble(args[1]);

if(Double.parseDouble(Type)==1)

{

Simulation model = new Simulation();

//model.enableTrace(true);

model.run();

}

else if(Double.parseDouble(Type)==2)

{

timetable= new double [100];

tabletimes=0;

iftable=true;

double sum=0.0;

for(int i=0;i<100;i++)

{

Simulation model = new Simulation();

//model.enableTrace(true);

model.run();

sum+=timetable[tabletimes];

tabletimes++;

}

//calculate mean/var

double mean=sum/100;

double var=0.0;

double temp=0.0;

for(int i=0;i<100;i++)

{

temp+=(timetable[i]-mean)\*(timetable[i]-mean);

}

var=temp/100;

double lower=mean-2.58\*Math.sqrt(var/100);

double higher=mean+2.58\*Math.sqrt(var/100);

System.out.println("99% confidence of interval based on 100 times simulation: "+lower+"-"+higher);

}

else if(Double.parseDouble(Type)==3)

{

timetable= new double [1000];

tabletimes=0;

iftable=true;

double mean=0.0;

double sum=0.0;

double var=0.0;

double temp=0.0;

double wide=0.0;

for(;wide==0||wide>=0.01\*mean;)

{

Simulation model = new Simulation();

//model.enableTrace(true);

model.run();

sum+=timetable[tabletimes];

tabletimes++;

mean=sum/tabletimes;

temp=0.0;

for(int i=0;i<tabletimes;i++)

{

temp+=(timetable[i]-mean)\*(timetable[i]-mean);

}

var=temp/tabletimes;

double lower=mean-2.58\*Math.sqrt(var/tabletimes);

double higher=mean+2.58\*Math.sqrt(var/tabletimes);

wide=higher-lower;

}

System.out.println("Compute the mean time-in-system to an accuracy of 1%,with 99% confidence.How many runs did it take to compute this? "+"\n"+tabletimes);

}

}

else if(args.length==4)

{

schdule=new File(args[2]);

traveltimes=new File(args[3]);

iffile=true;

Simulation model = new Simulation();

//model.enableTrace(true);

model.run();

}

}

public void run()

{

start(new Sim());

}

//simulation

double nextarrive\_time=0.0;

int train\_num=0;

static double T;

boolean busy=false;

LinkedList<train> dock=new LinkedList<train>();

static double poissontime;

int crewnum=0;

//file

static boolean iffile=false;

static File schdule;

static File traveltimes;

String array[]=null;

BufferedReader br; //schedule.txt

BufferedReader tr; //traveltimes.txt

boolean iffirst=true;//delete time0:00 train arrive

//statistics

int trainserved=0;

LinkedList<train> statisticlist=new LinkedList<train>();

double busytime=0.0;

double idletime=0.0;

double hogtime=0.0;

double timeinqueue=0.0;//when the train is hog out but it's its turn to get in the dock,this period of waiting crew time is in the queue

int queuemaxnum=0;

//confidence interval

static String Type="";

static double timetable[];

static int tabletimes;

static boolean iftable=false;

//initialize

void initialize()

{

nextarrive\_time=0.0;

train\_num=0;

busy=false;

dock=new LinkedList<train>();

crewnum=0;

//statistics

trainserved=0;

statisticlist=new LinkedList<train>();

busytime=0.0;

idletime=0.0;

hogtime=0.0;

timeinqueue=0.0;//when the train is hog out but it's its turn to get in the dock,this period of waiting crew time is in the queue

queuemaxnum=0;

}

public class crew

{

private double remaintime=0.0;

private int Number=0;

public void setremaintime(double temp)

{

remaintime=temp;

}

public double getremaintime()

{

return remaintime;

}

public void setcrewnum(int temp)

{

Number=temp;

}

public double getcrewnum()

{

return Number;

}

}

public class train

{

private double arrive\_time=0.0;

private int number=0;

private boolean leave=false;

public crew c=new crew();

private boolean hogout=false;

private double departuretime=0.0;

private double hogtime=0.0;

private double unloadtime=0.0;//could be change during each update unload time

private double totalunloadtime=0.0;//for statistic

private double temptime=0.0;//calculate crew remain time

private int hogtimes=0;

public double getarrivetime()

{

return arrive\_time;

}

public void setarrivetime(double temp)

{

arrive\_time=temp;

}

public int gettrainnum()

{

return number;

}

public void settrainnum(int temp)

{

number=temp;

}

public void setleave()

{

leave=true;

}

public boolean ifleave()

{

return leave;

}

public void printremaintime()

{

String temp=String.format("%.2f", clock());

int l=temp.length()+7;

for(int i=0;i<l;i++)

{

System.out.printf(" ");

}

System.out.println("crew "+(int)c.getcrewnum() + " with " + String.format("%.2f", c.getremaintime())+"h before hogout");

}

public boolean ifhogout()

{

return hogout;

}

public void sethogout(boolean temp)

{

hogout=temp;

}

public double getdeparturetime()

{

return departuretime;

}

public void setdeparturetime(double temp)

{

departuretime=temp;

}

public double gethogtime()

{

return hogtime;

}

public void sethogtime(double temp)

{

hogtime=temp;

}

public void setunloadtime(double temp)

{

unloadtime=temp;

}

public double getunloadtime()

{

return unloadtime;

}

public void settemptime(double temp)

{

temptime=temp;

}

public double gettemptime()

{

return temptime;

}

public void settotalunloadtime()

{

totalunloadtime=getunloadtime();

}

public double gettotalunloadtime()

{

return totalunloadtime;

}

public void addhogtimes()

{

hogtimes++;

}

public int gethogtimes()

{

return hogtimes;

}

}

private class Sim extends Process

{

public Sim()

{

super("Simulation");

}

public void run()

{

initialize();

add(new Arrival());

if(iffile==false)

{

hold(T);

System.out.println("Time "+String.format("%.2f", clock())+": simulation ended");

}

else

{

hold(75000);

System.out.println("Time "+String.format("%.2f", statisticlist.getLast().getdeparturetime())+": simulation ended");

}

statistics();

}

}

private class Arrival extends Process

{

public Arrival() {

super("Arrival");

}

public void run()

{

if(iffile==true)

{

try

{

InputStreamReader reader = new InputStreamReader(new FileInputStream(schdule));

br = new BufferedReader(reader);

InputStreamReader reader2 = new InputStreamReader(new FileInputStream(traveltimes));

tr = new BufferedReader(reader2);

}

catch (Exception e)

{

e.printStackTrace();

}

}

if(iftable==true)//I don't know why my random not random, so only this way...

{

rand.setSeed(tabletimes);

}

while(true)

{

train t=new train();

double arriveinterval=0.0;

train\_num++;

crewnum++;

t.settrainnum(train\_num);

t.c.setcrewnum(crewnum);

if(iffile==false)

{

t.setarrivetime(nextarrive\_time);

t.settemptime(nextarrive\_time);

arriveinterval= -Math.log(rand.uniform(0,1)) \* poissontime;

t.setunloadtime(rand.uniform(3.5, 4.5));

t.c.setremaintime(rand.uniform(6, 11));

t.settotalunloadtime();

dock.addLast(t);

System.out.println("Time "+String.format("%.2f", clock())+": train " + t.gettrainnum() + " arrive for "+String.format("%.2f",t.getunloadtime())+"h of unloading");

t.printremaintime();

if(dock.size()-1>queuemaxnum)//max train in queue(not include the train in the dock)

{

queuemaxnum=dock.size()-1;

}

if(!busy)

{

if(!statisticlist.isEmpty())

{

idletime+=clock()-statisticlist.getLast().getdeparturetime();

}

busy=true;

add(new enterdock(t));

}

else

{

t.sethogout(true);

add(new hogout(t));

}

hold(arriveinterval);

if(nextarrive\_time>T)

{

break;

}

nextarrive\_time+=arriveinterval;

}

else//read file

{

String line = "";

try {

line = br.readLine();

} catch (IOException e) {

e.printStackTrace();

}

if(line!=null)

{

array=line.split(" ");

t.setarrivetime(Double.parseDouble(array[0]));

t.settemptime(Double.parseDouble(array[0]));

arriveinterval=Double.parseDouble(array[0])-nextarrive\_time;

t.setunloadtime(Double.parseDouble(array[1]));

t.c.setremaintime(Double.parseDouble(array[2]));

t.settotalunloadtime();

hold(arriveinterval);

}

else

{

break;

}

dock.addLast(t);

System.out.println("Time "+String.format("%.2f", clock())+": train " + t.gettrainnum() + " arrive for "+String.format("%.2f",t.getunloadtime())+"h of unloading");

t.printremaintime();

if(dock.size()-1>queuemaxnum)//max train in queue(not include the train in the dock)

{

queuemaxnum=dock.size()-1;

}

if(!busy)

{

if(!statisticlist.isEmpty())

{

idletime+=clock()-statisticlist.getLast().getdeparturetime();

}

busy=true;

add(new enterdock(t));

}

else

{

t.sethogout(true);

add(new hogout(t));

}

nextarrive\_time=Double.parseDouble(array[0]);

}

}

}

}

private class enterdock extends Process

{

train temp=new train();

public enterdock(train t)

{

super("enterdock");

temp=t;

}

public void run()

{

double unloadinterval=dock.getFirst().getunloadtime();

if(temp.c.getremaintime()>=unloadinterval)

{

timeinqueue+=clock()-temp.getarrivetime();

System.out.println("Time "+String.format("%.2f", clock())+": train " + temp.gettrainnum() + " enterdock for " + String.format("%.2f", unloadinterval)+"h of unloading");

temp.printremaintime();

hold(unloadinterval);

departure(temp);

}

else if(temp.c.getremaintime()<unloadinterval&&temp.c.getremaintime()>0)

{

timeinqueue+=clock()-temp.getarrivetime();

System.out.println("Time "+String.format("%.2f", clock())+": train " + temp.gettrainnum() + " enterdock for " + String.format("%.2f", unloadinterval)+"h of unloading");

temp.printremaintime();

dock.getFirst().setunloadtime(unloadinterval-temp.c.getremaintime());

}

else if(temp.c.getremaintime()<0)

{

System.out.println("Time "+String.format("%.2f", clock())+": train " + temp.gettrainnum() + " crew "+(int)temp.c.getcrewnum()+" hasn't arrived yet,");

String temp=String.format("%.2f", clock());

int l=temp.length()+7;

for(int i=0;i<l;i++)

{

System.out.printf(" ");

}

System.out.println("cannot enter dock (SEVER HOGGED)");

}

}

}

private class hogout extends Process

{

train temp=new train();

public hogout(train t)

{

super("hogout");

temp=t;

}

public void run()

{

double hoginterval=temp.c.getremaintime();

if(temp.gettrainnum()==dock.getFirst().gettrainnum())

{

add(new enterdock(temp));

}

else

{

hold(hoginterval);

if(!temp.ifleave())

{

temp.sethogtime(clock());

System.out.println("Time "+String.format("%.2f", clock())+": train " + temp.gettrainnum() + " crew "+ (int)temp.c.getcrewnum()+" hoggedout during servicen(SERVER HOGGED)" );

crewnum++;

temp.c.setcrewnum(crewnum);

add(new replacement(temp));

}

}

}

}

private class replacement extends Process

{

train temp=new train();

public replacement(train t)

{

super("replace");

temp=t;

}

public void run()

{

double replacetime=0.0;

if(iffile==false)

{

replacetime= rand.uniform(2.5,3.5);

}

else

{

String line="";

try {

line = tr.readLine();

} catch (IOException e) {

e.printStackTrace();

}

replacetime=Double.parseDouble(line);

}

hold(replacetime);

System.out.println("Time "+String.format("%.2f", clock())+": train " + temp.gettrainnum() + " replacement crew "+ (int)temp.c.getcrewnum()+" arrives (SEVER UNHOGGED)");

temp.c.setremaintime(12.0-replacetime);

for(int i=0;i<dock.size();i++)//set crew

{

if(dock.get(i).gettrainnum()==temp.gettrainnum())

{

dock.get(i).settemptime(clock());

dock.get(i).c.setcrewnum((int)temp.c.getcrewnum());

}

}

hogtime+=clock()-temp.gethogtime();

temp.addhogtimes();

add(new hogout(temp));

}

}

public void departure(train temp)

{

dock.removeFirst();

temp.setleave();

System.out.println("Time "+String.format("%.2f", clock())+": train " + temp.gettrainnum() + " departure");

trainserved++;

temp.setdeparturetime(clock());

statisticlist.addLast(temp);

if(!dock.isEmpty())

{

dock.getFirst().c.setremaintime(dock.getFirst().c.getremaintime()-(clock()-dock.getFirst().gettemptime()));

add(new enterdock(dock.getFirst()));

}

else

{

busy=false;

}

}

public void statistics()

{

System.out.println("\n"+"Statistics"+"\n"+"----------"+"\n"+"Total number of trains served: "+trainserved);

//average/max time-in-system

double totaltimeinsys=0.0;

double maxtimeinsys=0.0;

double Time=statisticlist.getLast().getdeparturetime();

int [] histogram=new int [10];//max hog 10 times

for(int i=0;i<10;i++)

{

histogram[i]=0;

}

for(int i=0;i<trainserved;i++)

{

double temp=statisticlist.get(i).getdeparturetime()-statisticlist.get(i).getarrivetime();

totaltimeinsys+=temp;

if(temp>maxtimeinsys)

{

maxtimeinsys=temp;

}

busytime+=statisticlist.get(i).gettotalunloadtime();

histogram[statisticlist.get(i).gethogtimes()]++;

}

System.out.println("Average time-in-system per train: "+totaltimeinsys/trainserved+"h");

if(iftable==true)

{

timetable[tabletimes]=totaltimeinsys/trainserved;

}

System.out.println("Maximum time-in-system per train: "+maxtimeinsys+"h");

if(iffile==false)

{

System.out.println("Dock idle percentage: "+idletime\*100/Time+"%");

}

else//add the time lost about the first train

{

idletime+=statisticlist.getFirst().getarrivetime();

System.out.println("Dock idle percentage: "+idletime\*100/Time+"%");

}

System.out.println("Dock busy percentage: "+busytime\*100/Time+"%");

System.out.println("Dock hogged-out percentage: "+hogtime\*100/Time+"%");

System.out.println("Average time-in-queue over trains: "+timeinqueue/trainserved);

System.out.println("Maximum number of trains in queue: "+queuemaxnum);

System.out.println("Histogram of hogout count per train:");

for(int i=0;i<10;i++)

{

if(histogram[i]!=0)

{

System.out.println("["+i+"]: "+histogram[i]);

}

}

}

}