LUND UNIVERSITY, INSTITUTION FOR ELECTRICAL AND INFORMATION TECHNOLOGY

EITN95 - Simulation
Home assignment 1

Josefine Sandström elt14jsa@student.lu.se
Joel Bill elt14jbi@student.lu.se

May 6, 2018

TASK 1

The mean number of customers in Q2 for the inter arrival times 1, 2 and 5 seconds to Q1 can be seen in the table below, together with the probability that a customer is rejected at Q1.

	Inter arrival times		
	1	2	5
Mean number of customers	10.6	4.3	0.43
Probability of rejection	0.52	0.08	~0.0

TASK 2

When the delay distribution is constant, the mean number of jobs in the buffer is 138.4. When the delay distribution is exponential instead, the mean number of jobs is 117.6. After the priorities are changed so that jobs of type A have the higher priority and the delay distribution is constant again, the mean number of jobs in the buffer is 4.4. The difference is that in the latter case we prioritize the jobs that have a shorter service time. This result in a shorter queue.

TASK 3

The mean number of customers and the mean time a customer spends in the queuing network for the inter arrival times 2, 1.5 and 1.1 can be seen in the table below. There are both the simulated values and the theoretical values that were derived from the equations $N = 2/(x-1)$ and $T = 2x/(x-1)$. Here N is the mean number of customers in the system, x is the mean time between arrivals and T is the mean time a customer spends in the system. As seen in the table, the simulated values are very close to the expected theoretical values.

	Simulated values			Theoretical values		
	Inter arrival times			Inter arrival times		
	2	1.5	1.1	2	1.5	1.1
Mean number of customers	2.0	3.9	20.5	2.0	4.0	20.0
Mean time in system	4.0	5.9	22.5	4.0	6.0	22.0

TASK 4

The transient phase is approximately 80 measurements long in the first question. In the second problem the transient phase is approximately 7 measurements long. In the third problem the transient phase is approximately 207 measurements long.

The length of the 95 % confidence interval was 1.442 when T=4 and M=1000. When T=1 and M=4000 the length of the confidence interval was 1.398. The results are very close to each other. The second case is lower since we have a higher number of measurements. This gives a more accurate result. Finally, the case where T=4 and M=4000 was tested. This led to a confidence interval length of 0.817. It is lower than the first case because of the number of measurements. It is also lower than the second case because of a reduction in correlation between the measurements. The risk of including irregular values is less when T=4 than when T=1.

	x = 100, $\lambda = 8$	x = 10, $\lambda = 80$	x = 200, $\lambda = 4$
Length of transient phase	80	7	207

The different lengths of the transient phase when the service time x and arrival rate λ varies.

	T=1	T=4
M=1000	-	1.442
M=4000	1.398	0.817

TASK 5

Below are three tables, one for each algorithm, that states the mean number of jobs in the system, the mean time a job spends in the system and the measured value of λ in Little's theorem, $E(R) = \lambda \cdot E(T_s)$. The theorem states that the mean number of customers, $E(R)$, in a system is equal to the mean arrival rate, λ, multiplied by the mean time a customer spends in the system, $E(T_s)$. In this case the mean arrival time was 0.12 seconds, which gave us an expected value of $\lambda = 1/0.12 = 8.33$.

Random		
Mean number of jobs in system	Mean time in system	Lambda in Little's theorem
23.21	2.83	8.34

Round Robin		
Mean number of jobs in system	Mean time in system	Lambda in Little's theorem
14.16	1.69	8.33

Shortest Queue		
Mean number of jobs in system	Mean time in system	Lambda in Little's theorem
6.76	0.81	8.33

The table below shows the mean number of jobs in the system with the different arrival times 0.11 s, 0.15 s and 2.0 s for the different algorithms.

	Arrival times (s)		
	0.11	0.15	2.00
Random	46.6	9.4	0.3
Round Robin	27.7	5.9	0.3
Shortest Queue	10.6	3.9	0.3

When jobs arrive to the system at a higher rate, the Shortest Queue algorithm is the best alternative. However, when jobs arrive at a lower rate the three algorithms have similar results when it comes to mean number of jobs in the system.

TASK 6

The average time that the owner can leave the store is 18:07 and the average time for a customer to be in the store is 58 min.

TASK 7

The mean time until the system breaks down is 3.66 time units.

CODE

TASK 1

```
1  public class G{
2          public static final int arrivalTo1 = 1, departureFrom1 = 2,
               departureFrom2 = 3, MEASURE = 4;
3          public static double time = 0;
4  }
```

```
1  import java.util.*;
2  import java.io.*;
3
4  class State{
5          public int noArrivalTo1 = 0, numberInQueue1 = 0,
               numberInQueue2 = 0, accumulated1 = 0, accumulated2 = 0,
               noMeasurements = 0, noRejected = 0;
6          //public double rejectedProb;
7
8          Random slump = new Random();
9          SimpleFileWriter W = new SimpleFileWriter("number.m", false);
10
11         public void TreatEvent(Event x){
12                 switch (x.eventType){
13                         case G.arrivalTo1:{
14                                 noArrivalTo1++;
15                                 if (numberInQueue1 < 10){
16                                         if (numberInQueue1 == 0){
17                                                 EventList.InsertEvent(
                                                       G.departureFrom1, G
                                                       .time - (2.1)*Math.
                                                       log(slump.
                                                       nextDouble()));
18                                         }
19                                         numberInQueue1++;
20
21                                 } else {
22                                         noRejected++;
23                                 }
24                                 EventList.InsertEvent(G.arrivalTo1, G.
                                       time + 5); //interarrival time = 1,
                                        2 or 5
25                         } break;
26                         case G.departureFrom1:{
```

```
27                                      numberInQueue1--;
28                                      if (numberInQueue2 == 0){
29                                              EventList.InsertEvent(G.
                                                    departureFrom2, G.time + 2)
                                                    ;
30                                      }
31                                      numberInQueue2++;
32                                      if (numberInQueue1 > 0){
33                                              EventList.InsertEvent(G.
                                                    departureFrom1, G.time -
                                                    (2.1)*Math.log(slump.
                                                    nextDouble()));
34                                      }
35                              } break;
36                              case G.departureFrom2:{
37                                      numberInQueue2--;
38                                      if (numberInQueue2 > 0){
39                                              EventList.InsertEvent(G.
                                                    departureFrom2, G.time + 2)
                                                    ;
40                                      }
41                              } break;
42                              case G.MEASURE:{
43                                      noMeasurements++;
44                                      accumulated1 = accumulated1 +
                                            numberInQueue1;
45                                      accumulated2 = accumulated2 +
                                            numberInQueue2;
46                                      EventList.InsertEvent(G.MEASURE, G.
                                            time - (5.0)*Math.log(slump.
                                            nextDouble()));
47                                     W.println(String.valueOf(
                                            numberInQueue1));
48                              } break;
49                      }
50              }
51      }
```

```
1   import java.util.*;
2   import java.io.*;
3
4
5   public class Template2006 {
6
```

```
7       public static void main(String[] args) throws IOException {
8           Event actEvent;
9           State actState = new State();
10          new EventList();
11          EventList.InsertEvent(G.arrivalTo1, 0);
12          EventList.InsertEvent(G.MEASURE, 5);
13          while (actState.noMeasurements < 10000){
14                  actEvent = EventList.FetchEvent();
15                  G.time = actEvent.eventTime;
16                  actState.TreatEvent(actEvent);
17          }
18          double rejectedProb = 1.0*actState.noRejected/actState.
                noArrivalTo1;
19          System.out.println("Mean number of customers in Q2: " + 1.0*
                actState.accumulated2/actState.noMeasurements);      //The
                mean nr of customers in queue 2
20          System.out.println(actState.noMeasurements);     //Number of
                measurements
21          System.out.println("Prob rejection: " + rejectedProb);  //the
                probability that a customer is rejected at Q1
22
23          actState.W.close();
24      }
25  }
```

TASK 2

```
1   public class G2{
2           public static final int arrivalA = 1, arrivalB = 2, departureA
                = 3, departureB = 4, MEASURE = 5;
3           public static double time = 0;
4   }
```

```
1   import java.util.*;
2   import java.io.*;
3
4   class State2{
5           Random slump = new Random();
6
7           public int AinQueue = 0, BinQueue = 0, accumulated = 0,
                noMeasurements = 0;
8           public double xA = 0.002, xB = 0.004, lambda = 150, d = 1;
9
10          SimpleFileWriter W = new SimpleFileWriter("number.m", false);
11
```

```
12          public void TreatEvent(Event x){
13                  switch (x.eventType){
14                  case G2.arrivalA:{
15                          if (BinQueue+AinQueue == 0){
16                                  EventList.InsertEvent(G2.departureA,
                                        G2.time + xA);
17                          }
18                          AinQueue++;
19                          EventList.InsertEvent(G2.arrivalA, G2.time -
                                (1.0/150)*Math.log(slump.nextDouble()));
20                  } break;
21                  case G2.arrivalB:{
22                          if (AinQueue+BinQueue == 0){
23                                  EventList.InsertEvent(G2.departureB,
                                        G2.time + xB);
24                          }
25                          BinQueue++;
26                  } break;
27                  case G2.departureA:{
28                          AinQueue--;
29                          EventList.InsertEvent(G2.arrivalB, G2.time + d
                                );
30                          if (AinQueue > 0){
31                                  EventList.InsertEvent(G2.departureA,
                                        G2.time + xA);
32                          } else if (BinQueue > 0){
33                                  EventList.InsertEvent(G2.departureB,
                                        G2.time + xB);
34                          }
35                  } break;
36                  case G2.departureB:{
37                          BinQueue--;
38                          if (AinQueue > 0){
39                                  EventList.InsertEvent(G2.departureA,
                                        G2.time + xA);
40                          } else if (BinQueue > 0){
41                                  EventList.InsertEvent(G2.departureB,
                                        G2.time + xB);
42                          }
43                  } break;
44                  case G2.MEASURE:{
45                          noMeasurements++;
46                          accumulated = accumulated + AinQueue +
                                BinQueue;
```

```
47                          EventList.InsertEvent(G2.MEASURE, G2.time +
                                0.1);
48                          W.println(String.valueOf(accumulated));
49                  } break;
50                  }
51          }
52  }
```

```
1  import java.util.*;
2  import java.io.*;
3
4
5  public class Template2 {
6
7      public static void main(String[] args) throws IOException {
8          //Random slump = new Random();
9          //System.out.println((1.0/150)*Math.log(slump.nextDouble()));
10
11         Event actEvent;
12         State2 actState = new State2();
13         new EventList();
14         EventList.InsertEvent(G2.arrivalA, 0);
15         EventList.InsertEvent(G2.MEASURE, 5);
16         while (actState.noMeasurements < 1000){
17                 actEvent = EventList.FetchEvent();
18                 G2.time = actEvent.eventTime;
19                 actState.TreatEvent(actEvent);
20         }
21
22         System.out.println("The mean number of jobs in the buffer: " +
               1.0*actState.accumulated/actState.noMeasurements);
23         System.out.println(actState.noMeasurements);      //Number of
               measurements
24
25
26         actState.W.close();
27     }
28 }
```

TASK 3

```
1  public class G{
2          public static final int arrivalTo1 = 1, departureFrom1 = 2,
               departureFrom2 = 3, MEASURE = 4;
3          public static double time = 0;
```

```
}

import java.util.*;
import java.io.*;

class State3{
        public int noArrivalTo1 = 0, numberInQueue1 = 0,
            numberInQueue2 = 0, accumulated1 = 0, accumulated2 = 0,
            noMeasurements = 0, noDepartures = 0;
        public double totalTime = 0;
        Random slump = new Random();
        SimpleFileWriter W = new SimpleFileWriter("number.m", false);

        LinkedList<Double> arrivalTimes = new LinkedList<Double>();

        public void TreatEvent(Event x){
                switch (x.eventType){
                case G.arrivalTo1:{
                        arrivalTimes.addLast(G.time);
                        noArrivalTo1++;
                        if (numberInQueue1 == 0){
                                EventList.InsertEvent(G.departureFrom1
                                    , G.time - (1.0)*Math.log(slump.
                                    nextDouble()));
                        }
                        numberInQueue1++;
                        EventList.InsertEvent(G.arrivalTo1, G.time -
                            (2.0)*Math.log(slump.nextDouble())); //
                            Interarrival times = 2, 1,5 or 1,1
                } break;
                case G.departureFrom1:{
                        numberInQueue1--;
                        if (numberInQueue2 == 0){
                                EventList.InsertEvent(G.departureFrom2
                                    , G.time - (1.0)*Math.log(slump.
                                    nextDouble()));
                        }
                        numberInQueue2++;
                        if (numberInQueue1 > 0){
                                EventList.InsertEvent(G.departureFrom1
                                    , G.time - (1.0)*Math.log(slump.
                                    nextDouble()));
                        }
                } break;
```

```
33                  case G.departureFrom2:{
34                          noDepartures++;
35                          totalTime += G.time - arrivalTimes.poll();
36                          numberInQueue2--;
37                          if (numberInQueue2 > 0){
38                                  EventList.InsertEvent(G.departureFrom2
                                        , G.time - (1.0)*Math.log(slump.
                                        nextDouble()));
39                          }
40                  } break;
41                  case G.MEASURE:{
42                          noMeasurements++;
43                          accumulated1 = accumulated1 + numberInQueue1;
44                          accumulated2 = accumulated2 + numberInQueue2;
45                          EventList.InsertEvent(G.MEASURE, G.time -
                                (5.0)*Math.log(slump.nextDouble()));
46                         W.println(String.valueOf(numberInQueue1));
47                  } break;
48                  }
49          }
50  }
```

```
1   import java.util.*;
2   import java.io.*;
3
4
5   public class Template3 {
6
7       public static void main(String[] args) throws IOException {
8
9           Event actEvent;
10          State3 actState = new State3();
11          new EventList();
12          EventList.InsertEvent(G.arrivalTo1, 0);
13          EventList.InsertEvent(G.MEASURE, 5);
14          while (actState.noMeasurements < 10000){
15                  actEvent = EventList.FetchEvent();
16                  G.time = actEvent.eventTime;
17                  actState.TreatEvent(actEvent);
18          }
19          double accumulated = actState.accumulated1 + actState.
                accumulated2;
20          System.out.println("The mean number of customers in the
                queuing network: " + 1.0*accumulated/actState.
```

```
            noMeasurements);
21          System.out.println(actState.noMeasurements);      //Number of
            measurements
22          System.out.println("The mean time a customer spends in the
            system: " + (1.0)*actState.totalTime/actState.noDepartures)
            ;
23
24          actState.W.close();
25      }
26  }
```

TASK 4

```
1   public class G4{
2           public static final int ARRIVAL = 1, DEPARTURE = 2, MEASURE =
                3;
3           public static double time = 0;
4   }
```

```
1   import java.util.*;
2   import java.io.*;
3
4   class State4{
5           public int noBusyServers = 0, noMeasurements = 0, N = 100,
                lambda = 4, sTime = 10, T = 4;
6
7           Random slump = new Random();
8           SimpleFileWriter W = new SimpleFileWriter("number.m", false);
9
10          public void TreatEvent(Event x){
11                  switch (x.eventType){
12                  case G4.ARRIVAL:{
13                          if (noBusyServers < N){
14                                  noBusyServers++;
15                                  EventList.InsertEvent(G4.DEPARTURE, G4
                                        .time + sTime);
16                          }
17                          EventList.InsertEvent(G4.ARRIVAL, G4.time -
                                (1.0/lambda)*Math.log(slump.nextDouble()));
18                  } break;
19                  case G4.DEPARTURE:{
20                          noBusyServers--;
21                  } break;
22                  case G4.MEASURE:{
23                          noMeasurements++;
```

```
                        EventList.InsertEvent(G4.MEASURE, G4.time -
                            (1.0 * T)*Math.log(slump.nextDouble()));
                        W.println(String.valueOf(noBusyServers));
                    } break;
                    }
            }
}
```

```
import java.util.*;
import java.io.*;


public class Template4 {

    public static void main(String[] args) throws IOException {

        Event actEvent;
        State4 actState = new State4();
        new EventList();
        EventList.InsertEvent(G4.ARRIVAL, 0);
        EventList.InsertEvent(G4.MEASURE, 5);
        while (actState.noMeasurements < 4000){
                actEvent = EventList.FetchEvent();
                G4.time = actEvent.eventTime;
                actState.TreatEvent(actEvent);
        }


        System.out.println(actState.noMeasurements);      //Number of
            measurements


        actState.W.close();
    }
}
```

TASK 5

```
public class G5{
        public static final int ARRIVAL = 1, READY = 2, MEASURE = 3;
        public static double time = 0;
}
```

```
import java.util.*;
```

```
2  import java.io.*;
3
4  //It inherits Proc so that we can use time and the signal names
       without dot notation
5
6  class Gen extends Proc{
7
8          //The random number generator is started:
9          Random slump = new Random();
10
11         //There are two parameters:
12         public Proc sendTo;     //Where to send customers
13         public double mu;        //Mean arrival time
14
15         //What to do when a signal arrives
16         public void TreatSignal(Signal x){
17                 switch (x.signalType){
18                         case READY:{
19                                 SignalList.SendSignal(ARRIVAL, sendTo,
                                       time);
20                                 SignalList.SendSignal(READY, this,
                                      time + (2.0*mu)*slump.nextDouble())
                                      ;}
21                                 break;
22                 }
23         }
24 }
```

```
1  import java.util.*;
2
3  public class Dispatch extends Proc{
4
5          //The random number generator is started:
6          Random slump = new Random();
7          public Proc sendTo;     //Where to send customers
8          public QS Q1, Q2, Q3, Q4, Q5;
9          public int temp, noCustomers, noInQ1, noInQ2, noInQ3, noInQ4,
               noInQ5;
10         public List<QS> list = new ArrayList<QS>();
11
12
13
14
15
```

```
16          public void TreatSignal(Signal x) {
17                  switch (x.signalType){
18                  case ARRIVAL:{
19
20                          //Algorithm 1: Random
21                          temp = 1 + slump.nextInt(5);
22                          if (temp == 1){
23                                  sendTo = Q1;
24                          } else if (temp == 2){
25                                  sendTo = Q2;
26                          } else if (temp == 3){
27                                  sendTo = Q3;
28                          } else if (temp == 4){
29                                  sendTo = Q4;
30                          } else if (temp == 5){
31                                  sendTo = Q5;
32                          }
33
34                          //Algorithm 2: Round Robin
35  //                      noCustomers++;
36  //                      temp = noCustomers % 5;
37  //                      if (temp == 1){
38  //                              sendTo = Q1;
39  //                      } else if (temp == 2){
40  //                              sendTo = Q2;
41  //                      } else if (temp == 3){
42  //                              sendTo = Q3;
43  //                      } else if (temp == 4){
44  //                              sendTo = Q4;
45  //                      } else if (temp == 0){
46  //                              sendTo = Q5;
47  //                      }
48
49                          //Algorithm 3: Shortest queue
50  //                      int min = Integer.MAX_VALUE;
51  //                      int minIndex = 0;
52  //                      list.add(Q1);
53  //                      list.add(Q2);
54  //                      list.add(Q3);
55  //                      list.add(Q4);
56  //                      list.add(Q5);
57  //
58  //                      //Find min
59  //                      for(int i = 0; i < 5; i++){
```

```
60 //                                  if(compareTo(min,list.get(i).
   numberInQueue) > 0){
61 //                                          min = list.get(i).
   numberInQueue;
62 //                                          minIndex = i;
63 //                                  }
64 //                          }
65 //
66 //                          List<QS> tempList = new ArrayList<QS>();
67 //                          tempList.add(list.get(minIndex));
68 //
69 //                          //Check if several queues have same
   numberInQueue as the value min
70 //                          for(int i = 0; i < 5; i++){
71 //                                  if(min == list.get(i).numberInQueue &&
    minIndex != i){
72 //                                          tempList.add(list.get(i));
73 //                                  }
74 //                          }
75 //
76 //                          //Choose queue randomly
77 //                          int chosenIndex = slump.nextInt(tempList.size
   ());
78 //
79 //                          sendTo = tempList.get(chosenIndex);
80 //
81
82
83                          SignalList.SendSignal(ARRIVAL, sendTo, time);
84                          break;
85                  }
86
87                  }
88          }
89
90          int compareTo(int x, int y){
91                  if(x > y){
92                          return 1;
93                  } else if (x == y){
94                          return 0;
95                  } else {
96                          return -1;
97                  }
98          }
```

```
99 }
```

```
1  import java.util.*;
2  import java.io.*;
3
4  // This class defines a simple queuing system with one server. It
       inherits Proc so that we can use time and the
5  // signal names without dot notation
6  class QS extends Proc{
7          public int numberInQueue = 0, noReady;
8          public double accumulated, noMeasurements, totalTimeInSystem;
9          public Proc sendTo;
10         Random slump = new Random();
11         LinkedList<Double> arrivalTimes = new LinkedList<Double>();
12
13         public void TreatSignal(Signal x){
14                 switch (x.signalType){
15
16                         case ARRIVAL:{
17                                 arrivalTimes.addLast(time);
18                                 if (numberInQueue == 0){
19                                         SignalList.SendSignal(READY,
                                               this, time - (0.5)*Math.log
                                               (slump.nextDouble()));
20                                 }
21                                 numberInQueue++;
22
23                         } break;
24
25                         case READY:{
26                                 noReady++;
27                                 totalTimeInSystem += time -
                                       arrivalTimes.poll();
28                                 numberInQueue--;
29                                 if (numberInQueue > 0){
30                                         SignalList.SendSignal(READY,
                                               this, time - (0.5)*Math.log
                                               (slump.nextDouble()));
31                                 }
32
33                         } break;
34
35                         case MEASURE:{
36                                 noMeasurements++;
```

```
37                                      accumulated = accumulated +
                                            numberInQueue;
38                                      SignalList.SendSignal(MEASURE, this,
                                            time + 2.0*slump.nextDouble());
39                                  } break;
40                          }
41                  }
42          }
```

```
1   import java.util.*;
2   import java.io.*;
3
4
5   public class Template5 extends G5{
6
7           public static void main(String[] args) throws IOException {
8
9                   // The signal list is started and actSignal is
                        declaree. actSignal is the latest signal that has
                        been fetched from the
10                  // signal list in the main loop below.
11
12                  Signal actSignal;
13                  new SignalList();
14
15                  // Here process instances are created (five queues and
                         one generator) and their parameters are given
                        values.
16
17                  Gen Generator = new Gen();
18                  Dispatch disp = new Dispatch(); //Dispatch instance
                        are created
19                  Generator.sendTo = disp;                    // The
                        generated customers shall be sent to DISPATCH
20
21                  QS Q1 = new QS();
22                  disp.Q1 = Q1;
23
24                  QS Q2 = new QS();
25                  disp.Q2 = Q2;
26
27                  QS Q3 = new QS();
28                  disp.Q3 = Q3;
29
```

```
30              QS Q4 = new QS();
31              disp.Q4 = Q4;
32
33              QS Q5 = new QS();
34              disp.Q5 = Q5;
35
36
37              Generator.mu = 2.0; //mean arrival times = 0.11, 0.15
                    or 2.0
38
39              //To start the simulation the first signals are put in
                     the signal list
40              SignalList.SendSignal(READY,Generator, time);
41              SignalList.SendSignal(MEASURE, Q1, time);
42              SignalList.SendSignal(MEASURE, Q2, time);
43              SignalList.SendSignal(MEASURE, Q3, time);
44              SignalList.SendSignal(MEASURE, Q4, time);
45              SignalList.SendSignal(MEASURE, Q5, time);
46
47
48
49              // This is the main loop
50          while (time < 100000){
51              actSignal = SignalList.FetchSignal();
52              time = actSignal.arrivalTime;
53              actSignal.destination.TreatSignal(actSignal);
54          }
55
56
57          //Finally the result of the simulation is printed below:
58          double meanNoJobs = (Q1.accumulated / Q1.noMeasurements + Q2.
                accumulated / Q2.noMeasurements +
59                          Q3.accumulated / Q3.noMeasurements + Q4.
                                accumulated / Q4.noMeasurements + Q5.
                                accumulated / Q5.noMeasurements);
60          System.out.println("Mean number jobs in system: " + meanNoJobs
                );
61
62          double meanTimeSystem = (Q1.totalTimeInSystem / Q1.noReady +
                Q2.totalTimeInSystem / Q2.noReady + Q3.totalTimeInSystem /
63                          Q3.noReady + Q4.totalTimeInSystem / Q4.noReady
                                + Q5.totalTimeInSystem / Q5.noReady)/5;
64          System.out.println("Mean time in system: " + meanTimeSystem);
65
```

```
66          System.out.println("lambda in Little's theorem: " + meanNoJobs
                / meanTimeSystem);
67
68
69
70          }
71  }
```

TASK 6

```
1   public class G6{
2           public static final int ARRIVAL = 1, DEPARTURE = 2, MEASURE =
                3;
3           public static double time = 0;
4   }
```

```
1   import java.util.*;
2   import java.io.*;
3
4   class State6{
5           public int noMeasurements = 0, lambda = 4, noCustomers = 0,
                noInQueue = 0;
6           public double goHomeTime = 0, totalTimeCustomer = 0;
7
8           Random slump = new Random();
9           LinkedList<Double> arrivalTimes = new LinkedList<Double>();
10          SimpleFileWriter W = new SimpleFileWriter("number.m", false);
11
12          public void TreatEvent(Event x){
13                  switch (x.eventType){
14                  case G6.ARRIVAL:{
15                          if (G6.time <= 8){
16                                  arrivalTimes.addLast(G6.time);
17                                  if(noInQueue == 0){
18                                          EventList.InsertEvent(G6.
                                                DEPARTURE, G6.time + 1.0/6
                                                + (1.0/3-1.0/6)*slump.
                                                nextDouble());
19                                  }
20                                  noInQueue++;
21                                  EventList.InsertEvent(G6.ARRIVAL, G6.
                                        time - (1.0/lambda)*Math.log(slump.
                                        nextDouble()));
22                          } else {
23                                  if (noInQueue == 0){
```

```
                                        EventList.InsertEvent(G6.
                                            MEASURE, G6.time);
                                }
                        }
                } break;
                case G6.DEPARTURE:{
                        totalTimeCustomer += G6.time - arrivalTimes.
                            poll();
                        noCustomers++;
                        noInQueue--;
                        if(G6.time > 8){
                                if(noInQueue == 0){
                                        EventList.InsertEvent(G6.
                                            MEASURE, G6.time);
                                }
                        }
                        if(noInQueue > 0){
                                EventList.InsertEvent(G6.DEPARTURE, G6
                                    .time + 1.0/6 + (1.0/3-1.0/6)*slump
                                    .nextDouble());
                        }
                } break;
                case G6.MEASURE:{
                        noMeasurements++;
                        goHomeTime += G6.time % 24;
                        //System.out.println("totalTimeCustomer: " +
                            totalTimeCustomer);
                        //G6.time = 0;
                        EventList.InsertEvent(G6.ARRIVAL, 0);
                        W.println(String.valueOf(noMeasurements));
                } break;
                }
        }
}
```

```
import java.util.*;
import java.io.*;


public class Template6 {

    public static void main(String[] args) throws IOException {

        Event actEvent;
```

```
10          State6 actState = new State6();
11          new EventList();
12
13          EventList.InsertEvent(G6.ARRIVAL, 0);
14          //EventList.InsertEvent(G6.MEASURE, 5);
15
16          while (actState.noMeasurements < 1000){
17                  actEvent = EventList.FetchEvent();
18                  G6.time = actEvent.eventTime;
19                  actState.TreatEvent(actEvent);
20          }
21
22          System.out.println("The average work time: " + 1.0*actState.
                goHomeTime / 1000);
23          System.out.println("The average time in system " + 1.0*
                actState.totalTimeCustomer/actState.noCustomers);
24          System.out.println(actState.noMeasurements);      //Number of
                measurements
25
26
27          actState.W.close();
28      }
29  }
```

TASK 7

```
1   public class G7{
2           public static final int lifeSpan1 = 1, lifeSpan2 = 2,
                lifeSpan3 = 3, lifeSpan4 = 4, lifeSpan5 = 5, MEASURE = 6;
3           public static double time = 0;
4   }
```

```
1   import java.util.*;
2   import java.io.*;
3
4   class State7{
5           public int noMeasurements = 0;
6           public double totalTime = 0;
7           public boolean alive1 = true, alive2 = true, alive3 = true,
                alive4 = true, alive5 = true;
8
9           Random slump = new Random();
10          SimpleFileWriter W = new SimpleFileWriter("number.m", false);
11
12          public void TreatEvent(Event x){
```

```
13                  switch (x.eventType){
14                  case G7.lifeSpan1:{
15                           alive1 = false;
16                           alive2 = false;
17                           alive5 = false;
18                           if(!(alive1 || alive2 || alive3 || alive4 ||
                                alive5)){
19                                EventList.InsertEvent(G7.MEASURE, G7.
                                    time);
20                           }
21                  } break;
22                  case G7.lifeSpan2:{
23                          alive2 = false;
24                          if(!(alive1 || alive2 || alive3 || alive4 ||
                               alive5)){
25                                EventList.InsertEvent(G7.MEASURE, G7.
                                    time);
26                           }
27                  } break;
28                  case G7.lifeSpan3:{
29                           alive3 = false;
30                           alive4 = false;
31                           if(!(alive1 || alive2 || alive3 || alive4 ||
                                alive5)){
32                                EventList.InsertEvent(G7.MEASURE, G7.
                                    time);
33                           }
34                  } break;
35                  case G7.lifeSpan4:{
36                           alive4 = false;
37                           if(!(alive1 || alive2 || alive3 || alive4 ||
                                alive5)){
38                                EventList.InsertEvent(G7.MEASURE, G7.
                                    time);
39                           }
40                  } break;
41                  case G7.lifeSpan5:{
42                           alive5 = false;
43                           if(!(alive1 || alive2 || alive3 || alive4 ||
                                alive5)){
44                                EventList.InsertEvent(G7.MEASURE, G7.
                                    time);
45                           }
46                  } break;
```

```
47                  case G7.MEASURE:{
48                          noMeasurements++;
49                          totalTime += G7.time;
50                          alive1 = alive2 = alive3 = alive4 = alive5 =
                                true;
51                          EventList.InsertEvent(G7.lifeSpan1, 1 +
                                (5.0-1.0)*slump.nextDouble());
52                  EventList.InsertEvent(G7.lifeSpan2, 1 + (5.0-1.0)*
                        slump.nextDouble());
53                  EventList.InsertEvent(G7.lifeSpan3, 1 + (5.0-1.0)*
                        slump.nextDouble());
54                  EventList.InsertEvent(G7.lifeSpan4, 1 + (5.0-1.0)*
                        slump.nextDouble());
55                  EventList.InsertEvent(G7.lifeSpan5, 1 + (5.0-1.0)*
                        slump.nextDouble());
56                          W.println(String.valueOf(noMeasurements));
57                  } break;
58                  }
59          }
60  }
```

```
1   import java.util.*;
2   import java.io.*;
3
4
5   public class Template7 {
6
7       public static void main(String[] args) throws IOException {
8           Random slump = new Random();
9
10          Event actEvent;
11          State7 actState = new State7();
12          new EventList();
13
14          EventList.InsertEvent(G7.lifeSpan1, 1 + (5.0-1.0)*slump.
                nextDouble());
15          EventList.InsertEvent(G7.lifeSpan2, 1 + (5.0-1.0)*slump.
                nextDouble());
16          EventList.InsertEvent(G7.lifeSpan3, 1 + (5.0-1.0)*slump.
                nextDouble());
17          EventList.InsertEvent(G7.lifeSpan4, 1 + (5.0-1.0)*slump.
                nextDouble());
18          EventList.InsertEvent(G7.lifeSpan5, 1 + (5.0-1.0)*slump.
                nextDouble());
```

```
19
20
21          while (actState.noMeasurements < 1000){
22                  actEvent = EventList.FetchEvent();
23                  G7.time = actEvent.eventTime;
24                  actState.TreatEvent(actEvent);
25          }
26
27          System.out.println("The mean time until system breaks down: "
                + 1.0*actState.totalTime / 1000);
28          System.out.println(actState.noMeasurements);      //Number of
                measurements
29
30
31          actState.W.close();
32      }
33  }
```