Chapter 3 & 4
General Principles & Simulation Software

Banks, Carson, Nelson & Nicol

Discrete-Event System Simulation

Major Concepts

- ***System:*** A collection of entities that interact together over time to accomplish one or more goals.
- ***Model:*** An abstract representation of a system, usually containing structural, logical, or mathematical relationships that describe a system in terms of state, entities and their attributes, sets, processes, events, activities, and delays.
- ***System state:*** A collection of variables that contain all the information necessary to describe the system at any time.
- ***Entity:*** Any object or component in the system that requires explicit representation in the model (e.g., a server, a customer, a machine).
- ***Attributes:*** The properties of a given entity (e.g., the priority of a waiting customer, the routing of a job through a job shop).
- ***List:*** A collection of (permanently or temporarily) associated entities, ordered in some logical fashion (such as all customers currently in a waiting line, ordered by "first come, first served," or by priority).

Major Concepts (continued ..)

- ***Event:*** An instantaneous occurrence that changes the state of a system (such as an arrival of a new customer).
- ***Event notice:*** A record of an event to occur at the current or some future time, along with any associated data necessary to execute the event; at a minimum, the record includes the event type and the event time.
- ***Event list:*** A list of event notices for future events, ordered by time of occurrence; also known as the future event list (FEL).
- ***Activity:*** A duration of time of specified length (e.g., a service time or inter-arrival time), which is known when it begins (although it may be defined in terms of a statistical distribution).
- ***Delay:*** A duration of time of unspecified indefinite length, which is not known until it ends (e.g., a customer's delay in a last-in-first-out waiting line which, when it begins, depends on future arrivals).
- ***Clock:*** A variable representing simulated time

Event Scheduling / Time Advance Algorithm

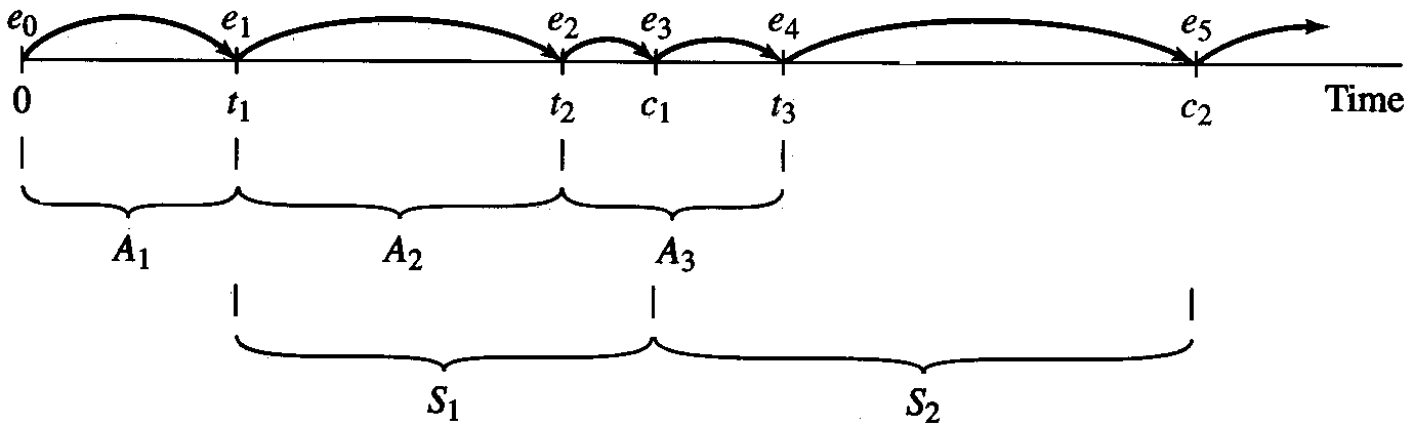
- The mechanism for advancing simulation time and guaranteeing that all events occur in **correct chronological order** is based on the ***Future Event List (FEL).***
- This list contains all event notices for events that have been scheduled to occur at a future time.
- Scheduling a future event means that, at the instant an activity begins, its duration is computed or drawn as a sample from a statistical distribution; and that the end-activity event, together with its event time, is placed on the future event list.
- In the real world, most future events are not scheduled but merely happen such as random breakdowns or random arrivals.
- In the model, such random events are represented by the end of some activity, which in turn is represented by a statistical distribution.
- At any given time *t*, the FEL contains all previously scheduled future events and their associated event times (called *t1, t2, . ..*).
- **The FEL is ordered by event time**, meaning that the events are arranged chronologically, that is, the event times satisfy $t < t1 \leq t2 \leq t3 \leq \ldots \leq tn$.

Time Advance (continued ..)

- Time *t* is the value of ***CLOCK***, the current value of simulated time.
- The event associated with time *t1* is called the **imminent event**; that is, it is the next event that will occur.
- After the system snapshot at simulation time *CLOCK = t* has been updated, the *CLOCK* is advanced to simulation time *CLOCK = t1*, the imminent event notice is removed from the FEL, and the event is executed.
- Execution of the imminent event means that a new system snapshot for time *t1* is created, one based on the old snapshot at time *t* and the nature of the imminent event.
- At time *t1*, new future events may or might not be generated, but if any are, they are scheduled by creating event notices and putting them into their proper position on the FEL.
- After the new system snapshot for time *t1* has been updated, the clock is advanced to the time of the new imminent event and that event is executed. This process repeats until the simulation is over.

Single Server Queue Example

- Next-event time advance for the single-server queue
  - t_i = time of arrival of ith customer ($t_0 = 0$)
  - $A_i = t_i - t_{i-1}$ = interarrival time between (i-1)st and ith customers (a random variable from some probability distribution)
  - S_i = service-time requirement of ith customer (another random variable)
  - D_i = delay in queue of ith customer
  - $c_i = t_i + D_i + S_i$ = time ith customer completes service and departs
  - e_j = time of occurrence of the jth event (of any type), j = 1, 2, 3, …



List Processing

- The length and contents of the FEL are constantly changing as the simulation progresses, and thus its efficient management in a computerized simulation will have a major impact on the efficiency of the computer program representing the model.
- The management of a list is called ***list processing***.
- The major list processing operations performed on a FEL are removal of the imminent event, addition of a new event to the list, and occasionally removal of some event (called cancellation of an event).

Operations in Lists

- Removing a record from the top of the list
- Removing a record from any location on the list
- Adding an entity record to the top or bottom of ths list
- Adding a record at any arbitrary position in the list

An entity along with its attributes or an event notice will be referred to as record

Stopping the Simulation

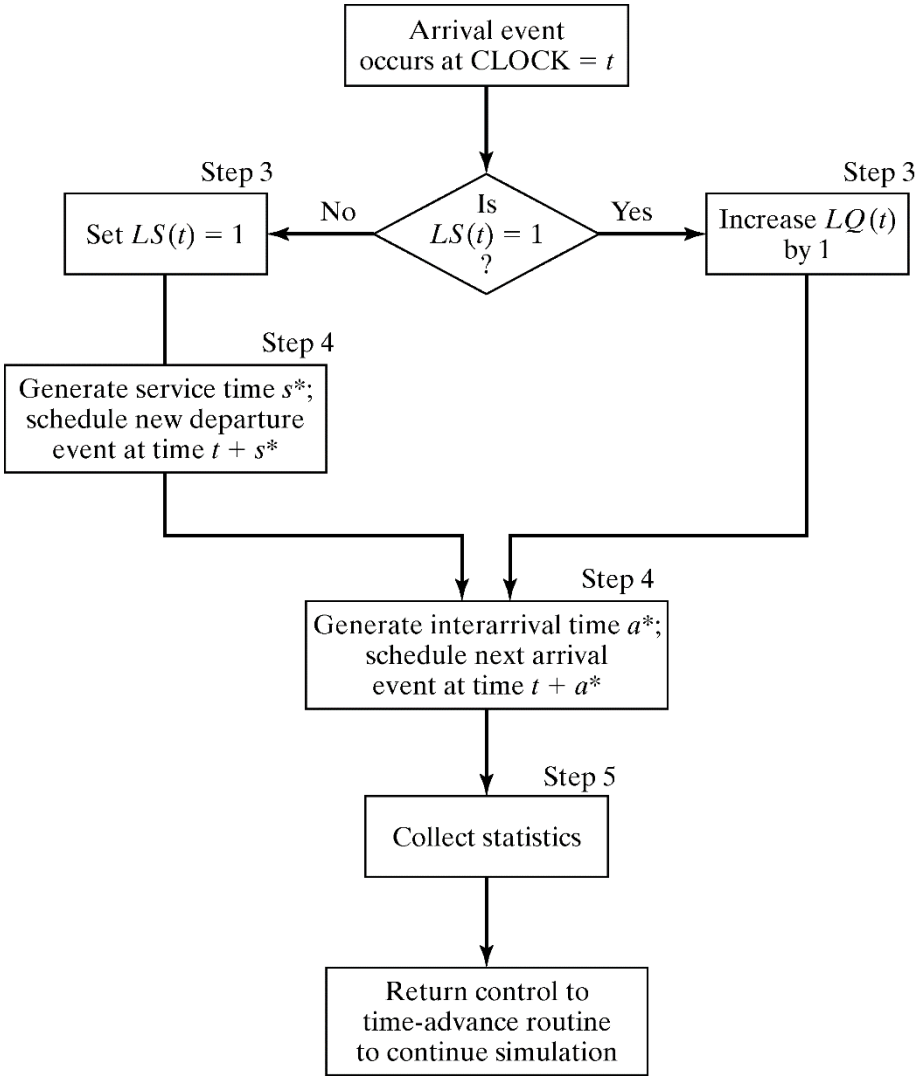
- Every simulation must have a **stopping event**, here called *E*, which defines **how long the simulation will run**. There are generally two ways to stop a simulation:

1. At time 0, schedule a stop simulation event at a specified future time T_E. Thus, before simulating, it is known that the simulation will run over the time interval *[0, T_E].* Example: Simulate a job shop for T_E = *40* hours.
2. Run length T_E is determined by the simulation itself. Generally, T_E is the time of occurrence of some specified event *E*. Examples: T_E is the time of the 100th service completion at a certain service center; T_E is the time of breakdown of a complex system; T_E is the time of abort or total bag drops (whichever occurs first) in a food relief simulation. T_E is the time at which a distribution center ships the last carton in a day's orders

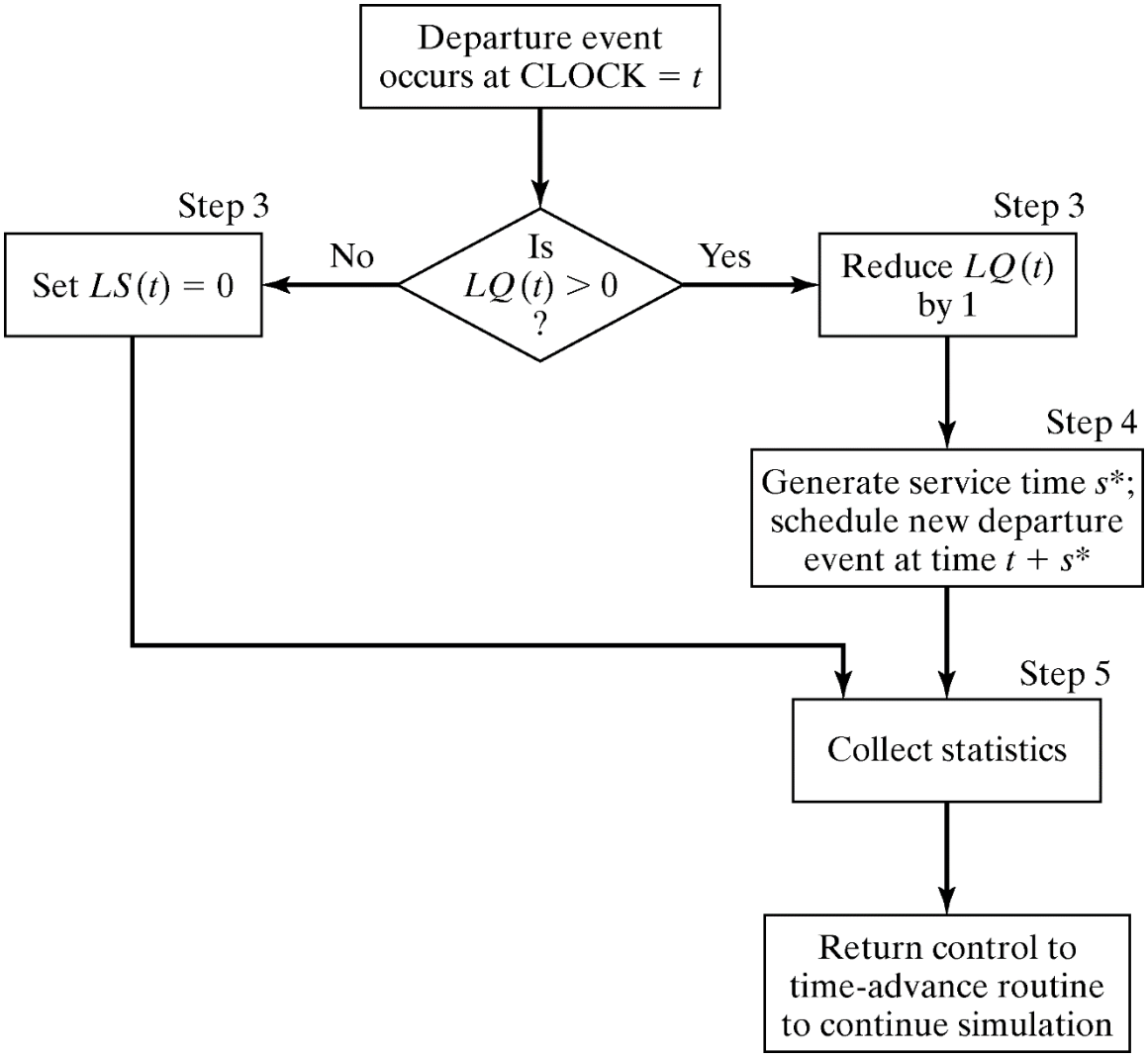
Grocery Store Using Event Scheduling

- **System State (LQ(t), LS(t))** – *LQ(t)* is the number of customers in waiting line and *LS(t)* is the number being served (0 or 1) at time *t*
- **Events:** Arrival (A), Departure (D), Stopping (E)
- **Event notice:** (A, t) – arrival; (D,t) – departure; (E,60) – stop simulation at 60
- **Activities** – Interarrival Time, Service Time
- **Delay** – Customer time spent in waiting line

Grocery Store Arrival Event



Grocery Store Departure Event



World Views

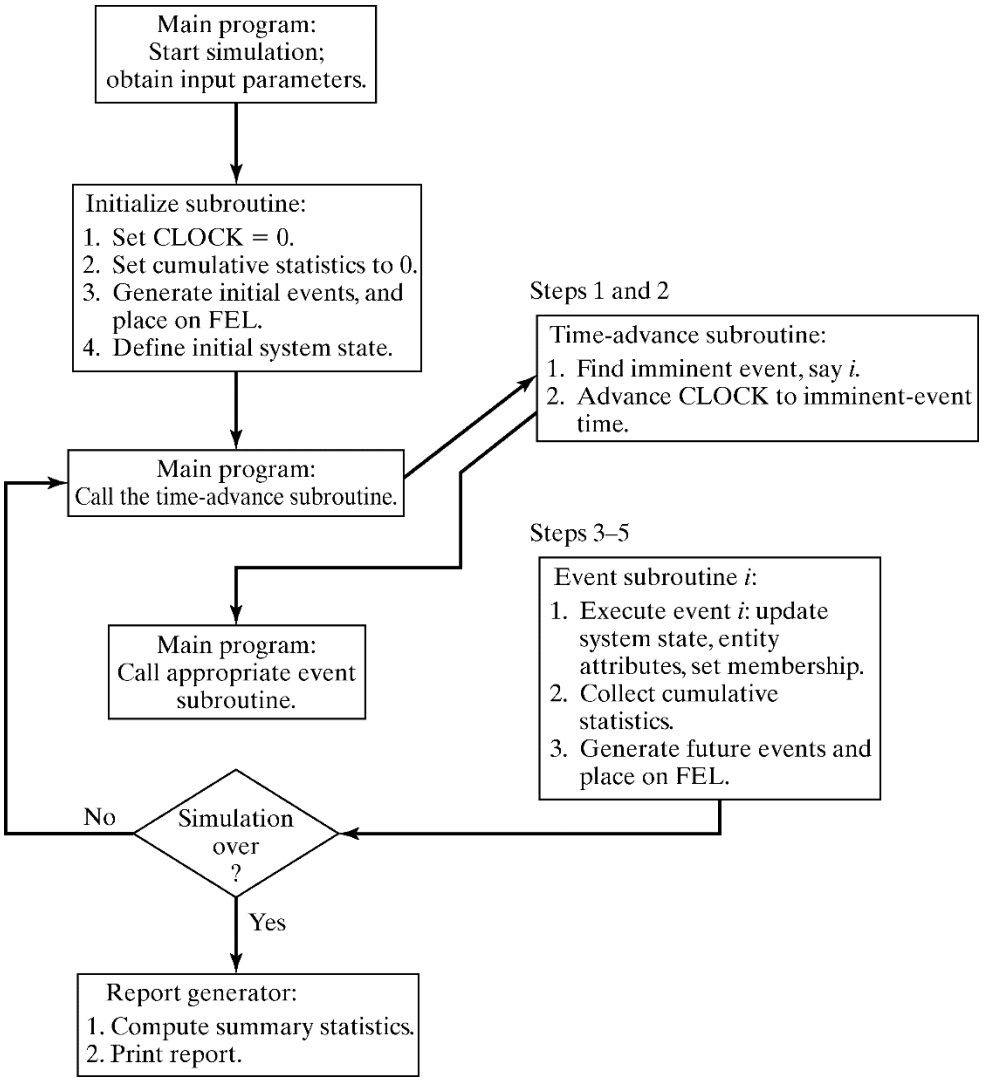
- Models are developed with a world view or orientation
- Most popular are:
  - Event Scheduling World View
    - Model focuses on events and their effect on system state
  - Process Interaction World View
    - Model focuses on objects and their life cycle as they flow through the system
  - Activity Scanning World View
    - Model focuses on activities of a model and conditions that allow an activity to begin

Simulation Software

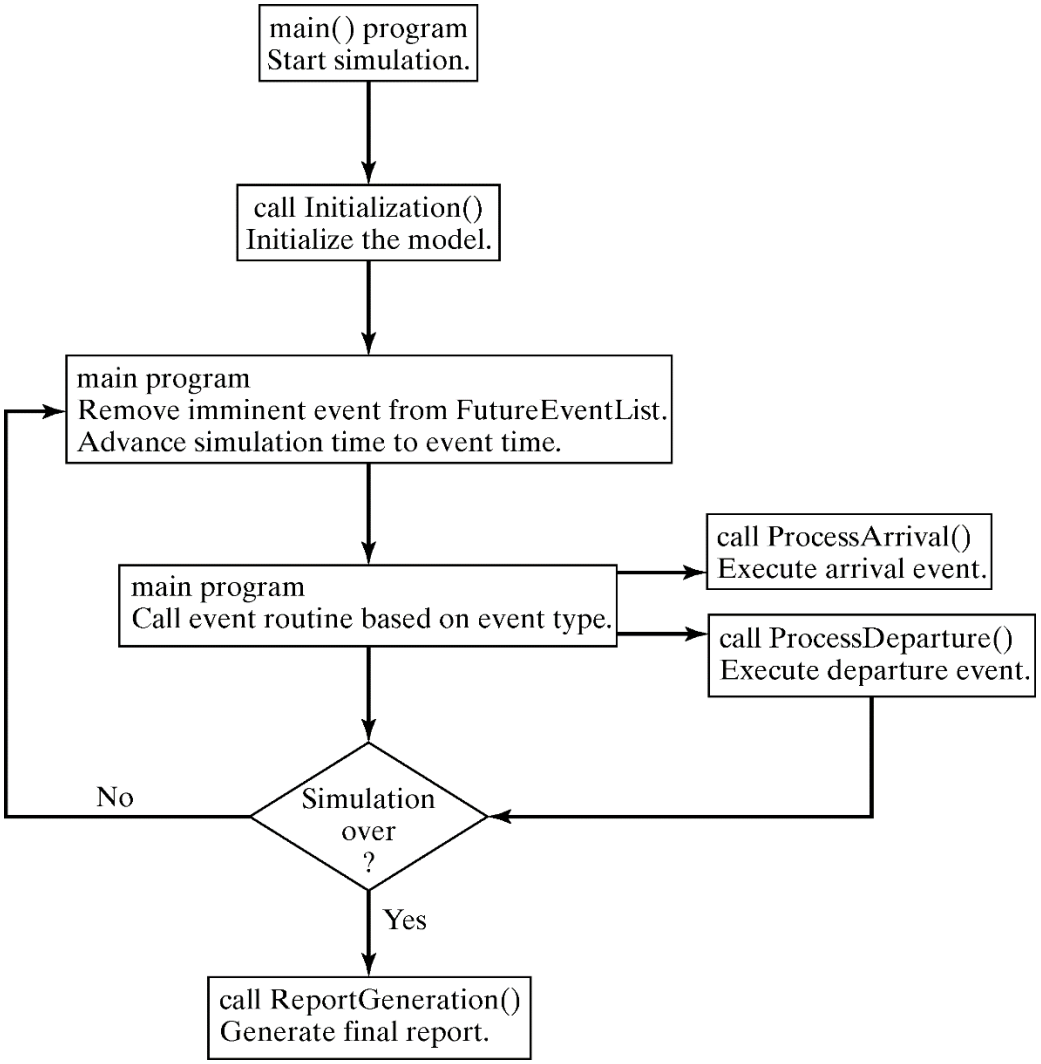
Features

- ***Model Building:*** World Views, Input-data analysis capability (e.g, raw data), Graphical model building, Simulation language and its capabilities – Application Programming Interfaces (APIs), Random variates, any specialized components and templates for specific task, interface with general programming languages
- ***Run Time Environment:*** Execution speed, Model size limits (# of variables, attributes etc), Interactive debugger, Model status display and statistics, Run time licenses
- ***Animation and Layout features:*** Type of animation (true to scale or iconic), import drawing and object files, dimension (2D or 3D), Movement (status indicators), Quality of movement, Library of common objects, Navigation, Views, Hardware requirements, backup features
- ***Output:*** Scenario manager, Runtime manager, **Warmup capability**, Report generation, Statistical analysis, database maintenance
- ***Support:*** Vendor support, Training, Documentation, Upgrades and maintenance, Track record

Structure of an event-scheduling simulation



Single Server Queue in Java



Main Program

```
public static void main(String argv[]) {

 MeanInterArrivalTime = 4.5; MeanServiceTime = 3.2;
 SIGMA                = 0.6; TotalCustomers  = 1000;
 long seed            = Long.parseLong(argv[0]);

 stream = new Random(seed);          // initialize rng stream
 FutureEventList = new EventList();
 Customers = new Queue();

 Initialization();

 // Loop until first "TotalCustomers" have departed
 while(NumberOfDepartures < TotalCustomers ) {
   Event evt = (Event)FutureEventList.getMin();  // get imminent event
   FutureEventList.dequeue();                    // be rid of it
   Clock = evt.get_time();                       // advance simulation time
   if( evt.get_type() == arrival ) ProcessArrival(evt);
   else  ProcessDeparture(evt);
   }
 ReportGeneration();
}
```

Arrival Process

```
public static void ProcessArrival(Event evt) {
  Customers.enqueue(evt);
  QueueLength++;
  // if the server is idle, fetch the event, do statistics
  // and put into service
  if( NumberInService == 0) ScheduleDeparture();
  else TotalBusy += (Clock - LastEventTime);  // server is busy

  // adjust max queue length statistics
  if (MaxQueueLength < QueueLength) MaxQueueLength = QueueLength;

  // schedule the next arrival
  Event next_arrival = new Event(arrival, Clock+exponential(stream,
      MeanInterArrivalTime));
  FutureEventList.enqueue( next_arrival );
  LastEventTime = Clock;
}
```

Departure Process

```
public static void ScheduleDeparture() {
  double ServiceTime;
  // get the job at the head of the queue
  while (( ServiceTime = normal(stream, MeanServiceTime, SIGMA)) < 0 );
  Event depart = new Event(departure,Clock+ServiceTime);
  FutureEventList.enqueue( depart );
  NumberInService = 1;
  QueueLength--;
 }


public static void ProcessDeparture(Event e) {
 // get the customer description
 Event finished = (Event) Customers.dequeue();
 // if there are customers in the queue then schedule
 // the departure of the next one
  if( QueueLength > 0 ) ScheduleDeparture();
  else NumberInService = 0;
  // measure the response time and add to the sum
  double response = (Clock - finished.get_time());
  SumResponseTime += response;
  if( response > 4.0 ) LongService++; // record long service
  TotalBusy += (Clock - LastEventTime );
  NumberOfDepartures++;
  LastEventTime = Clock;
 }
```

Report Generation

```
public static void ReportGeneration() {
double RHO   = TotalBusy/Clock;
double AVGR  = SumResponseTime/TotalCustomers;
double PC4   = ((double)LongService)/TotalCustomers;


System.out.println( "SINGLE SERVER QUEUE SIMULATION - GROCERY STORE CHECKOUT COUNTER ");
System.out.println( "\tMEAN INTERARRIVAL TIME                    "
      + MeanInterArrivalTime );
System.out.println( "\tMEAN SERVICE TIME                         "
      + MeanServiceTime );
System.out.println( "\tSTANDARD DEVIATION OF SERVICE TIMES          " + SIGMA );
System.out.println( "\tNUMBER OF CUSTOMERS SERVED                  " + TotalCustomers );
System.out.println();
System.out.println( "\tSERVER UTILIZATION                        " + RHO );
System.out.println( "\tMAXIMUM LINE LENGTH                         " + MaxQueueLength );
System.out.println( "\tAVERAGE RESPONSE TIME                       " + AVGR + "  MINUTES" );
System.out.println( "\tPROPORTION WHO SPEND FOUR ");
System.out.println( "\t MINUTES OR MORE IN SYSTEM                  " + PC4 );
System.out.println( "\tSIMULATION RUNLENGTH                        " + Clock + " MINUTES" );
System.out.println( "\tNUMBER OF DEPARTURES                        " + TotalCustomers );
}
```

Random Variates

```
public static double exponential(Random rng, double mean) {
 return -mean*Math.log( rng.nextDouble() );
}

public static double SaveNormal;
public static int  NumNormals = 0;
public static final double  PI = 3.1415927 ;

public static double normal(Random rng, double mean, double sigma) {
      double ReturnNormal;
      // should we generate two normals?
      if(NumNormals == 0 ) {
        double r1 = rng.nextDouble();
        double r2 = rng.nextDouble();
        ReturnNormal = Math.sqrt(-2*Math.log(r1))*Math.cos(2*PI*r2);
        SaveNormal   = Math.sqrt(-2*Math.log(r1))*Math.sin(2*PI*r2);
        NumNormals = 1;
      } else {
        NumNormals = 0;
        ReturnNormal = SaveNormal;
      }
      return ReturnNormal*sigma + mean ;
 }
```