

# **Chapter 9**

# **Simulation Software**

# Simulation Software

- Simulation software is a program that allows the user to **observe an operation through simulation** without actually performing that operation
- Simulation software helps us predict the behavior of a system.
- We can use simulation software to evaluate a new design, diagnose problems with an existing design, and test a system under conditions that are hard to reproduce, such as a satellite in outer space.
- Simulation software also includes visualization tools, such as data displays and 3D animation, to help monitor the simulation as it runs.
- Engineers and scientists use simulation software for a variety of reasons:
  - Creating and simulating models is less expensive than building and testing hardware prototypes.
  - We can use simulation software to test different designs before building one in hardware.
  - We can connect simulation software to hardware to test the integration of the full design.

# History of Simulation Software

## □ 1955 - 1960 The Period of Search

- Search for unifying concepts and the development of reusable routines to facilitate simulation.
- Mostly conducted in FORTRAN

## □ 1961 - 1965 The Advent

- The Simulation Programming Language in use today appeared in this period.
- The first process interaction SPL(Simulation Programming Language), GPSS was developed at IBM.
- GPSS got popularity due to easy in use.

## □ 1966 - 1970 The Formative Period

- Concepts were reviewed and refined to promote a more consistent representation of each language's worldview.
- In this phase due to rapid hardware advancements and user demands GPSS was forced to undergo major revision.

## □ **1971 - 1978 The Expansion Period**

- Major advances in GPSS came from outside IBM
- Attempts were made to simplify modeling process.
- GPSS/NORDEN, a pioneering effort that offered an interactive, visual online environment
- GPSS/H(1997): For IBM mainframes, later for microcomputers and PC.
- GASP-IV(1971): It uses state events in addition to time events.

## □ **1979 - 1986 The Period of Consolidation and Regeneration**

- Beginnings of PSLs written for, or adapted to, desktop computers and microcomputers.
- Two major descendants of GASP appeared: SLAM II and SIMAN(provide multiple modeling perspectives and combined modeling capabilities).

## □ 1987 – Now The Period of Integrated Environments

- Growth of SPLs on the personal computer and the emergence of simulation environments with graphical user interfaces, animation and other visualization tools.
- Many of these environment also contain input and output data analyzer.
- Recent advancements have been made in web-based simulation.

□ Three types of software for simulation models developments:

1. General-purpose programming languages, e.g., Java, C.
  - a. Not specifically designed for use in simulation.
  - b. Simulation libraries, e.g., SSF, are sometimes available for standardized simulation functionality.
  - c. Helps to understand the basic concepts and algorithms.
2. Simulation programming languages, e.g., GPSS/HTM, SIMAN V® and SLAM II®.
  - a. Designed specifically for simulation of certain systems, e.g. queuing systems.
3. Simulation environment, e.g., Arena, AutoMod.
  - a. Output analyzer is an important component, e.g. experimental design, statistical analysis.
  - b. Many packages offer optimization tools as well.

# Characteristics of Simulation Software

□ Common characteristics:

- Graphical user interface, animation. For animation, some emphasize scale drawings in 2-D or 3-D; others emphasize iconic-type animation.
- Automatically collected outputs.
- Most provide statistical analyses, e.g., confidence intervals.

## **Advice for Selecting Simulation Software**

- Following points are advised to be considered while selecting simulation software:
- 1. The language chosen should be ease of learning and using.
- 2. The accuracy should be high.
- 3. Execution speed is important.
- 4. It should have vendor support, and applicability to our applications
- 5. Beware of advertising claims and demonstrations.
- 6. Should provide Model status and statistics. Should provide standardized report and statistical analysis.
- 7. Tutorials and documentations should be available for the simulation software.
- 8. Runtime Environment: It is the feature that determines how the model acts during the simulation run. It includes execution speed, model size, interactive debugger, model statistics and so on.



# Simulation In Java

- Java is widely used programming language that has been used extensively in simulation.
- It does not provide any modules directly aimed for simulation system.
- There are runtime libraries in java that provides random number generators.
- The components that all the simulation models written in JAVA are as follows:
  1. Clock : It is a variable that defines the simulated time.
  2. Initialization Method : It is a method to define the system state at initial time.
  3. Min-time event method : It is a method that identifies the imminent (about to happen) event.
  4. Event method: It is a method for each event that update the system state when it occurs.

5. Random variate generator : It is a method to generate random samples from the desired probability distributions.
6. Main program : It is the core of the simulation system that controls the overall event scheduling algorithms.
7. Report generator : It is a method that summarizes the collected statistics to give the report at the end of the simulation.

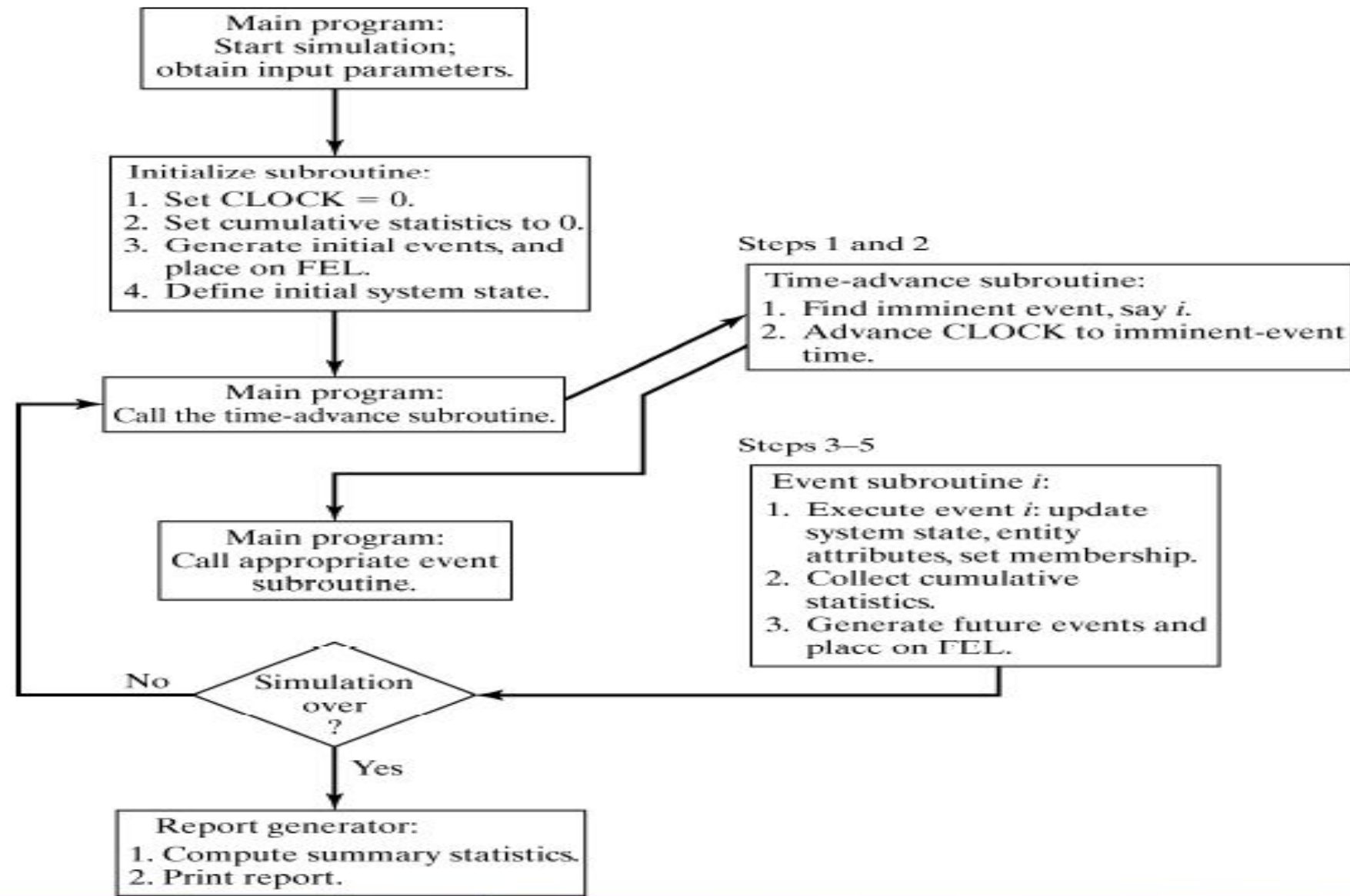


Figure: Overall Structure of Java Simulation

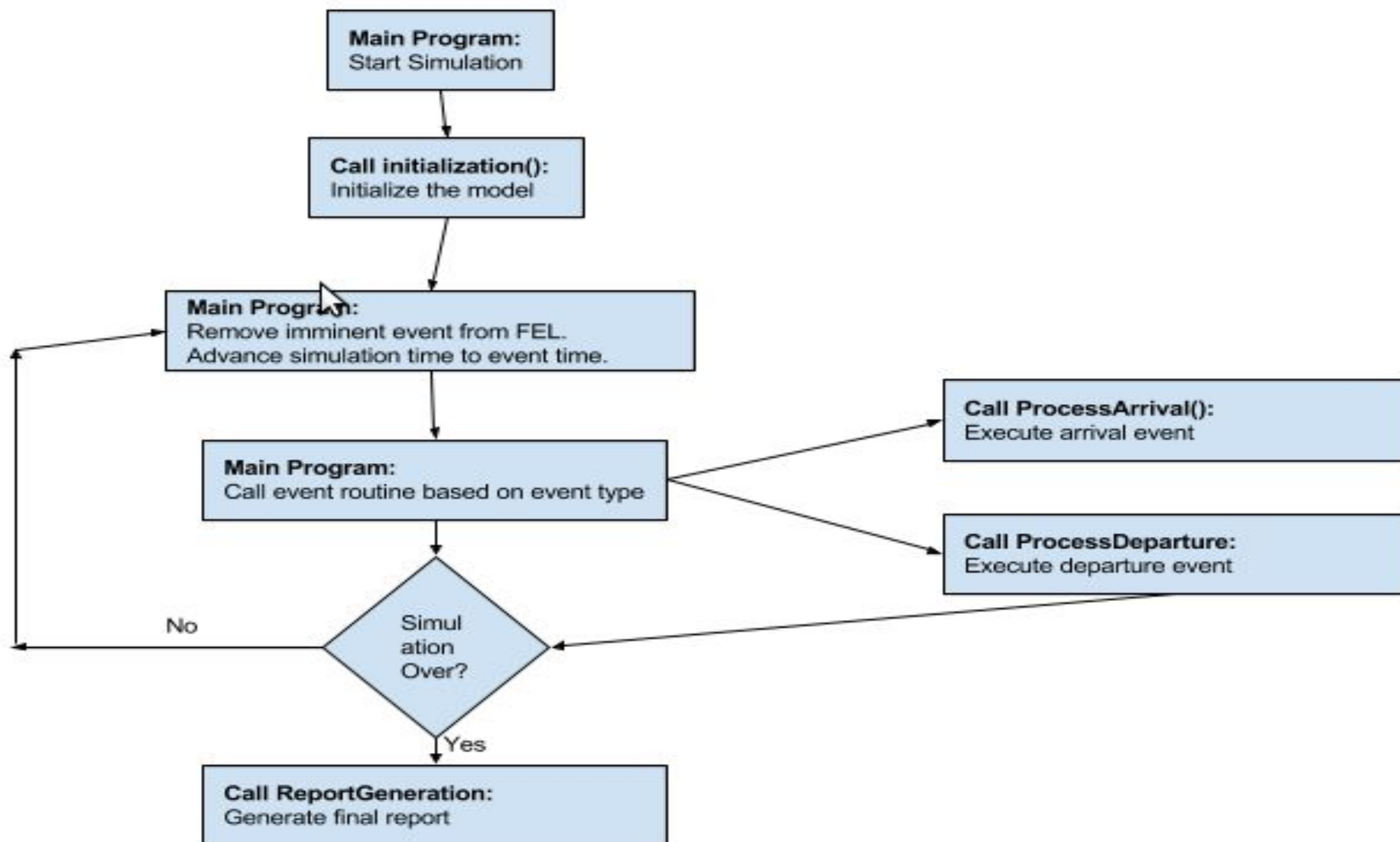
# Explanation of Flowchart

1. Simulation begins by setting clock to zero, initializing cumulative statistic to zero, generating any initial events and placing them in the Future Event List (FEL).
2. The simulation program then cycles repeatedly passing the current least time event to approximate event methods until the simulation is over.
3. At each step, clock is advanced to the time of the imminent event after finding the imminent event but before calling the event method.
4. The appropriate event method is called to execute imminent event, update cumulative statistics and generate future events.
5. All actions in an event method takes place at one instant of the simulated time.
6. When the simulation is over, the control passes to the report generator.

## Example:

### Single Server Queue Simulation

- Consider a grocery checkout counter. The simulation will run until 1000 customers have been serviced. It is assumed that the inter-arrival times are exponentially distributed with mean 4.5 minutes and the service times are normally distributed with mean of 3.2 minutes and standard deviation of 0.6 minutes. When cashier is busy, a queue forms with no customers turned away.
- 1. Class Event represents an event that stores code for arrival or departure and the event time stamp.
- 2. It consists of associated methods for creating an event and accessing its data.
- 3. It also has method compareTo that compares the event with another and reports whether the first event should be considered smaller, equal or greater than the argument event.



## Variables Used

- System state: QueueLength ,NumberInService
- Entity attributes and set: Customers (FCFS queue of customers)
- Future event List: FutureEventList
- Activity durations: MeanInterArrivalTime, MeanServiceTime
- Input parameters: MeanInterArrivalTime, MeanServiceTime, SIGMA (standard deviation), TotalCustomers (The stopping criterion)
- Simulation variables: Clock
- Statistical accumulators: LastEventTime ,TotalBusy, Max QueueLength, SumResponseTime, NumberOfDepartures ,LongService (who spends 4 or more minutes)
- Summary statistics:  $RHO = \text{BusyTime} / \text{Clock}$  Proportion of time server is busy AVGR average response time ,PC4 proportion of customers who spent 4 or more minutes.

## □ Functions Used:

1. `exponential(mu)`
2. `normal(xmu, Sigma)`

## □ Methods Used:

1. `Initialization()`
2. `ProcessArrival()`
3. `ProcessDeparture()`
4. `ReportGeneration()`



# Single Server Queue Example

```
class Sim {  
    public static double Clock, MeanInterArrivalTime, MeanServiceTime,  
        SIGMA, LastEventTime, TotalBusy, MaxQueueLength, SumResponseTime;  
    public static long NumberOfCustomers, QueueLength, NumberInService,  
        TotalCustomers, NumberOfDepartures, LongService;  
    public final static int arrival = 1;  
    public final static int departure = 2;  
    public static EventList FutureEventList;  
    public static Queue Customers;  
    public static Random stream;  
}
```

```
public static void main(String args[]) {  
    MeanInterArrivalTime = 4.5; MeanServiceTime = 3.2;  
    SIGMA = 0.6; TotalCustomers = 1000; long seed = 1000;  
    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();  
}  
}
```

# Initialization Method

```
public static void Initialization() {  
    Clock = 0.0;  
    QueueLength = 0;  
    NumberInService = 0;  
    LastEventTime = 0.0;  
    TotalBusy = 0 ;  
    MaxQueueLength = 0;  
    SumResponseTime = 0;  
    NumberOfDepartures = 0;  
    LongService = 0;  
    // create first arrival event  
    Event evt = new Event(arrival, exponential( stream, MeanInterArrivalTime));  
    FutureEventList.enqueue( evt );  
}
```

# Arrival Event Method

Steps involved:

- Update server status
- Collect statistics
- Schedule next arrival

```
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 );
    next_LastEventTime = Clock;
}
```

## Schedule Departure Method

```
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--;  
}
```

## Process Departure Method

```
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 Generator Method

```
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( "\t MEAN INTERARRIVAL TIME "+ MeanInterArrivalTime  
);  
    System.out.println( "\t MEAN SERVICE TIME “ + MeanServiceTime );  
    System.out.println( "\t STANDARD DEVIATION OF SERVICE TIMES”+  
SIGMA );  
}
```



```
System.out.println( "\t NUMBER OF CUSTOMERS SERVED" + TotalCustomers );
System.out.println( "\t SERVER UTILIZATION" + RHO );
System.out.println( "\t MAXIMUM LINE LENGTH" + MaxQueueLength );
System.out.println( "\t AVERAGE RESPONSE TIME" + AVGR + " MINUTES" );
System.out.println( "\t PROPORTION WHO SPEND FOUR ");
System.out.println( "\t MINUTES OR MORE IN SYSTEM " + PC4 );
System.out.println( "\t SIMULATION RUNLENGTH" + Clock + " MINUTES" );
System.out.println( "\t NUMBER OF DEPARTURES" + TotalCustomers );
}
```

# Methods to generate exponential and normal 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(-*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 ;  
}
```

# Simulation In GPSS

- GPSS (General Purpose Simulation System) is a highly structured and special purpose simulation language based on process interaction approach and oriented toward queuing systems.
- The system being simulated is described by the block diagram using various GPSS blocks. Blocks represents events, delays and other actions that affect transaction flow
- Provides a convenient way to describe the system with over 40 standard blocks.
- GPSS model is developed by converting the block diagram into block statements and adding the control statements.
- The 1st version was released by IBM in 1961.
- GPSS/H is the most widely used version today.
  - Released in 1977
  - Flexible yet powerful.
  - The animator is Proof Animation™.

# Transaction in GPSS

- A process that represents the real-world system we are modeling.
- Transaction is executed by moving from block to block.
- Each transaction in the model is contained in exactly one block, but one block may contain many transactions.

# BLOCKS IN GPSS

## GENERATE Block

□ It generates the transaction.

□ GENERATE A,B,C,D,E means:

A: Mean interval between generation of two transactions

B: Half width of uniform distribution or function modifier used to generate random interval between generation of transactions

C: Delay starting time

D: Limit on total transactions to be created

E: Priority level

□ GENERATE 300 means interval times = 300

□ If B does not specify a function, both A and B are evaluated numerically and a random number between A-B and A+B is used as the time increment.

□ GENERATE 300,100 means interval times = [200,400]

## **ASSIGN Block**

- Used to place or modify a value in a Transaction Parameter(Local Variable).
- If no such Parameter exists, it is created.

## **SAVEVALUE Block**

- Changes the value of a **Savevalue Entity**.
- SAVEVALUE A,B    A: Savevalue Entity Number  
                          B: Value to be stored

## **TERMINATE Block**

- Destroys the active transaction.
- TERMINATE means transaction ends
- TERMINATE 1 means simulation ends

## **ADVANCE Block**

- Delays the progress of a transactions for a specified amount of simulated time.
- ADVANCE 100, 50



# Using Facilities

- GPSS provides the facility modeling concept to represent limited availability of a service.
- **A facility is a resource that can be used by only one transaction at a time.**
- To request a facility transaction should enter in SEIZE block.
- Once a transaction has entered the SEIZE block, it owns the facility and other transactions are not allowed to enter this block.
  
- SEIZE A : gets the ownership of A
- RELEASE A : release the ownership of A

# Using Storage

- GPSS provides the storage modeling concept to represent a limited number of unit capacity.
- A storage is a resource that can be used by several transactions at a time until it becomes empty.
- To get units from a storage, transaction should call an ENTER block.
- To put units to a storage, transaction should call a LEAVE block.

## **ENTER A,B**

- Either takes or waits for a specified number of storage units.
- Where A: Storage Entity name or number  
B: Number of units by which to decrease the available storage capacity

## **LEAVE A,B**

- Increases the accessible storage units at a storage entity.
- A: Storage Entity name or number.  
B: Number of units by which to increase the available storage capacity.

## Collecting Time Statistics

- To collect data about how long it takes transactions to traverse a given segment of model.

### QUEUE A,B

- Updates queue entity statistics to reflect an increase in content
- A: Queue Entity name or number
- B: Number of units by which to increase the content of the Queue Entity. Default value is 1.
- Example: QUEUE WaitingLine, 1
- The content of the Queue Entity named WaitingLine is increased by one and the associated statistics accumulators are updated

## **DEPART A,B**

- Register statistics, which indicates a reduction in the content of a Queue Entity.
- A: Queue Entity name or number
- B: Number of units by which to decrease the content of the Queue Entity. Default value is 1.
- Example: DEPART WaitingLine, 1
- The content of the Queue Entity named WaitingLine is reduced by one and the associated statistics accumulators are updated.

# Branching Operation

## **TRANSFER Block**

- Causes the Active Transaction to jump to a new Block location.
- TRANSFER Block operates in “Unconditional Mode”.

## **TEST Block**

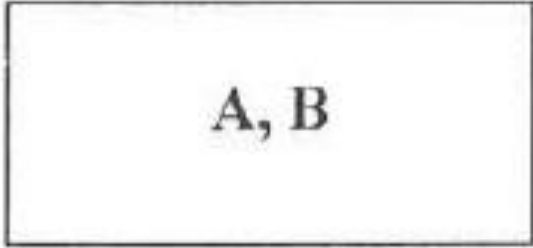
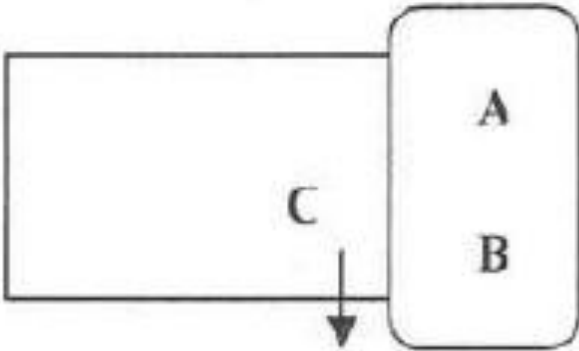
- To branch on some condition of the system.
- Transaction continues to next sequential program block if test is successful.
- Compares values, and controls the destination of the active transaction based on the result of the comparison.

Sample1


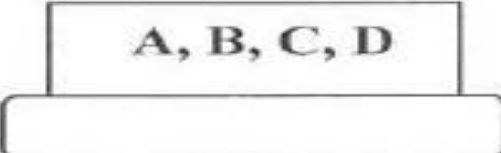

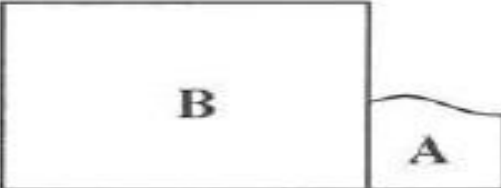
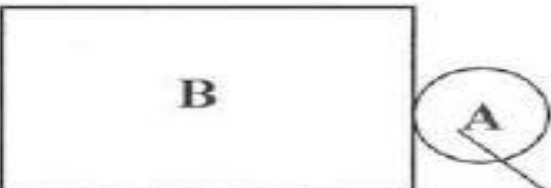
```
|; GPSS World Sample File - SAMPLE1.GPS
*****
*
*           Barber Shop Simulation
*
*****


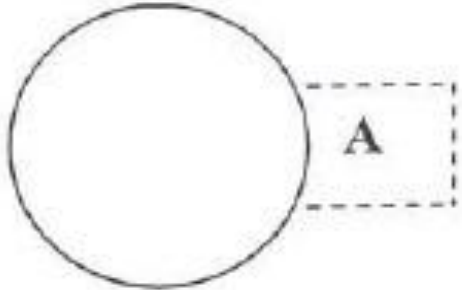

      GENERATE  300,100           ;Create next customer.
      QUEUE    Barber           ;Begin queue time.
      SEIZE     Barber           ;Own or wait for barber.
      DEPART    Barber           ;End queue time.
      ADVANCE   400,200          ;Haircut takes a few minutes.
      RELEASE   Barber           ;Haircut done. Give up the barber.
      TERMINATE 1                ;Customer leaves.
```

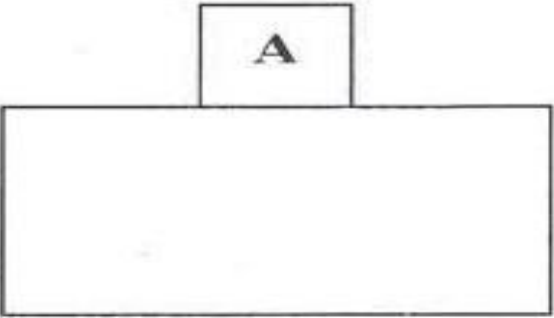
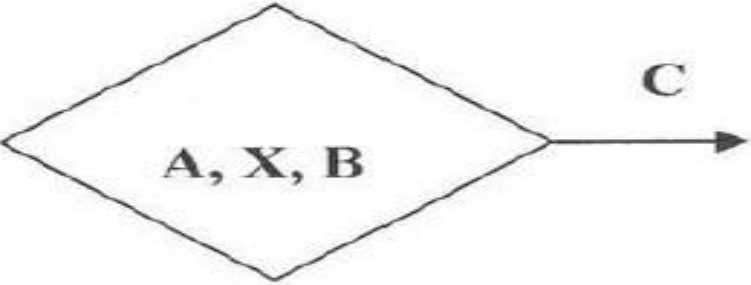
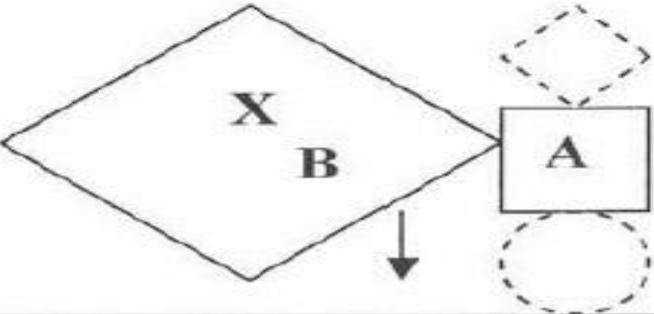
# GPSS Block-Diagram Symbols

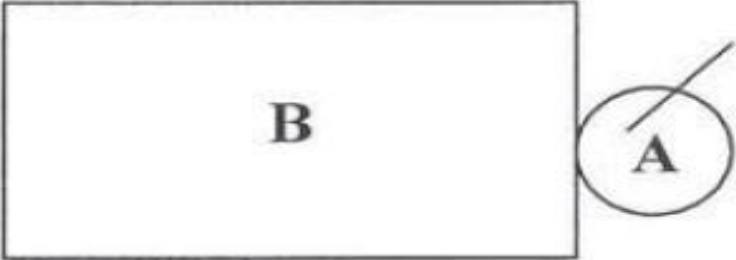
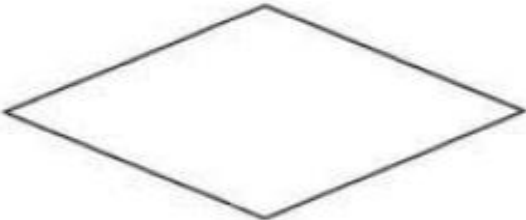
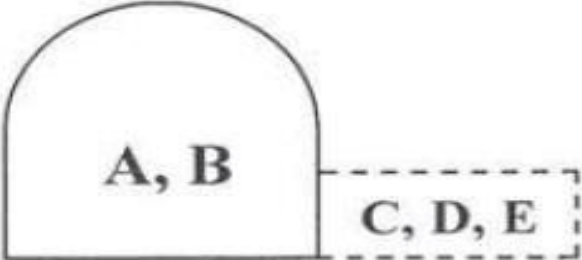
S.N.	Representation/Symbols	Meaning
1		Advance
2		Link


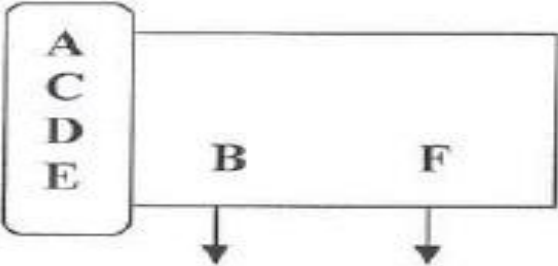
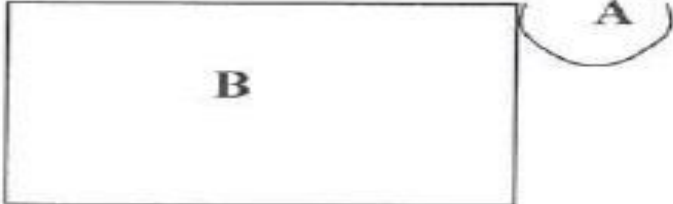
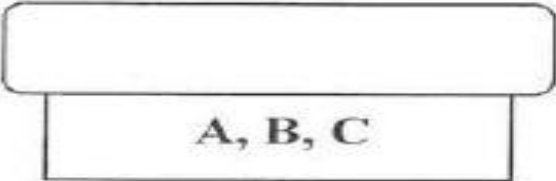


3		Seize
4		Assign
5		Logic
6		Tabulate
7		Depart

8		Mark
9		Terminate
10		Enter

11		Priority
12		Test
13		Gate

14		Queue
15		Transfer
16		Generate

17		Release
18		Unlink
19		Leave
20		Save Value

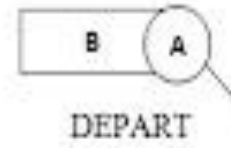
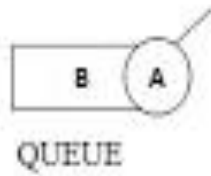
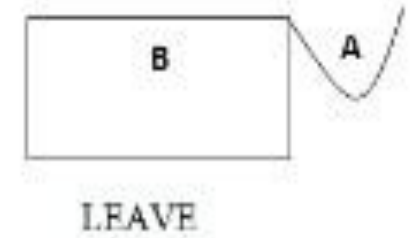
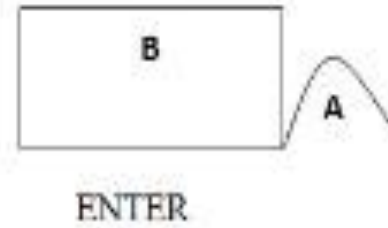
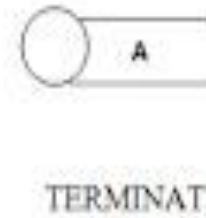
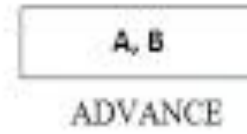
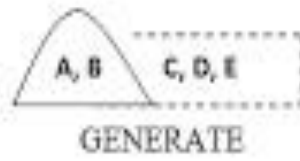
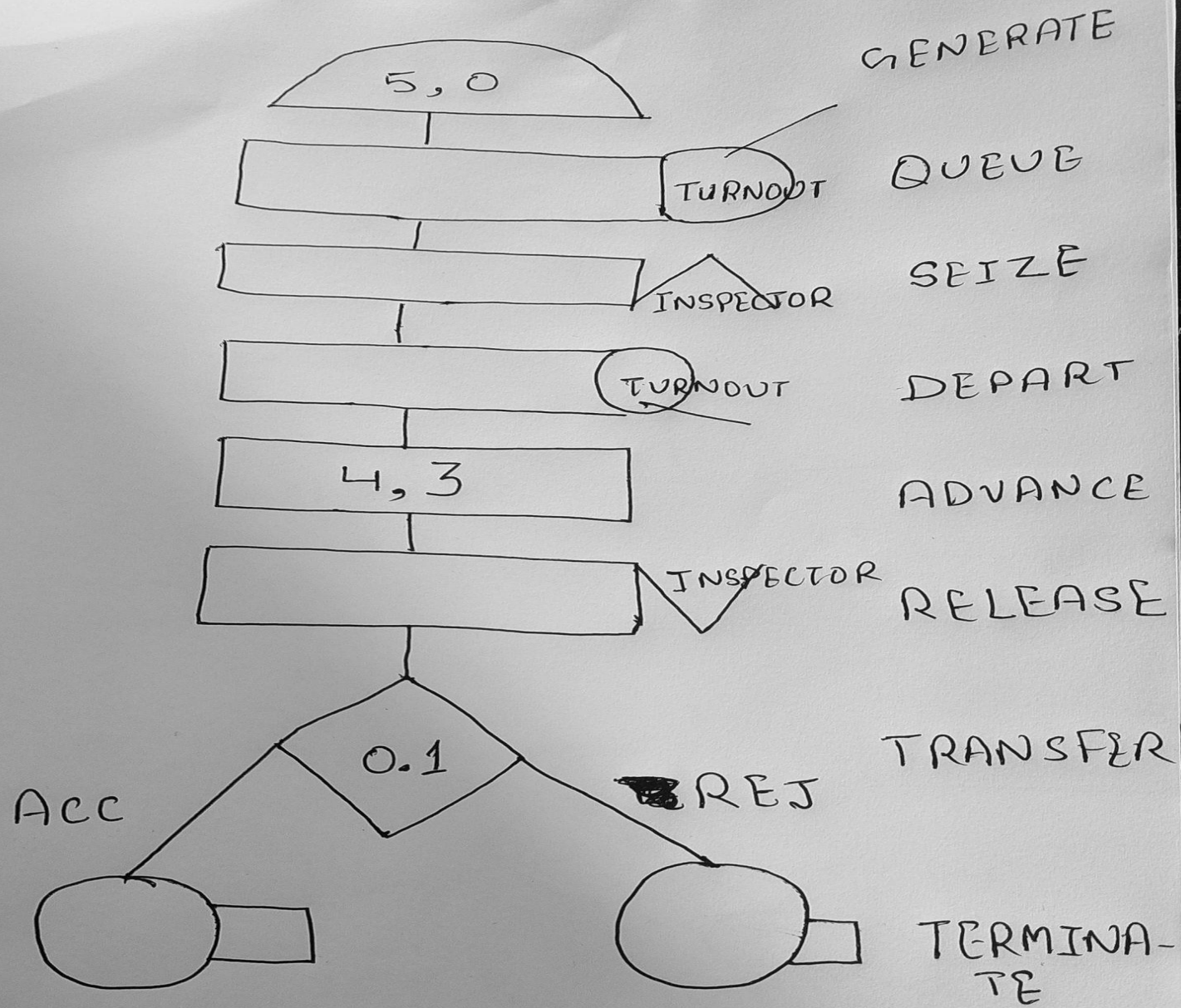


Figure: Mostly Used GPSS Models

## Example 1 - Manufacturing Shop Model Simulation

A machine tool is turning out parts at a rate of 1 per every 5 minutes. As they are finished, the part goes to an inspector, who takes 4 (+ or -) 3 minutes to examine each one and rejects about 10 % of the part. Simulate the system using GPSS model.

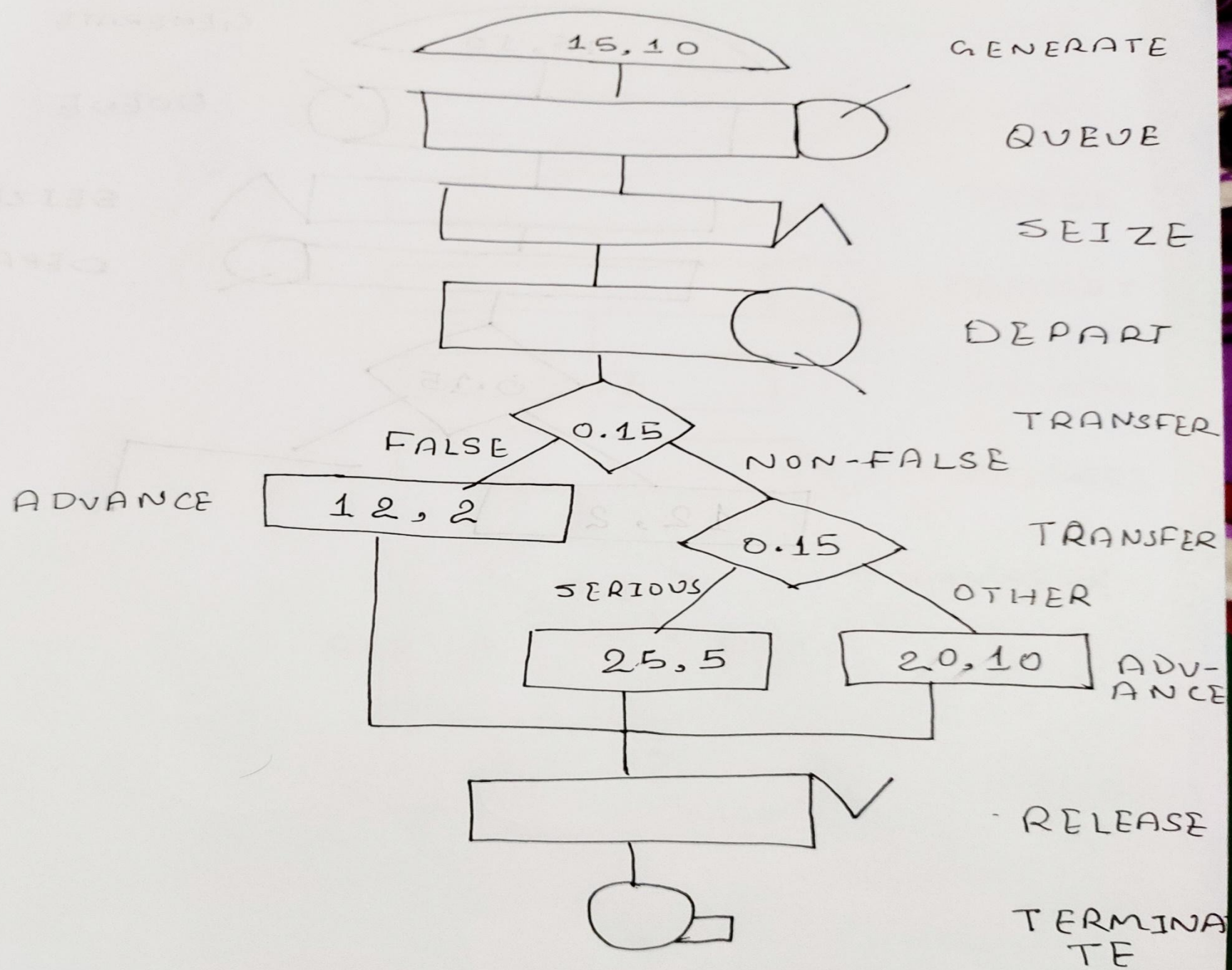




GENERATE 5, 0  
QUEUE TURNOUT  
SEIZE INSPECTOR  
DEPART TURNOUT  
  
ADVANCE 4, 3  
RELEASE INSPECTOR  
TRANSFER 0.1 REJ ACC  
TERMINATE

## Example 2

Ambulances are dispatched at a rate of one every 15 (+ or -) 10 mins. Fifteen percent of the calls are false alarms, which require 12 (+ or -) 2 mins to complete. All other calls can be one of two kinds. The first kind are classified as serious. They constitute 15% of non false alarms and take 25 (+ or -) 5 mins to complete. The other calls take 20 (+ or -) 10 mins. Simulate the model using GPSS.



GENERATE 15, 10

QUEUE

SEIZE

DEPART

TRANSFER 0.15 FALSE NON-FALSE

FALSE ADVANCE 12, 2

NON-FALSE TRANSFER 0.15 SERIOUS OTHER

SERIOUS ADVANCE 25, 5

OTHER ADVANCE 20, 10

RELEASE

TERMINATE

\* GPSS/H Block Section

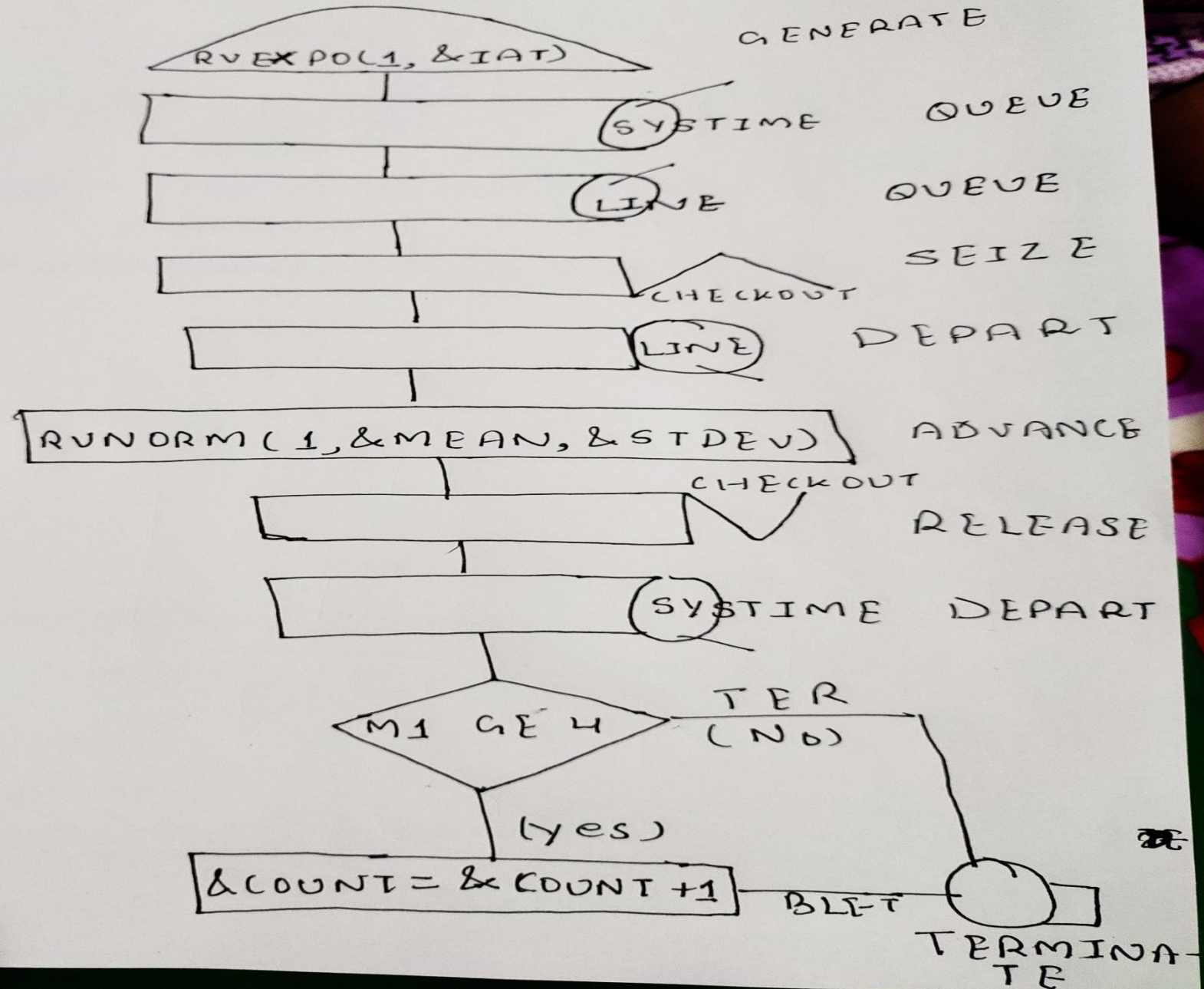
\*

GENERATE	RVEXPO(1,&IAT)	Exponential arrivals
QUEUE	SYSTIME	Begin response time data collection
QUEUE	LINE	Customer joins waiting line
SEIZE	CHECKOUT	Begin checkout at cash register
DEPART	LINE	Customer starting service leaves queue
ADVANCE	RVNORM(1,&MEAN,&STDEV)	Customer's service time
RELEASE	CHECKOUT	Customer leaves checkout area
DEPART	SYSTIME	End response time data collection
TEST GE	M1,4,TER	Is response time GE 4 minutes?
BLET	&COUNT=&COUNT+1	If so, add 1 to counter
TER	TERMINATE	1

\*

START      &LIMIT      Simulate for required number





# Simulation In SSF

- SPF stands for **Scalable Simulation Framework**.
- SSF is an API(Application Program Interface).
- It describes a set of capabilities for **object-oriented, process-view simulation**.
- SSF provides a single, unified interface for discrete-event simulation.
- It is designed to achieve high performance.
- The API is sparse and allows implementations to achieve high performance, e.g. on parallel computers.
- Can be widely used in network simulation by using the add-on framework SSFNet.
- SSF bridges the gap between models developed in **pure Java** and models developed in languages **specifically designed for simulation**.
- It also provides the flexibility offered by a general-programming language, yet has essential support for simulation.

# SSF Provides Five Base Classes

- 1. Processes:** Processes implement threads of control (where the *action* method contains the execution body of the thread)
- 2. Entity:** They describe the simulation objects.
- 3. inChannel:** Communication endpoint
- 4. outChannel:** Communication endpoint
- 5. Events:** Events define the messages sent between entities.



# SSF Model

## 1. Starting the Simulation :

- A simulation starts when any **entity's startAll() method is called**, commonly from the `main()` routine.
- The caller of `startAll()` specifies two timestamps: the simulation's start time, which defaults to 0, and the end time.

## 2. Initialization

- After **startAll()** has been called, but before any process has started executing, the framework calls the **init() routines of all processes and entities**.
- The Entity method shall return the simulation start time for the duration of initialization.

## 3. Process Execution

- After all processes have been initialized, they become eligible to run at the start time; **their actual execution is controlled by the framework**, which executes the process's `action()` callback method.

- Every time the `action()` method returns to the caller, the framework executes it again immediately, as many times are necessary to reach the end of the simulation.

#### **4. Framework Inner Loop**

- Within the framework, simulation time advances at a (possibly variable) rate determined by the arrival of events on channels, and the **duration of `waitFor()` statements**.
- When simulation time exceeds the end time specified in the original `startAll()` call, the simulation ends.

#### **5. Start, Pause, Resume and Join**

- The `pauseAll()` method allows the simulation to be paused gracefully. After `pauseAll()`, a call to `resumeAll()` resumes the simulation's forward progress.
- At any point, a call to `joinAll()` (e.g., from `main()`) will block without possibility of pause or resumption until simulation execution is complete.
- The `startAll()`, `pauseAll()`, `resumeAll()`, and `joinAll()` methods may not be called from within process code.

## 6. Framework Concurrency

- Each entity is said to be aligned to some object.
- Processes that are eligible to execute at each instant of simulation time will be scheduled by the framework: **sequentially within an alignment group** (but in implementation-dependent order), and **concurrently across alignment groups**.

## 7. Simulation Time

- The behavior of an SSF model is defined by the collective actions of its processes on its state.
- Every process resumption takes place at a particular simulation time, and each modification of model state by that resumption of the process takes place at that time.

## Example : Single Server Queue System (Checkout Counter)

- 1. SSQueue Class :** It is the class that contains the whole simulation experiment. It defines the experimental constants, contains SSF communication endpoints and defines inner class arrival.
- 2. Arrival process :** It is a SSF process that stores identity of entity, creates a random number generator and enqueue the generated new arrivals, then blocks for inter arrival time.
- 3. Server process :** It is the process that is called when a job has completed service or by a signal from the arrival process. It also updates statistics. Customers are dequeued from the waiting list or the process suspends if no customers were waiting.

# Single Server Queue Example

## SSQueue Class

```
class SSQueue extends Entity {  
  
    private static Random rng;  
    public static final double MeanServiceTime = 3.2;  
    public static final double SIGMA = 0.6;  
    public static final double MeanInterarrivalTime = 4.5;  
    public static final long ticksPerUnitTime = 1000000000;  
    public long generated=0;  
    public Queue Waiting;  
    outChannel out;  
    inChannel in;  
  
    public static long      TotalCustomers=0, MaxQueueLength=0,  
                           TotalServiceTime=0;  
    public static long      LongResponse=0, umResponseTime=0,  
                           jobStart;  
  
}
```

# Arrival class

```
class Arrivals extends process {
    private Random rng;
    private SSQueue owner;
    public Arrivals (SSQueue _owner, long seed) {
        super(_owner); owner = _owner;
        rng = new Random(seed);
    }
    public boolean isSimple() { return true; }
    public void action() {
        if ( generated++ > 0 ) {
            // put a new Customer on the queue with the present arrival time
            int Size = owner.Waiting.numElements();
            owner.Waiting.enqueue( new arrival(generated, now()));
            if( Size == 0) owner.out.write( new Event() ); // signal start of burst
        }
        waitFor(owner.d2t( owner.exponential(rng,
            owner.MeanInterarrivalTime)) );
    }
}
}
```

## Service Class

```
class Server extends process {  
    private Random rng;  
    private SSQueue owner ;  
    private arrival in_service;  
    private long service_time;  
  
    public Server(SSQueue _owner, long seed) {  
        super(_owner);  
        owner = _owner;  
        rng = new Random(seed);  
    }  
    public boolean isSimple() { return true; }
```



```

public void action() {
// if in_service is not null, we entered because of a job completion
    if( in_service != null ) {
        owner.TotalServiceTime += service_time;
        long in_system = (now() - in_service.arrival_time);
        owner.SumResponseTime += in_system;
        if( owner.t2d(in_system) > 4.0 ) owner.LongResponse++;
        in_service = null;
        if( owner.MaxQueueLength < owner.Waiting.numElements() + 1 )
            owner.MaxQueueLength = owner.Waiting.numElements() + 1;
        owner.TotalCustomers++;
    }
    if( owner.Waiting.numElements() > 0 ) {
        in_service = (arrival)owner.Waiting.dequeue();
        service_time = -1;
        while ( service_time < 0.0 )
            service_time = owner.d2t(owner.normal( rng, owner.MeanServiceTime, owner.SIGMA));
        waitFor( service_time );
    } else {
        waitOn( owner.in ); // we await a wake-up call
    }
}
}

```

---



# Other Simulation Software

## 1. Matlab

- Matlab is the easiest and most productive software environment for engineers and scientists.
- It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation.
- Typical uses include:
  - Math and computation
  - Algorithm development
  - Modeling, simulation, and prototyping
  - Data analysis, exploration, and visualization
  - Scientific and engineering graphics
  - Application development, including Graphical User Interface building

## 2. Arena

- Arena can be used for simulating discrete and continuous systems.
- At the heart of Arena is the **SIMAN** simulation language.
- Arena has a graphical user interface (GUI) built around the SIMAN language
- **Arena's Input Analyzer** automates the process of selecting the proper distribution and its inputs.
- The **Output Analyzer and Process Analyzer** automate comparison of different design alternatives.
- Arena is far more convenient than SIMAN, because it provides many handy features, such as high-level modules for model building, statistics definition and collection, animation of simulation runs (histories), and output report generation. Model building tends to be particularly intuitive, since many modules represent actual subsystems in the conceptual model or the real-life system under study.
- Complex models usually require both Arena modules and SIMAN blocks

### The Arena Basic Edition:

- For modeling business processes and other systems in support of high-level analysis needs.

### ❖ The Arena Standard Edition:

- For modeling more detailed discrete and continuous systems.
- Models are built from graphical objects called modules to define system logic and physical components.
- Includes modules focused on specific aspects of manufacturing and material-handling systems.

### ❖ The Arena Professional Edition:

- With capability to craft custom simulation objects that mirror components system terminology logic data etc of real system, including terminology, process logic, data, etc.

# AutoMod

- AutoMod is the 3 - D simulation tool that can model the **largest and most complex manufacturing, distribution, and material handling systems** by combining the ease-of-use features of a simulation language.
- It provides detailed, large models used for planning, operational decision support, and control-system testing.
- It mainly focuses on manufacturing and material-handling systems.
- An AutoMod model consists of one or more systems:
  - a. A system can be either a **process system** or a **movement system**.
  - b. A model may contain any number of systems, which can be saved and reused as objects in other models.
- Optimization can be done which is based on an evolutionary strategies algorithm.

## Assignment:

1. QUEST
2. Extend
3. Flexsim
4. Micro Saint
5. ProModel
6. Simul8
7. WITNESS