**Simulador básico em Java**

// Classe para geração de números aleatórios segundos várias distribuições

// Apenas a distribuição exponencial negativa está definida

public class Aleatorio {

// Gera um número segundo uma distribuição exponencial negativa de média m

static double exponencial (double m){

return (-m\*Math.log(Math.random()));

}

}

// Classe que representa um cliente

// Como são indistintos neste exemplo, está vazia

public class Cliente {

}

// Classe de onde vão ser derivados todos os eventos.

// Contém apenas os atributos e métodos comuns a todos os eventos.

// Por isso é uma classe abstracta. Não haverá instâncias desta classe num simulador.

public abstract class Evento {

protected double instante; // Instante de ocorrência do evento

protected Simulador s; // Simulador onde ocorre o evento

//Construtor

Evento (double i, Simulador s){

instante = i;

this.s = s;

}

// Método que determina se o evento corrente ocorre primeiro, ou não, do que o evento e1

// Se sim, devolve true; se não, devolve false

// Usado para ordenar por ordem crescente de instantes de ocorrência a lista de eventos do simulador

public boolean menor (Evento e1){

return (instante < e1.instante);

}

// Método que executa um evento; a ser definido em cada tipo de evento

abstract void executa (Servico s);

// Método que devolve o instante de ocorrência do evento

public double getInstante() {

return instante;

}

}

// Classe que representa a chegada de um cliente. Deriva de Evento.

public class Chegada extends Evento {

//Construtor

Chegada (double i, Simulador s){

super (i, s);

}

// Método que executa as acções correspondentes à chegada de um cliente

void executa (Servico serv){

// Coloca cliente no serviço - na fila ou a ser atendido, conforme o caso

serv.insereServico (new Cliente());

// Agenda nova chegada para daqui a Aleatorio.exponencial(s.media\_cheg) instantes

s.insereEvento (new Chegada(s.getInstante()+Aleatorio.exponencial(s.getMedia\_cheg()), s));

}

// Método que descreve o evento.

// Para ser usado na listagem da lista de eventos.

public String toString(){

return "Chegada em " + instante;

}

}

// Classe que representa a saída de um cliente. Deriva de Evento.

public class Saida extends Evento {

//Construtor

Saida (double i, Simulador s){

super(i, s);

}

// Método que executa as acções correspondentes à saída de um cliente

void executa (Servico serv){

// Retira cliente do serviço

serv.removeServico();

}

// Método que descreve o evento.

// Para ser usado na listagem da lista de eventos.

public String toString(){

return "Saída em " + instante;

}

}

import java.util.\*;

// Classe que contém, em cada instante, os eventos a serem executados, ordenados por instantes de ocorrência crescentes.

// Funciona como uma agenda.

// Deriva da classe LinkedList.

public class ListaEventos extends LinkedList <Evento> {

private static final long serialVersionUID = 1; // número para serialização {

private Simulador s; // Simulador a que pertence a lista de eventos

// Construtor

ListaEventos (Simulador s){

this.s = s;

}

// Método para inserir um evento na lista de eventos

public void insereEvento (Evento e1){

int i = 0;

// Determina posição correcta do evento e1 na lista

// A lista é ordenada por ordem crescente dos instantes de ocorrência dos eventos

while (i < size() && ((Evento)get(i)).menor(e1)) i++;

// Coloca evento e1 na lista

add(i, e1);

}

// Método informativo apenas. Imprime o conteúdo da lista de eventos em cada instante

public void print (){

int i;

System.out.println ("--- Lista de eventos em " + s.getInstante() + " ---");

for (i = 0; i < size(); i++) System.out.println ("Evento " + (i+1) + " é uma " + (Evento)(get(i)));

}

}

import java.util.\*;

// Classe que representa um serviço com uma fila de espera associada

public class Servico {

private int estado; // Variável que regista o estado do serviço: 0 - livre; 1 - ocupado

private int atendidos; // Número de clientes atendidos até ao momento

private double temp\_ult, soma\_temp\_esp, soma\_temp\_serv; // Variáveis para cálculos estatísticos

private Vector<Cliente> fila; // Fila de espera do serviço

private Simulador s; // Referência para o simulador a que pertence o serviço

// Construtor

Servico (Simulador s){

this.s = s;

fila = new Vector<Cliente> (); // Cria fila de espera

estado = 0; // Livre

// Tempo que passou desde o último evento. Neste caso 0, porque a simulação ainda não começou.

temp\_ult = s.getInstante();

atendidos = 0; // Inicialização de variáveis

soma\_temp\_esp = 0;

soma\_temp\_serv = 0;

}

// Método que insere cliente (c) no serviço

public void insereServico (Cliente c){

if (estado == 0) { // Se serviço livre,

estado ++; // fica ocupado e

// agenda saída do cliente c para daqui a s.getMedia\_serv() instantes

s.insereEvento (new Saida(s.getInstante()+s.getMedia\_serv(), s));

}

else fila.addElement(c); // Se serviço ocupado, o cliente vai para a fila de espera

}

// Método que remove cliente do serviço

public void removeServico (){

atendidos++; // Regista que acabou de atender + 1 cliente

if (fila.size()== 0) estado --; // Se a fila está vazia, liberta o serviço

else { // Se não,

// vai buscar próximo cliente à fila de espera e

Cliente c = (Cliente)fila.firstElement();

fila.removeElementAt(0);

// agenda a sua saida para daqui a s.getMedia\_serv() instantes

s.insereEvento (new Saida(s.getInstante()+s.getMedia\_serv(),s));

}

}

// Método que calcula valores para estatísticas, em cada passo da simulação ou evento

public void act\_stats(){

// Calcula tempo que passou desde o último evento

double temp\_desde\_ult = s.getInstante() - temp\_ult;

// Actualiza variável para o próximo passo/evento

temp\_ult = s.getInstante();

// Contabiliza tempo de espera na fila

// para todos os clientes que estiveram na fila durante o intervalo

soma\_temp\_esp += fila.size() \* temp\_desde\_ult;

// Contabiliza tempo de atendimento

soma\_temp\_serv += estado \* temp\_desde\_ult;

}

// Método que calcula valores finais estatísticos

public void relat (){

// Tempo médio de espera na fila

double temp\_med\_fila = soma\_temp\_esp / (atendidos+fila.size());

// Comprimento médio da fila de espera

// s.getInstante() neste momento é o valor do tempo de simulação,

// uma vez que a simulação começou em 0 e este método só é chamdo no fim da simulação

double comp\_med\_fila = soma\_temp\_esp / s.getInstante();

// Tempo médio de atendimento no serviço

double utilizacao\_serv = soma\_temp\_serv / s.getInstante();

// Apresenta resultados

System.out.println("Tempo médio de espera "+temp\_med\_fila);

System.out.println("Comp. médio da fila "+comp\_med\_fila);

System.out.println("Utilização do serviço "+utilizacao\_serv);

System.out.println("Tempo de simulação "+s.getInstante()); // Valor actual

System.out.println("Número de clientes atendidos "+atendidos);

System.out.println("Número de clientes na fila "+fila.size()); // Valor actual

}

// Método que devolve o número de clientes atendidos no serviço até ao momento

public int getAtendidos() {

return atendidos;

}

}

public class Simulador {

// Relógio de simulação - variável que contém o valor do tempo em cada instante

private double instante;

// Médias das distribuições de chegadas e de atendimento no serviço

private double media\_cheg, media\_serv;

// Número de clientes que vão ser atendidos

private int n\_clientes;

// Serviço - pode haver mais do que um num simulador

private Servico servico;

// Lista de eventos - onde ficam registados todos os eventos que vão ocorrer na simulação

// Cada simulador só tem uma

private ListaEventos lista;

// Construtor

public Simulador() {

// Inicialização de parâmetros do simulador

media\_cheg = 1;

media\_serv = 1.5;

n\_clientes = 100;

// Inicialização do relógio de simulação

instante = 0;

// Criação do serviço

servico = new Servico (this);

// Criação da lista de eventos

lista = new ListaEventos(this);

// Agendamento da primeira chegada

// Se não for feito, o simulador não tem eventos para simular

insereEvento (new Chegada(instante, this));

}

// programa principal

public static void main(String[] args) {

// Cria um simulador e

Simulador s = new Simulador();

// põe-o em marcha

s.executa();

}

// Método que insere o evento e1 na lista de eventos

void insereEvento (Evento e1){

lista.insereEvento (e1);

}

// Método que actualiza os valores estatísticos do simulador

private void act\_stats(){

servico.act\_stats();

}

// Método que apresenta os resultados de simulação finais

private void relat (){

System.out.println();

System.out.println("------- Resultados finais -------");

System.out.println();

servico.relat();

}

// Método executivo do simulador

public void executa (){

Evento e1;

// Enquanto não atender todos os clientes

while (servico.getAtendidos() < n\_clientes){

lista.print(); // Mostra lista de eventos - desnecessário; é apenas informativo

// Retira primeiro evento (é o mais iminente) da lista de eventos

e1 = (Evento)(lista.removeFirst());

instante = e1.getInstante(); // Actualiza relógio de simulação

act\_stats(); // Actualiza valores estatísticos

e1.executa(servico); // Executa evento

};

relat(); // Apresenta resultados de simulação finais

}

// Método que devolve o instante de simulação corrente

public double getInstante() {

return instante;

}

// Método que devolve a média dos intervalos de chegada

public double getMedia\_cheg() {

return media\_cheg;

}

// Método que devolve a média dos tempos de serviço

public double getMedia\_serv() {

return media\_serv;

}

}