# ПРИЛОЖЕНИЕ А

# ИСХОДНЫЙ КОД ПРИЛОЖЕНИЯ МОДЕЛИ БЕСПРОВОДНОЙ СЕНСОРНОЙ СЕТИ

**import** com.sgu.magistr.StatsGraphBuild;  
**import** java.io.FileWriter;  
**import** java.io.IOException;  
  
**public class** AModel {

*//интенсивность потока в систему S1 (микропроцессор)*  
 **private double L01** = 0.0;

*//интенсивность потока в в систему S2 (приемо-передатчик)* **private double L02** = 0.0;

*//общая интенсивность поступления требований в узел* **private double L0**;

*//интенсивность обслуживания требований класса 1 в системе S1* **private static final double *MU11*** = 200000.0;

*//интенсивность обслуживания требований класса 2 в системе S1* **private static final double *MU12*** = 400000.0;

*//интенсивность обслуживания требований классов 1 и 2 в системе S2*

**private static final double *MU2*** = 25000.0;

*//параметр экспон ф-ции распред-я длит восстановления узла* **private static final double *ALPHA*** = 0.5;

*//параметр экспон ф-ции распред-я длит восстановления узла* **private static final double *BETTA*** = 1000.0; **private static final int *Q\_CNT*** = 40; **private static** FileWriter *writer* = **null**;  
 **private double p1**[] = **new double**[***Q\_CNT***];  
 **private double p2**[] = **new double**[***Q\_CNT***];  
 **private double p23**[] = **new double**[***Q\_CNT***];  
 **private double s01**, **s02** = 0.0;  
 **private double s**, **p** = 0.0;  
 **private double q11**, **q12**, **q22**, **q23** = 0.0;  
 **private double u11**, **u12**, **u22**, **u23** = 0.0;  
 **private int cellNum**;  
  
 **private void** setL0(**double** l1) {  
 **L01** = l1;  
 **L02** = **L01** \* 1000.0;  
 **L0** = **L01** + **L02**;  
 }  
  
 *//число сочетаний из n по m* **private double** NS(**int** n, **int** m) {  
 **double** a = 1.0;  
 **for** (**int** i = 1; i <= n - m; i++) {  
 a = a \* (m + i) / i;  
 }  
 **return** a;  
 }  
  
 *//Вычисление характеристики для для 1-й СМО* **private double** calcS1Characteristics() **throws** IOException {  
 **for** (**int** n = 0; n < ***Q\_CNT***; n++) {  
 **p1**[n] = 0;  
 **p2**[n] = 0;  
 }  
 **for** (**int** n = 1; n <= ***Q\_CNT***; n++) {  
 **for** (**int** m = 0; m <= n; m++) {  
 **p** = NS(n, m) \* Math.*pow*(**L01** / ***MU11***, m) \* Math.*pow*(**L02** / ***MU12***, n - m);  
 **if** (m > 0)  
 **p1**[m - 1] = **p1**[m - 1] + **p**;  
 **if** (n - m > 0)  
 **p2**[n - 1 - m] = **p2**[n - 1 - m] + **p**;  
 **s** += **p**;  
 }  
 }  
 **s01** = 1 / (1 + **s**);  
 **for** (**int** n = 0; n < ***Q\_CNT***; n++) {  
 **p1**[n] = **p1**[n] \* **s01**;  
 **p2**[n] = **p2**[n] \* **s01**;  
 }  
 **for** (**int** n = 1; n <= ***Q\_CNT***; n++) {  
 **q11** = **q11** + n \* **p1**[n - 1];  
 }  
 **s** += **q11**;  
 **for** (**int** n = 0; n < ***Q\_CNT***; n++) {  
 **q12** += (n + 1) \* **p2**[n];  
 }  
 **s** += **p**;  
 **return s**;  
 }  
  
 **private double** calcS2Characteristics() **throws** IOException {  
 **s02** = 1 - (**L0** + **L02**) / ***MU2***;  
 **for** (**int** n = 1; n <= ***Q\_CNT***; n++) {  
 **p23**[n - 1] = **s02** \* Math.*pow*((**L0** + **L02**) / ***MU2***, n);  
 }  
 **p** = 0.0;  
 **for** (**int** n = 1; n <= ***Q\_CNT***; n++) {  
 **p** = **p** + n \* **p23**[n - 1];  
 }  
 **q22** = **p** \* **L02** / (**L0** + **L02**);  
 **q23** = **p** \* **L0** / (**L0** + **L02**);  
  
 **s** = **s** + **p**;  
 **return s**;  
 }  
  
 **private void** calcS2S2TimeCharacteristics() **throws** IOException {  
 **u11** = **q11** / **L01**;  
 **u12** = **q12** / **L02**;  
 **u22** = **q22** / **L0**;  
 **u23** = **q23** / **L02**;  
 }  
  
 **private void** writeOutput(**int** writeFlag) **throws** IOException {  
 *writer*.write(  
 **"L01: "** + **L01** +  
 **"\nМО числа требований 1 класса в S1: "** + **q11** +  
 **"\nМО числа требований 2 класса в S1: "** + **q12** +  
 **"\nМО числа требований 2 класса в S2: "** + **q22** +  
 **"\nМО числа требований 3 класса в S2: "** + **q23** +  
 **"\nМО числа требований в сети: "** + **s** +  
 **"\nВремя реакции сети: "** + **s** / **L0** +  
 **"\nВероятность, что узел пуст: "** + **s01** \* **s02** +  
 **"\nМО числа потерянных пакетов: "** + **L01** / ***BETTA*** +  
 **"\nМО длительности пребывания требований 1 класса в S1: "** + **u11** +  
 **"\nМО длительности пребывания требований 2 класса в S1: "** + **u12** +  
 **"\nМО длительности пребывания требований 2 класса в S2: "** + u22 +  
 **"\nМО длительности пребывания требований 3 класса в S2: "** + u23 + **"\n\n"** );  
  
 StatsGraphBuild.updateCell(1 + writeFlag, cellNum, L01);  
 StatsGraphBuild.updateCell(2 + writeFlag, cellNum, q11);  
 StatsGraphBuild.updateCell(3 + writeFlag, cellNum, q12);  
 StatsGraphBuild.updateCell(4 + writeFlag, cellNum, q22);  
 StatsGraphBuild.updateCell(5 + writeFlag, cellNum, q23);  
 StatsGraphBuild.updateCell(6 + writeFlag, cellNum, s);  
 StatsGraphBuild.updateCell(7 + writeFlag, cellNum, u11);  
 StatsGraphBuild.updateCell(8 + writeFlag, cellNum, u12);  
 StatsGraphBuild.updateCell(9 + writeFlag, cellNum, u22);  
 StatsGraphBuild.updateCell(10 + writeFlag, cellNum, u23);  
 writer.flush();  
 }  
  
 **public** AModel(**double** l1, FileWriter writer, **int** cellNum) **throws** IOException {  
 **this**.writer = writer;  
 **this**.cellNum = cellNum;  
 setL0(l1);  
 calcS1Characteristics();  
 calcS2Characteristics();  
 calcS2S2TimeCharacteristics();  
 writeOutput(0);  
 binomReliability();  
 writeOutput(44);  
 writer.flush();  
 }  
  
 **private double** binomReliability() {  
 **double** p = BETTA / (ALPHA + BETTA);  
 **double** p1;  
 **double** qq11 = 0.0;  
 **double** qq12 = 0.0;  
 **double** qq22 = 0.0;  
 **double** qq23 = 0.0;  
 **double** uu11 = 0.0;  
 **double** uu12 = 0.0;  
 **double** uu22 = 0.0;  
 **double** uu23 = 0.0;  
 **double** ss = 0.0;  
  
 **for** (**int** i = 0; i < 1000; i++) {  
 p1 = calculateReliability(1000, i, p);  
 qq11 += q11 \* p1;  
 qq12 += q12 \* p1;  
 qq22 += q22 \* p1;  
 qq23 += q23 \* p1;  
 uu11 += u11 \* p1;  
 uu12 += u12 \* p1;  
 uu22 += u22 \* p1;  
 uu23 += u23 \* p1;  
 uu23 += u23 \* p1;  
 ss += s \* p1;  
 }  
  
 q11 = qq11;  
 q12 = qq12;  
 q22 = qq22;  
 q23 = qq23;  
 u11 = uu11;  
 u12 = uu12;  
 u22 = uu22;  
 u23 = uu23;  
 s = ss;  
 **return** 0;  
 }  
  
 **private double** calculateReliability(**int** n, **int** k, **double** p) {  
  
 **return** NS(n, k) \* Math.pow(p, k) \* Math.pow(1.0 - p, n - k);  
 }  
}  
  
**public class** Event {  
  
 **private double time**;  
 **private double genTime**;  
 **private** EventTypesEnum **eventType**;  
  
 **public** Event(**double** time, EventTypesEnum eventType) {  
 **this**.**time** = time;  
 **this**.**eventType** = eventType;  
 }  
  
 **public** Event(**double** time, **double** genTime, EventTypesEnum eventType) {  
 **this**.**time** = time;  
 **this**.**genTime** = genTime;  
 **this**.**eventType** = eventType;  
 }  
  
 **public double** getTime() {  
 **return time**;  
 }  
  
 **public** EventTypesEnum getEventType() {  
 **return eventType**;  
 }   
}

**package** com.sgu.magistr.imitationmodel;  
  
**public enum** EventTypesEnum {  
 ***SEND\_TO\_MP\_CLASS\_1***, *// Поступление требования 1 класса в очередь микропроцессора* ***SEND\_TO\_PP\_CLASS\_2***, *// Поступление требования 2 класса в очередь приемопередатчика* ***SEND\_TO\_MP\_CLASS\_2***, *// Поступление требования 2 класса в очередь микропроцессора* ***SEND\_TO\_PP\_CLASS\_3***, *// Поступление требования 3 класса в очередь микропроцессора* ***FINISH\_SERVE\_IN\_MP***, *// Завершение обслуживания требований класса 2 в микропроцессоре* ***FINISH\_SERVE\_IN\_PP***, *// Завершение обслуживания в приемопередатчике* ***FINISH\_SERVE\_CLASS\_3***; *// Завершение обслуживания класса 3*}

**import** com.sgu.magistr.StatsGraphBuild;  
  
**import** java.io.FileWriter;  
**import** java.io.IOException;  
**import** java.util.\*;  
  
**public class** IModel {  
  
 **private final int eRowNum**;  
 *//интенсивность поступления требований 1 класса в S1* **private double L01** = 0.0;  
 *//интенсивность поступления требований 2 класса в S2* **private double L02** = 0.0;  
 *//интенсивность обработки требований 1 класса в S1* **private static final double *MU11*** = 1.0 / 200000;  
 *//интенсивность обработки требований 2 класса в S1* **private static final double *MU12*** = 1.0 / 400000;  
 *//интенсивность обработки требований S2* **private static final double *MU23*** = 1.0 / 25000;  
 *//модельное время* **private double tMod** = 0.0;  
  
 *// очередь требований в систему S1:* **private** List<Requirement> **S1Queue** = Collections.*synchronizedList*(**new** ArrayList<Requirement>());  
 *// очередь требований в систему S2:* **private** List<Requirement> **S2Queue** = Collections.*synchronizedList*(**new** ArrayList<Requirement>());  
 *//список генерируемых событий* **private** Set<Event> **EventsList** = **new** HashSet<Event>();  
 *//ассоц. массив для сумм k эл-тов 1 класса в очереди S1* **private** Map<Integer, Double> **S11KMap** = **new** HashMap<Integer, Double>();  
 *//ассоц. массив для сумм k эл-тов 2 класса в очереди S1* **private** Map<Integer, Double> **S12KMap** = **new** HashMap<Integer, Double>();  
 *//ассоц. массив для сумм k эл-тов 2 класса в очереди S2* **private** Map<Integer, Double> **S22KMap** = **new** HashMap<Integer, Double>();  
 *//ассоц. массив для сумм k эл-тов 3 класса в очереди S2* **private** Map<Integer, Double> **S23KMap** = **new** HashMap<Integer, Double>();  
 **private** Random **random** = **new** Random();  
  
 *//Значение текущего момента времени* **private double currTime** = 0;  
 *//Значение предыдущего момента времени для S1* **private double S1prevTime** = 0.0;  
 *//Значение предыдущего момента времени для S2* **private double S2prevTime** = 0.0;  
 *//Флаг занятости микропроцессора* **private boolean isBusyS1** = **false**;  
 *//Флаг занятости приемопередатчика* **private boolean isBusyS2** = **false**;  
  
 **private static** FileWriter *writer* = **null**;  
 **private int cellNum**;  
  
 **private void** setL0(**double** l1) {  
 **L01** = l1;  
 **L02** = **L01** \* 1000.0;  
 **double** L0 = **L01** + **L02**;  
 **tMod** = 100000 / L0;  
 }  
  
 */\*\*  
 \* Управляющая подпрограмма.  
 \*/* **private void** emulate() **throws** IOException {  
 Requirement curReq = **null**;  
 Requirement nextReq = **null**;  
 **EventsList**.add(**new** Event(**currTime**, EventTypesEnum.***SEND\_TO\_MP\_CLASS\_1***));  
 **EventsList**.add(**new** Event(**currTime**, EventTypesEnum.***SEND\_TO\_PP\_CLASS\_2***));  
  
 **double** TS11 = 0.0; *//Сумма длительностей пребывания требований 1 класса в S1* **double** TS12 = 0.0; *//Сумма длительностей пребывания требований 2 класса в S1* **double** TS22 = 0.0;*//Сумма длительностей пребывания всех требований в S2* **double** TS23 = 0.0; *//Сумма длительностей пребывания всех требований в S2* **int** NS11 = 1; *//Сумма требований 1 класса поступающих в S1* **int** NS12 = 1; *//Сумма требований 2 класса поступающих в S1* **int** NS22 = 1; *//Сумма требований 2 класса поступающих в S2* **int** NS23 = 1; *//Сумма требований 3 класса поступающих в S2* **double** MS11; *//МО времени пребывания требований 1 класса в S1* **double** MS12; *//МО времени пребывания требований 2 класса в S1* **double** MS22; *//МО времени пребывания требований 2 класса в S2* **double** MS23; *//МО времени пребывания требований 3 класса в S2* **while** (**currTime** < **tMod**) {  
 Event currEvent = getMinTimeEvent(**EventsList**);  
 **currTime** = currEvent.getTime();  
  
 **switch** (currEvent.getEventType()) {  
 **case *SEND\_TO\_MP\_CLASS\_1***:  
 curReq = **new** Requirement(**currTime**, 1);  
 mTimeAccumulator(**"MP\_1"**, getQueueCount(**S1Queue**, 1), **currTime** - **S1prevTime**);  
 **S1prevTime** = **currTime**;  
 **S1Queue**.add(curReq);  
 **EventsList**.add(**new** Event(**currTime** + genExp(**L01**), EventTypesEnum.***SEND\_TO\_MP\_CLASS\_1***));  
 **if** (!**isBusyS1**) {  
 curReq.setProcStartTime(**currTime**);  
 **EventsList**.add(**new** Event(**currTime** + ***MU11***, EventTypesEnum.***FINISH\_SERVE\_IN\_MP***));  
 **isBusyS1** = **true**;  
 }  
 **break**;  
 **case *SEND\_TO\_PP\_CLASS\_2***:  
 curReq = **new** Requirement(**currTime**, 2);  
 mTimeAccumulator(**"PP\_2"**, getQueueCount(**S2Queue**, 2), **currTime** - **S2prevTime**);  
 **S2prevTime** = **currTime**;  
 **S2Queue**.add(curReq);  
 **EventsList**.add(**new** Event(**currTime** + genExp(**L02**), EventTypesEnum.***SEND\_TO\_PP\_CLASS\_2***));  
 **if** (!**isBusyS2**) {  
 curReq.setProcStartTime(**currTime**);  
 **EventsList**.add(**new** Event(**currTime** + ***MU23***, EventTypesEnum.***FINISH\_SERVE\_IN\_PP***));  
 **isBusyS2** = **true**;  
 }  
 **break**;  
  
 **case *SEND\_TO\_PP\_CLASS\_3***:  
 curReq = **new** Requirement(**currTime**, 3);  
 mTimeAccumulator(**"PP\_3"**, getQueueCount(**S2Queue**, 3), **currTime** - **S2prevTime**);  
 **S2prevTime** = **currTime**;  
 **S2Queue**.add(curReq);  
 **if** (!**isBusyS2**) {  
 curReq.setProcStartTime(**currTime**);  
 **EventsList**.add(**new** Event(**currTime** + ***MU23***, EventTypesEnum.***FINISH\_SERVE\_CLASS\_3***));  
 **isBusyS2** = **true**;  
 }  
 **break**;  
  
 **case *SEND\_TO\_MP\_CLASS\_2***:  
 curReq = **new** Requirement(**currTime**, 2);  
 mTimeAccumulator(**"MP\_2"**, getQueueCount(**S1Queue**, 2), **currTime** - **S1prevTime**);  
 **S1Queue**.add(curReq);  
 **S1prevTime** = **currTime**;  
 **if** (!**isBusyS1**) {  
 curReq.setProcStartTime(**currTime**);  
 **EventsList**.add(**new** Event(**currTime** + ***MU12***, EventTypesEnum.***FINISH\_SERVE\_IN\_MP***));  
 **isBusyS1** = **true**;  
 }  
 **break**;  
  
 **case *FINISH\_SERVE\_IN\_MP***:  
 curReq = **S1Queue**.get(0);  
 **if** (curReq.getReqClass() == 1) {  
 TS11 += **currTime** - curReq.getGenerationTime();  
 NS11++;  
 mTimeAccumulator(**"MP\_1"**, getQueueCount(**S1Queue**, 1), **currTime** - **S1prevTime**);  
 } **else if** (curReq.getReqClass() == 2) {  
 TS12 += **currTime** - curReq.getGenerationTime();  
 NS12++;  
 mTimeAccumulator(**"MP\_2"**, getQueueCount(**S1Queue**, 2), **currTime** - **S1prevTime**);  
 }  
 **EventsList**.add(**new** Event(**currTime**, curReq.getGenerationTime(), EventTypesEnum.***SEND\_TO\_PP\_CLASS\_3***));  
 **S1prevTime** = **currTime**;  
 **S1Queue**.remove(curReq);  
 **if** (!**S1Queue**.isEmpty()) {  
 nextReq = **S1Queue**.get(0);  
 nextReq.setProcStartTime(**currTime**);  
 **if** (nextReq.getReqClass() == 1)  
 **EventsList**.add(**new** Event(**currTime** + ***MU11***, EventTypesEnum.***FINISH\_SERVE\_IN\_MP***));  
 **else  
 EventsList**.add(**new** Event(**currTime** + ***MU12***, EventTypesEnum.***FINISH\_SERVE\_IN\_MP***));  
 **isBusyS1** = **true**;  
 } **else  
 isBusyS1** = **false**;  
 **break**;  
  
 **case *FINISH\_SERVE\_IN\_PP***:  
 curReq = **S2Queue**.get(0);  
 NS22++;  
 TS22 += **currTime** - curReq.getGenerationTime();  
 **EventsList**.add(**new** Event(**currTime**, curReq.getGenerationTime(), EventTypesEnum.***SEND\_TO\_MP\_CLASS\_2***));  
 mTimeAccumulator(**"PP\_2"**, getQueueCount(**S2Queue**, 2), **currTime** - **S2prevTime**);  
 **S2prevTime** = **currTime**;  
 **S2Queue**.remove(curReq);  
 evalMath();  
 **break**;  
  
 **case *FINISH\_SERVE\_CLASS\_3***:  
 NS23++;  
 curReq = **S2Queue**.get(0);  
 TS23 += **currTime** - curReq.getGenerationTime();  
 curReq.setReleaseTime(**currTime**);  
 mTimeAccumulator(**"PP\_3"**, getQueueCount(**S2Queue**, 3), **currTime** - **S2prevTime**);  
 **S2prevTime** = **currTime**;  
 **S2Queue**.remove(curReq);  
 evalMath();  
 **break**;  
 }  
 **EventsList**.remove(currEvent);  
 }  
 MS11 = TS11 / NS11;  
 MS12 = TS12 / NS12;  
 MS22 = TS22 / NS22;  
 MS23 = TS23 / NS23;  
 **double** S11Cnt = calculateMCount(**S11KMap**);  
 **double** S12Cnt = calculateMCount(**S12KMap**);  
 **double** S22Cnt = calculateMCount(**S22KMap**);  
 **double** S23Cnt = calculateMCount(**S23KMap**);  
  
 *writer*.write(**"L01: "** + **L01** +  
 **"\nМО числа требований 1 класса в S1: = "** + S11Cnt + **"\n"** +  
 **"МО числа требований 2 класса в S1: "** + S12Cnt + **"\n"** +  
 **"МО числа требований 2 класса в S2: "** + S22Cnt + **"\n"** +  
 **"МО числа требований 3 класса в S2: "** + S23Cnt + **"\n"** +  
 **"МО числа требований в сети: "** + (S11Cnt + S22Cnt + S12Cnt + S23Cnt) + **"\n"** +  
 **"МО длительности пребывания требований 1 класса в S1: "** + MS11 + **"\n"** +  
 **"МО длительности пребывания требований 2 класса в S1: "** + MS12 + **"\n"** +  
 **"МО длительности пребывания требований 2 класса в S2: "** + MS22 + **"\n"** +  
 **"МО длительности пребывания требований 3 класса в S2: "** + MS23 + **"\n\n"**);  
  
 StatsGraphBuild.*updateCell*(13, **cellNum**, **L01**);  
 StatsGraphBuild.*updateCell*(**eRowNum** + 14, **cellNum**, S11Cnt);  
 StatsGraphBuild.*updateCell*(25, **cellNum**, S12Cnt);  
 StatsGraphBuild.*updateCell*(26, **cellNum**, S22Cnt);  
 StatsGraphBuild.*updateCell*(27, **cellNum**, S23Cnt);  
 StatsGraphBuild.*updateCell*(28, **cellNum**, (S11Cnt + S22Cnt + S12Cnt + S23Cnt));  
 StatsGraphBuild.*updateCell*(**eRowNum** + 29, **cellNum**, MS11);  
 StatsGraphBuild.*updateCell*(40, **cellNum**, MS12);  
 StatsGraphBuild.*updateCell*(41, **cellNum**, MS22);  
 StatsGraphBuild.*updateCell*(42, **cellNum**, MS23);  
 }  
  
 **private int** getQueueCount(List<Requirement> queue, **int** classNum){  
 **int** counter = 0;  
 **for** (Requirement req : queue) {  
 **if**(req.getReqClass() == classNum)  
 counter++;  
 }  
 **return** counter;  
 }  
  
 **private void** evalMath() {  
 Requirement nextReq;  
 **if** (!**S2Queue**.isEmpty()) {  
 nextReq = **S2Queue**.get(0);  
 nextReq.setProcStartTime(**currTime**);  
 **if** (nextReq.getReqClass() == 2)  
 **EventsList**.add(**new** Event(**currTime** + ***MU23***, EventTypesEnum.***FINISH\_SERVE\_IN\_PP***));  
 **else  
 EventsList**.add(**new** Event(**currTime** + ***MU23***, EventTypesEnum.***FINISH\_SERVE\_CLASS\_3***));  
 **isBusyS2** = **true**;  
 } **else  
 isBusyS2** = **false**;  
 }  
  
 */\*\*  
 \* Получение ближайшего события.  
 \*  
 \** ***@param eventsList*** *- текущий набор событий для выбора  
 \* события с минимальным моментом наступления  
 \*/* **private** Event getMinTimeEvent(Set<Event> eventsList) {  
 Event minTimeEvent = **null**;  
 **double** mintime = **tMod** \* 2;  
 **double** eTime;  
 **for** (Event event : eventsList) {  
 **if** ((eTime = event.getTime()) < mintime) {  
 minTimeEvent = event;  
 mintime = eTime;  
 }  
 }  
 **return** minTimeEvent;  
 }  
  
 */\*\*  
 \* Заполняет ассоциативный массив для каждого из k требований в очереди.  
 \*  
 \** ***@param systemType*** *- значения типа очереди  
 \* для аккумулирования суммы длительностей пребывания  
 \** ***@param queSize*** *- текущий размер очереди  
 \** ***@param interval*** *- интервал для суммирования с предыдущими  
 \*/* **private void** mTimeAccumulator(String systemType, **int** queSize, **double** interval) {  
 **if** (systemType.equals(**"MP\_1"**)) {  
 **if** (!**S11KMap**.containsKey(queSize)) {  
 **S11KMap**.put(queSize, interval);  
 } **else** {  
 **S11KMap**.put(queSize, **S11KMap**.get(queSize) + interval);  
 }  
 } **else if** (systemType.equals(**"MP\_2"**)) {  
 **if** (!**S12KMap**.containsKey(queSize)) {  
 **S12KMap**.put(queSize, interval);  
 } **else** {  
 **S12KMap**.put(queSize, **S12KMap**.get(queSize) + interval);  
 }  
 }**else if** (systemType.equals(**"PP\_2"**)) {  
 **if** (!**S22KMap**.containsKey(queSize)) {  
 **S22KMap**.put(queSize, interval);  
 } **else** {  
 **S22KMap**.put(queSize, **S22KMap**.get(queSize) + interval);  
 }  
 } **else** {  
 **if** (!**S23KMap**.containsKey(queSize)) {  
 **S23KMap**.put(queSize, interval);  
 } **else** {  
 **S23KMap**.put(queSize, **S23KMap**.get(queSize) + interval);  
 }  
 }  
 }  
  
 */\*\*  
 \* Вычисление мат ожидания числа требований.  
 \*  
 \** ***@param kMap*** *- ассоциативный массив сумм промежутков времени для k  
 \*/* **private double** calculateMCount(Map<Integer, Double> kMap) {  
 **double** mCount = 0;  
  
 **for** (Integer k : kMap.keySet()) {  
 mCount += ((k \* kMap.get(k)) / **tMod**);  
 }  
 **return** mCount;  
 }  
  
 */\*\*  
 \* Генерация промежутков времени между типами событий.  
 \*  
 \** ***@param lambda*** *параметр показательного распределения  
 \*/* **private double** genExp(**double** lambda) {  
 **return** -(Math.*log*(**random**.nextDouble()) / lambda);  
 }  
  
 **public** IModel(**double** l1, FileWriter writer, **int** cellNum, **int** eRowNum) **throws** IOException {  
 **this**.*writer* = writer;  
 **this**.**cellNum** = cellNum;  
 **this**.**eRowNum** = eRowNum;  
 setL0(l1);  
 emulate();  
 }  
}  
  
**public class** Requirement {  
 **private double procStartTime**; *//фактическое время начала обслуживания* **private double releaseTime**; *//время завершения обслуживания* **private double procTime**; *//время обслуживания* **private double generationTime**; *//время поступления требования в систему* **private int reqClass**;  
 **public** Requirement(**double** generationTime, **int** reqClass) {  
 **this**.**reqClass** = reqClass;  
 **this**.**generationTime** = generationTime;  
 }  
 **public** Requirement(**int** reqClass) {  
 **this**.**reqClass** = reqClass;  
 }  
 **public void** setProcStartTime(**double** procStartTime) {  
 **this**.**procStartTime** = procStartTime;  
 }  
 **public void** setReleaseTime(**double** releaseTime) {  
 **this**.**releaseTime** = releaseTime;  
 }  
 **public double** getGenerationTime() {  
 **return generationTime**;  
 }  
 **public int** getReqClass() {  
 **return reqClass**;  
 }  
 @Override  
 **public** String toString() {  
 **return "Class: "** + **reqClass** + **" Generated at: "** + String.*format*(**"%.10f"**, **generationTime**)   
 + **" Process started: "** + String.*format*(**"%.10f"**, **procStartTime**)   
 + **" Processing time: "** + String.*format*(**"%.10f"**, **procTime**)   
 + **" Release time: "** + String.*format*(**"%.10f"**, **releaseTime**);  
 }  
}  
  
**import** com.sgu.magistr.analiticsmodel.AModel;  
**import** com.sgu.magistr.imitationmodel.IModel;  
**import** java.io.FileWriter;  
**import** java.io.IOException;  
  
**public class** InitExperiment {  
  
 **private static final double**[] ***l1Array*** = {0.01, 0.05, 0.1, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0};  
  
 **public static void** main(String[] args) **throws** IOException {  
 FileWriter writer = **new** FileWriter(**"RES.txt"**, **false**);  
 writer.write(**"Данные аналитической модели: \n"**);  
 **for** (**int** i = 0; i < ***l1Array***.**length**; i++) {  
 **new** AModel(***l1Array***[i], writer, i + 1);  
 }  
 writer.write(**"Данные имитационной модели: \n"**);  
 **for** (**int** i = 0; i < 10; i++)  
 **for** (**int** j = 0; j < ***l1Array***.**length**; j++) {  
 **new** IModel(***l1Array***[j], writer, j + 1, i);  
 }  
 writer.close();  
 }  
}

**package** com.sgu.magistr;  
**import** org.apache.poi.xssf.usermodel.XSSFCell;  
**import** org.apache.poi.xssf.usermodel.XSSFSheet;  
**import** org.apache.poi.xssf.usermodel.XSSFWorkbook;  
  
**import** java.io.\*;  
  
**public class** StatsGraphBuild {  
  
 **static** File *file*;  
 **static** InputStream *inputStream*;  
 **static** XSSFWorkbook *workbook*;  
 **static** XSSFSheet *sheet*;  
 **static** XSSFCell *cell*;  
 **static** {  
 *file* = **new** File(**"CharacteristicsDependancy.xlsx"**);  
 **try** {  
 *inputStream* = **new** FileInputStream(*file*);  
 *workbook* = **new** XSSFWorkbook(*inputStream*);  
 } **catch** (FileNotFoundException e) {  
 e.printStackTrace();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 *sheet* = *workbook*.getSheetAt(0);  
 }  
 **public static void** updateCell(**int** rowNum, **int** cellNum, **double** value) **throws** IOException {  
 *cell* = *sheet*.getRow(rowNum).createCell(cellNum);  
 *cell*.setCellValue(value);  
 *inputStream*.close();  
 FileOutputStream out = **new** FileOutputStream(*file*);  
 *workbook*.write(out);  
 out.close();  
 }  
}

# ПРИЛОЖЕНИЕ Б

## **РЕЗУЛЬТАТЫ МОДЕЛИРОВАНИЯ БЕСПРОВОДНОЙ СЕНСОРНОЙ СЕТИ**

Таблица 1 — Данные аналитической модели

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *λ*01 |  |  |  |  |  |  |  |  |  |
| 0,01 | 5E-08 | 2,5E-05 | 0,0004 | 0,000401 | 0,000826 | 5E-06 | 2,5E-06 | 4E-05 | 4,01E-05 |
| 0,05 | 2,5E-07 | 0,000125 | 0,002008 | 0,00201 | 0,004144 | 5E-06 | 2,5E-06 | 4,01E-05 | 4,02E-05 |
| 0,1 | 5E-07 | 0,00025 | 0,004032 | 0,004036 | 0,00832 | 5E-06 | 2,5E-06 | 4,03E-05 | 4,04E-05 |
| 0,5 | 2,5E-06 | 0,001252 | 0,020834 | 0,020855 | 0,042945 | 5,01E-06 | 2,5E-06 | 4,16E-05 | 4,17E-05 |
| 1 | 5,01E-06 | 0,002506 | 0,04348 | 0,043524 | 0,08952 | 5,01E-06 | 2,51E-06 | 4,34E-05 | 4,35E-05 |
| 2 | 1,01E-05 | 0,005025 | 0,095247 | 0,095342 | 0,195635 | 5,03E-06 | 2,51E-06 | 4,76E-05 | 4,77E-05 |
| 3 | 1,51E-05 | 0,007557 | 0,15792 | 0,158078 | 0,323584 | 5,04E-06 | 2,52E-06 | 5,26E-05 | 5,27E-05 |
| 4 | 2,02E-05 | 0,010101 | 0,235349 | 0,235585 | 0,481076 | 5,05E-06 | 2,53E-06 | 5,88E-05 | 5,89E-05 |
| 5 | 2,53E-05 | 0,012659 | 0,333444 | 0,333778 | 0,679932 | 5,06E-06 | 2,53E-06 | 6,66E-05 | 6,68E-05 |
| 6 | 3,05E-05 | 0,015229 | 0,461752 | 0,462213 | 0,939255 | 5,08E-06 | 2,54E-06 | 7,69E-05 | 7,7E-05 |
| 7 | 3,56E-05 | 0,017812 | 0,636769 | 0,637406 | 1,292058 | 5,09E-06 | 2,54E-06 | 9,09E-05 | 9,11E-05 |
| 8 | 4,08E-05 | 0,020409 | 0,889679 | 0,890569 | 1,800739 | 5,1E-06 | 2,55E-06 | 0,000111 | 0,000111 |
| 9 | 4,6E-05 | 0,023019 | 1,287338 | 1,288625 | 2,599074 | 5,12E-06 | 2,56E-06 | 0,000143 | 0,000143 |
| 10 | 5,13E-05 | 0,025642 | 2,001567 | 2,003568 | 4,03088 | 5,13E-06 | 2,56E-06 | 0,0002 | 0,0002 |

Таблица 2 — Данные имитационной модели

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *λ*01 |  |  |  |  |  |  |  |  |  |
| 0,01 | 5,1E-08 | 2,5E-05 | 0,0004 | 0,0004 | 0,000825 | 4,95E-06 | 2,5E-06 | 4E-05 | 4E-05 |
| 0,05 | 2,5E-07 | 0,000125 | 0,002 | 0,002002 | 0,004127 | 4,95E-06 | 2,5E-06 | 4,01E-05 | 4,02E-05 |
| 0,1 | 5E-07 | 0,00025 | 0,004003 | 0,004007 | 0,00826 | 4,95E-06 | 2,5E-06 | 4,02E-05 | 4,03E-05 |
| 0,5 | 2,7E-06 | 0,001249 | 0,020173 | 0,020208 | 0,041632 | 4,95E-06 | 2,5E-06 | 4,09E-05 | 4,16E-05 |
| 1 | 5E-06 | 0,002504 | 0,040835 | 0,040962 | 0,084306 | 4,95E-06 | 2,5E-06 | 4,18E-05 | 4,33E-05 |
| 2 | 9,5E-06 | 0,005007 | 0,083647 | 0,08384 | 0,172504 | 4,95E-06 | 2,5E-06 | 4,41E-05 | 4,7E-05 |
| 3 | 1,6E-05 | 0,007503 | 0,129534 | 0,129565 | 0,266615 | 4,96E-06 | 2,5E-06 | 4,71E-05 | 5,14E-05 |
| 4 | 2,1E-05 | 0,009999 | 0,179509 | 0,178418 | 0,367947 | 4,96E-06 | 2,5E-06 | 5,08E-05 | 5,62E-05 |
| 5 | 2,5E-05 | 0,012518 | 0,235654 | 0,232987 | 0,481186 | 4,98E-06 | 2,5E-06 | 5,53E-05 | 6,19E-05 |
| 6 | 2,9E-05 | 0,014932 | 0,299858 | 0,292021 | 0,606841 | 4,96E-06 | 2,5E-06 | 6,16E-05 | 6,91E-05 |
| 7 | 3,4E-05 | 0,017494 | 0,387658 | 0,370941 | 0,776132 | 4,97E-06 | 2,5E-06 | 7,12E-05 | 8,01E-05 |
| 8 | 4E-05 | 0,019928 | 0,500487 | 0,465633 | 0,98609 | 4,97E-06 | 2,5E-06 | 8,48E-05 | 9,45E-05 |
| 9 | 4,6E-05 | 0,022648 | 0,684022 | 0,611298 | 1,31801 | 4,98E-06 | 2,5E-06 | 0,000107 | 0,000118 |
| 10 | 5E-05 | 0,025007 | 0,943368 | 0,81326 | 1,781685 | 4,98E-06 | 2,5E-06 | 0,000141 | 0,000152 |

Таблица 3 — Данные модели ненадежной сети

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *λ*01 |  |  |  |  |  |  |  |  |  |
| 0,01 | 2E-08 | 9,84E-06 | 0,000157 | 0,000158 | 0,000325 | 1,97E-06 | 9,84E-07 | 1,57E-05 | 3,15E-05 |
| 0,05 | 9,8E-08 | 4,92E-05 | 0,00079 | 0,000791 | 0,00163 | 1,97E-06 | 9,84E-07 | 1,58E-05 | 3,16E-05 |
| 0,1 | 2E-07 | 9,84E-05 | 0,001586 | 0,001588 | 0,003273 | 1,97E-06 | 9,84E-07 | 1,58E-05 | 3,18E-05 |
| 0,5 | 9,8E-07 | 0,000492 | 0,008196 | 0,008204 | 0,016894 | 1,97E-06 | 9,85E-07 | 1,64E-05 | 3,28E-05 |
| 1 | 2E-06 | 0,000986 | 0,017105 | 0,017122 | 0,035217 | 1,97E-06 | 9,86E-07 | 1,71E-05 | 3,42E-05 |
| 2 | 4E-06 | 0,001977 | 0,03747 | 0,037507 | 0,076961 | 1,98E-06 | 9,88E-07 | 1,87E-05 | 3,75E-05 |
| 3 | 5,9E-06 | 0,002973 | 0,062125 | 0,062187 | 0,127296 | 1,98E-06 | 9,91E-07 | 2,07E-05 | 4,15E-05 |
| 4 | 7,9E-06 | 0,003974 | 0,092585 | 0,092678 | 0,189252 | 1,99E-06 | 9,93E-07 | 2,31E-05 | 4,63E-05 |
| 5 | 1E-05 | 0,00498 | 0,131175 | 0,131306 | 0,267481 | 1,99E-06 | 9,96E-07 | 2,62E-05 | 5,25E-05 |
| 6 | 1,2E-05 | 0,005991 | 0,18165 | 0,181832 | 0,369497 | 2E-06 | 9,98E-07 | 3,02E-05 | 6,06E-05 |
| 7 | 1,4E-05 | 0,007007 | 0,250501 | 0,250751 | 0,508287 | 2E-06 | 1E-06 | 3,58E-05 | 7,16E-05 |
| 8 | 1,6E-05 | 0,008029 | 0,349994 | 0,350344 | 0,708399 | 2,01E-06 | 1E-06 | 4,37E-05 | 8,76E-05 |
| 9 | 1,8E-05 | 0,009056 | 0,50643 | 0,506937 | 1,022459 | 2,01E-06 | 1,01E-06 | 5,62E-05 | 0,000113 |
| 10 | 2E-05 | 0,010088 | 0,787403 | 0,788191 | 1,585722 | 2,02E-06 | 1,01E-06 | 7,87E-05 | 0,000158 |