

**Практикум «Моделирование систем в среде AnyLogic 6.4.1»**

**Часть 1**

**МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ  
ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ(МАДИ)**

**©Мезенцев К.Н**

**Доцент кафедры автоматизированных систем управления**

## Оглавление

Введение .....	4
1. Дискретно – событийное моделирование .....	5
1.1. Моделирование колебательного процесса .....	5
1.2. Контрольные задания .....	18
Контрольные вопросы .....	20
2. Анимация презентации.....	21
2.1. Движение объекта по заданной траектории.....	21
2.2. Контрольное задание «Движение по параболе» .....	29
2.3. Модель «Жизнь» .....	30
2.4. Контрольное задание «Фигуры» .....	37
Контрольные вопросы .....	38
3. Реагирующие системы .....	38
3.1. Модель светофора для управления движением.....	38
3.2. Модель пешеходного перехода .....	44
3.3. Контрольное задание “Переход, управляемый пешеходом” .....	47
3.4. Контрольное задание “Модель кодового замка” .....	48
3.5. Модель трех разрядного счетчика .....	51
3.6. Контрольное задание «Запуск события по истечении контрольного времени» .....	55
Контрольные вопросы .....	56
4. Системная динамика .....	57
4.1. Модель реализации продукта по Бассу.....	57
4.2. Контрольное задание «Учет повторных покупок» .....	61
4.3. Контрольное задание «Модель распространения эпидемии» ...	62
4.4. Взаимодействие активных классов.....	63

4.7. Контрольное задание «Визуализация модели динамики численности населения» .....	72
5. Агентное моделирование .....	74
5.4. Контрольное задание «Учет повторных покупок агентами» .....	81
Контрольные вопросы .....	82
6 Задания для самостоятельной работы .....	82
Приложение .....	88
Классы Java .....	88
Типы данных Java .....	91
Операции Java .....	93
Арифметические операции языка .....	93
Управляющие операторы .....	95
Массивы и их задание .....	98
Обработка строк .....	98
Класс Math. Математические функции .....	100
Обработка исключительных ситуаций .....	101
Цвет и его кодирование .....	102
Элементы управления и фигуры презентации .....	103
Список литературы .....	109

## Введение

Практикум по дисциплине «Моделирование систем» предназначен для освоения технологий моделирований различных объектов с помощью программного продукта AnyLogic версии 6.4.1.

В первой части практикума изучаются следующие технологии моделирования:

- дискретно – событийное моделирование;
- моделирование путем использования метода системной динамики;
- агентное моделирование.

Практикум состоит из четырех глав.

В первой главе рассматривается технология построения дискретно – событийных моделей физических процессов.

Во второй главе изучаются возможности наглядного представления работы моделей и особенности использования языка Java.

Третья глава практикума преследует цель изучения технологии использования конечных автоматов – стейтчартов в моделях. В этой главе изучается так же технология обмена сообщениями и управления событиями.

Четвертая часть практикума служит для изучения технологии агентного моделирования и моделирования по методу системной динамики.

Каждая глава практикума содержит несколько заданий с необходимыми указаниями для их выполнения. Для контроля предлагается выполнить дополнительные задания.

В практикуме содержится раздел, в котором даются задания для самостоятельной работы.

В приложении приводятся сведения по объектно-ориентированному языку программирования Java в объеме необходимом для выполнения заданий практикума. В приложении приводятся также сведения о методах объектов AnyLogic, которые используются при построении моделей заданий практикума.

## 1. Дискретно – событийное моделирование

### 1.1. Моделирование колебательного процесса

Постановка задачи. Построить модель для исследования процесса незатухающих гармонических колебаний. Колебательный процесс описывается дискретным уравнением гармонических колебаний следующего вида:

$$x[n] = a \sin \varphi$$

$$\varphi = \omega t[n] + \varphi_0$$

где:

$\varphi$  – фаза колебаний;

$\varphi_0$  – начальная фаза;

$\omega$  – угловая частота;

$t[n]$  – дискретное время;

$a$  – амплитуда колебаний.

Дискретное время вычисляется по формуле:

$$t[n] = T_0 \cdot n, \text{ где } n = 0, \dots, \infty$$

здесь  $T_0$  – период дискретизации модели

### Построение модели

Чтобы создать новую модель нужно выбрать команду меню программы AnyLogic: «Файл» > «Создать» > «Модель». Нужно выбрать название файла модели и затем указать, что модель создается с нуля.

Виды окон показаны на рисунках 1.1 и 1.2.

**Новая модель**

Новая модель  
Создание новой модели

Имя модели: Model2

Местоположение: C:\Documents and Settings\worker\Models Выбрать...

Java пакет: model2

Будет создана следующая модель:

C:\Documents and Settings\worker\Models\Model2\Model2.alp

< Назад Далее > Готово Отмена

Рис.1.1. Окно первой фазы создания новой модели

**Новая модель**

Новая модель  
Выберите, хотите ли Вы использовать один из шаблонов моделей или начать моделирование "с нуля"

☒ Начать создание модели "с нуля" (будет создан чистый холст презентации).

☐ Использовать шаблон модели

Выберите метод моделирования:

< Назад Далее > Готово Отмена

Рис.1.2. Окно выбора типа модели

При моделировании в среде AnyLogic главным объектом модели является корневой объект Main – формируется автоматически. Он

может быть переименован при создании модели. Такой объект при моделировании так же называют корневым (root).

Модель строится в графическом поле этого объекта с помощью соответствующих инструментов палитры. Разместим в графическом поле объекта переменные и параметры. При моделировании нужно определить, какие значения модели будут представлены параметрами, а какие переменными. Переменные - изменяющиеся значения в процессе моделирования. В качестве параметров выбирают значения, которые остаются постоянными в течение периода моделирования, либо они могут изменяться исследователем модели для определения их влияния на рассчитываемые значения переменных.

В таблице 1.1. Приводится классификация переменных и параметров модели.

Таблица №1.1. Модельные характеристики

№	Характеристика модели	Тип характеристики
1	Амплитуда колебаний ( $a$ )	Параметр
2	Частота колебаний ( $w$ )	Параметр
3	Период дискретизации ( $t_0$ )	Параметр
4	Начальная фаза ( $\varphi_0$ )	Параметр
5	Дискретное время ( $t_n$ )	Переменная
6	Текущее значение величины колебательного процесса ( $x_n$ )	Переменная
7	Счетчик дискретных шагов процесса ( $n$ )	Переменная

При моделировании будем считать, что начальная фаза колебаний  $\varphi_0$  равна нулю. Период дискретизации  $t_0$  примем равным 0.1 секунде.

Для размещения в графическом поле объекта Main нужных элементов следует использовать вкладку палитры «Основная». Используя технологию Drag & Drop, следует перенести в поле объекта пиктограмму «Параметр» и пиктограмму «Простая переменная» и разместить их, так как это показано на рисунке 1.3.

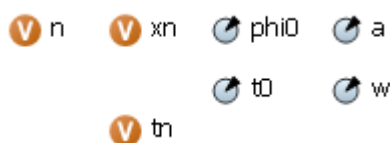


Рис.1.3. Переменные и параметры модели

При работе в среде AnyLogic разработчик может убирать панели, либо отображать их по своему усмотрению. Для выполнения такого действия следует использовать команду меню программы «Панель > Имя нужной панели».

Для задания свойств переменным и параметрам следует использовать панель «Свойства». При работе с переменными следует задать следующие основные свойства: «Имя», «Тип» и «Начальное значение». Для переменных задается «Значение по умолчанию» вместо свойства «Начальное значение» для параметров. В окне свойств нужно использовать для задания этих значений вкладку «Основные». Зададим имена переменным и параметрам в соответствии с рисунком 1.1, в таблице 1.2 приводится описание свойств.

Таблица 1.2. Переменные и параметры

№	Имя	Начальное значение/Значение по умолчанию	Тип
1	a	1	double
2	w	0.25	double
3	t0	0.1	double
4	phi0	0	double
5	tn	0	double
6	xn	0	double
7	n	0	int

Чтобы активизировать процесс моделирование в среде AnyLogic нужно использовать технологию управления процессом моделирования с помощью событий. В поле класса Main следует перенести из раздела палитры «Основная» пиктограмму «Событие».



Для решения задачи моделирования следует настроить свойства этого элемента в соответствии с таблицей 1.3. Настройка свойств выполняется на вкладке «Основные» панели «Свойства».

В результате вид графического поля класса Main примет вид, показанный на рисунке 1.4, настройка события должна соответствовать таблице 1.3.



Рис.1.4. Размещение события

Таблица №1.3. Элементы класса Main

№	Свойство	Значение
1	Имя	event
2	Тип события	По таймауту
3	Режим	Циклический
4	Время первого срабатывания	time()
5	Период	t0
6	Действие	getX()

Функция `time()` возвращает текущее модельное время.

Свойство «Действие» используется для задания программного кода, который будет выполняться при выполнении события. В среде моделирования AnyLogic программный код пишется на объектно-ориентированном языке программирования Java.

Для нашей модели напомним функцию `getX`, обращение к которой выполняется при выполнении события. С помощью этой функции будем вычислять значение для вычисления переменной `xn`, которая является выходной в данной модели.

Для этого следует сделать щелчок левой кнопкой мыши в любом месте поля объекта Main. В окне свойств объекта следует ввести код нашей функции. Код записывается в разделе «Дополнительные» и должен иметь вид, который показан на рисунке 1.5.

Дополнительный код класса:

```
public void getX() {
    n++; // шаг
    tn=t0*n; // дискретное время
    double phi;
    phi=w*tn+phi0; // аргумент
    xn=a*Math.sin(phi);
}
```

Рис.1.5. Код функции

Код функции, как и любой другой код, написанный для модели AnyLogic, пишется на языке программирования Java. При написании кода для класса требуется его записать в виде функции. Язык программирования Java был создан как дальнейшее развитие языка программирования C++. От языка C++ были заимствован синтаксис и многие управляющие конструкции. В отличие от языка C++ язык Java является полностью объектно-ориентированным. В приложении приведены сведения о языке Java необходимые при построении моделей практикума.

При написании кода следует иметь в виду, что для вызова встроенных математических функций Java, используется статический класс `Math` и его методы. Методы этого класса реализуют различные математические функции. Поскольку этот класс статический для вызова методов не нужно получать экземпляр этого класса. В коде вызывается метод `sin()` для вычисления значения синуса. Подробнее с методами класса `Math` можно ознакомиться в приложении.

### Запуск модели



Перед запуском модели нужно настроить эксперимент. В левой части окна программы выводится дерево проекта, на котором

расположен основной эксперимент *Simulation*. Этот объект модели создается автоматически при ее формировании.

Основные свойства эксперимента настраиваются на его панели свойств с помощью вкладки «Модельное время». Нужно выполнить следующие настройки:

«Единицы модельного времени»=секунды

«Остановить»=Нет

Протестировать созданную модель можно с помощью с помощью кнопки «Запуск»  панели управления AnyLogic. Рекомендуется перед запуском выполнить предварительную компиляцию модели с помощью кнопки «Построить модель» .

Если модель содержит ошибки, то их описание приводится в панели «Ошибки». Двойной щелчок левой кнопкой мыши на описании ошибки позволяет перейти в то место модели, которая ее вызвала. Если модель скомпилировано без ошибок, то выводится окно с кнопкой запуска модели. Пример такого окна показан на рисунке 1.6.

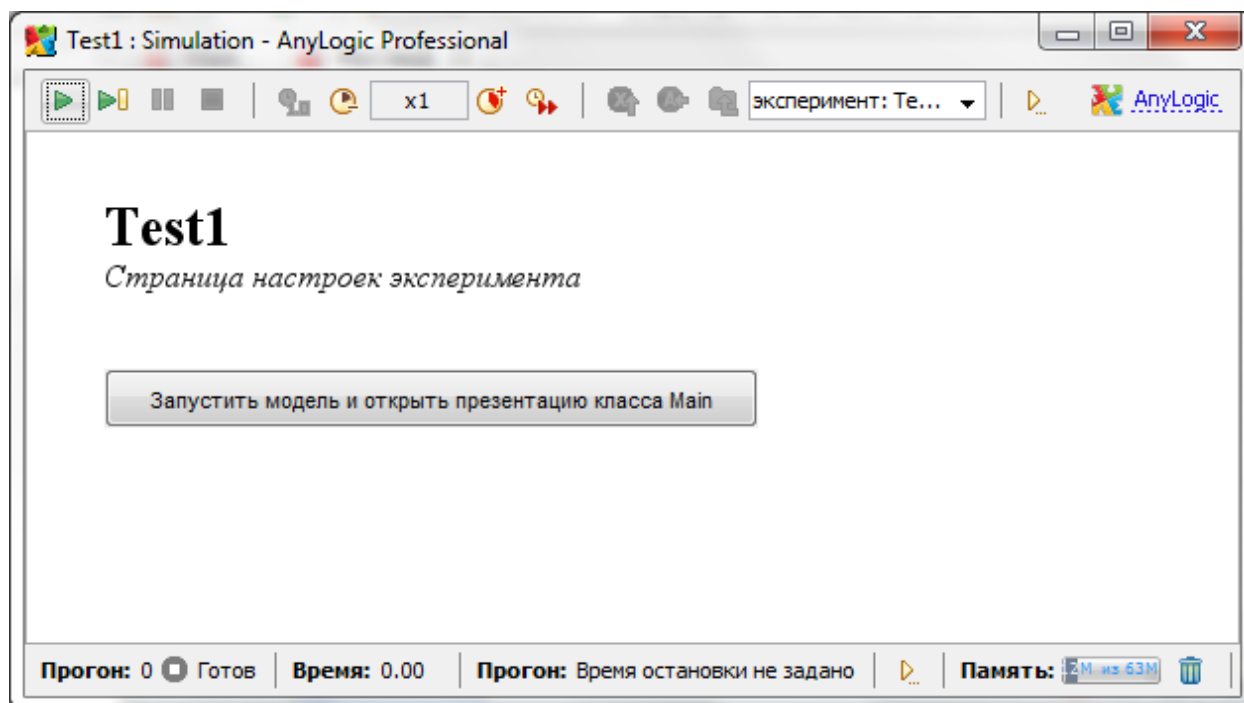


Рис.1.6. Окна запуска модели

После активизации кнопки «Запустить модель и открыть презентацию класса Main» открывается окно выполнения модели. Вид модели в окне показан на рисунке 1.7.

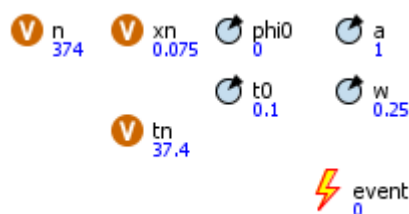


Рис.1.7. Фрагмент окна выполнения модели

Для управления процессом исполнения моделью служит набор кнопок панелей управления, показанный на рисунке 1.8. Значение кнопок слева на право:

1. Запустить с текущего состояния;
2. Выполнить шаг;
3. Приостановить исполнение;
4. Прекратить выполнение
5. Выбрать режим реального времени;
6. Замедлить;
7. Ускорить;
8. Реальное/Виртуальное время.

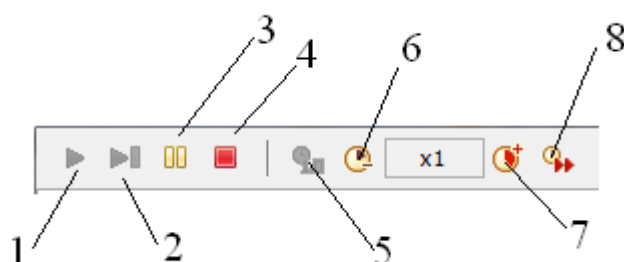


Рис.1.8. Основные кнопки управления

Примечание: По умолчанию модель выполняется в режиме реального времени. При переключении на виртуальное время модель обрабатывается с максимальной скоростью, которая возможна в данной вычислительной системе.

В режиме исполнения модели можно использовать «инспекты». Это специальные окна, которые отображают текущее значение элементов модели в числовом или графическом виде.

Для получения окна «инспекта» нужно выполнить щелчок левой кнопкой мыши на имени элемента модели. В правом верхнем углу окна находятся кнопки для переключения вида представления значения элемента. На рисунке 1.9 показан вид окна модели, в котором выводится график изменения значения переменной  $x_n$ .

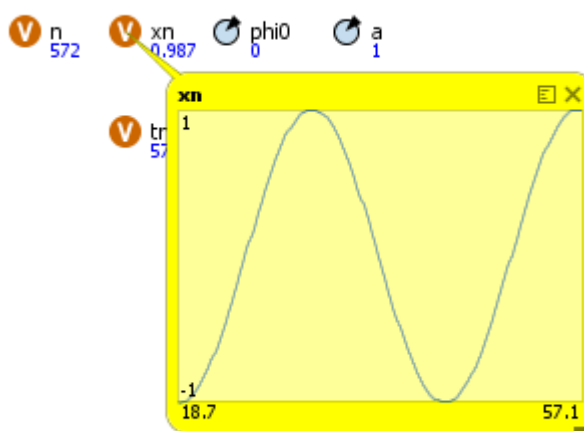


Рис.1.9. Просмотр изменения значения  $x_n$

### Размещение графика

Для повышения информативности модели разместим в модель график, который будет автоматически выводиться в поле модели.

Размещение графика выполняется путем использования вкладки палитры «Статистика». В поле модели Main нужно разместить пиктограмму «График». В окне свойств графика на вкладке «Основные» нужно нажать кнопку «Добавить элемент данных». Переключатель «Значение» должен быть включен затем следует указать имена переменных значения которых будут выводиться по оси X и Y. Далее задается заголовок для графика (см. рисунок 1.10).

Значение Набор данных Заголовок: Колебательный процесс

Значение по оси X: tn

Значение по оси Y: xn

Стиль маркера: —●— Цвет: slateBlue

☒ Рисовать линию Толщина линии: 1 pt 1 Интер

Рис.1.10. Настройка графика

По умолчанию, при выводе значений, на графике показывается 100 последних значений. Для построения более полного графика это число следует увеличить до 500. Свойство «Отображать до»=500, а свойство «Период»= $t_0$ .

Дополнительно можно изменить цвет маркера и цвет линии. Размер графика регулируется с помощью размерных маркеров его полей.

Вид окна модели после запуска показан на рисунке 1.11.

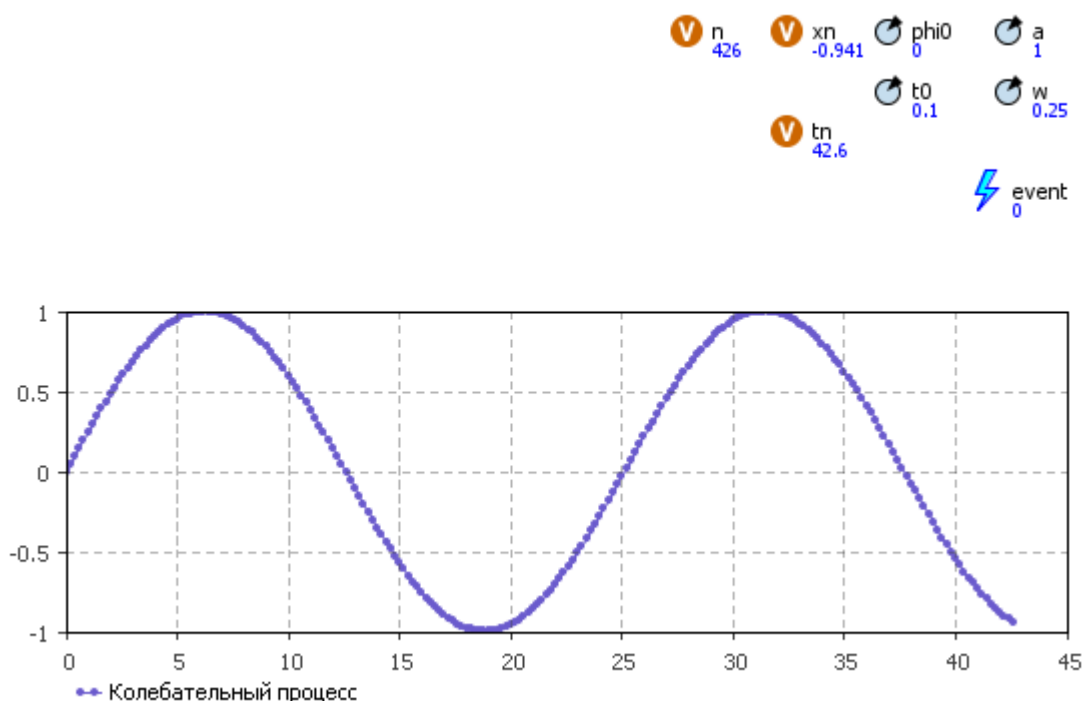


Рис.1.11. Окно модели

## Настройка презентации модели

При обработке модели AnyLogic формирует в ее окне презентацию. Итоговый вид презентации модели показан на рисунке 1.12.

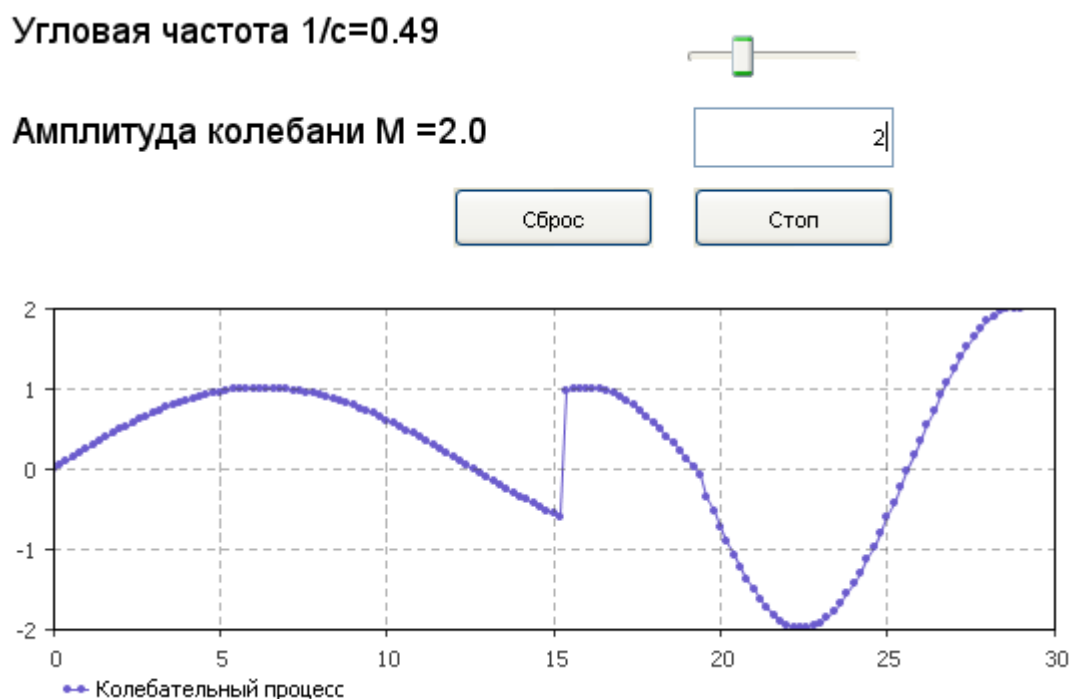


Рис.1.12. Модель колебаний в момент переключения параметров

Сначала уберем изображения переменных и параметров. Для этого выберем в окне класса Main требуемые элементы и на вкладке «Основные» окна свойств, сбросим флажок «На презентации».

Затем разместим элементы управления для изменения параметров модели. Откроем вкладку «Элементы управления» и разместим в поле класса Main два элемента: «Текстовое поле» и «Бегунок». С помощью первого элемента будем изменять значение амплитуды, а с помощью второго значение частоты колебаний. Настройка элементов выполняется на вкладке «Основные» окна свойств.

Значения свойств сведены в таблицу 1. 4.

Таблица 1.4. Переменные и параметры

№	Элемент	Свойство	Значение
1	Текстовое поле	Имя	editboxA
		Связать с	активен, a
		Минимальное значение	1
2	Бегунок	Имя	sliderF
		Связать с	Активен, w
		Минимальное значение	0,25
		Максимальное значение	1

Перейдем на вкладку палитры «Презентация» и разместим в области класса Main два элемента «Текст». Настройка текстовых полей должна соответствовать таблице 1.5.

Таблица 1.5. Текстовые элементы

№	Элемент	Свойство	Значение
1	Текст	Имя	textF
		Шрифт	18 пунктов
2	Текст	Имя	textA
		Шрифт	18 пунктов

В поле «Действие» элемента event нужно ввести код, показанный на рисунке 1.13.

Действие:

```
getX();
String bufer;
bufer="Угловая частота 1/c=";
textF.setText (bufer+w);
bufer="Амплитуда колебани M =";
textA.setText (bufer+a);
```

Рис. 1.13. Код события



Элементы для вывода текстовых значений представляют собой объекты, которые содержат методы. Для размещения текста служит метод `setText(String msg)`, где `msg` – требуемый текст. В языке Java при выполнении операции `+` (соединения) числовой переменной с текстовой она автоматически переводится в текст.

### Размещение командных кнопок

Используя вкладку палитры «Элементы управления» разместим две командные кнопки «Сброс» и «Стоп». Первая командная кнопка будет возвращать элементы управления в исходное состояние, и присваивать значения амплитуде и частоте выбранные по умолчанию. Вторая кнопка служит для остановки расчета выхода модели.

Код для кнопок пишется в поле «Действие» на вкладке «Основные» окна свойств.

Для кнопки «Сброс» код примет вид:

Действие:

```
w=0.25;
a=1;
sliderF.setValue(w);
editboxA.setText(" "+a);
```

Кнопка «Стоп» должна содержать код:

```
event.reset();
```

При работе с кнопками можно использовать метод `setText(String mes)` для изменения текста на кнопке.

Элемент бегунок обладает методом `setValue(double v)`, который позволяет установить определенное значение положению ползунка `v`.

Для работы с циклическими событиями определено два метода:

- `reset()` - останавливает генерацию события;
- `restart()` - возобновляет генерацию.

### Определение областей просмотра

Области просмотра позволяют отдельно просматривать элементы презентации. Назначим две области просмотра. В одной

области будем показывать всю презентацию, а в другой только график процесса.

Откроем вкладку «Презентация» и перенесем элемент «Область просмотра» в область класса Main. Назначим элементу свойства:

Имя= viewAreaSinus

Заголовок=Незатухающие колебания

Выделим график и «вырежем» его в буфер обмена. Используя



кнопку панели инструментов (Области просмотра), выберем область просмотра «Незатухающие колебания» и поместим в нее график из буфера обмена. Разместим еще одну область просмотра со следующими параметрами:


Имя= viewAreaMain

Заголовок=Модель колебаний

Вырежем в буфер обмена все элементы модели кроме графика, перейдем в область просмотра и вставим из буфера выделенные элементы.

Выполним запуск модели, в окне выполнения модели для перехода в области просмотра используется кнопка панели



(Показать область). Если этой кнопки нет, нужно с помощью кнопки  окна презентации модели вывести панель «Вид».

Для программного управления переходом на области просмотра следует использовать метод `navigateTo`. Обращение к методу имеет вид `ИмяОбластиПросмотра.navigateTo()`.

## 1.2. Контрольные задания

№1. Разместить командные кнопки для переключения между областями просмотра. Одну кнопку для просмотра графика (График), а другую для просмотра всей презентации (Назад) см. рисунок 1.14.

Угловая частота  $1/c=0.25$



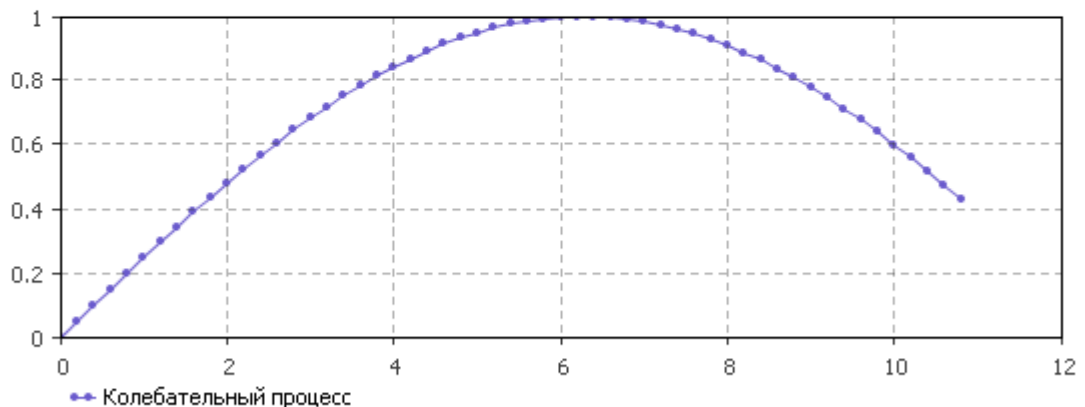
Амплитуда колебани  $M = 1.0$

1.0

График

Сброс

Стоп



Назад

Рис.1.14. Запуск презентации с кнопками управления просмотром

№2. Изменить код для кнопки «Стоп». Кнопка должна работать как переключатель первое нажатие - остановка генерации события модели, повторное нажатие запуск генерации. Процесс повторяется циклически. Надпись на кнопке менять по смыслу ее использования: «Стоп» или «Пуск».

№3. Разместить текстовые элементы для вывода текущего значения частоты колебаний в герцах и периода колебаний. Расчетные формулы:

$$f = \frac{\omega}{2\pi} (\Gamma\text{ц})$$

$$T = \frac{1}{f} (\text{с})$$

№4. Построить новую дискретную модель затухающих колебаний.

Математические зависимости:

$$y[n] = y_0 e^{-\delta t[n]} \sin \varphi$$

$y[n]$  – отклонение выходной величины

$y_0$  – начальная амплитуда

$e$  – основание натурального логарифма

$\delta$  – коэффициент затухания

$\varphi = \omega_{\text{зат}} t [n] + \varphi_0$  – фаза колебаний

$$\omega_{\text{зам}} = \sqrt{\omega_0^2 - \delta^2}$$

Начальная амплитуда 2 метра, коэффициент затухания 0,05.

Начальная фаза 0 радиан, частота собственных колебаний

$$\omega_0 = 0,25 \text{ с}^{-1}.$$

Разместить на презентацию бегунок для изменения коэффициента затухания в диапазоне от 0 до 1.

Разместить на презентацию бегунок для изменения частоты собственных колебаний от 0,25 до 2 1/с.

Разместить на презентацию текстовое поле для задания значения начальной амплитуды.

Разместить на презентацию текстовое поле для задания начальной фазы  $\varphi_0$

Создать области просмотра для вывода графика процесса и всей презентации.

Разместить командные кнопки для выполнения следующих действий:

- Остановки генерации события в модели.
- Возврата элементов управления и параметров исходное состояние.
- Переключения между областями просмотра.

### Контрольные вопросы

1. Какую структуру имеет модель AnyLogic?
2. Какими основными свойствами обладает простая переменная?
3. Для чего используются параметры в моделях AnyLogic?
4. Что такое область просмотра?
5. Как выполняется управление программным кодом областями просмотра?

6. Дайте классификацию событий AnyLogic.
7. Дайте развернутую классификацию событий происходящих по таймауту.
8. Как выполняется настройка простых событий модели?
9. Какие методы системного класса Event используются для программного управления событиями.
10. Чем отличается простое событие от динамического события?
11. Как выполняется планирование и обслуживание динамических событий?
12. Как выполняется настройка простого эксперимента AnyLogic?
13. Что такое активный класс модели и какова его структура?
14. Как выполняется репликация активных классов?
15. Как программируется опрос свойств реплицированных классов?
16. Как выполняется обслуживание программным кодом текстового поля?
17. Как программируется доступ к статическому тексту?
18. Как используется и обслуживается программным кодом элемент слайдер?
19. Как выполняется обслуживание программным кодом командных кнопок?

## 2. Анимация презентации

### 2.1. Движение объекта по заданной траектории

Разработать модель, которая демонстрирует движение сферического объекта синего цвета по заданной траектории. Траектория объекта изменяется по синусоидальному закону. Когда траектория объекта приближается к значению амплитуды траектории с погрешностью равной 0,01 окрасить его в красный цвет, при удалении восстановить синий цвет.

Таблица 2.1.Параметры траектории

№	Параметр	Значение
1	Амплитуда	3 метра
2	Угловая частота	0,5 рад/с

3	Модельное время	500с
---	-----------------	------

Для создания модели нужно сформировать новый проект. Модель создается с нуля. Название модели AnimalBall2.

Используя контекстное меню узла модели AnimalBall2, создайте активный класс Ball (см рисунок 2.1).

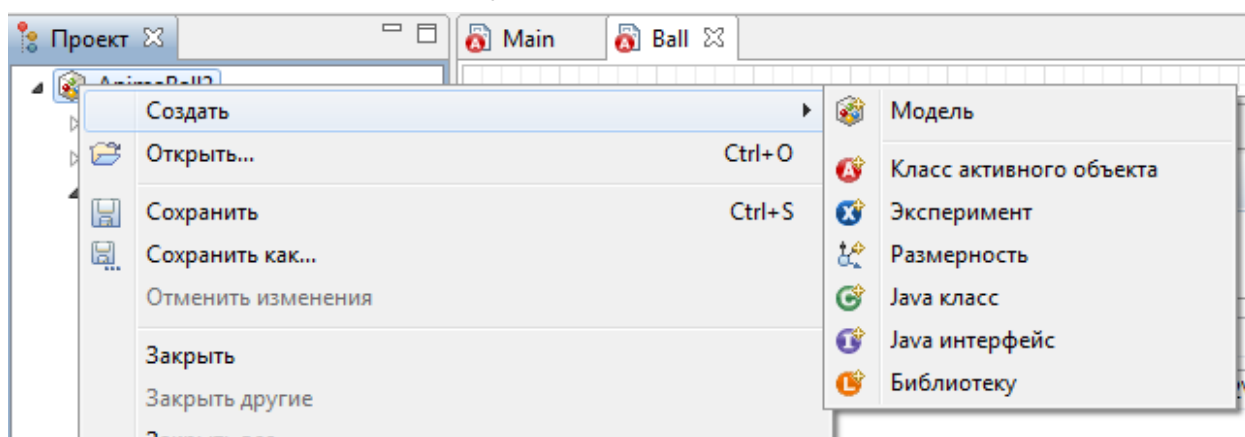


Рис.2.1. Создание нового активного

На рисунке 2.2 приводятся элементы активного объекта Ball

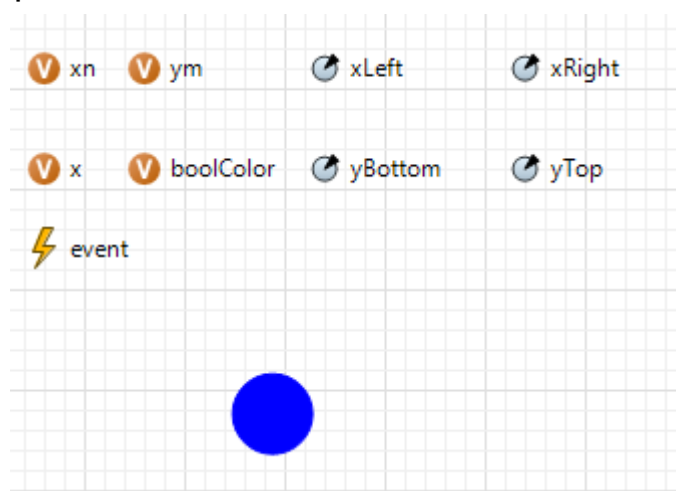


Рис.2.2. Структура активного класса Ball

Для размещения закрашенного круга используйте палитру «Презентация» и элемент «Овал». Настройте это элемент так, как это показано на рисунке 2.3.

☐ oval - Овал

**Основные**

Имя:  ☐ Отображать имя

**Дополнительные**

Цвет заливки:  , [Динамический](#)

Цвет линии:

**Динамические**

**Описание**

Рис.2.3. Свойства активного элемента oval

Элементы активного класса должны соответствовать таблице 2.2.

Таблица 2.2. Активный класс Ball

№	Элемент	Назначение
1	$x_n$	Положение шара по оси X – физическая координата
2	x	мировое - расчетное значение траектории
3	$y_m$	Положение шара по оси Y – физическая координата
4	boolColor	Логическая переменная, начальное значение «ЛОЖЬ»
5	xLeft	Левая координата физическая области вывода по оси X=0
6	yBottom	Левая координата физическая области вывода по оси Y=-3
7	xRight	Правая координата физическая области вывода по оси X=500
8	yTop	Правая координата физическая области вывода по оси X=+3
9	event	Событие

Настройка события должна соответствовать рисунку 2.4.

Имя:  ☒ Отображать имя ☐ Исключить ☐ На верхне

---

Тип события:  Режим:

Время первого срабатывания (абсолютное) ☒  ☐

Период:

---

Действие:

```
double A=3;//Амплитуда
double w=0.5;//Частота рад/с
x=A*Math.sin(w*time());
getX();getY();
//Погрешность и изменение цвета
double eps;
eps=A-Math.abs(x);
if (eps<0.01){
boolColor=true;
}
else{
boolColor=false;
}
```

Рис.2.4. Настройка элемента event

При достижении погрешности по отношению к амплитуде равной 0,01 переменная `boolColor` получает значение «ИСТИНА», что является сигналом для смены цвета шара.

Для того чтобы шар двигался и изменялся его цвет нужно настроить его свойства так как это показано в таблице 2.3.

Таблица 2.3. Динамические свойства

№	Свойство	Значение
1	X	$x_n$
2	Y	$y_m$
3	Цвет заливки	<code>boolColor ?</code> <code>new Color(255,0,0) :</code> <code>new Color(0,0,255)</code>

Чтобы вывод анимации осуществлялся в заданную область, следует выполнить пересчет мировых – расчетных координат объекта в физические координаты. Физические координаты это координаты в пикселах в области вывода объекта при его анимации.



Соответствие между физическими и мировыми координатами показано на рисунке 2.5.

В общем виде формулы пересчета имеют вид:

$$x_n = \frac{x - x_{Left}}{x_{Right} - x_{Left}} \times (n_{Right} - n_{Left}) + n_{Left}$$

$$y_m = \frac{y - y_{Bottom}}{y_{Top} - y_{Bottom}} \times (m_{Top} - m_{Bottom}) + m_{Bottom}$$

Где:  $x_n$  и  $y_m$  физические координаты

Откройте код настройки активного класса Ball и введите код для двух функций, осуществляющих пересчет физических координат в мировые, так как это показано на рисунке 2.6.

Движение сферического объекта будет моделироваться в области объекта Main. Для этого следует в поле этого объекта ввести прямоугольную область с помощью элемента палитры «Презентация» «Прямоугольник». Настройка прямоугольника должна соответствовать рисунку 2.6.

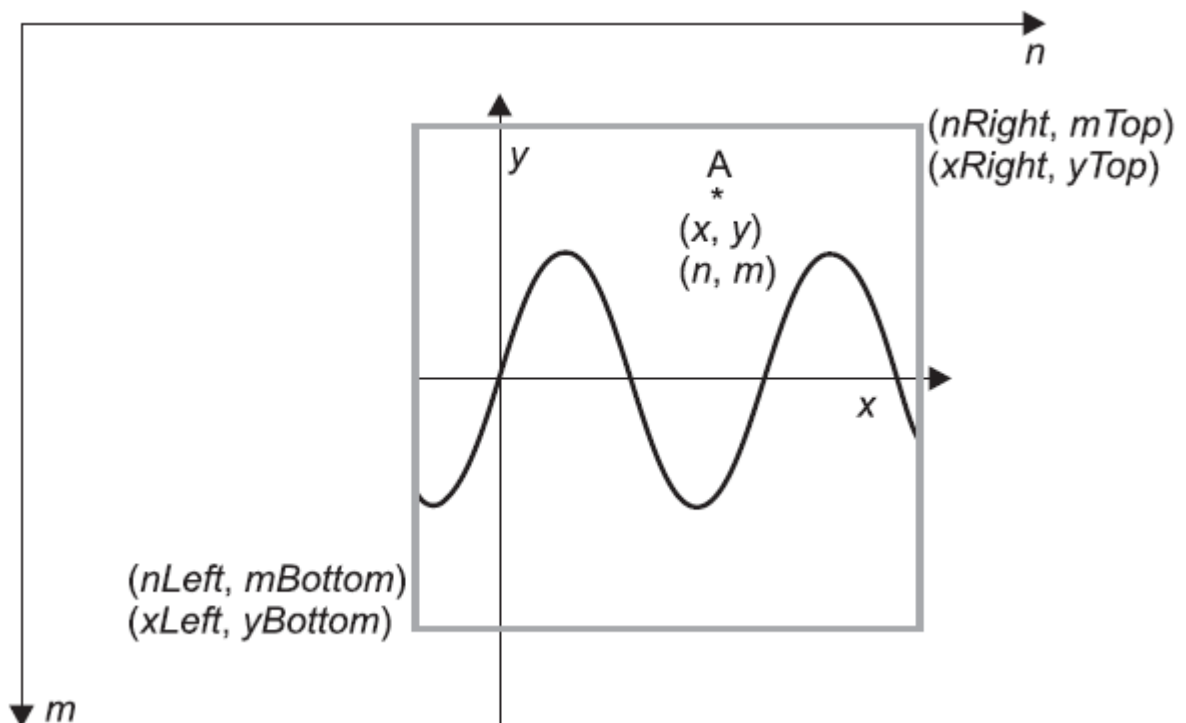


Рис. 2.5. Соответствие физических и мировых координат.

На рисунке 2.5 Префиксы n и m соответствуют физическим координатам, x и y мировым координатам

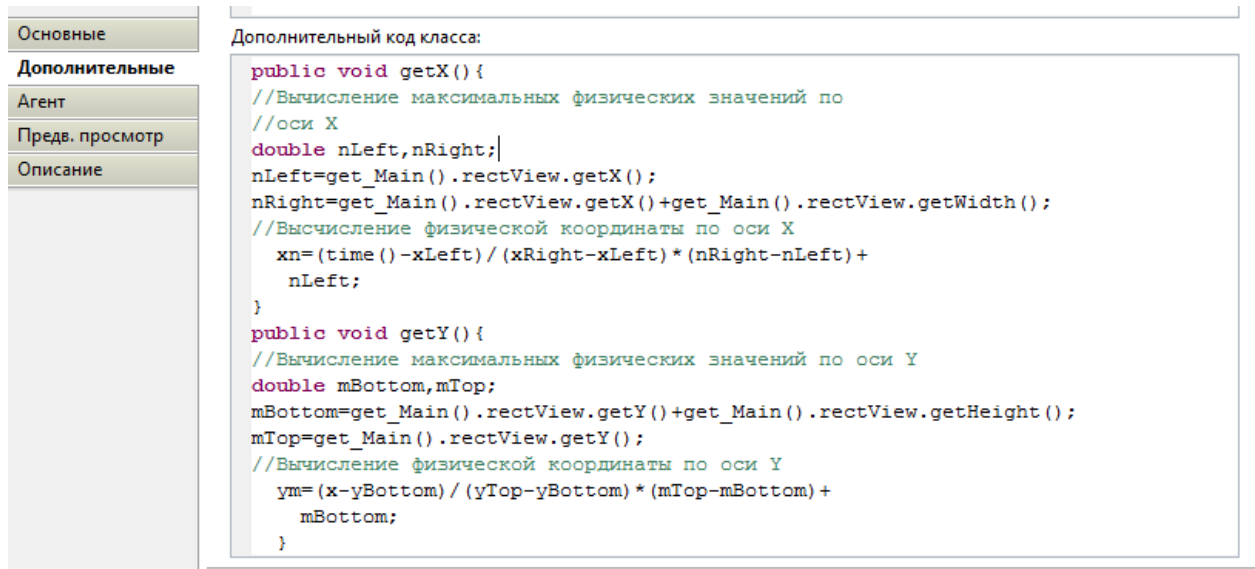


Рис.2.6. Функции пересчета мировых координат в физические

При анимации объект Ball будет двигаться в прямоугольной области объекта Main. Для размещения этой области нужно сделать активным этот объект и используя палитру «Презентация» создать прямоугольную область с помощью элемента «Прямоугольник». Настройки прямоугольной области показаны на рисунке 2.8.

При программировании пересчета мировых координат в физические координаты нужно автоматизировать определение краевых значений физической области вывода (см. рисунок 2.5).

Для обращения к объекту Main, с целью получения экземпляра прямоугольной области, используется оператор

Java get\_Main().Имя\_объекта для нашей модели именем объекта является идентификатор прямоугольной области rectView.

Как любая фигура в технологии Java прямоугольник характеризуется свойствами, показанными на рисунке 2.7

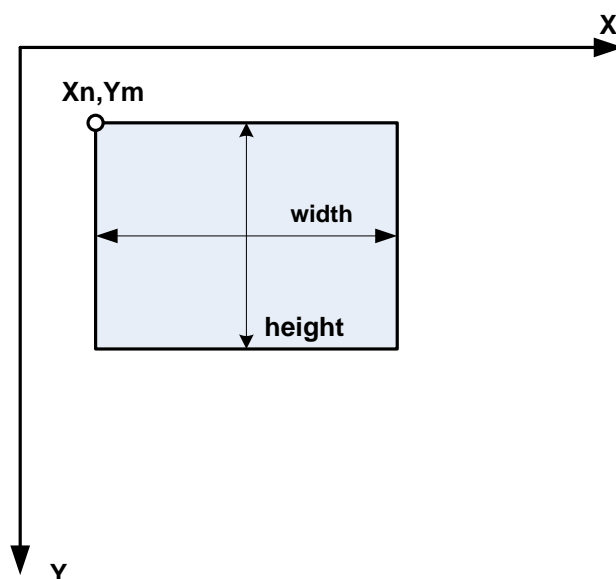


Рис.2.7. Фигура Java и ее свойства

На рисунке 2.7 `width` и `height` – ширина и высота фигуры, `Xn`, `Ym` – координаты точки прорисовки фигуры

Для определения свойств фигуры используются ее методы, показанные в таблице 2.4.

Таблица 2.4. Методы фигуры

№	Метод	Назначение
1	<code>double getWidth()</code>	Получение ширины
2	<code>double getHeight()</code>	Получение высоты
3	<code>double getX()</code>	Точка <code>Xn</code>
4	<code>double getY()</code>	Точка <code>Ym</code>

Основные

Дополнительные

Динамические

Описание

Имя: 
  
Цвет заливки: 
  
Цвет линии:

Основные

Дополнительные

Динамические

Описание

Расположение

X:

Y:

Ширина:

Высота:


Поворот:

☒ Разрешить программное управление

Рис.2.8. Настройка прямоугольника класса Main

После выполнения настройки объектов нужно сделать объект Ball вложенным в объекте Main. Для этого нужно открыть дерево проекта. Оно находится слева. Если его нет, то его нужно вывести командой меню AnyLogic Панель > Проект.

Затем нужно выбрать объект Main двойным щелчком мыши, используя технологию Drag & Drop, перетащить объект Ball в поле объекта Main, так как это показано на рисунке 2.9.

Если вложенный объект скрывается за прямоугольной областью, то следует воспользоваться инструментами панели AnyLogic  для изменения взаимного расположения элементов.

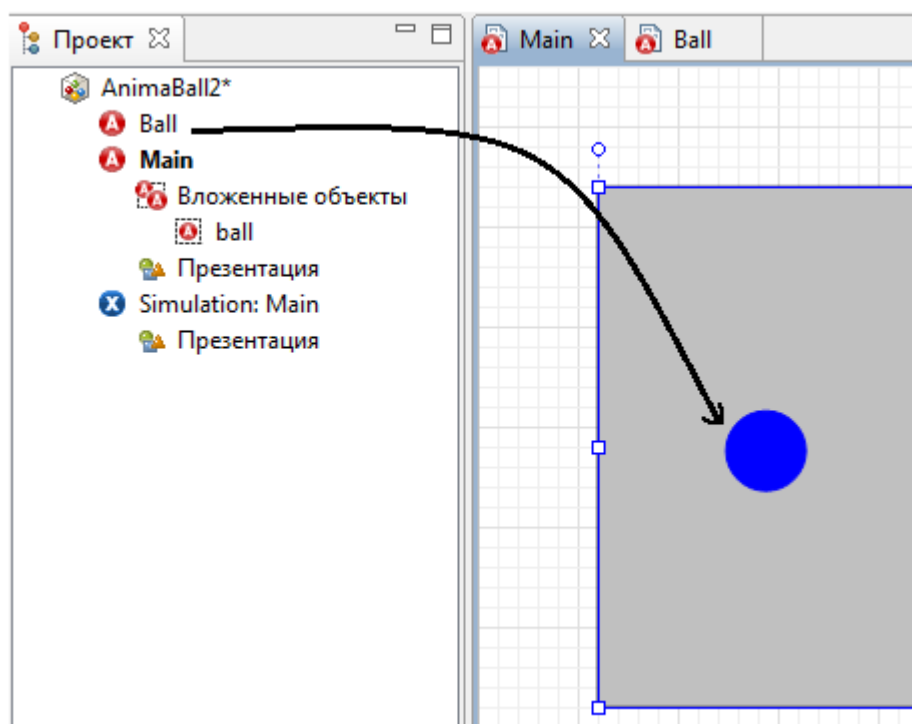


Рис.2.9. Создание вложенного объекта в классе Main

Внимание: После размещения встроенного объекта его нельзя перемещать в поле вывода, так как это сделает невозможным правильный автоматический пересчет мировых координат в физические!

После получения вложенного объекта следует сохранить модель и настроить параметры эксперимента Simulation, так как это показано на рисунке 2.10.

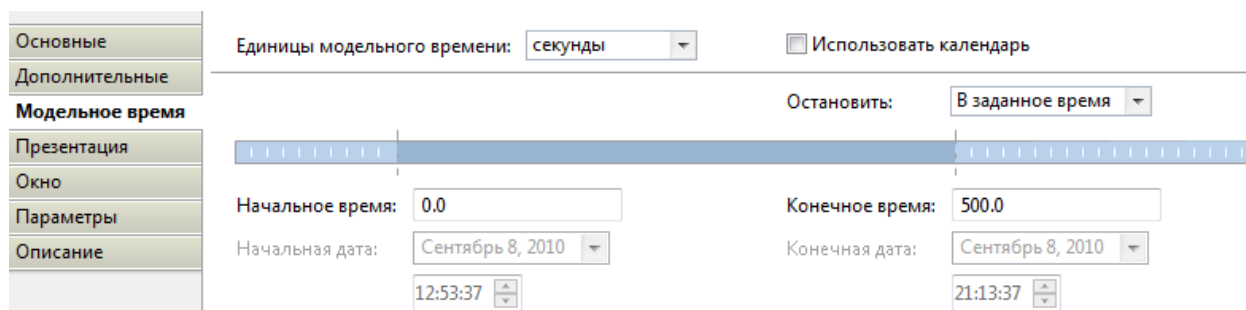


Рис.2.10. Настройка эксперимента

На рисунке 2.11а показана модель в действии.

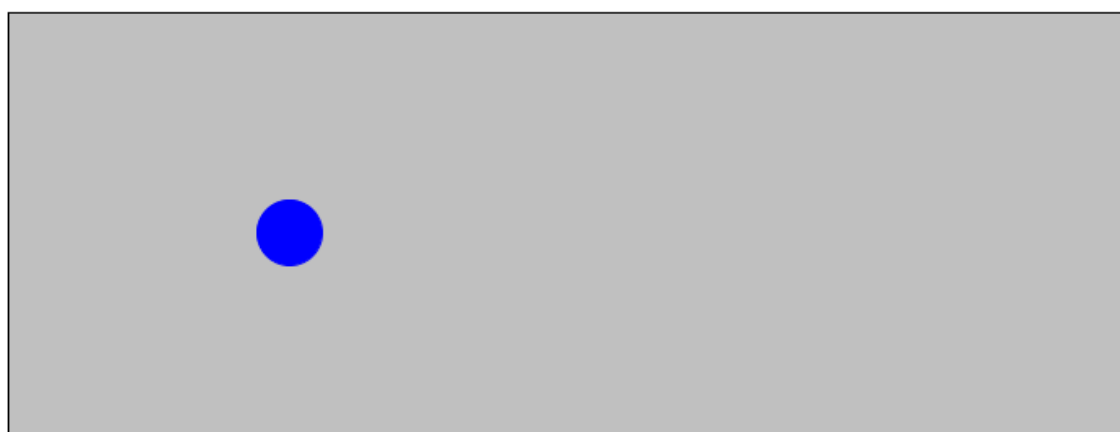
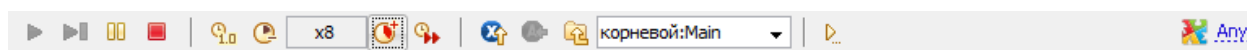


Рис.2.11а. Движение сферы в прямоугольной области.  
Коэффициент ускорения обработки модели восемь.

## 2.2. Контрольное задание «Движение по параболе»

Создайте модель, которая показывает движение красной сферы по параболической траектории. Уравнение траектории имеет вид:

$$y = x^2$$

$$x = [-10...+10]$$

Шаг изменения в диапазоне равен 0,05. В поле корневого объекта модели Main в текстовых полях выводить текущие значения  $y$  и  $x$ .

Вид окна модели показан на рисунке 2.116.

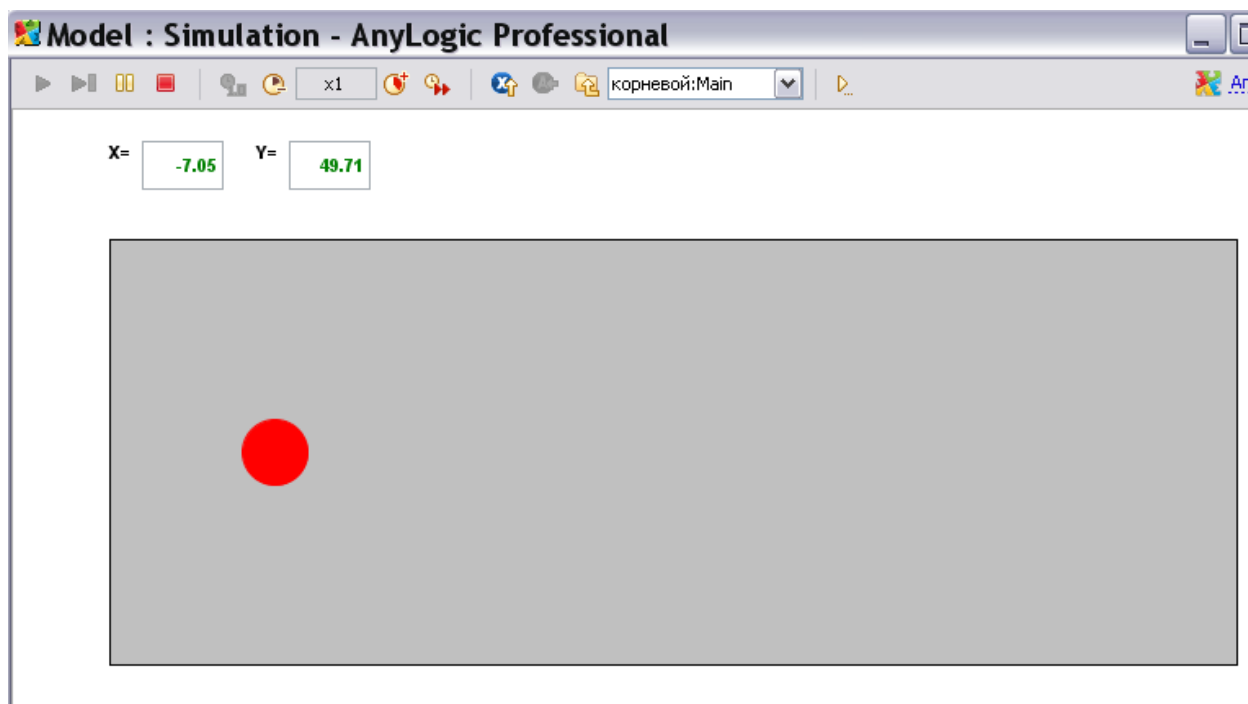


Рис.2.116. Движение по параболе

Если объект движется в интервале  $x$  от  $-5$  до  $+5$  поменять его цвет на ярко зеленый. Вне интервала цвет сферы красный. Выводимые значения зеленого цвета, округлять до второго знака после запятой.

### 2.3. Модель «Жизнь»

Модель имитирует жизненные процессы в виде плоской тороидальной решетки. Ярко синяя клетка - живая клетка. Темно синяя – мертвая клетка.

Создайте проект Life. Модель создается с нуля. В поле объекта Main разместите прямоугольник из палитры «Презентация» и Параметр N целого типа со значением по умолчанию равным 45. Прямоугольник настройте так, как это показано в таблице 2.5. Поле объекта показано на рисунке 2.12.



Рис.2.12. Поле объекта модели

Таблица 2.5. Свойства прямоугольника

№	Свойство	Значение
1	Имя (Вкладка Основные)	Cell
2	Высота(Вкладка Дополнительные)	10
3	Ширина (Вкладка Дополнительные)	10
4	Расположение X (Вкладка Дополнительные)	310
5	Расположение Y (Вкладка Дополнительные)	40
6	X (Вкладка Динамические)	$10 * (\text{index} \% N)$
7	Y (Вкладка Динамические)	$10 * (\text{index} / N)$
8	Количество (Вкладка Динамические)	$N * N$

Где:  $\text{index}$  – параметр, который автоматически изменяется при выполнении модели для перебора все ячеек решетки.

Протестируйте модель. Вид модели после запуска показан на рисунке 2.13. Она представляет собой решетку.

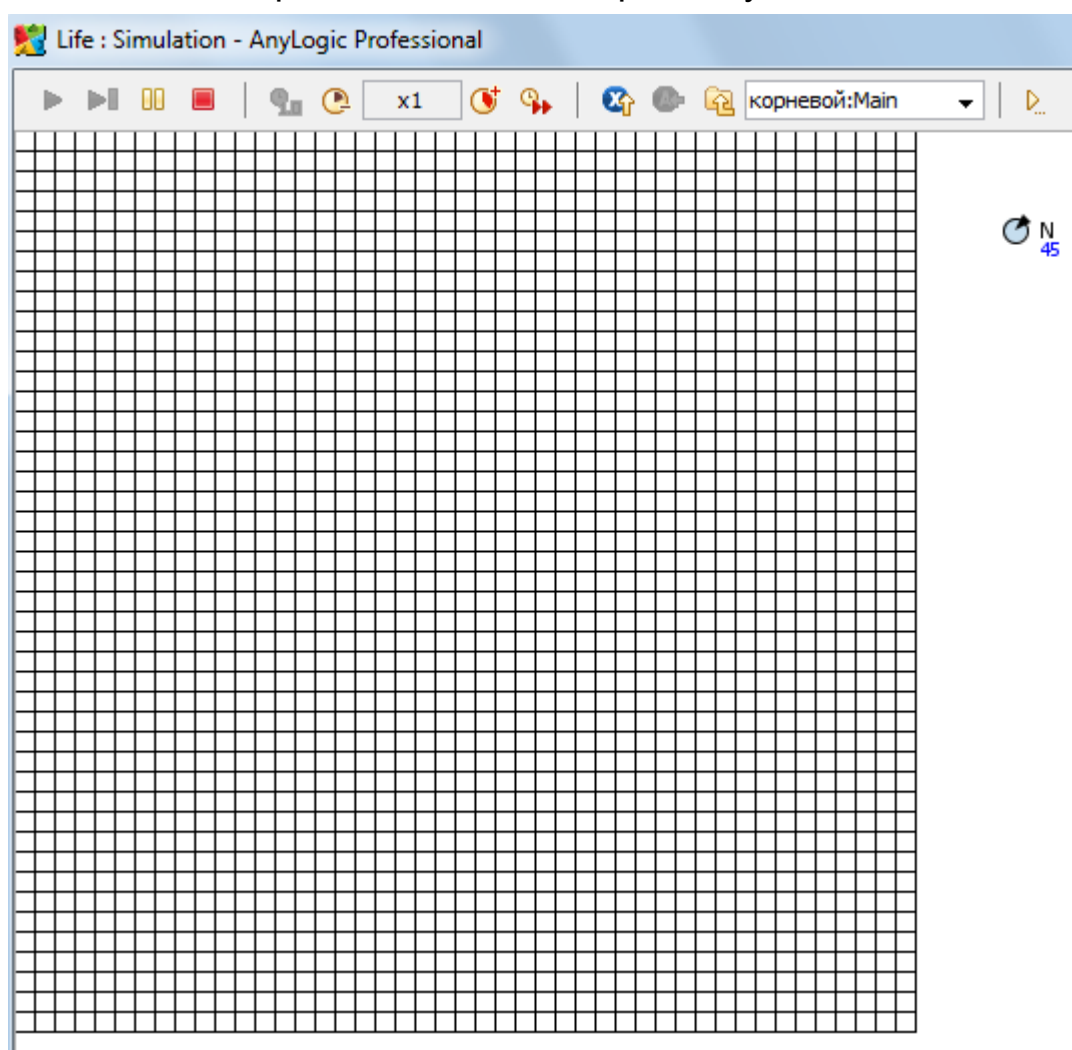


Рис.2.13. Решетка поля жизни

Добавьте в поле модели переменную `alive`. Она представляет собой массив логического типа для моделирования жизненных процессов. При ее задании нужно на вкладке «Основные» выбрать тип «Другой» и задать ее как массив `boolean[][]`. Поле объекта изменится так, как это показано на рисунке 2.14.



Рис.2.14. Измененная модель

На вкладке «Основные» класса `Main` нужно ввести код инициализации решетки (см. рисунок 2.15), исходя из 20% живых клеток.

Действие при запуске:

```
alive=new boolean[N][N];
for (int i=0;i<N;i++)
  for (int j=0;j<N;j++)
    alive[i][j]=randomTrue(0.2); //20% живых ячеек
```

Рис.2.15. Код инициализации

Откройте вкладку «Динамические» прямоугольника ячейки и введите код в свойство «Цвет заливки» для изменения цвета живых не живых ячеек:

```
alive[index%N][index/N] ? new Color(50,210,255):
new Color(30,100,130)
```

После запуска модели решетка должна получить вид, показанный на рисунке 2.16. на котором видны живые и не живые клетки.



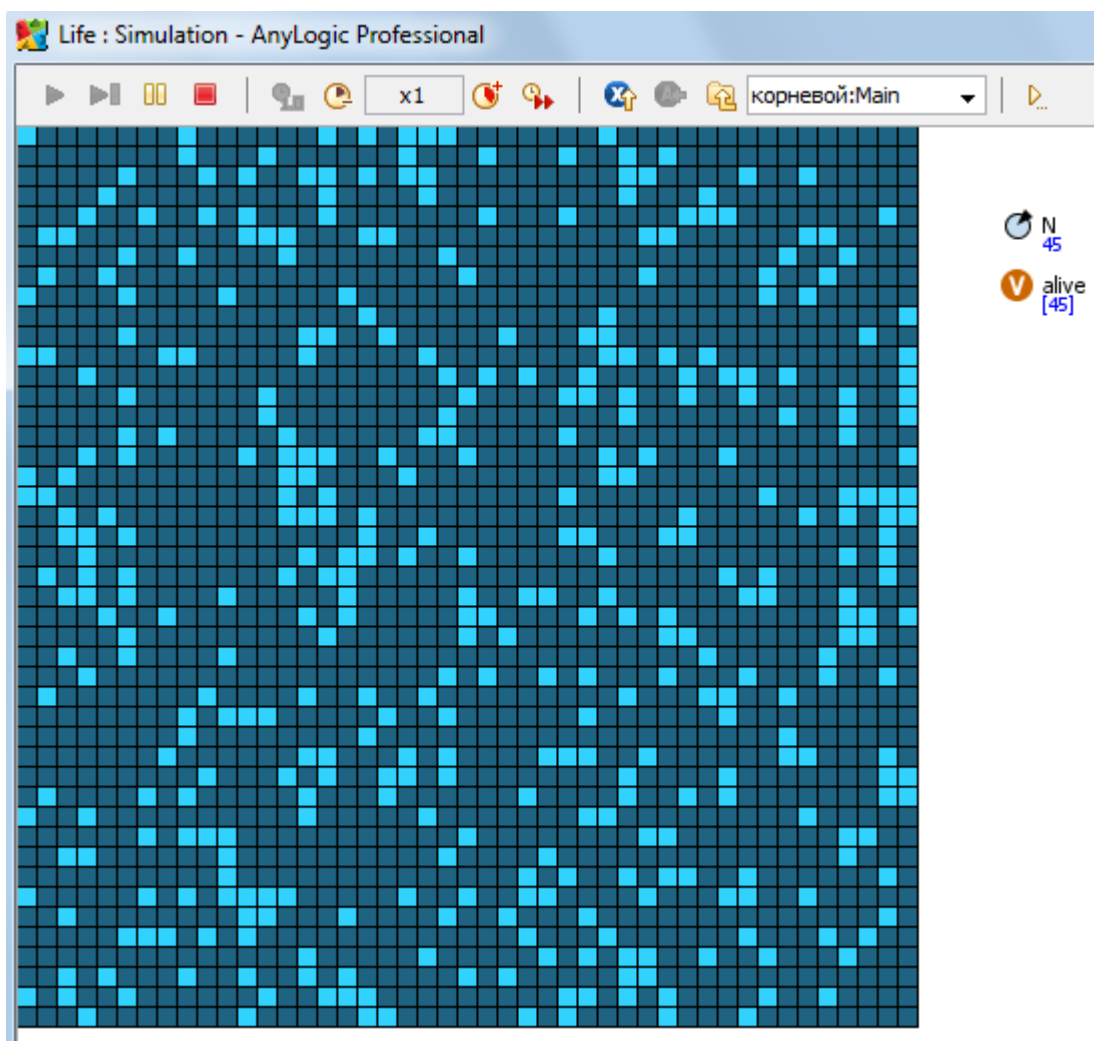


Рис.2.16. Модельная решетка и живые не живые клетки

Введем изменения в модель. Добавим в объект переменную `count` – массив счетчиков ячеек целого типа `int[][]`. Далее нужно настроить действие при запуске объекта `Main`, добавив оператор инициализации массива `count` в виде `count=new int [N][N]`.

Для контроля при переходе живой клетки за границы решетки, создадим функцию `torus`. Функция возвращает результат целого типа `int`. Для ее создания нужно открыть палитру «Основная». На вкладке функции «Основные» нужно задать формальный параметр – аргумент функции `i` целого типа `int`. Код функции имеет вид:

```
return (i<0)?N-1:(i==N)? 0:i;
```

Структура окна объекта `Main` показана на рисунке 2.17.

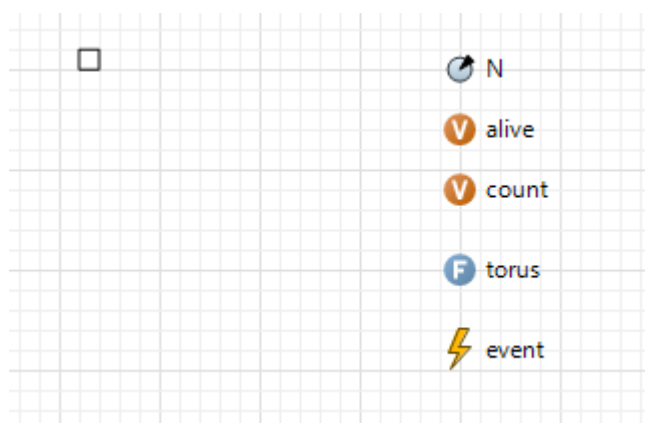


Рис.2.17. Элементы класса

Настройка события event должна соответствовать рисунку 2.18.

<div>Основные</div> <div>Описание</div>	Имя: <input type="text" value="event"/>	<input checked="" type="checkbox"/> Отображать имя	<input type="checkbox"/> Исключить	<input type="checkbox"/> На верхнем ур
	Тип события: <input type="text" value="По таймауту"/>	Режим: <input type="text" value="Циклический"/>		
	Время первого срабатывания (абсолютное) <input checked="" type="radio"/> <input type="text" value="time ()"/> <input type="radio"/> <input type="text" value="Сентябрь 13, 2010"/> <input type="text" value="16:49:11"/>			
	Период: <input type="text" value="second ()"/>			
	Действие: <input type="text" value="changeStates ()"/>			

Рис.2.18. Настройка события event

Кроме новых элементов нужно разместить в код класса два метода – функции. Первая функция используется для подсчета живых соседей у ячейки `countAliveNeighbors`. Вторая функция `changeStates` используется для изменения состояния ячеек. Ячейки становятся живыми или мертвыми. Код данных методов приводится на рисунке 2.19.

Дополнительный код класса:

```
public void countAliveNeighbors () {
    for (int i=0;i<N;i++)
        for (int j=0;j<N;j++) {
            count[i][j]=alive[i][j]?-1:0;
            for (int k=-1;k<2;k++)
                for (int m=-1;m<2;m++)
                    count[i][j]+=alive[torus(i+k)][torus(j+m)]?1:0;
        }
}

public void changeStates () {
    boolean changed=false;
    countAliveNeighbors();
    for (int i=0;i<N;i++)
        for (int j=0;j<N;j++)
            if (!alive[i][j] && count[i][j]==3)
                alive[i][j]=changed=true;
            else if (alive[i][j]&&(count[i][j]==2||count[i][j]==3));
            else{
                if(alive[i][j]) changed=true;
                alive[i][j]=false;
            }
    if (!changed) new Engine().finish();
}
```

Рис.2.19. Методы класса Main

После запуска модель примет вид, показанный на рисунке 2.20.

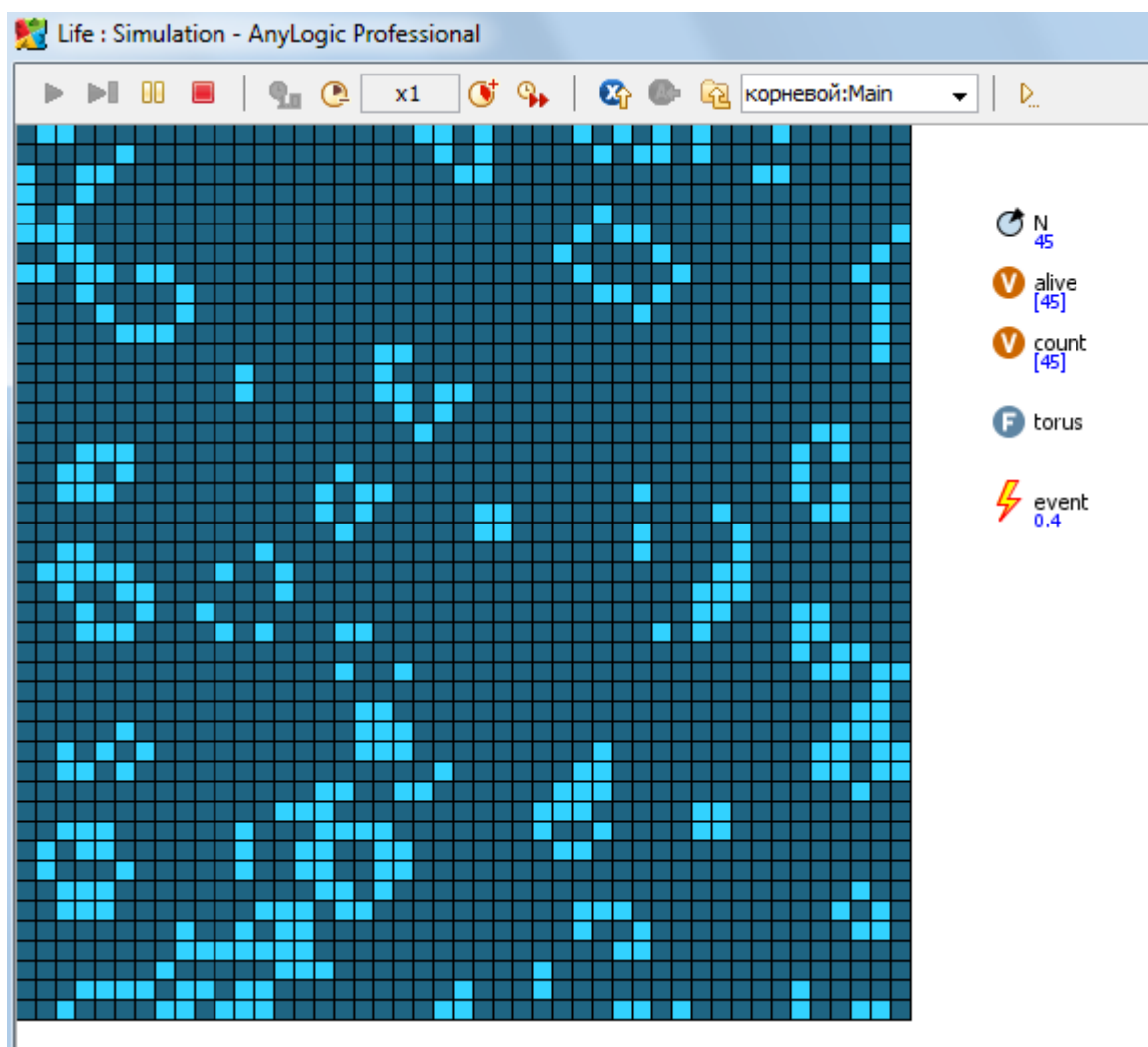


Рис.2.20 Модель в действии

Дополним код модели функцией выбора начальной конфигурации живых и не живых ячеек. Выберем начальную конфигурацию такую, как это показано на рисунке 2.22. При нажатии кнопки 1 решетка «Сбрасывается», все ячейки «умирают» и процесс начинается с начальной конфигурации. При задании начальной конфигурации нужно помнить, что координаты ячеек «перевернуты».

Кнопка размещается из палитры «Элементы управления», а код записывается в свойстве «Действие» на вкладке «Основные». Текст кода приведен на рисунке 2.21.

Действие:

```
for (int i=0; i<N; i++)
    for (int j=0; j<N; j++)
        alive[i][j]=false;
alive[22][23]=true; alive[23][23]=true;
alive[22][24]=true; alive[21][24]=true;
alive[22][25]=true;
```

Рис.2.21. Код кнопки конфигурации

Модель в действии после нажатия кнопки показана на рисунке 2.22.

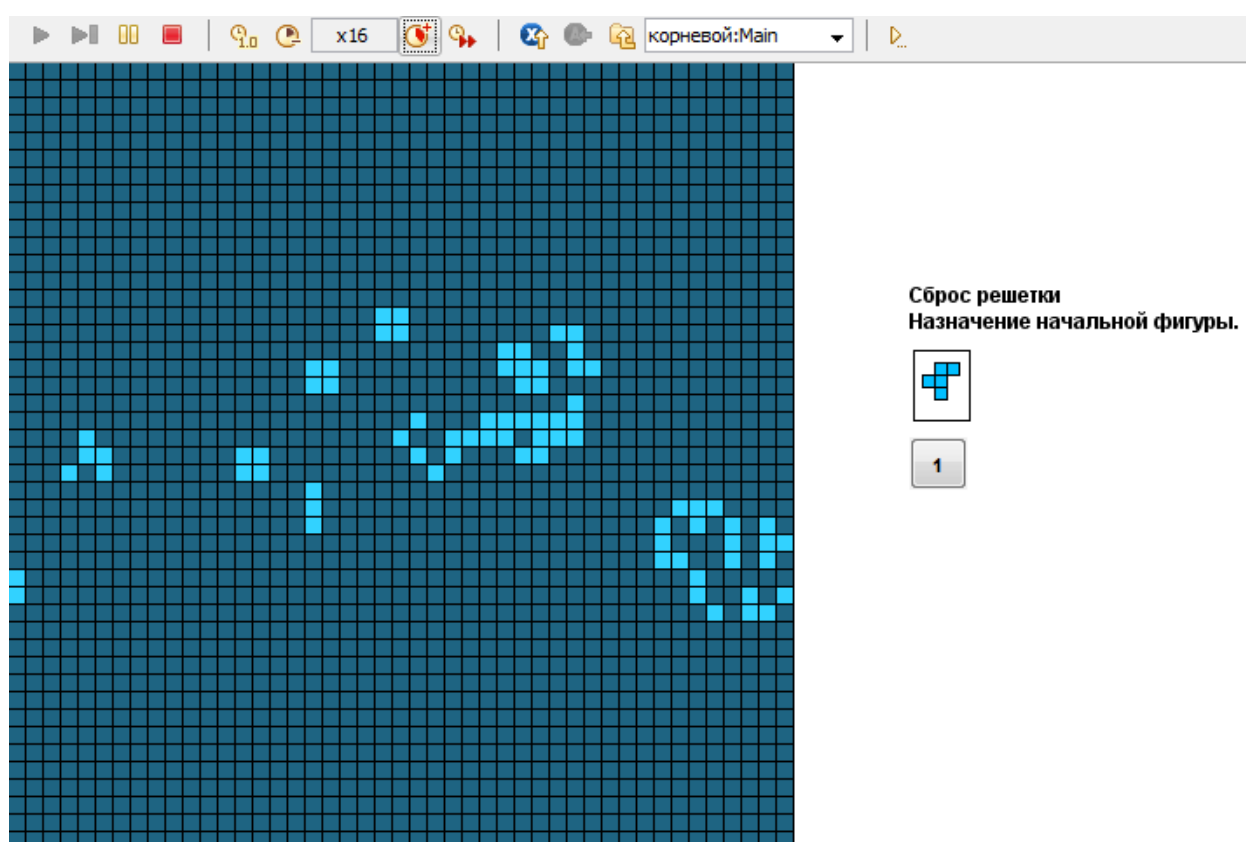


Рис.2.22. Запуск модели при задании начальной фигуры

## 2.4. Контрольное задание «Фигуры»

Дополните модель двумя кнопками с пиктограммами, для начала моделирования с фигур, показанных на рисунке 2.23.

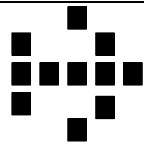
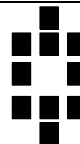
Вариант а	Вариант б
	

Рис.2.23. Фигуры начального состояния решетки

### Контрольные вопросы

1. Что такое физические и мировые координаты модели?
2. Дайте классификацию переменных Java.
3. Как в языке Java выполняется объявление массивов?
4. Как в языке Java выполняется обслуживание массивов?
5. Как в языке Java выполняется обработка строк?
6. Как выполнить преобразование строки с кодами цифр в число?
7. Как выполнить преобразование числа в строку?
8. Для чего используется класс Java Math?
9. Какую структуру имеет класс Java?
10. Как получить экземпляр класса Java?
11. Как выполняется вызов методов и доступ к свойствам класса Java?
12. Какие виды доступа к операциям класса бывают в Java?
13. Что такое исключительная ситуация Java?
14. Как выполняется перехват исключительных ситуаций Java?

## 3. Реагирующие системы

### 3.1. Модель светофора для управления движением

Требуется разработать конечный автомат, который моделирует работу светофора, управляющего движением автотранспорта.

Чтобы построить конечный автомат, нужно создать стейтchart AnyLogic с помощью инструментов палитры «Диаграмма состояний» (см. рисунок 3.1).

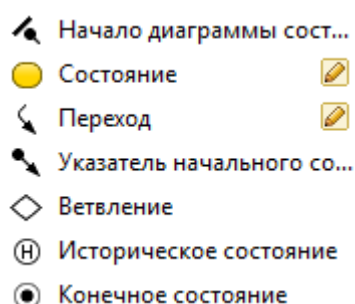


Рис. 3.1. Инструменты для создания диаграммы состояний

Для создания диаграммы нужно использовать три основных инструмента:

- «Начало диаграммы» - отмечает начальную точку обработки стейтчарта.
- «Состояние» - задает состояние диаграммы.
- «Переход» - используется для соединения состояний.
- «Указатель начального состояния» - служит для отметки состояния, с которого начинается обработка вложенной последовательности состояний.
- «Конечное состояние» - отмечает точку завершения обработки состояний.

Наличие начала диаграммы обязательно.

Создайте новую модель с нуля. Разместите в поле класса Main модели три логические переменные. Эти переменные фиксируют состояния светофора:

- `red` – красный сигнал;
- `yellow` – желтый сигнал;
- `green` – зеленый сигнал.

Постройте стейтчарт, так как это показано на рисунке 3.2.

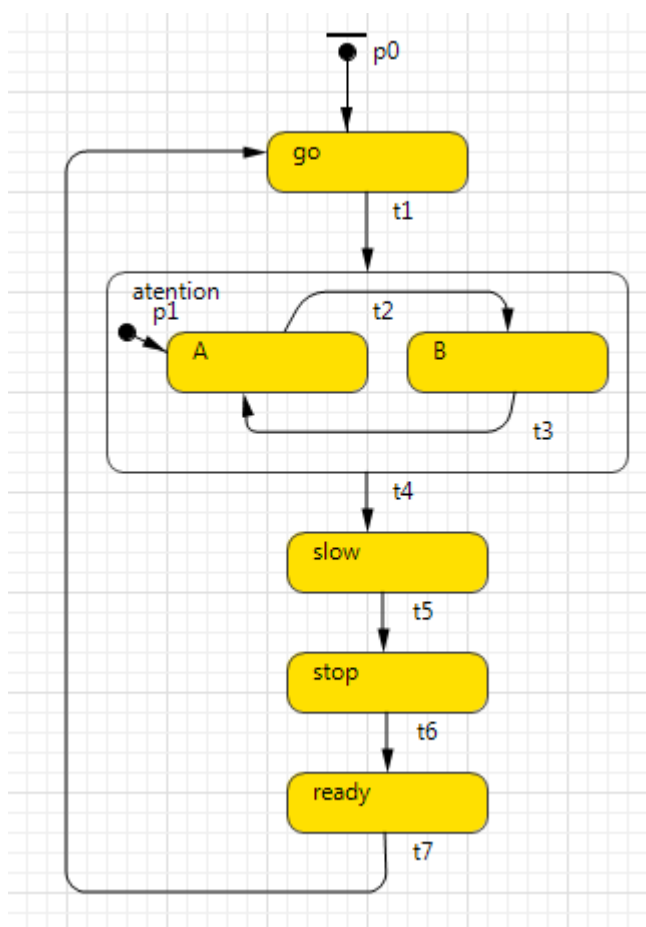


Рис. 3.2. Стейтchart светофора

Для соединения состояний используйте инструмент «Переход». При правильном соединении состояний концевые точки перехода помечаются зеленым цветом. На рисунке 3.3 показан пример правильного соединения двух состояний.

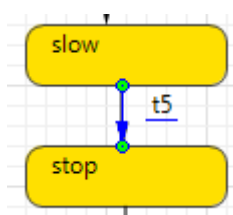



Рис. 3.3. Правильное соединение состояний

Чтобы показать переход, который соединяет состояния по дуге нужно использовать инструмент рисования , который размещен возле инструмента перехода. Его нужно выделить двойным щелчком мыши и делая щелчки мышью в требуемых местах изгиба соединить два состояния.



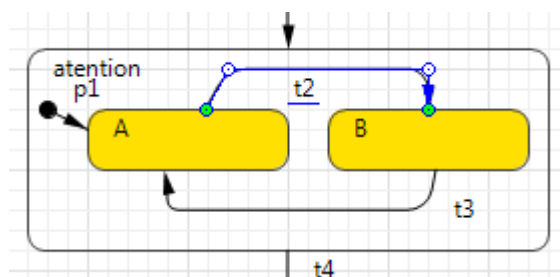


Рис.3.4. Пример соединения состояний

На рисунке 3.4 показаны два состояния соединенных переходом в виде дуги. Места изгиба отмечаются белым круговым маркером.

После создания перехода его вид и места соединения с состояниями можно изменить, передвигая нужные маркеры с помощью мыши, при нажатой левой кнопке.

Настройка стейтчарта должна соответствовать таблицам 3.1 и 3.2.

Таблица 3.1. Настройка состояний

№	Имя	Действие при входе	Действие при выходе
1	go	green=true	green=false
2	attention		
3	A		
4	B	green=true	green=false
5	slow	yellow=true	yellow=false
6	stop	red=true	red=false
7	ready	red=true; yellow=true;	red=false; yellow=false;

Настройка переходов должна соответствовать таблице 3.2.

Перед настройкой переходов присвойте началу стейтчарта имя p0, а указателю начального состояния композитного состояния имя p1.

Таблица 3.2. Настройка переходов

№	Имя	Тип	Период
1	t1	По таймауту	25
2	t2	По таймауту	1
3	t3	По таймауту	1

4	t4	По таймауту	5
5	t5	По таймауту	5
6	t6	По таймауту	25
7	t7	По таймауту	5

Примечание: Чтобы имя перехода выводилось на диаграмме нужно установить его свойство «Отображать имя» в активное состояние. Добиться нужного положения имени перехода на диаграмме можно после выделения линии перехода мышью и захватив имя перехода отбуксировать его в нужное положение.

Следующий шаг заключается в размещении в модель изображения светофора (см. рисунок 3.5).

Используя панель инструментов «Презентация» разместите три овала и задайте поворот на +45 градусов. Вкладка «Дополнительные» > Поворот.

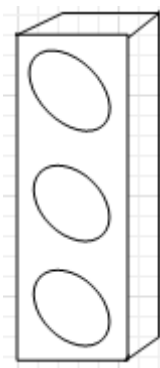


Рис. 3.5. Элемент модели светофор

Верхний овал будет отображать сигнал – красный, средний – желтый сигнал, нижний овал – зеленый сигнал.

Для отображения сигналов нужно на вкладке «Динамические» овалов задать код Java для свойства «Цвет заливки». Код приводится ниже.

Красный сигнал:


```
red ? Color.red : Color.gray
```

Желтый сигнал:

```
yellow ? Color.yellow: Color.gray
```

Зеленый сигнал:

```
green ? Color.green: Color.gray
```

Поместите созданные овалы в прямоугольник, боковые грани нарисуйте с помощью инструмента «Ломанная». Выполните двойной щелчок, на элементе ломанной  и перейдите в режим рисования. Каждая точка ломанной отмечается щелчком мыши, для прекращения рисования нужно выполнить двойной щелчок на замыкающей точке. Вид светофора должен соответствовать рисунку 3.5

Настройте модельное время эксперимента модели – объект Simulation.

- Единицы модельного времени – секунды;
- Остановить – нет.

Работающая модель показана на рисунке 3.6.

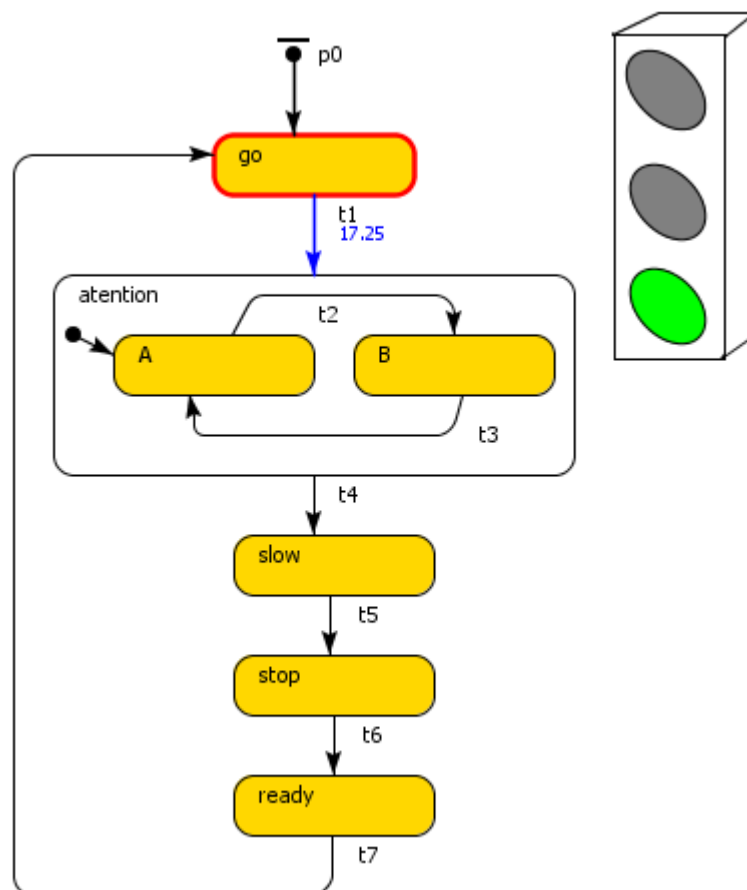


Рис. 3.6 Вид работающей модели управления светофором

3.2. Модель пешеходного перехода

Дополните модель, созданную при выполнении задания №1, моделью светофора пешеходного перехода. При появлении красного сигнала на светофоре движения автотранспорта, должен включаться зеленый сигнал светофора пешеходного перекрестка. При появлении сигнала красного цвета на светофоре пешеходного перехода дается зеленый сигнал на светофоре движения. Процесс повторяется циклически.

Введите в модель две переменные логического типа redP, greenP.

Постройте стейтchart, который соответствует рисунку 3.7

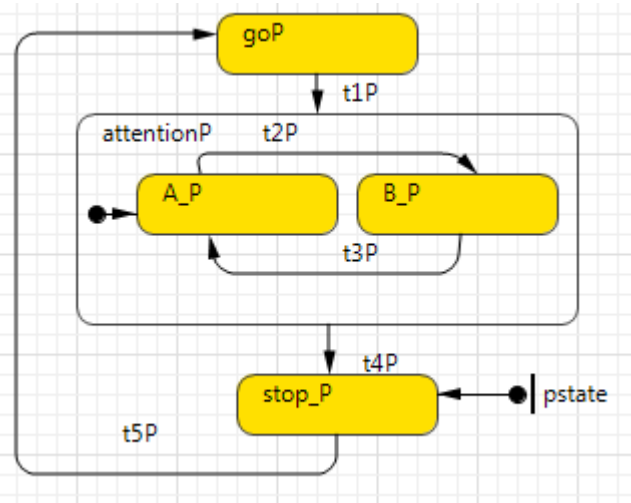


Рис. 3.7. Стейтchart светофора пешеходного перехода

Настройка переходов и состояний должна соответствовать таблицам 3.3, 3.4.

Таблица 3.3. Настройка переходов пешеходного стейтchartа

№	Имя	Тип	Период
1	t1p	По таймауту	15
2	t2p	По таймауту	1
3	t3p	По таймауту	1
4	t4p	По таймауту	15
5	t5p	По таймауту	1

Таблица 3.4. Настройка состояний стейтchartа пешеходного перехода

№	Имя	Действие при входе	Действие при выходе

1	goP	greenP=true	greenP=false
3	B_P	greenP=true	greenP=false
4	Stop_P	redP=true	redP=false

Для организации взаимодействия между стейтчартами в стейтchart светофора управления движением внести следующие изменения:

Отредактируйте переход t5, добавив в свойство «Действие» код Java `pstate.fireEvent("ПЕШЕХОДЫ");`

Данный оператор передает стейтчарту светофора пешеходного перехода контрольное сообщение для включения зеленого сигнала.

В стейтчарте светофора пешеходного перехода нужно изменить настройку перехода t5P, так как это показано на рисунке 3.8.

Рис. 3.8. Настройка перехода

Когда цикл работы светофора пешеходного перехода закончил работу, нужно подать сигнал светофору управляющему движением. В свойстве «Действие» для перехода t4P введите оператор Java `p0.fireEvent("ТРАФИК")`.

В стейтчарте светофора управляющего движением для перехода t6 измените настройки свойств в соответствии с рисунком 3.9.

Рис.3.9. Настройка перехода стеитчарта светофора управления движением

Рядом со стеитчартом управления работой светофора пешеходного перехода разместите его изображение.

Разместите две окружности.

Верхняя окружность служит для показа красного сигнала, а нижняя зеленого. Для окружностей введите код Java для изменения цвета заливки:

```
redP ? Color.red : Color.gray
greenP ? Color.green : Color.gray
```

Окружности поместите в прямоугольник, так как это показано на рисунке 3.10.

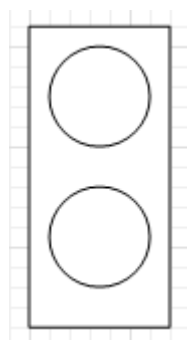


Рис. 3.10. Светофор пешеходного перехода

Протестируйте созданную модель. Вид работающей модели должен соответствовать рисунку 3.11.

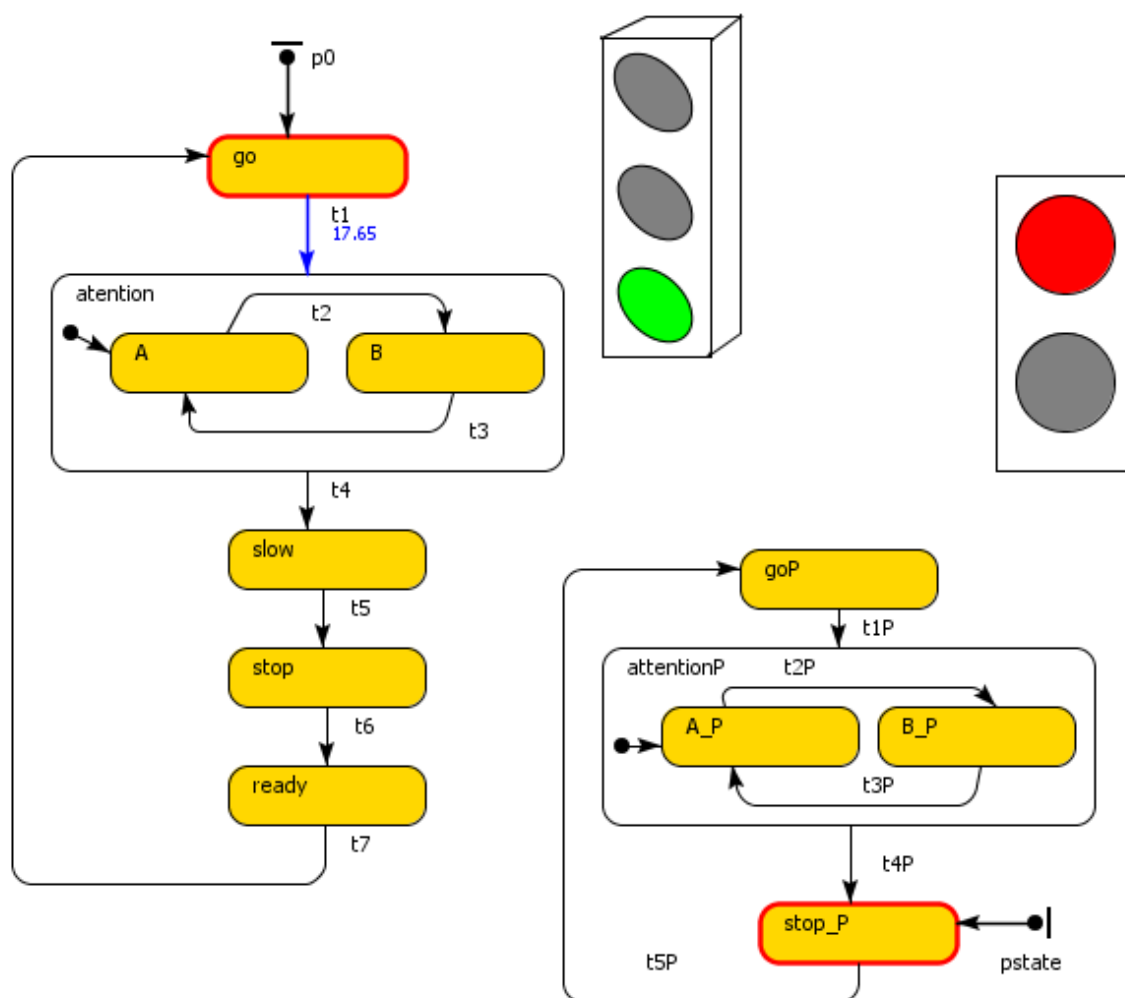


Рис.3.11. Модель пешеходного перехода

### 3.3. Контрольное задание “Переход, управляемый пешеходом”

Создайте модель пешеходного перехода со следующими особенностями:

Светофор управления движением дает зеленый сигнал транспорту, светофор пешеходного перехода дает запрещающий сигнал пешеходам.

Что бы пешеход мог перейти дорогу он должен нажать кнопку «ЖДУ» на светофоре перехода. Параметры переключения сигналов светофоров аналогичны заданию №2.

Методические указания:

Модель перехода должна иметь вид, показанный на рисунке 3.12.

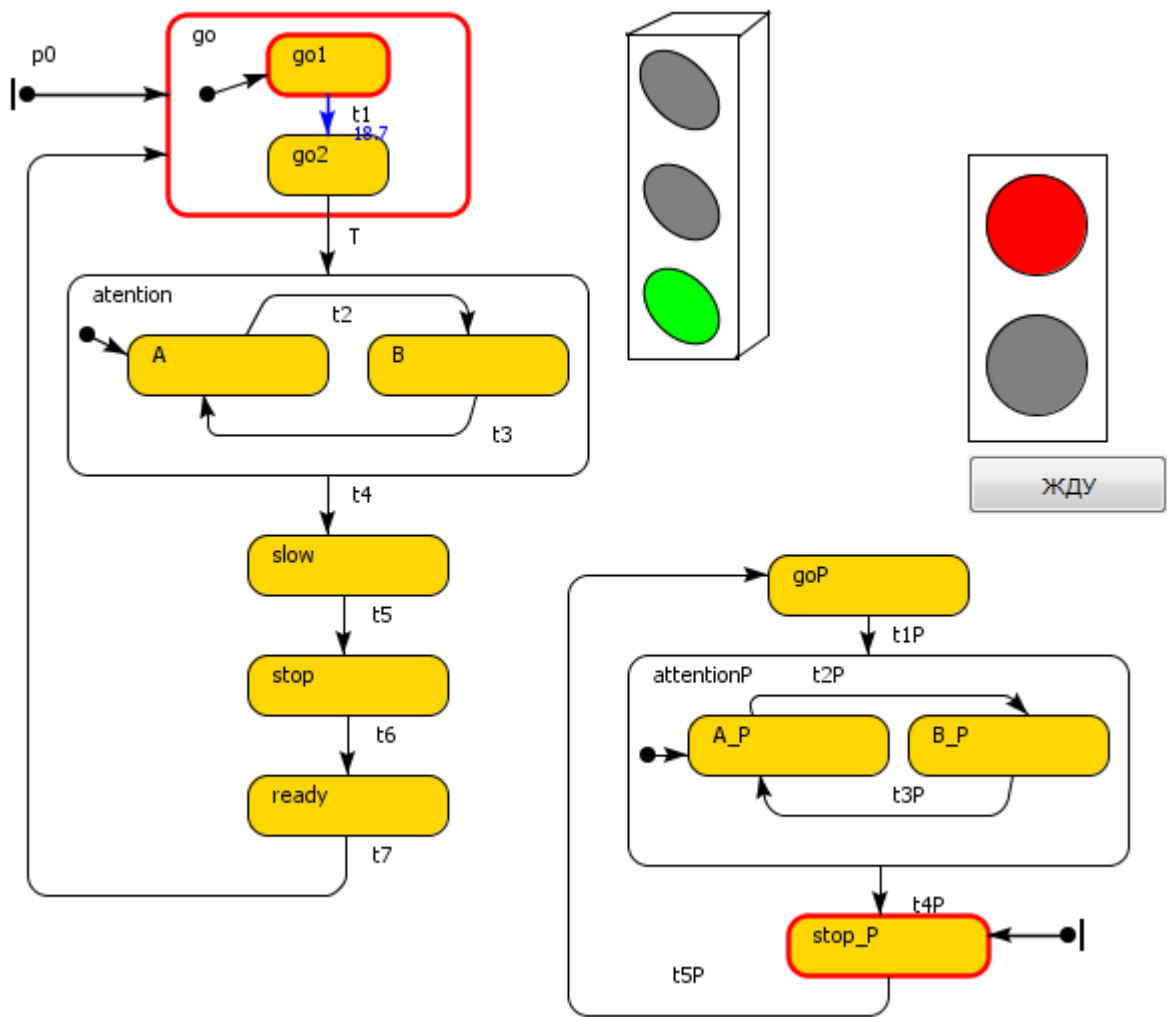


Рис.3.12. Переход, управляемый пешеходом

В модель нужно ввести логическую переменную `waiting`. Когда пешеход нажимает кнопку, то ей присваивается значение `true`.

В стейтchart управления светофором движения нужно ввести композитное состояние `go`, переход `T` должен срабатывать, когда пешеход нажал кнопку «ЖДУ».

Когда цикл работы светофора пешеходного перехода заканчивается, то переменной `waiting` нужно присвоить значение `false` при выполнении перехода `t1P`.

### 3.4. Контрольное задание “Модель кодового замка”

Создайте модель с конечным автоматом для имитации работы кодового замка. Замок открывается при наборе кодовой шести разрядной последовательности 137819.



Начальное состояние замка мигающий символ # зеленого цвета. Пользователь замка нажимает кнопки для набора кода, при этом вводимый код не отображается. Кнопка «С» позволяет сбросить кодовую последовательность при наборе.

Когда набран шести разрядный код, замок переходит в состояние его проверки. Код выводится на дисплей и происходит задержка на 5 секунд. Если код набран верно, то выводится сообщение «OK !» иначе «ERROR!». Для сброса состояния замка пользователь нажимает клавишу «С» и замок переходит в начальное состояние. На рисунках 3.13-3.15 показаны состояния замка.



Рис. 3.13. Начальное состояние замка

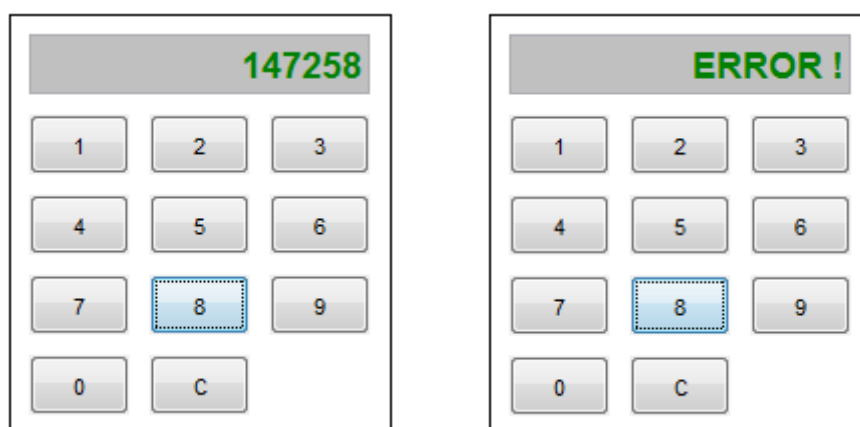


Рис. 3.14. Ввод неверного кода

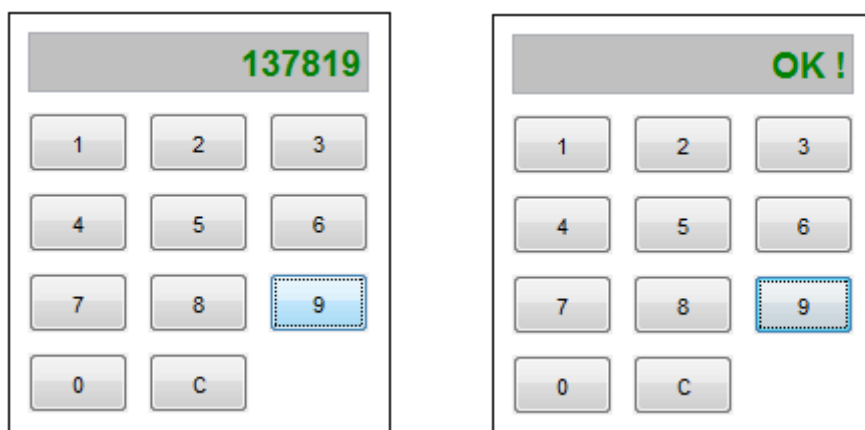


Рис. 3.15. Код верный

Методические указания:

Для управления работой замка нужно создать стейтчарт, который показан на рисунке 3.16.

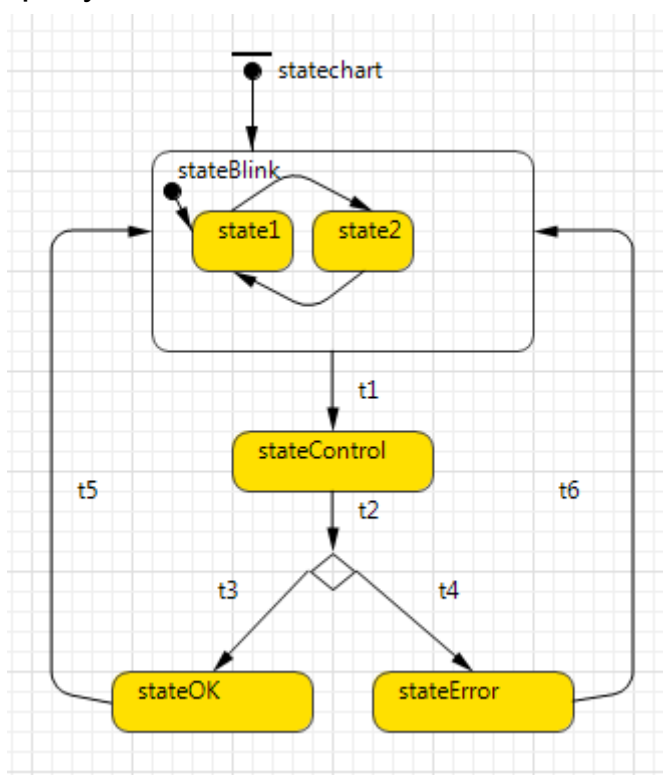


Рис. 3.16. Стейтчарт кодового замка.

Состояния кодового замка:

- `stateBlink` – задает мерцание сигнала дисплея;
- `stateControl` – проверка кода введенного пользователем;
- `stateOK` – вывод сообщение о том, что код верный;
- `stateError` – вывод сообщения о неверном коде.

Для контроля за состоянием замка нужно ввести две переменные. Одну логического типа для фиксирования правильности ввода кода и символьную для сохранения введенной кодовой последовательности.

Чтобы обслуживать текстовый дисплей замка используйте методы текстового поля:

`setText(String s)` – запись в поле строки;

`String getText()` – чтение строки из поля.

### 3.5. Модель трех разрядного счетчика

Требуется разработать модель счетчика, для отсчета целых трехразрядных чисел. Работа счетчика синхронизируется генератором тактовых импульсов.

Создайте модель с нуля. Разместите в нее активный класс `Gen` (генератор тактовых импульсов). Структура класса показана на рисунке 3.17.

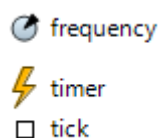


Рис. 3.17. Генератор тактовых импульсов

Разместите в поле класса параметр `frequency`. Этот параметр определяет частоту срабатывания генератора. Тип параметра `double`. Значение по умолчанию равно 2.

Разместите в поле генератора порт с помощью инструмента палитры «Основная» «Порт».

Порт позволяет принимать и отсылать сообщения. Настройте свойства порта. Присвойте порту имя `tick`.

Разместите элемент «Событие» и настройте его свойства так, как это показано на рисунке 3.18.

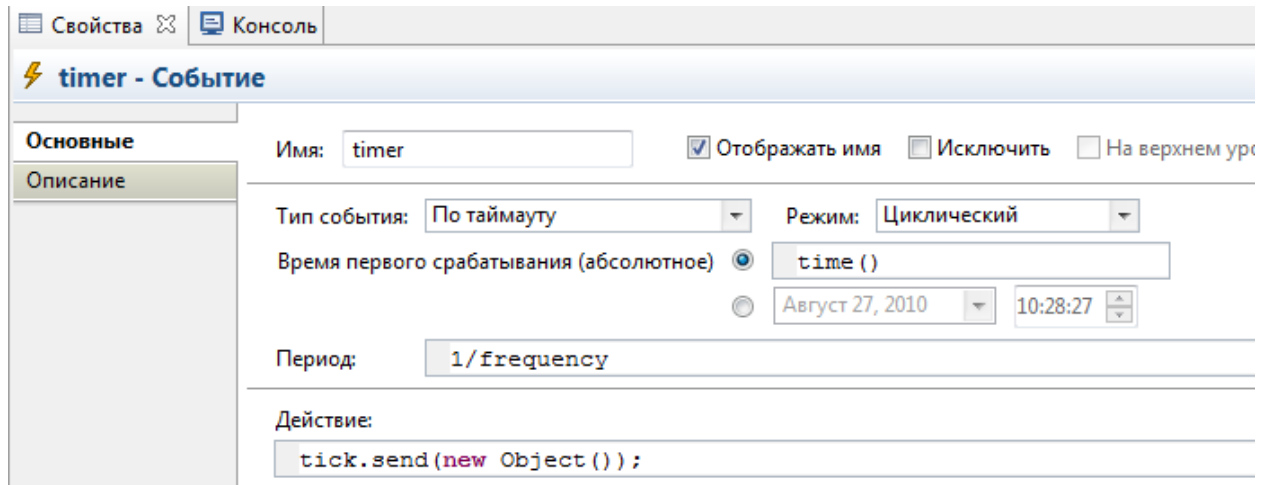


Рис. 3.18. Настройка события

В поле действие события следует ввести оператор Java для обращения к методу send порта генератора импульсов. Методу передается экземпляр абстрактного объекта.

Порты образуют интерфейс активного класса. Для создания интерфейса нужно создать значок активного класса. Поместите в поле активного класса скругленный прямоугольник из палитры «Презентация» и активизируйте его свойство «Значок». Задайте ширину прямоугольника 80 пикселей и высоту 60 пикселей. На прямоугольник поместите элемент «Текст» и введите строку текста ГТИ (генератор тактовых импульсов). Для текста активизируйте свойство «Значок». Расположите порт на контуре прямоугольника, так как это показано на рисунке 3.19.

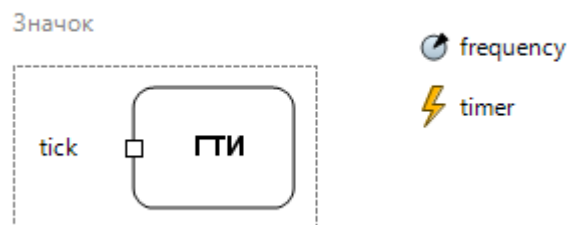


Рис. 3.19. Значок активного класса

Свойство порта «На верхнем уровне» должно быть активным. Свойство «Отображать имя» сделайте не активным.

Создайте активный класс Counter (разряд счетчика). Структура активного класса показана на рисунке 3.20.

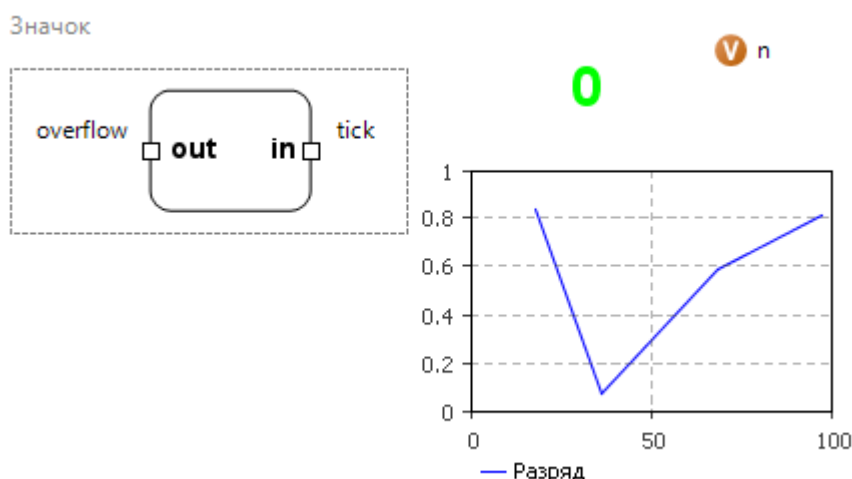


Рис. 3.20. Активный класс разряда счетчика

Поместите в класс переменную `n` целого типа, для подсчета числа единиц в разряде.

Затем следует расположить два порта.

Порт `overflow` – наличие сообщения в этом порте говорит о переполнении числа единиц в разряде счетчика.

Порт `tick`. Принимает сообщение. В свойстве «Действие при получении» следует ввести код Java:

```
if (n==9) {
n=0;
overflow.send(msg);
}
else
n++;
```

В коде запрограммирован подсчет числа единиц. Если число единиц равно 9, нужно вызвать метод `send` порта `overflow` и поместить в него абстрактное сообщение, что является сигналом переполнения.


Создайте интерфейс активного класса. Для создания интерфейса используйте скругленный прямоугольник с шириной 80 пикселей и высотой 60 пикселей. Расположите на значке порты так, как это показано на рисунке 3.20. Поместите поясняющий текст для портов.

Затем следует расположить в поле класса временной график для отображения числа единиц счетчика n.

Чтобы можно было наблюдать текущее значение единиц в разряде счетчика, следует расположить элемент «Текст» из палитры «Презентация»

Цвет текста выберите зеленый, размер шрифта 28 пунктов. Перейдите на вкладку «Динамические» и задайте свойство Текст="" + n.

У портов сделайте свойство «Отображать имя» не активным, а свойство на «На верхнем уровне» активным.

Откройте корневой класс модели Main и разместите в него экземпляр активного класса Gen и три экземпляра класса Counter. Для соединения экземпляров по портам используйте элемент «Соединитель»  из палитры «Основная». При правильном соединении точки соединения помечаются зеленым цветом.

Вид поля класса Main показан на рисунке 3.21.

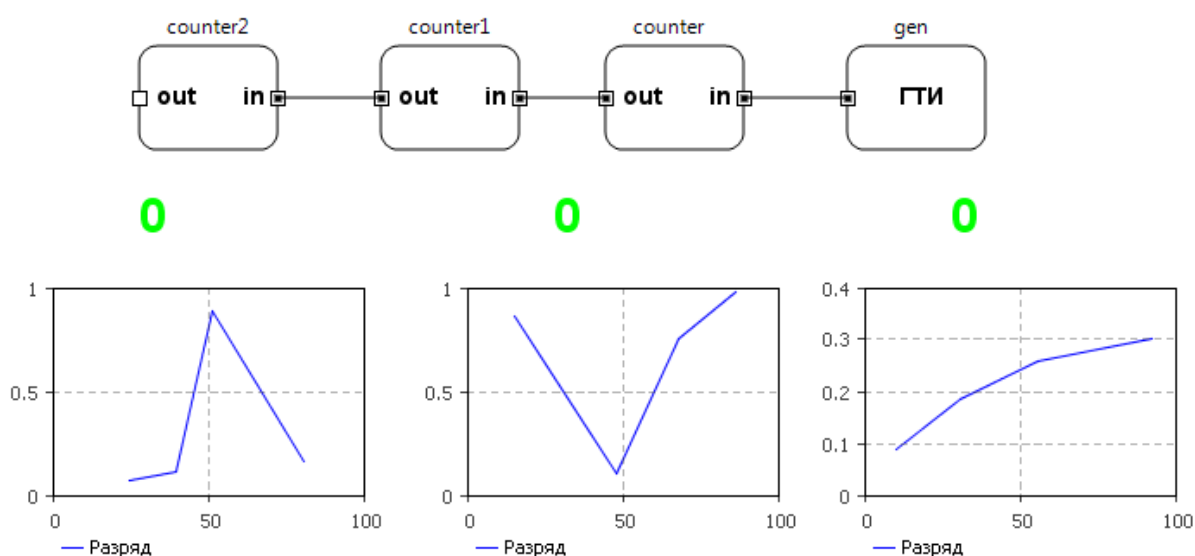


Рис. 3.21. Корневой класс Main

Настройте модельное время эксперимента Simulation. Единицы модельного времени – секунды. Граничное время моделирование не указано. Запустите модель. Вид модели должен соответствовать рисунку 3.22.

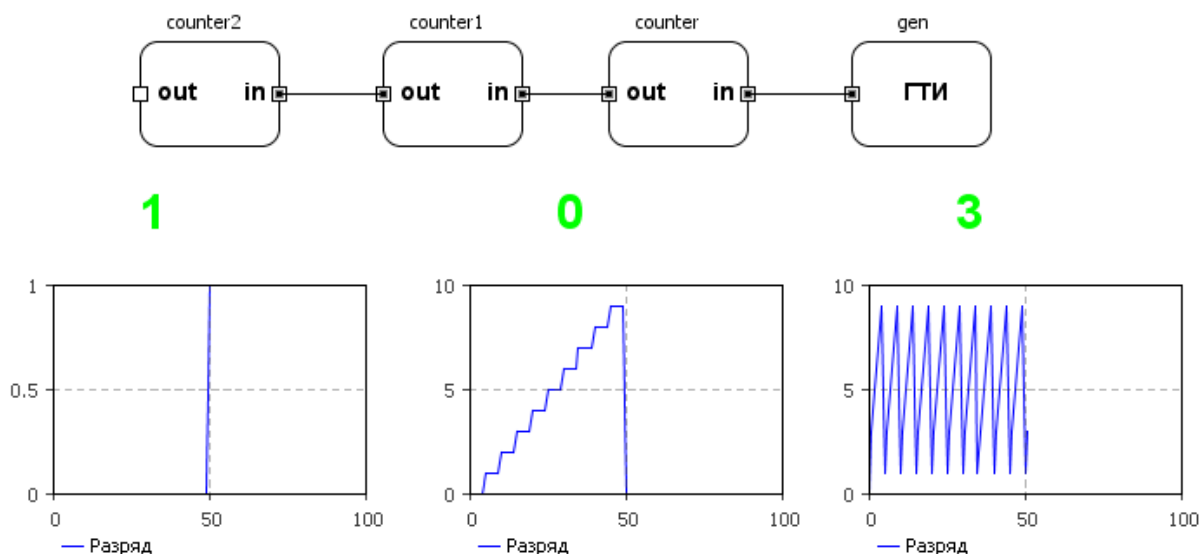


Рис.3.22. Модель трехразрядного счетчика

### 3.6. Контрольное задание «Запуск события по истечении контрольного времени»

Постройте модель с трех разрядным счетчиком. Когда счетчик выдает показание боле 150 единиц, срабатывает генератор случайных целых чисел имеющих равномерное распределение в диапазоне от 10 до 20. Генератор прекращает работу, когда счетчик насчитывает 300 единиц. Для вывода показаний разрядов счетчика используйте текстовые поля. Вывод случайных чисел производится в текстовое поле с округлением до третьего знака после запятой. Вид работающей модели показан на рисунке 3.23. Генератор случайных чисел активен.

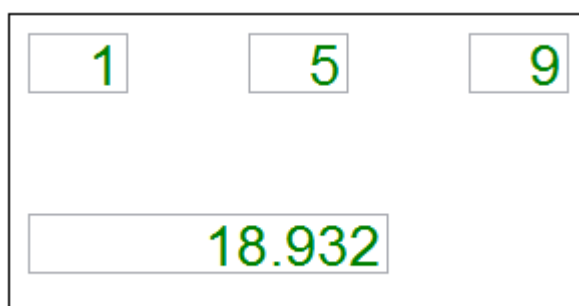


Рис. 3.23. Вид работающей модели

Методические указания:

В класс `Counter` следует поместить текстовое поле для отображения числа единиц, вместо элемента «Текст». Количество единиц записывается с помощью метода `setText(String s)` поля.

В поле класса `Main` нужно поместить два события и переменную-буфер целого типа. Первое событие выполняется циклически и сканирует показания числа единиц разрядов счетчика. Чтобы получить число сгенерированное счетчиком нужно использовать упаковщик типов Java. Для данного случая упаковщик `Integer`:

```
intBufer=new Integer (strNumber).intValue()
```

Где: `strNumber` – строковое представление числа полученного от счетчика, а `intBufer` – буфер целого типа, куда записывается число.

Второе событие должно выполняться по условию и генерировать равномерно распределенное число и помещать его в поле вывода. Такое число можно получить, используя встроенную функцию `AnyLogic uniform(double min,double max)`, где `min` и `max` границы диапазона генерации числа.

Для управления событием, выполняющимся по условию нужно использовать методы:

- `reset()` – сброс события и перевод его в состояние ожидания выполнения условия его запуска;
- `restart()` – запуск события.

### Контрольные вопросы

1. Как представляется конечный автомат в нотации UML?
2. Дайте классификацию событий UML.
3. Как показывается событие в конечном автомате UML?
4. Дайте классификацию переходов UML.
5. Что такое композитное состояние UML?
6. Объясните особенности исторических состояний UML.



7. Как строятся конечные автоматы с параллельными вложенными конечными автоматами?
8. Для чего используется псевдосостояние синхронизации в UML?
9. Как виды переходов определены в UML для конечных автоматов?
10. Дайте классификацию состояний стейтчарта AnyLogic?
11. Как выполняется управление переходами в стейтчарте AnyLogic?
12. Как строятся стейтчарты с альтернативным выбором переходов в AnyLogic?
13. Как в AnyLogic строятся стейтчарты с поверхностным историческим состояниями?
14. Как в AnyLogic строятся стейтчарты с глубокими историческими состояниями?
15. Как выполняется обмен сообщениями между стейтчартами AnyLogic?
16. Как строится взаимодействие между экземплярами активных классов с помощью портов?
17. Как настраивается порт активного класса?

#### 4. Системная динамика

##### 4.1. Модель реализации продукта по Бассу

Модель реализации продукции описывается системой уравнений:

$$\frac{d(PotentialAdopters)}{dt} = -AdoptionRate$$

Начальное значение :  $TotalPopulation$

$$\frac{d(Adopters)}{dt} = +AdoptionRate$$

$$AdoptionFromAd = PotentialAdopters \times AdEffectiveness$$

$$AdoptionFromWOM = Adopters \times ContactRate \times AdoptionFraction \times \frac{PotentialAdopters}{TotalPopulation}$$

$$AdoptionRate = AdoptionFromAd + AdoptionFromWOM$$

В модели приняты следующие условные обозначения для накопителей:

- `PotentialAdopters` (Потенциальные потребители продукции);
- `Adopters` (Потребители, которые уже купили продукт).

Поток, моделирующий процесс потребления обозначен как `AdoptionRate`.

В модели используются переменные:

- `AdoptionFromAd` - число потребителей продукта, которые его приобрели под влиянием рекламы;
- `AdoptionFromWOM` - число потребителей продукта, которые его приобрели под влиянием общения с потребителями, которые уже купили продукт.

Интенсивность процесса, приобретения продукта моделируется потоком `AdoptionRate`.

Константы-параметры модели:

- `TotalPopulation` (Численность населения);
- `ContactRate` (Число контактов);
- `AdEffectiveness` (Эффективность рекламы);
- `AdoptionFraction` (Сила убеждения);

Модель создается с «нуля». Построение модели начинается с создания накопителей `PotentialAdopters` и `Adopters`, соединенных потоком `AdoptionRate`. Для создания модели нужно использовать палитру «Системная динамика».

Что бы создать поток нужно, соединяющий накопители нужно:

Разместить накопители, задать им имена (см. рисунок 4.1).



Рис.4.1. Размещение накопителей

Выделите накопитель `PotentialAdopters`, затем выполните на нем двойной щелчок левой кнопкой мыши и соедините его с

помощью стрелки потока с накопителем *Adopters*, выполнив на нем двойной щелчок мышью (см. рисунок 4.2).



Рис.4.2. Соединение накопителей. Фаза 1.

Затем нужно присвоить потоку имя *AdoptionRate*, так как это показано на рисунке 4.3.

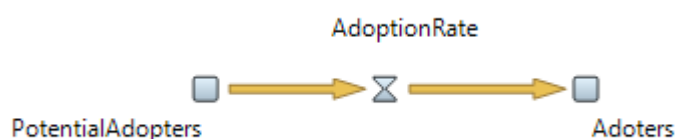


Рис.4.3. Накопители, соединенные потоком. Фаза 2.

Примечание. При задании имени потоку следует вводить имени завершить нажатием комбинации клавиш **CTRL+ENTER**, ответив утвердительно на все последующие запросы. Такая комбинация клавиш автоматически запускает процесс AnyLogic для контроля за именами в уравнениях модели. Имя, введенное после нажатия комбинации клавиш **CTRL+ENTER**, автоматически будет проставлено во всех уравнениях модели, где было использовано старое имя.

Затем следует разместить параметры модели и переменные в соответствии с уравнениями модели.

В таблице 4.1. приводятся значения параметров модели.

Таблица 4.1. Параметры динамической модели

№	Параметр	Значение
1	TotalPopulation	100000
2	ContactRate	100
3	AdEffectiveness	0,011
4	AdoptionFraction	0,015

Вид модели показан на рисунке 4.4.

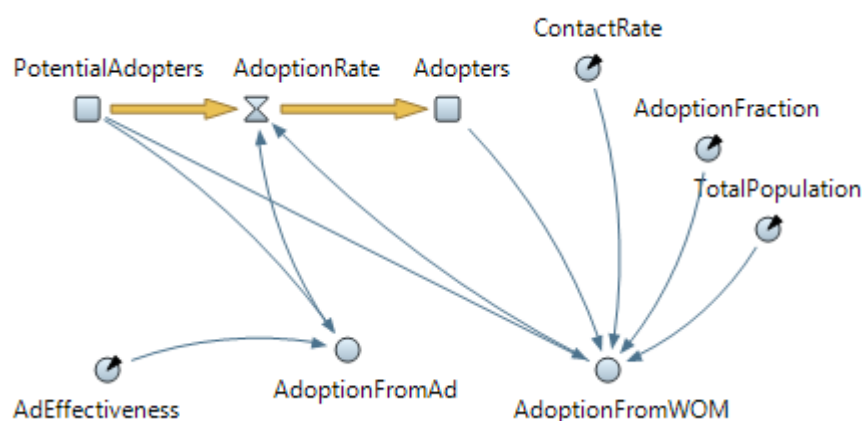


Рис.4.4. Модель реализации продукта

Чтобы проследить за процессами, протекающими в модели, следует разместить временные графики.

Первый график показывает динамику изменения числа потенциальных потребителей продукта и числа лиц, которые приобрели продукт.

Для получения наглядных графиков при моделировании следует период обновления выбрать равным 0,25, а количество отображаемых точек задать равным 500.

Настройте эксперимент модели Simulation.

Единицы модельного времени задайте минуты, модель должна останавливаться в заданное время, при достижении модельного времени 100 единиц.

Протестируйте модель. Вид графиков работающей модели должен соответствовать рисунку 4.5.

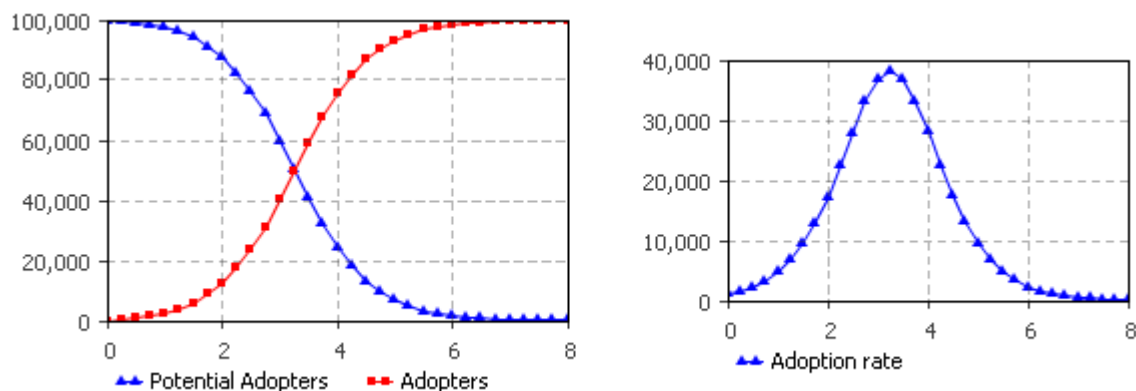


Рис.4.5. Динамика процессов модели Басса

#### 4.2. Контрольное задание «Учет повторных покупок»

Создайте модель, которая моделирует процесс реализации продукции по Бассу с учетом, того что продукт, купленный потребителем со временем приходит в негодность и потребитель покупает новый продукт на его замену.

Методические указания:

Модель реализации продукта примет вид:

$$\frac{d(PotentialAdopters)}{dt} = -AdoptionRate + DiscardRate$$

Начальное значение :  $TotalPopulation$

$$\frac{d(Adopters)}{dt} = +AdoptionRate - DiscardRate$$

$$AdoptionFromAd = PotentialAdopters \times AdEffectiveness$$

$$AdoptionFromWOM = Adopters \times ContactRate \times AdoptionFraction \times \frac{PotentialAdopters}{TotalPopulation}$$

$$AdoptionRate = AdoptionFromAd + AdoptionFromWOM$$

Здесь  $DiscardRate$  – новый поток, отражающий повторные покупки. Для его определения в модели Басса вводится формула:

$$DiscardRate = f(AdoptionRate, ProductLifeTime).$$

Где:  $ProductLifeTime=2$  (года) – параметр, который задает время годности продукта в процессе его использования пользователем.

Функция  $f$  представляет собой задержку и вычисляется в зависимости от двух аргументов: основного потока и константы, которая характеризует задержку – время жизни продукта.

Чтобы получить такую функцию в AnyLogic нужно использовать встроенную функцию `delay`.

Модель реализации продукта примет вид, показанный на рисунке 4.6.

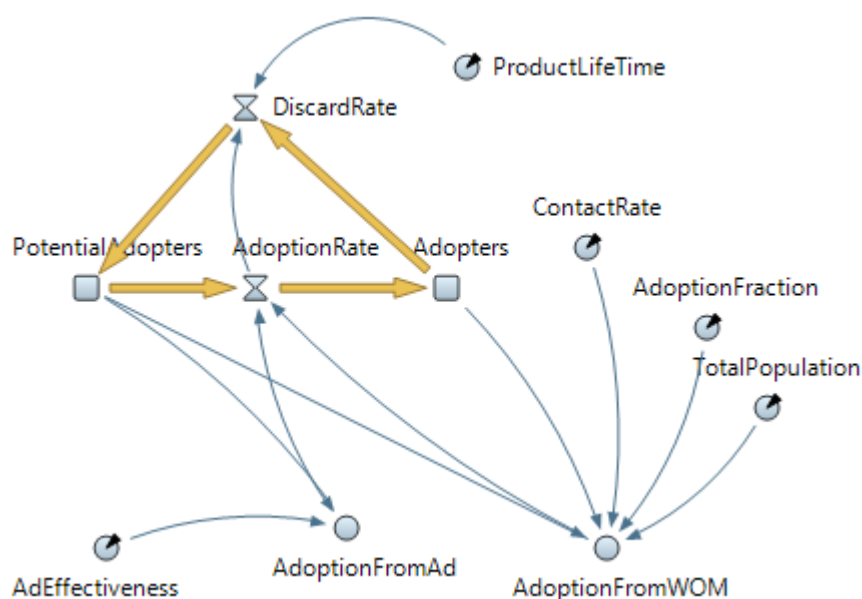


Рис.4.6. Модель, созданная с учетом совершения повторных покупок

Поток `DiscardRate` моделирует интенсивность повторных покупок по формуле:

Протестируйте модель. Динамика процессов модели должна соответствовать рисунку 4.7.

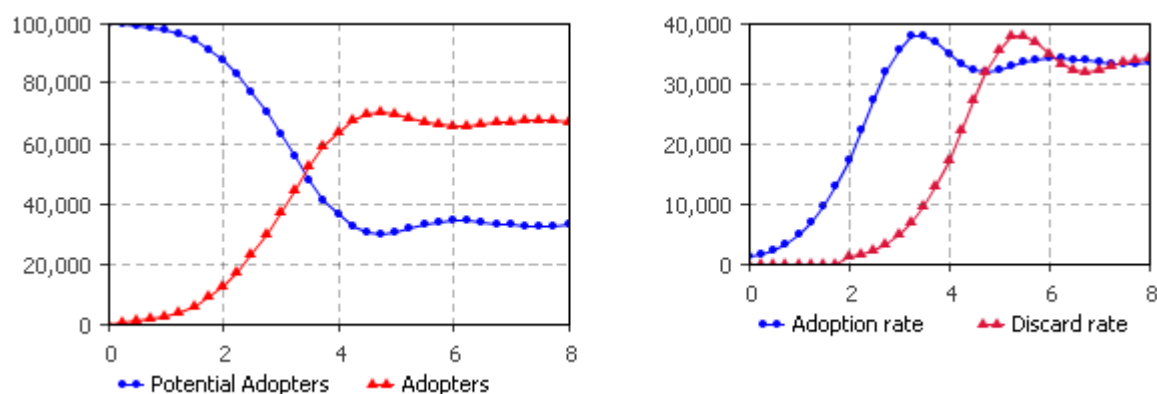


Рис.4.7. Реализация продукта, с учетом повторных покупок

На втором графике показана также динамика изменения интенсивности повторных покупок `Discard rate`.

### 4.3. Контрольное задание «Модель распространения эпидемии»

Постройте модель распространения эпидемии. Модель описывается системой уравнений:

$$\frac{d(\text{susceptible})}{dt} = -\text{get\_stick}$$

Начальное значение накопителя : 1000

$$\frac{d(\text{infected})}{dt} = \text{get\_stick} - \text{get\_well}$$

Начальное значение накопителя : 1

$$\frac{d(\text{recovered})}{dt} = \text{get\_well}$$

$$\text{get\_stick} = \text{infected} \times \text{susceptible} \times \text{infection\_rate}$$

$$\text{get\_well} = \text{infected} \times \text{recovery\_rate}$$

Здесь `get_stick` – интенсивность протекания заболевания, `get_well` – интенсивность выздоровления. Параметры `infection_rate=0.00218` и `recovery_rate=0.5` – факторы, влияющие на процесс заболевания и выздоровления.

Накопители:

- `susceptible` – не заболевшие;
- `infected` – инфицированные;
- `recovered` – выздоровевшие.

Постройте временные графики для трех накопителей. Динамика изменения процесса эпидемии показана на рисунке 4.8.

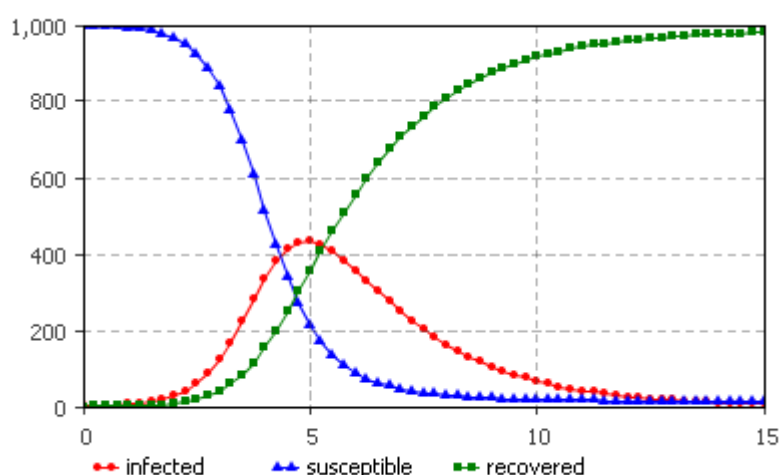


Рис. 4.8. Динамические процессы протекания заболевания

#### 4.4. Взаимодействие активных классов

Разработать модель изменения динамики численности городского населения.

Данная модель должна учитывать динамику роста населения в зависимости от жилищных условий.

Построение такой модели следует выполнить на базе двух взаимодействующих активных классов (см. рисунок 4.9).

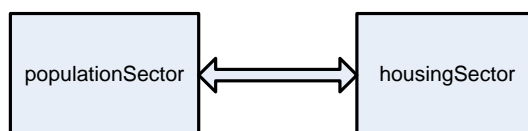


Рис.4.9. Взаимодействующие классы

### Активный класс сектора населения

Активный класс `populationSector` (сектор населения) моделирует динамику роста численности населения, а класс `housingSector` (жилищный сектор) динамику роста жилья.

Для построения модели сектора населения нужно использовать параметры, показанные в таблице 4.2.

Таблица 4.2. Параметры сектора населения

№	Параметр	Описание	Значение
1	<code>birthRate</code>	Уровень рождаемости	0,03
2	<code>imigrationNormal</code>	Коэффициент миграции	0,1
3	<code>populationInitial</code>	Начальная численность населения	50000
4	<code>averageLifetime</code>	Средняя продолжительность жизни	64
5	<code>householdSize</code>	Среднее количество человек в составе семьи	4
6	<code>emigrationNormal</code>	Доля эмиграции	0,07

Уравнение системной динамики численности населения примет вид:

$$\frac{d(\text{population})}{dt} = \text{births} + \text{imigration} - \text{deaths} - \text{emigration}$$

Начальное значение : `populationInitial`

Где:

- `births` – уровень рождаемости;
- `imigration` – уровень миграции;
- `deaths` – уровень смертности;
- `emigration` – уровень эмиграции.



### Переменные модели:

$$\text{householdsToHousesRatio} = \frac{\text{population}}{\text{houses} \cdot \text{householdSize}}$$

$$\text{attractionDueToHousing} = \text{HousingTable}(\text{householdsToHousesRatio})$$

Переменные: `houses` (число домов, которые построены в городе) и `householdToHousesRatio` (заселенность города) образуют интерфейс активного класса сектора населения, `attractionDueToHousing` (спрос на жилье).

Создайте новую модель с «нуля» и разместите в ней активный класс `populationSector`.

### Потоки:

$$\text{births} = \text{birthRate} \cdot \text{population}$$

$$\text{imigration} = \text{population} \cdot \text{imigrationNormal} \cdot \text{attractionDueToHousing}$$

$$\text{deaths} = \frac{\text{population}}{\text{averageLifetime}}$$

$$\text{emigration} = \text{population} \cdot \text{emigrationNormal}$$

В поле активного класса постройте по уравнению системной динамики, с учетом переменных, параметров и потоков модель, используя палитру «Системная динамика». Вид модели должен соответствовать рисунку 4.10.

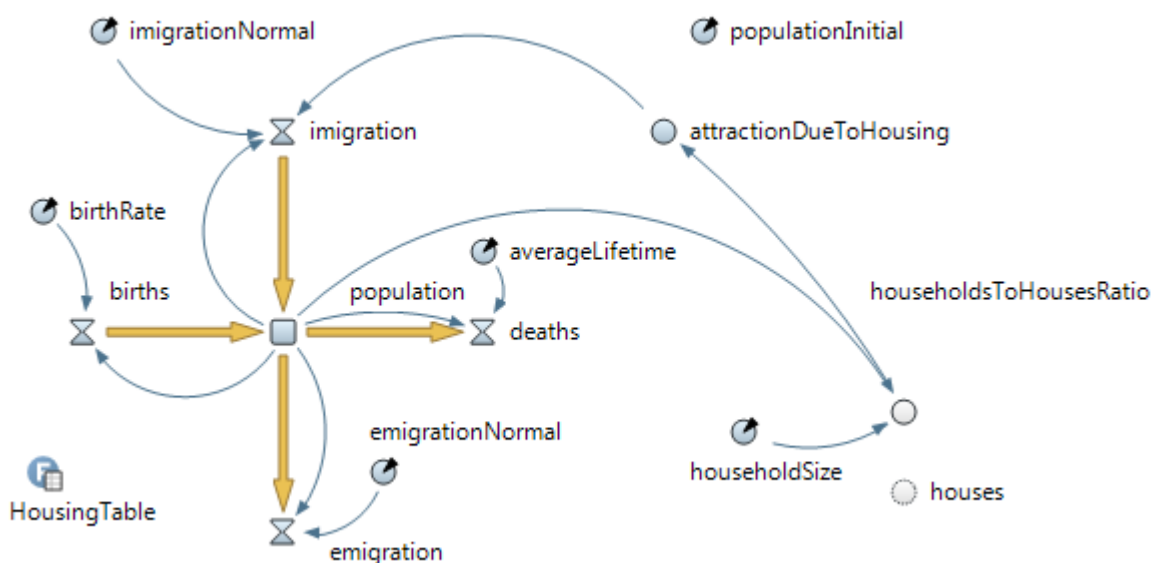


Рис.4.10. Активный класс сектора населения

Примечание: Если в качестве источника в накопитель входит

поток, то он создается из динамической переменной палитры «Системная динамика».

Аналогично создается поток, если данные передаются от накопителя потоку.

При построении активного класса сектора населения нужно в его поле добавить функцию `HousingTable`. Эта функция создается с помощью инструмента «Табличная функция» панели системная динамика. После размещения функции в класс нужно на вкладке «Основные» ее свойств задать ей имя `HousingTable`, тип интерполяции должен быть линейный, если аргумент функции выходит за пределы табличных значений, то выбирается ближайший. Затем следует сформировать таблицу значение функции так, как это показано на рисунке 4.11.

Табличные данные:

Аргумент	Значение
0	1.4
0.2	1.4
0.4	1.35
0.6	1.3
0.8	1.15
1	1
1.2	.8
1.4	.65
1.6	.5
1.8	.45
2	.4

Рис.4.11. Значения функции

Для возможности взаимодействия активного класса с другим активным классом следует создать интерфейс. Для активного класса сектора населения интерфейс будет образован двумя переменными `houses` и `householdToHousesRatio`.

Для создания интерфейса класса разместите в поле класса из палитры «Презентация» элемент «Скругленный прямоугольник» и активизируйте его свойство «Значок». В созданный прямоугольник поместите фигуру человека. Для этого откройте палитру «Картинки» и разместите изображение человека с помощью одноименного

элемента. У изображения активизируйте свойство «Значок».

Создайте копии переменных `houses` и `householdToHousesRatio`. Для этого выделите нужную переменную, вызовите контекстное меню и выполните команду «Создать копию» (см. рисунок 4.12).

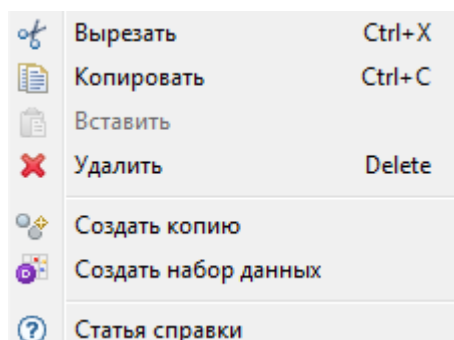


Рис.4.12. Контекстное меню динамической переменной

Оставьте копии переменных на модели, а сами переменные расположите на контуре значка активного класса, так как это показано на рисунке 4.13.

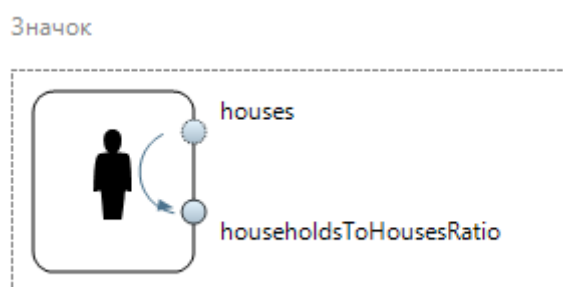


Рис.4.13. Значок активного класса

Для переменной `houses` установите свойства «На верхнем уровне» и «Внешняя» в состояние активности. Такая настройка позволяет создать внешнюю зависимую переменную, которая может принимать значение из другого активного класса.

Для переменной `householdToHousesRatio` установите свойство «На верхнем уровне» в состояние активности. Такая переменная может передавать значение в другой активный класс.

### **Активный класс сектора жилищного строительства**

Создайте новый активный класс `housingSector`. Активный класс обладает набором параметров, которые приводятся в таблице

## 4.3.

Таблица 4.3 . Параметры сектора жилищного строительства

№	Параметр	Описание	Значение
1	constructionNormal	Норма возведения нового жилья	0,07
2	landPerHouse	Доля городской земли на строение	0,1
3	area	Городская площадь	8000
4	housesInitial	Начальное число домов в городе	14000
5	demolitionNormal	Норма сноса ветхого жилья	0,015

Уравнение системной динамики жилищного строительства

примет вид:

$$\frac{d(houses)}{dt} = constructionRate - demolitionRate$$

Начальное значение накопителя : *housesInitial*

$$constructionRate = constrMultiplier \cdot constructionNormal \cdot houses$$

$$demolitionRate = houses \cdot demolitionNormal$$

Переменные модели:

$$constrMultiplier = constrDueToHousingAv \cdot constrDueToLandAv$$

$$constrDueToHousingAv = HousingAvTable(householdsToHousesRatio)$$

$$constrDueToLandAv = LandAvTable(fractionOfOccupiedLand)$$

$$fractionOfOccupiedLand = \frac{houses \cdot landPerHouse}{area}$$

$$housesExport = houses$$

$$householdsToHousesRatio$$

Описание переменных приводится в таблице 4.4.

Таблица 4.4. Переменные сектора жилья

№	Переменная	Описание
1	constrMultiplier	Интенсивность строительства
2	constrDueToHousingAv	Реальная потребность в строительстве
3	constrDueToLandAv	Земельные участки, имеющиеся для строительства

4	fractionOfOccupiedLand	Доля использованной под застройку земли
5	housesExport	Число возведенных домов
6	householdsToHousesRatio	Заселенность города

Вид модели приводится на рисунке 4.14.

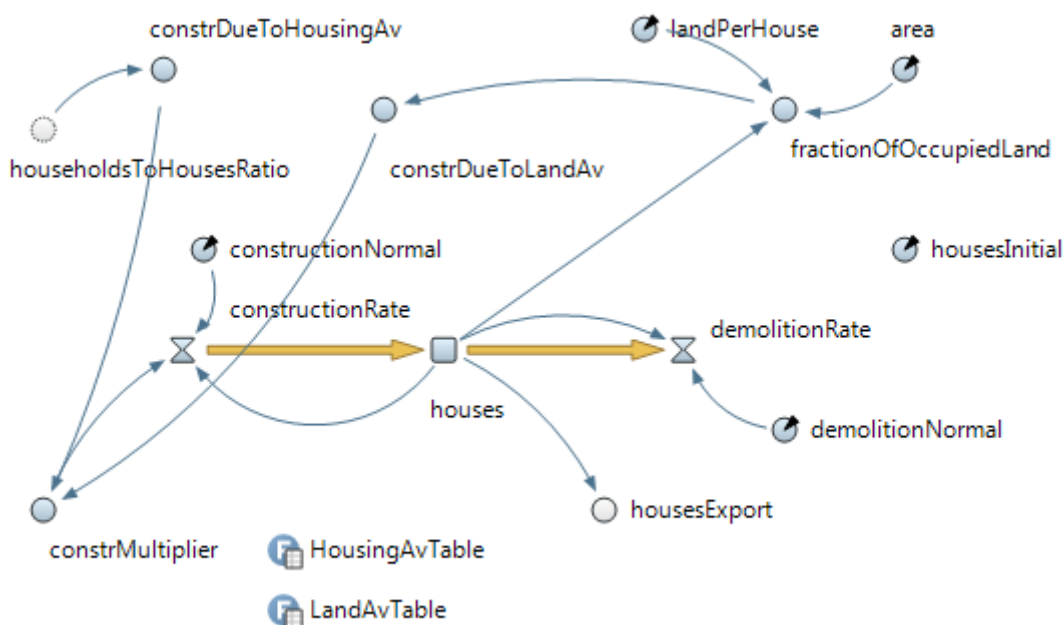


Рис. 4.14. Активный класс сектора жилищного строительства

В модель нужно разместить две табличные функции HousingAvTable и LandAvTable. Значение таблицы первой функции приводятся на рисунке 4.15, а значения второй функции на рисунке 4.16.

Табличные данные:

Аргумент	Значение
0	.2
0.2	.25
0.4	.35
0.6	.5
0.8	.7
1	1
1.2	1.35
1.4	1.6
1.6	1.8
1.8	1.95
2	2

Рис.4.15. Таблица функции HousingAvTable

Обе функции должны реализовывать линейный метод интерполяции, при отсутствии значения берется ближайшее значение.

Табличные данные:

Аргумент	Значение
0	1
0.1	.92
0.2	.83
0.3	.75
0.4	.66
0.5	.57
0.6	.47
0.7	.35
0.8	.24
0.9	.11
1	0

Рис.4.16. Таблица функции LandAvTable

Создайте значок для представления активного класса. В качестве элементов значка используйте «Скругленный прямоугольник» и изображение здания из палитры «Картинки» элемент «Дом». Вид значка показан на рисунке 4.17.

Значок

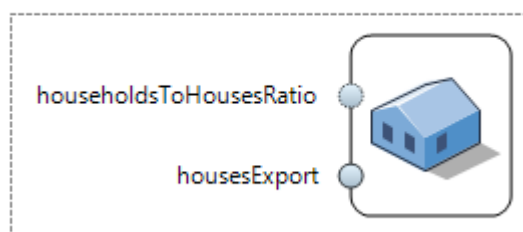


Рис.4.17. Значок активного класса сектора жилья

Создайте интерфейс активного класса. Получите копии переменных `householdsToHousesRatio` и `housesExport`. Копии оставьте на модели, переменные поместите на значок, так как это показано на рисунке 4.17. Настройте свойства переменных таким образом, чтобы переменная `householdsToHousesRatio` могла принимать значение из активного класса, а переменная `housesExport` передавать значение в активный класс.

### Настройка корневого объекта модели

Перейдите в модель активного класса сектора населения и настройте интерфейсные переменные значка таким образом, что бы не отображались их имена, аналогично настройте интерфейс активного класса сектора жилищного строительства.

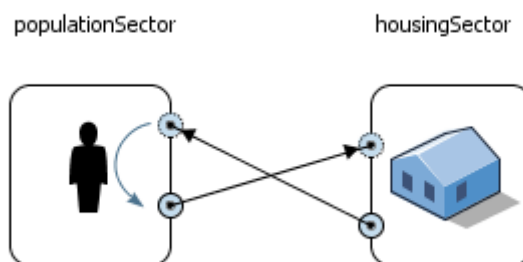


Рис.4.18. Соединение активных классов

Перейдите в графическое поле корневого класса Main и разместите в нем активные классы `populationSector` и `housingSector`. Используйте элемент «Соединитель» палитры «Системная динамика» и соедините классы с помощью их интерфейсов, так как это показано на рисунке 4.18.

Данные можно передавать внешней переменной класса от переменной интерфейса класса источника. Внешние переменные очерчены не сплошной линией. При правильном соединении стрелка должна быть направлена от переменной класса источника к внешней переменной класса приемника.

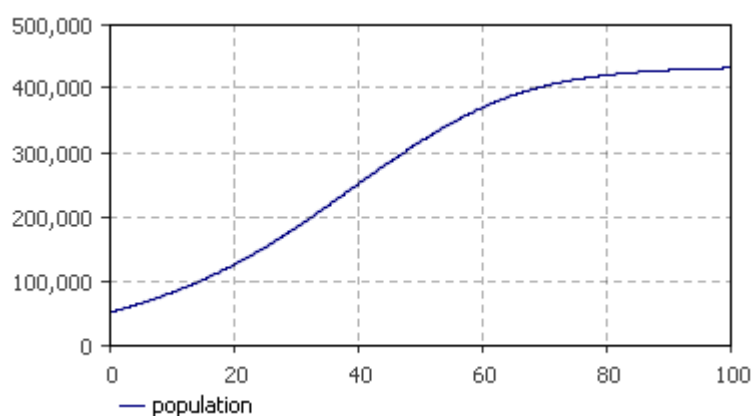


Рис.4.19. График динамики роста населения

Разметите временной график для отображения роста

численности городского населения - population.

Настройте эксперимент модели. На вкладке «Модельное время» задайте единицу измерения модели дни, модель должна останавливаться в заданное время, конечное время задайте равным 100 единиц.

Перейдите на вкладку «Презентация» и переключите режим выполнения в режим выполнения «Виртуальное время».

Протестируйте модель, вид графика изменения численности населения должен соответствовать рисунку 4.19.

#### 4.7. Контрольное задание «Визуализация модели динамики численности населения»

Создайте дополнительный временной график, для вывода динамики изменения городской площади, отведенной под жилищное строительство. Вид графика показан на рисунке 4.20.

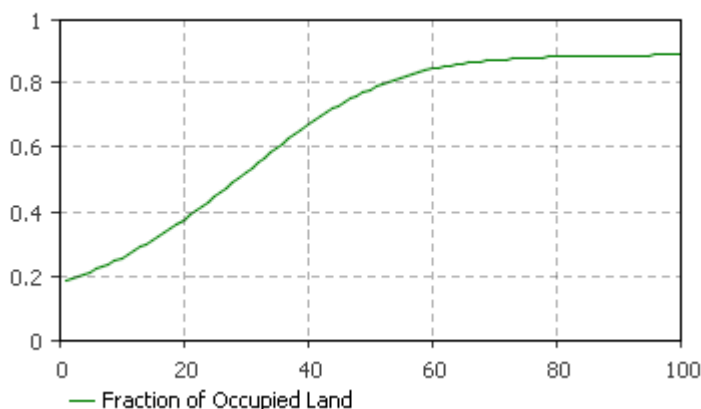
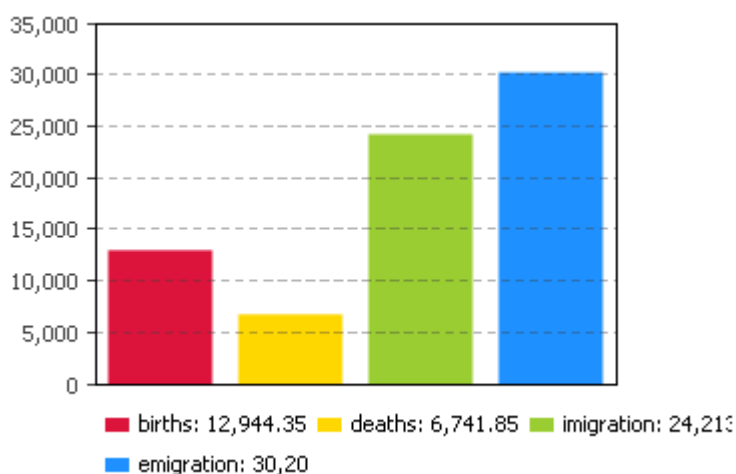


Рис.4.20. Изменение площади под застройку





### Рис.4.21. Распределение населения

Создайте столбиковую диаграмму, которая отображает количество родившихся жителей, умерших, покинувших город, прибывших в город (см. рисунок 4.21).

Протестируйте модель, вид графиков должен соответствовать контрольным рисункам.

### Контрольные вопросы

1. В чем заключается концепция построения моделей системной динамики по Дж. Форрестору?
2. Каков принцип построения уравнения для накопителя модели системной динамики?
3. Дайте классификацию элементов AnyLogic используемых для построения моделей системной динамики.
4. Опишите алгоритм построения модели системной динамики в AnyLogic.
5. Дайте характеристику динамическим переменным AnyLogic.
6. Как строится интерфейс между активными классами в моделях системной динамики?
7. Для чего используется элемент «Табличная функция» в моделях системной динамики?
8. Дайте описание модели реализации продукта по Бассу без учета повторных покупок.
9. Дайте описание модели реализации продукта по Бассу с учетом повторных покупок.
10. Опишите модель развития эпидемии.
11. Дайте описание модели исследования численности активного населения.

## 5. Агентное моделирование

Требуется построить динамическую модель реализации продукции, используя технологию агентного моделирования.

Создайте новую модель. При создании модели выберите шаблон агентной модели, так как это показано на рисунке 5.1.

### Новая модель

Выберите, хотите ли Вы использовать один из шаблонов моделей или начать моделирование "с нуля"

- ☐ Начать создание модели "с нуля" (будет создан чистый холст презентации).
- ☒ Использовать шаблон модели

Выберите метод моделирования:

- ☐ Системная динамика
- ☐ Дискретно-событийное моделирование
- ☐ Моделирование транспортных сетей
- ☒ Агентное моделирование
- ☐ Моделирование движения пешеходов

Рис. 5.1. Создание модели по шаблону

После перехода к следующему шагу следует задать свойства агента, так как это показано на рисунке 5.2.

### Создание агентной модели

Задайте свойства агента

Имя класса агента:

Начальное количество агентов:


 people [...]

Рис.5.2. Задание свойств агента.

Затем на следующем шаге нужно задать свойства агентного пространства и анимации в соответствии с рисунком 5.3.

### Создание агентной модели

Задайте свойства пространства и анимации

☒ Добавить пространство   
 ☒ Непрерывное   
 ☐ Дискретное  
 Ширина:    
 Высота:    
 Нач. расположение:   
 Столбцов:    
 Строк:    
 Анимация:   
 people [...]  
 environment

Рис. 5.3. Задание свойств пространства агентов

На следующем шаге нужно задать свойства сети, так как это показано на рисунке 5.4.

### Создание агентной модели

Задайте параметры сети

☒ Использовать сеть  
☒ Случайное   
 ☐ Решеточное кольцо   
 ☐ Малый мир   
 ☐ Безразмерная   
 ☐ На расстоянии  
☒ Показывать связи   
☐ Добавить случайное движение  
 people [...]  
 environment

Рис.5.4. Параметры сети агентов

Последний шаг конфигурации модели заключается в задании поведения агентам, так как это показано на рисунке 5.5.

## Создание агентной модели

Добавить простое поведение и сбор статистики по агентам

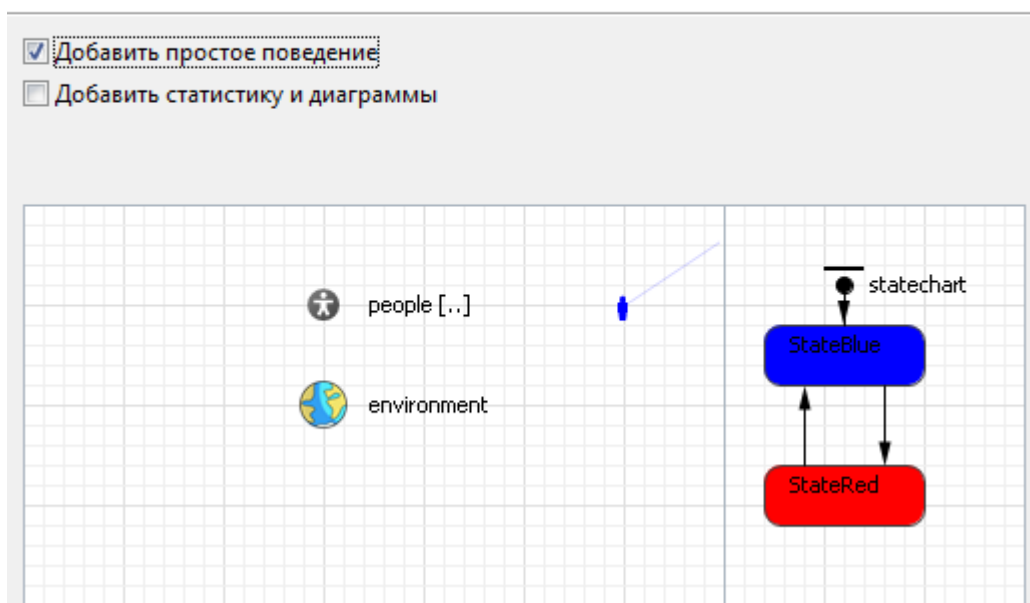


Рис.5.5. Задание поведения агента

### 5.1. Моделирование покупки товара под влиянием рекламы

На дереве проекта выберите активный класс `Person`. Настройте конечный автомат (стейчарт) объекта в соответствии с рисунком 5.6.

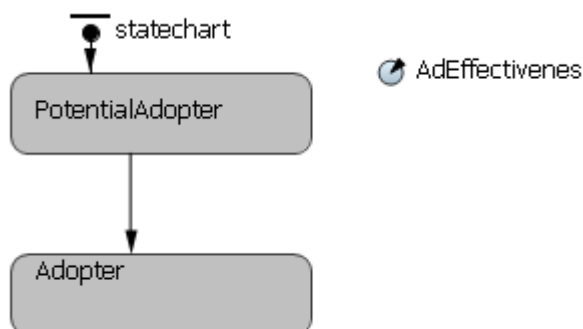


Рис.5.6. Модель класса Person

Цвет заливки состояний – серого цвета.

Действие при входе для состояния `PotentialAdopter`:

```
person.setFillColor(new Color(0,0,255));
```

Потенциальные потребители будут отображаться синим цветом.

Действие при входе для состояния `Adopter`:

```
person.setFillColor(new Color(255,0,0));
```

Потребители товара будут отображаться красным цветом.

Добавьте параметр `AdEffectiveness=0.011`, который задает влияние рекламы на процесс приобретения товара.

Переход выполняется с заданной интенсивностью равной параметру.

Настройте эксперимент. Модельное время измеряется в минутах, конечное время равно 8. Режим остановки в заданное время.

После запуска модели она будет иметь вид, показанный на рисунке 5.7.

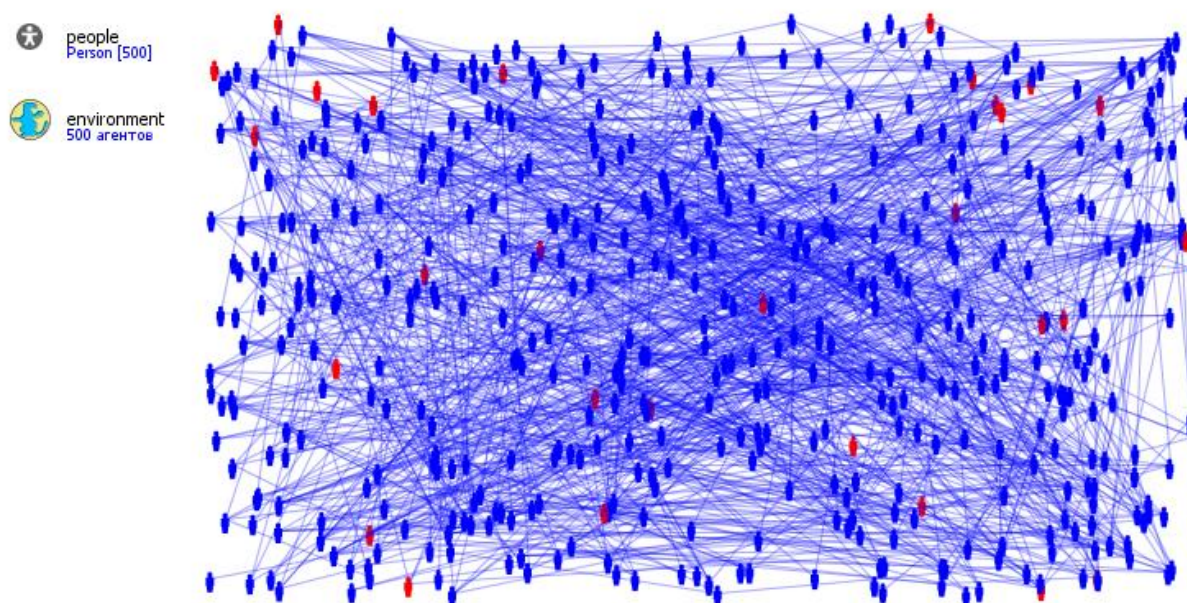


Рис. 5.7. Модель после запуска

## 5.2. Сбор статистики

Для сбора статистики в модели нужно перейти в редактор корневого класса модели и выбрать реплицированный класс `people`. В окне свойств используется вкладка «Статистика».

На этой вкладке нужно создать функции сбора статистики. Общий вид окна статистики показан на рисунке 5.8.

Нужно определить текущее количество потенциальных потребителей (`potentialAdopters`), и количество потребителей продукта (`adopters`). Выражения для условий представлены в таблице.

The screenshot shows a configuration window with a sidebar on the left containing tabs: 'Основные', 'Параметры', 'Статистика' (selected), and 'Описание'. The main area contains two identical-looking forms for defining statistical functions.

**Form 1 (top):**

- Имя:
- Тип: ☒ Кол-во ☐ Сумма ☐ Среднее ☐ Мин. ☐ Макс.
- Выражение:
- Условие:

**Form 2 (bottom):**

- Имя:
- Тип: ☒ Кол-во ☐ Сумма ☐ Среднее ☐ Мин. ☐ Макс.
- Выражение:
- Условие:

Рис.5.8. Задание статистических функций

Описание созданных статистических функций приводится в таблице 5.1.

Таблица 5.1. Функции сбора статистики

Имя	Действие	Условие
potentialAdopters	Подсчет количества потенциальных потребителей	<i>item.statechart.isStateActive(Person.PotentialAdopter);</i>
adopters	Подсчет количества потребителей	<i>item.statechart.isStateActive(Person.Adopter);</i>

Подсчет количества выполняется при входе в активности соответствующего состояния объекта `statechart` методом:

```
public boolean isStateActive(short state)
```

Где: `item` – указатель на текущий реплицированный объект, который используется при подсчете AnyLogic статистического показателя.

Разместите в поле корневого объекта временной график для вывода статистических показателей. Вид графика, после запуска модели показан на рисунке 5.9.

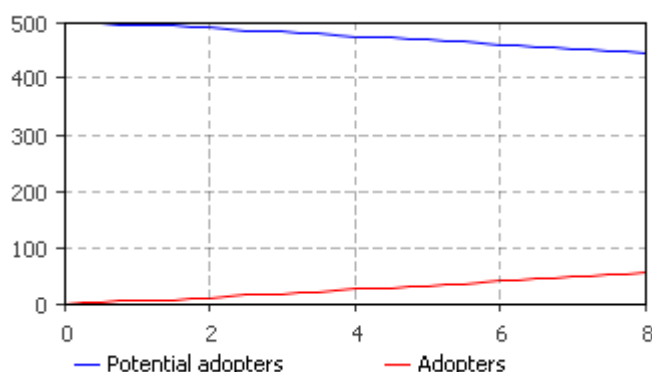


Рис.5.9. Продажа под действием рекламы

При формировании графика нужно учитывать, что у класса `people` статистику возвращают функции, имена которых заданы в свойстве «Имя».

### 5.3. Учет влияния общения между потребителями

Требуется учесть в модели продаж влияние общения между потребителями продукта на покупку товара.

Откройте активный класс `person` и измените его модель в соответствии с рисунком 5.10.

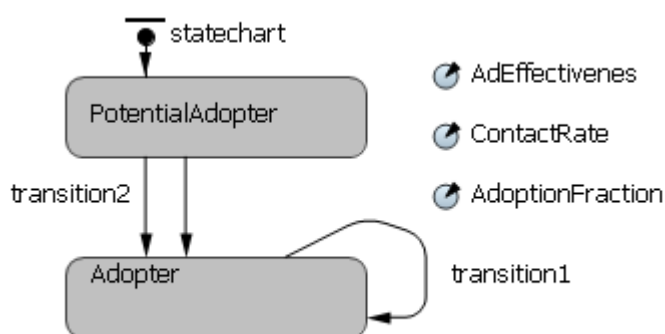


Рис.5.10. Учет общения потребителей

Для учета общения потребителей нужно добавить в модель два параметра `ContactRate=100` (интенсивность контактов), `AdoptionFraction=0.015` (вероятность покупки потенциальными потребителями продукта).

Переход `transition1`, моделирует общение потребителей. Он происходит с интенсивностью равной `ContactRate`. При этом должно формироваться сообщение о желании купить продукт. Такое действие

моделируется с помощью оператора AnyLogic

```
send("buy!", RANDOM_CONNECTED) .
```

Константа `RANDOM_CONNECTED` означает, что сообщение адресовано произвольному потенциальному покупателю. Выполнение перехода `transition2` моделирует покупка товара потенциальным покупателем, он происходит при получении сообщения `"buy!"` символьного типа. Что бы моделировать случайный характер покупок введем дополнительное условие по параметру `AdoptionFraction`, используя функцию `randomTrue(AdoptionFraction)`. Функция возвращает значение `true` с заданной вероятностью `AdoptionFraction`.

Кроме того упорядочим сеть агентов. Изменим характер сети (`environment`).

Откройте корневой объект модели и выберите сеть. В окне свойств на вкладке «Дополнительные» измените тип сети на «Согласно расстоянию». Задайте радиус соединения равным 25.

Протестируйте модель. Вид сети агентов должен быть упорядоченным (см. рисунок 5.11), а график динамических процессов должен соответствовать рисунку 5.12.



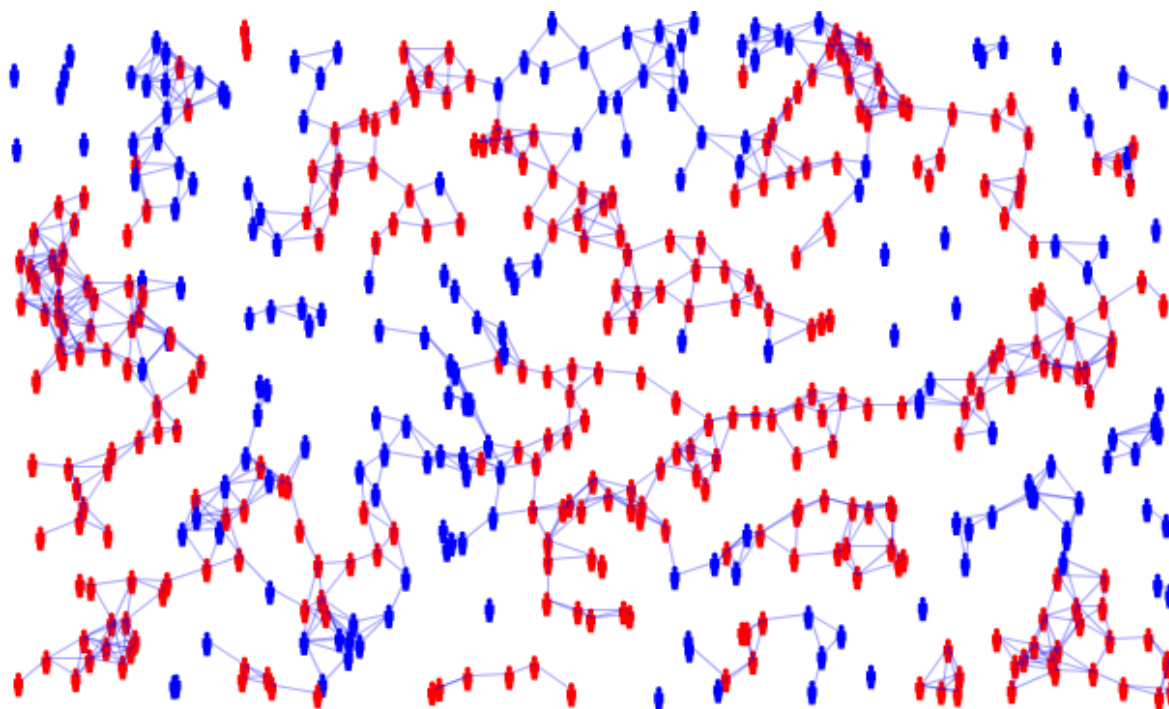


Рис.5.11. Упорядоченная сеть агентов

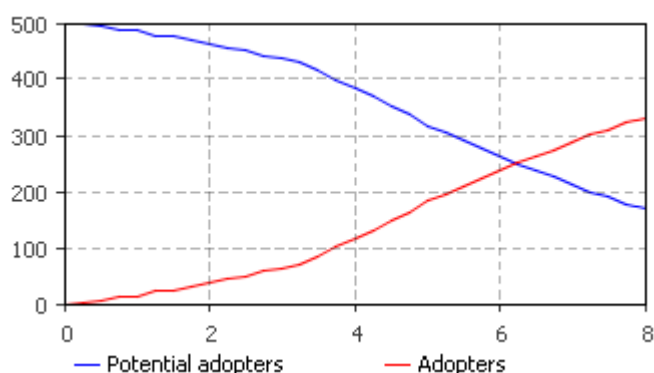


Рис.5.12. Учет общения потребителей

#### 5.4. Контрольное задание «Учет повторных покупок агентами»

Дополните модель покупок, таким образом, чтобы в ней учитывались повторные покупки товара по истечению контрольного срока его службы  $DiscardTime=1$  (году).

Методическое указание: в рамках модели агентов, повторная покупка это переход потребителя в разряд потенциальных покупателей.

Вид динамических процессов должен соответствовать рисунку 5.13

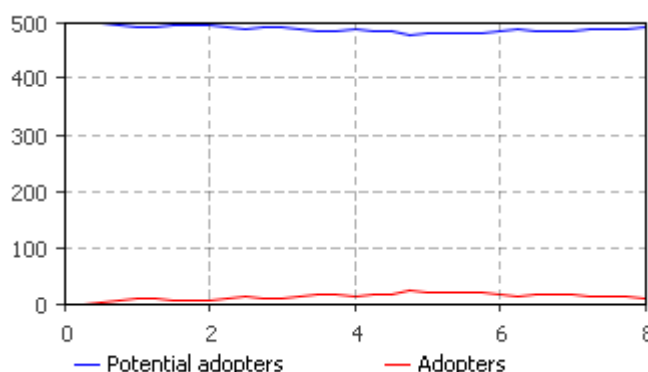


Рис.5.13. Динамика процессов с учетом повторных покупок

### Контрольные вопросы

1. Что представляет собой агент модели?
2. Какими основными свойствами обладает агент модели?
3. Как можно задать поведение агентов?
4. Что подразумевается под средой агентной модели?
5. Дайте описание процесса построения агентной модели с использованием шаблона AnyLogic?
6. Как выполняется сбор статистических показателей для агента модели?
7. Что такое сеть агентов и каковы ее особенности?
8. Дайте описание агентной модели покупки товаров потребителями.
9. Как учесть повторные покупки потребителями товара?
10. Какие статистические показатели определяются в модели покупки товаров.
11. Объясните разницу между графиками покупок из-за повторной покупки товаров клиентами.

### 6 Задания для самостоятельной работы

Задание №1.

Дана функция  $f(a,b) = \frac{\sin(3\pi a) + \cos(\pi b)}{a} - b$

Аргумент функции  $a$  принадлежит диапазону  $a \in [0.1, \dots, 3]$ . Аргумент  $b$  равен 2.

Построить дискретно-событийную модель для исследования функции. Аргумент  $a$  изменяется с шагом равным 0,1.

Для исследования влияния параметра  $b$  разместите в модель элемент слайдер. Данный элемент должен позволять менять значение параметра  $b$  в процессе моделирования от 2 до 10.

Единицы модельного времени секунды, период моделирования равен 30 секундам.

Результаты моделирования показаны на рисунке 6.1.



Рис.6.1. модель при  $b=2$

## Задание №2.

Постройте дискретно-событийную модель для отображения движения материальной точки по траектории, которая задается функцией задания №1. Движение происходит в прямоугольной области. Параметр  $b=2$ .

Материальная точка – шарик. Если значение координаты траектории по оси  $Y$  больше или равно нулю, то шарик окрашен в синий цвет, иначе в красный. В прямоугольной области выведите значение функции (координата по оси  $y$ ), округленное до второго знака после запятой. Координата должна меняться в процессе моделирования.

Вид модели показан на рисунке 6.2

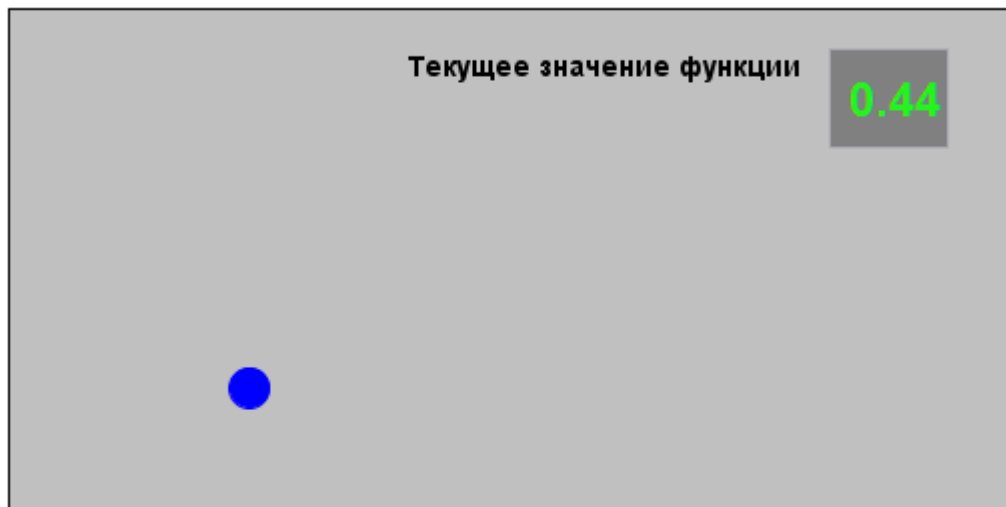


Рис.6.2. Движение материальной точки

Задание №3.

Постройте модель в виде конечного автомата для вычисления по входному множеству значений  $\Sigma=\{A,B,C\}$  выходного множества  $\Omega=\{D,E_1,E_2\}$ .

Преобразование входных значений в выходные осуществляется по формулам:

$$D = A^2 - B^2 - C^2$$

$$E_1 = \begin{cases} \frac{A-B}{C}; & C < B < A \\ 0 & ; C = 0 \\ A \cdot B \cdot C; & \text{в остальных случаях} \end{cases}$$

$$E_2 = \begin{cases} \sqrt{D}; & D \geq 0 \\ e^D; & D < 0 \end{cases}$$

Контрольные значения:

Вариант а:

$$\Sigma=\{10,9,11\} \rightarrow \Omega\{-102,990,5.03457E-45\}$$

Вариант б:

$$\Sigma=\{10,2,1\} \rightarrow \Omega\{95,8,9.7468\}$$

Вид модели показан на рисунке 6.3.

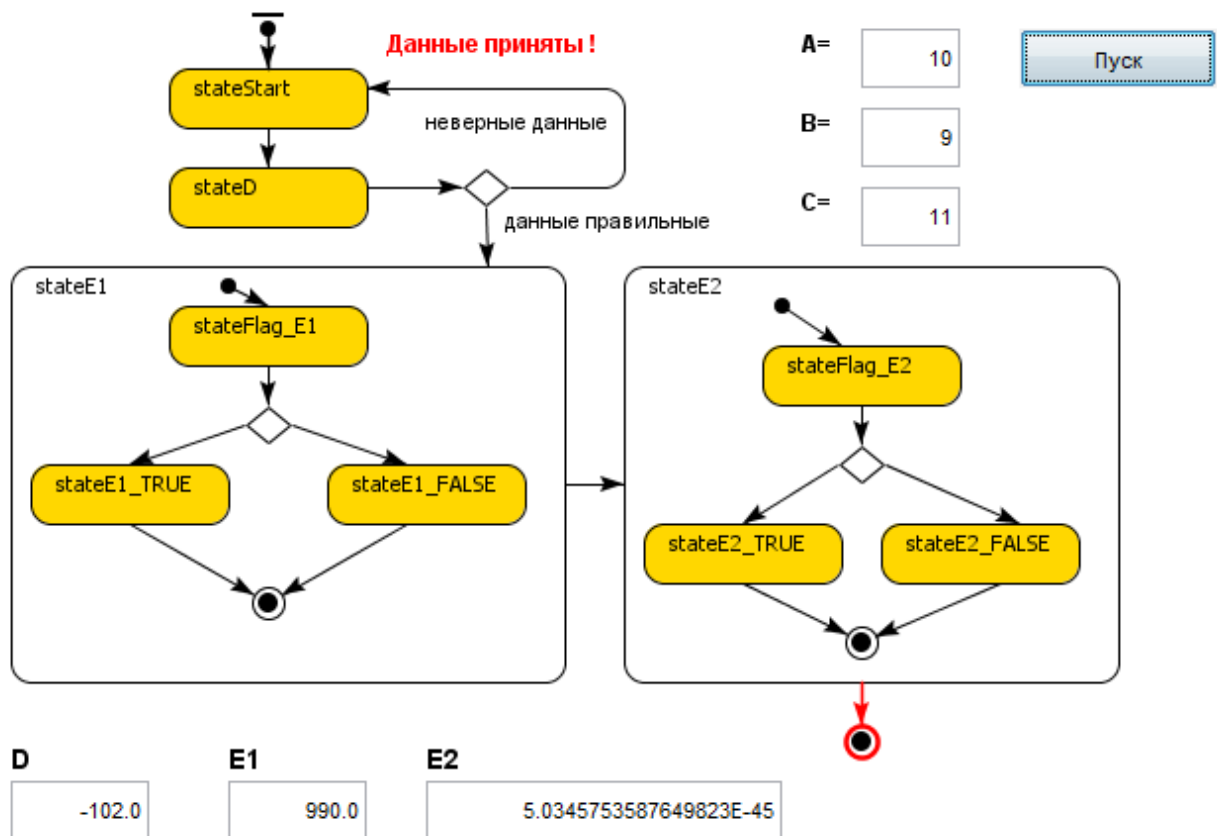


Рис.6.3. Модель конечного автомата. Контрольный набор а

Значения входного множества вводятся в текстовых полях, значения выходного множества также выводятся в текстовых полях. Ввод данных в автомат начинается после нажатия кнопки «Пуск». Если введены данные не числовые, то выводится сообщения об ошибке и данные запрашиваются снова.

Задание №4.

Постройте модель жизни популяции, используя модель системной динамики.

Модель популяции описывается уравнениями:

$$\frac{d(\text{population})}{dt} = \text{births} - \text{deaths}$$

$$\text{births} = \text{birthsRate} \times \text{population}$$

$$\text{deaths} = \text{deathRate} \times \text{population} + \text{stabilityFactor} \times \text{population}^2$$

$$\text{StabRate} = \frac{\text{birthRate} - \text{deathRate}}{\text{stabilityFactor}}$$

Здесь birthRate- коэффициент роста популяции на одного индивидуума, deathRate- коэффициент гибели, stabilityFactor – коэффициент замедления роста.

Интенсивность рождения определяется значением births, а смертность в популяции значением deaths. Значение StabRate – определяет устойчивое значение популяции.

Модель популяции создайте в виде активного класса, создайте значок. Разработайте интерфейс активного класса в виде двух переменных: StabRate и outPopulation. Переменная outPopulation возвращает текущую численность популяции (см. рисунок 6.4).

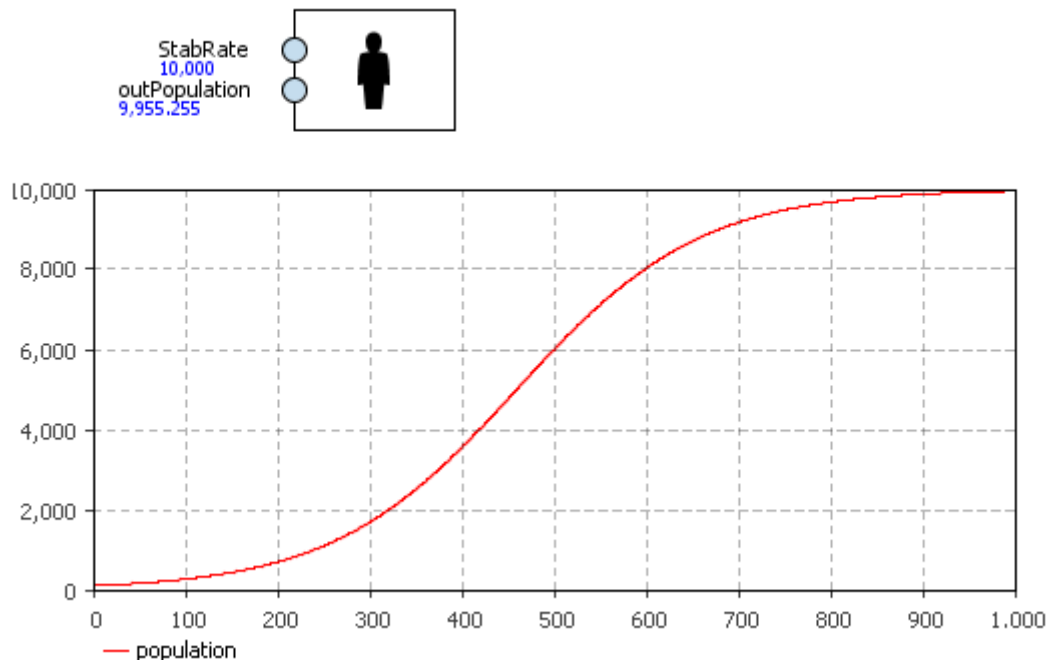


Рис.6.4. Модель жизни популяции

Разместите активный класс в поле корневого класса модели и постройте временной график изменения численности популяции при следующих исходных данных:

birthRate=0,02

deathRate=0,01

stabilityFactor=0.000001

Начальное значение популяции равно 100 особей. Вид работающей модели жизни популяции показан на рисунке.

Единицы модельного времени – дни, период моделирования 1000 дней. Модель должна выполняться в режиме виртуального времени.

## Приложение

### Основные сведения о языке программирования Sun Java

#### Классы Java

Язык программирования Java является полностью объектно-ориентированным. Поэтому программный код должен быть записан внутри определенного класса. В общем виде структура любого класса Java имеет следующий вид:

```
доступ class MyClass{
    доступ тип имя_атрибута1;
    доступ тип имя_атрибута2;
    ...
    доступ тип_результата имя_метода1 (аргументы) { };
    доступ тип_результата имя_метода2 (аргументы) { };
    ...
}
```

В качестве Доступа выбирается один из трех: `public`(глобальный), `private`(закрытый), `protected`(защищенный). Тип и Тип\_результата – любой из допустимых типов Java. В языке Java доступ может не указываться, в этом случае элементы класса и класс считаются глобальными в рамках данной программы или пакета т.е не могут экспортироваться в другой пакет.

Атрибуты класса является его элементами и могут рассматриваться как переменные.

Методы класса используются для выполнения действий над его атрибутами. В общем случае метод обладает следующей структурой

```
доступ тип_результата имя_метода (аргументы) {
    //локальные переменные и операторы
};
```



Аргументы могут отсутствовать. В общем виде это список формальных параметров метода. Каждый параметр задается в виде `t p` где `t` – тип параметра допустимый по синтаксису Java, а `p` имя параметра. Если метод не возвращает никаких результатов, то в качестве типа результата указывается значение `void`. Если метод возвращает результат, то он должен содержать оператор `return v` где `v` – возвращаемое значение. Его тип должен соответствовать типу результата метода.

Для использования класса нужно получить его экземпляр – объект. Получение экземпляра выполняется с помощью оператора `new`:

```
MyClass objMy=new MyClass();
```

Допускается выполнять получение экземпляра в два этапа.

```
MyClass objMy;
```

```
objMy=new MyClass();
```

Получив объект можно обращаться к его методом и атрибутам, если доступ позволяет это сделать. Для этого используется оператор разыменования `objMy.m()` либо `objMy.a=значение`.

Если метод класса не содержит формальных параметров, то все равно следует при вызове метода указать пустые скобки.

Примеры классов.

Пример №1. Класс с параметризованным методом

```
class ExeClass1{
    double x;
    public void m(double y){
        x=y*y;
    }
}
//Получение объекта
ExeClass1 objExe=new ExeClass1();
objExe.m(3.14);
double z=objExe.x;
```

Пример номер №2. Класс с не параметризованным методом

```

class ExeClass2{
    public double y;
    public double m(){
        double x;
        x=y*y;
        return x;
    }
}
//Получение объекта
ExeClass2 objExe=new ExeClass2();
objExe.y=3.14;
double z=objExe.m();

```

В составе класса может быть специальный метод – конструктор. Конструктор не может возвращать результат, но может принимать фактические значения. Ключевое слово `void` для конструктора не используется. Конструктор автоматически вызывается при получении экземпляра класса – объекта. Если конструктора нет, то используется конструктор по умолчанию. В примере №2 это `ExeClass2()`. Допускается создавать несколько конструкторов различающихся набором формальных параметров.

**Пример №3. Использование конструктора.**

```

class ExeClass3{
    private double y;
    //Конструктор
    ExeClass3(double a){
        y=a;
    }

    public double m(){
        double x;
        x=y*y;
        return x;
    }
}

```

```
//Получение объекта
ExeClass3 objExe=new ExeClass3(3.14);
double z=objExe.m();
```

При написании кода класса может использоваться комментарий.  
Однострочный комментарий имеет вид:

```
//Текст комментария
```

Многострочный комментарий:

```
/*
Это комментарий
Из нескольких строк
*/
```

## Типы данных Java

В языке используются типы данных, приведенные в таблице 1.

Таблица 1. Типы данных

Тип	Разрядность	Особенности
long	64	Целый тип -9,223,372,036,854,775,808 до 9,223,372,036,854,775,807
int	32	Целый тип от -2,147,483,648 до 2,147,483,647
short	16	Целый тип с диапазоном кодирования: +32768 до -32767
byte	8	Целый тип с диапазоном кодирования -128 до +127
double	64	Вещественный тип с диапазоном кодирования: +1.7e±308 до -1.7e±308
float	32	Вещественный тип с диапазоном кодирования: +3.4e±38 до -3.4e±38
char	16	Код символа в международной кодировке Unicode.
boolean		Логические тип данных для хранения одного из двух значений true или false

Переменная задается в следующем виде:

```
тип переменная=начальное_значение;
```

Задавать начальное значение не обязательно.

При работе с числами рекомендуется использовать типы `int` и `double`, так как современные процессы такие типы обрабатывают с оптимальной скоростью. Кроме того, многие математические методы Java возвращают результат как `double`.

### **Присваивание значений в выражениях**

В общем виде оператор присваивания имеет вид:

```
v=результат_выражения;
```

v – переменная определенного типа.

При выполнении присваивания нужно учитывать правила приведения типов. Если тип переменной в левой части и тип результата выражения совместны, то происходит автоматическое преобразование типов. Такое преобразование возможно, если выполнено два условия:

- два типа совместимы;
- тип, к которому выполняется приведение, обладает большей разрядностью, чем исходный.

При выполнении этих условий говорят о расширяющем преобразовании. Например, к типу `int` (разрядность 32) возможно преобразование типа `byte` (8 разрядов).

В языке Java определены следующие правила расширения:

- если операнды в выражении относятся к типам `byte` и `short` то они автоматически расширяются до типа `int` перед проведением вычислений;
- если один операнд имеет тип `long`, тип целого выражения расширяется до `long`;
- если один операнд – типа `float`, то тип всего выражения расширяется до `float`;
- если тип любого из операндов – `double`, то тип результата – также `double`.

Результат деления целых чисел дает вещественное значение.

При преобразовании переменных несовместимых типов используется явное приведение типов:

`v=(type) результат_выражения;`

Где: `type` тип переменной в левой части оператора присваивания.

Преобразование такого типа бывает:

- Сужающим – когда выполняется приведение от типа с большей разрядностью к типу с меньшей разрядностью.
- Усеченным – когда вещественный тип преобразуется к целому при этом дробная часть числа отбрасывается.

## Операции Java

### Арифметические операции языка

Основные арифметические операции сведены в таблицу 2.

Таблица 2. Арифметические операции

Операция	Назначение
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Получение остатка от деления
++	Инкремент
+=	Присваивание со сложением
-=	Присваивание с вычитанием
*=	Присваивание с умножением
/=	Присваивание с делением
%=	Присваивание с модулем (остаток от деления)
--	Декремент

Кроме арифметических операций в языке Java определены битовые – поразрядные операции. Они применимы к переменным типа `long`, `int`, `byte`, `char`, `short` (см. таблицу 3).

Таблица 3. Поразрядные операции

Операция	Назначение
~	Отрицание
&	Конъюнкция (И)
	Дизъюнкция (ИЛИ)
^	Исключающее ИЛИ
>>	Правый сдвиг
>>>	Сдвиг в право с заполнением старшего бита нулем
<<	Левый сдвиг
&=	Присваивание И
=	Присваивание ИЛИ
^=	Присваивание с исключающим ИЛИ
>>=	Присваивание с правым сдвигом
>>>=	Присваивание со сдвигом вправо, старший бит заполняется нулем
<<=	Присваивание со сдвигом влево

При составлении условий используются логические операции из таблицы 4.

Таблица 4. Логические операции

Операция	Назначение
==	Сравнение
!=	Не равно
>	Больше
<	Меньше
>=	Больше, либо равно
<=	Меньше, либо равно

Результат логической операции – значение типа `boolean`. Результаты выполнения логических операций можно использовать в логических выражениях. Такие выражения составляют с помощью булевых операций таблицы 5.

Таблица 5. Булевы операции

Операция	Назначение
!	Отрицание
& &	Логическая операция И
	Логическая операция ИЛИ
^	Логическая операция исключающее ИЛИ

## Приоритет в выражениях

Операции в выражениях выполняются в соответствии с определенным приоритетом таблицы 6.

Таблица 6. Приоритеты

Приоритет	Операции
1	++, --, ~, !
2	*, /, %
3	+, -
4	>>, >>>, <<
5	>, >=, <, <=
6	==, !=
7	&
8	^
9	
10	&&
11	

Для изменения приоритета в выражениях используются круглые скобки.

## Управляющие операторы

Условный оператор.

```
if (условие) {
    //Операторы при условии = true
}
else {
    //Операторы при условии = false
}
```

условие – логическое выражение. Допускается не использовать ветку `else` «иначе».

Многозвенный оператор ветвления.

```
if (условие) {
}
else if (условие1) {
}
```

```

else if (условие2) {
}
else {
}

```

Операторы `if` выполняются последовательно сверху вниз. Как только одно из условий становится равным `true`, то выполняется оператор, связанный с этим `if`, а остальные проверки условий пропускаются. Если ни одно из условий не даст значение `true`, то будет выполнен последний оператор `else`.

Заключительная часть `else` действует как условие по умолчанию, если все другие условные проверки не успешны.

Оператор селектор.

```

switch (условие) {
case значение_1{
//операторы
break
}
case значение_2{
//операторы
break
}
...
default{
//операторы
}
}

```

Здесь `условие` – селектор переменная или выражение, которое должно возвращать значение типа `byte`, `int`, `short`, `char`.

Выполняется та часть блока `case`, где значение совпадет с селектором, оператор `break`, завершает работу оператора `switch`.

Если совпадений нет, то выполняются операторы блока `default`. Этот блок может отсутствовать.



Оператор цикла `while`:

```
while (условие) {
//операторы
}
```

Выполняется до тех пор, пока условие равно `true`.

Оператор цикла `do while`.

```
do {
//операторы
}
while (условие);
```

Такой цикл выполняется всегда один раз.

Цикл типа `for`.

```
for (exp1;exp2;exp3){
//операторы
}
```

Используется для выполнения тела цикла определенное число раз.

Где:

`exp1` – начальное значение счетчика цикла;

`exp2` – условие выхода из цикла;

`exp3` – выражение для изменения счетчика цикла.

Для управления работой циклов используются операторы: `break` и `continue`.

Первый оператор позволяет прервать работу цикла. Для передачи управления на заданный оператор используется формат оператора с меткой `break метка_оператора`; Метка оператора должна заканчиваться двоеточием.

Второй оператор служит для перехода к следующему шагу – итерации цикла. Все операторы лежащие «ниже» его в теле цикла пропускаются. В циклах `while` и `do while` выполнение оператора приводит к передаче управления условию, а в цикле `for` выражению, которое изменяет счетчик цикла. Оператор может быть использован с

меткой. Метка должна помечать оператор цикла, которому передается управление. Такая форма используется при создании вложенных циклов.

Оператор `break` используется также для завершения работы оператора ветвления с передачей управления на определенный помеченный оператор.

Оператор `return`. Оператор используется для принудительного завершения работы метода класса. Если метод возвращает значение, то оператор должен содержать аргумент, тип которого совпадает с типом возвращаемого значения:

```
return аргумент;
```

В языке Java нет оператора `goto`.

### Массивы и их задание

Массивы в языке Java могут быть как одномерными, так и многомерными.

а) одномерный массив:

```
тип имя_переменной[]=new тип[число_элементов];
```

б) многомерный массив:

```
тип имя_переменной[][]=new тип[строки][столбцы];
```

Примеры:

```
int vector[]=new int[5];
```

```
int numbers={59,89,78,90,35};
```

```
doudble matrix[][]=new double[3][3];
```

Индексация элементов массива осуществляется с 0. Массив – объект, он обладает свойством `length`, которое возвращает число элементов в массиве.

### Обработка строк

В Java отсутствует тип для работы со строками. Вместо него используется специальный класс `String`. Строка – экземпляр этого класса.

Пример задания строки:

```
String mes="Hello World !".
```

В строки допускается включать управляющие ESC последовательности:

- \n – создание новой строки;
- \t – табуляция;
- \r – переход в начало строки;
- \b – возврат на один символ.

Например:

```
String info="Клиенты\nна сегодня:"
```

Базовыми операциями со строками являются присваивание и конкатенация с помощью символа +. Для работы со строками используются методы класса String приведенные в таблице 7.

Примеры строковых конструкторов:

```
String MyStr="New string";
String MyStr= new String("Test");
char charArray[]={ 'T', 'e', 's', 't' }
String MyStr=new String(charArray);
```

Конструкторы строки которые используют массивы ASCII кодов:

```
String(byte asciiChars[])
String(byte asciiChars[],
int startIndex,int numChars)
```

Где: startIndex – начальный индекс массива, numChars – число символов.

Таблица 7. Строковые методы

Методы	Назначение
<code>equals(s)</code>	Сравнение двух строк
<code>substring(n1,n2)</code>	Извлечение подстроки
<code>trim()</code>	Удаление пробелов
<code>length()</code>	Получение длины строки
<code>indexOf(s)</code>	Определение вхождения одной строки в другую
<code>equalsIgnoreCase(s)</code>	Проверка на равенство двух строк без учета

	регистра
<code>charAt (i)</code>	Выделение отдельного символа
<code>toLowerCase ()</code>	Переход к нижнему регистру
<code>toUpperCase ()</code>	Переход к верхнему регистру

Примечания:

символы в строке индексируются, начиная с 0;

В таблице 7:  $n1$  – начальная позиция,  $n2$  – конечная позиция символов. При выделении подстроки номер последнего символа не учитывается.

В случае удачного поиска возвращается позиция вхождения подстроки, иначе -1.

Получить массив ASCII кодов из строки можно с помощью метода `byte[] getBytes ()`.

### Класс Math. Математические функции

При программировании вычислений могут быть использованы методы класса Math. Методы, реализующие математические встроенные функции возвращают результат типа `double` (см. таблицу 8).

Таблица 8. Математические методы

Математическое описание	Метод
$\sin(x)$	<code>sin(x)</code>
$\cos(x)$	<code>cos(x)</code>
$e^x$	<code>exp(x)</code>
$\ln(x)$	<code>log(x)</code>
$y^x$	<code>pow(y, x)</code>
$\sqrt{x}$	<code>sqrt(x)</code>
$ x $	<code>abs(x)</code>
Округление, число большее или равное $x$	<code>ceil(x)</code>
Округление, число меньшее или равное $x$	<code>floor(x)</code>

Округление float, к ближайшему int	<code>round(x)</code>
Минимальное значение из двух переменных	<code>min(x, y)</code>
Максимальное значение из двух переменных	<code>max(x, y)</code>
Случайное значение double в интервале от 0 до 1	<code>random()</code>

Пример. Вычисление с округлением до третьего знака после запятой.

```
double x, y, z;
x=78.98;
y=7.45;
z=Math.sqrt(x*x+y*y);
z=Math.ceil(z*1000)/1000;
```

Генерация случайного число вещественного типа в заданном диапазоне выполняется по формуле:

$$x = (b - a) \cdot rnd + a$$

Где  $a$  и  $b$  начальное и конечное значение диапазона,  $rnd$  – значение, полученное от генератора случайных чисел ЭВМ.

Число подчиняются равномерному закону распределения.

### Обработка исключительных ситуаций

Исключительная ситуация – фатальная ошибка в программе, возникшая в процессе ее выполнения. Обычно такая ошибка возникает из – за некорректных данных, полученных при вводе, либо в процессе вычислений.

Перехват ошибки выполняется с помощью блока `try ... catch`:

```
try{
    //Опасные операторы
}
catch (Класс_ошибки_1 e) {
    //Операторы
```

```

}
catch (Класс_ошибки_2 e){
    //Операторы
}

```

Таблица 9. Некоторые классы ошибок

Класс ошибки	Описание
Exception	Абстрактная ошибка
ArithmeticException	Арифметическая ошибка
ArrayIndexOutOfBoundsException	Неверное обращение к массиву

Обработка абстрактной ошибки должна выполняться первой. В блок try допускается добавлять блок finally для выполнения действий после обработки ошибки. Блок выполняется всегда, даже если ошибки не было.

Пример:

```

try{
    c=a/b;
}
catch (ArithmeticException e){
//Некорректное выполнение деления
    c=0;
}

```

### Цвет и его кодирование

Для работы с цветом можно использовать свойства класса Color, либо конструктор данного класса, который позволяет кодировать цвет в палитре RGB.

Свойства класса – цветовые константы: black(черный), blue(синий), white(белый), green(зеленый), cyan(светлосиний), magenta(фиолетовый), gray(серый), lightGray(яркосерый), darkGray(темносерый), orange(оранжевый), pink(розовый),

red(красный), yellow(желтый). Обращение к цвету как к свойству выполняется в виде: `Color.green`.

Конструктор класса `Color` имеет вид:

```
public Color(int r, int g, int b).
```

В конструкторе используются формальные параметры `r` – кодирует интенсивность красной составляющей цвета, `g` – кодирует интенсивность зеленой составляющей, `b` – кодирует интенсивность синей составляющей. Диапазон изменения кода каждой интенсивности лежит в пределах от 0 до 255.

Пример создания красного цвета:

```
Color cRed=new Color(255,0,0);
```

## Элементы управления и фигуры презентации

### Текстовое поле

Для обслуживания текстового поля используются методы:

```
public java.lang.String getText().
```

Метод позволяет получить строку текста. Что бы разместить в поле строку текста нужно использовать метод:

```
public void setText(java.lang.String text).
```

Необходимо учитывать, что из текстового поля можно получить только строку символов. Если нужно получить из поля числовое значение, то его нужно преобразовать из строки в число.

Преобразования может быть выполнено с помощью классов типов Java: `Integer`, `Long`, `Byte`, `Short`, `Double`, `Float`.

Оператор преобразования в общем виде имеет следующую структуру:

```
v= new КонструкторКлассаТипа(строка_с_числом).  
    метод_преобразования();
```

Где: `v` – переменная нужного типа.

Методы преобразования: `longValue()`, `intValue()`, `floatValue()`, `doubleValue()`.

Пример:

```
double n;
String ns=myTextF.getText();
n=new Double(ns.trim()).doubleValue();
```

Где: `myTextF`, текстовое поле для ввода числа.

Необходимо иметь ввиду, если из поля поступит строка с символами, которые нельзя преобразовать в число, возникает исключительная ситуация. Поэтому преобразование рекомендуется выполнять в блоке `try catch`.

Обратное преобразование в Java выполняется после конкатенации числа с пустой строкой.

### Элемент слайдер

Позволяет получить число из определенного диапазона. Обслуживание элемента выполняется методами:

```
public double getMin().
```

Возвращает минимальное число диапазона.

```
public double getMax().
```

Возвращает максимальное число диапазона.

Следующие два метода используются для получения числа от слайдера и его записи в слайдер:

```
public double getValue()
```

```
public void setValue(double val)
```

Получить значение слайдера можно также с помощью свойства `value` при написании событийного кода для этого элемента.

### Командная кнопка

Обслуживание командных кнопок выполняется методами:

```
public void action()
```

Позволяет выполнить действия, связанные с кнопкой.

Чтобы разместить на кнопке текст, либо его прочитать служат методы `setText`, `getText`.

Управление доступом к кнопкам выполняется двумя методами:



```
public void setEnabled(boolean yes)
public boolean isEnabled()
```

Первый метод позволяет установить доступ к кнопке, значение формального параметра `true`, значение `false` закрывает доступ. Второй метод позволяет определить наличие доступа к кнопке.

### **Элемент «Текст»**

Этот элемент презентации используется для размещения статического текста. Он обслуживается методами `setText`, `getText`.

### **Элемент «Прямоугольник»**

Такой элемент широко используется для формирования областей вывода графической информации на презентации. Элемент обладает рядом методов, который позволяют выполнить его конфигурацию программным кодом.

Задание координат точки прорисовки фигуры и ее чтение по оси `X` и `Y` выполняются методами:

```
public void setX(double x)
public double getX()
public void setY(double y)
public double getY()
```

Задание высоты фигуры и чтение ее значения выполняются методами:

```
public void setHeight(double height)
public double getHeight()
```

Для работы с шириной фигурой служат методы:

```
public void setWidth(double width)
public double getWidth()
```

Для создания моделей такого используется механизм событий (см. Таблицу 10).

Таблица 10. Классификация событий

№	Тип события	Вид/Режим
1	По таймауту	а)Срабатывает один раз
		б)Циклический
		с)«Ручной»
2	С заданной интенсивностью	Нет
3	При выполнении условия	Нет

В случае использования режима а нужно задать период срабатывания в единицах модельного времени.

При использовании первого события с режимом б нужно указать

- Время первого срабатывания.
- Период срабатывания (число единиц модельного времени).

При выборе режима с событие должно управляться вызовом специального метода `restart(double t)`, где `t` – период срабатывания события.

Событие, происходящее с заданной интенсивностью, используется для моделирования потока независимых событий (пуассоновский поток). Такое событие выполняется периодически с интервалами времени, подчиняющимися экспоненциальному закону распределения с параметром, равным заданной интенсивности. Например, если интенсивность равна 5, то событие будет происходить в среднем 5 раз в единицу модельного времени.

Третий тип события выполняется один раз при выполнении определенного условия, чтобы продолжить проверку выполнения условия и следовательно повторить выполнение события нужно вызвать его метод `restart()`.

В общем случае для управления событиями определен ряд методов класса `Event`.

`void reset()` - Отменяет запланированное событие (если в текущий момент это событие запланировано на какой-то момент в

будущем). Если событие работает в циклическом режиме, то цикл не возобновится до тех пор, пока не будет вызван метод `restart()` или `restart(double t)`.

`void restart()` - Перезапускает событие (отменяет запланированное событие (если в текущий момент это событие запланировано на какой-то момент в будущем) и планирует его на другой момент времени согласно текущему значению Таймаута).

`void restart(double t)` - Перезапускает событие (отменяет запланированное событие (если в текущий момент это событие запланировано на какой-то момент в будущем) и планирует его через заданный таймаут `t`). Если событие циклическое, то в дальнейшем оно продолжит планироваться согласно изначально заданному таймауту.

Параметр: `t` - время (от текущего момента), на которое будет запланировано событие.

`void suspend()` - Приостанавливает событие. Отменяет запланированное событие, если в текущий момент это событие запланировано на какой-то момент в будущем, и запоминает время, оставшееся до его происхождения, для того, чтобы впоследствии можно было возобновить его выполнение путем вызова метода `resume()`.

Если же на момент вызова метода `suspend()` это событие не запланировано, то при последующем вызове метода `resume()` ничего не произойдет.

`void resume()` - Возобновляет выполнение ранее приостановленного события (в качестве таймаута такого события будет установлено время, оставшееся до его происхождения на момент приостановки этого события).

`double getRest()` - Возвращает время, оставшееся до запланированного происхождения события или

`Double.POSITIVE_INFINITY`, если событие в данный момент времени не запланировано.

### **Список литературы**

1. Буч Г., Джекобсон, Рамбо Д. Язык UML. Руководство пользователя: Пер. с англ. — М.: ДМК Пресс, 2001.
2. Карпов Ю. Имитационное моделирование систем. Введение в моделирование с AnyLogic 5. — Спб.: БХВ Питербург, 2005.
3. Патрик Ноутон, Герберт Шилдт. Java 2. Наиболее полное руководство: Пер. с англ. — Спб.:ВНУ Питербург, 2007.