ANYLOGIC™

Учебное пособие по Enterprise Library

www.xjtek.com

XJ Technologies Company Ltd
AnyLogic@xjtek.com
http://www.xjtek.com/products/anylogic

Содержание

Библиотека дискретно-событийного моделирования Enterprise Library

С помощью библиотеки Enterprise Library пакета AnyLogic Вы можете быстро создавать сложные дискретно-событийные модели, такие как:

- модели производственных процессов;
- модели систем обслуживания (банки, аэропорты, и т.д.);
- модели бизнес-процессов с оценкой затрат операций;
- модели логистики и цепочек доставки.

Библиотека объектов Enterprise Library позволяет создавать гибкие модели с наглядной визуализацией моделируемого процесса и возможностью сбора необходимой статистики.

Это учебное пособие научит Вас создавать простые модели с помощью Enterprise Library. Эти модели наглядно продемонстрируют возможность применения пакета AnyLogic для моделирования производственных процессов, систем массового обслуживания и бизнес-процессов (с оценкой затрат операций).

При создании более сложных моделей могут оказаться полезными примеры моделей AnyLogic. Вы можете найти пример с похожей постановкой задачи и применить предложенный в примере способ реализации в Вашей модели. Примеры моделей Вы можете открыть со *Стартовой страницы*, автоматически появляющейся при закрытии редактируемой модели.

Если при создании модели Вы будете сталкиваться с какими-то трудностями, то Вы можете сравнить Вашу модель с одной из контрольных моделей. Контрольные модели соответствуют ключевым моментам создания моделей учебного пособия; Вы можете открыть контрольные моделей со *Стартовой страницы*.

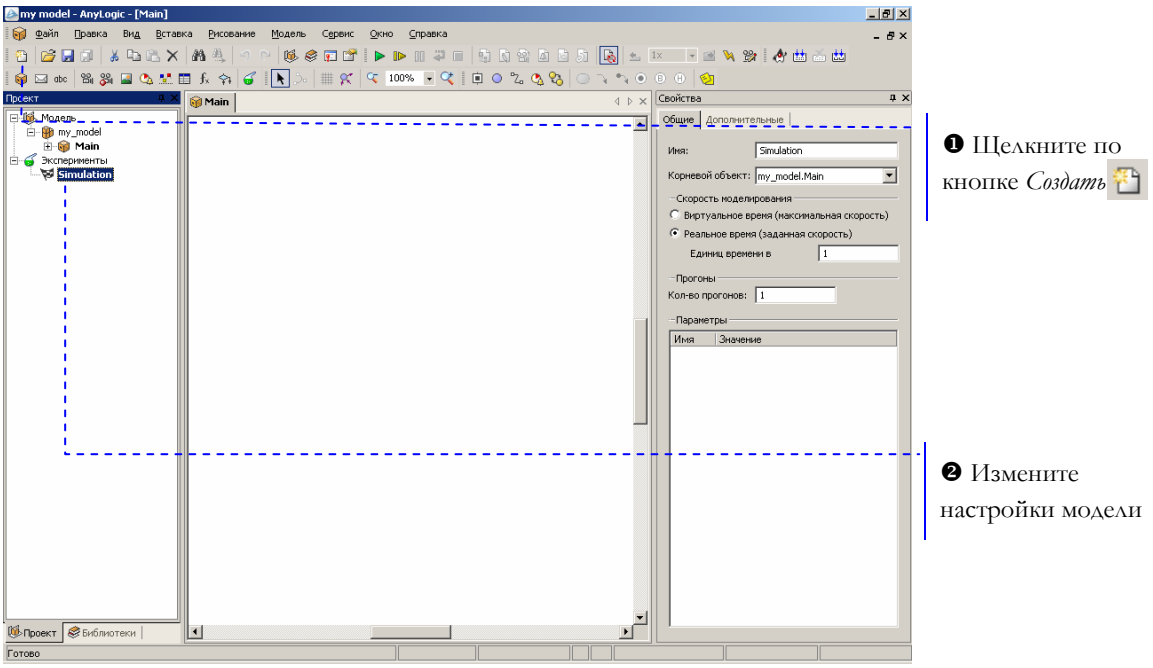
1. С чего начать работу в AnyLogic

Этот раздел содержит общую информацию о создании моделей в Enterprise Library. Вы научитесь создавать новые модели и изменять их настройки, а также добавлять и соединять объекты библиотеки.

1.1 Как создать модель

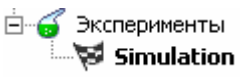
В этом разделе объясняется, как создать новую модель и как изменить ее настройки.

► Как создать новую модель

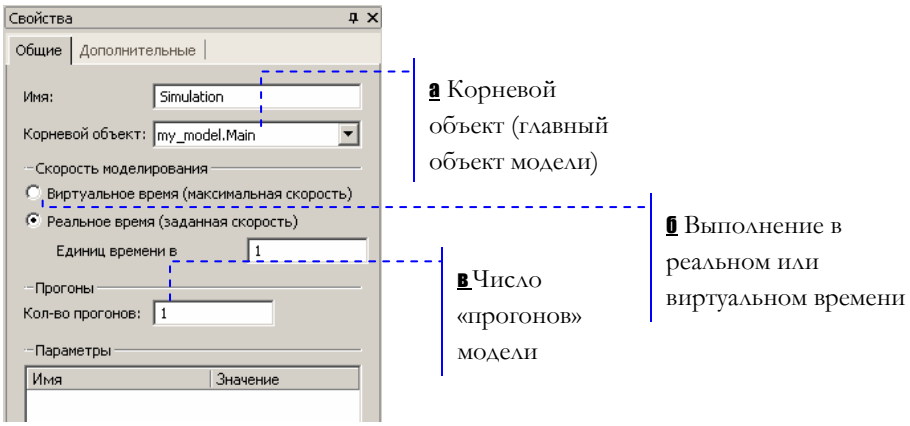


❶ Когда Вы щелкнете мышью по кнопке *Создать*, появится диалоговое окно, в котором Вы должны будете дать имя файлу Вашей модели и выбрать директорию, где он будет храниться.

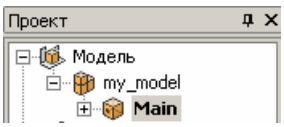
❷ Вы можете создать различные наборы конфигурационных настроек модели, называемые *экспериментами*. В дереве модели эксперименты отображаются под элементом *Эксперименты*.



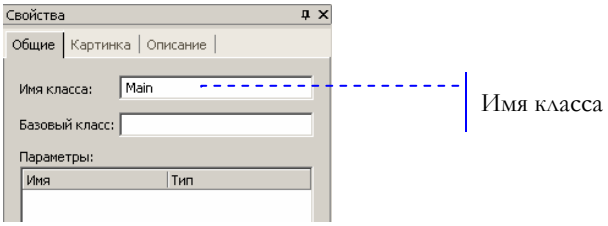
Один эксперимент, названный *Simulation*, создается автоматически. Выберите его щелчком мыши по элементу дерева и измените настройки модели в окне *Свойства*.



а Выберите класс, который будет запущен при запуске модели. По умолчанию в качестве корневого объекта выбран объект класса *Main*, автоматически создаваемого в каждой модели.



Вы можете переименовывать классы модели. Для этого нужно выделить класс щелчком мыши по значку класса в дереве модели и затем изменить его имя в окне *Свойства*.



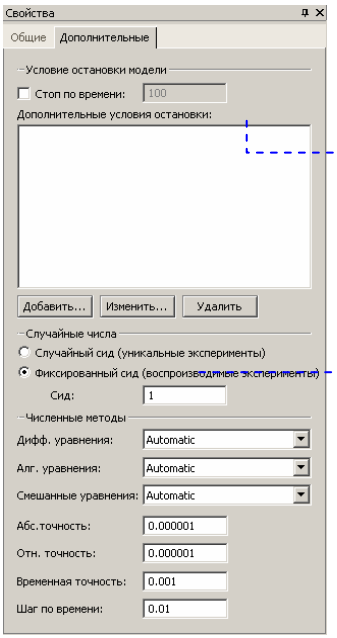
б В режиме реального времени задается связь модельного времени с физическим, то есть задается количество единиц модельного времени, выполняемых в одну секунду. Режим реального времени лучше всего подходит для показа анимации. В режиме виртуального времени модель выполняется без привязки к физическому времени – она просто выполняется так быстро, насколько это возможно. Этот режим лучше всего подходит, когда требуется моделировать работу системы в

течение достаточно длительного периода времени.

➲ В этом учебном пособии мы в основном будем пользоваться *Режимом реального времени*, чтобы иметь возможность отображать анимацию модели с фиксированной скоростью.

в Вы можете сконфигурировать повторный запуск модели с разными значениями параметров. Так Вы можете проследить поведение системы при различных условиях, собирая сравнительную статистику по нескольким запускам.

Дополнительные свойства эксперимента позволяют управлять выполнением модели.



а Условие останова модели

б Свойства генератора случайных чисел

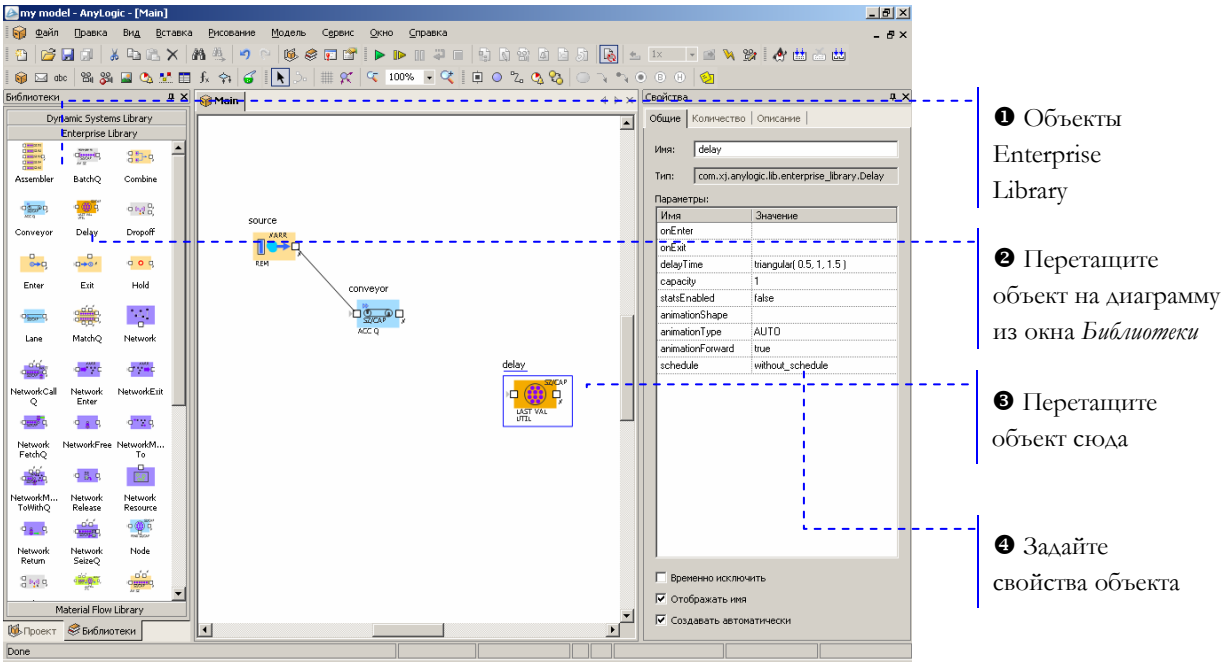
а Вы можете запустить модель так, чтобы она работала бесконечно, но можете и остановить ее в заданный момент времени. Вы можете остановить модель по достижению переменной заданного значения или по выполнению какого-нибудь определенного условия.

б Если в модели используются случайные числа, то Вы можете выбрать, хотите ли Вы генерировать уникальные случайные числа (это нужно при сборе сравнительной статистики по нескольким запускам) или генерировать одинаковые числа и добиваться многократного запуска одной и той же модели (это нужно для изучения поведения модели при нескольких запусках).

1.2 Как добавлять объекты

Для построения модели в Enterprise Library нужно перетащить объекты с вкладки *Enterprise Library* окна *Библиотеки* на диаграмму, а затем задать требуемые свойства объектов и соединить их друг с другом.

► Как добавлять объекты



❶ Все объекты библиотеки отображаются на вкладке *Enterprise Library* окна *Библиотеки*.

❷ Чтобы добавить объект на блок-схему модели, щелкните по объекту в окне *Библиотеки* и перетащите его мышью на структурную диаграмму.

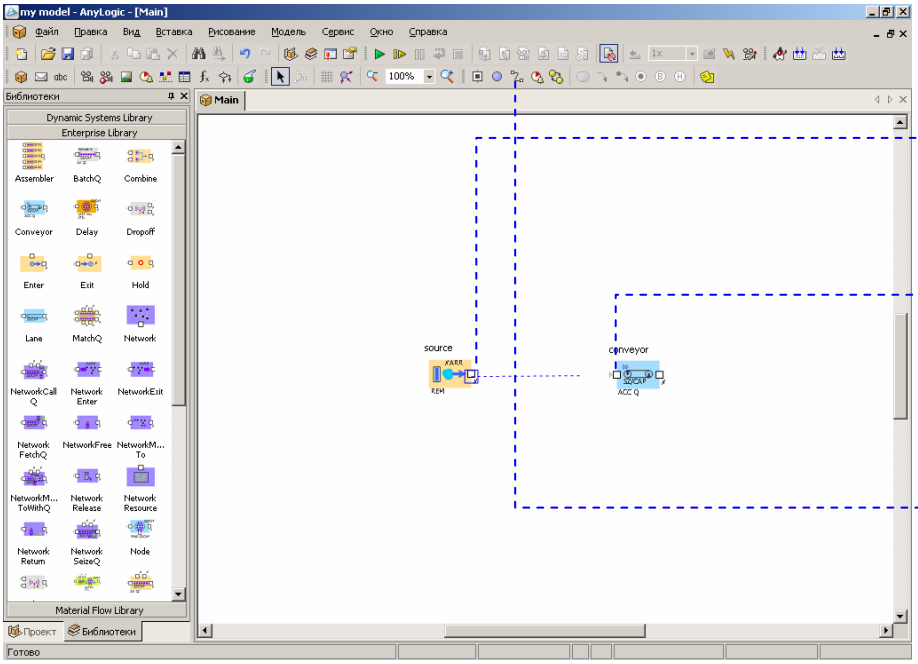
❸ Когда Вы поместите элемент на структурную диаграмму, элемент будет выбран, и его свойства будут отображены в окне *Свойства*.

❹ В окне *Свойства* Вы можете изменять свойства элемента в соответствии с требованиями Вашей модели. Позднее для изменения свойств элемента нужно будет вначале щелчком мыши выделить его на диаграмме или в дереве проекта.

1.3 Как соединять объекты

Объекты должны взаимодействовать между собой, поэтому Вы должны будете соединять их друг с другом. Вы можете соединять объекты с помощью мыши, перетаскиванием порта одного объекта на порт другого, или с помощью специального средства *Соединитель*.

► **Для соединения объектов**



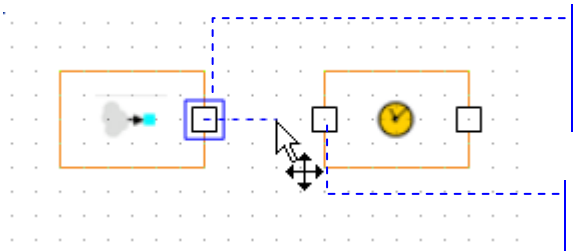
❶ Щелкните мышью по первому порту

❷ Перетащите его на второй порт. Появится соединитель.

❸ Можете воспользоваться инструментом *Соединитель*

❶ Для соединения двух соседних объектов пользуйтесь методом перетаскивания:

❷



a Выберите первый порт щелчком мыши по нему

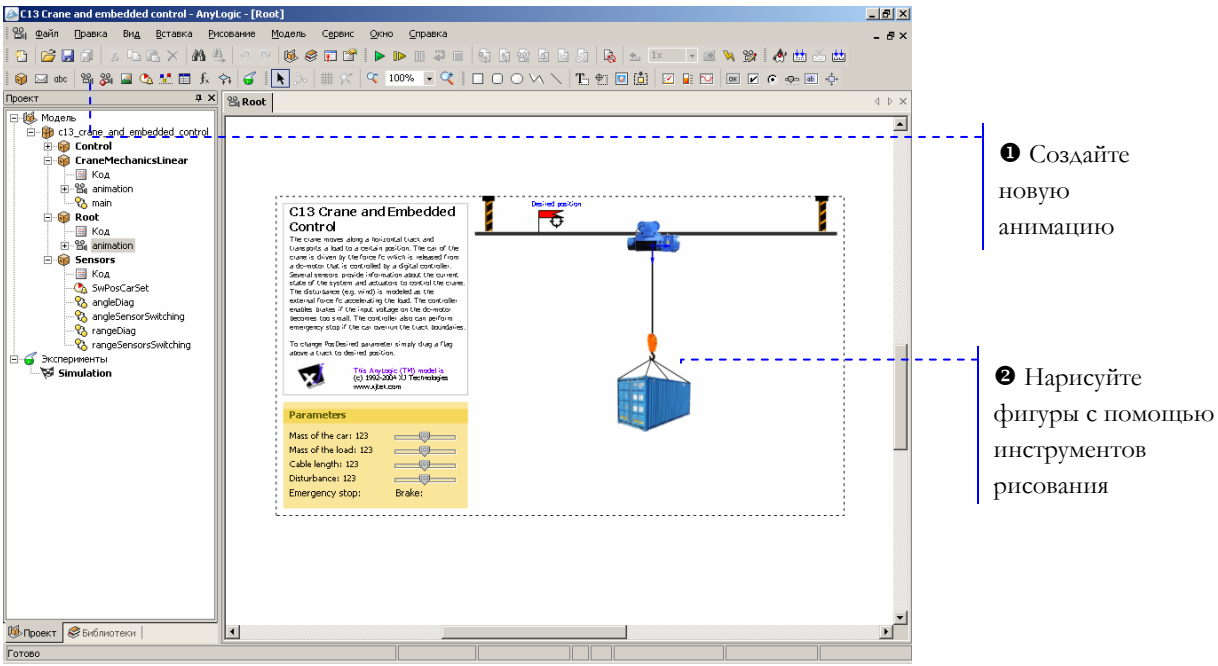
б Создайте соединитель, перетащив один порт на другой

❸ Чтобы нарисовать соединители сложной формы, щелкните мышью по кнопке панели инструментов *Соединитель*, щелкните по первому порту, а затем щелкните по второму порту.

1.4 Как создать анимацию

Вы можете создать анимацию модели, чтобы визуально отображать ее поведение. Анимация в AnyLogic рисуется на анимационной диаграмме.

► **Для создания анимации**



❶ Чтобы создать новую анимацию, щелкните мышью по кнопке панели инструментов *Новая анимация* .

❷ Вы можете рисовать фигуры (линии, круги, прямоугольники и т.д.), добавлять на анимацию картинки, элементы управления (кнопки, переключатели, флажки и т.д.), индикаторы. Все анимационные фигуры могут менять свой внешний вид во время работы модели.

2. Модель банковского отделения

В этом разделе учебного пособия мы создадим модель простой системы обслуживания, а именно модель банковского отделения. В банковском отделении находятся банкомат и стойки банковских кассиров, что позволяет быстро и эффективно обслуживать посетителей банка. Операции с наличностью клиенты банка производят с помощью банкомата, а более сложные операции, такие как оплата счетов – с помощью кассиров.

Мы произведем оценку затрат операций и увидим, сколько денег тратится на обслуживание одного клиента, и какую часть этой суммы составляют накладные расходы на оплату работы персонала банка, а какую – на обслуживание посетителей.

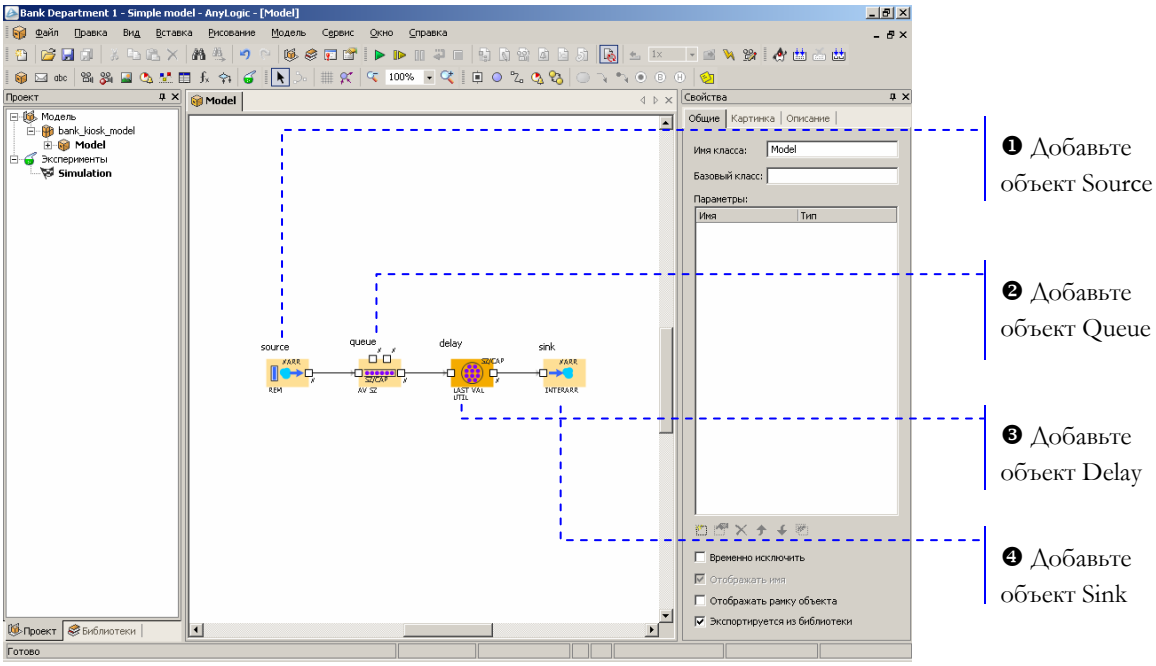
2.1 Создание нового проекта

Создайте новую модель, как описано в разделе 1.1, “Как создать модель”. Переименуйте класс `Main` в `Model`. В свойствах эксперимента *Simulation* задайте выполнение модели в режиме реального времени с выполнением одной единицы модельного времени в одну секунду. В этой модели под единицей модельного времени мы будем понимать одну минуту работы банковского отделения.

2.2 Создание блок-схемы

Сейчас мы создадим блок-схему модели, которая пока будет состоять только из банкомата. Для этого создайте и соедините объекты так, как показано на рисунке ниже.

► Создайте блок-схему



❶ Объект Source генерирует *сущности* (entities) определенного типа через заданный временной интервал. В Enterprise Library с помощью сущностей моделируются активные объекты модели – это могут быть клиенты в системе обслуживания, детали в модели производства, документы в модели документооборота и т.д. В нашем примере сущностями будут посетители банка, а объект Source будет моделировать их приход в банковское отделение.

За детальным описанием объектов Enterprise Library, пожалуйста, обращайтесь к *Справочному руководству по Enterprise Library*. Там Вы найдете описание всех функций объектов библиотеки и их параметров. Для вызова *Справочного руководства по Enterprise Library*, выберите соответствующий пункт меню *Справка*.

❷ Объект Queue моделирует очередь клиентов, ожидающих обслуживания.

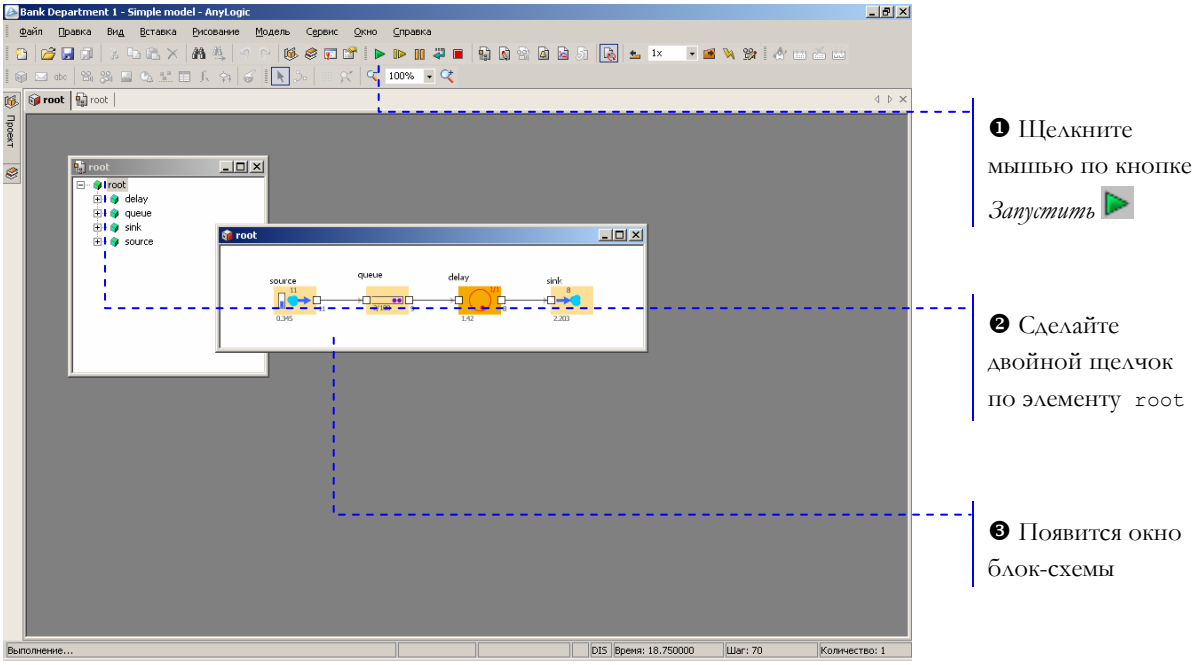
❸ Объект Delay моделирует задержку. В нашем примере он будет моделировать банкомат, тратящий определенное время на обслуживание клиента.

❹ Объект Sink обозначает конец блок-схемы.

2.3 Запуск модели и изучение ее поведения

Теперь мы можем запустить созданную модель. Для каждой модели, созданной в Enterprise Library, автоматически создается блок-схема с наглядной визуализацией процесса, с помощью которой Вы можете изучить текущее состояние модели, например, длину очереди, количество обслуженных человек и так далее.

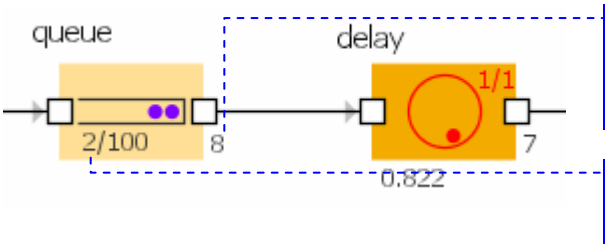
► **Запустите модель и откройте окно визуализированной блок-схемы**



❶ Щелкните мышью по кнопке *Запустить*  . AnyLogic переключится в режим «прогона» модели.

❷ Сделайте двойной щелчок мышью по самому верхнему элементу дерева окна *root*. Если окно скрыто, откройте его щелчком мыши по кнопке панели инструментов *Корневой объект модели*  .

❸ С помощью визуализированной блок-схемы Вы можете проследить, сколько человек находится в очереди, сколько человек в данный момент обслуживается и т.д.

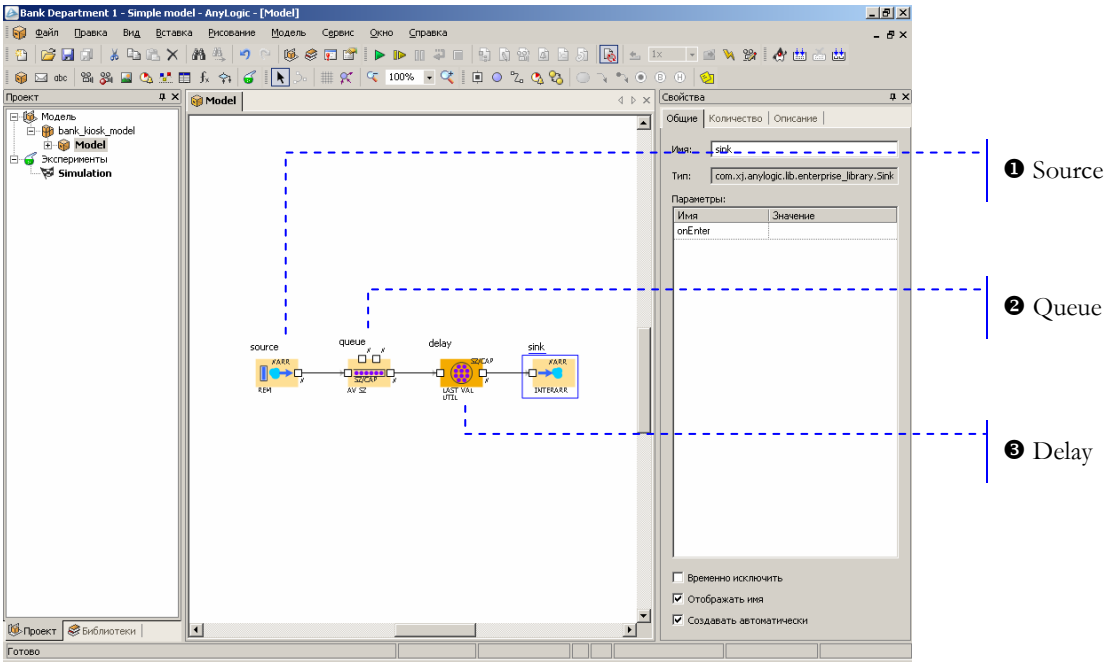


Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Bank Department 1 - Simple model.alp.

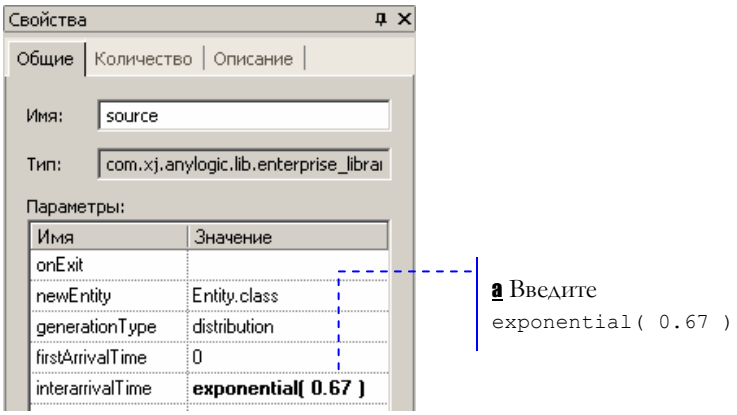
2.4 Задание данных

Теперь, изменяя свойства созданных объектов, мы зададим данные нашей модели.

► **Измените свойства объектов блок-схемы**



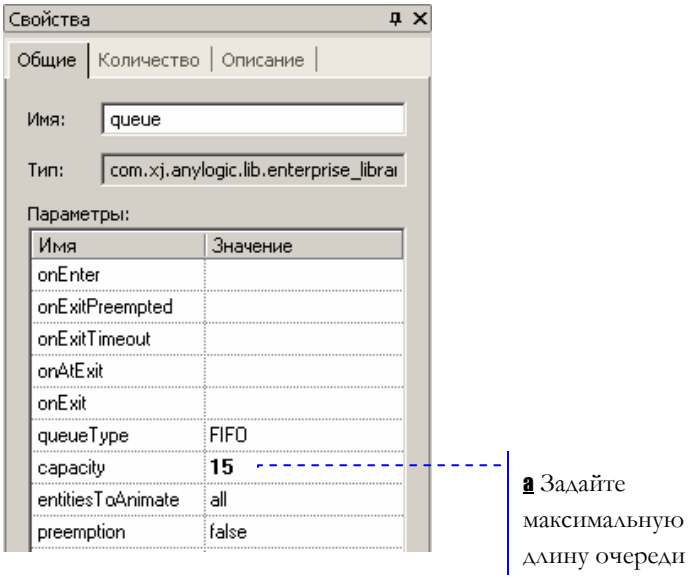
❶ В свойстве объекта `interarrivalTime` укажите, как часто в отделение приходят клиенты.



a Интервал между приходом клиентов экспоненциально распределен со средним значением равным 1.5 единицам модельного времени. Заметьте, что аргумент функции `exponential()` равен 0.67, потому что в качестве аргумента задается интенсивность прихода клиентов.

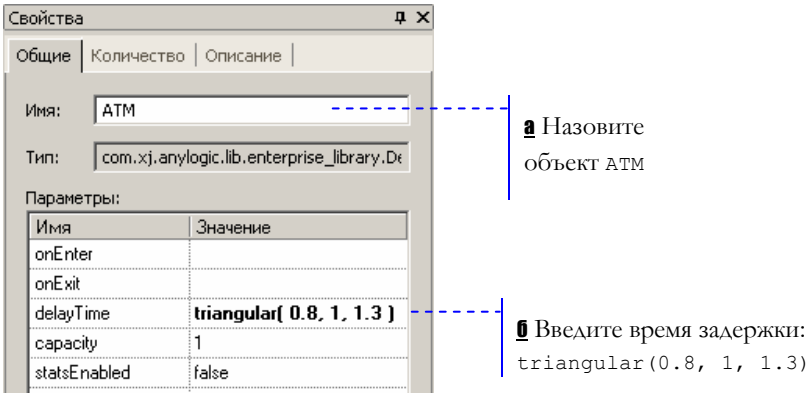
Функция `exponential()` является стандартной функцией генератора случайных чисел AnyLogic. AnyLogic предоставляет функции и других случайных распределений, таких как нормальное, равномерное, треугольное, и т.д. За детальным описанием функций и их параметров обращайтесь к *Руководству пользователя* или *Справочнику классов* (смотрите методы класса `Func`). Для вызова *Руководства пользователя* или *Справочника классов* AnyLogic, выберите соответствующие пункты меню *Справка*.

❷ Задайте следующие свойства объекта:



a В очереди будут находиться не более 15 человек.

❸ Задайте свойства объекта Delay:



б Обслуживание одного клиента занимает примерно 1 минуту. Задайте время обслуживания, распределенное по треугольному закону со средним значением, равным 1, минимальным - равным 0.8 и максимальным - 1.3 минутам.

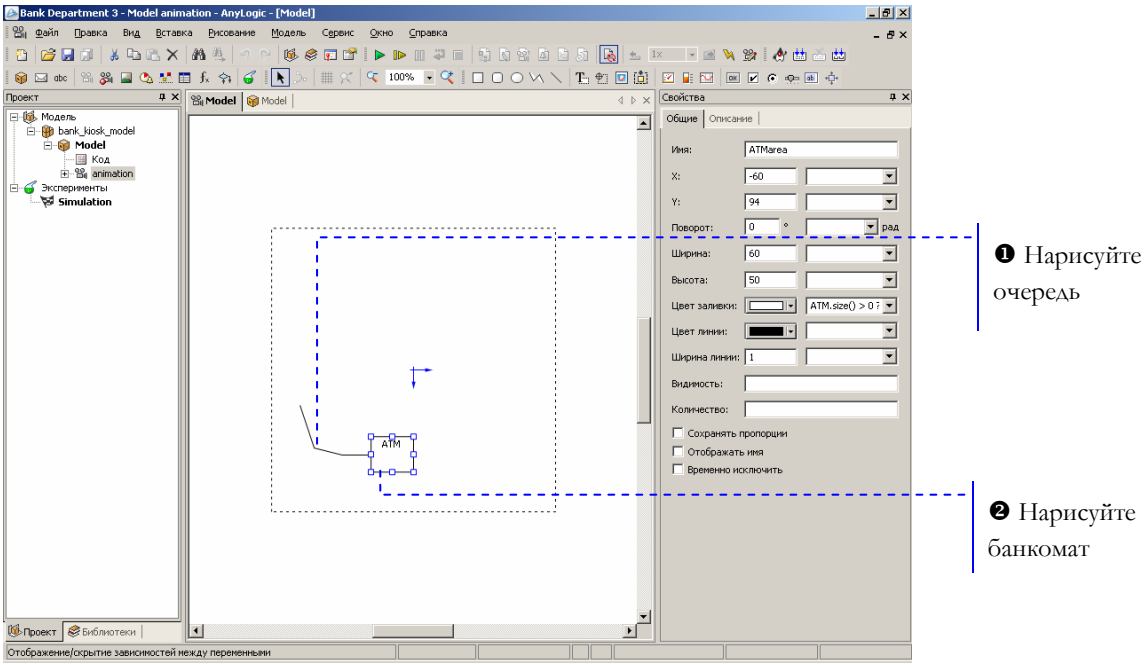
Теперь Вы можете запустить модель щелчком по кнопке *Запустить* и изучить поведение модели.

Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Bank Department 2 - Model data.alp.

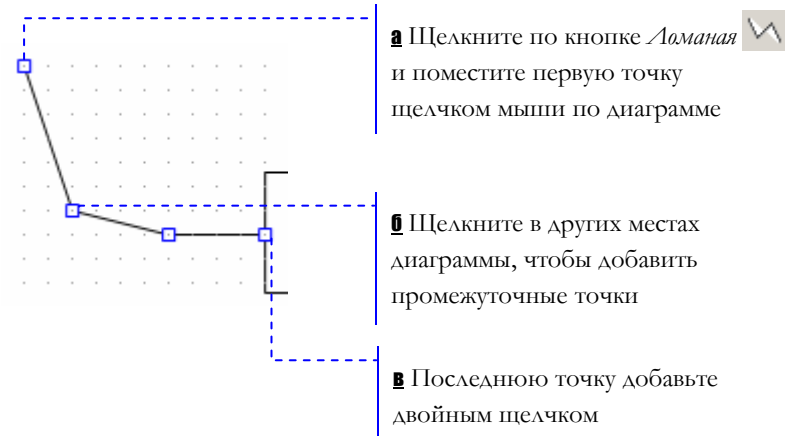
2.5 Создание анимации

Визуализация процесса с помощью блок-схемы очень удобна, поскольку визуализированная блок-схема создается для каждой модели автоматически. Но с помощью анимации AnyLogic Вы можете создать намного более наглядную визуализацию Вашего процесса. В этом примере мы хотим создать визуализированный план банковского отделения. Для этого нам будет нужно вначале создать анимационную диаграмму, а затем нарисовать на ней банкомат и очередь.

► **Нарисуйте банкомат и очередь**

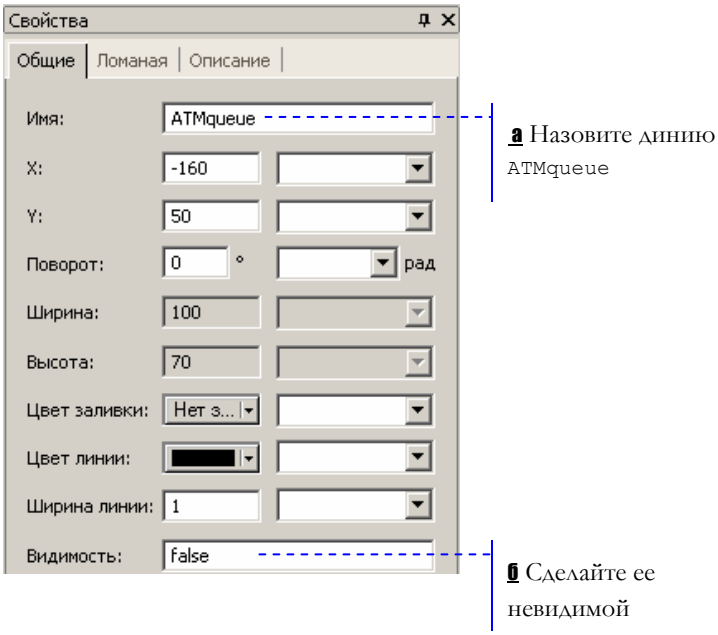


❶ Нарисуйте очередь с помощью инструмента рисования *Ломаная*.



➲ Очень важно, какую точку ломаной Вы создаете первой. По умолчанию, сущности будут двигаться от точки, которую Вы нарисуете первой, к точке, которую Вы нарисуете последней.[1]

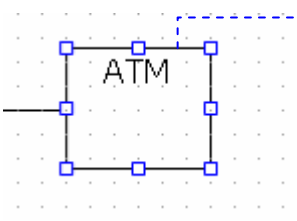
Задайте следующие свойства ломаной линии:



б Чтобы сделать линию невидимой на анимации, выделите линию щелчком мыши и введите `false` в свойстве линии *Видимость*.

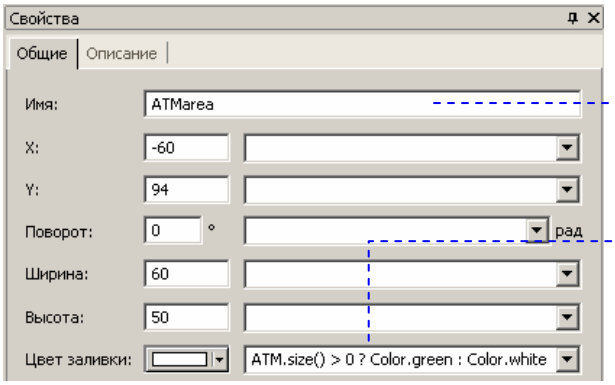
[1] Если Вам нужно, чтобы движение происходило в обратном направлении, Вы можете установить свойство объекта `animationForward` в `false`.

❷ Нарисуйте прямоугольник:



а Щелкните по кнопке *Прямоугольник*  , а затем по анимационной диаграмме

Задайте следующие свойства:



а Назовите прямоугольник `ATMarea`

б Задайте цвет заливки

б Введите Java выражение, задающее цвет прямоугольника во время работы модели:

```
ATM.size() > 0 ? Color.green : Color.white
```

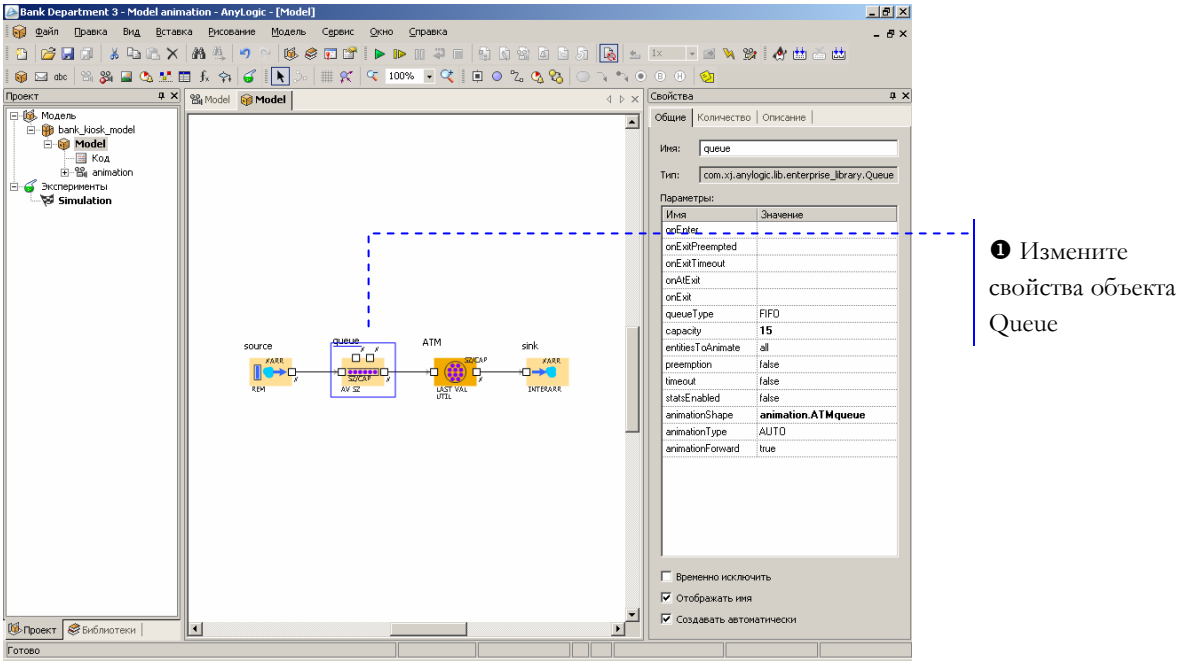
Здесь `ATM` – это имя созданного нами объекта Delay. Функция `size()` возвращает число человек, обслуживаемых в данный момент времени. Если банкомат занят, то цвет прямоугольника будет зеленым, в противном случае - белым.

`Color` – это класс Java, позволяющий использовать стандартные цвета (черный, синий, красный, голубой, желтый и т.д.), и создавать любые другие.[2]

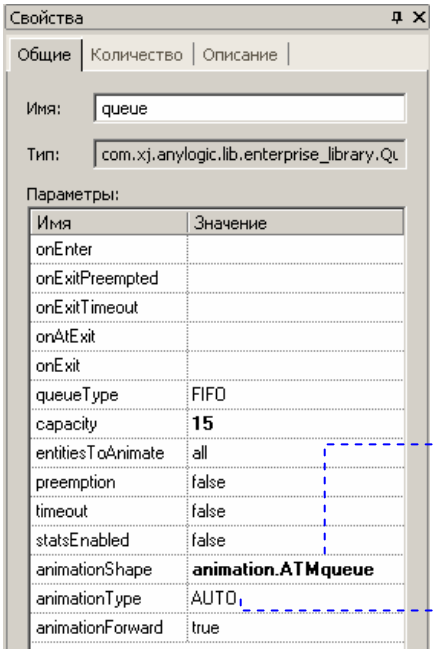
[2] Чтобы просмотреть список стандартных цветов и методов класса Color, позволяющих создавать другие цвета, зайдите по адресу http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Color.html.

Теперь мы должны задать созданные анимационные объекты в качестве анимационных фигур объектов блок-схемы нашей системы.

► Задайте анимационные свойства объектов блок-схемы



❶ Задайте следующие свойства объекта:

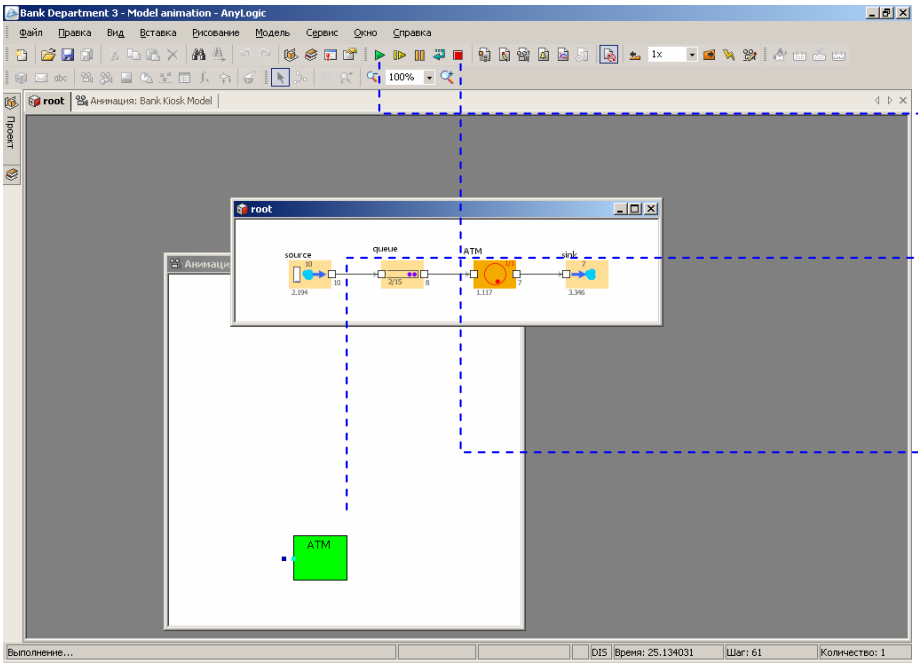


а Щелкните мышью по полю и выберите `animation.ATMqueue` из выпадающего списка.

ⓘ Объекты Enterprise Library поддерживают несколько анимационных стилей. Например, очередь может отображаться в виде линии, упорядоченного или неупорядоченного набора элементов. Подробное описание анимационных стилей дано в *Справочном руководстве по Enterprise Library*. Стиль AUTO определяет стиль автоматически в зависимости от заданной для объекта анимационной фигуры. В нашем случае очередь будет отображаться линией.

Теперь Вы можете запустить модель и изучить ее поведение.

► **Запустите модель**



❶ Щелкните по кнопке *Запустить*

❷ Модель начнет выполнение, и появится окно анимации

❸ Чтобы остановить выполнение, щелкните по кнопке *Остановить*

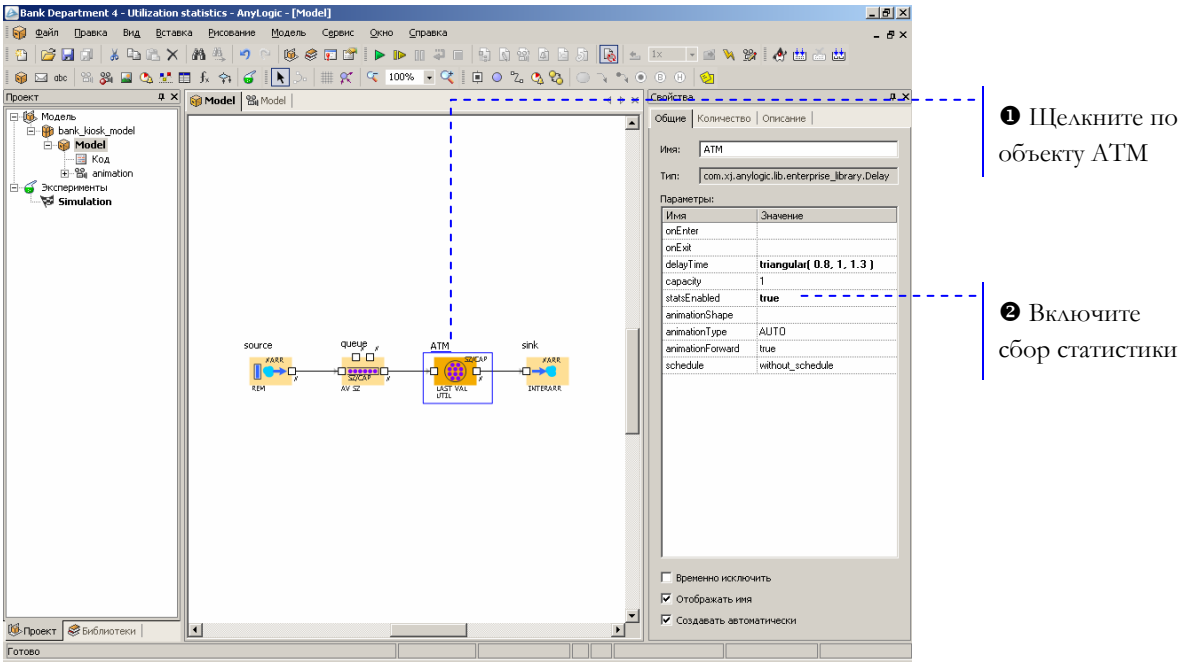
❷ Запустив модель, Вы увидите окно анимации. Цвет прямоугольника будет меняться в зависимости от того, обслуживается ли клиент в данный момент времени.

Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Bank Department 3 - Model animation.alp.

2.6 Сбор статистики

AnyLogic позволяет производить сбор сложной статистики. Для этого нужно лишь включить у объекта режим сбора статистики, поскольку по умолчанию он отключен для повышения скорости выполнения модели.

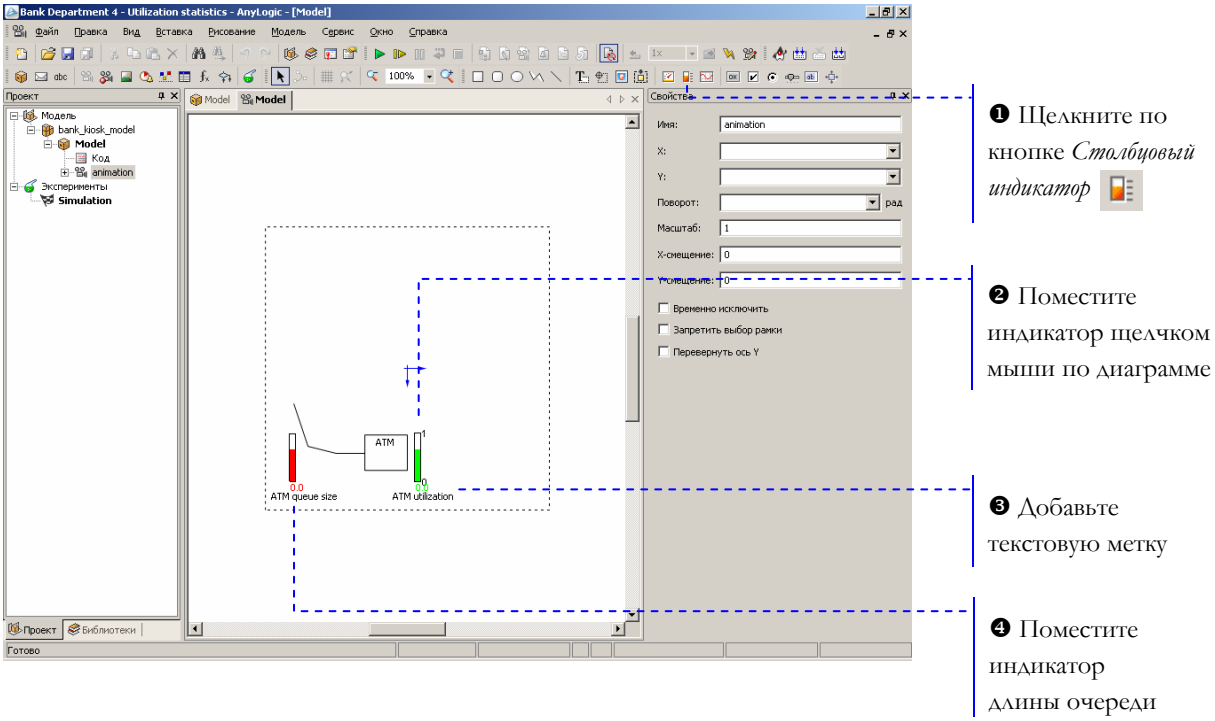
► **Включите сбор статистики**



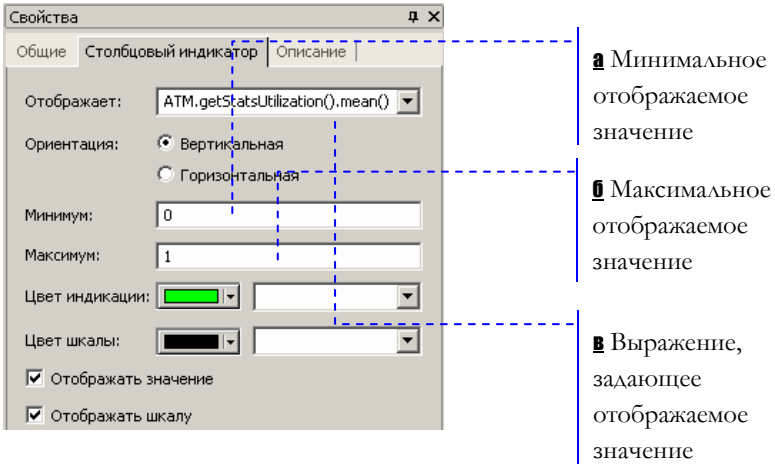
❷ Чтобы включить сбор статистики для объекта, выберите `true` в свойстве `statsEnabled`.

Вы можете просмотреть собранную статистику с помощью диаграмм и графиков, или выводя числовые значения на анимацию. Мы же покажем статистику занятости банкомата с помощью индикатора.

► Добавьте индикатор на анимацию



❷ Измените некоторые свойства индикатора:



в Задайте следующее выражение:

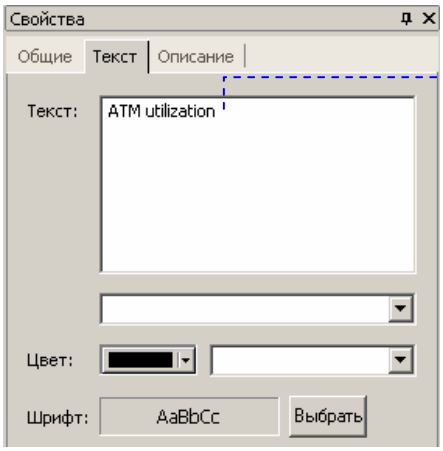
```
ATM.getStatsUtilization().mean()
```

`ATM` – это имя созданного нами объекта Delay. Функция `getStatsUtilization()` объекта Delay возвращает статистику занятости объекта, а функция `mean()` возвращает среднее из измеренных значений. Вы можете использовать и другие методы, например, `min()` и `max()`, для получения минимального и максимального измеренных значений

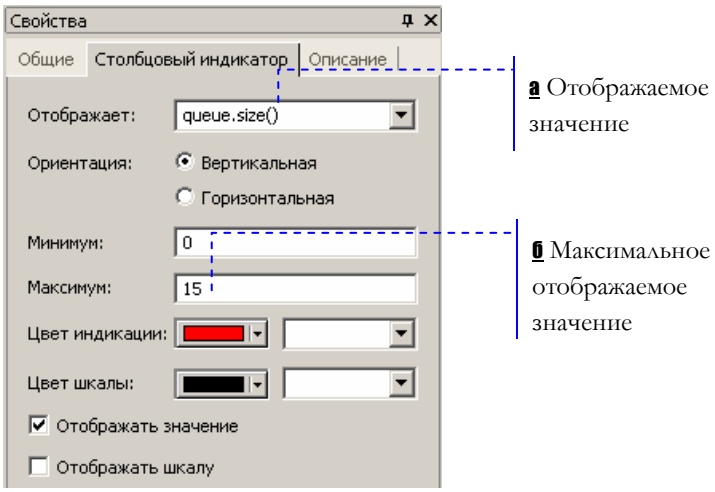
соответственно. Список методов смотрите в *Справочнике классов*, на странице `DataSet`.

Подробную информацию о методах объекта Delay, предназначенных для сбора статистики, Вы можете найти в *Справочном руководстве по Enterprise Library.*

❸ Добавьте текстовую метку, щелкнув мышью по кнопке панели инструментов *Текст*  , и затем щелкнув по диаграмме под индикатором.



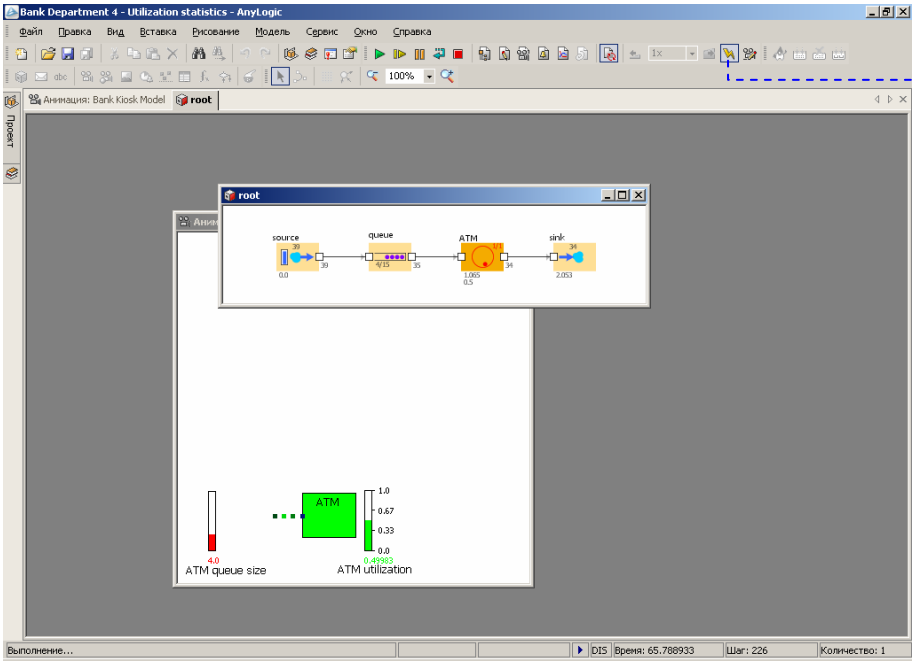
❹ Добавьте индикатор текущей длины очереди.



a Функция `size()` объекта Queue возвращает текущее количество людей в очереди.

б Поскольку максимальная длина очереди равна 15, то мы устанавливаем максимальное значение также равным 15.

► **Запустите модель**



❶ Для ускорения работы модели, переключитесь в режим виртуального времени

❶ В режиме виртуального времени модель будет выполняться с максимально возможной скоростью.

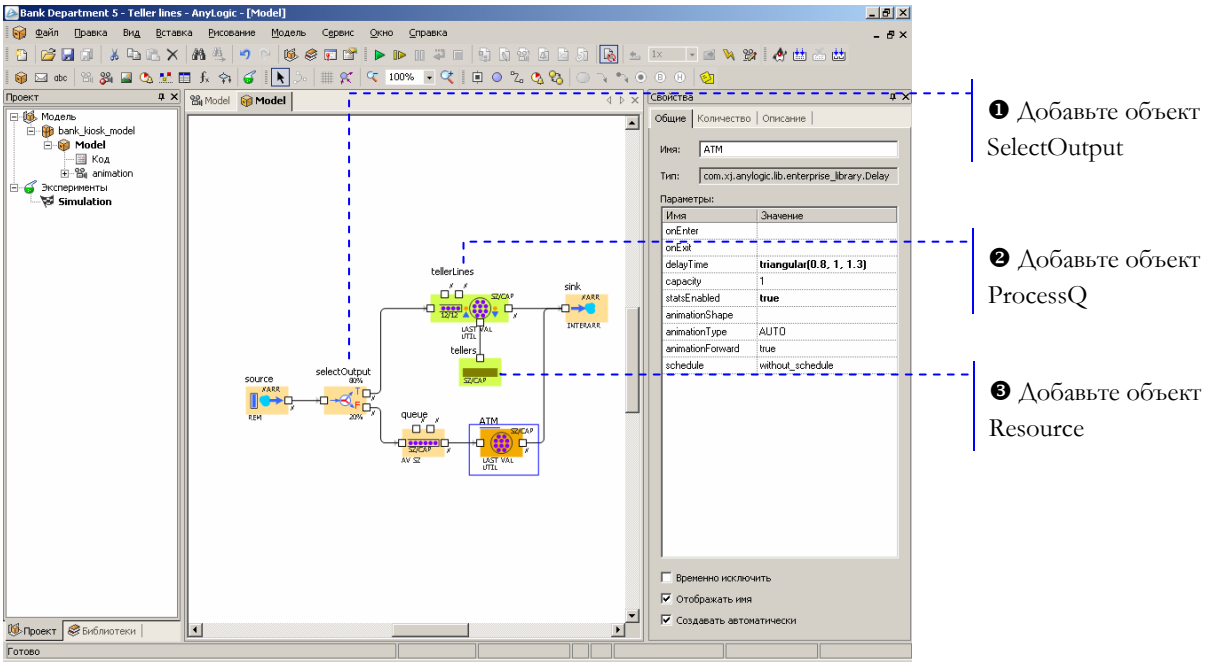
Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Bank Department 4 - Utilization statistics.alp.

Вы научились основам создания моделей в Enterprise Library. Теперь Вы готовы к созданию более сложной модели.

2.7 Моделирование банковских кассиров

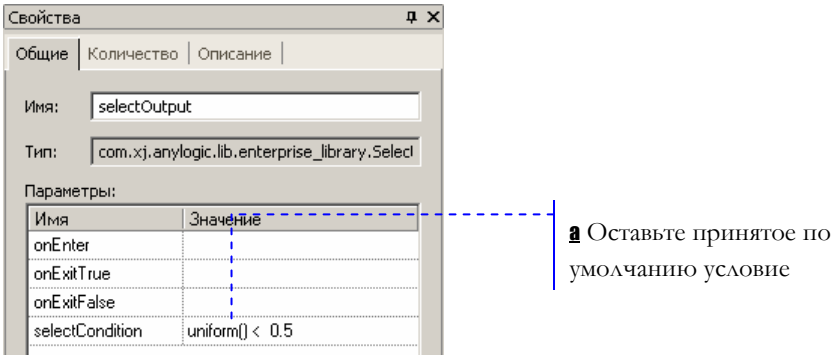
Теперь мы усложним нашу модель, добавив в нее служащих – банковских кассиров. Мы могли бы промоделировать кассиров, как и банкомат, с помощью объектов Delay. Но куда более удобным представляется моделирование кассиров с помощью *ресурсов*. Ресурс – это специальный объект Enterprise Library, который может потребоваться сущности для выполнения какой-то задачи. В нашем примере посетителям банковского отделения (сущностям) необходимо получить помощь у банковских служащих (ресурсов).

► Измените блок-схему



❶ Объект SelectOutput является блоком принятия решения. В зависимости от заданного Вами условия, сущность, поступившая в объект, будет поступать на один из двух выходов объекта.

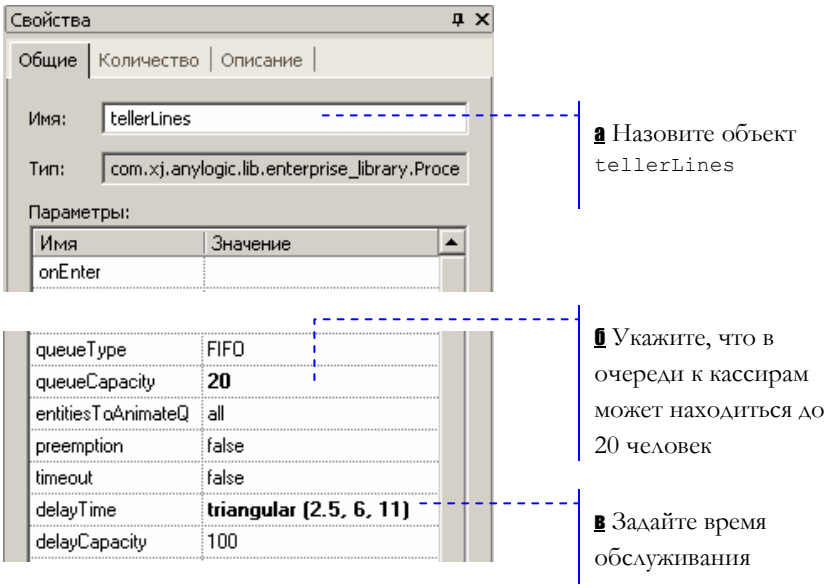
Задайте следующие свойства объекта:



a Оставьте условие `uniform()<0.5`. В этом случае к кассирам и банкомату будет приходить примерно равное количество клиентов.

❷ Объект ProcessQ моделирует занятие сущностью ресурса на определенное время. С помощью этого объекта мы промоделируем обслуживание клиента кассиром.

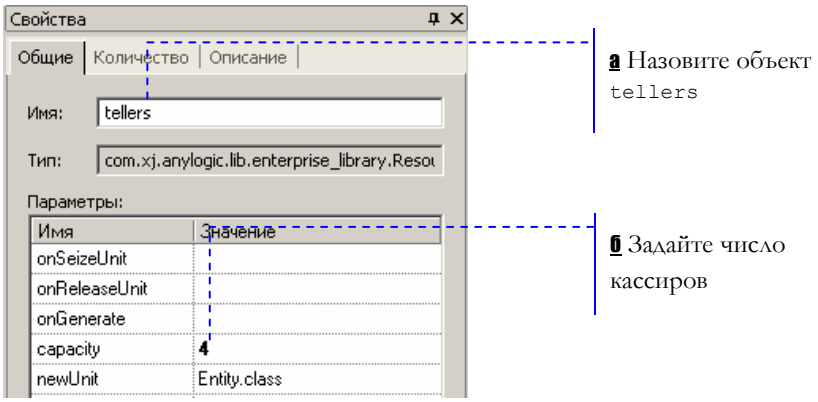
Задайте следующие свойства объекта:



в Мы полагаем, что время обслуживания имеет треугольное распределение с минимальным значением равным 2.5, средним - 6, и максимальным - 11 минутам.

❸ Объект Resource задает ресурсы определенного типа. Он должен быть подсоединен к объектам, моделирующим занятие и освобождение ресурсов (в нашем случае это объект ProcessQ).

Задайте следующие свойства объекта:



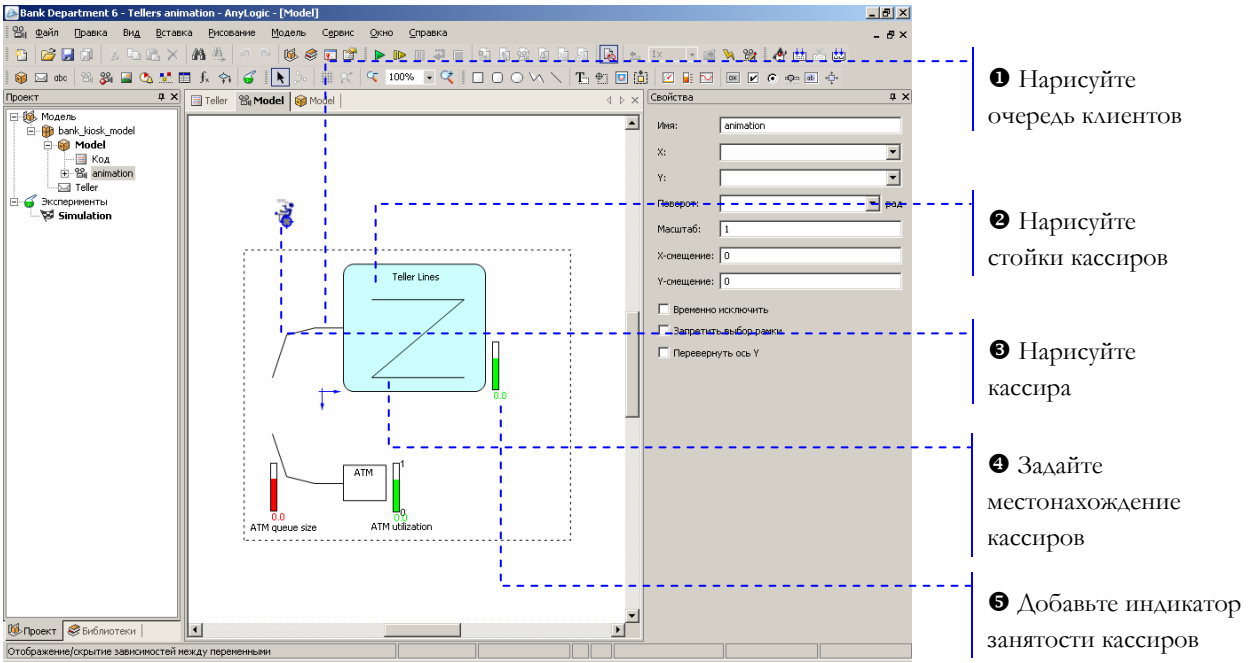
Запустите модель и изучите ее поведение.

Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Bank Department 5 - Teller lines.alp.

2.8 Отображение кассиров на анимации

Поскольку модель изменилась, мы должны изменить и ее анимацию.

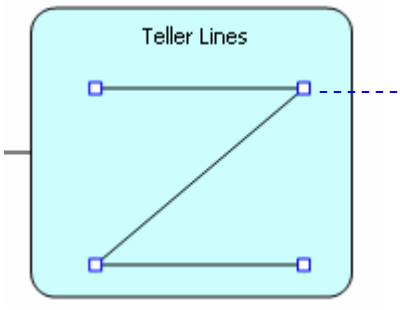
► **Измените анимацию модели**



❶ Пользуясь инструментом *Ломаная*  , нарисуйте очередь клиентов, ожидающих обслуживания.

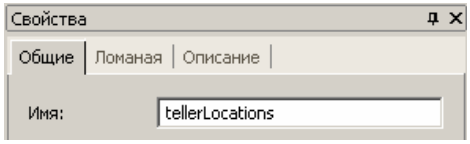
❷ Пользуясь инструментом *Скругленный прямоугольник*  , нарисуйте стойки кассиров. Выберите подходящий цвет заливки (прямоугольник на рисунке имеет цвет заливки с компонентами 204, 255, 255). Добавьте метку `Teller lines`.

❸ Нарисуйте *Ломаную*  , чтобы указать на анимации места, в которых будут находиться банковские служащие.



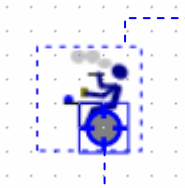
a Точки ломаной линии будут соответствовать местоположению служащих на анимации

Назовите ломаную линию `tellerLocations`.



❹ Мы нарисуем кассиров внутри прямоугольника. С помощью разных картинок мы покажем, обслуживает ли кассир в данный момент какого-нибудь клиента или нет. Чтобы создать картинку, щелкните по кнопке *Картинка* и затем щелкните по диаграмме.

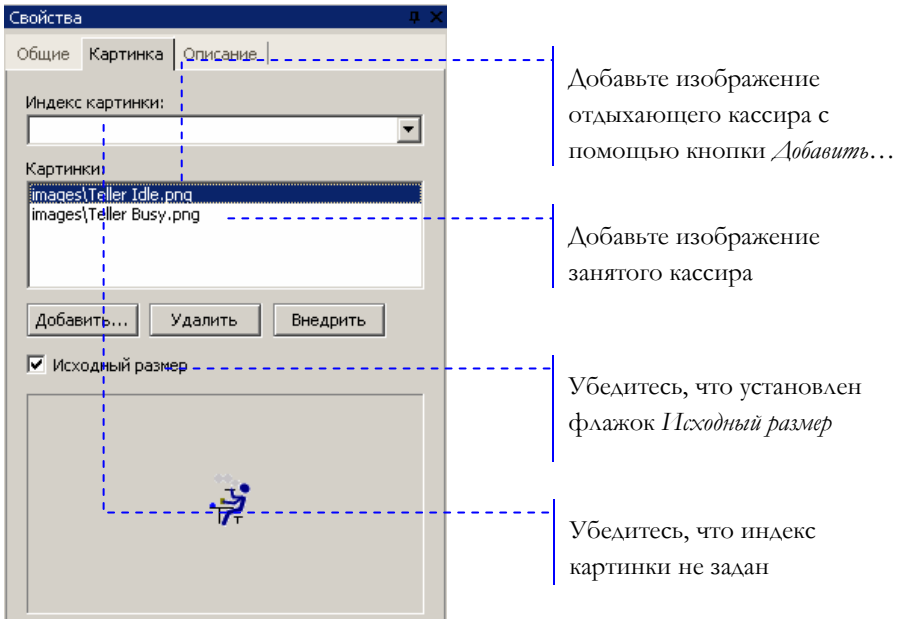
Нарисуйте кассира с помощью динамически создаваемой группы фигур:



a Создайте *Картинку* и добавьте изображения занятого и свободного кассиров

б Создайте *Группу фигур* и добавьте в нее созданную картинку

a Картинка должна содержать изображения работающего и отдыхающего кассира. Убедитесь, что картинка названа `image`.



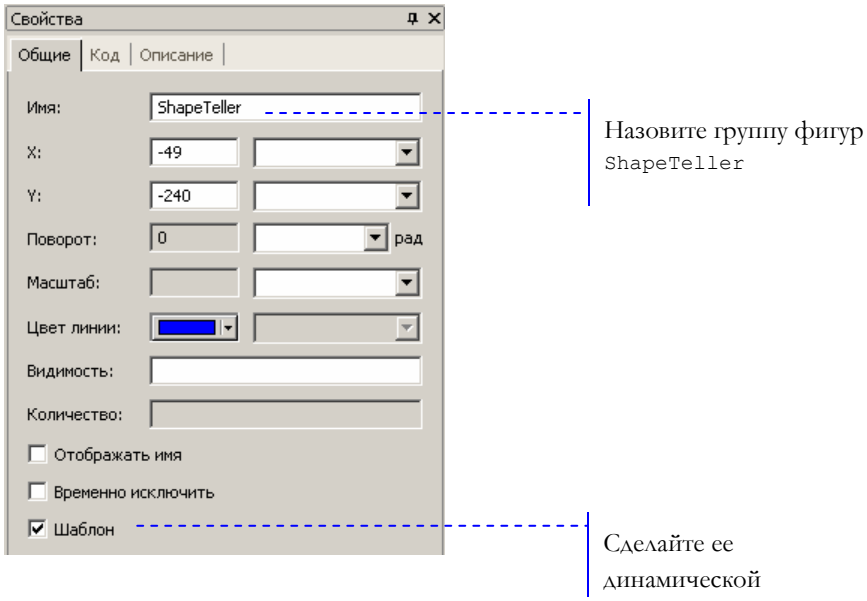
Картинка может содержать несколько изображений, показывая одно из них в зависимости от значения заданного выражения.

Вы можете использовать картинки Examples \ Enterprise Library Tutorial Models \ images \ Teller Idle.png и Teller Busy.png.

Если опция *Исходный размер* не будет выбрана, то масштаб картинки будет изменяться, чтобы размер картинки соответствовал области фигуры картинки на анимационной диаграмме.

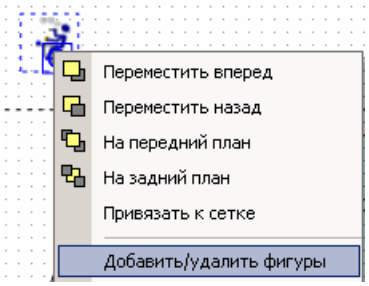
Выражение *Индекс картинки* определяет, какую картинку из списка отображать. Оставьте поле *Индекс картинки* пустым.

6 Щелкните мышью по кнопке *Группа фигур*  , а потом по диаграмме.



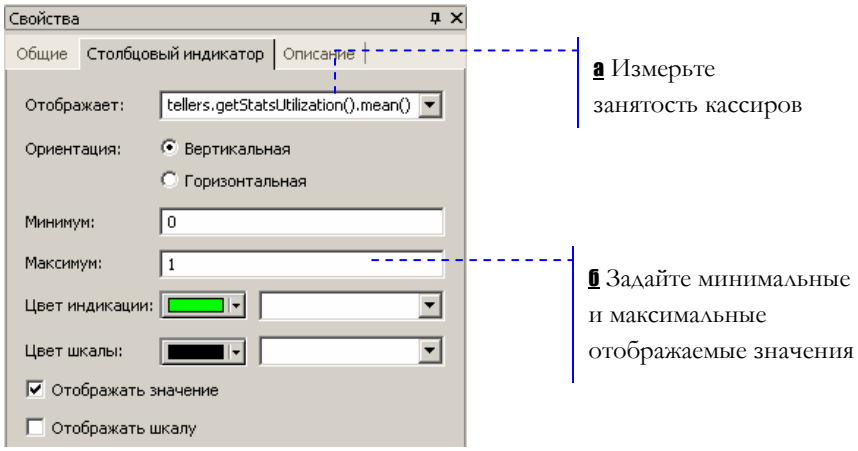
Динамические группы фигур создаются и добавляются на анимацию во время «прогона» модели.

Чтобы добавить фигуру в группу фигур, щелкните правой кнопкой мыши по значку группы фигур и выберите *Добавить/удалить фигуры* из контекстного меню.



Затем щелкните мышью по фигуре картинки для того, чтобы добавить ее к этой группе фигур. Картинка будет подсвечена. Чтобы выйти из режима добавления/удаления, щелкните мышью по анимационной диаграмме.

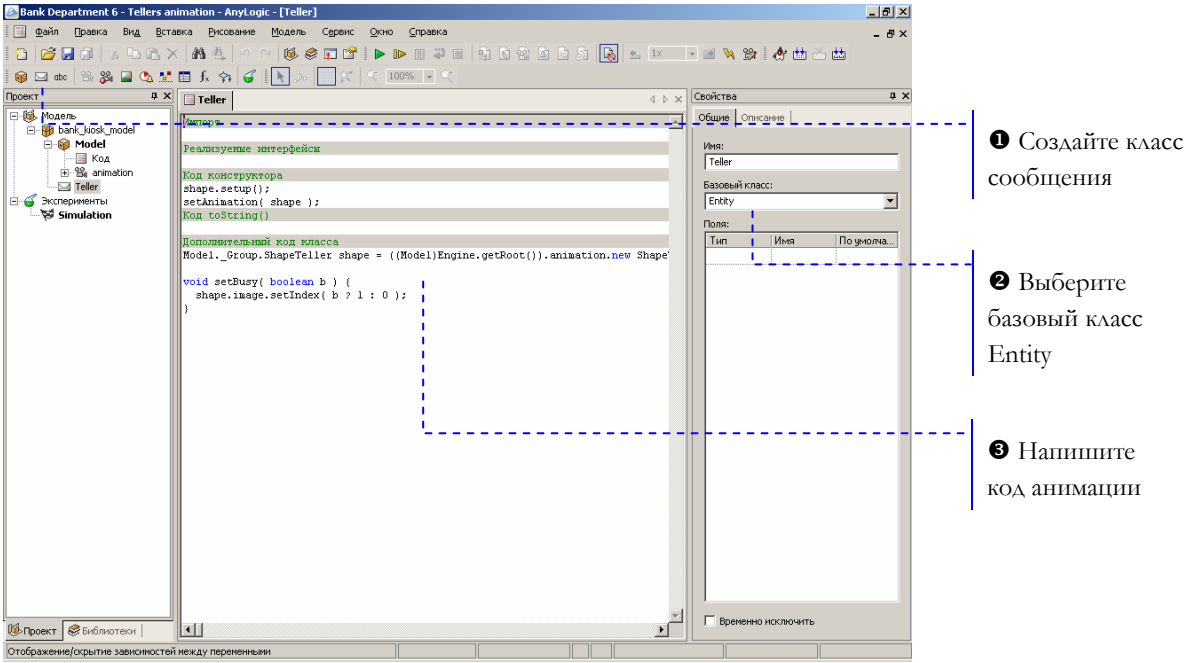
❺ Поместите на диаграмму столбцовый индикатор, показывающий то, какую часть своего рабочего времени кассир тратит на обслуживание клиентов. Задайте следующие свойства индикатора:



а Чтобы измерить занятость ресурса, мы воспользуемся функцией `getStatsUtilization()` объекта Resource. Затем мы получим среднее значение с помощью функции `mean()`.

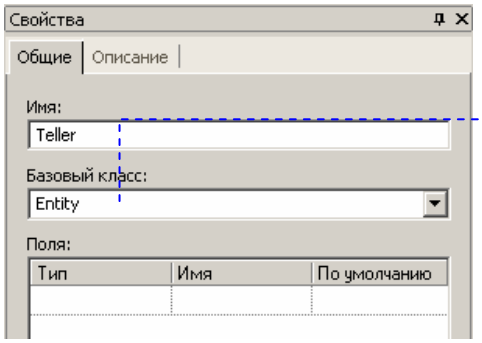
Теперь мы создадим новый класс сообщения, чтобы задать ресурсы модели. Сообщения этого класса будут представлять в нашей модели кассиров.

► Создайте класс сообщений `Teller`



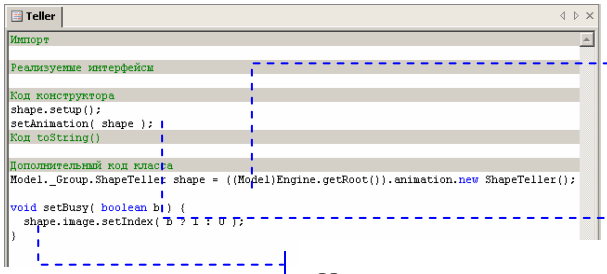
❶ Чтобы создать новый класс сообщения, щелкните мышью по кнопке *Новый класс сообщения*  . Назовите класс `Teller`.

❷



а Введите имя базового класса для созданного сообщения

❸



а Создайте анимацию ресурса

б Проинициализируйте анимацию ресурса

в Напишите код для изменения анимации во время работы модели

а Чтобы создать анимацию этого ресурса (экземпляр динамической группы фигур `ShapeTeller`), напишите следующий код в поле *Дополнительный код класса*:

```
Model._Group.ShapeTeller shape =
((Model)Engine.getRoot()).animation.new
ShapeTeller();
```

б Чтобы добавить созданную группу фигур на анимацию, напишите следующий код в поле *Код инициализации*:

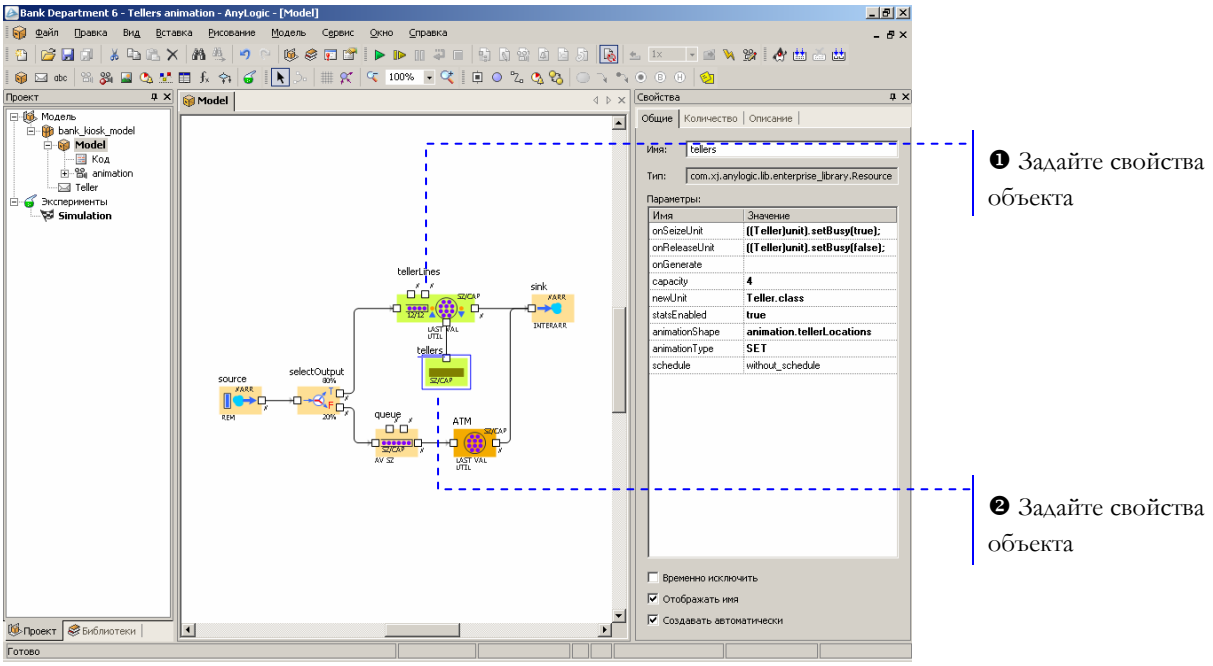
```
shape.setup();
setAnimation( shape );
```

➲ В следующей версии AnyLogic анимационный подход будет упрощен.

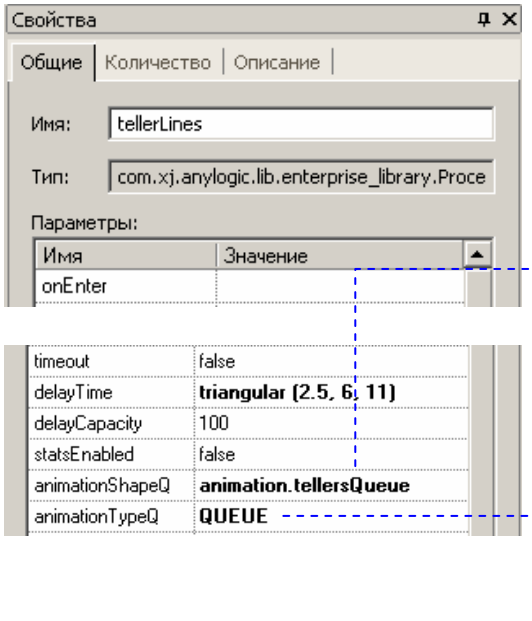
в Чтобы изменить внешний вид анимации во время работы модели, напишите следующую функцию:

```
void setBusy( boolean b ) {
  shape.image.setIndex( b ? 1 : 0 );
}
```

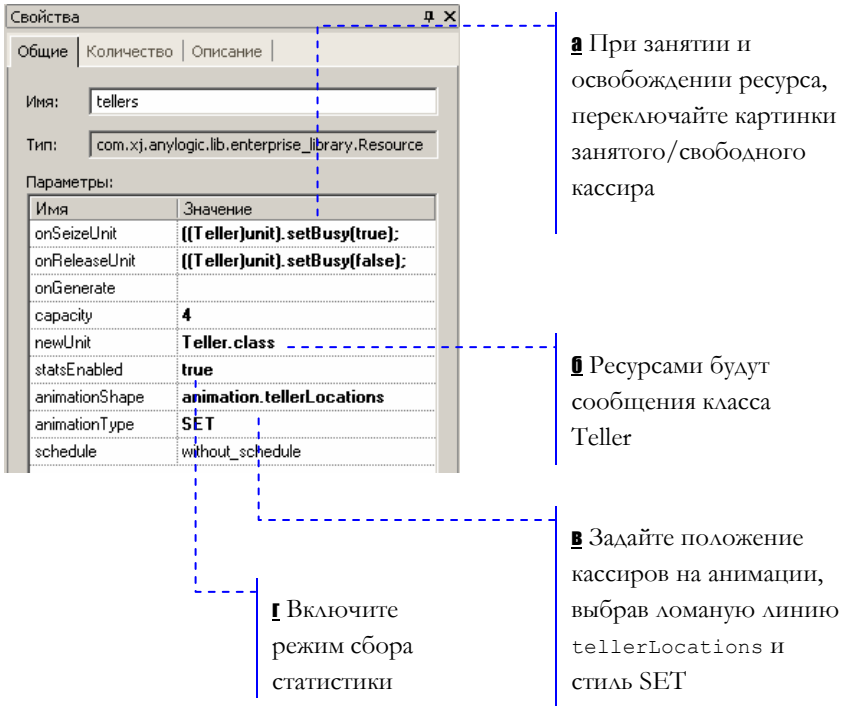
Теперь мы зададим анимационные свойства для объектов блок-схемы.



❶ Задайте следующие свойства объекта:



❷ Задайте следующие свойства объекта:



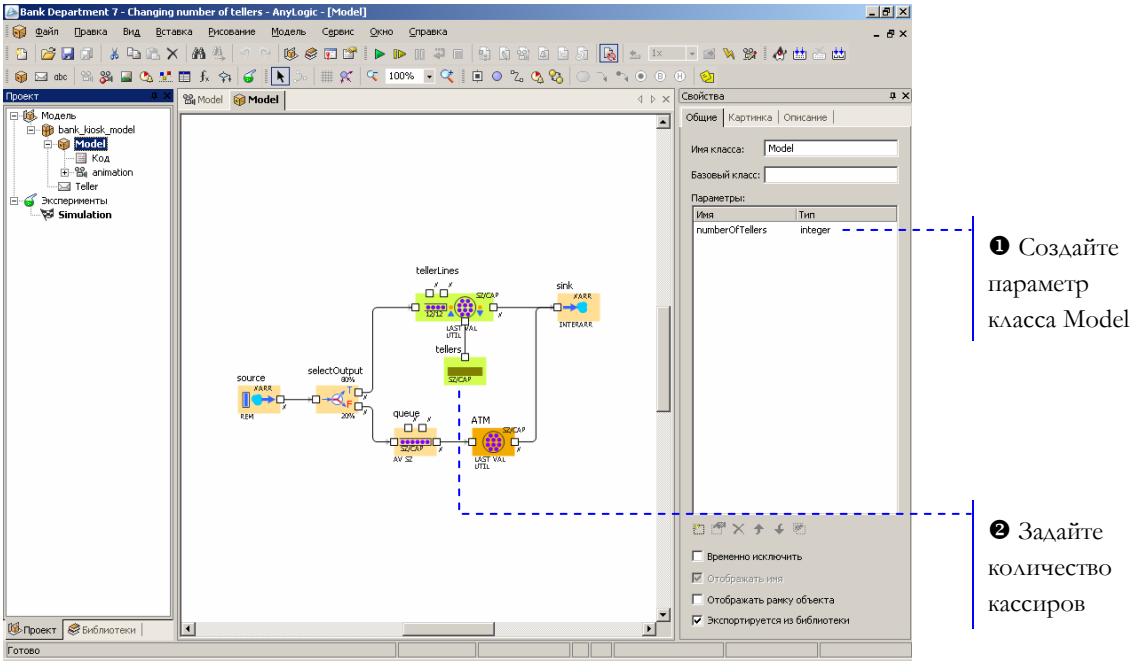
Запустите модель щелчком мыши по кнопке *Запустить*. С помощью созданной анимации Вы сможете проследить, сколько кассиров в данный момент времени занято обслуживанием клиентов.

Вы можете изменить скорость выполнения модели с помощью кнопок панели инструментов *Уменьшить скорость* и *Увеличить скорость*.

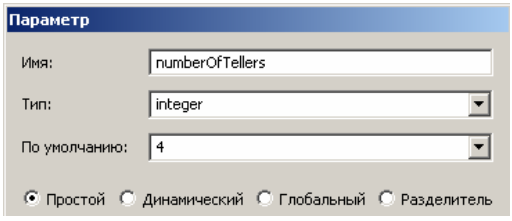
Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Bank Department 6 - Tellers animation.alp.

2.9 Изменение количества кассиров

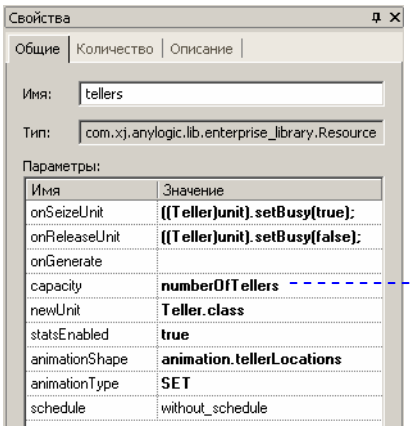
Теперь мы добавим специальный элемент управления, чтобы иметь возможность изменять количество кассиров во время работы модели.



❶ Добавьте параметр `numberOfTellers`, задающий количество кассиров:

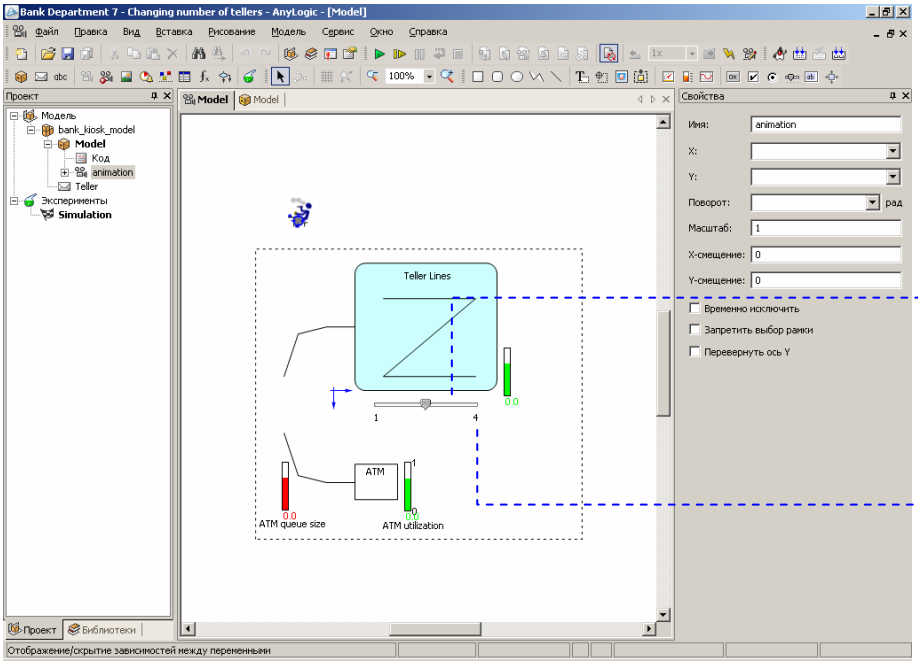


❷

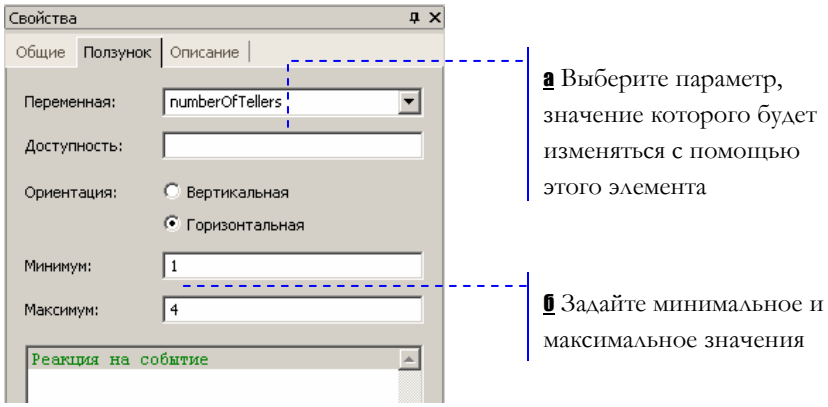


a Задайте количество кассиров параметром

► Добавьте элемент управления для изменения числа кассиров



❶ Задайте следующие свойства:



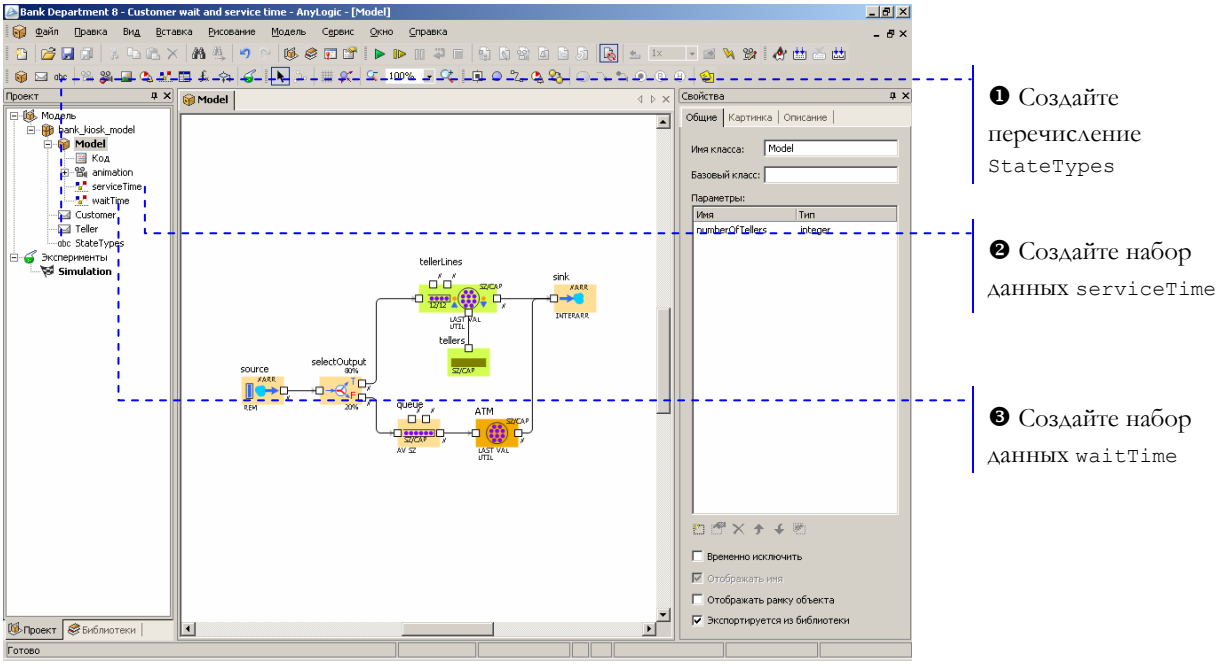
❷ Добавьте текстовые метки, отображающие границы диапазона значений ползунка.

Запустите модель щелчком мыши по кнопке *Запустить* . Теперь, изменяя количество кассиров во время работы модели, Вы можете сделать вывод о том, сколько служащих необходимо для нормальной работы банковского отделения при заданной интенсивности прихода клиентов.

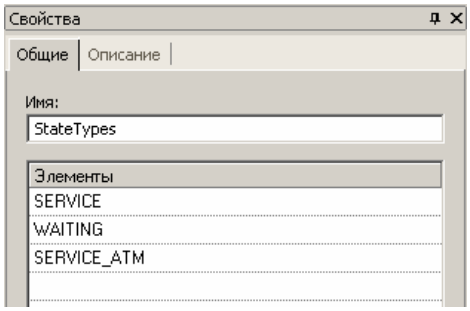
Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Bank Department 7 - Changing number of tellers.alp.

2.10 Сбор статистики о времени обслуживания клиента

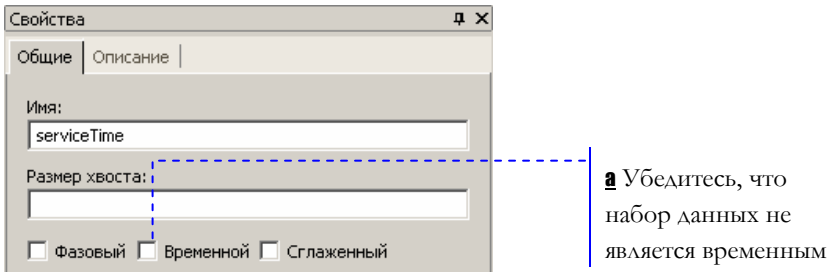
Мы хотим знать, сколько времени клиент проводит в банковском отделении и сколько времени он теряет, ожидая своей очереди. Мы соберем эту статистику с помощью наборов данных AnyLogic.



❶ Создайте перечисление `StateTypes` щелчком мыши по кнопке *Новое перечисление* . Создайте следующие элементы перечисления:



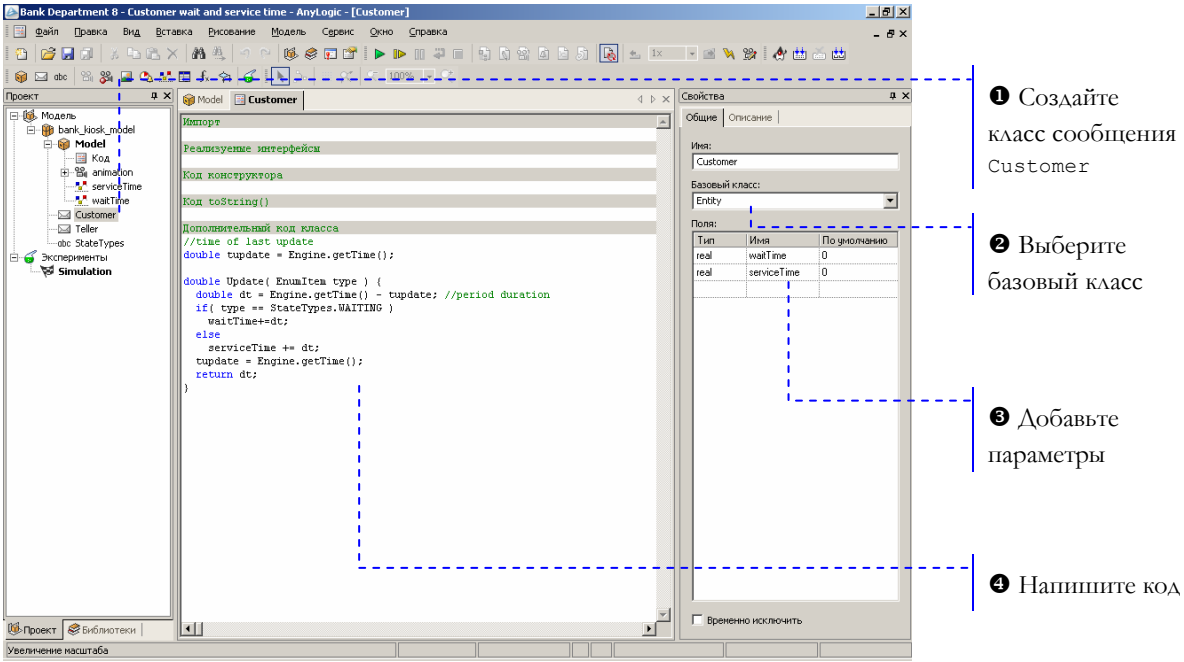
❷ Создайте набор данных `serviceTime` для подсчета времени обслуживания клиента, щелкнув мышью по кнопке *Новый набор данных* . Оставьте принятые по умолчанию свойства набора данных.



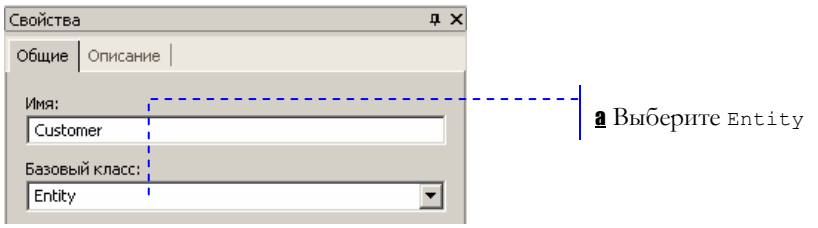
❸ Аналогично создайте набор данных `waitTime`.

Теперь мы создадим класс сообщения `Customer`. Сообщения этого класса будут представлять клиентов банковского отделения. Написав для этого класса специальную функцию, мы сможем проводить сбор статистики о том, сколько времени клиент провел в нашем банке.

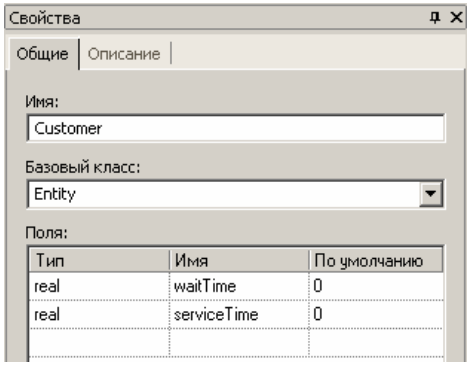
► Создайте класс сообщения



❷



❸ Добавьте параметры для хранения информации о проведенном времени:



Значения параметров будут обновляться по мере того, как будут обслуживаться клиенты.

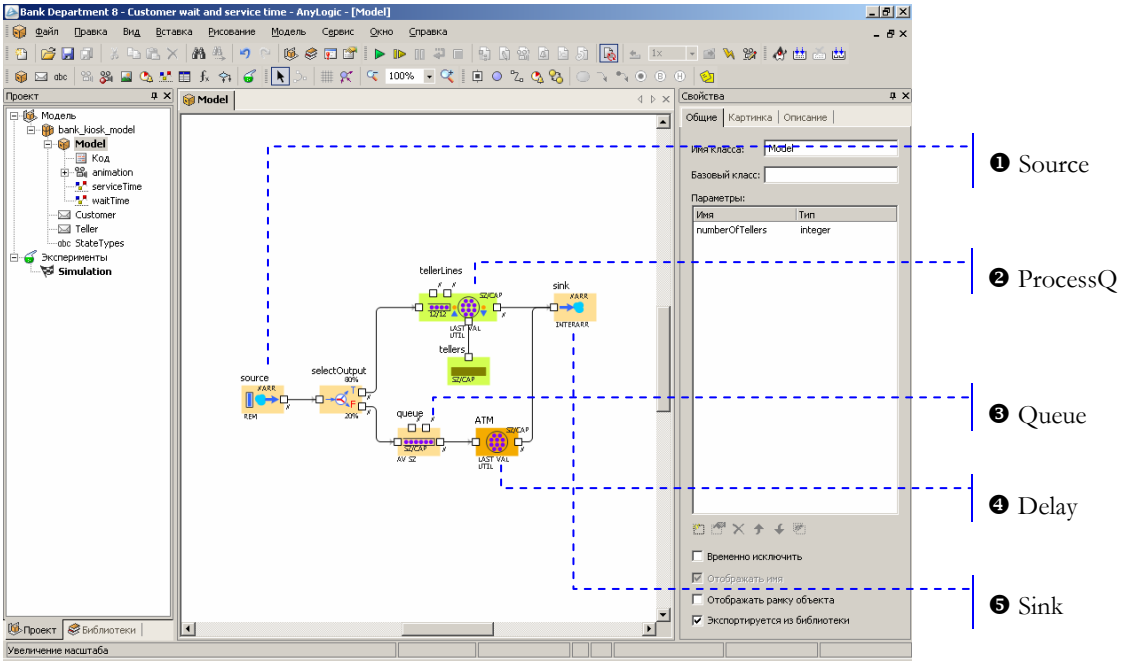
❹ Напишите следующий код в поле *Дополнительный код класса*:

```
double tupdate = Engine.getTime();
double Update( EnumItem type ) {
  double dt = Engine.getTime() - tupdate;
  if( type == StateTypes.WAITING )
    waitTime+=dt;
  else
    serviceTime+=dt;
  tupdate = Engine.getTime();
  return dt;
}
```

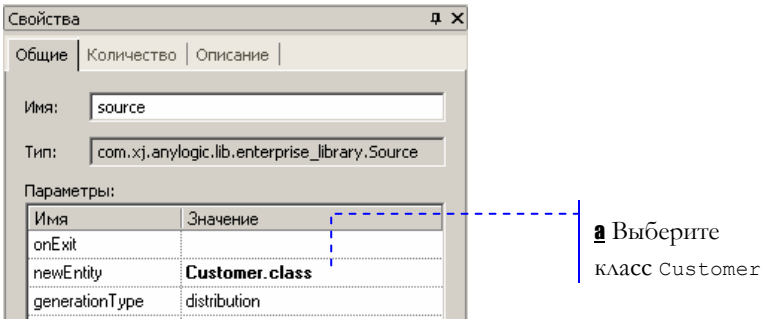
Функция `Update()` производит сбор временной статистики. Она вызывается после того, как будет проведена какая-то операция, и добавляет время, затраченное клиентом на эту операцию к соответствующему набору данных.

Теперь мы можем вычислить время, которое тратится персоналом банка на обслуживание клиентов, и время, которое клиенты тратят на ожидание своей очереди.

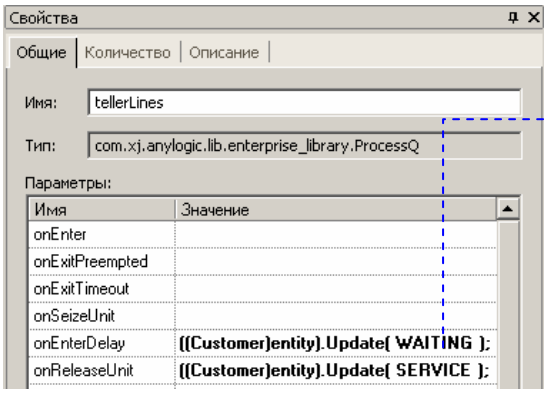
► Вычислите время обслуживания и время ожидания клиента



❶ Задайте следующие свойства объекта:

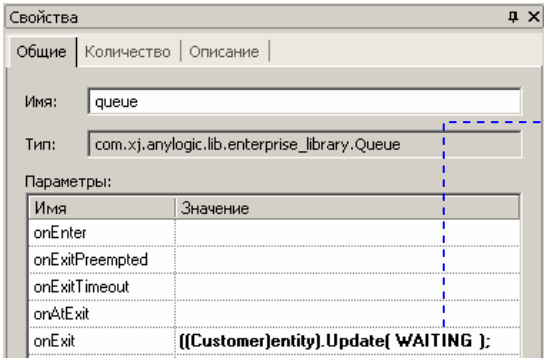


❷ Задайте следующие свойства объекта:



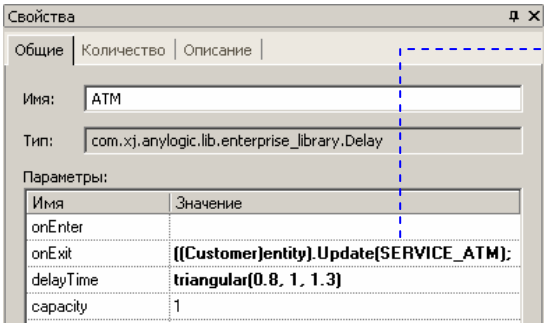
a Напишите следующий код

❸ Задайте следующие свойства объекта:



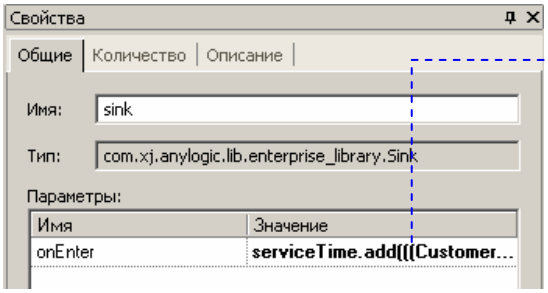
a Напишите следующий код

❹ Задайте следующие свойства объекта:



a Напишите следующий код

❺ Задайте следующие свойства объекта:



a Напишите следующий код

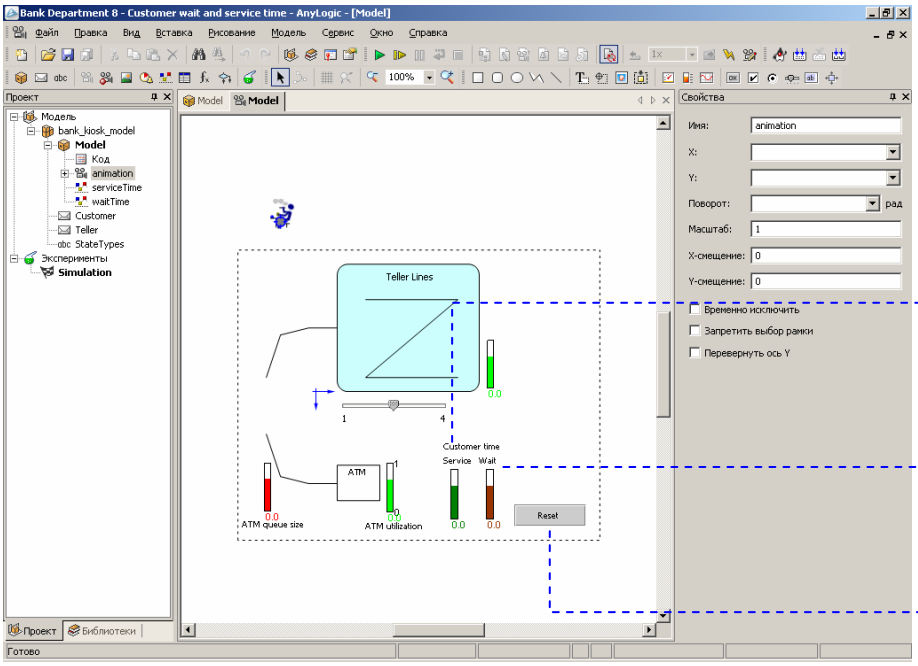
a Напишите следующий код, чтобы сохранить в наборах данных

данные о клиенте, покидающем банковское отделение:

```
serviceTime.add(((Customer)entity).serviceTime);
waitTime.add(((Customer)entity).waitTime);
```

Мы хотим отобразить собранную статистику на анимации.

► Добавьте на анимацию индикаторы

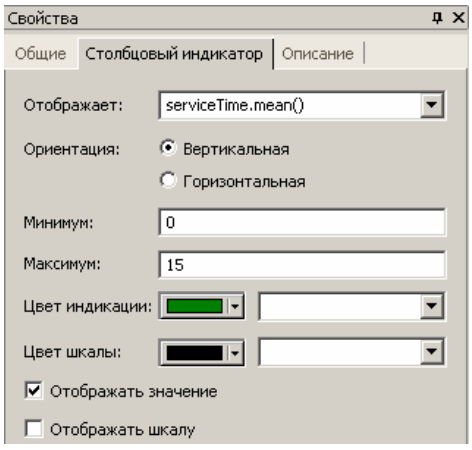


❶ Добавьте индикатор времени обслуживания клиента

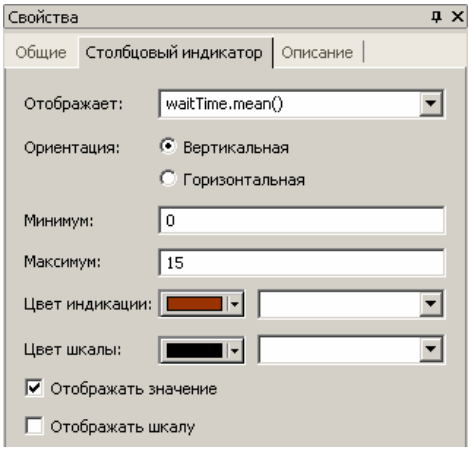
❷ Добавьте индикатор времени ожидания клиента

❸ Добавьте кнопку сброса статистики

❶ Задайте следующие свойства индикатора:

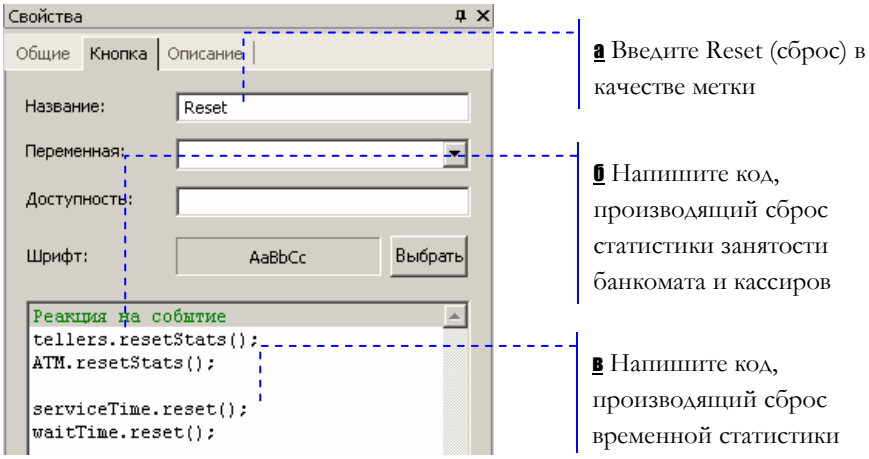


❷ Задайте следующие свойства индикатора:



❸ Мы добавим кнопку, чтобы при изменении числа кассиров сбрасывать статистику, собранную для старого значения.

Щелкните мышью по кнопке *Кнопка*  , а затем щелкните по анимационной диаграмме. Задайте следующие свойства кнопки:



Статистика сбрасывается с помощью специальных функций:

б Функция объекта Delay `resetStats()` сбрасывает статистику, собранную этим объектом.

в Функция набора данных `reset()` сбрасывает статистику, собранную набором данных.

Запустите модель. Теперь в нашей модели происходит сбор временной статистики. Сбросить статистику можно щелкнув по созданной нами кнопке Reset.

➲ Вы можете производить сброс статистики в заданный момент времени или по происхождению какого-то события с помощью таймеров (подробную информацию о таймерах Вы можете найти в *Руководстве пользователя*).

➘ Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Bank Department 8 - Customer wait and service time.alp.

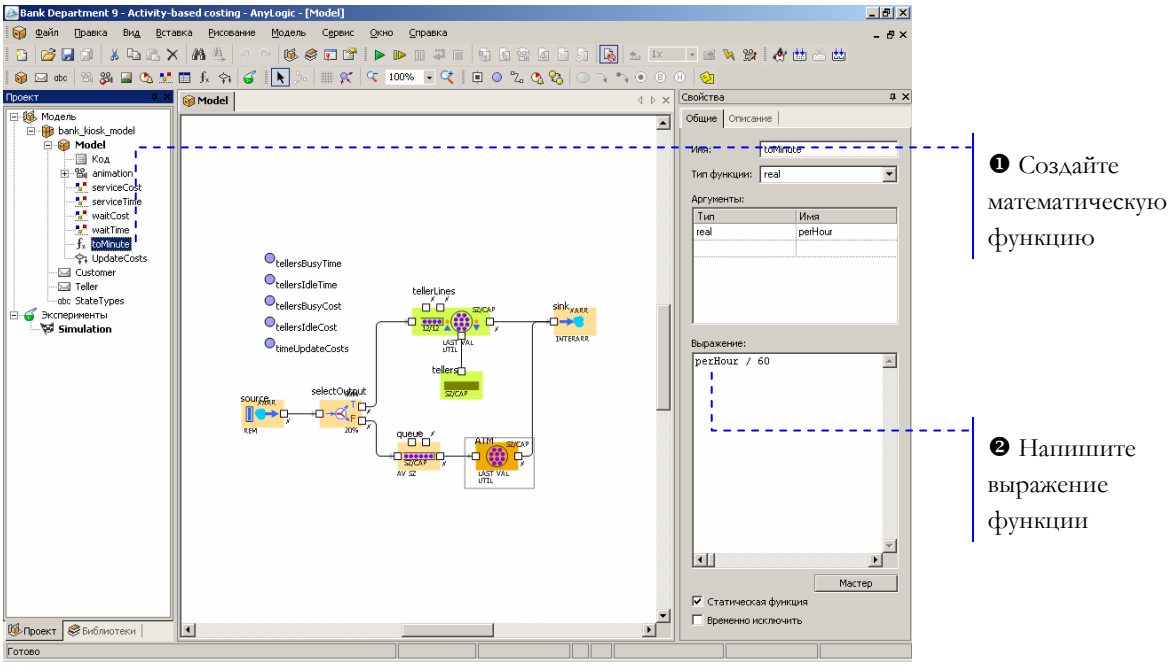
2.11 Оценка затрат операций

Никогда еще моделирование и анализ бизнес-процессов не было таким простым, как в AnyLogic. Enterprise Library предоставляет инструменты для проведения в Вашей системе оценки затрат операций. Метод *оценки затрат операций* (activity-based costing, ABC-метод) оценивает процесс и эффективность операций, определяет стоимость обслуживания/производства и указывает возможности для усовершенствования продуктивности и эффективности процесса. С помощью этого метода производится количественная оценка стоимости и производительности операций, эффективности использования ресурсов и стоимости объектов.

Мы проведем учет затрат операций в нашем примере, чтобы понять, во сколько в среднем обходится обслуживание одного клиента, и какие накладные расходы связаны с обслуживанием клиентов, ожидающих своей очереди.

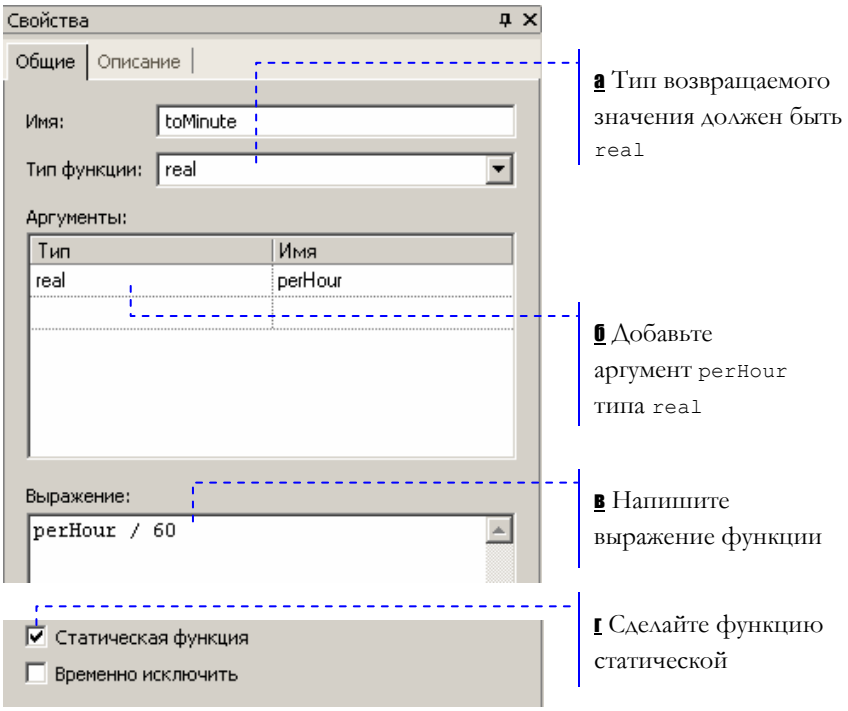
Вначале мы напишем вспомогательную функцию для пересчета почасовой зарплаты в поминутную.

► Напишите функцию пересчета зарплаты



❶ Щелкните мышью по кнопке панели инструментов *Новая математическая функция* . Назовите функцию `toMinute`.

❷ На странице свойств задайте тип, аргументы и выражение функции:



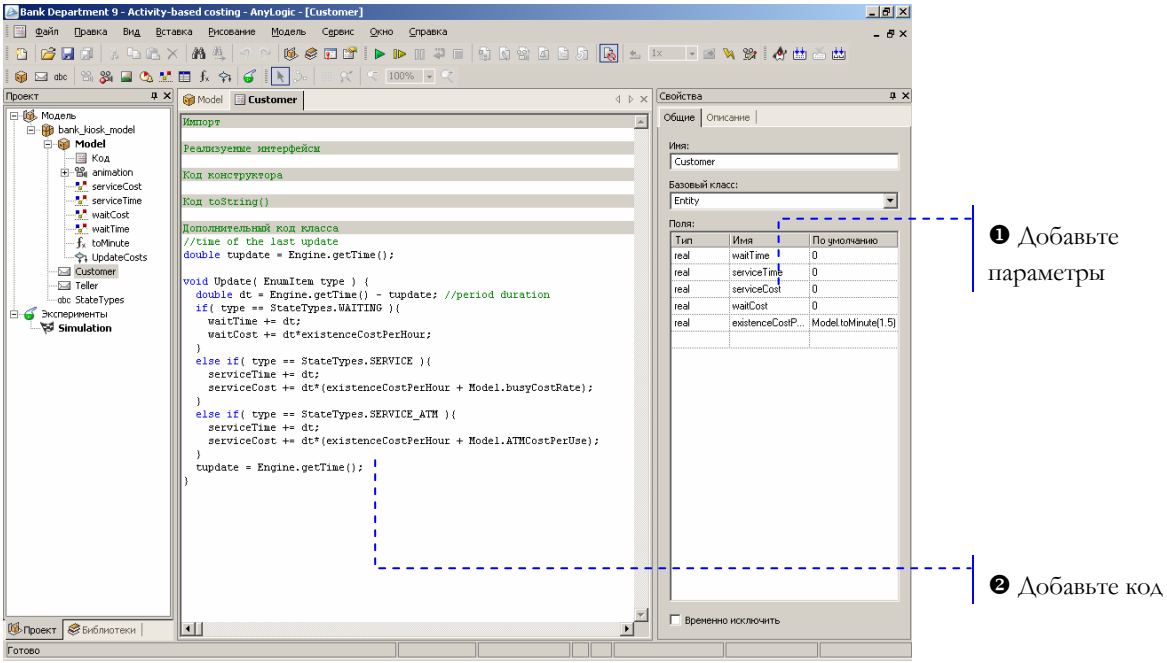
В Напишите следующее выражение:

```
perHour / 60
```

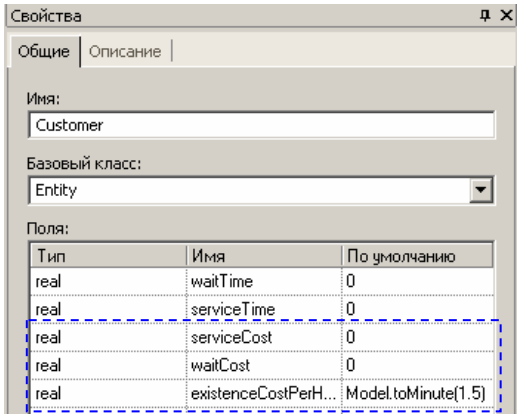
Г Мы сделаем функцию статической, поскольку эта функция не использует значений, специфичных для конкретного экземпляра класса Model. Функция связана лишь с самим классом Model, и мы будем вызывать ее так: `Model.toMinute()`. В противном случае нам пришлось бы вызывать ее со ссылкой на конкретный экземпляр класса Model.

Теперь мы изменим класс сообщений, чтобы собирались и данные о затратах.

► Измените класс сообщения `Customer`



❶ Добавьте параметры, хранящие информацию о затратах:



Параметр `serviceCost` будет хранить информацию о том, во сколько компании обходится обслуживание этого клиента.

Заметим, что банк несет издержки, связанные с обслуживанием клиентов, ожидающих своей очереди.

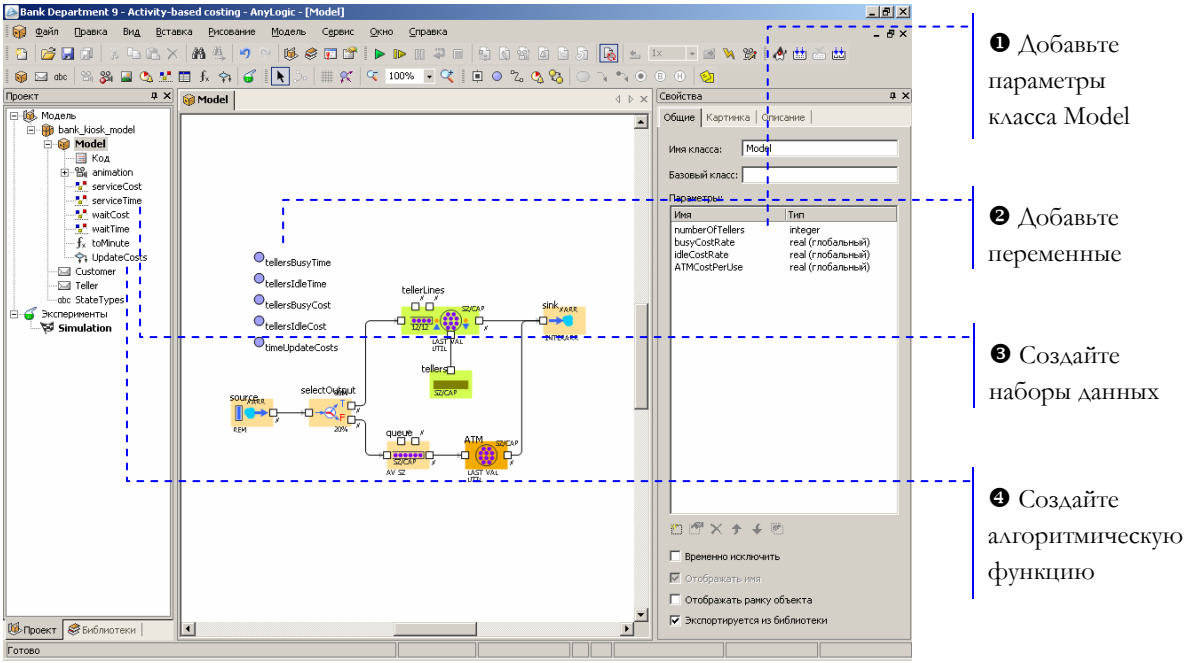
Параметр `existenceCostPerHour` задает, во сколько обходится компании пребывание клиента в банке.

❷ Откройте окно класса `Customer` двойным щелчком мыши по элементу `Customer` в дереве модели, и измените функцию `Update()` в поле *Дополнительный код класса*:

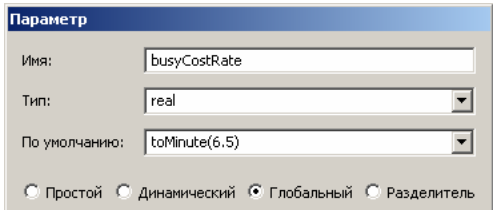
```
void Update( EnumItem type ) {
  double dt = Engine.getTime() - tupdate;
  if( type == StateTypes.WAITING ){
    waitTime += dt;
    waitCost += dt*existenceCostPerHour;
  }
  else if( type == StateTypes.SERVICE ){
    serviceTime += dt;
    serviceCost += dt*(existenceCostPerHour +
Model.busyCostRate);
  }
  else if( type == StateTypes.SERVICE_ATM ){
    serviceTime += dt;
    serviceCost += dt*(existenceCostPerHour +
Model.ATMCostPerUse);
  }
  tupdate = Engine.getTime();
}
```

Функция будет обновлять статистику затрат на обслуживание и ожидание клиента.

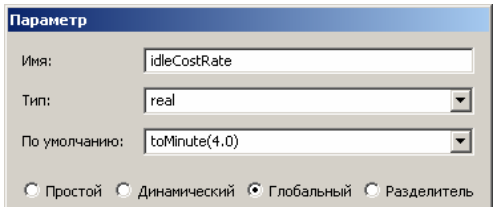
Теперь мы добавим в модель вспомогательные элементы, собирающие статистику затрат компании.



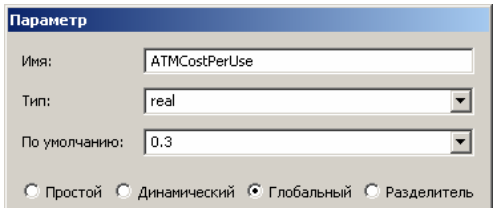
❶ Создайте параметры, задающие заработную плату кассиров. Задайте параметры следующим образом:



Мы платим кассиру $6.5 в час, если он был занят обслуживанием клиентов и $4.0, если он был свободен.



Создайте параметр, задающий расходы, связанные с работой банкомата:



Одна операция банкомата обходится компании в $0.30.

❷ Создайте переменную `timeUpdateCosts`, щелкнув мышью по кнопке панели инструментов *Переменная*  , а затем щелкнув по структурной диаграмме. Создайте еще несколько переменных и назовите их так:

- `tellersIdleTime`
- `tellersBusyTime`
- `tellersIdleCost`
- `tellersBusyCost`

Эти переменные будут хранить информацию о том, сколько времени кассиры были заняты обслуживанием клиентов, и сколько им требуется выплатить за работу.

❸ Создайте два набора данных и назовите их так:

- `waitCost`
- `serviceCost`

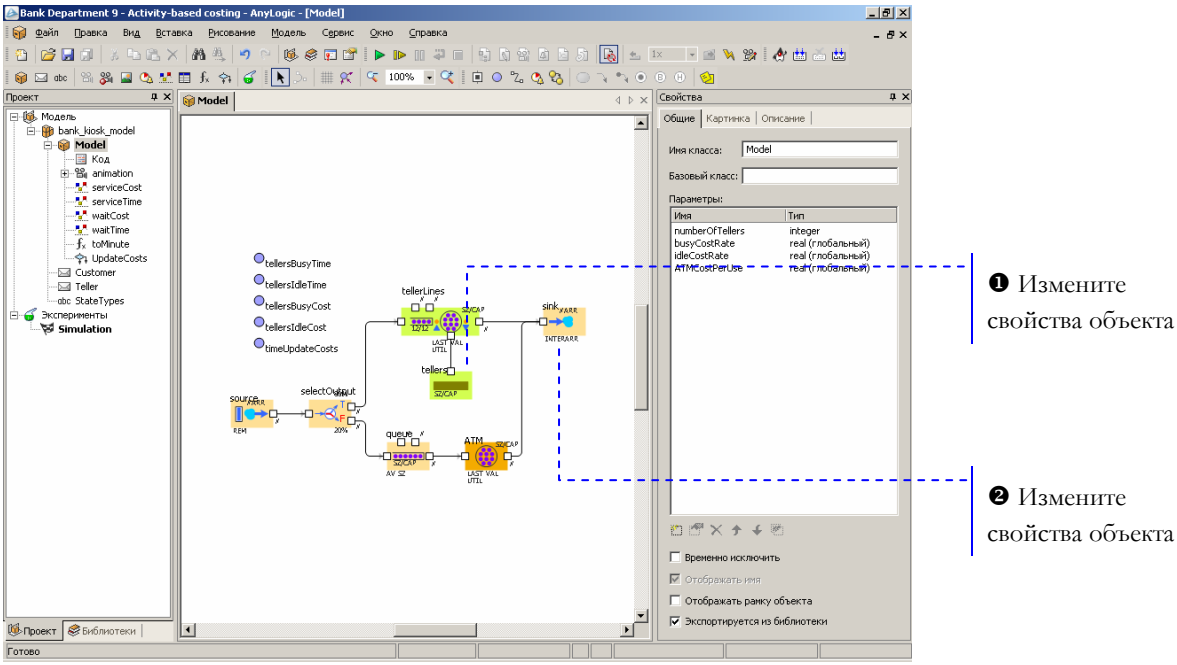
Убедитесь, что наборы данных не являются временными; то есть у обоих наборов данных флажок *Временной* сброшен. Эти наборы данных будут хранить статистику затрат компании на обслуживание клиентов.

❹ Создайте алгоритмическую функцию, которая будет обновлять статистику. Для этого щелкните мышью по кнопке панели инструментов *Новая алгоритмическая функция*  . Назовите ее `UpdateCosts`.

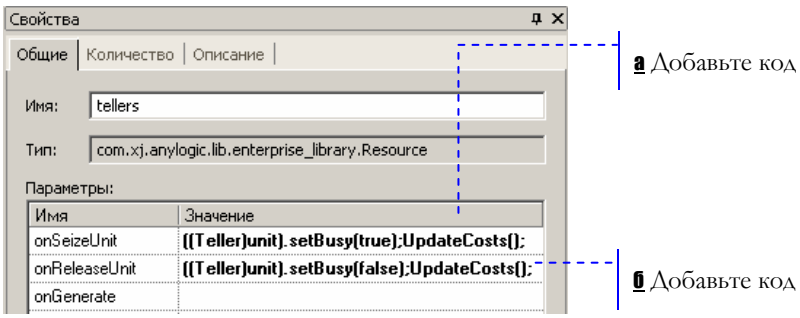
```
double dt = getTime() - timeUpdateCosts;
  tellersIdleTime += tellers.size()*dt;
  tellersBusyTime += (tellers.capacity-
tellers.size())*dt;
  tellersIdleCost += tellersIdleTime*idleCostRate;
  tellersBusyCost += tellersBusyTime*busyCostRate;
```

```
timeUpdateCosts = getTime();
```

► Сосчитайте затраты на обслуживание клиента



❶ Задайте следующие свойства:



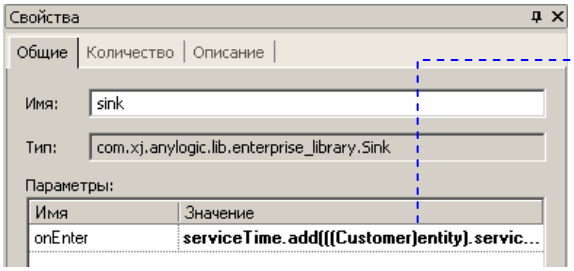
а Добавьте код, выделенный жирным шрифтом:

```
((Teller)unit).setBusy(true); UpdateCosts();
```

б Добавьте код, выделенный жирным шрифтом:

```
((Teller)unit).setBusy(true); UpdateCosts();
```

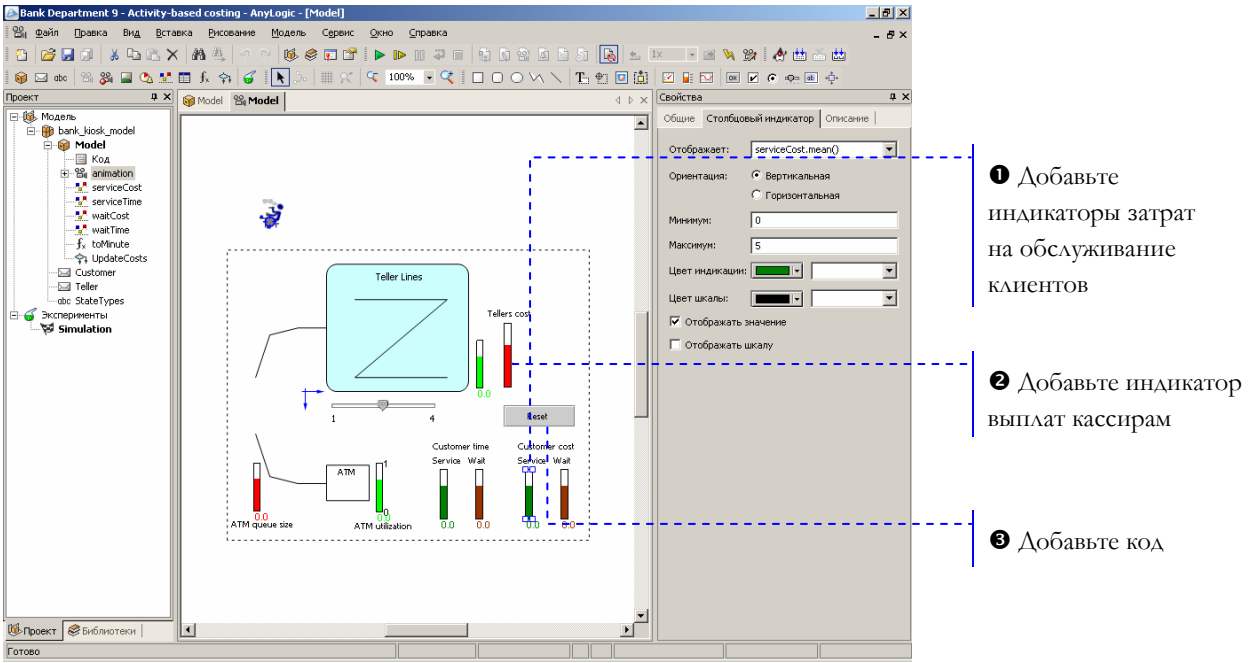
❷ Задайте следующие свойства объекта:



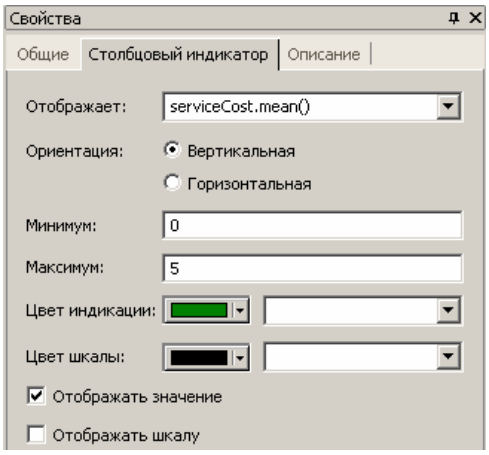
a Добавьте строки, выделенные жирным шрифтом, чтобы учесть затраты на обслуживание клиента:

```
serviceTime.add(((Customer)entity).serviceTime);
waitTime.add(((Customer)entity).waitTime);
serviceCost.add(((Customer)entity).serviceCost);
waitCost.add(((Customer)entity).waitCost)
```

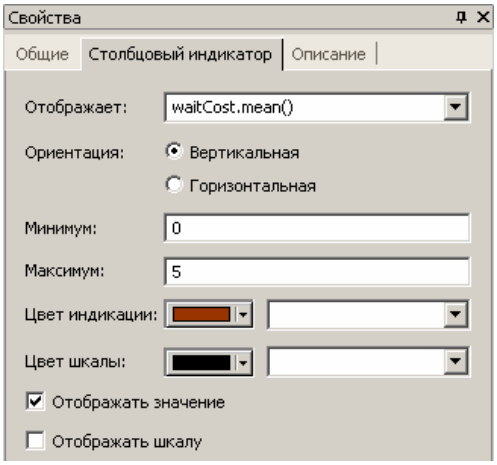
► Добавьте на анимацию индикаторы затрат



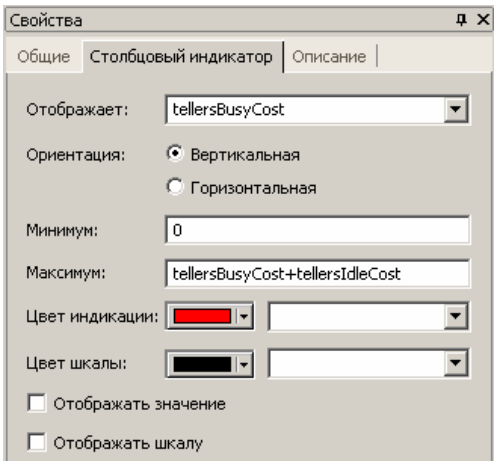
❶ Задайте следующие свойства индикатора стоимости обслуживания клиента:



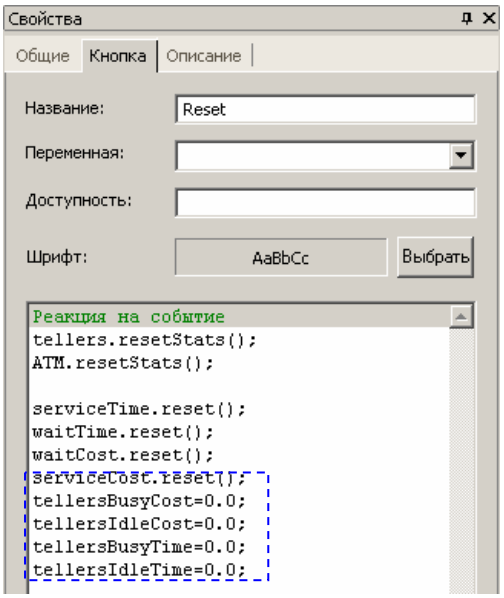
Задайте следующие свойства индикатора затрат, связанных с обслуживанием клиентов, ожидающих своей очереди:



❷ Добавьте индикатор, показывающий то, насколько эффективно используются кассиры. Задайте следующие свойства индикатора:



❸ Добавьте следующие строки кода:



Запустите модель щелчком мыши по кнопке *Запустить*  . Теперь Вы можете сделать вывод о том, откуда накапливается итоговая стоимость, и сделать соответствующие выводы, как улучшить эффективность работы, в то же время предоставляя лучший сервис.

 Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Bank Department 9 - Activity-based costing.alp.

3. Модель цеха предприятия

В этом разделе мы создадим модель цеха предприятия, производящего сборку изделий. Цех работает по следующей схеме:

- По конвейеру в цех поступают детали двух типов.
- На обрабатывающей станции производится обработка деталей.
- Далее детали сортируются и по разным конвейерам доставляются к сборочной станции.
- На сборочной станции производится сборка изделий.
- Готовые изделия увозятся из цеха по конвейеру.

3.1 Создание нового проекта

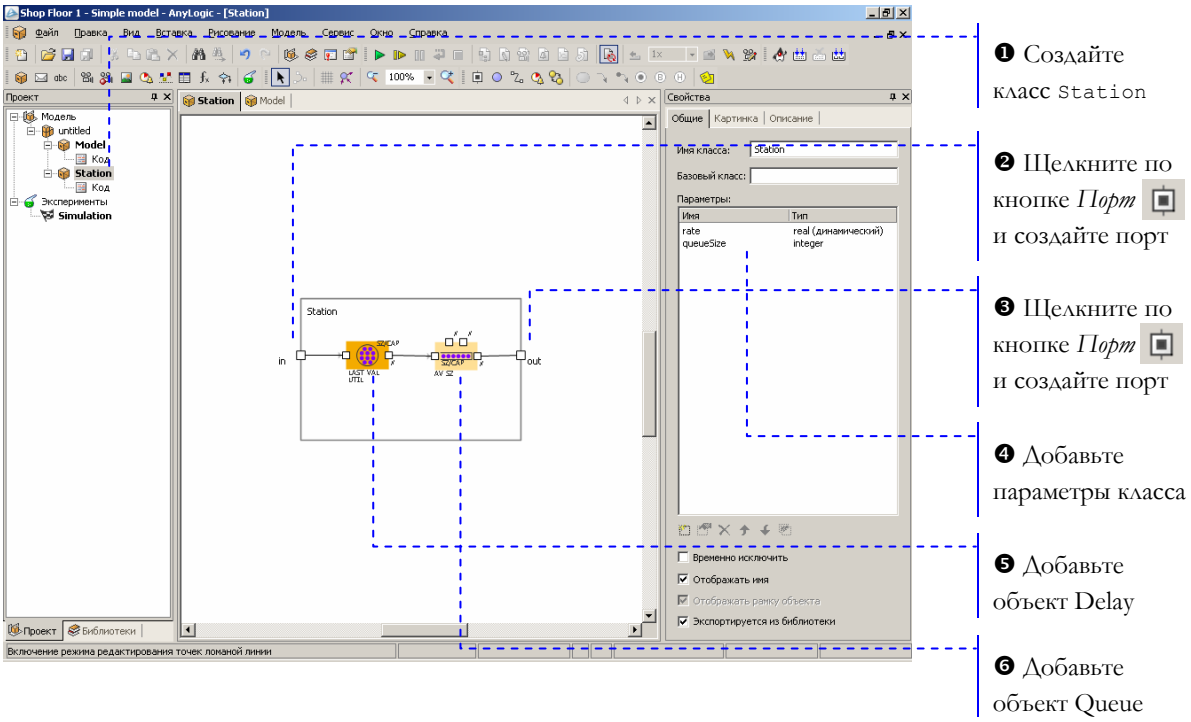
Создайте проект как описано в разделе 1.1, “Как создать модель”. Переименуйте класс `Main` в `Model`. Задайте режим реального времени с выполнением одной единицы модельного времени в одну секунду. В этой модели под единицей модельного времени мы будем понимать одну минуту работы цеха.

3.2 Создание простой модели

Сейчас мы создадим модель, моделирующую только поставку деталей и их обработку на обрабатывающей станции.

Если объект Вашей системы не может быть адекватно представлен ни одним из объектов библиотеки, Вы можете создать свой собственный класс объекта с необходимыми свойствами и структурой. В нашем случае нам нужно будет создать новый класс для того, чтобы представить обрабатывающую станцию. Объект созданного класса добавляется в модель так же просто, как и объект библиотеки – простым перетаскиванием класса мышью из окна *Проект* на блок-схему системы.

► Создайте новый класс

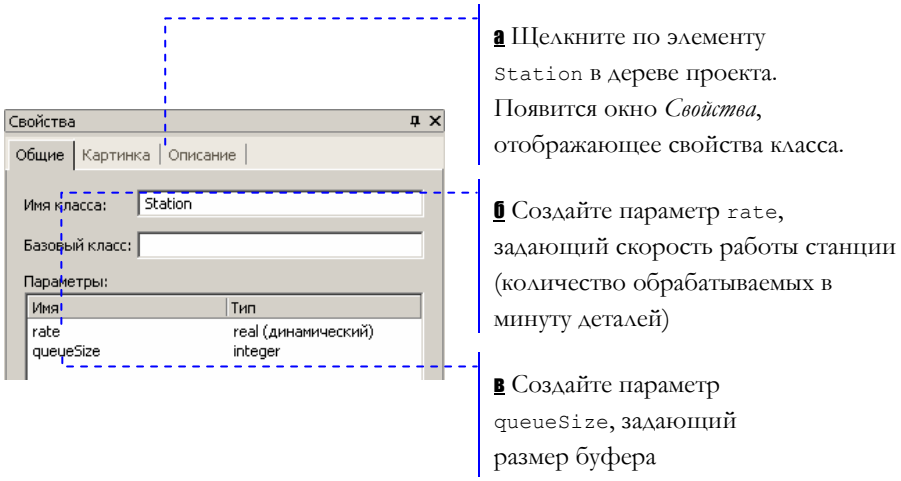


❶ Чтобы создать новый класс, щелкните мышью по кнопке панели инструментов *Новый класс активного объекта* . Назовите класс `Station`. Автоматически появится окно структурной диаграммы класса.

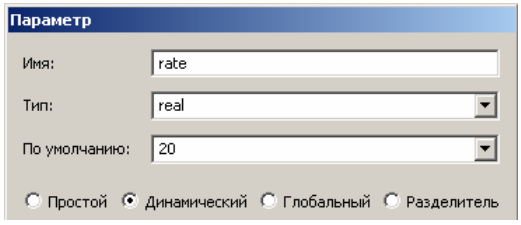
❷ Создайте порт и назовите его `in`. Чтобы переименовать порт, щелкните мышью по порту и нажмите F2.

❸ Создайте порт `out`. Оставьте принятые по умолчанию свойства.

❹ С помощью параметров класса Вы можете задавать разные характеристики для разных объектов этого класса (индивидуально для каждой обрабатывающей станции).



6 В окне создания параметра, укажите, что это *Динамический* параметр типа `real`. Введите значение по умолчанию: 20.



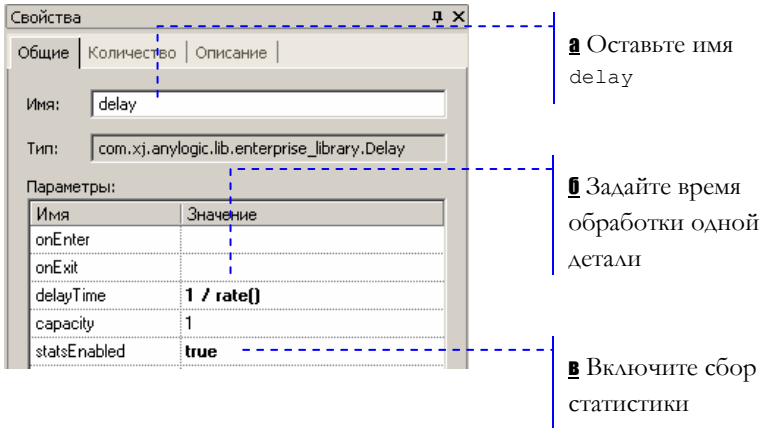
Значение динамического параметра пересчитывается при каждом вызове параметра. Поэтому Вы можете задавать динамически меняющиеся выражения, например, случайные числа, и каждый вызов параметра будет возвращать новое значение выражения: например, следующее случайное число.

Мы выбираем динамический параметр, потому что допускаем возможность изменения скорости работы станции.

B Размер буфера изменяться не будет, поэтому мы используем *Простой* параметр типа `integer`. Задайте значение по умолчанию: 2.

Чтобы получить значение динамического параметра, используйте круглые скобки, как при вызове функции, например, `rate()`. Чтобы получить значение простого параметра, просто используйте имя параметра; например, `queueSize`.

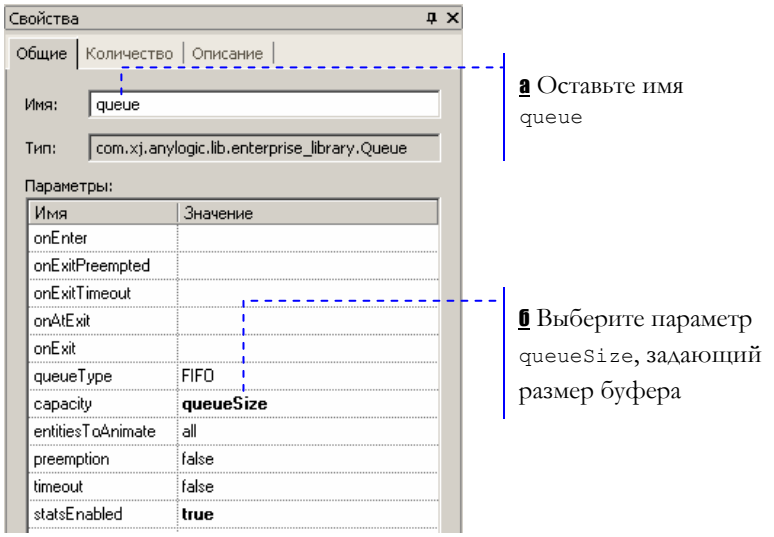
❺ Задайте следующие свойства объекта:



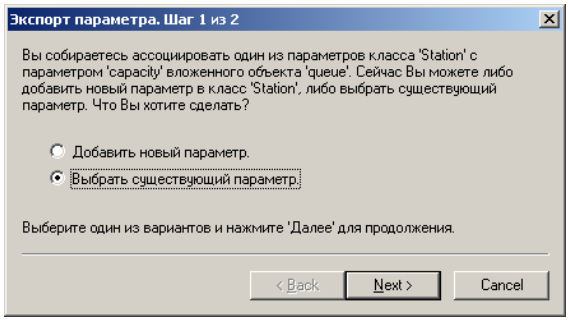
б Поскольку параметр `rate` задает количество обрабатываемых деталей в единицу времени, то выражение `1/rate()` будет высчитывать время обработки одной детали.

❻ Детали будут помещаться в буфер в случае, если они не смогут быть сразу же помещены на конвейер, который ведет от обрабатывающей станции.

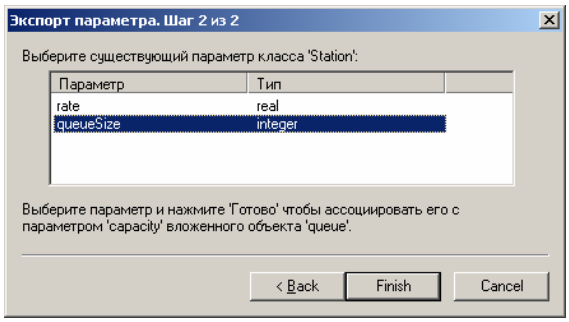
Задайте следующие свойства объекта:



❻ Чтобы связать параметр с параметром класса, щелкните правой кнопкой мыши по строке параметра, выберите *Экспортировать…* из контекстного меню, затем в появившемся диалоговом окне выберите *Выбрать существующий параметр*.



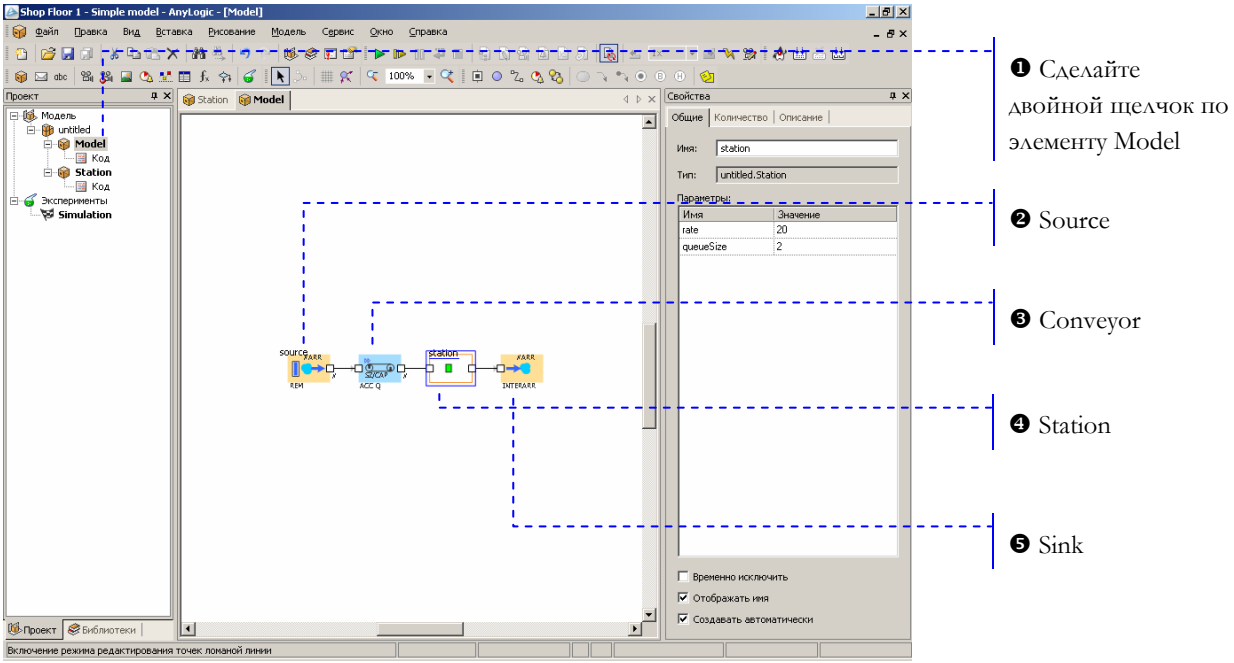
Выберите `queueSize` из списка параметров класса.



❼ Каждый объект Enterprise Library отображается на блок-схеме по-своему. Вы можете изменить изображение объекта созданного Вами класса, нарисовав для него значок. Для этого щелкните правой кнопкой мыши по классу (в нашем случае `Station`) в дереве модели, выберите *Новый значок* из контекстного меню и нарисуйте значок в появившемся окне.

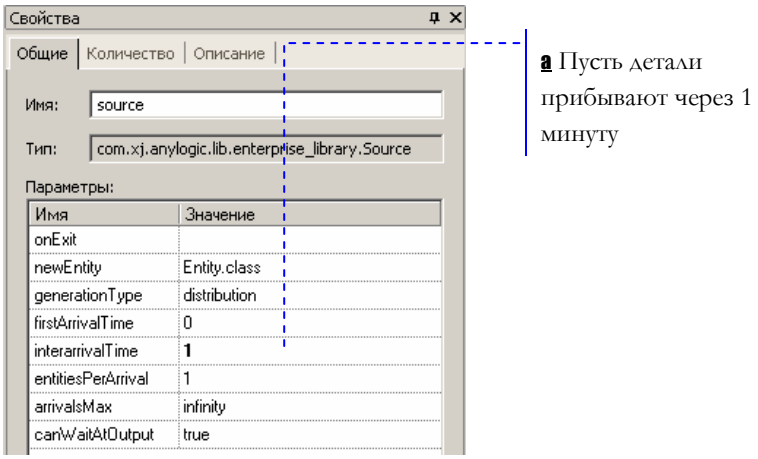
Теперь мы создадим блок-схему нашей модели.

► Создайте блок-схему



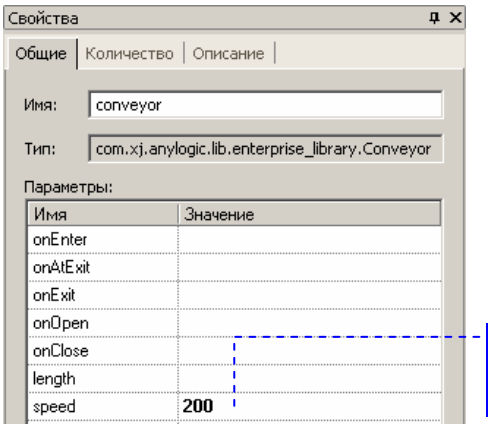
❷ Добавьте на блок-схему объект Source, моделирующий поставку деталей.

Задайте следующие свойства объекта:



❸ Добавьте объект конвейера Conveyor.

Задайте следующие свойства объекта:



❹ Оставьте принятые по умолчанию свойства объекта.

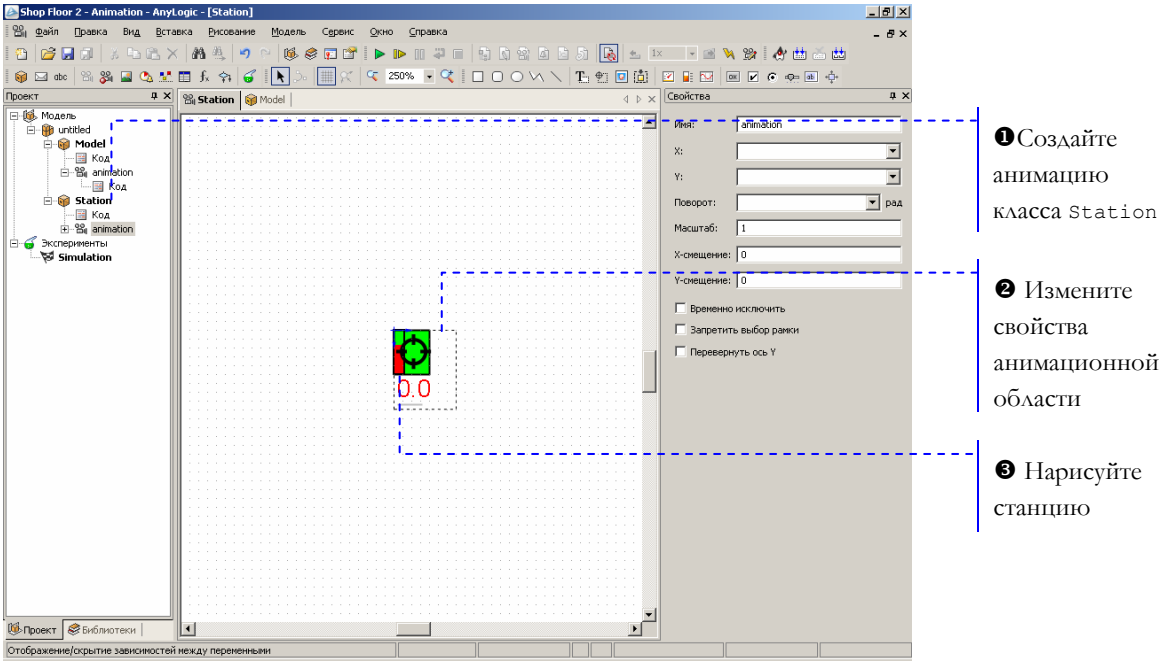
❺ Оставьте принятые по умолчанию свойства объекта.

Запустите модель щелчком по кнопке *Запустить*  .

Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Shop Floor 1 - Simple model.alp.

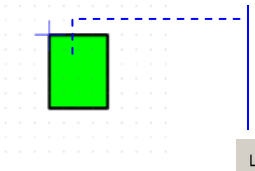
3.3 Создание анимации

► **Создайте анимацию станции**

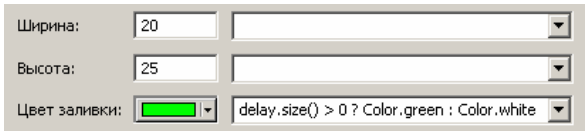


❷ Измените размер рамки, чтобы она была примерно 30 на 30 пикселов, и поместите левый верхний угол рамки в центр начала координат.

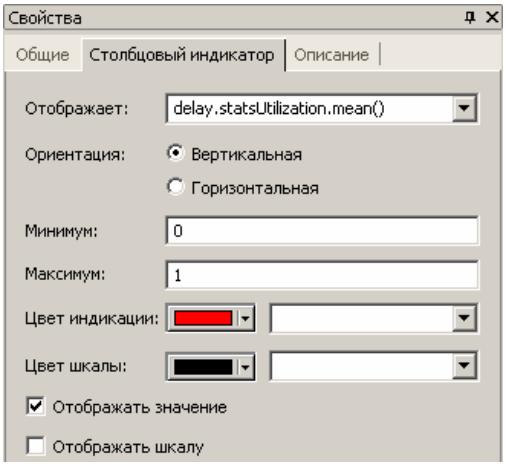
❸

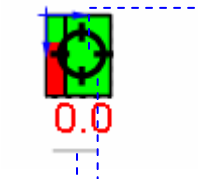


a Нарисуйте прямоугольник 20 на 25 пикселов и задайте цвет заливки:



б Нарисуйте индикатор занятости. Задайте прозрачный фон, выключите отображение шкалы, задайте максимальное значение равным 1 и минимальное – равным 0 и задайте выражение, определяющее отображаемое значение: `delay.getStatsUtilization().mean()`



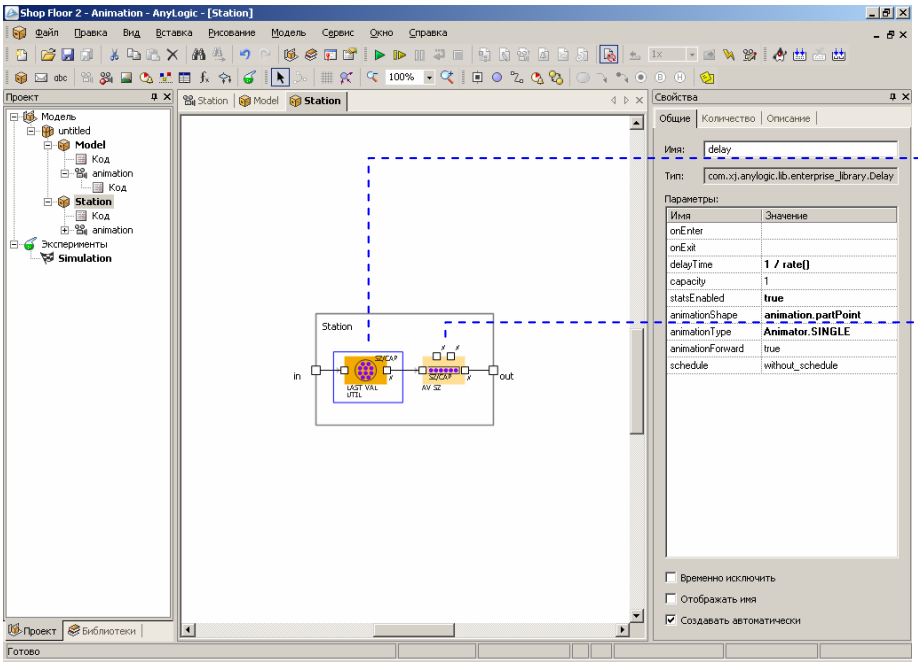


в Щелкните по кнопке *Группа фигур*  и поместите точку, чтобы показать местоположение обрабатываемой детали

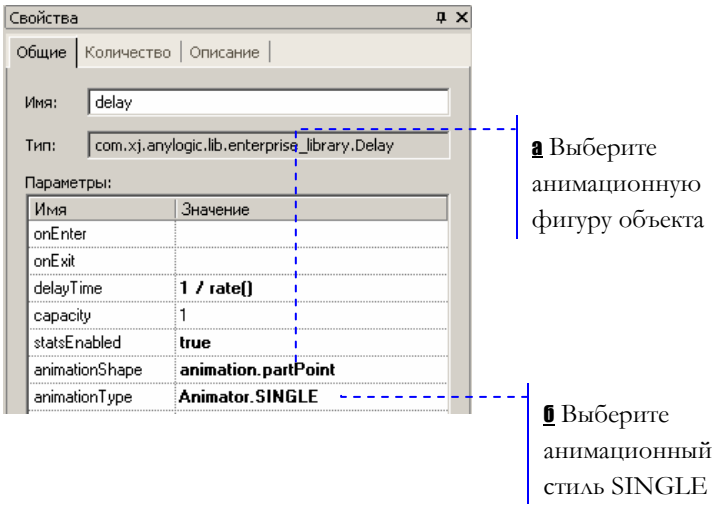
г Назовите группу фигур `partPoint`

д Создайте ломаную линию, показывающую наличие деталей в буфере. Назовите ломаную `queuePath` и спрячьте ее, введя `false` в поле *Видимость*

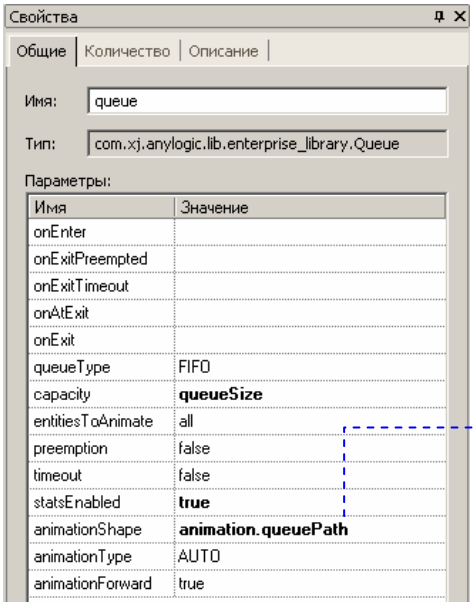
► Задайте анимационные свойства объектов обрабатывающей станции



❶ Задайте следующие свойства объекта:

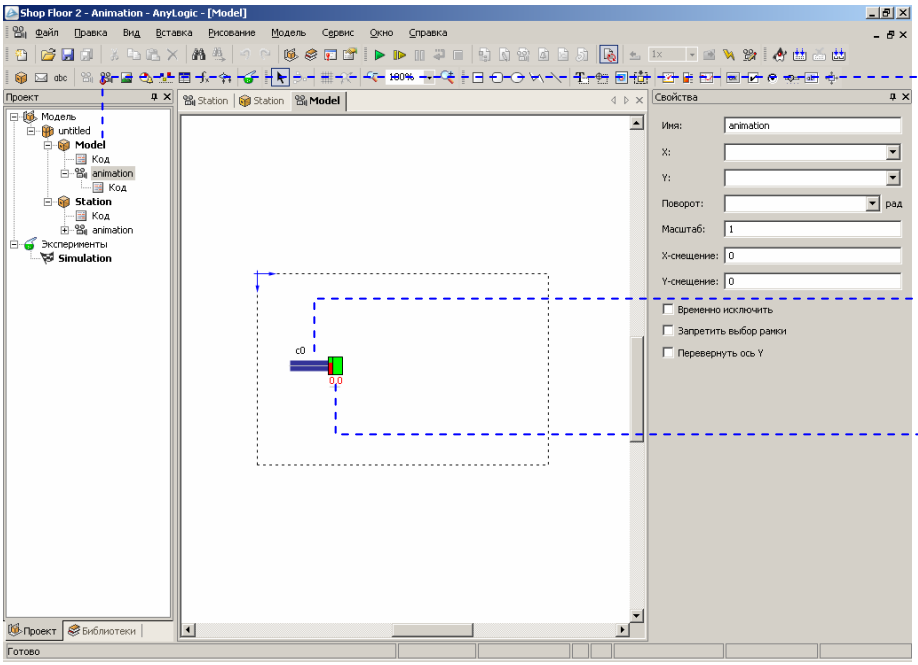


❷ Задайте следующее свойство объекта:



a Задайте фигуру анимации: `animation.queuePath`

► Создайте анимацию модели



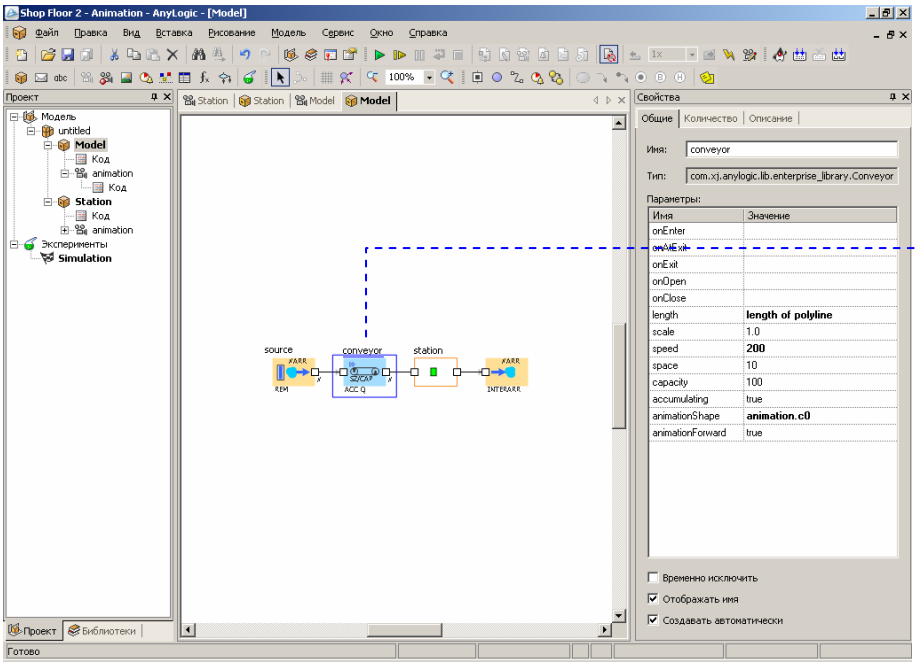
❶ Создайте новую анимацию

❷ Нарисуйте конвейер

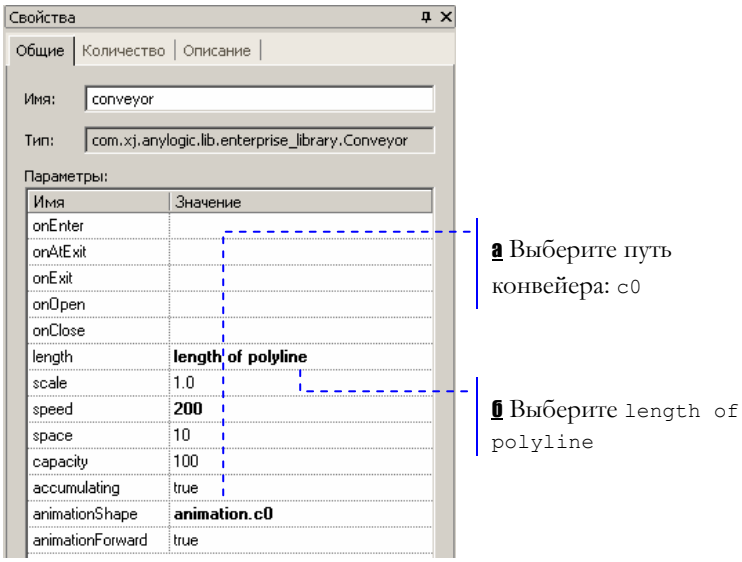
❸ Поместите анимацию станции сюда

❷ Нарисуйте конвейер толстой синей ломаной линией шириной 15 пикселов. Затем поверх синей нарисуйте ломаную линию белого цвета шириной 1; эта линия будет задавать путь движения деталей. Назовите тонкую ломаную линию `c0`.

► Задайте анимационные свойства объектов модели



❶ Задайте следующие свойства конвейера:



б Длина конвейера будет равна длине ломаной линии (в пикселах). Для преобразования длины из пикселей, например, в метры, может быть использован параметр `scale`.

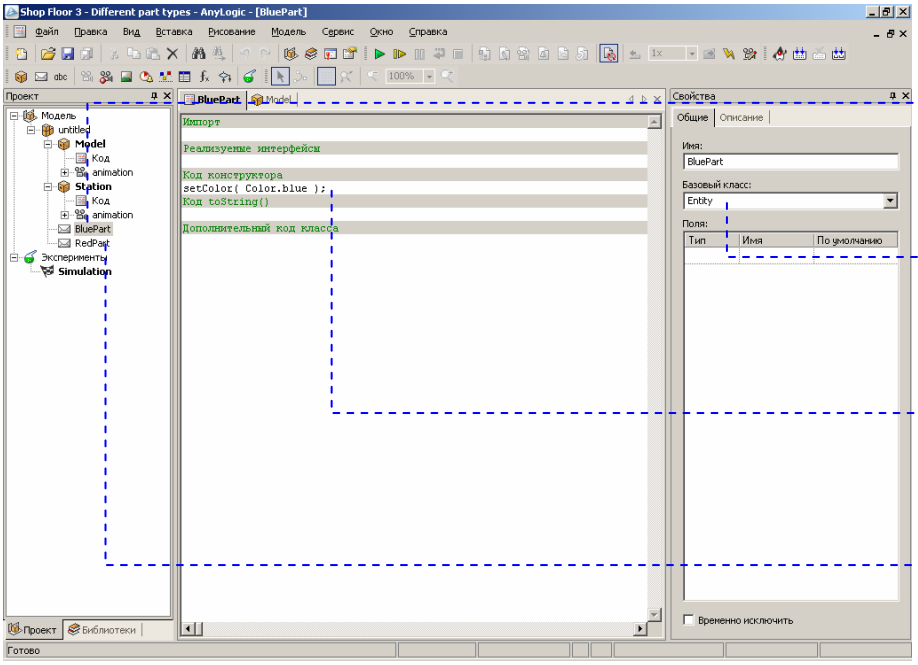
Запустите модель щелчком по кнопке *Запустить*  .

Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Shop Floor 2 - Animation.alp.

3.4 Создание различных типов деталей

Наше предприятие работает с деталями двух типов. Чтобы различить детали, мы создадим для каждого типа свой класс сообщения, с помощью которого зададим цвет, которым детали будут отображаться на анимации. Одни детали будут отображаться красным цветом, а другие – синим.

► Создайте классы сообщений



❶ Создайте класс сообщений `BluePart`

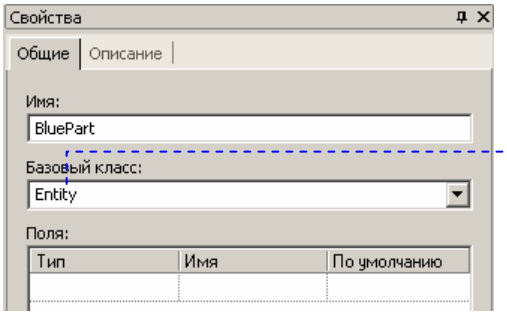
❷ Унаследуйте сообщение от класса Entity

❸ Задайте внешний вид детали

❹ Аналогично создайте еще один класс сообщения

❶ Чтобы создать новый класс сообщений, щелкните по кнопке панели инструментов *Новый класс сообщений*  . Назовите класс сообщений `BluePart`.

❷

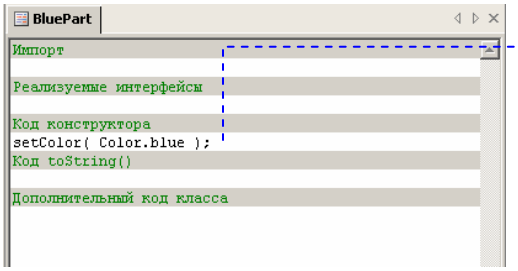


a Выберите `Entity` в качестве базового класса сообщения

a Класс `Entity` определен в Enterprise Library.

❸ С помощью кода инициализации задайте цвет, которым деталь

будет отображаться на анимации.

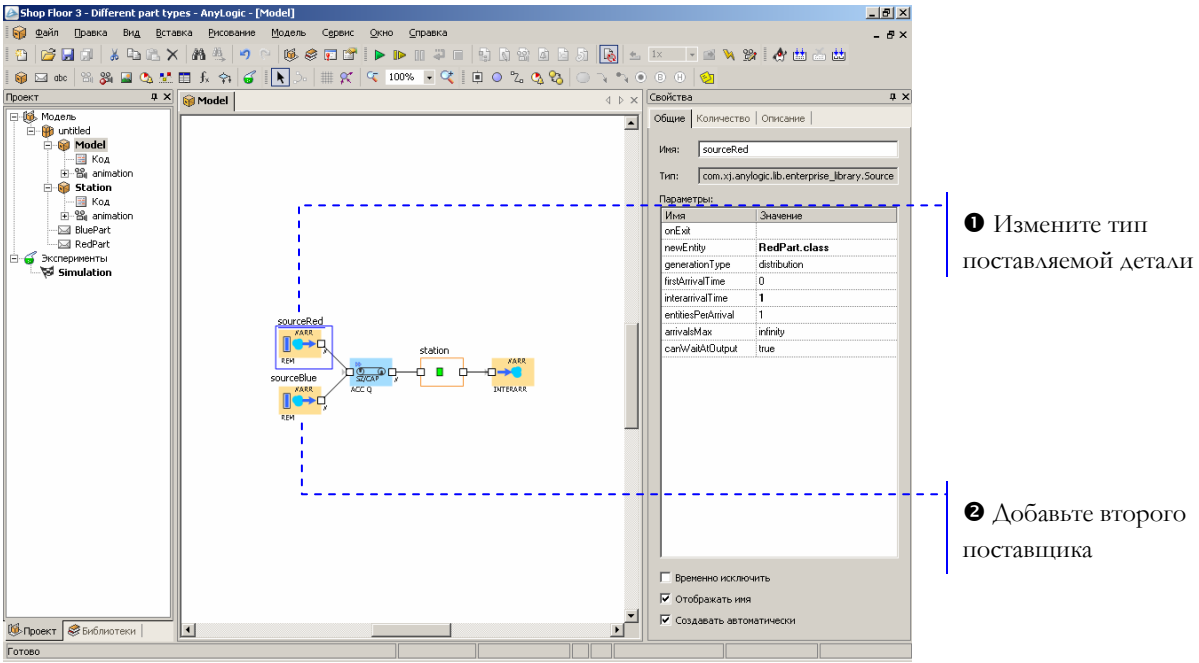


a Мы изменяем цвет с помощью функции `setColor()` класса `Entity`. За детальной информацией о функциях класса `Entity`, пожалуйста, обращайтесь к *Справочному руководству по Enterprise Library*.

❹ Аналогично создайте класс сообщений `RedPart`. Чтобы деталь этого типа отображалась на анимации красным цветом, напишите следующий *Код инициализации*:

```
setColor( Color.red );
```

► Создайте поставщиков деталей различных типов



❶ Выберите `RedPart` в свойстве `newEntity`. Оставьте интервал поставки деталей равным 1 минуте.

❷ Выберите `BluePart` в свойстве `newEntity`. Задайте интервал

поставки деталей равным 2 минутам.

Запустите модель щелчком по кнопке *Запустить* . Вы увидите, что по конвейерам движутся красные и синие детали.

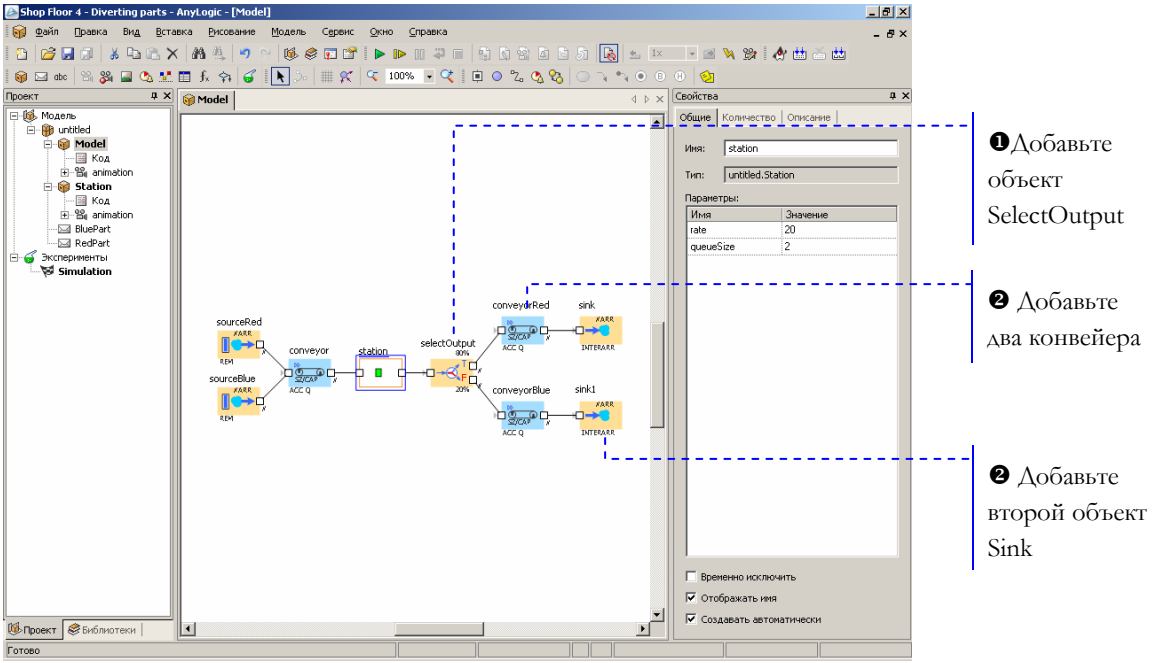
Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Shop Floor 3 - Different part types.alp.

3.5 Сортировка деталей

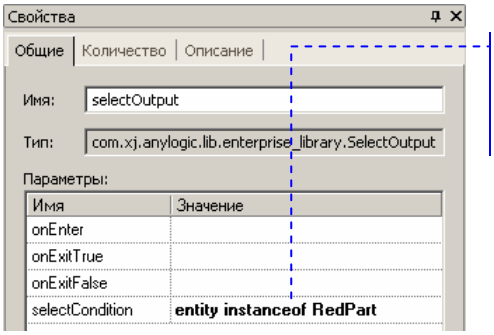
На сборочной станции производится сборка изделий из двух деталей: красной и синей. Мы должны организовать процесс сборки, отсортировав детали, поступающие на сборочную станцию.

Мы промоделируем то, как детали различных типов сортируются и направляются на различные конвейеры.

► **Измените блок-схему**



❶ Задайте следующее свойство объекта:



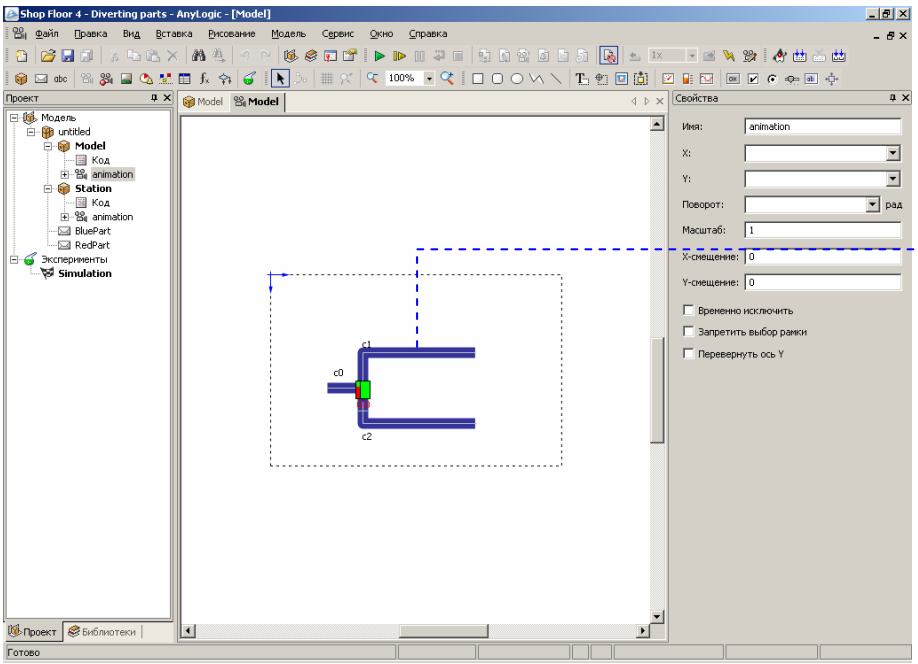
a Задайте условие выбора направления движения детали: `entity instanceof RedPart`

a Мы направляем красные детали на верхний конвейер, а синие – на нижний. Оператор `instanceof` определяет тип детали (она доступна как `entity`). Если условие равно `true`, то выбирается верхний конвейер, в противном случае - нижний.

❷ Назовите верхний конвейер `conveyorRed`, а нижний - `conveyorBlue`. Задайте скорости конвейеров 300.

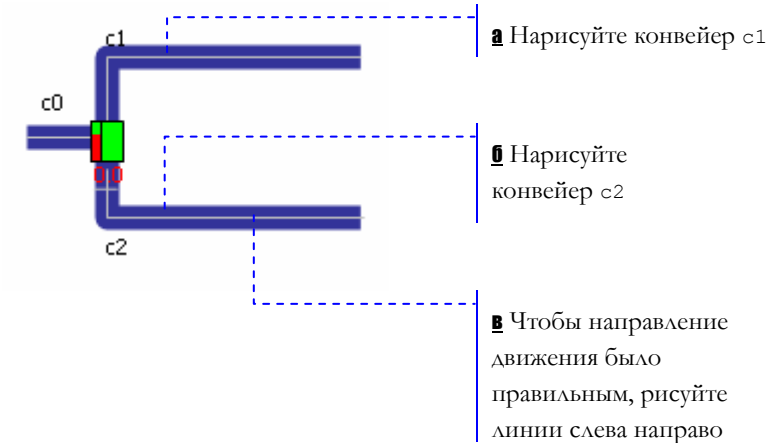
❷ Добавьте второй объект Sink. Оставьте принятые по умолчанию свойства объекта.

► Нарисуйте конвейеры

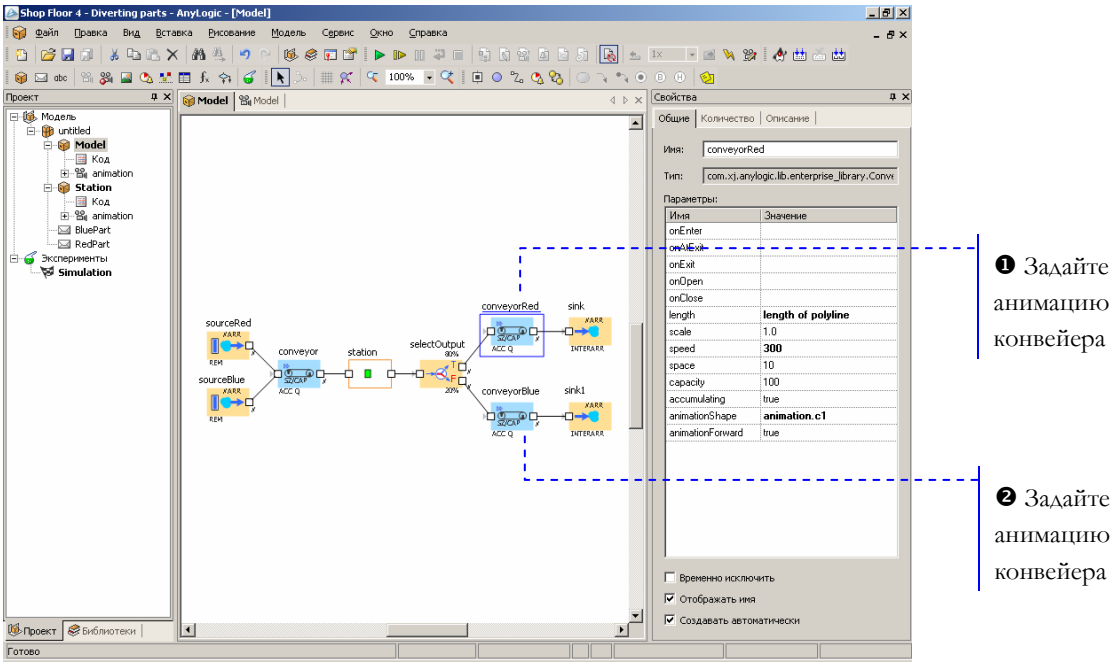


❶ Нарисуйте конвейеры

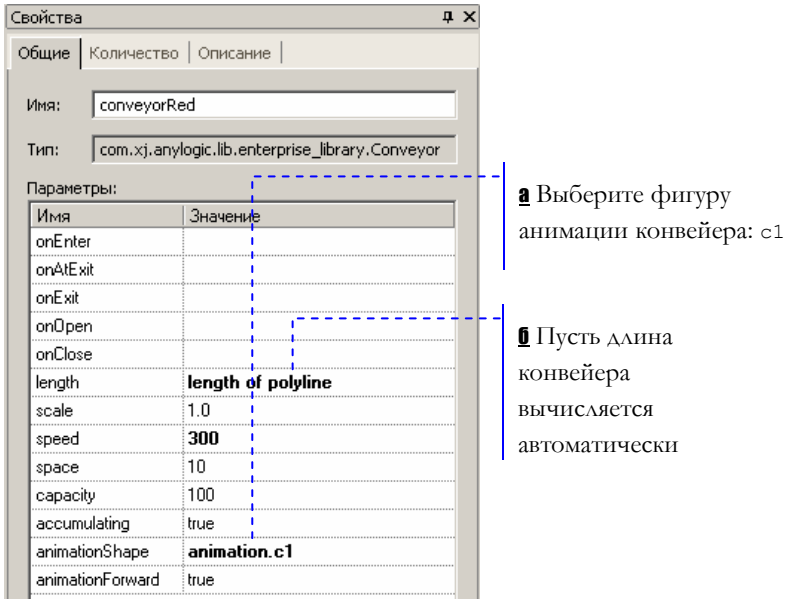
❷ Нарисуйте анимацию, как показано на рисунке ниже:



► Задайте анимационные свойства объектов



❶ Задайте следующие свойства объекта:

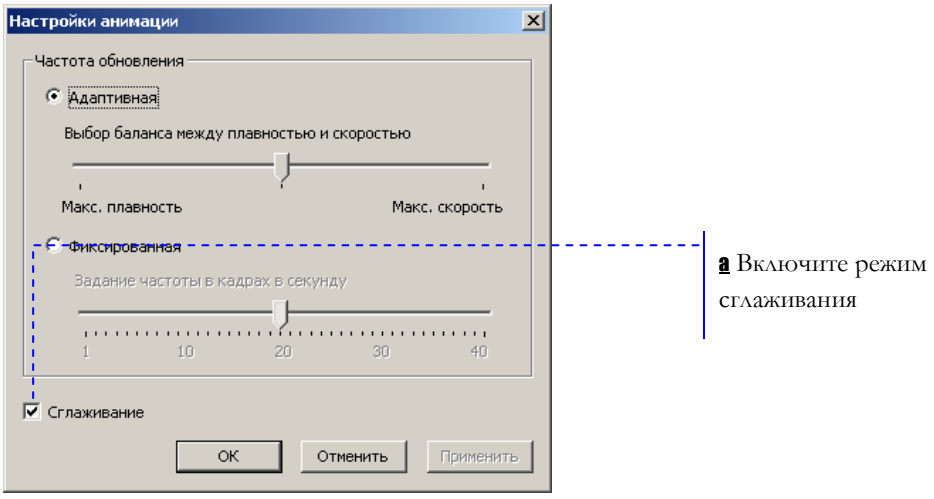


❷ Аналогично задайте анимацию для второго конвейера, за исключением того, что в качестве пути конвейера выберите линию `c2`.

Теперь запустите модель щелчком по кнопке *Запустить*  .

Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Shop Floor 4 - Diverting parts.alp.

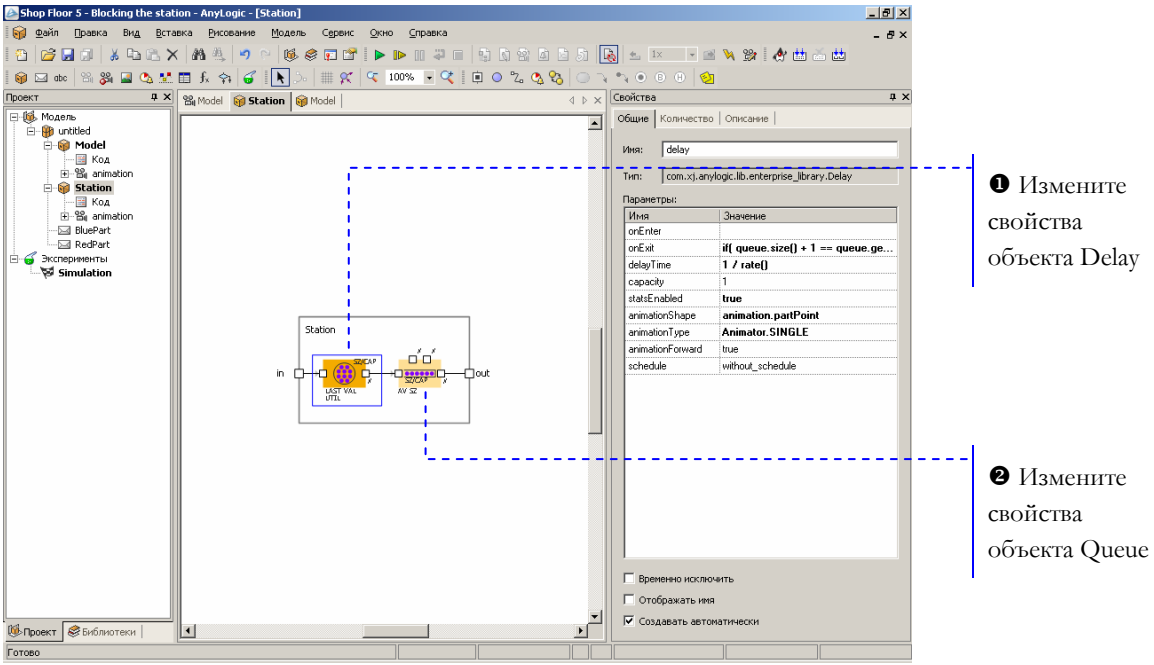
Если Вы хотите, чтобы анимация выглядела лучше, то можете включить режим сглаживания, отключенный по умолчанию для повышения скорости выполнения модели. Щелкните мышью по кнопке панели инструментов *Настройки анимации…*  , и в появившемся диалоговом окне установите флажок *Сглаживание*.



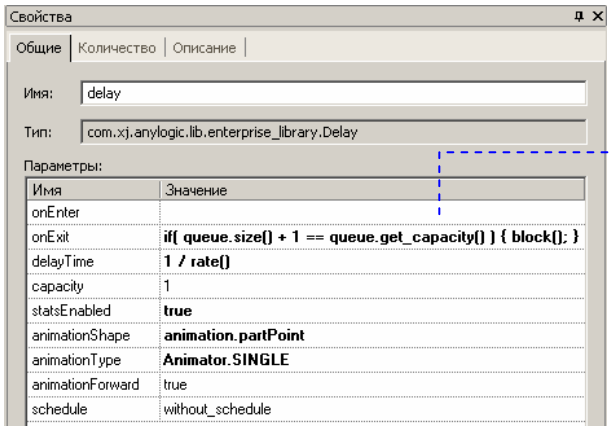
3.6 Блокировка станции

Если буфер деталей переполнится, то станцию нужно будет заблокировать, прекратив прием новых деталей.

► **Добавьте возможность блокировки станции**



❶ В случае переполнения буфера заблокируйте объект:



a Введите следующий код в свойстве объекта *onExit*:

```
if(queue.size()+1 == queue.get_capacity())  block();
```

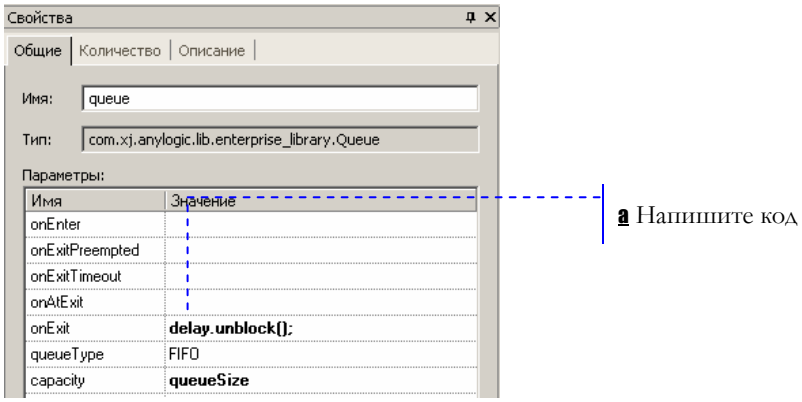
Код будет выполняться после того, как будет обработана

очередная деталь. Степень заполнения буфера (мы получаем это значение с помощью функции `queue.size()`) сравнивается с его вместимостью (`get_capacity()`). Чтобы учесть только что обработанную деталь, к полученному результату добавляется единица.

➲ Вы можете узнать текущее значение свойства объекта и изменить его с помощью функций `get_x()` и `set_x()`, где `x` является именем свойства.

Объект Delay будет заблокирован с помощью функции `block()`. Детали не будут приниматься на обработку, пока не будет вызвана функция `unblock()`.

❷ Разблокируйте станцию, как только деталь будет забрана из буфера.



a Функция `unblock()` вызывается даже тогда, когда объект Delay не заблокирован. Если же Вы хотите узнать, заблокирован ли объект, вызовите функцию `blocked()`.

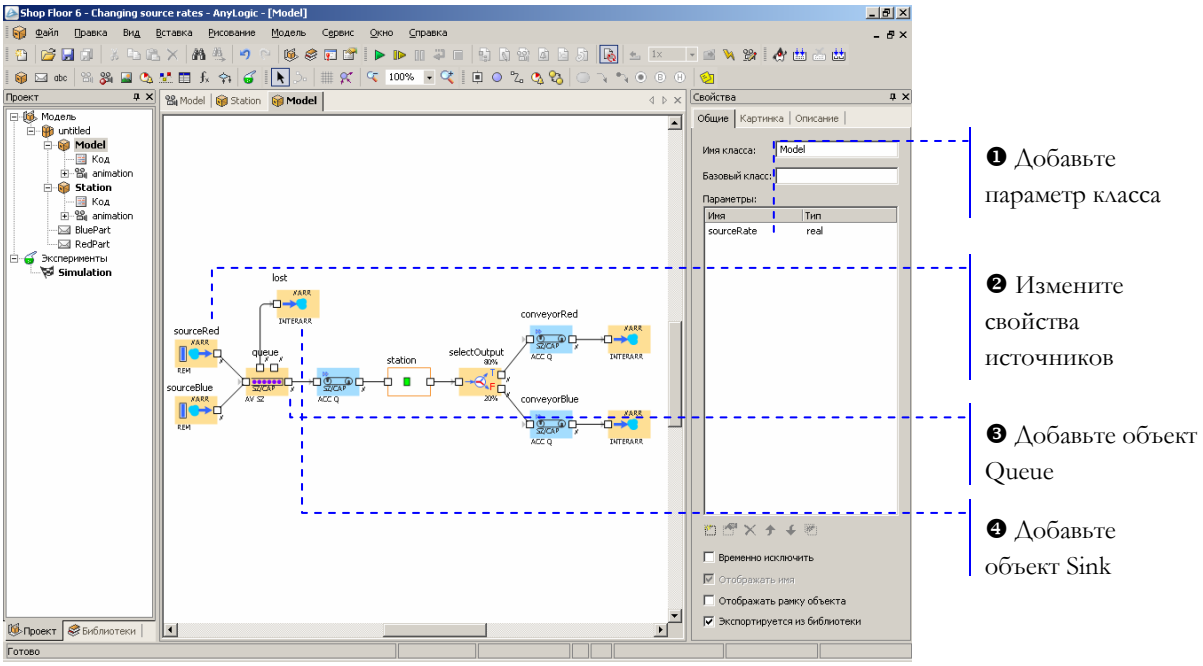
Запустите модель щелчком по кнопке *Запустить*  .

➘ Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Shop Floor 5 - Blocking the station.alp.

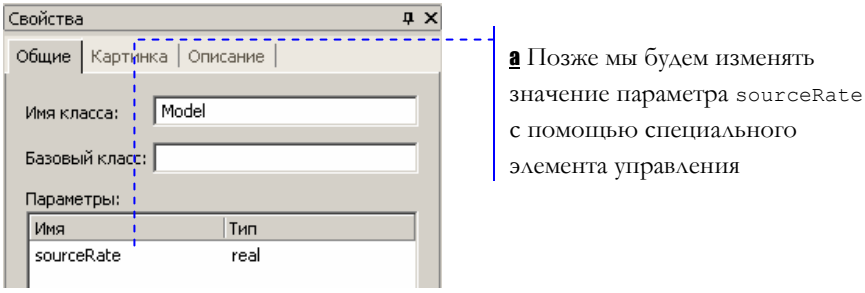
3.7 Изменение интенсивности поставки деталей

Теперь мы хотим добавить возможность изменения интенсивности поставки деталей.

► Задайте интенсивность поставки

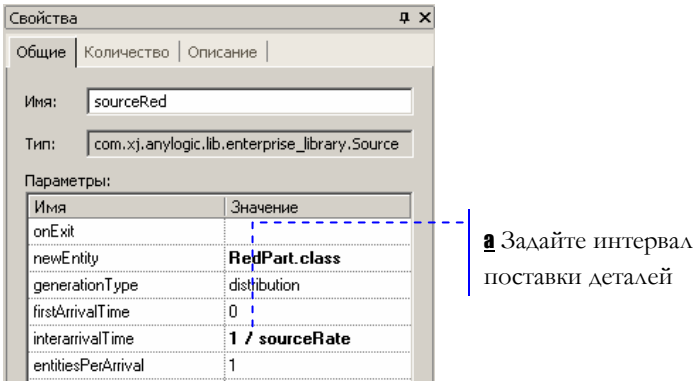


❶ Создайте параметр, задающий интенсивность поставки деталей. Щелкните мышью по элементу *Model* в дереве проекта и создайте параметр со следующими свойствами:



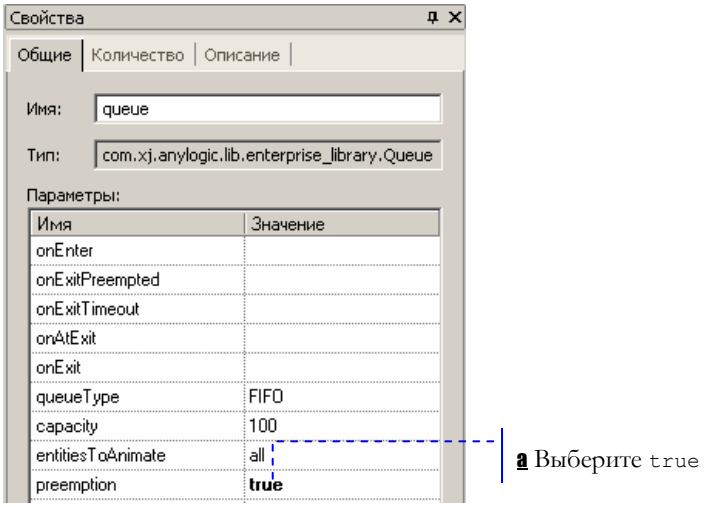
a Назовите параметр `sourceRate`, выберите тип `real`, и задайте значение 10.

❷ Задайте следующее свойство у обоих объектов Source:



❸ Мы добавим буфер для хранения деталей, которые не смогут быть помещены на конвейер.

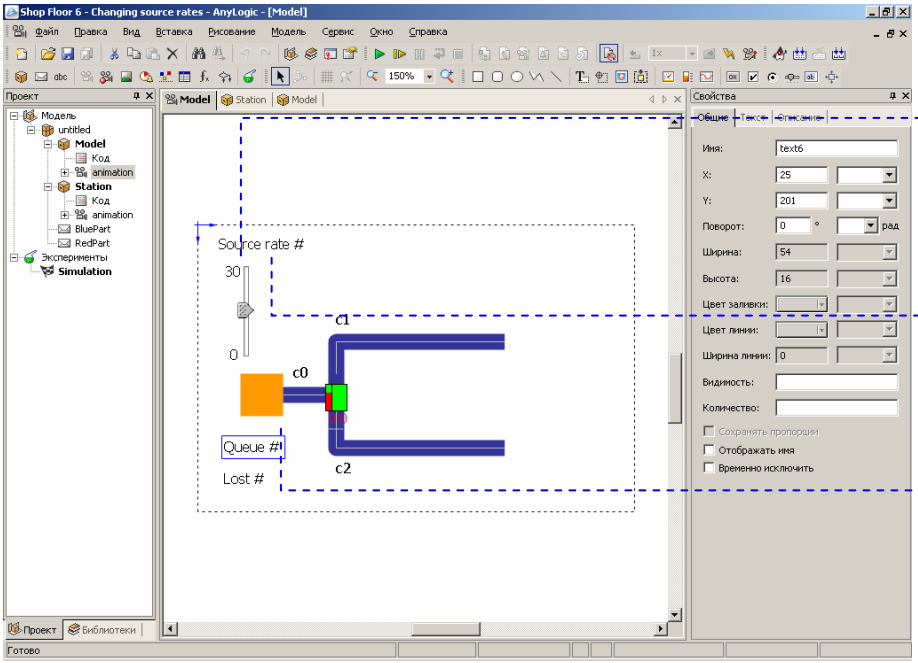
Задайте следующие свойства объекта:



a Мы включаем режим вытеснения старых деталей новыми. За детальной информацией о режиме *preemption*, пожалуйста, обращайтесь к *Справочному руководству по Enterprise Library.*

❹ Добавьте объект Sink для сбора статистики о потерянных деталях. Назовите объект `lost` и соедините его порт с левым верхним портом объекта Queue.

► Добавьте на анимацию элементы управления

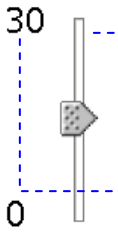


❶ Добавьте ползунок для изменения интенсивности

❷ Покажите текущее значение интенсивности

❸ Нарисуйте буфер и добавьте метки размера буфера и числа потерянных деталей

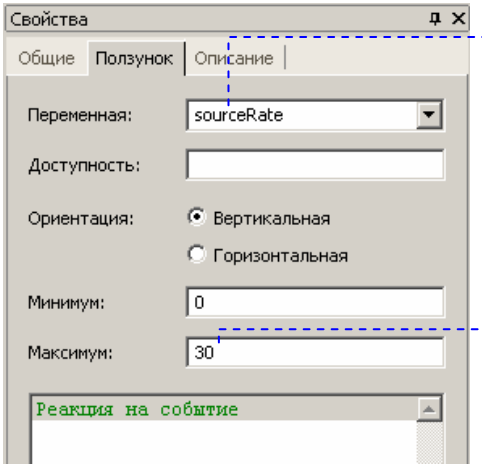
❶



a Поместите ползунок

б Поместите метки границ диапазона ползунка

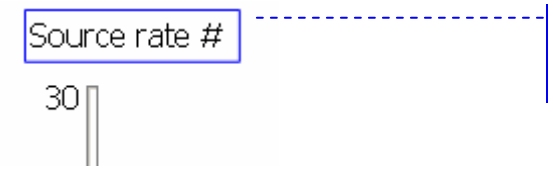
Задайте следующие свойства созданного ползунка:



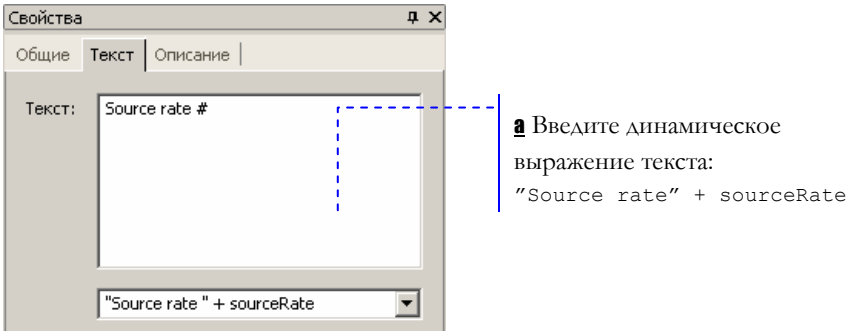
a Выберите параметр `sourceRate`

б Задайте пределы диапазона значений ползунка

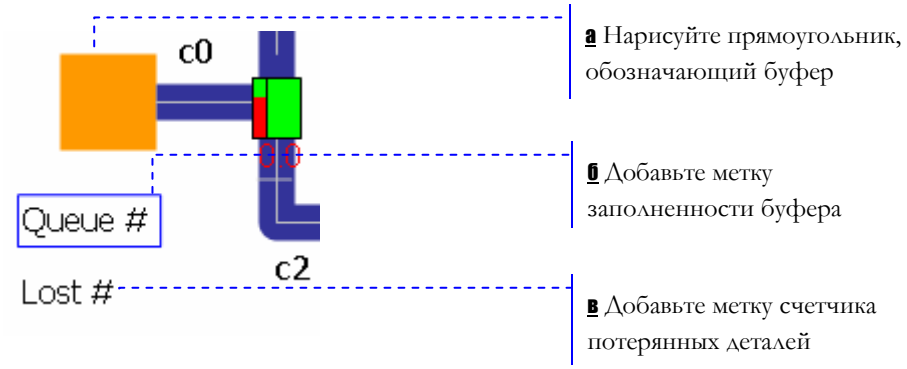
❷



Чтобы показать значение интенсивности поставки деталей, задайте следующий динамический текст:

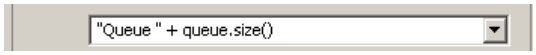


❸ Добавьте метки, отображающие текущую заполненность буфера и количество потерянных деталей:

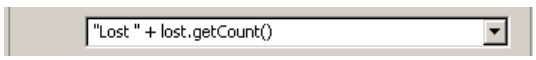


а Назовите прямоугольник `queueArea`. Выберите цвет заливки и стиль линии на Ваше усмотрение, например, выключите отображение линий, и задайте оранжевый цвет заливки.

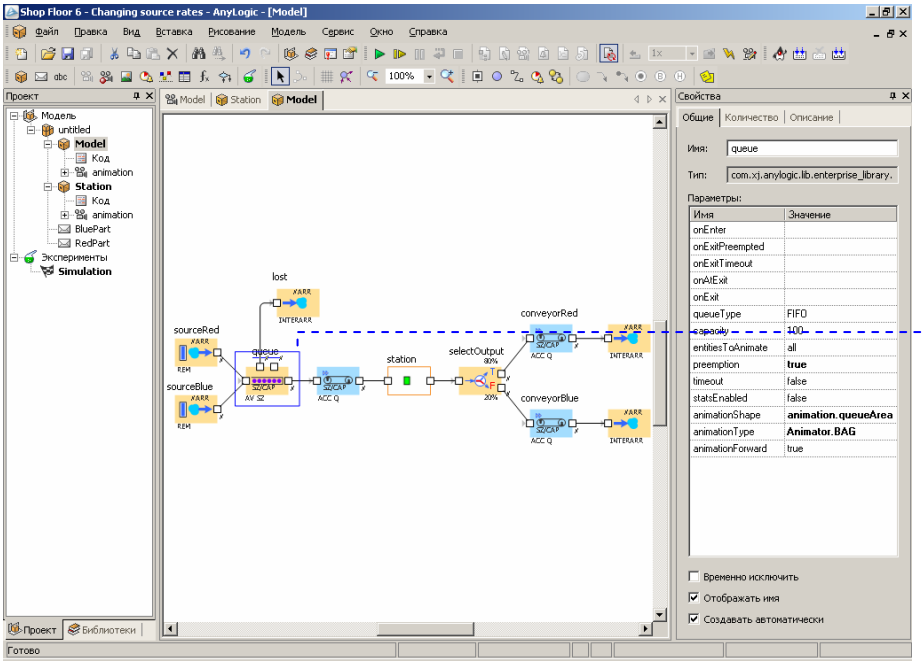
б Задайте следующий динамический текст:



в Задайте следующий динамический текст:

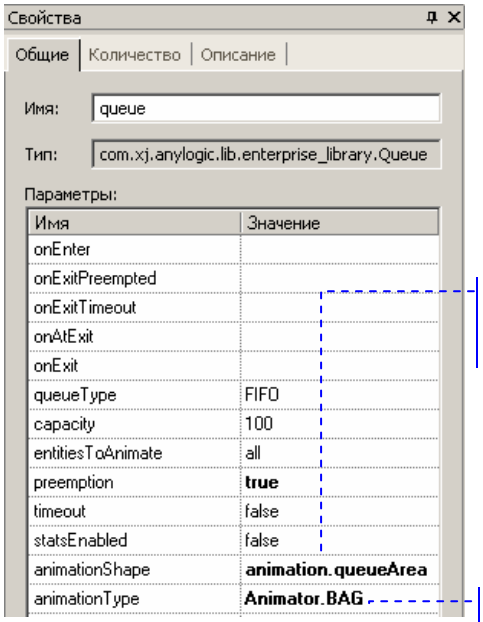


► Задайте анимационные свойства объектов блок-схемы



❶ Задайте свойства объекта `Queue`

❶ Задайте следующие свойства объекта:



a Задайте анимационную фигуру

б Задайте анимационный стиль

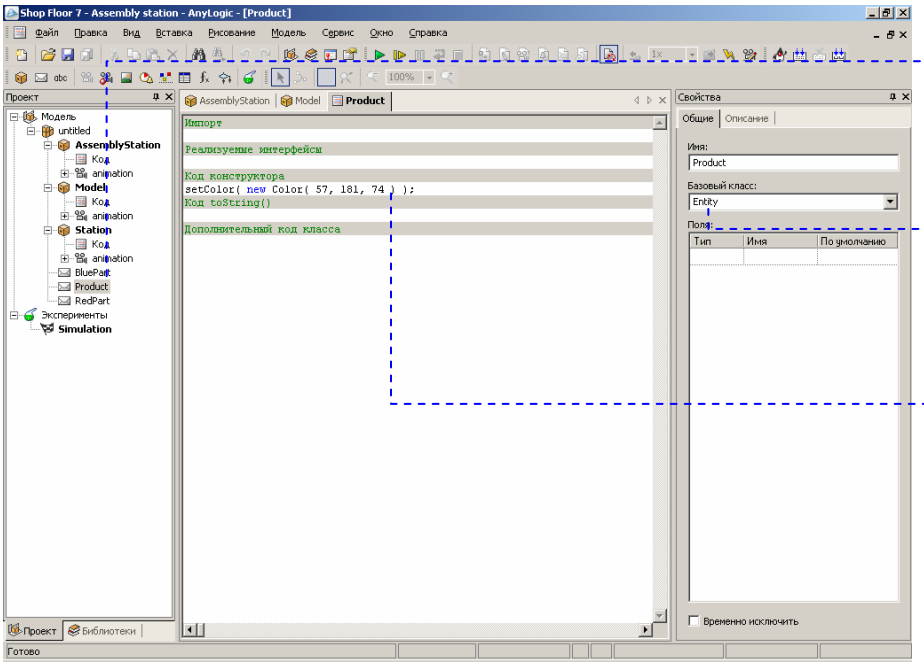
Запустите модель и изучите ее поведение при различных интенсивностях поставки деталей, которую Вы теперь можете изменять с помощью созданного ползунка.

Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Shop Floor 6 - Changing source rates.alp.

3.8 Добавление сборочной станции

Теперь мы добавим в нашу модель сборочную станцию. Но вначале мы создадим класс сообщения `Product`, с помощью которого будем представлять изделия, изготавливаемые нашим предприятием.

► **Создайте класс сообщения `Product`**

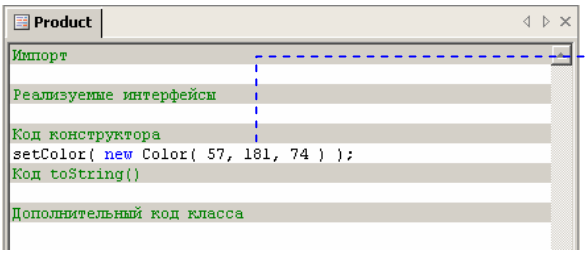


❶ Создайте класс сообщения Product

❷ Унаследуйте класс от класса `Entity`

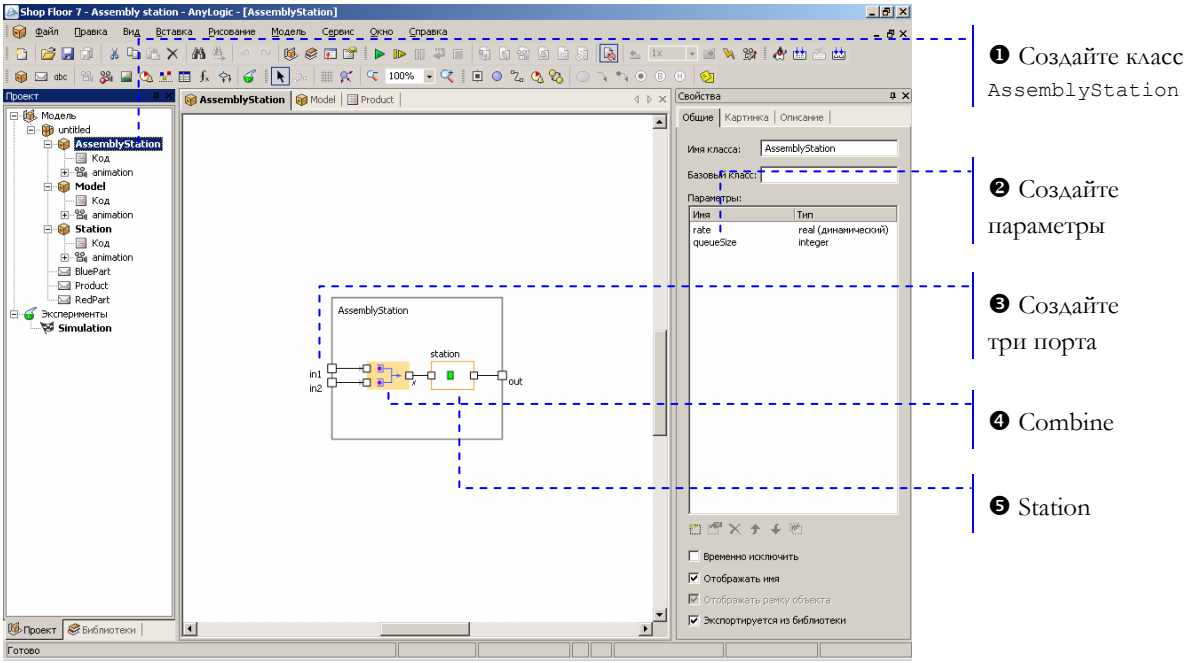
❸ Измените внешний вид изделия

❸ Задайте цвет, которым будут отображаться изделия на анимации:

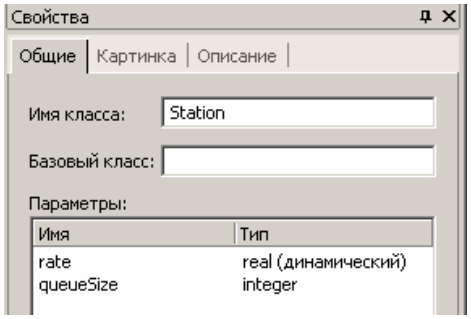


a Задайте темно-зеленый цвет с красной компонентой 57, зеленой - 181 и синей - 74

► Создайте класс сборочной станции



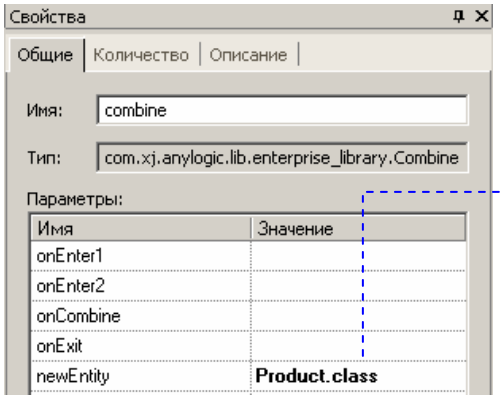
❷ Создайте *Динамический* параметр `rate` типа `real` и *Простой* параметр `queueSize` типа `integer`.



Задайте значения параметров: 20 для `rate`, и 2 – для `queueSize`.

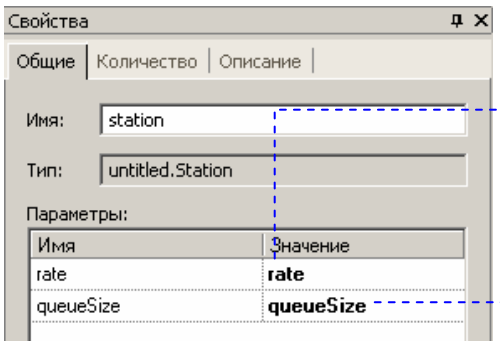
❸ Можете назвать порты `in1`, `in2` и `out`. Оставьте заданные по умолчанию свойства портов.

❹ Задайте следующие свойства объекта:



a На станции будет собираться изделие (сообщение класса `Product`)

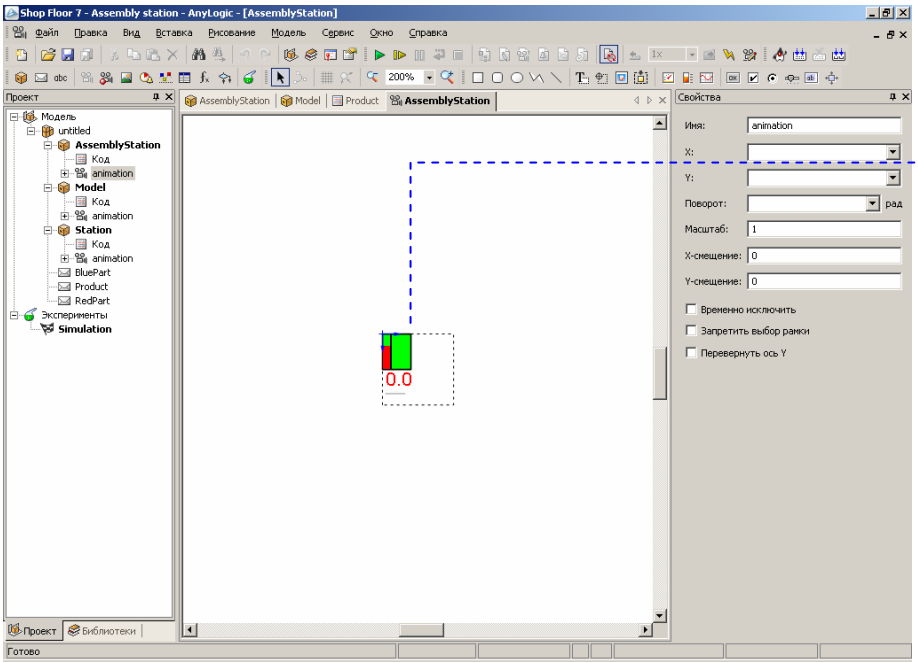
❻ Задайте следующие свойства объекта:



a Скорость работы станции задается параметром класса сборочной станции

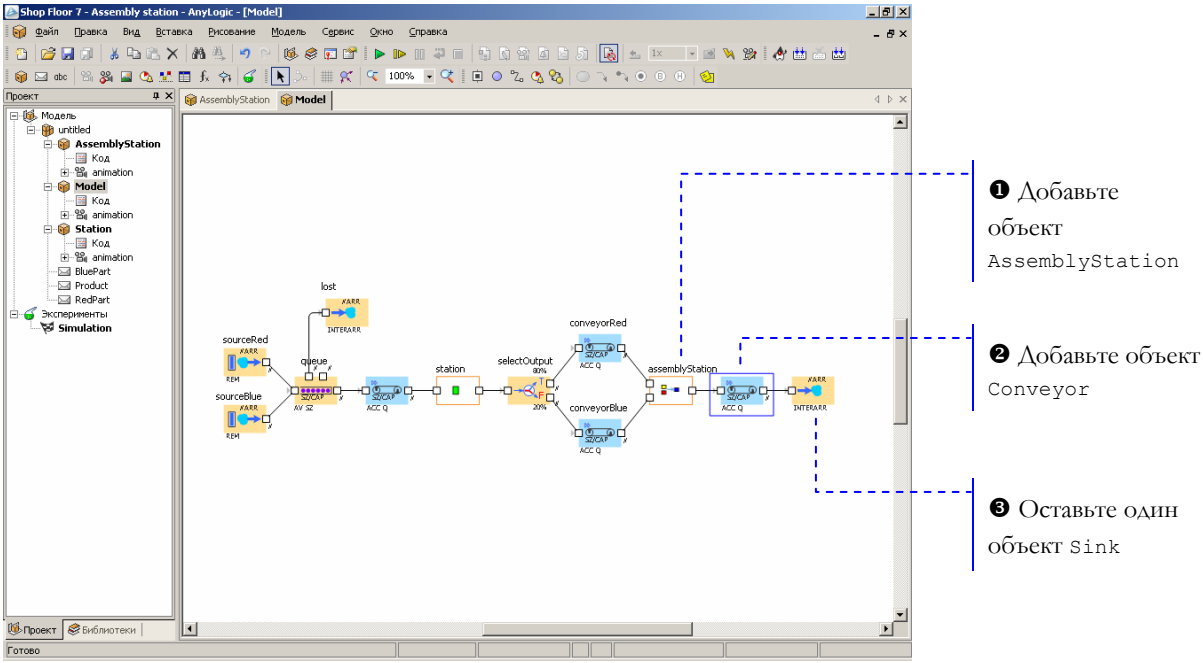
б Размер буфера также задается параметром класса

► Измените анимацию класса `AssemblyStation`



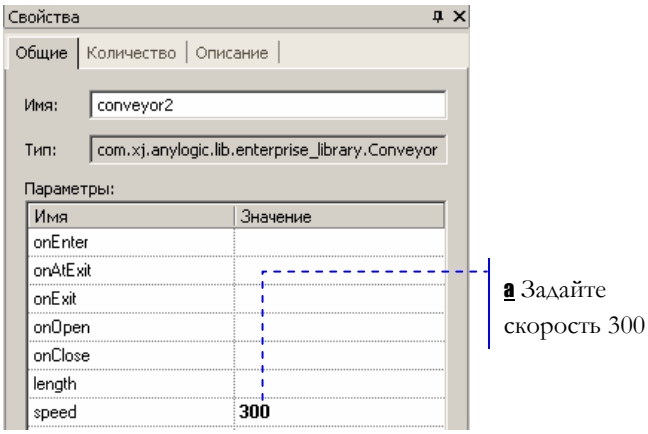
❶ Поместите анимацию станции в анимационную область

► Измените блок-схему



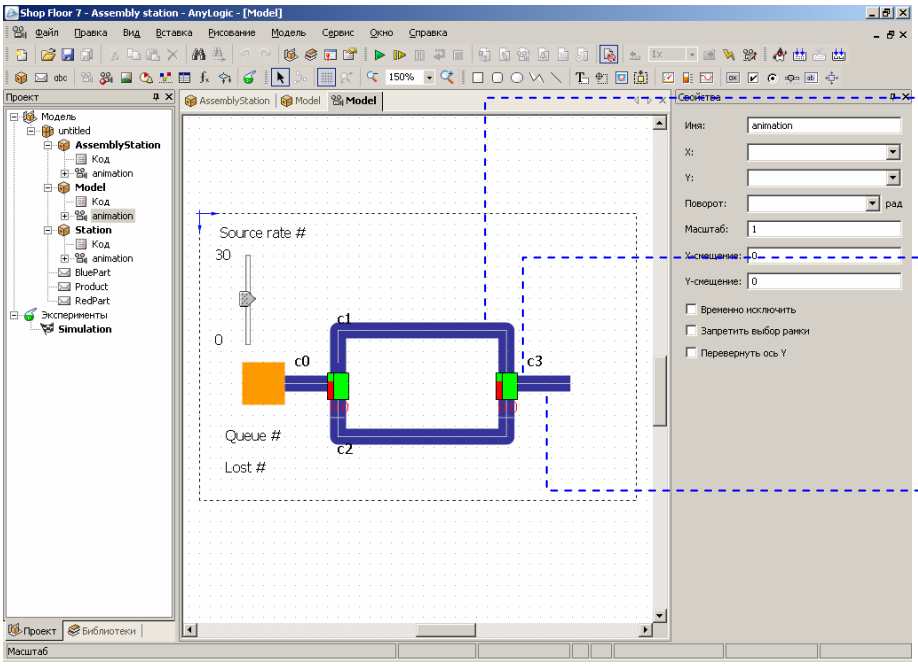
❶ Добавьте на блок-схему объект сборочной станции. Оставьте заданные по умолчанию свойства.

❷ Задайте следующие свойства объекта:



❸ Оставьте только один объект Sink. Пусть он называется `sink`.

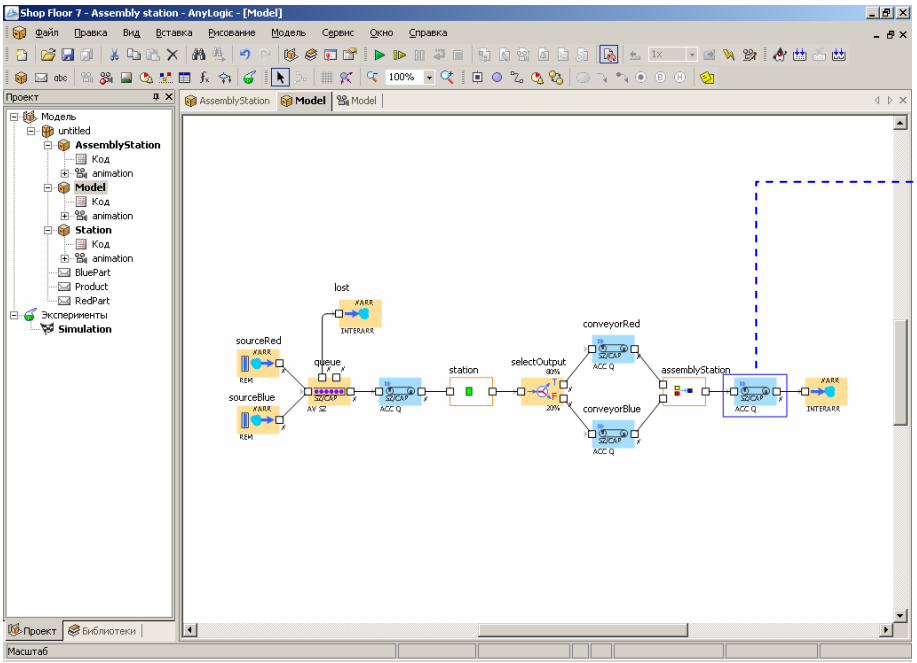
► Нарисуйте анимацию



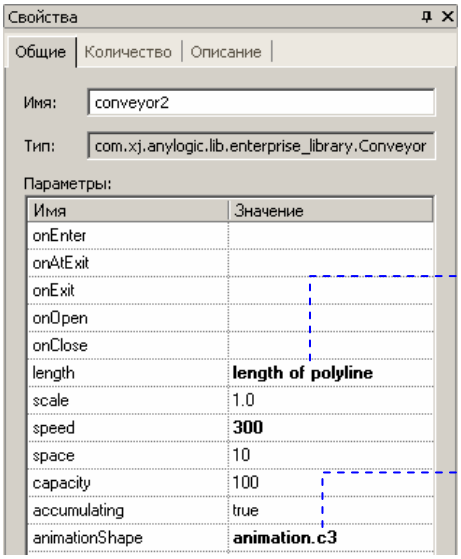
❶ Сделайте так, чтобы конвейеры `c1` и `c2` вели к сборочной станции.

❸ Нарисуйте конвейер `c3`, ведущий от сборочной станции.

► Задайте анимационные свойства объектов блок-схемы



❶ Задайте следующие свойства объекта:



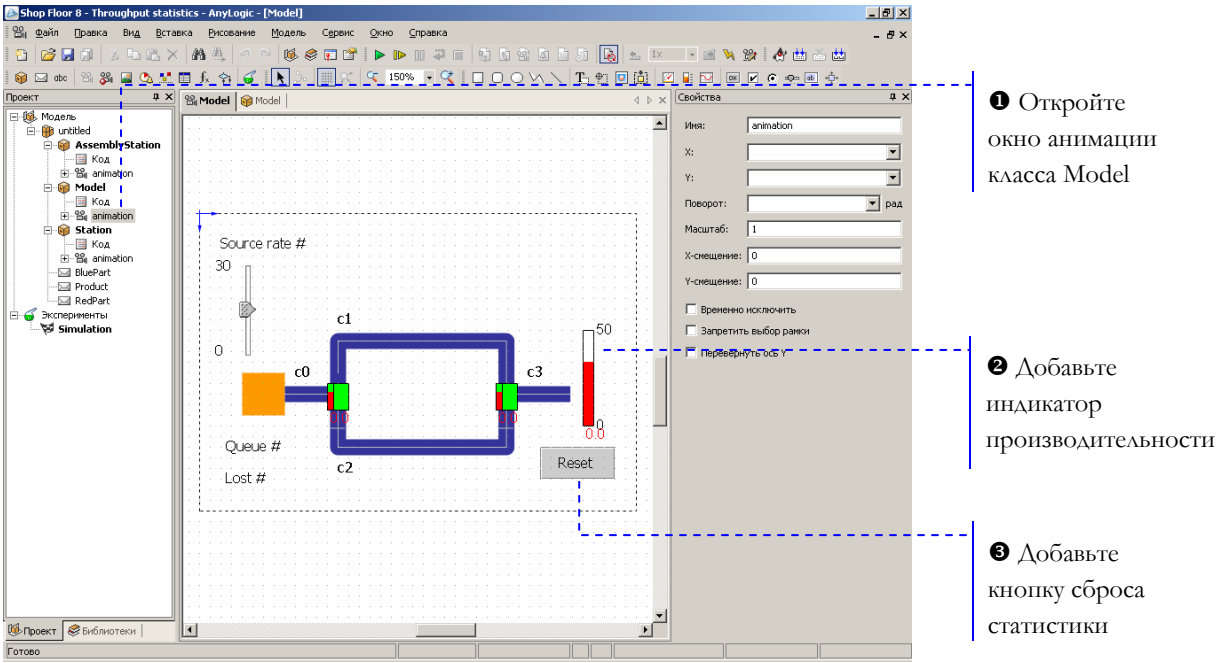
Мы закончили создание нашей модели. Запустите модель, щелкнув мышью по кнопке *Запустить* .

Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Shop Floor 7 - Assembly station.alp.

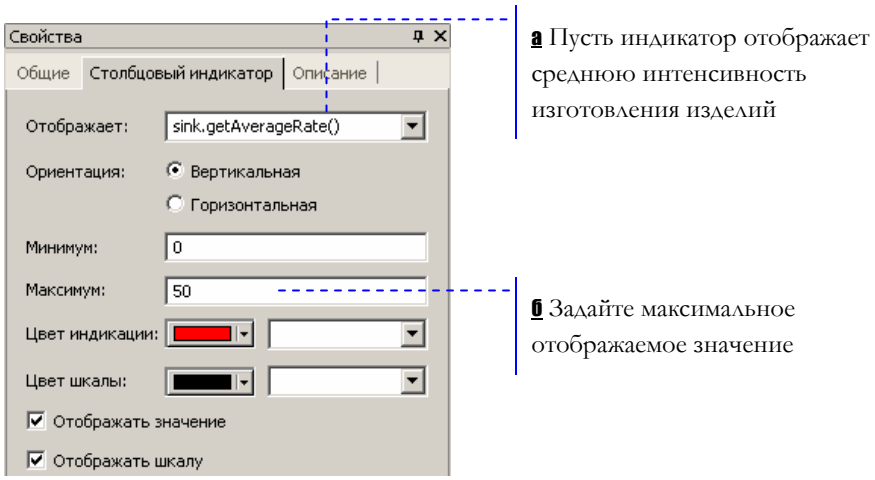
3.9 Сбор статистики производительности

Теперь мы хотим узнать, насколько эффективно работает наше предприятие. Это можно легко сделать с помощью статистики производительности, автоматически собираемой в AnyLogic.

► **Покажите статистику производительности**

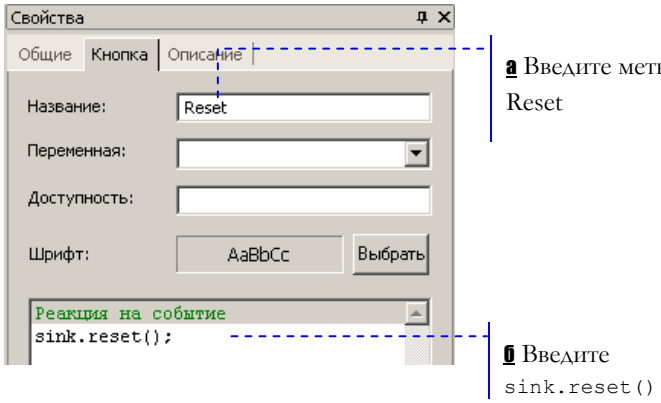


❷ Задайте следующие свойства индикатора:



а Введите `sink.getAverageRate()`.

❸ Измените размер кнопки и задайте следующие свойства:



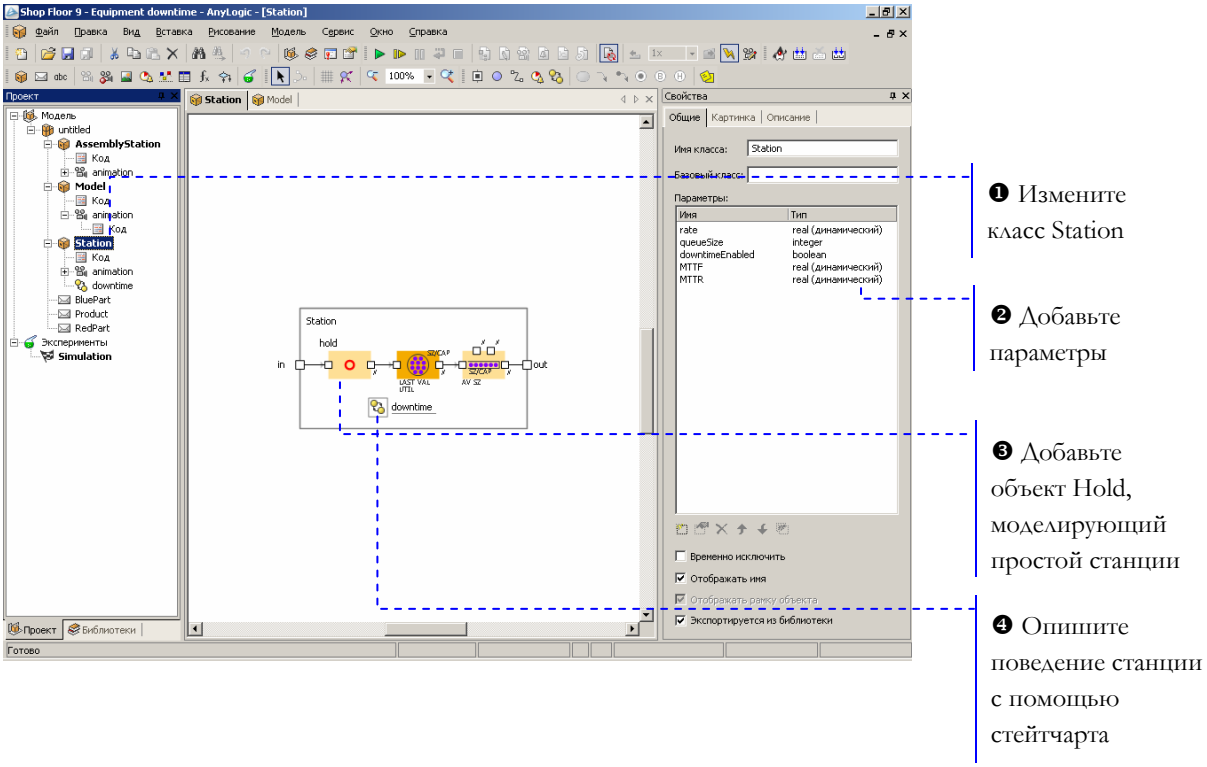
Запустите модель и изучите статистику производительности системы.

➘ Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Shop Floor 8 - Throughput statistics.alp.

3.10 Моделирование выхода оборудования из строя

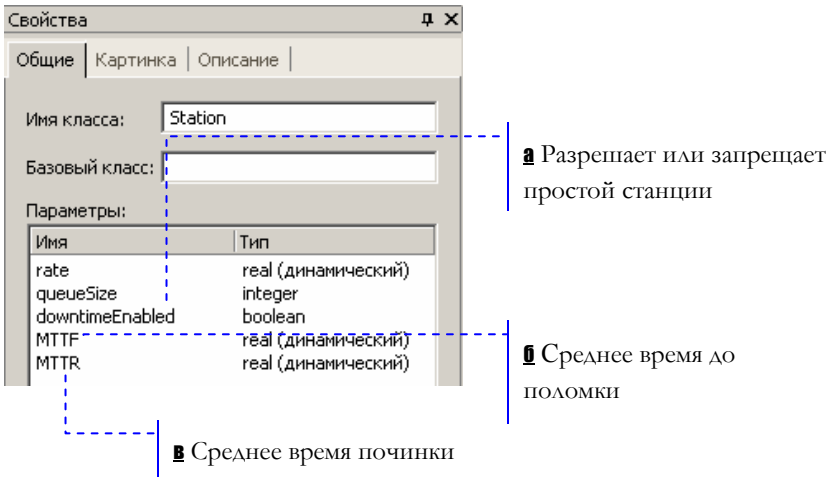
На данный момент модель не учитывает возможность выхода станции из строя. А ведь простои, связанные с починкой вышедшего из строя оборудования, оказывают значительное влияние на производительность предприятия. Сейчас мы сделаем нашу модель более реалистичной, исправив эту некорректность.

► **Промоделируйте выход станции из строя**



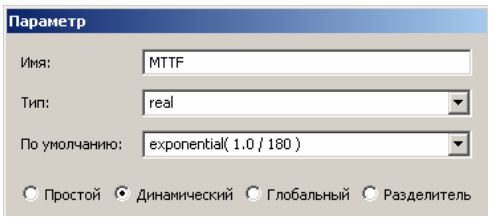
❶ Откройте структурную диаграмму класса Station, сделав двойной щелчок мышью по элементу дерева.

❷ Создайте параметры станции, задающие среднее время поломки (MTTF) и среднее время починки (MTTR).

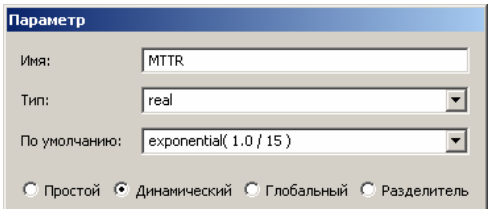


а Создайте параметр `downtimeEnabled` типа `boolean`, задающий возможность простоя станции. Пусть возможность выхода оборудования из строя допускается изначально (задайте значение `true`).

б Создайте *Динамический* параметр `MTTF` типа `real`. В нашем примере время до поломки будет распределено экспоненциально со средним значением, равным трем часам (180 минут). В качестве аргумента функции `exponential()` задается интенсивность, поэтому мы задаем обратное значение. Заметьте, что мы пишем 1.0, а не 1, потому что результат выражения имеет тот же тип, что и операнды, и если бы мы написали 1/180, то результат был бы равен 0.



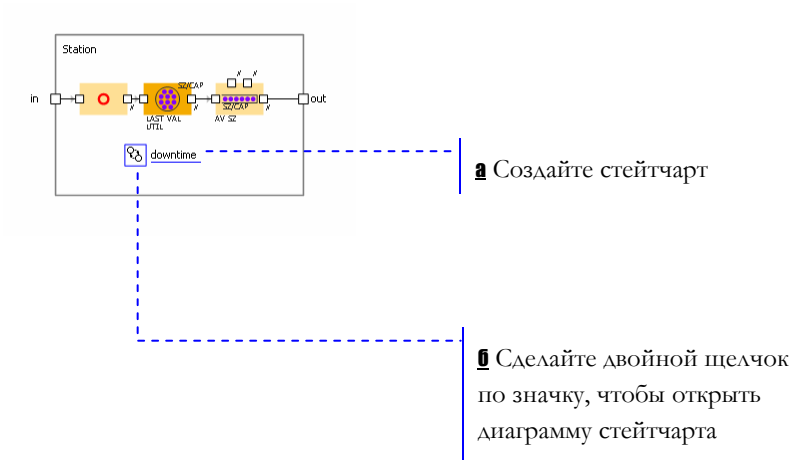
в Создайте *Динамический* параметр `MTTR` типа `real`, задающий время починки станции.



В среднем на починку станции будет уходить 15 минут.

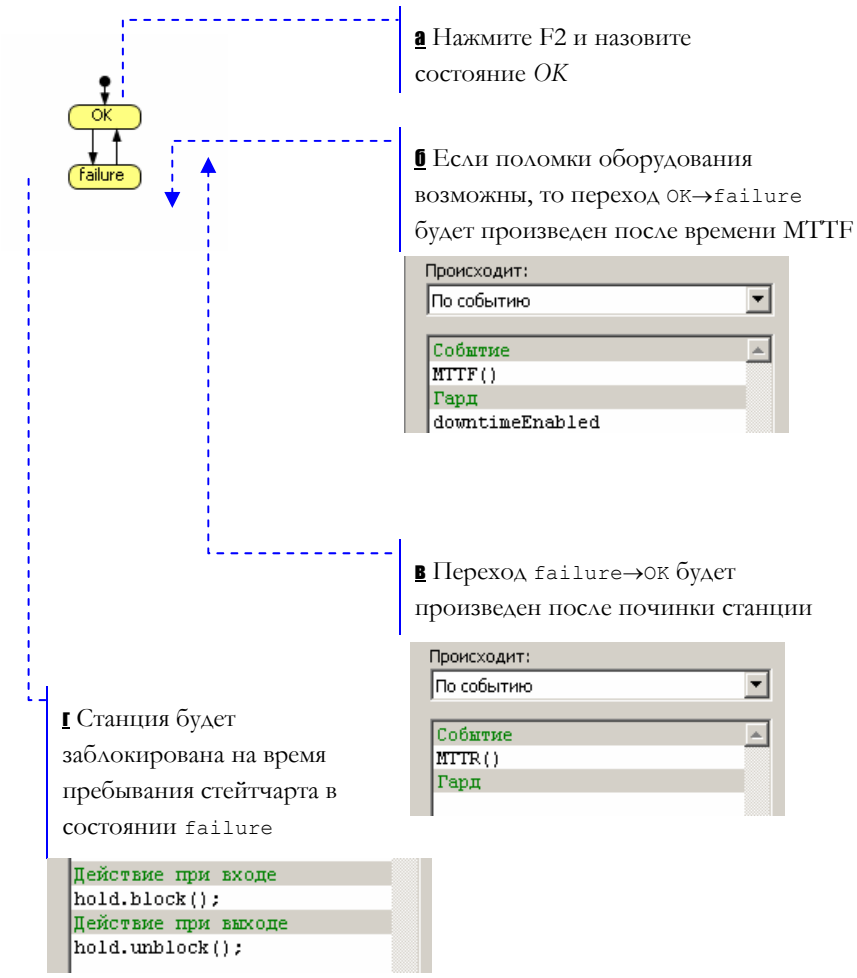
❸ Объект Hold будет блокировать станцию на время поломки. Пусть объект называется `hold`.

❹ Задайте поведение станции визуально с помощью стейтчарта (диаграммы состояний):



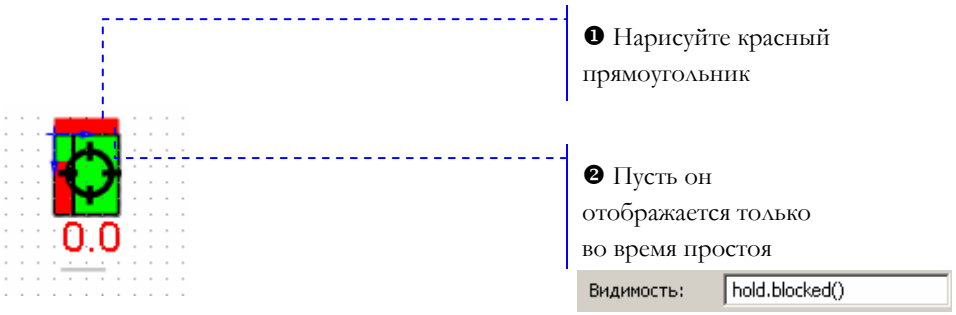
а Щелкните мышью по кнопке *Стейтчарт*  , а затем по диаграмме. На диаграмме появится значок стейтчарта.

С помощью кнопок панели инструментов *Стейтчарт* нарисуйте на диаграмме стейтчарт, показанный на рисунке ниже:

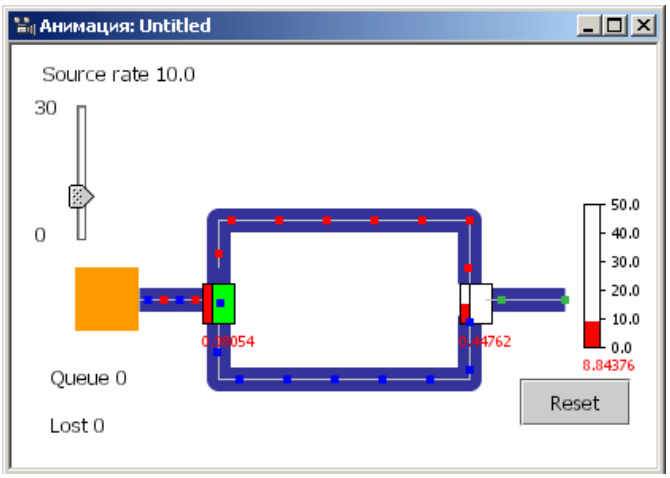


Мы промоделировали возможность поломки оборудования предприятия. Чтобы было легче определить моменты выхода станции из строя, мы нарисуем на анимации специальный индикатор.

► Нарисуйте индикатор простоя станции



Запустите модель щелчком по кнопке *Запустить* . С помощью анимации Вы можете изменять характеристики Вашей системы, и отслеживать то, как это влияет на поведение модели.



Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Shop Floor 9 - Equipment downtime.alp.

4. Модель отделения офтальмологии

В этом разделе учебного пособия мы создадим модель больничного отделения, в котором проводится процедура офтальмоскопии. Пациент, прибывающий в отделение, вначале проходит регистрацию в приемном покое. Затем он направляется для проведения процедуры в указанную процедурную комнату. Если все процедурные комнаты оказываются занятыми, то пациент ждет в приемном покое, пока какая-нибудь из комнат не освободится. Только тогда медсестра отводит пациента в освободившуюся комнату и вызывает туда офтальмолога. Врач осматривает пациента с помощью офтальмоскопа, который он специально приносит из комнаты хранения инструментов. После проведения процедуры врач относит офтальмоскоп обратно и отправляется в ординаторскую, а пациент покидает отделение офтальмологии.

Эту модель мы создадим с помощью объектов Enterprise Library для моделирования транспортных сетей. Эта библиотека позволяет легко создавать транспортные модели, обладающие сложной структурой. В транспортных моделях используется усовершенствованный механизм разделения ресурсов.

4.1 Создание нового проекта

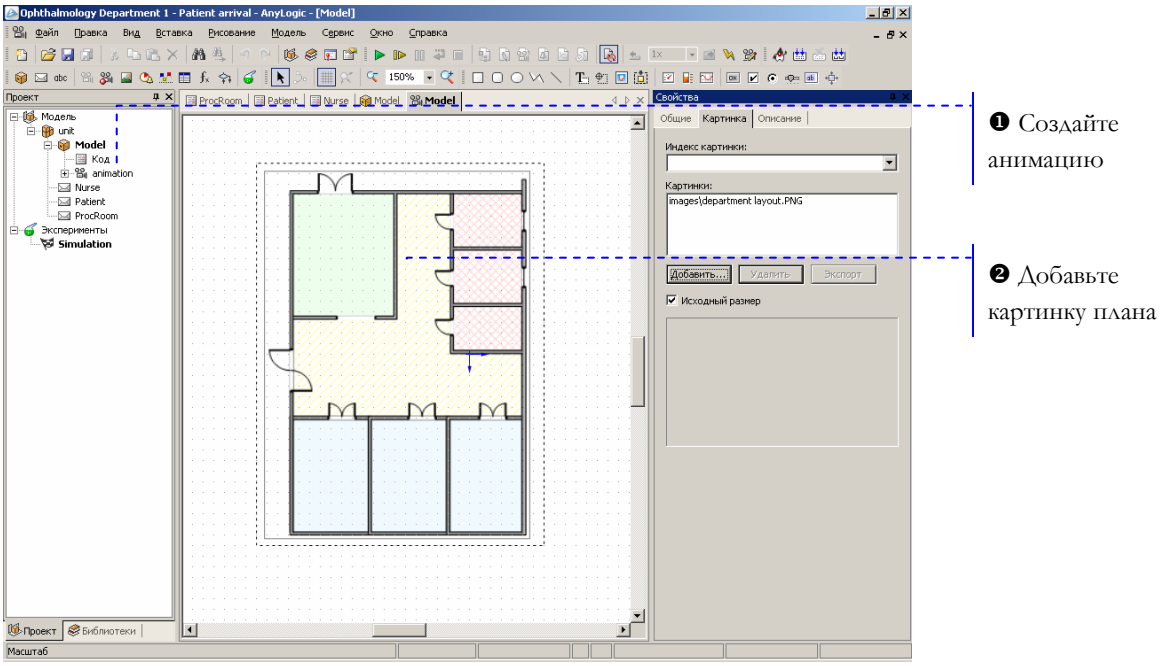
Создайте новый проект, как описано в разделе 1.1, “Как создать модель”. Переименуйте класс `Main` в `Model`. Задайте режим реального времени с выполнением одной единицы модельного времени в одну секунду.

4.2 Создание анимации

Прежде всего, нам нужно создать анимацию, поскольку именно она будет определять структуру нашей модели – ее транспортную сеть.

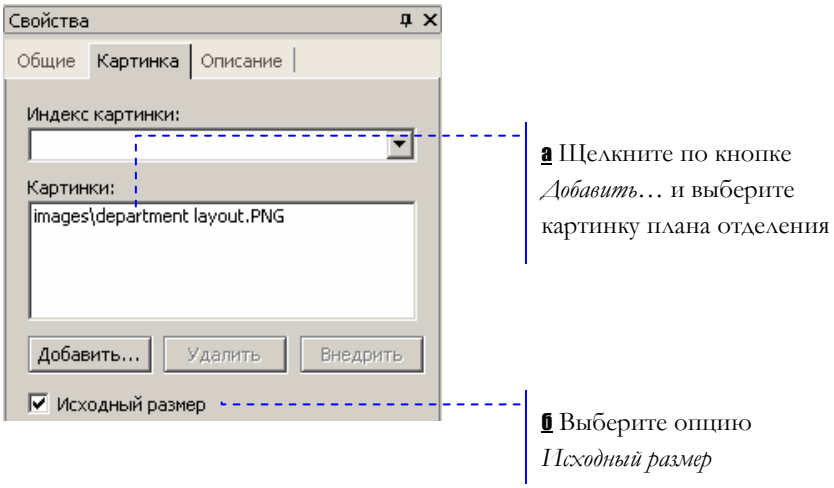
Чтобы облегчить рисование, вначале мы добавим изображение плана отделения. Мы не будем рисовать план в редакторе анимации AnyLogic, а просто вставим уже готовую картинку.

► Нарисуйте план отделения



❶ Чтобы создать новую анимацию, щелкните мышью по кнопке *Новая анимация* .

❷ Чтобы добавить картинку, щелкните мышью вначале по кнопке панели инструментов *Картинка* , а затем по анимационной диаграмме. Задайте следующие свойства картинки:

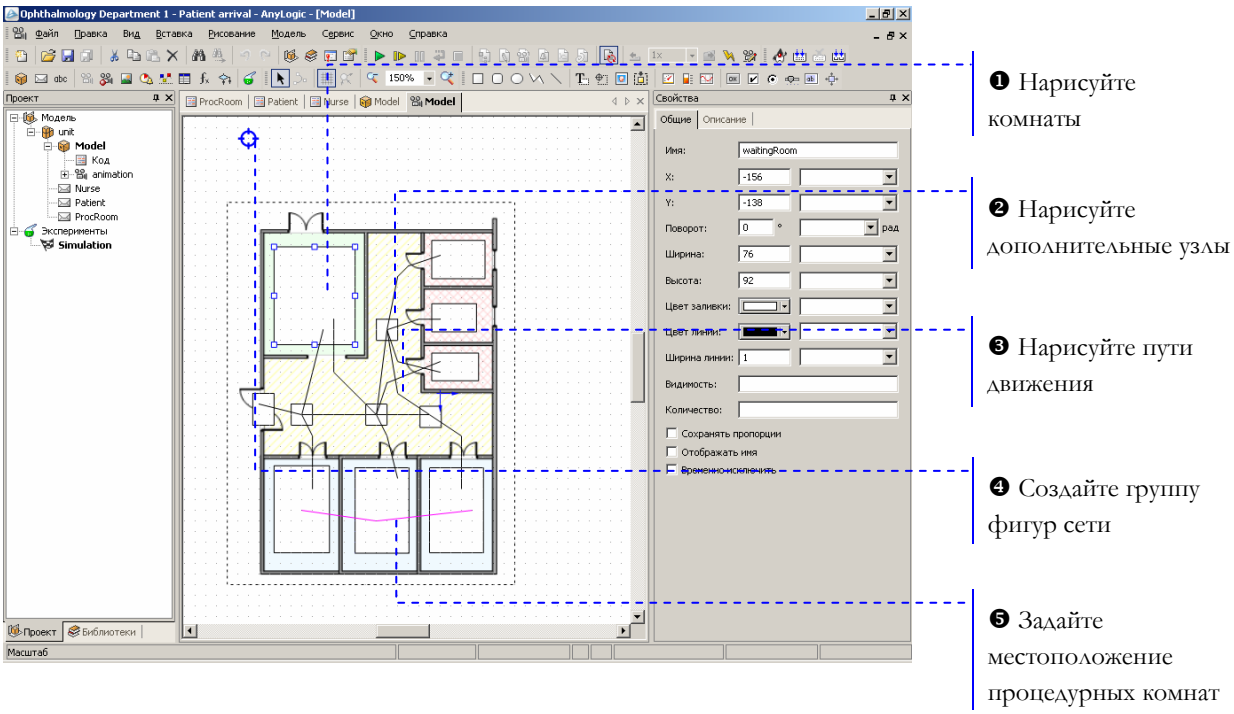


а Выберите картинку Examples \ Enterprise Library Tutorial Models \ images \ department layout.png

б Чтобы сохранить исходный размер изображения, установите флажок *Исходный размер*.

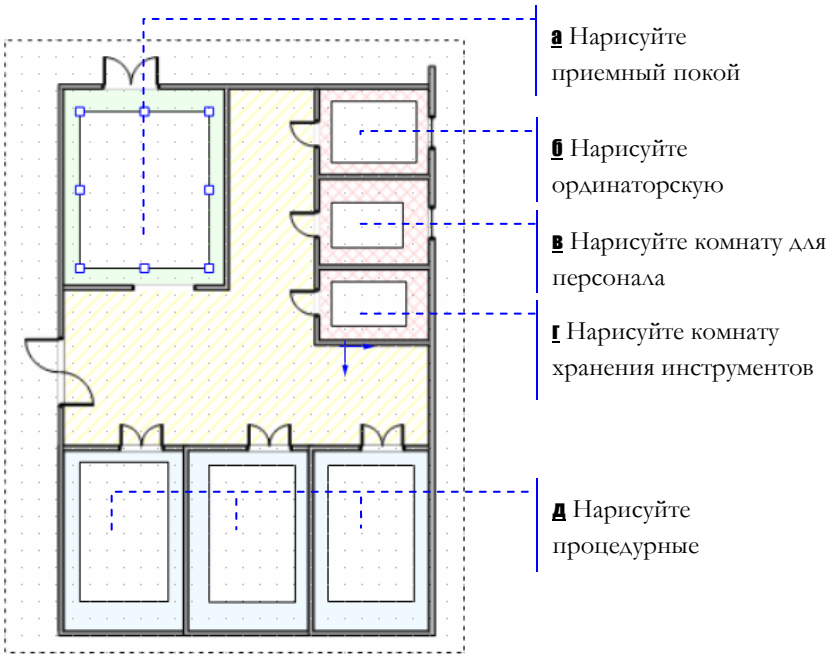
Теперь мы нарисуем анимацию модели. На основе анимации строится транспортная сеть модели: прямоугольники соответствуют узлам сети, а ломаные линии – связям между ними, играющим роль путей движения в модели. Поэтому, чтобы создать требуемую транспортную сеть, мы должны нарисовать на анимации прямоугольниками помещения нашего отделения, и соединить их ломаными линиями.

► Нарисуйте анимацию модели



❶ Отделение офтальмологии включает в себя приемный покой, три процедурные, комнату хранения офтальмоскопов, ординаторскую и комнату для персонала.

Нарисуйте помещения отделения с помощью инструмента рисования *Прямоугольник* . Измените размер прямоугольников так, чтобы они помещались в соответствующих областях на плане отделения, как показано на рисунке ниже:



а Назовите прямоугольник `waitingRoom`.

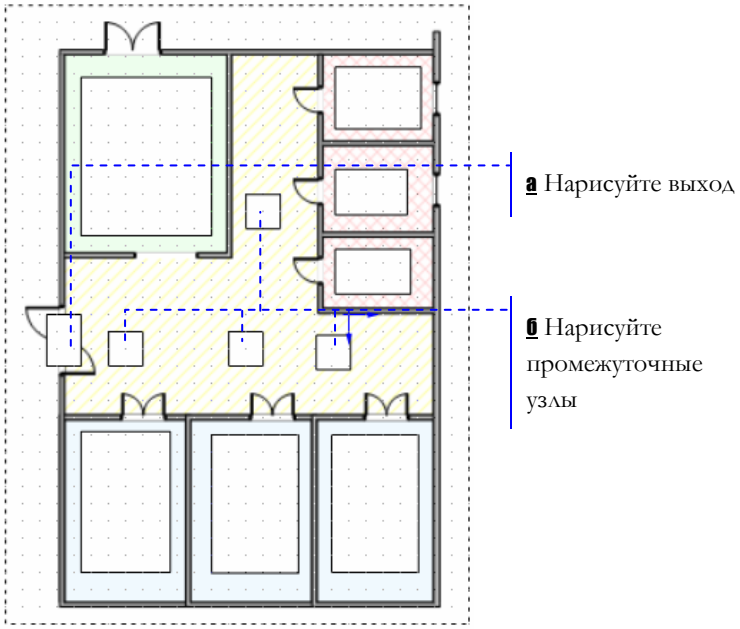
б Назовите прямоугольник `staffroom`.

в Назовите прямоугольник `staffLounge`.

г Назовите прямоугольник `storageRoom`.

д Нарисуйте три прямоугольника и назовите их `procRoom1`, `procRoom2` и `procRoom3`. Они будут соответствовать процедурным комнатам отделения.

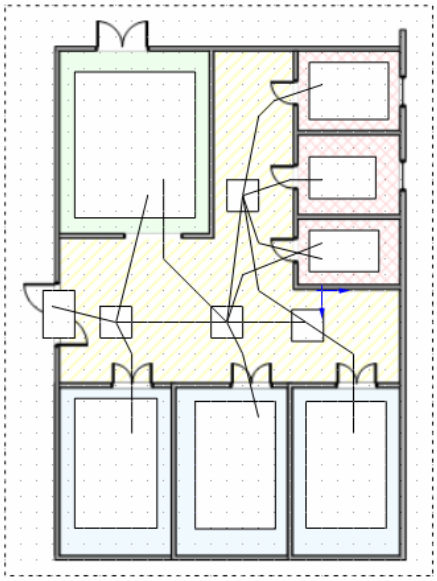
❷ Чтобы сделать пути движения людей на анимации более реалистичными, добавьте дополнительные узлы сети, нарисовав еще несколько прямоугольников:



а Назовите прямоугольник `exit`. Он будет обозначать выход из отделения.

б Нарисуйте четыре прямоугольника и расположите их так, как показано на рисунке. Назовите прямоугольники `upNode`, `leftNode`, `middleNode` и `rightNode`, в соответствии с их положением на анимации.

❸ С помощью инструмента рисования *Ломаная*  , нарисуйте ломаные линии, как показано на рисунке ниже.



Ломаные линии задают пути движения пациентов и персонала больницы. Чтобы задать требуемую транспортную сеть модели, соедините соседние узлы, а именно:

- `leftNode` с `exit`, `waitingRoom`, `procRoom1` и `middleNode`;
- `middleNode` — с `waitingRoom`, `procRoom2`, `storageRoom`, `rightNode` и `upNode`;
- `rightNode` — с `procRoom3` и `upNode`;
- `upNode` — с `storageRoom`, `staffLounge` и `staffroom`.

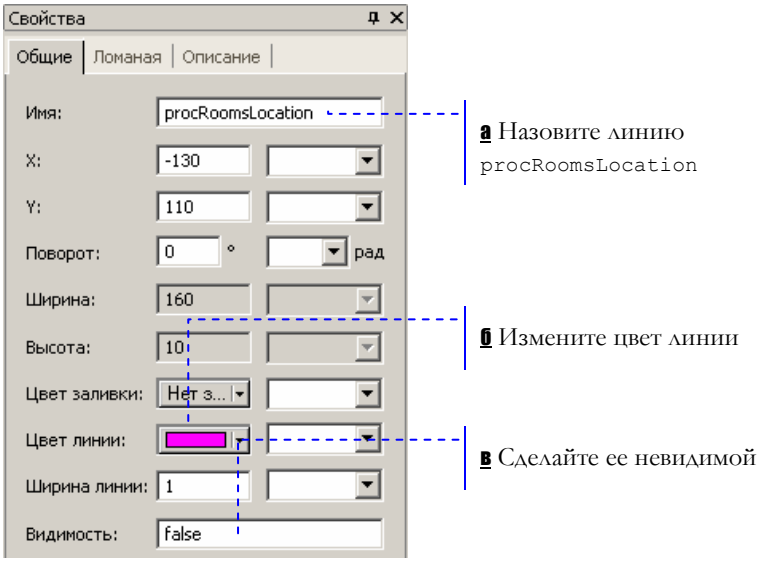
➲ Все начальные и конечные точки линий должны обязательно находиться внутри соединяемых прямоугольников.

❹ Создайте группу фигур и назовите ее `networkPivot`. Логическая структура сети будет сконструирована на базе элементов, добавленных в эту группу фигур.

Добавьте все созданные фигуры в группу фигур. Для этого сделайте щелчок правой кнопкой мыши по значку группы фигур и выберите *Добавить/удалить фигуры* из контекстного меню. Затем выберите все нарисованные ранее фигуры и щелкните мышью по анимационной диаграмме.

❺ Нарисуйте ломаную линию, соединяющую прямоугольники, представляющие собой процедурные комнаты. Она понадобится нам чуть позже.

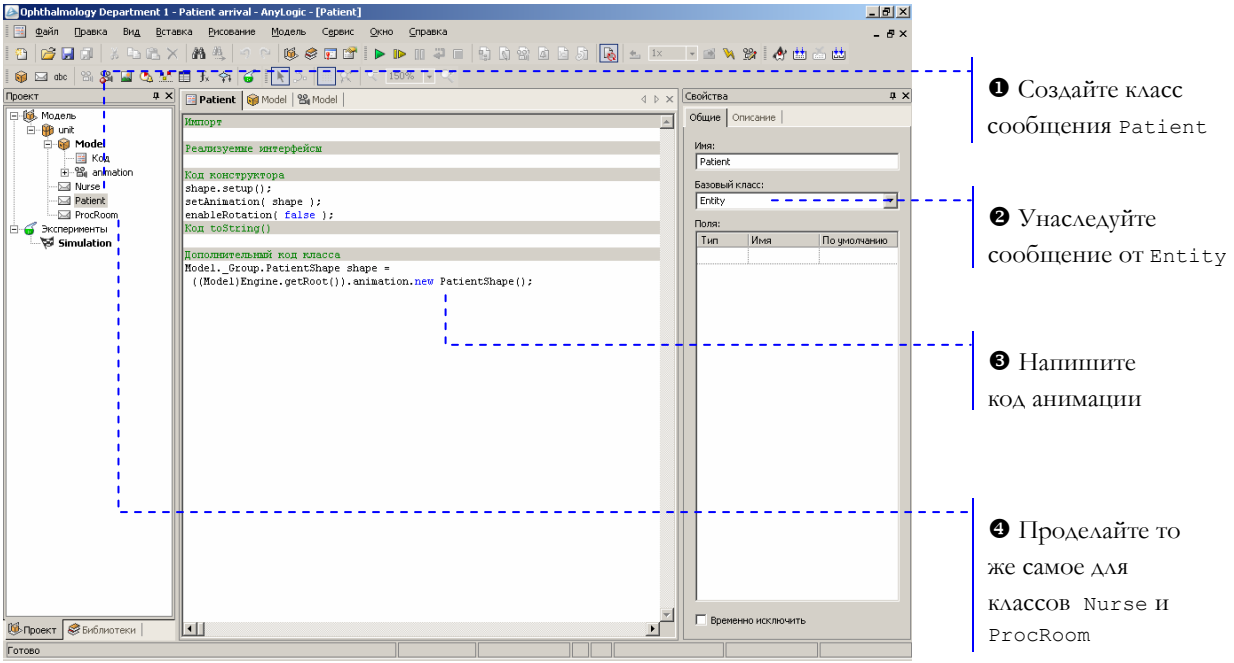
Поместите точки ломаной внутрь прямоугольников `procRoom1`, `procRoom2` и `procRoom3`. Задайте следующие свойства:



4.3 Создание классов сообщений

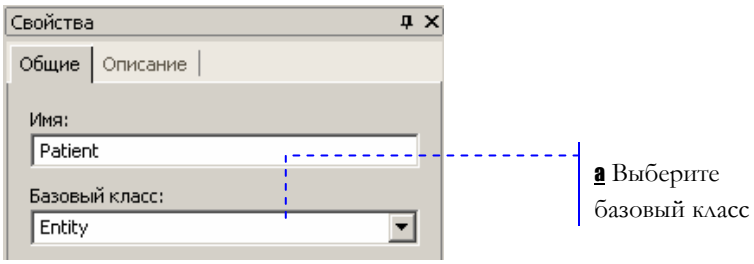
Теперь надо сделать так, чтобы мы смогли различить на анимации пациентов и медсестер. Для этого мы создадим для каждого из них свой класс сообщения с уникальной анимацией.

► **Создайте классы сообщений**



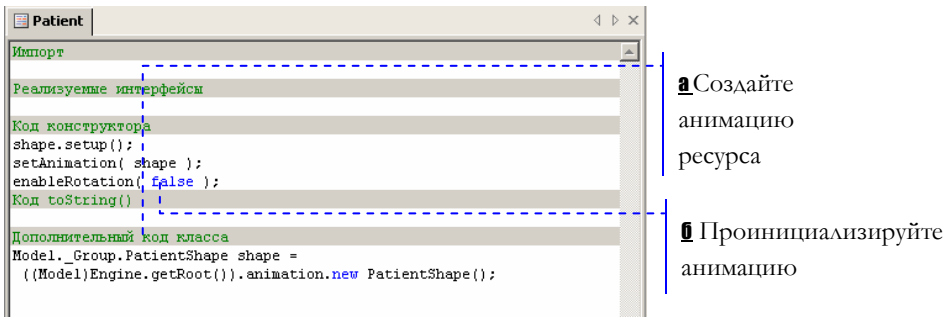
❶ Создайте класс сообщения щелчком мыши по кнопке панели инструментов *Новый класс сообщения*  . Назовите его `Patient`. Сообщения этого класса будут представлять в нашей модели пациентов.

❷



a При работе с Enterprise Library в качестве базового класса нужно выбирать класс сообщения `Entity`.

❸



a Напишите следующий код в поле *Дополнительный код класса*:

```
Model._Group.PatientShape shape =
((Model)Engine.getRoot()).animation.new
PatientShape();
```

Этот код создает новый экземпляр динамической группы фигур `PatientShape`.

б Чтобы добавить созданную группу фигур на анимацию, напишите следующий *Код инициализации*:

```
shape.setup();
setAnimation( shape );
enableRotation( false );
```

❹ Создайте класс сообщений `Nurse`. Сообщения этого класса будут представлять в нашей модели медсестер. Создайте класс так же, как и в предыдущем случае, за исключением того, что *Дополнительный код класса* должен быть таким:

```
Model._Group.NurseShape shape =
```



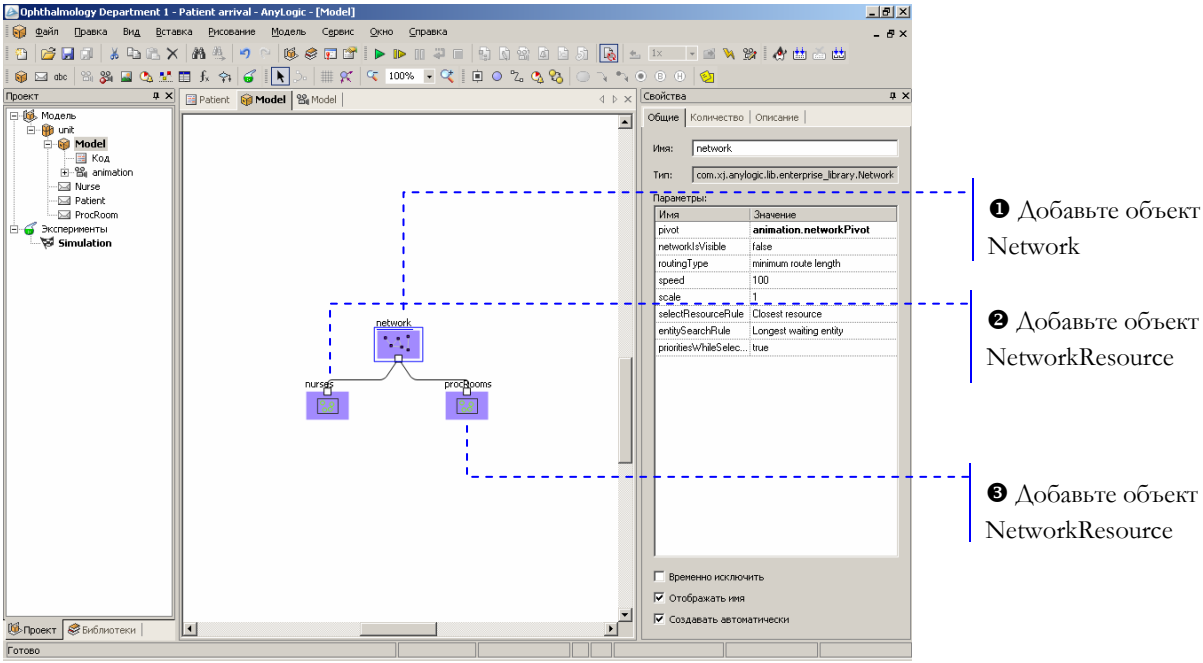
```
((Model)Engine.getRoot()).animation.new NurseShape();
```

Аналогично создайте и класс сообщения `ProcRoom`. С помощью сообщений этого класса в модели будут заданы процедурные комнаты. Поскольку на анимации они отображаться не будут, код анимации писать не нужно.

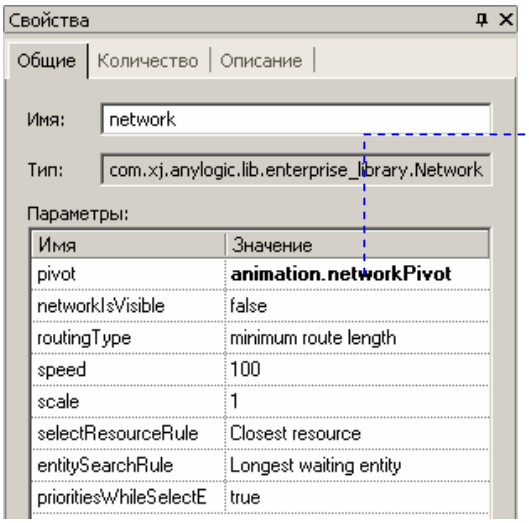
4.4 Задание транспортной сети

Теперь мы добавим объекты, описывающие саму сеть и ее ресурсы. Ресурсы могут быть трех видов: *персонал* (staff), *переносные* (portable) и *статические* (static). В нашем случае медсестры и врачи будут заданы ресурсами типа «персонал», офтальмоскопы – портативными ресурсами, а процедурные комнаты – статическими. Сейчас мы создадим только ресурсы, задающие медсестер и процедурные комнаты; другие ресурсы мы создадим позднее.

► **Задайте сеть и ее ресурсы**



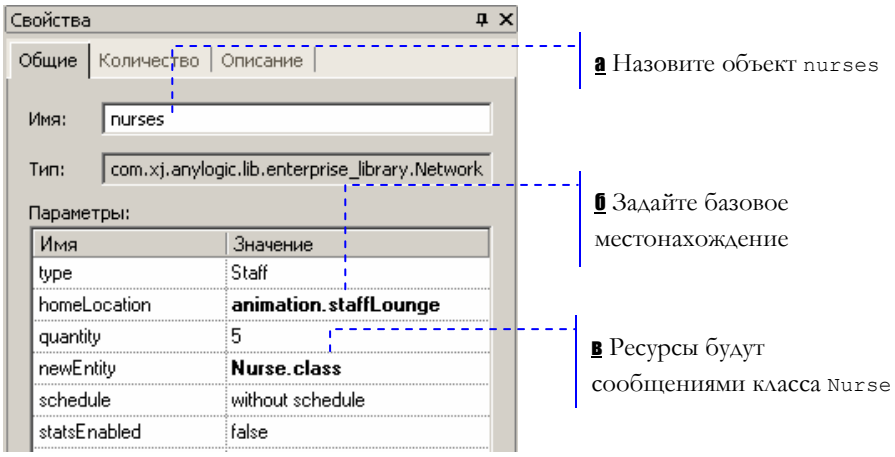
❶ Объект Network задает свойства транспортной сети модели. Задайте следующие свойства объекта:



а Выберите группу фигур, определяющую структуру транспортной сети.

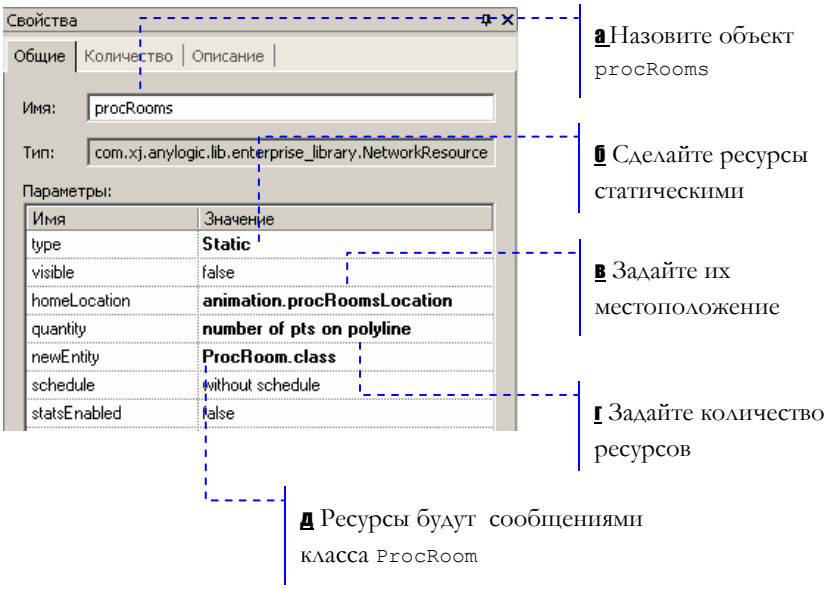
❷ Объект NetworkResource описывает ресурсы определенного типа. Этот объект будет задавать свойства ресурсов, представляющих в нашей модели медсестер.

Задайте следующие свойства объекта:



б Ресурс типа «персонал» возвращается в заданное место, когда становится свободным.

❸ Этот объект будет задавать свойства ресурсов, представляющих в нашей модели процедурные комнаты. Задайте следующие свойства объекта:

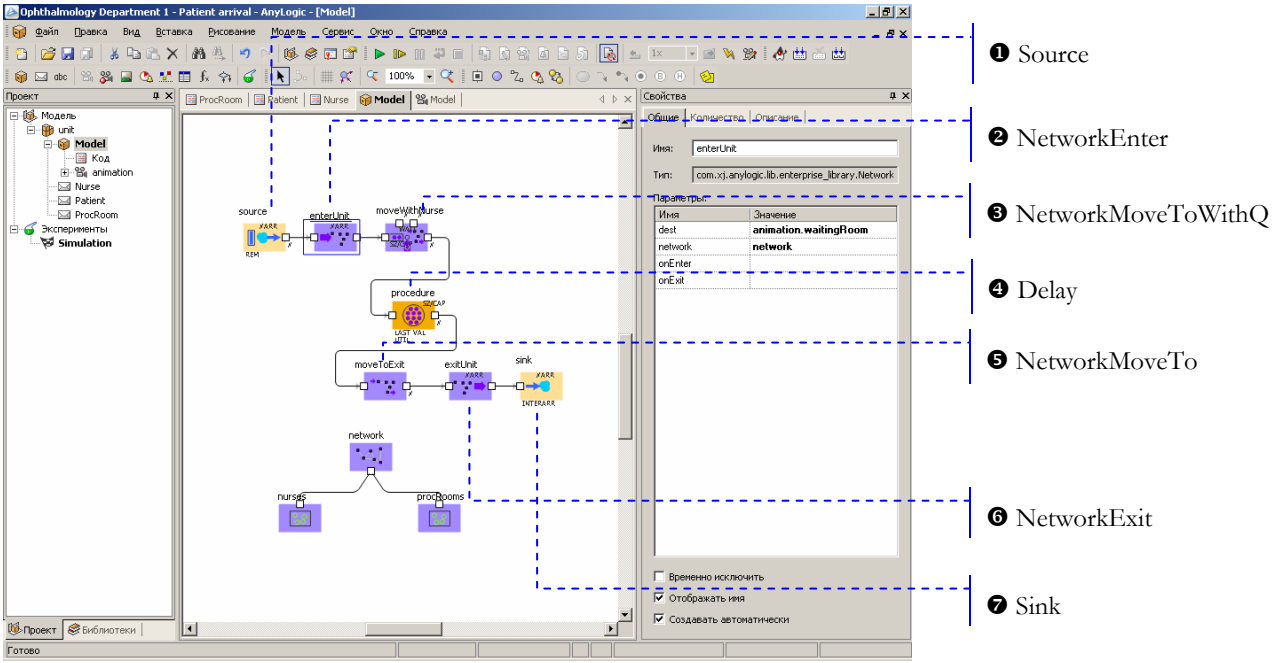


в Статические ресурсы всегда находятся в месте, указанном как базовое. Вы можете задать несколько таких мест, для этого нужно нарисовать ломаную линию с точками, лежащими в соответствующих прямоугольниках, и выбрать эту линию в свойстве объекта `homeLocation`. Выберите ранее созданную для этой цели ломаную `procRoomsLocation`.

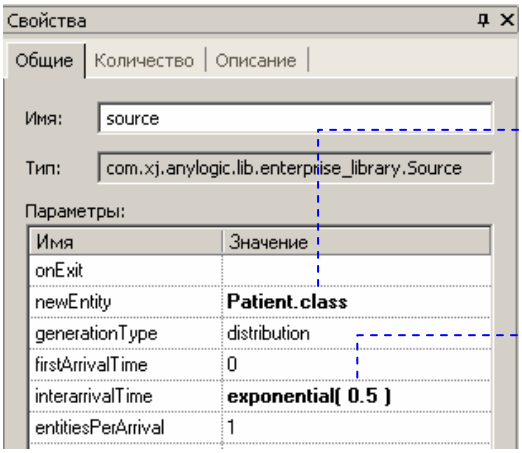
г Укажите, что количество ресурсов равно числу точек заданной ломаной линии.

4.5 Создание блок-схемы

Теперь мы создадим блок-схему нашей модели.



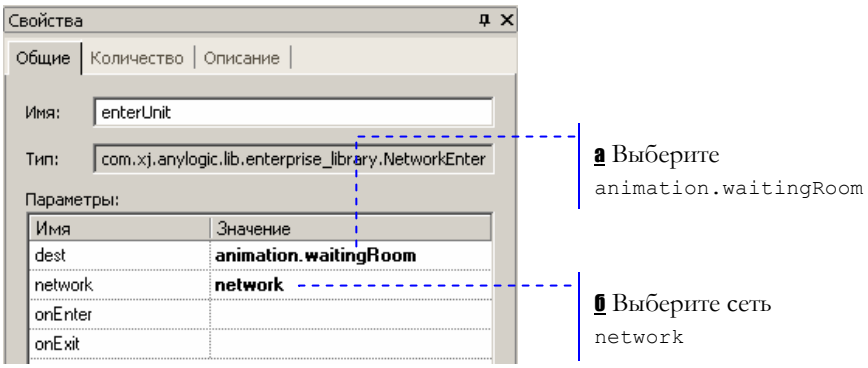
❶ Добавьте объект Source. Задайте следующие свойства объекта:



б Задайте экспоненциально распределенное время со средним значением, равным 2 минутам.

❷ Объект NetworkEnter добавляет сущности в заданное место сети.

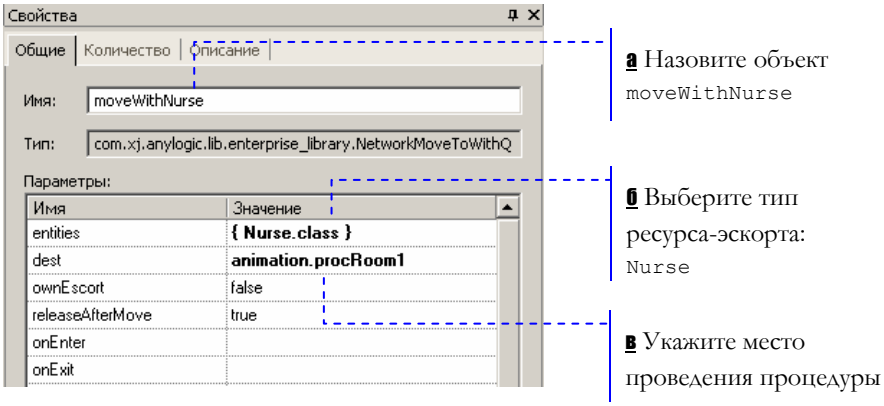
Задайте следующие свойства объекта:



а Пациенты будут приходить в приемный покой отделения.

б Выберите объект, задающий транспортную сеть модели.

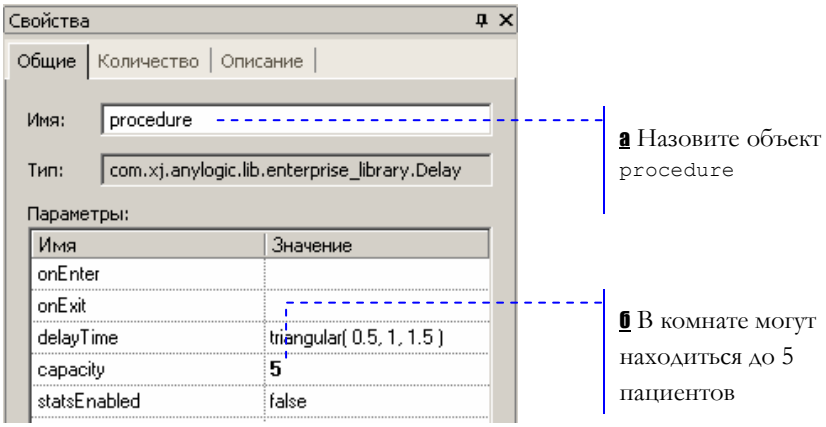
❸ Объект MoveToWithQ перемещает сущность в заданное место в сопровождении эскорта – ресурсов типа «персонал». Этот объект нужен нам, чтобы промоделировать то, как пациенты отводятся медсестрами в процедурные комнаты.



в На данный момент все пациенты будут осматриваться в первой процедурной комнате.

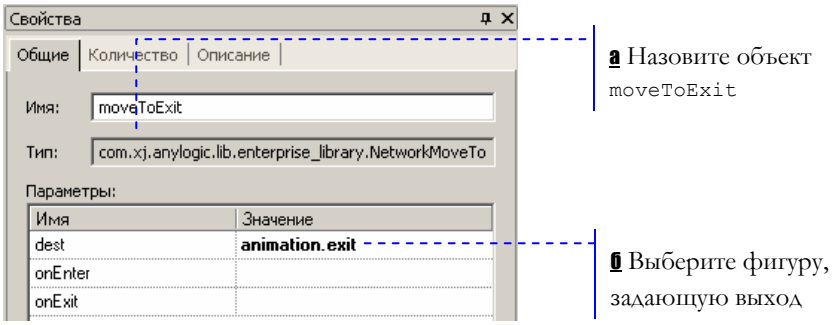
❹ Объект Delay моделирует задержку, связанную с проведением процедуры.

Задайте следующие свойства объекта:



❺ Объект MoveTo моделирует то, как пациенты направляются к выходу после проведения процедуры.

Задайте следующие свойства объекта:

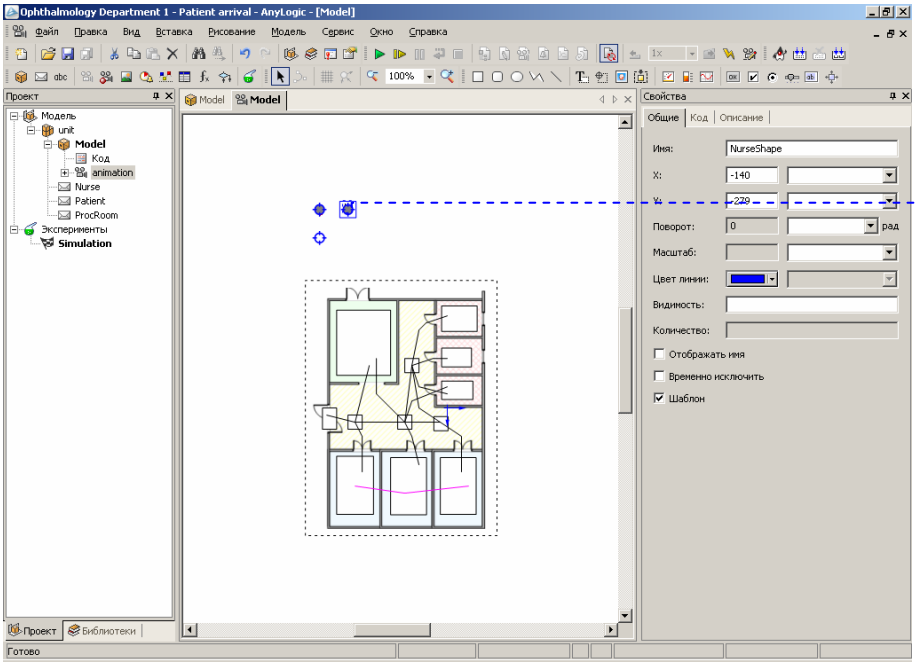


❻ Объект NetworkExit моделирует уход пациентов из отделения. Оставьте принятые по умолчанию свойства объекта.

❼ Объект Sink завершает блок-схему. Оставьте принятые по умолчанию свойства объекта.

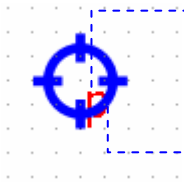
4.6 Анимация ресурсов модели

Чтобы анимация была более наглядной, мы нарисуем специальные символы, с помощью которых будем отображать ресурсы нашей модели.



❶ Нарисуйте символы

❶ Нарисуйте символ, которым на анимации будет отображаться пациент.

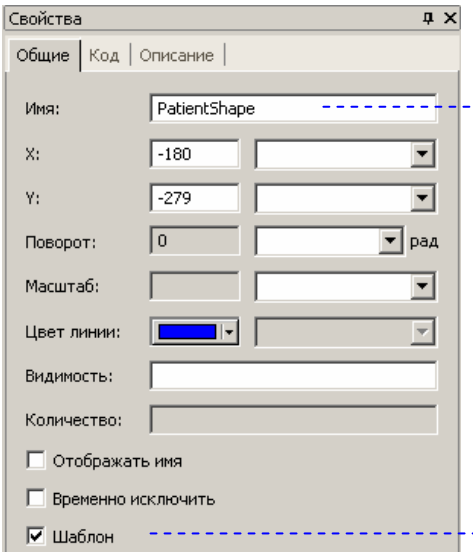


a Создайте метку p

б Создайте группу фигур и добавьте в нее созданную метку

a Создайте текстовую метку щелчком мыши по кнопке панели инструментов *Текст*  , а затем щелкнув по анимационной диаграмме. Задайте красный шрифт размера 7.

б Создайте динамическую группу фигур.

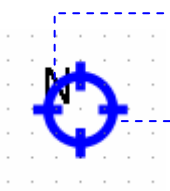


в Назовите группу фигур `PatientShape`

г Сделайте ее динамической

Добавьте метку к группе фигур. Сделайте щелчок правой кнопкой мыши по фигуре группы фигур, выберите *Добавить/удалить фигуры* из контекстного меню, а затем щелкните по метке. После этого поместите метку поверх значка группы фигур.

Теперь нарисуйте символ, которым будут отображаться медсестры.



д Создайте метку N

е Создайте динамическую группу фигур `NurseShape`, добавьте к ней метку, и поместите ее на значок группы фигур

Мы закончили создание простой транспортной модели. Запустите модель щелчком мыши по кнопке *Запустить*  . Вы увидите, что поведение модели совпадает с ожидаемым: пациенты прибывают в приемный покой, затем медсестра отводит их в процедурную, которую они покидают после прохождения процедуры.

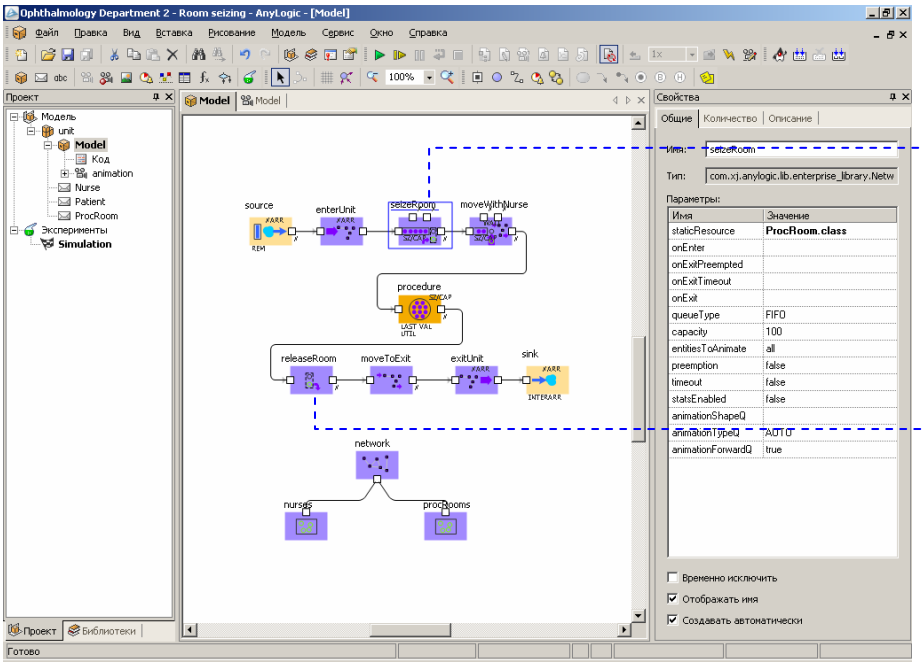
Вы можете изменить скорость анимации с помощью кнопок панели инструментов *Уменьшить скорость*  и *Увеличить скорость*  .

Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Ophthalmology Department 1 - Simple model.alp.

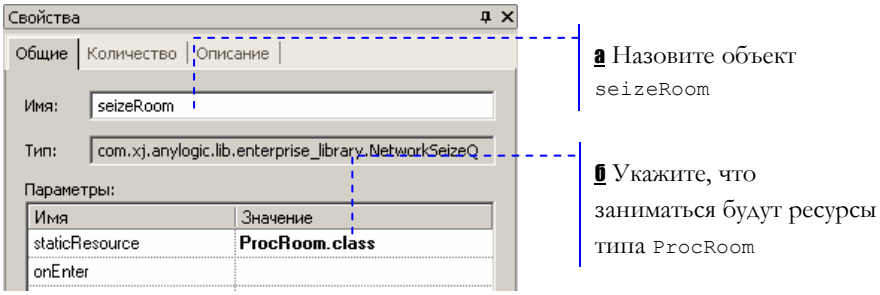
4.7 Моделирование занятия комнат

На данный момент все пациенты осматриваются только в первой процедурной комнате. Более того, несколько пациентов осматриваются в одной комнате одновременно. Сейчас мы улучшим нашу модель, задав процедурные комнаты статическими ресурсами. Пациенты будут теперь осматриваться в любой из трех процедурных комнат. На время проведения процедуры комната будет занята, так что никто другой не сможет быть осмотрен в ней в то же время.

► **Измените блок-схему**

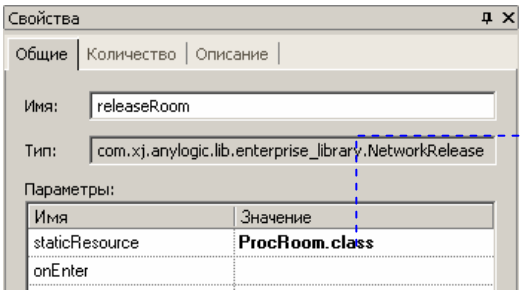


❶ Объект NetworkSeizeQ занимает статические ресурсы. Мы добавляем этот объект для того, чтобы смоделировать то, как пациент занимает процедурную комнату. Задайте следующие свойства:



❷ Объект NetworkRelease освобождает ранее занятые статические ресурсы. С помощью этого объекта мы будем освобождать ранее занятые процедурные комнаты после проведения процедуры.

Задайте следующие свойства объекта:



a Укажите, что освобождаться будут ресурсы типа `ProcRoom`

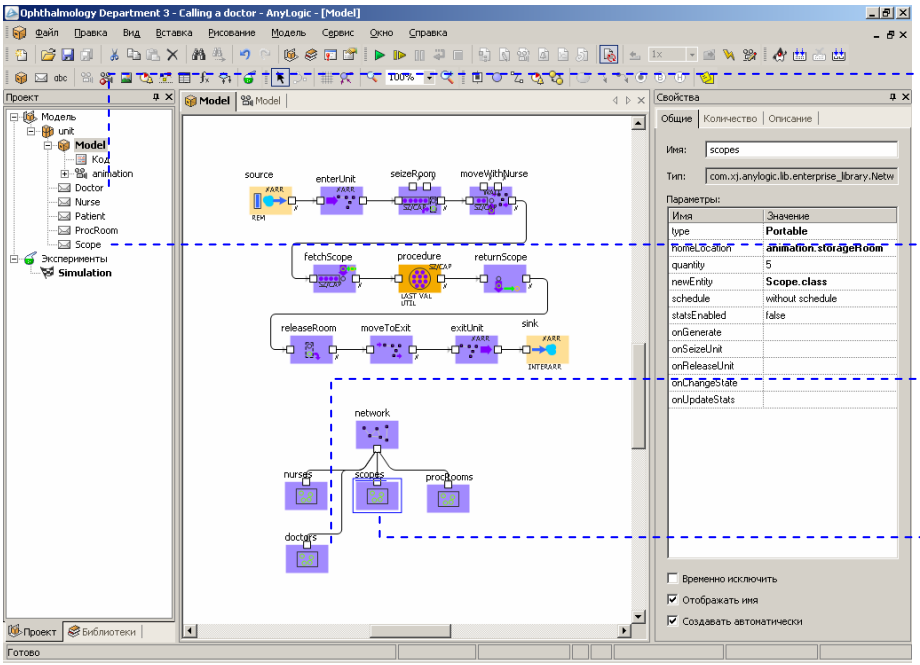
Запустите модель щелчком мыши по кнопке *Запустить*  .

Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Ophthalmology Department 2 - Room seizing.alp.

4.8 Моделирование вызова врача

Теперь мы закончим создание нашей модели, промоделировав вызов офтальмолога в процедурную комнату.

► Создайте новые ресурсы



❶ Создайте класс сообщения Doctor

❷ Создайте класс сообщения Scope

❸ Добавьте объект NetworkResource

❹ Добавьте объект NetworkResource

❶ Создайте класс сообщения `Doctor`. Сообщения этого класса будут представлять в нашей модели офтальмологов.

Унаследуйте класс сообщения от класса Entity.

а Чтобы создать экземпляр группы фигур этого класса, напишите следующий *Дополнительный код класса*:

```
Model._Group.DoctorShape shape =
((Model)Engine.getRoot()).animation.new
DoctorShape();
```

б Чтобы добавить на анимацию созданную группу фигур, напишите следующий *Код инициализации*:

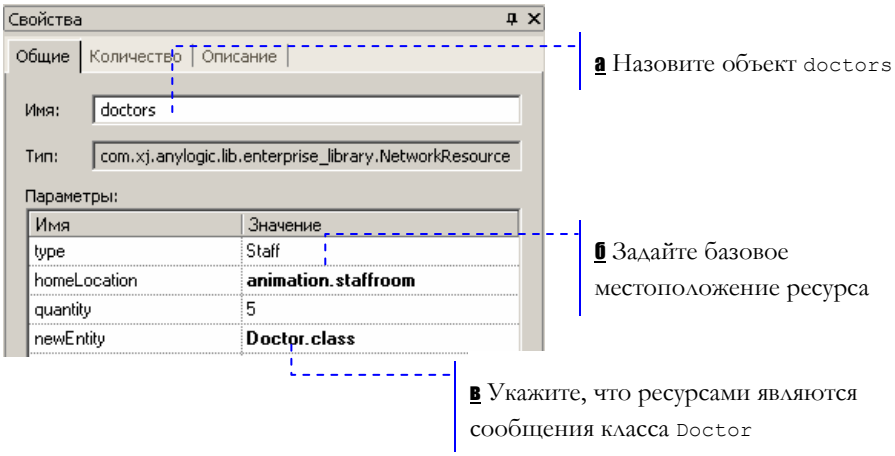
```
shape.setup();
setAnimation( shape );
enableRotation( false );
```

❷ Создайте класс сообщения `Scope`. Сообщения этого класса будут представлять офтальмоскопы.

Унаследуйте класс сообщения от класса Entity. Анимационный код писать не надо, поскольку мы не будем задавать для объекта уникальную анимацию.

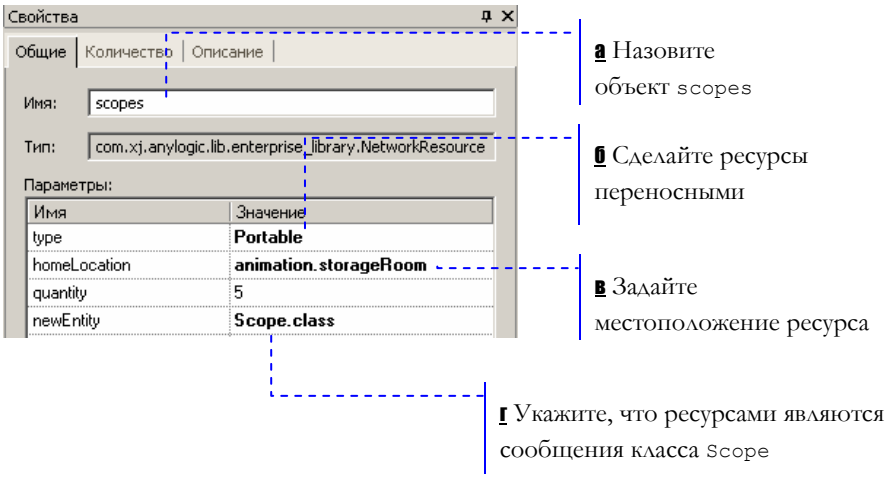
❸ Объект будет задавать свойства ресурсов типа Doctor.

Задайте следующие свойства объекта:

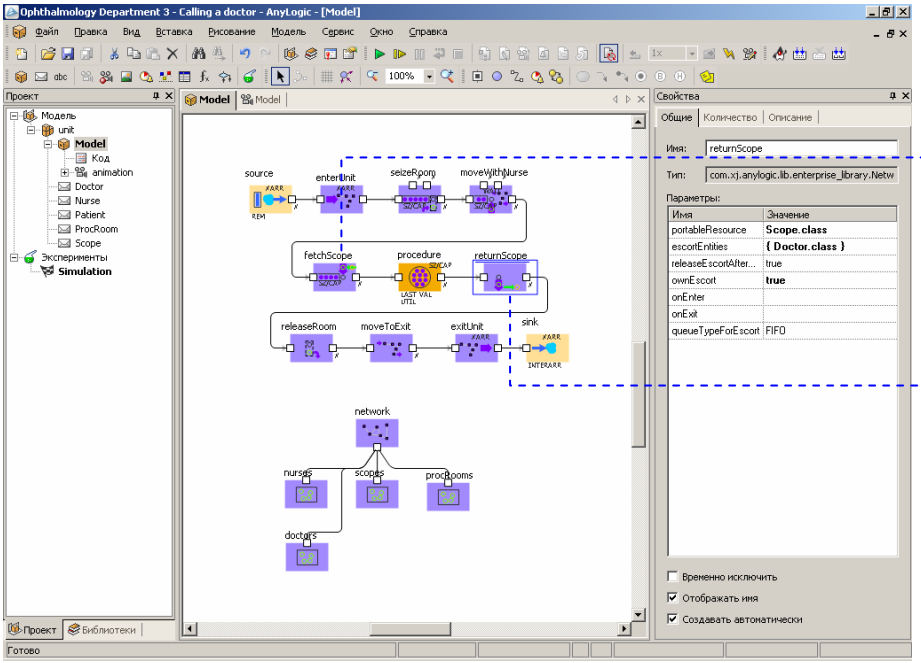


❹ Объект будет задавать свойства ресурсов – офтальмоскопов.

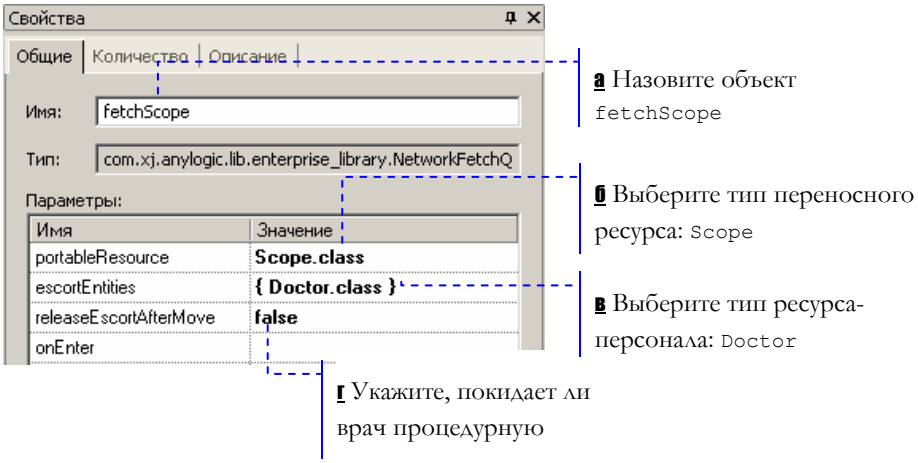
Задайте следующие свойства объекта:



► Измените блок-схему



❶ Этот объект моделирует доставку переносных ресурсов с помощью персонала. Задайте следующие свойства объекта:

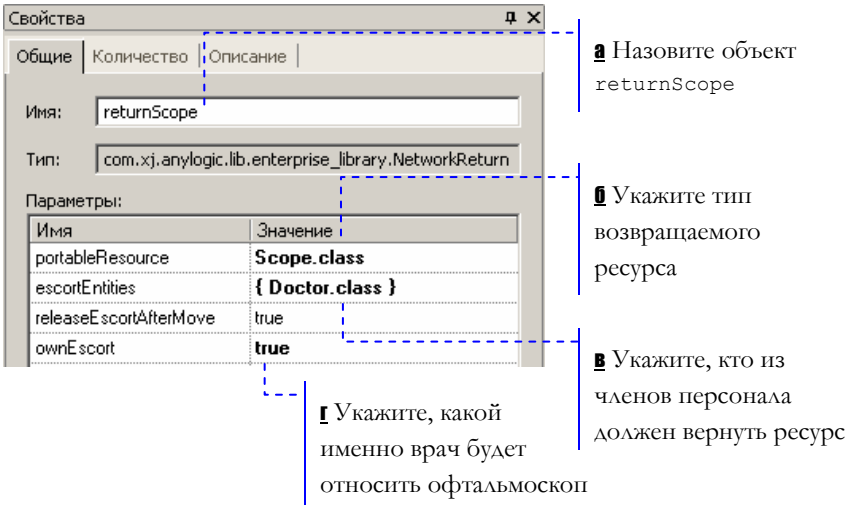


В Переносные ресурсы могут приноситься и несколькими разными членами персонала, подробную информацию смотрите в *Справочном руководстве по Enterprise Library*.

Г Когда офтальмоскоп будет принесен, мы можем отпустить врача. Однако в нашей модели он должен остаться в процедурной для того, чтобы провести процедуру.

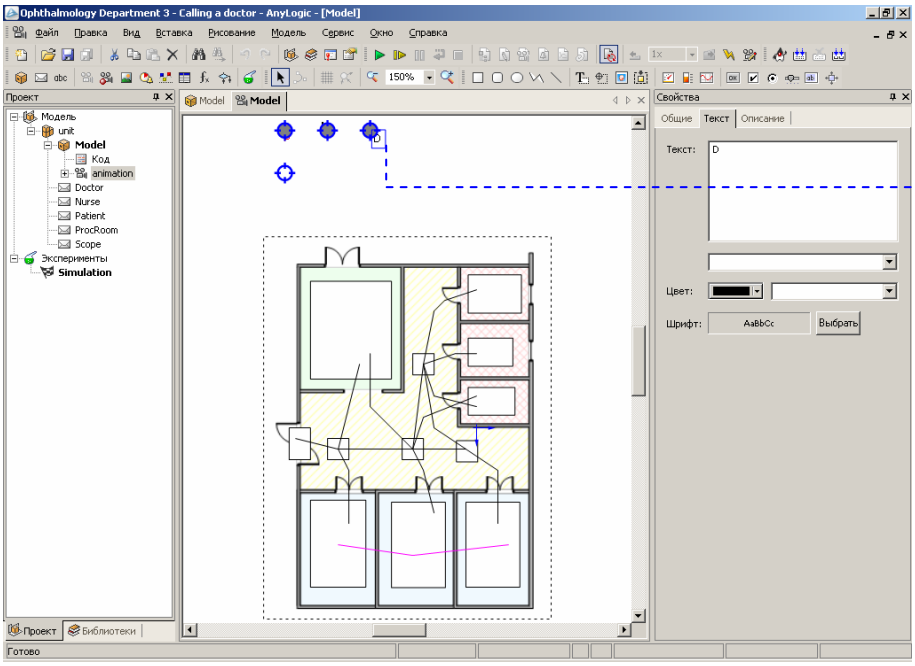
❷ С помощью объекта NetworkReturn мы промоделируем то, как врачи относят офтальмоскопы обратно в комнату для инструментов.

Задайте следующие свойства объекта:



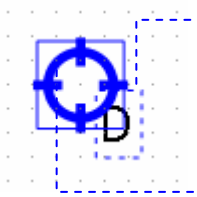
г Мы не будем вызывать еще одного врача; офтальмоскоп будет относить тот самый врач, который его и принес.

Теперь мы зададим для врачей уникальную анимацию, аналогично тому, как мы делали это для медсестер и пациентов.



❶ Нарисуйте символ, которым врач будет отображаться на анимации

❶



a Создайте метку D

б Создайте группу фигур, добавьте в группу метку, и затем поместите метку над фигурой группы

б Создайте динамическую группу фигур. Назовите ее `DoctorShape`.

Мы закончили создание модели отделения офтальмологии. Запустите модель щелчком мыши по кнопке *Запустить*  . Теперь для проведения офтальмоскопии в процедурную вызывается офтальмолог с офтальмоскопом.

Текущая контрольная модель: Examples \ Enterprise Library Tutorial Models \ Ophthalmology Department 3 - Calling a doctor.alp

5. Заключение

Это учебное пособие научило Вас создавать модели с помощью библиотеки дискретно-событийного моделирования Enterprise Library. Рассмотренные модели демонстрируют возможность применения пакета в производственной сфере, а также для моделирования систем массового обслуживания и бизнес-процессов.

Чтобы лучше понять принципы построения моделей в AnyLogic, Вы можете изучить примеры моделей, поставляемые вместе с пакетом. Среди других примеров Вы можете найти и модели, созданные в Enterprise Library, в том числе:

- Модель терминала аэропорта
- Модель отделения выписки счетов
- Модель производства напитков
- Модель упаковочной линии
- Модель склада и сборочного модуля
- Модель центров обработки заказов

В том случае, если Вам нужно расширить функциональность Вашей модели и выйти за рамки дискретно-событийного моделирования, Вы можете использовать любой другой подход моделирования: например, задать сложное и нетривиальное поведение объектов модели с помощью стейтчартов. За детальной информацией о техниках и подходах моделирования, поддерживаемых пакетом AnyLogic, пожалуйста, обращайтесь к *Руководству пользователя*.