



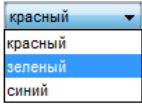
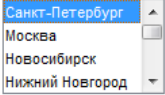
Создание интерактивных моделей с помощью элементов управления

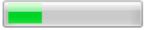
Модели AnyLogic можно сделать интерактивными, добавив в интерфейс модели различные [элементы управления](#) (кнопки, бегунки, текстовые поля и т.д.), а также задав действия, выполняемые в качестве реакции на щелчки мыши. Элементы управления могут использоваться как для задания значений параметров перед началом выполнения модели, так и для изменения модели прямо по ходу ее выполнения.

Элементы управления располагаются на палитре Элементы управления. Они создаются и редактируются так же, как и [фигуры](#). Элементы управления можно [группировать](#) с фигурами и другими элементами управления, и можно реплицировать. Как и у фигур, у элементов управления есть [динамические свойства](#), с помощью которых во время выполнения модели можно изменять их размер, местоположение, доступность и видимость. Элементы управления вложенного объекта можно настроить, чтобы они появлялись на презентации объекта верхнего уровня (см. [Иерархические модели](#)).

Элементы управления всегда отображаются поверх любой другой графики (фигур, элементов модели и т.д.) вне зависимости от z-порядка и группировки. Избегайте наложения элементов управления на другие элементы, поскольку это может вызвать нежелательные визуальные эффекты.

В AnyLogic у элементов управления, у которых есть состояние или содержимое (таких, как бегунок, переключатель, текстовое поле и т.д.), также есть [значение](#), и они могут быть [связаны](#) с переменными и параметрами, так что когда пользователь изменяет состояние такого элемента управления, изменяется и значение связанного с ним элемента (но не наоборот). Кроме того, вы можете задать для элемента управления любое действие, например: вызов функции, планирование события, посылку сообщения, остановку модели и т.д. Действие будет выполняться каждый раз, когда пользователь тронет элемент управления. Значение элемента управления обычно доступно в коде его поля Действие как `value`, а также возвращается методом элемента управления `getValue()`. В таблице ниже приведена краткая информация обо всех элементах управления AnyLogic.

Элемент управления	Тип значения	Может быть связан с типом	Комментарии
<p>Кнопка</p> 			Используется для немедленного выполнения заданных пользователем действий. Вы задаете код в поле Действие, и этот код выполняется при нажатии кнопки пользователем.
<p>Флажок</p> <input checked="" type="checkbox"/> Отображать анимацию	boolean	boolean	
<p>Текстовое поле</p> <input type="text" value="152.14"/>	String	String или любой численный тип (double , int и т.д.)	Помимо возможности связывания с переменной, вы можете задать собственный код, который будет обрабатывать ввод пользователя (проверять его правильность, принимать или отвергать).
<p>Переключатель</p> <input checked="" type="radio"/> вариант1 <input type="radio"/> вариант2	int	int	Первому варианту соответствует значение 0, второму 1, и так далее.
<p>Бегунок</p> 	double	double или любой численный тип (int и т.д.)	Вы можете ограничить интервал значений бегунка минимальным и максимальным значениями.
<p>Выпадающий список</p> 	String	String	Может быть редактируемым или фиксированным, с жестко заданным набором вариантов.
<p>Список</p> 	String	String	Может работать в режиме выбора одного или нескольких элементов. В режиме Выбор нескольких элементов список нельзя привязать, и его значение доступно с помощью метода <code>getValues()</code> , возвращающего массив строк String[] .
<p>Элемент выбора файла</p> <input type="text" value="C:\file.txt"/> ...	String		Отображает системное диалоговое окно Открыть файл или Сохранить файл и сохраняет результат, содержащий имя файла и полный путь к нему, в строке String . Вы можете задать фильтры, основанные на расширениях файлов.

<p>Индикатор прогресса</p> 	<p>double</p>	<p>double или любой численный тип</p>	<p>В фиксированном режиме отображает процент выполнения задачи. В нефиксированном просто анимирует факт выполнения активности. Свойства Значение прогресса и Фиксированный являются динамическими, т.е. их значения постоянно вычисляются заново во время выполнения модели.</p>
--	----------------------	--	--

Пример: Бегунок, связанный с параметром модели (Slider linked to a model parameter)

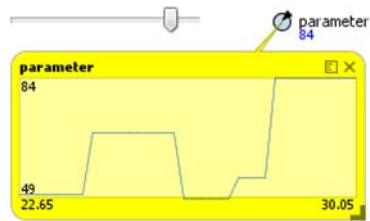
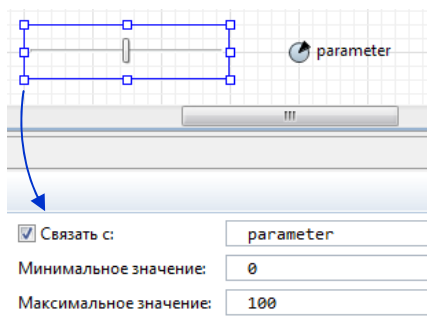
Мы создадим параметр и бегунок и свяжем их вместе. В дальнейшем мы продолжим развивать этот пример.

▶ **Выполните следующие шаги:**

1. Создайте параметр, перетащив элемент Параметр из палитры Основная на диаграмму.
2. На странице свойств параметра Основные задайте его значение по умолчанию равным 50.
3. Откройте палитру Элементы управления и перетащите элемент Бегунок на диаграмму рядом с параметром. Немного увеличьте размер бегунка, как показано на рисунке.
4. На странице свойств бегунка Основные установите флажок Связать с.
5. Задайте следующие свойства:
Связать с: **parameter**
Минимальное значение: **0**
Максимальное значение: **100**
6. Запустите модель. Перемещайте бегунок, наблюдая, как при этом будет меняться значение параметра (можете щелкнуть по параметру, чтобы посмотреть его временной график).

Как вы могли заметить, при запуске модели позиция бегунка устанавливается в значение 50 (значение параметра по умолчанию). Если значение параметра выйдет за границы интервала допустимых значений бегунка, то бегунок примет ближайшее допустимое значение, но значение параметра не изменится, пока вы не коснетесь бегунка.

Точной настройки позиции бегунка можно добиться с помощью клавиш стрелок.



Выполнение модели:
значение параметра меняется согласно вашим перемещениям бегунка

Бегунок, связанный с параметром

Вы должны помнить, что если значение параметра изменяется "извне", т.е. не с помощью бегунка, то позиция бегунка *не будет* автоматически изменяться в соответствии с новым значением. Если вы хотите, чтобы связь всегда работала "в обоих направлениях", то вы можете, например, добавить соответствующий код в поле Действие при изменении параметра (см. следующий пример).

Пример: Кнопки, изменяющие значение параметра (Buttons changing the parameter value)

Мы добавим две кнопки в модель предыдущего примера. Эти кнопки будут увеличивать и уменьшать значение параметра на единицу, сохраняя при этом значение в диапазоне 0-100.

► **Добавьте кнопки:**

1. Создайте две кнопки (перетащите их из палитры Элементы управления) и расположите их, как показано на рисунке.
2. На странице свойств Основные левой кнопки задайте:
 Метка: **-1**
 Доступность: `parameter >= 1`
 Действие: `set_parameter(parameter-1);`
3. На странице свойств Основные правой кнопки задайте:
 Метка: **+1**
 Доступность: `parameter <= 99`
 Действие: `set_parameter(parameter+1);`
4. Запустите модель. Переместите бегунок и понажимайте созданные вами кнопки.

Не путайте *имя* элемента управления с его *меткой*. Имя является именем объекта Java, созданного для этого элемента управления (и используется для доступа к методам элемента управления), а метка задает текст, отображающийся на экране на элементе управления или рядом с ним. Метку можно динамически менять во время выполнения модели.

Генерируемый автоматически метод `set_parameter(parameter-1);` вызывается в коде действия кнопки вместо более простого кода `parameter-`, чтобы быть уверенными в том, что значение параметра изменяется правильным образом, а именно - что выполняется код его Действия при изменении (см. главу [Параметры](#)). Выражение, которое вы задаете в поле элемента управления Доступность, постоянно вычисляется заново во время выполнения модели. Если оно равно `false`, то элемент управления становится недоступен (блокируется), и наоборот. В нашем случае мы не хотим, чтобы пользователь мог задавать значения, выходящие за границы интервала 0-100, поэтому когда значение приближается к границе интервала меньше чем на 1, мы блокируем соответствующую кнопку.

Имя:	button	←	Отображать имя	Имя Java объекта
Метка:	-1	←		Текстовая метка на кнопке
Доступность:	<code>parameter >= 1</code>	←		Это выражение постоянно вычисляется заново во время выполнения модели
Действие:	<code>set_parameter(parameter-1);</code>	←		Этот код выполняется при нажатии кнопки пользователем

Выполнение модели:
 с помощью кнопок значение параметра было установлено равным 0. Левая кнопка стала недоступной.

Кнопки, увеличивающие и уменьшающие значение параметра

Как вы можете видеть, когда значение параметра изменяется в результате вашего нажатия на кнопку, позиция бегунка не меняется, и возникает несогласованность значения бегунка со значением параметра. Чтобы устранить ее, нам нужно перемещать бегунок при каждом изменении значения параметра.

► **Задайте для параметра код Действия при изменении**

5. Напишите в поле Действие при изменении на странице свойств параметра Основные следующий код: `slider.setValue(parameter);`
6. Запустите модель. Понажимайте на кнопки и наблюдайте за поведением бегунка.

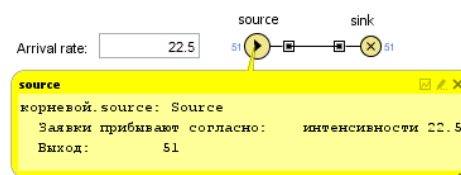
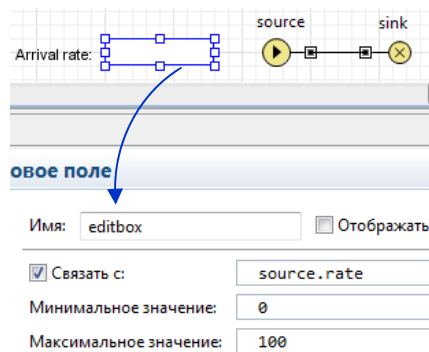
Теперь и значение параметра, и положение бегунка, и кнопки разрешения/запрещения доступа согласованы друг с другом.

Пример: Текстовое поле, связанное с параметром вложенного объекта (Edit box linked to a parameter of embedded object)

Элементы управления могут быть связаны не только с параметрами и переменными "текущего" активного объекта (т.е. того объекта, которому они принадлежат), но и с параметрами других объектов, например, вложенных. В этом примере мы свяжем текстовое поле с параметром `rate` (задающим интенсивность создания заявок) объекта **Source** в простой [процессной модели](#).

► **Выполните следующие шаги:**

1. Откройте палитру Основной библиотеки и перетащите на диаграмму объекты **Source** и **Sink**.
2. Сделайте двойной щелчок по выходному порту объекта `source` и соедините его с входным портом объекта `sink`.
3. Откройте палитру Элементы управления и перетащите элемент Текстовое поле на диаграмму. Поместите его слева от объекта `source`.
4. На странице свойств текстового поля Основные задайте:
Связать с: флажок установлен, значение: `source.rate` (используйте [мастер подстановки кода](#))
Минимальное значение: 0
Максимальное значение: 100
5. [необязательно] Добавьте поясняющий текст "Arrival rate" (Интенсивность прибытия) слева от текстового поля.
6. Запустите модель.
7. Щелкните по объекту `source`, чтобы открыть его окно инспекта.
8. Попробуйте задать различные значения интенсивности прибытия: 20, 0, -1, "abc" и т.д.



Выполнение модели: интенсивность создания заявок меняется в соответствии с вводимыми пользователем значениями

Текстовое поле, связанное с параметром вложенного объекта Source

Параметр `rate` объекта `source` изменяется при вводе каждого нового допустимого значения; при этом каждый раз вызывается соответствующий метод `set_rate()`. Если текстовое поле связано с параметром численного типа, то ошибочно введенные значения, которые невозможно привести к численному типу, равно как и численные значения, лежащие за пределами интервала допустимых значений, будут автоматически отвергнуты, и значение при этом не изменится.

Пример: Переключатель для выбора режима просмотра (Radio buttons changing the view mode)

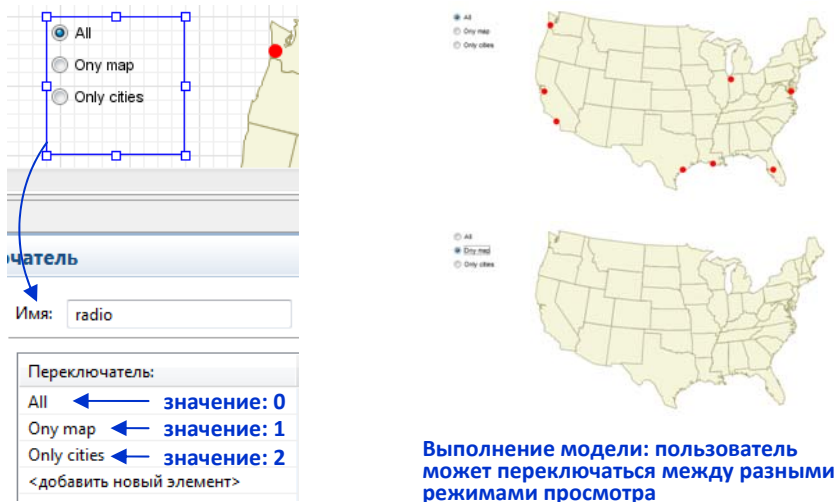
Давайте создадим переключатель, который будет как часть интерфейса изменять режим просмотра модели с картой США. Предположим, что мы хотим дать пользователю возможность выбора одного из следующих трех видов просмотра, при которых отображается: а) карта с основными городами, б) только карта или с) только города, без карты.

▶ **Выполните следующие шаги:**

1. Перетащите Карту США из палитры Картинки на диаграмму.
2. Нарисуйте маленький красный круг, перетащив его из палитры Презентация, изменив размер на 10 пикселей и задав красный цвет заливки.
3. Поместите кружок на тихоокеанском побережье, в южной Калифорнии, примерно в месте нахождения Лос-Анджелеса.
4. Увеличьте масштаб диаграммы и создайте копии кружка (перетащив его несколько раз с нажатой клавишей Ctrl) на месте городов Сан-Франциско, Нью-Йорк, Орlando, Хьюстон (или любых других).
5. Выделите все кружки (перетащив мышью с нажатой правой кнопкой вокруг области с этими фигурами), но не включайте в область выделения карту. Если карта все-таки будет выделена, то исключите ее из выделенных фигур, щелкнув по ней с нажатой клавишей Shift.
6. Щелкните правой кнопкой мыши по одному из кружков и выберите Группировка | Создать группу из контекстного меню. Все кружки должны попасть в одну группу.
7. Откройте палитру Элементы управления и перетащите элемент Переключатель на диаграмму, слева от карты.
8. На странице свойств переключателя Основные задайте следующий список предлагаемых пользователю вариантов: All (Все), Only map (Только карта), Only cities (Только города).
9. Выделите карту и введите в поле Видимость на странице свойств Динамические: `radio.getValue() != 2`.
10. Выделите группу городов (щелкнув по одному из кружков) и введите в поле Видимость на странице свойств Динамические: `radio.getValue() != 1`.
11. Запустите модель. Попробуйте различные режимы просмотра.

В этом примере переключатель не привязан ни к каким переменным, но его API, а именно - метод `getValue()`, используется в динамических свойствах фигур, управляющих их видимостью.

Значение переключателя является не строкой, а целым числом. Нумерация кнопок начинается с нуля (см. рисунок).



Переключатель, управляющий видимостью карты и городов

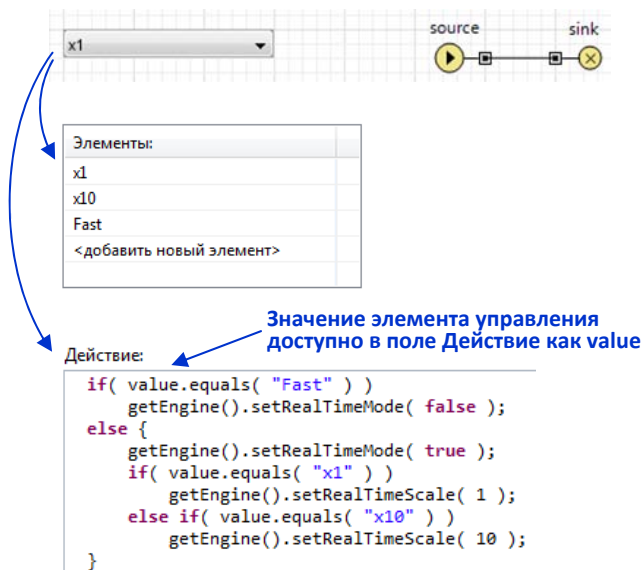
Пример: Выпадающий список, управляющий скоростью выполнения модели (Combo box controlling the simulation speed)

Мы создадим выпадающий список для управления скоростью выполнения модели. Хотя в нашем распоряжении и есть специальная панель инструментов Управление режимом времени, преимущество такого элемента управления будет состоять в том, что с его помощью мы сможем ограничить набор возможных скоростей выполнения определенными значениями, имеющими смысл для данной конкретной модели.

► **Выполните следующие шаги:**

1. Перетащите Выпадающий список из палитры Элементы управления на диаграмму и увеличьте его размер, как показано на рисунке.
2. На странице свойств выпадающего списка Основные введите следующие значения в списке Элементы:
 x1 (real time)
 x10
 As fast as possible
3. На той же странице свойств введите следующий код в поле Действие:


```
if( value.equals( "As fast as possible" ) )
    getEngine().setRealTimeMode( false );
else {
    getEngine().setRealTimeMode( true );
    if( value.equals( "x1 (real time)" ) )
        getEngine().setRealTimeScale( 1 );
    else if( value.equals( "x10" ) )
        getEngine().setRealTimeScale( 10 );
}
```
4. Чтобы заметить изменения скорости выполнения модели, создайте простейшую процессную модель, состоящую из объекта **Source**, соединенного с объектом **Sink** (перетащите оба объекта из палитры Основная библиотека и соедините их порты).
5. В панели Проекты выделите в дереве эксперимент модели **Simulation**. На странице свойств эксперимента Модельное время выберите из выпадающего списка Остановить опцию Нет.
6. Запустите модель. Задайте различные скорости с помощью выпадающего списка и наблюдайте за скоростью создания заявок.



Выпадающий список - нестандартный элемент управления скоростью выполнения

Значение выпадающего списка является строкой, поэтому в поле Действие мы сравниваем значение с тремя разными константами типа **String**. Объяснение того, почему строки сравниваются с помощью метода `equals()`, а не оператора `==`, вы можете найти в главе по [Java](#).

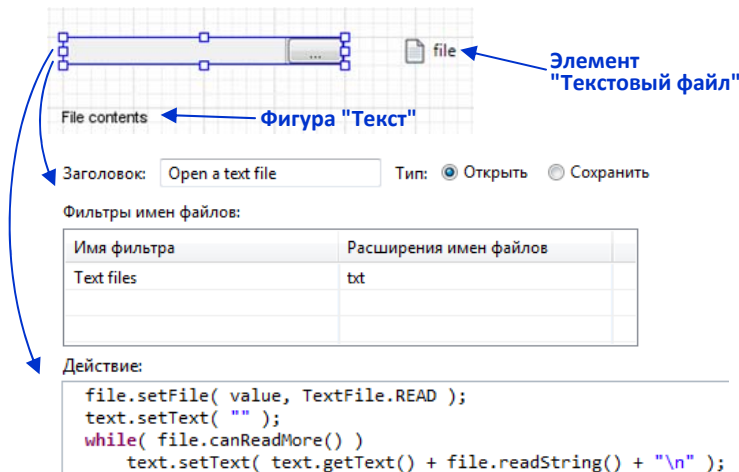
Пример: Элемент выбора текстовых файлов (File chooser for text files)

В этом примере Элемент выбора файла будет совместно работать с элементом Текстовый файл таким образом, что пользователь модели сможет выбрать файл (например, со значениями параметров модели) перед запуском модели.

► Выполните следующие шаги:

1. Перетащите Элемент выбора файла из палитры Элементы управления на диаграмму и немного увеличьте его размер, как показано на рисунке.
2. Откройте палитру Внешние данные и перетащите элемент [Текстовый файл](#) на диаграмму, поместив его справа от элемента выбора файла.
3. Откройте палитру Презентация и перетащите элемент Текст под элемент выбора файла, как показано на рисунке. Смените размер шрифта текста на 11 п.
4. На странице свойств Основные элемента выбора файла задайте в качестве Заголовка **Open a text file** (Открыть текстовый файл).
5. На этой же странице свойств добавьте фильтр файлов со следующими свойствами:
Имя фильтра: **Text files** (Текстовые файлы)
Расширения имен файлов: **txt**
6. На странице свойств элемента выбора файла введите следующий код в поле Действие:

```
file.setFile( value, TextFile.READ );
text.setText( "" );
while( file.canReadMore() )
    text.setText( text.getText() + file.readString() + "\n" );
```
7. Запустите модель.
8. Щелкните по кнопке элемента выбора файла. Выберите любой текстовый файл и нажмите по кнопке **Open**. Содержимое файла отобразится на экране.



Элемент выбора файла, работающий с элементом Текстовый файл

Элемент выбора файла настраивается следующим образом. Заданный заголовок становится заголовком диалогового окна **Open file** (Открыть файл). Имя фильтра сообщает диалогу о том, что открывать следует только файлы с расширением "txt". Когда пользователь выбирает файл, выполняется действие элемента выбора файла. Оно связывает выбранный файл с Текстовым файлом `file` – специальным элементом, который может производить чтение и запись в текстовые файлы. Затем текст фигуры `text` очищается (путем присвоения пустой строки ""), после чего все содержимое текстового файла строка за строкой добавляется в эту фигуру ("\\n" является символом конца строки).

❖ Неделимость действий элементов управления и событий модели

Связанные с элементами управления действия порождаются пользователем и могут выполняться во время выполнения модели. Поэтому у вас может возникнуть логичный вопрос - а как эти действия синхронизируются с событиями модели?

AnyLogic гарантирует неделимость всех действий элементов управления и всех событий модели. Это означает, что код действий элементов управления выполняется непрерывно (является атомарным) и никогда не прерывается выполнением событий модели (и наоборот). Допустим, пользователь изменяет состояние элемента управления во время выполнения непрерывной фазы модели, например, во время численного решения динамических уравнений. В этом случае действие элемента управления рассматривается как просто еще одно дискретное событие, так что решатель корректно останавливается перед выполнением действия элемента управления и возобновляет свою работу после его выполнения.

Динамические свойства элементов управления

Как и у фигур, динамические свойства элементов управления постоянно вычисляются заново во время выполнения модели, и это может быть использовано для динамического изменения доступности элемента управления, его видимости, размера, местоположения и т.д. Почти все динамические свойства находятся на странице свойств Динамические. Некоторые наиболее часто используемые располагаются на странице Основные.

Почти у всех элементов управления есть свойство Доступность. Если вы зададите в нем логическое выражение, то элемент управления будет доступен только тогда, когда это выражение будет возвращать **true**, иначе он будет недоступен (см. пример [Buttons](#)). Расположенное на странице Динамические свойство Видимость используется для временного скрытия элементов управления. Свойства, отвечающие за размер элемента, используются редко, поскольку обычно нет необходимости в динамическом изменении размера элементов управления во время выполнения модели.

Пример: Переключатель, разрешающий/запрещающий доступ к другим элементам управления (Radio buttons enabling/disabling other controls)

Вы можете добавить в модель "продвинутое" поведение, наподобие задаваемого в диалогах, связав свойства Видимость и/или Доступность одних элементов управления с состояниями других элементов управления. Предположим, вы хотите предложить пользователю на выбор два режима: использовать параметры, заданные по умолчанию, или же задать какие-то другие значения. С помощью переключателя вы можете разрешать/запрещать доступ к параметрам модели.

► Выполните следующие шаги:

1. Создайте переключатель, состоящий из двух кнопок, перетащив элемент Переключатель из палитры Элементы управления.
2. В свойствах переключателя задайте в качестве метки первой кнопки "Use default settings" (Использовать настройки по умолчанию), а в качестве метки второй – "Use custom settings" (Использовать другие настройки).
3. Создайте под переключателем бегунок, как показано на рисунке. Вам может понадобиться изменить размер переключателя для того, чтобы избежать его наложения на бегунок.
4. Создайте параметр справа от бегунка. Задайте значение параметра по умолчанию равным **50**.
5. На странице свойств бегунка Основные задайте следующие свойства:
Связать с: флажок установлен, значение: **parameter**
Доступность: **radio.getValue() == 1**
6. На странице свойств переключателя Основные введите следующий код в поле Действие:

```
if( value == 0 )
    set_parameter( 50 );
else
    set_parameter( slider.getValue() );
```

7. Запустите модель. Поиграйте с кнопками и бегунком и проследите за тем, как меняется значение параметра.



Доступность: `radio.getValue() == 1`

Переключатель, разрешающий и запрещающий доступ к бегунку

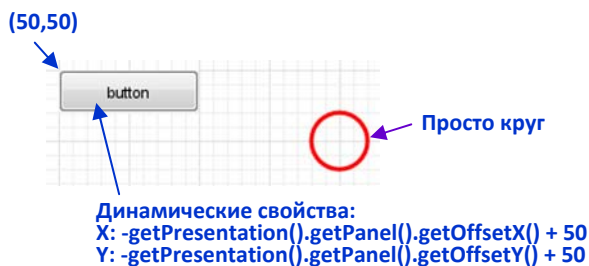
Обратите внимание, что бегунок запоминает свою позицию на время переключения в первый режим. И при переключении ко второму режиму параметру присваивается значение бегунка.

❖ Пример: Сохранение элементов управления в левом верхнем углу окна (Keeping controls in the top left corner of the window)

Предположим, что в вашей модели есть один или несколько элементов управления, и вы хотите, чтобы они отображались в верхнем левом углу окна модели вне зависимости от того, промотал ли пользователь холст презентации модели или же перешел к какой-либо области просмотра. Чтобы добиться такого поведения, мы воспользуемся динамическими свойствами X и Y элемента управления (или группы – как вы помните, элементы управления можно группировать так же, как и фигуры).

▶ **Выполните следующие шаги:**

1. Добавьте кнопку на диаграмму и переместите ее так, чтобы ее координаты стали равны (50,50).
2. Нарисуйте рядом кружок – он нужен нам для индикации того, как мы будем проматывать холст презентации.
3. На странице свойств кнопки Динамические задайте:
 X: `-getPresentation().getPanel().getOffsetX() + 50`
 Y: `-getPresentation().getPanel().getOffsetY() + 50`
4. Запустите модель. Перетащите холст, перемещая мышью с нажатой правой кнопкой мыши. Кружок при этом переместится, а кнопка - нет.



Кнопка будет всегда оставаться в точке (50,50) от левого верхнего угла окна модели

Выражение `getPresentation().getPanel().getOffsetX()` возвращает смещение начала координат презентации (0,0) по оси X от верхнего левого угла окна модели. Поэтому, чтобы сохранить кнопку в точке с координатой X=50, нам нужно сместить ее на то же расстояние в обратном направлении и добавить 50 (что является первоначальным (статическим) смещением кнопки). То же самое нужно сделать и для оси Y. Вы можете расширить этот пример, добавив несколько [областей просмотра](#) и переключаясь между ними.

❖ Пример: Реплицированная кнопка (Replicated button)

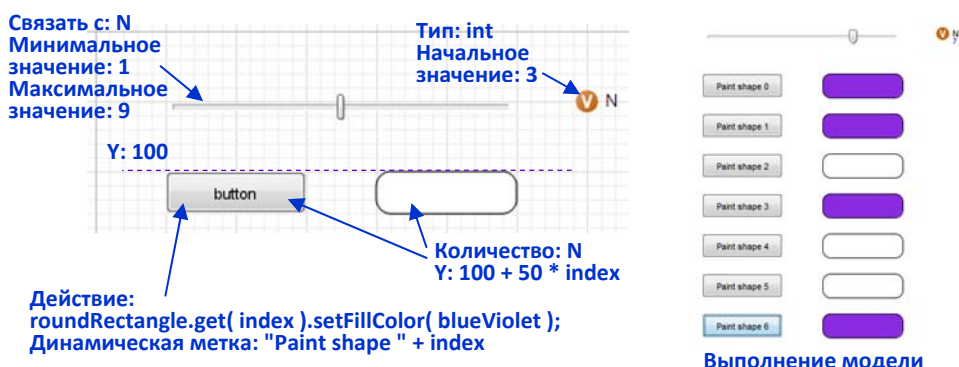
Иногда вам может понадобиться создать массив схожих элементов управления, например, для того, чтобы иметь возможность изменять массив сходных элементов во время выполнения модели. Используя *реплицированные элементы управления* AnyLogic, вы можете сэкономить время, требуемое на рисование элементов, а также сделать вашу модель масштабируемой. В этом примере мы создадим реплицированную кнопку и будем с ее помощью изменять цвет заливки *реплицированной фигуры*. Более того, мы будем динамически управлять количеством копий кнопки и фигуры с помощью бегунка.

▶ Выполните следующие шаги:

1. Создайте четыре элемента, как показано на рисунке: бегунок, переменную, кнопку и скругленный прямоугольник.
2. Переименуйте переменную в **N**, смените ее тип на целочисленный (**int**), а в качестве начального значения задайте **3**.
3. Свяжите бегунок с переменной **N** и задайте его минимальное и максимальное значения равными **1** и **9** соответственно.
4. Задайте для скругленного прямоугольника следующие динамические свойства:
Количество: **N**
Y: **100 + 50 * index**
5. Задайте для кнопки следующие динамические свойства:
Количество: **N**
Y: **100 + 50 * index**
Метка: **"Paint shape " + index**
6. На странице свойств кнопки Основные задайте Действие:
`roundRectangle.get(index).setFillColor(blueViolet);`
7. Запустите модель. Переместите бегунок и нажимайте кнопки.

Как вы можете видеть, количество кнопок меняется согласно тому, как бегунок изменяет значение переменной **N**. С помощью номера копии кнопки, доступного в поле Действие как переменная **index**, вы можете производить разные действия с различными кнопками. В нашем случае мы изменяем цвет фигуры, чей номер равен номеру кнопки. Метка кнопки также зависит от ее номера.

Если вы закрасите фигуру, скажем, с номером 8, затем смените количество фигур на 5, после чего снова сделаете его равным 9, то эта 8-я фигура не будет закрасена, потому что это будет уже новый объект. Прежняя же фигура будет к тому моменту полностью удалена.



Реплицированная кнопка, управляющая реплицированной фигурой

Программный интерфейс элементов управления

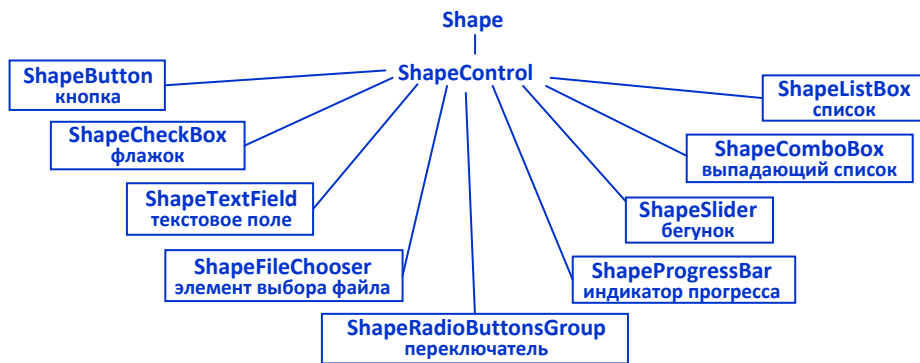
Как и все в AnyLogic, элементы управления транспонируются в Java объекты и предоставляют разработчику моделей свой программный интерфейс (API). Во многих случаях вы можете добиться одного и того же результата как с помощью динамических свойств элемента

управления, так и с помощью вызова его методов. Сделать выбор в пользу того или другого способа вам может помочь специальное [отступление](#) в разделе про программный интерфейс фигур.

Наиболее часто используются следующие методы элементов управления:

- `setVisible(boolean)` – отображает или прячет элемент управления
- `setEnabled(boolean)` – разрешает или запрещает доступ к элементу управления
- `action()` – выполняет заданное для элемента управления действие
- `setValueToDefault()` – задает значение элемента управления равным заданному по умолчанию значению
- `setValue(..., boolean)` – присваивает элементу управления переданное значение, при этом может также выполнить заданное для элемента управления действие

Полный список методов можно найти в [Справочнике Java классов AnyLogic](#). На приведенном ниже рисунке изображена иерархия Java классов элементов управления. Обратите внимание, что базовым классом для всех элементов управления является `ShapeControl`, подкласс класса `Shape`, поэтому элементы управления реализуют и методы класса `Shape`.

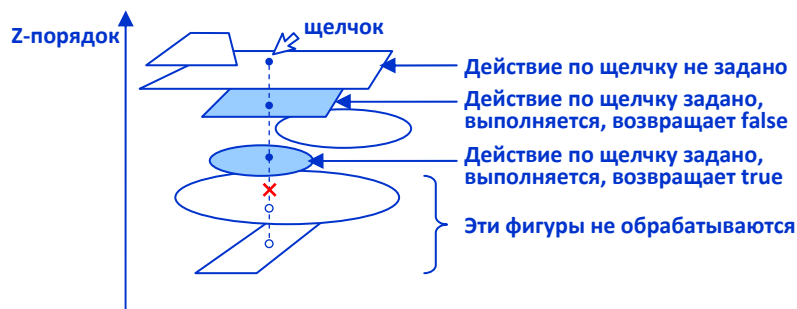


Java классы AnyLogic для элементов управления

Обработка щелчков мышью

Обработка щелчков мыши является еще одним способом добавления интерактивности в модель. С помощью обработки щелчков можно отображать дополнительную информацию по объектам (см. пример [Callout](#)), создавать гиперссылки, задавать местоположения объектов на карте, управлять определенными элементами модели и так далее.

Щелчки мыши в окне модели обрабатываются следующим образом. AnyLogic проходит в цикле по всем фигурам согласно их [Z-порядку](#), начиная с самой верхней фигуры. Если область фигуры включает в себя точку щелчка, и для этой фигуры задано Действие по щелчку, то это действие выполняется (и по умолчанию возвращает `false`). Если действие возвращает `true` (что вам нужно будет сделать самостоятельно), то обработка щелчка прекратится. Иначе обработка фигур продолжится, пока не дойдет до самой нижней фигуры, см. рисунок.



Обработка щелчков мышью в AnyLogic

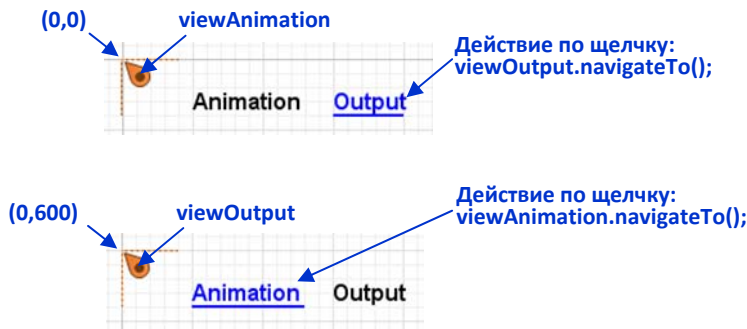
Точные координаты щелчка относительно координат фигуры доступны в коде Действия по щелчку как **clickx** (координата по оси X) и **clicky** (координата по оси Y).

Пример: Меню гиперссылок для навигации между областями просмотра (Hyper link menu to navigate between view areas)

Обработка щелчков часто используется для реализации дополнительных средств навигации по модели. Разработчики моделей добавляют текст или графику и делают их гиперссылками, осуществляющими переход к различным важным областям презентации модели. В данном примере мы покажем, как создать простое меню гиперссылок для переключения между двумя областями, помеченными **областями просмотра**. Одна область может содержать анимацию модели, а другая - отображать результаты работы модели.

► Выполните следующие шаги:

1. Создайте новую модель, не используя при этом готовые шаблоны моделей.
2. Создайте одну область просмотра в точке начала координат (0,0) диаграммы класса Main. Элемент Область просмотра находится в палитре Презентация.
3. На странице свойств Основные области просмотра задайте:
Имя: **viewAnimation**
Заголовок: **Animation**
4. Создайте еще одну область просмотра в точке (0,600).
5. Задайте для этой области просмотра:
Имя: **viewOutput**
Заголовок: **Output**
6. Нарисуйте круг или любую другую графическую фигуру в центре области просмотра **Animation**, т.е. в точке (400, 300), просто для того, чтобы идентифицировать эту область во время выполнения модели.
7. С той же целью перетащите временной график или любую другую диаграмму из палитры Статистика в центр области просмотра **Output**, т.е. в точку (400, 900).
8. Добавьте два текста (фигуры "Текст"): "Animation" (Анимация) в точке (50,20) и "Output" (Результаты) в точке (150,20). Смените шрифт на Полужирный, размера 16.
9. Смените цвет текста "Output" на **blue** (синий).
10. На странице свойств Динамические текста "Output" введите следующий код в поле Действие по щелчку: **viewOutput.navigateTo();**
11. Нарисуйте синюю линию под текстом "Output", чтобы он выглядел как гиперссылка, см. рисунок.
12. Выделите оба текста, "Animation" и "Output", а также синюю линию. Перетащите выделенные элементы с нажатой клавишей Ctrl, чтобы создать их копию.
13. Перетащите созданную копию элементов на вторую область просмотра, так, чтобы координаты текста "Animation" стали равны (50, 620).
14. Смените цвет этого текста "Animation" на синий, а соседнего текста "Output" – на черный.
15. Переместите синюю линию от текста "Output" к тексту "Animation" и растяните ее, чтобы она соответствовала длине текста.
16. Вырежьте код из поля Действие по щелчку со страницы свойств Динамические этого текста "Output" в аналогичное поле текста "Animation" text и измените его на: **viewAnimation.navigateTo();**
17. Запустите модель. Щелкните по созданным вами гиперссылкам.



Навигация между областями просмотра с помощью гиперссылок

Синие тексты в этом примере будут реагировать на щелчки мышь. Их Действия по щелчку будут вызывать методы `navigateTo()` областей просмотра, отображающих соответствующие участки диаграммы.

Если вам не нужно заботиться о дальнейшей обработке щелчка (как, например, в том случае, когда под фигурами, реагирующими на щелчки, нет других фигур, или если такие фигуры сами на щелчки не реагируют), вы можете не добавлять оператор `return` в конец кода Действия по щелчку – как мы, собственно, и сделали в этом примере.

❖ Пример: Рисование точек в местах щелчков (Creating dots at the click coordinates)

Вы можете узнать не только фигуру, по которой щелкнул пользователь модели, но и точные координаты щелчка (относительно фигуры). В данном примере мы воспользуемся этой возможностью для рисования небольших точек в местах щелчков мыши.

▶ Выполните следующие шаги:

1. Добавьте прямоугольник размером 500 x 500. Пусть его верхний левый угол находится примерно в точке (50,50).
2. Назовите прямоугольник `clickArea`.
3. Напишите следующий код в поле Действие по щелчку на странице свойств Динамические прямоугольника:

```
ShapeOval dot = new ShapeOval();
dot.setRadius( 2 );
dot.setFillColor( blue );
dot.setLineColor( null );
dot.setPos( clickArea.getX() + clickx, clickArea.getY() + clicky );
presentation.add( dot );
```

4. Запустите модель. Щелкните в области прямоугольника.

Переменные `clickx` и `clicky`, доступные в поле Действие по щелчку, являются координатами щелчка *относительно начала координат фигуры, которая обрабатывает этот щелчок*. Чтобы преобразовать их в абсолютные координаты, нам нужно сложить их с координатами самой фигуры (полагая при этом, что фигура принадлежит группе верхнего уровня `presentation`).

❖ Пример: Обработка щелчков мыши по холсту (Catching mouse clicks anywhere on the canvas)

Как вы уже знаете, щелчки мыши обрабатываются фигурами. А что если нам понадобится обрабатывать и щелчки, которые производятся в произвольных местах холста презентации модели? Одним из возможных решений является программное создание одной очень большой невидимой фигуры, которая и будет обрабатывать щелчки.

► **Выполните следующие шаги:**

1. Нарисуйте где-нибудь на диаграмме небольшой круг (радиусом 5 пикселей). Оставьте заданное по умолчанию имя **oval**. С помощью этого кружка мы будем показывать место щелчка.
2. Напишите следующий код в поле **Дополнительный код класса** на странице свойств **Дополнительные классы** активного объекта:

```
class ClickDetector extends ShapeRectangle {

    ClickDetector() {
        super( true, -100000, -100000, 0, null, null,
              200000, 200000, 0, LINE_STYLE_SOLID );
    }

    @Override
    public boolean onClick( double clickx, double clicky ) {
        clickx += getX();
        clicky += getY();
        oval.setPos( clickx, clicky ); //Код обработки щелчка
        return false;
    }

}
```

3. В поле **Действие при запуске** на странице свойств **Основные классы** активного объекта напишите: `presentation.add(new ClickDetector());`
4. Запустите модель. Щелкните мышью в различных местах презентации модели.

Поскольку фигуры в AnyLogic можно создавать динамически, ничто не мешает нам создать очень большую фигуру, которая будет покрывать всю значимую область презентации модели. Чтобы заставить эту фигуру реагировать на щелчки мыши, мы создадим свой подкласс класса **ShapeRectangle** и переопределим его метод **onClick()**, в котором и зададим реакцию на щелчок – в данном примере мы будем просто перемещать кружок в точку щелчка. В коде действия при запуске мы создаем реагирующий на щелчки прямоугольник (экземпляр класса **ClickDetector**) и добавляем его в находящуюся на верхнем уровне группу **presentation**.