Набор свойств для разных компонентов различен. В частности, у невидимых объектов нет ни «ширины», ни «высоты».

События (*Events*) — набор воздействий на компонент, вызывающих выполнение им некоторых программных операций. Эти воздействия могут быть как от внешних устройств (например, `OnMouseDown` — нажатие клавиши мыши в момент, когда указатель мыши находится над компонентом), так и программными (например, `OnCreate` — действия, которые нужно выполнить при создании формы).

Методы (*Methods*) — набор процедур и функций, которые предстоит выполнить компоненту в ответ на соответствующее событие. Программист с помощью окна `Object Inspector` обычно записывает код лишь в нужные процедуры, оставляя незаполненными ненужные реакции на события. В более сложном варианте можно программировать «акциями» (первая строка на закладке `Events` в окне `Object Inspector`). В этом случае все методы программируются отдельно и просто назначаются данному компоненту.

Еще две характеристики — *Сообщения* (*Messages*) и *Контекст* (*Context*) — определяют правила передачи управляющей информации между компонентами и режим графического отображения компонентов. В данном учебном пособии они не будут рассматриваться, поскольку их программирование выходит далеко за рамки элементарного визуального программирования. Однако, если вас заинтересуют какие-либо специальные пакеты в Delphi (например, OpenGL), программу придется писать с использованием сообщений.

Первые шаги в программировании визуальных компонентов мы сделаем, написав небольшую программу, которая сведется к назначению некоторых свойств ограниченного числа компонентов и программированию элементарных методов.

В качестве примера рассмотрим создание диалога с помощью визуального компонента `SpeedButton` ( на закладке `Additional`). Допустим, нам необходимо создать программу, в которой на форму вынесены две большие кнопки с картинками (именно для этого и используется компонент `SpeedButton`). На одной кнопке изображена кошка, на другой — мышка. Вначале кошка стоит напротив мышиной норки, откуда торчит только мышиный хвостик. Когда пользователю надоедает ждать, он может нажать на кнопку «Кошка». Кошка мяукает и ложится спать. Из норки появляется мордочка мыши. Если теперь нажать на кнопку «Мышка», мышь выскакивает из норки, видит кошку, пугается и прячется в норку, оставив снаружи только хвостик. При этом кошка просыпается и опять стоит у норки. Все четыре фазы действия этой программы показаны на рис. 4.3.

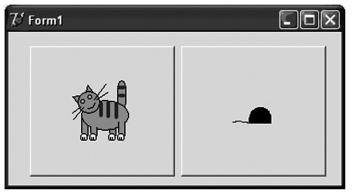
Если отвлечься от образного описания задачи, можно заключить: программист сталкивается здесь с проблемой трехкратной замены картинок на кнопках. Картинка кошки: в активном состоянии она «стоит», в недоступном состоянии — «спит», в нажатом состоянии — «мяукает». Картинка мышки: в активном состоянии она «выглядывает», в недоступном — «спряталась», в нажатом — «выбегает». Возможность подобной замены картинок уже предусмотрена самим компонентом `SpeedButton`. В данном случае, хотя это и может показаться странным, написание программы заключается в основном не в записи программного кода, а в работе в графическом редакторе. С него и начнем.

Компонент `SpeedButton` имеет свойство `Glyph`, в которое вписывается имя файла, содержащего картинку для вывода на кнопку. Определение этого свойства следует производить, когда картинка будет готова. Особенность компонента заключается в том, что в разных режимах он может выводить одну, две или три картинки в зависимости от состояния кнопки.

Картинки не масштабируются. Поэтому, если наши кнопки, например, имеют размеры 50×50, в графическом редакторе следует рисовать картинки высотой 45 пиксел и шириной 135 пиксел (т. е. трижды по 45 пиксел). Удобнее это сделать, нарисовав сначала три одноцветные рамочки (рис. 4.4), которые потом будут стерты.

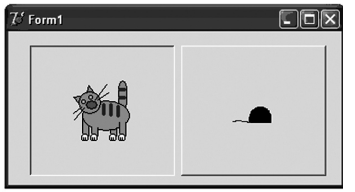
Порядок рисования картинки определяется компонентом `SpeedButton`: ее левый фрагмент выводится при ненажатой активной кнопке, средний — при отключенной (свойство «доступность» `Enabled` установлено в состояние `false`), правый — временно выводится в момент нажатия кнопки.

Чтобы картинка не смещалась при нажатии кнопки, левый и правый ее фрагменты лучше копировать, изменяя только часть деталей (рис. 4.5). По окончании рисования опорные рамочки нужно стереть.

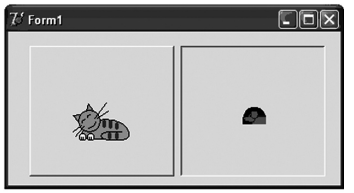


Исходное состояние

После нажатия кнопки «Кошка»



Нажатие кнопки «Кошка»



Нажатие кнопки «Мышка»

Рис. 4.3. Реакция программы на разные нажатия кнопок

Рис. 4.4. Прорисовка рамочек для выполнения трех равных фрагментов рисунка

Рис. 4.5. Рисунок кошки (три фрагмента), подготовленный для вставки в `Glyph`