



ГЛАВА 20

Размещение компонентов в окнах

При размещении в окне нескольких компонентов возникает вопрос их взаимного расположения и минимальных размеров. Следует помнить, что по умолчанию размеры окна можно изменять, а значит, необходимо перехватывать событие изменения размеров и производить перерасчет позиции и размера каждого компонента. Библиотека PyQt избавляет нас от лишних проблем и предоставляет множество компонентов-контейнеров, которые производят такой перерасчет автоматически. Все, что от нас требуется, это выбрать нужный контейнер, добавить в него компоненты в определенном порядке, а затем поместить контейнер в окно или в другой контейнер.

20.1. Абсолютное позиционирование

Прежде чем изучать контейнеры, рассмотрим возможность абсолютного позиционирования компонентов в окне. Итак, если при создании компонента указана ссылка на родительский компонент, то он выводится в позицию с координатами (0, 0). Иными словами, если мы добавим несколько компонентов, то все они отобразятся в одной и той же позиции, наложившись друг на друга. Последний добавленный компонент окажется на вершине этой кучи, а остальные компоненты станут видны лишь частично или вообще не видны. Размеры добавляемых компонентов будут соответствовать их содержимому.

Для перемещения компонента можно воспользоваться методом `move()`, а для изменения размеров — методом `resize()`. Выполнить одновременное изменение позиции и размеров позволяет метод `setGeometry()`. Все эти методы, а также множество других, позволяющих изменять позицию и размеры, мы уже рассматривали в разд. 18.3 и 18.4. Если компонент не имеет родителя, эти методы изменяют характеристики окна, а если родительский компонент был указан, они изменяют характеристики только самого компонента.

Для примера выведем внутри окна надпись и кнопку, указав позицию и размеры для каждого компонента (листинг 20.1).

Листинг 20.1. Абсолютное позиционирование

```
# -*- coding: utf-8 -*-
from PyQt5 import QtWidgets
import sys
app = QtWidgets.QApplication(sys.argv)
window = QtWidgets.QWidget()
window.setWindowTitle("Абсолютное позиционирование")
```

```
window.resize(300, 120)
label = QtWidgets.QLabel("Текст надписи", window)
button = QtWidgets.QPushButton("Текст на кнопке", window)
label.setGeometry(10, 10, 280, 60)
button.resize(280, 30)
button.move(10, 80)
window.show()
sys.exit(app.exec_())
```

Абсолютное позиционирование имеет следующие недостатки:

- ◆ при изменении размеров окна необходимо самостоятельно пересчитывать и изменять характеристики всех компонентов в программном коде;
- ◆ при указании фиксированных размеров надписи на компонентах могут выходить за их пределы. Помните, что в разных операционных системах используются разные стили оформления, в том числе и характеристики шрифта. Подогнав размеры в одной операционной системе, можно прийти в ужас при виде приложения в другой операционной системе, где размер шрифта в два раза больше. Поэтому лучше вообще отказаться от указания фиксированных размеров или задавать размер и название шрифта для каждого компонента явно. Кроме того, приложение может поддерживать несколько языков интерфейса, а поскольку длина слов в разных языках различается, это также станет причиной искажения компонентов.

20.2. Горизонтальное и вертикальное выравнивание

Компоненты-контейнеры (их еще называют менеджерами компоновки и менеджерами геометрии) лишены недостатков абсолютного позиционирования. При изменении размеров окна производится автоматическое изменение характеристик всех компонентов, добавленных в контейнер. Настройки шрифта при этом также учитываются, поэтому изменение размеров шрифта в два раза приведет только к увеличению компонентов и окон.

Для автоматического выравнивания компонентов используются два класса:

- ◆ QHBoxLayout — выстраивает все добавляемые компоненты по горизонтали (по умолчанию — слева направо). Конструктор класса имеет следующий формат:
`<Объект> = QHBoxLayout ([<Родитель>])`
- ◆ QVBoxLayout — выстраивает все добавляемые компоненты по вертикали (по умолчанию — сверху вниз). Формат конструктора класса:
`<Объект> = QVBoxLayout ([<Родитель>])`

Иерархия наследования для этих классов выглядит так:

```
(QObject, QLayoutItem) – QLayout – QHBoxLayout – QVBoxLayout
(QObject, QLayoutItem) – QLayout – QVBoxLayout – QVBoxLayout
```

Обратите внимание, что указанные классы не являются наследниками класса QWidget, а следовательно, не обладают собственным окном и не могут использоваться отдельно. Поэтому контейнеры обязательно должны быть привязаны к родительскому компоненту. Передать ссылку на родительский компонент можно в конструкторе классов QHBoxLayout и QVBoxLayout. Кроме того, можно передать ссылку на контейнер в метод setLayout() роди-

тельского компонента. После этого все компоненты, добавленные в контейнер, автоматически привязываются к родительскому компоненту.

Типичный пример использования класса `QHBoxLayout` показан в листинге 20.2, а увидеть результат выполнения этого кода можно на рис. 20.1.

Листинг 20.2. Использование контейнера `QHBoxLayout`

```
# -*- coding: utf-8 -*-
from PyQt5 import QtWidgets
import sys
app = QtWidgets.QApplication(sys.argv)
window = QtWidgets.QWidget() # Родительский компонент – окно
window.setWindowTitle("QHBoxLayout")
window.resize(300, 60)
button1 = QtWidgets.QPushButton("1")
button2 = QtWidgets.QPushButton("2")
hbox = QtWidgets.QHBoxLayout() # Создаем контейнер
hbox.addWidget(button1) # Добавляем компоненты
hbox.addWidget(button2)
window.setLayout(hbox) # Передаем ссылку родителю
window.show()
sys.exit(app.exec_())
```

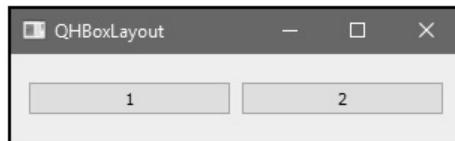


Рис. 20.1. Контейнер `QHBoxLayout` с двумя кнопками

Добавить компоненты в контейнер, удалить их и заменить другими позволяют следующие методы:

- ◆ `addWidget()` добавляет компонент в конец контейнера. Формат метода:
`addWidget(<Компонент>, stretch=0 [, alignment=0])`

В первом параметре указывается ссылка на компонент. Необязательный параметр `stretch` задает фактор растяжения для ячейки, а параметр `alignment` – выравнивание компонента внутри ячейки. Два последних параметра можно задавать в порядке следования или по именам в произвольном порядке:

```
hbox.addWidget(button1, 10, QtCore.Qt.AlignRight)
hbox.addWidget(button2, stretch=10)
hbox.addWidget(button3, alignment=QtCore.Qt.AlignRight)
```

- ◆ `insertWidget()` добавляет компонент в указанную позицию контейнера. Формат метода:

```
insertWidget(<Индекс>, <Компонент>, [stretch=0] [, alignment=0])
```

Если в первом параметре указано значение 0, компонент будет добавлен в начало контейнера, а если отрицательное значение, компонент добавляется в конец контейнера.

Иное значение указывает определенную позицию. Остальные параметры аналогичны параметрам метода `addWidget()`:

```
hbox.addWidget(button1)
hbox.insertWidget(-1, button2) # Добавление в конец
hbox.insertWidget(0, button3) # Добавление в начало
```

- ◆ `removeWidget(<Компонент>)` — удаляет указанный компонент из контейнера;
- ◆ `replaceWidget()` — заменяет присутствующий в контейнере компонент другим. Формат метода:

```
replaceWidget(<Заменяемый компонент>, <Заменяющий компонент>
              [, options= FindChildrenRecursively])
```

В необязательном параметре `options` можно задать режим поиска заменяемого компонента с помощью одного из атрибутов класса `QtCore.Qt`: `FindDirectChildrenOnly` (0, искать только среди содержимого текущего контейнера) и `FindChildrenRecursively` (1, искать среди содержимого текущего и всех вложенных в него контейнеров — поведение по умолчанию);

- ◆ `addLayout(<Контейнер>[, stretch=0])` — добавляет другой контейнер в конец текущего контейнера. С помощью этого метода можно вкладывать один контейнер в другой, создавая таким образом структуру любой сложности. Формат метода:

```
addLayout(<Контейнер>[, stretch=0])
```

- ◆ `insertLayout(<Индекс>, <Контейнер>[, stretch=0])` — добавляет другой контейнер в указанную позицию текущего контейнера. Если в первом параметре задано отрицательное значение, контейнер добавляется в конец. Формат метода:

```
insertLayout(<Индекс>, <Контейнер>[, stretch=0])
```

- ◆ `addSpacing(<Размер>)` — добавляет пустое пространство указанного размера в конец контейнера. Размер пустого пространства задается в пикселях:

```
hbox.addSpacing(100)
```

- ◆ `insertSpacing(<Индекс>, <Размер>)` — добавляет пустое пространство указанного размера в определенную позицию. Размер пустого пространства задается в пикселях. Если первым параметром передается отрицательное значение, то пространство добавляется в конец;

- ◆ `addStretch([stretch=0])` — добавляет пустое растягиваемое пространство с нулевым минимальным размером и фактором растяжения `stretch` в конец контейнера. Это пространство можно сравнить с пружиной, вставленной между компонентами, а параметр `stretch` — с жесткостью пружины;

- ◆ `insertStretch(<Индекс>[, stretch=0])` — метод аналогичен методу `addStretch()`, но добавляет растягиваемое пространство в указанную позицию. Если в первом параметре задано отрицательное значение, пространство добавляется в конец контейнера.

Параметр `alignment` в методах `addWidget()` и `insertWidget()` задает выравнивание компонента внутри ячейки. В этом параметре можно указать следующие атрибуты класса `QtCore.Qt`:

- ◆ `AlignLeft` — 1 — горизонтальное выравнивание по левому краю;
- ◆ `AlignRight` — 2 — горизонтальное выравнивание по правому краю;
- ◆ `AlignHCenter` — 4 — горизонтальное выравнивание по центру;

- ◆ AlignJustify — 8 — заполнение всего пространства;
- ◆ AlignTop — 32 — вертикальное выравнивание по верхнему краю;
- ◆ AlignBottom — 64 — вертикальное выравнивание по нижнему краю;
- ◆ AlignVCenter — 128 — вертикальное выравнивание по центру;
- ◆ AlignBaseline — 256 — вертикальное выравнивание по базовой линии;
- ◆ AlignCenter — AlignVCenter | AlignHCenter — горизонтальное и вертикальное выравнивание по центру;
- ◆ AlignAbsolute — 16 — если в методе `setLayoutDirection()` из класса `QWidget` указан атрибут `QtCore.Qt.RightToLeft`, атрибут `AlignLeft` задает выравнивание по правому краю, а атрибут `AlignRight` — по левому краю. Чтобы атрибут `AlignLeft` всегда соответствовал именно левому краю, необходимо указать комбинацию `AlignAbsolute | AlignLeft`. Аналогично следует поступить с атрибутом `AlignRight`.

Можно задавать и комбинации атрибутов. В них может присутствовать только один атрибут горизонтального выравнивания и только один атрибут вертикального выравнивания. Например, комбинация `AlignLeft | AlignTop` задает выравнивание по левому и верхнему краям. Противоречивые значения приводят к непредсказуемым результатам.

Помимо рассмотренных, контейнеры поддерживают также следующие методы (здесь приведены только основные — полный их список ищите в документации):

- ◆ `setDirection(<Направление>)` — задает направление вывода компонентов. В параметре можно указать следующие атрибуты из класса `QBoxLayout`:
 - `LeftToRight` — 0 — слева направо (значение по умолчанию для горизонтального контейнера);
 - `RightToLeft` — 1 — справа налево;
 - `TopToBottom` — 2 — сверху вниз (значение по умолчанию для вертикального контейнера);
 - `BottomToFront` — 3 — снизу вверх;
- ◆ `setContentsMargins()` — задает величины отступов от границ контейнера до компонентов. Форматы метода:


```
setContentsMargins(<Слева>, <Сверху>, <Справа>, <Снизу>)
setContentsMargins(<QMargins>)
```

Примеры:

```
hbox.setContentsMargins(2, 4, 2, 4)
m = QtCore.QMargins(4, 2, 4, 2)
hbox.setContentsMargins(m)
```

- ◆ `setSpacing(<Расстояние>)` — задает расстояние между компонентами.

20.3. Выравнивание по сетке

Помимо выравнивания компонентов по горизонтали и вертикали, существует возможность размещения компонентов внутри ячеек сетки. Для этого предназначен класс `QGridLayout`. Иерархия его наследования:

(QObject, QLayoutItem) — QLayout — QGridLayout

Создать экземпляр класса `QGridLayout` можно следующим образом:

```
<Объект> = QGridLayout([<Родитель>])
```

В необязательном параметре можно указать ссылку на родительский компонент. Если такой не указан, следует передать ссылку на сетку в метод `setLayout()` родительского компонента. Код, иллюстрирующий типичный пример использования класса `QGridLayout`, представлен в листинге 20.3, а результат его выполнения — на рис. 20.2.

Листинг 20.3. Использование контейнера `QGridLayout`

```
# -*- coding: utf-8 -*-
from PyQt5 import QtWidgets
import sys
app = QtWidgets.QApplication(sys.argv)
window = QtWidgets.QWidget() # Родительский компонент — окно
window.setWindowTitle("QGridLayout")
window.resize(150, 100)
button1 = QtWidgets.QPushButton("1")
button2 = QtWidgets.QPushButton("2")
button3 = QtWidgets.QPushButton("3")
button4 = QtWidgets.QPushButton("4")
grid = QtWidgets.QGridLayout() # Создаем сетку
grid.addWidget(button1, 0, 0) # Добавляем компоненты
grid.addWidget(button2, 0, 1)
grid.addWidget(button3, 1, 0)
grid.addWidget(button4, 1, 1)
window.setLayout(grid) # Передаем ссылку родителю
window.show()
sys.exit(app.exec_())
```



Рис. 20.2. Компонент `QGridLayout` с четырьмя кнопками

Добавить компоненты позволяют следующие методы:

- ◆ `addWidget()` добавляет компонент в указанную ячейку сетки. Метод имеет следующие форматы:

```
addWidget(<Компонент>, <Строка>, <Столбец>[, alignment=0])
addWidget(<Компонент>, <Строка>, <Столбец>, <Количество строк>,
         <Количество столбцов>[, alignment=0])
```

В первом параметре указывается ссылка на компонент, во втором параметре — индекс строки, а в третьем — индекс столбца. Нумерация строк и столбцов начинается с нуля. Параметр `<количество строк>` задает количество занимаемых компонентом ячеек по

вертикали, а параметр <Количество столбцов> — по горизонтали. Параметр alignment задает выравнивание компонента внутри ячейки. Значения, которые можно указать в этом параметре, мы рассматривали в предыдущем разделе.

Пример:

```
grid = QtGui.QGridLayout()
grid.addWidget(button1, 0, 0, alignment=QtCore.Qt.AlignLeft)
grid.addWidget(button2, 0, 1, QtCore.Qt.AlignRight)
grid.addWidget(button3, 1, 0, 1, 2)
```

- ◆ addLayout() — добавляет контейнер в указанную ячейку сетки. Метод имеет следующие форматы:

```
addLayout(<Контейнер>, <Строка>, <Столбец>[, alignment=0])
addLayout(<Контейнер>, <Строка>, <Столбец>, <Количество строк>,
          <Количество столбцов>[, alignment=0])
```

В первом параметре указывается ссылка на контейнер. Остальные параметры аналогичны параметрам метода addWidget().

Для удаления и замены компонентов следует пользоваться методами removeWidget() и replaceWidget(), описанными в разд. 20.2.

Класс QGridLayout поддерживает следующие методы (здесь приведены только основные методы — полный их список смотрите на странице <https://doc.qt.io/qt-5/qgridlayout.html>):

- ◆ setRowMinimumHeight(<Индекс>, <Высота>) — задает минимальную высоту строки с индексом <Индекс>;
- ◆ setColumnMinimumWidth(<Индекс>, <Ширина>) — задает минимальную ширину столбца с индексом <Индекс>;
- ◆ setRowStretch(<Индекс>, <Фактор растяжения>) — задает фактор растяжения по вертикали для строки с индексом <Индекс>;
- ◆ setColumnStretch(<Индекс>, <Фактор растяжения>) — задает фактор растяжения по горизонтали для столбца с индексом <Индекс>;
- ◆ setContentsMargins() — задает величины отступов от границ сетки до компонентов. Форматы метода:
 - setContentsMargins(<Слева>, <Сверху>, <Справа>, <Снизу>)
 - setContentsMargins(<QMargins>)
- ◆ setSpacing(<Значение>) — задает расстояние между компонентами по горизонтали и вертикали;
- ◆ setHorizontalSpacing(<Значение>) — задает расстояние между компонентами по горизонтали;
- ◆ setVerticalSpacing(<Значение>) — задает расстояние между компонентами по вертикали;
- ◆ rowCount() — возвращает количество строк сетки;
- ◆ columnCount() — возвращает количество столбцов сетки;
- ◆ cellRect(<Индекс строки>, <Индекс колонки>) — возвращает экземпляр класса QRect, который хранит координаты и размеры ячейки, расположенной на пересечении строки и колонки с указанными индексами.

20.4. Выравнивание компонентов формы

Класс `QFormLayout` позволяет выравнивать компоненты формы. Контейнер по умолчанию состоит из двух столбцов: первый предназначен для вывода надписи, а второй — для вывода самого компонента. При этом надпись связывается с компонентом, что позволяет назначать клавиши быстрого доступа, указав символ & перед буквой внутри текста надписи. По нажатию комбинации клавиш быстрого доступа (комбинация `<Alt>+<буква>`) в фокусе окажется компонент, расположенный справа от надписи. Иерархия наследования выглядит так:

(QObject, QLayoutItem) — QLayout — QFormLayout

Создать экземпляр класса `QFormLayout` можно следующим образом:

```
<Объект> = QFormLayout([<Родитель>])
```

В необязательном параметре можно указать ссылку на родительский компонент. Если такой не указан, необходимо передать ссылку на контейнер в метод `setLayout()` родительского компонента.

В листинге 20.4 показан код, создающий форму с контейнером `QFormLayout`. Результат выполнения этого кода можно увидеть на рис. 20.3.

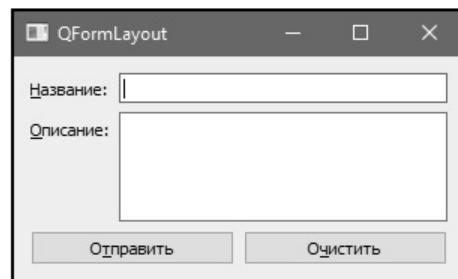


Рис. 20.3. Пример использования контейнера `QFormLayout`

Листинг 20.4. Использование контейнера `QFormLayout`

```
# -*- coding: utf-8 -*-
from PyQt5 import QtWidgets
import sys
app = QtWidgets.QApplication(sys.argv)
window = QtWidgets.QWidget()
window.setWindowTitle("QFormLayout")
window.resize(300, 150)
lineEdit = QtWidgets.QLineEdit()
textEdit = QtWidgets.QTextEdit()
button1 = QtWidgets.QPushButton("Отправить")
button2 = QtWidgets.QPushButton("Очистить")
hbox = QtWidgets.QHBoxLayout()
hbox.addWidget(button1)
hbox.addWidget(button2)
form = QtWidgets.QFormLayout()
form.addRow("&Название:", lineEdit)
```

```
form.addRow("&Описание:", textEdit)
form.addRow(hbox)
window.setLayout(form)
window.show()
sys.exit(app.exec_())
```

Класс `QFormLayout` поддерживает следующие методы (здесь приведены только основные — полный их список смотрите на странице <https://doc.qt.io/qt-5/qformlayout.html>):

- ◆ `addRow()` — добавляет строку в конец контейнера. Форматы метода:

```
addRow(<Текст надписи>, <Компонент> | <Контейнер>)
addRow(<QLabel>, <Компонент> | <Контейнер>)
addRow(<Компонент> | <Компонент>)
```

Первый формат позволяет задать текст надписи, которая будет помещена в первую колонку формы, и компонент или контейнер, помещаемый во вторую колонку. В тексте надписи можно указать символ `&`, который пометит клавишу быстрого доступа для этого компонента (контейнера). Второй формат использует в качестве надписи компонент класса `QLabel` (он представляет надпись) — в этом случае связь с компонентом (контейнером) необходимо устанавливать вручную, передав ссылку на него в метод `setBuddy()`. Третий формат заставляет компонент (контейнер) занять сразу обе колонки формы;

- ◆ `insertRow()` — добавляет строку в указанную позицию контейнера. Если в первом параметре задано отрицательное значение, компонент добавляется в конец контейнера. Форматы метода:

```
insertRow(<Индекс>, <Текст надписи>, <Компонент> | <Контейнер>)
insertRow(<Индекс>, <QLabel>, <Компонент> | <Контейнер>)
insertRow(<Индекс>, <Компонент> | <Контейнер>)
```

- ◆ `setFormAlignment(<Режим>)` — задает режим выравнивания формы (допустимые значения мы рассматривали в разд. 20.2):

```
form.setFormAlignment(
    QtCore.Qt.AlignRight | QtCore.Qt.AlignBottom)
```

- ◆ `setLabelAlignment(<Режим>)` — задает режим выравнивания надписи (допустимые значения мы рассматривали в разд. 20.2). Вот пример выравнивания по правому краю:

```
form.setLabelAlignment(QtCore.Qt.AlignRight)
```

- ◆ `setRowWrapPolicy(<Режим>)` — задает местоположение надписей. В качестве параметра указываются следующие атрибуты класса `QFormLayout`:

- `DontWrapRows` — 0 — надписи расположены слева от компонентов;
- `WrapLongRows` — 1 — длинные надписи могут находиться выше компонентов, а короткие надписи — слева от компонентов;
- `WrapAllRows` — 2 — надписи всегда расположены выше компонентов;

- ◆ `setFieldGrowthPolicy(<Режим>)` — задает режим управления размерами компонентов. В качестве параметра указываются следующие атрибуты класса `QFormLayout`:

- `FieldsStayAtSizeHint` — 0 — компоненты всегда будут принимать рекомендуемые (возвращаемые методом `sizeHint()`) размеры;
- `ExpandingFieldsGrow` — 1 — компоненты, для которых установлена политика изменения размеров `QSizePolicy.Expanding` или `QSizePolicy.MinimumExpanding`, займут

всю доступную ширину. Размеры остальных компонентов всегда будут соответствовать рекомендуемым;

- AllNonFixedFieldsGrow — 2 — все компоненты (если это возможно) займут всю доступную ширину;
- ◆ setContentsMargins() — задает величины отступов от границ сетки до компонентов. Форматы метода:
setContentsMargins(<Слева>, <Сверху>, <Справа>, <Снизу>)
setContentsMargins(<QMargins>)
- ◆ setSpacing(<Значение>) — задает расстояние между компонентами по горизонтали и вертикали;
- ◆ setHorizontalSpacing(<Значение>) — задает расстояние между компонентами по горизонтали;
- ◆ setVerticalSpacing(<Значение>) — задает расстояние между компонентами по вертикали.

Для удаления и замены компонентов следует пользоваться методами `removeWidget()` и `replaceWidget()`, описанными в разд. 20.2.

20.5. Классы `QStackedLayout` и `QStackedWidget`

Класс `QStackedLayout` реализует стек компонентов — в один момент времени показывается только один компонент, а вывод другого компонента выполняется программно. Иерархия наследования выглядит так:

(QObject, QLayoutItem) — QLayout — QStackedLayout

Создать экземпляр класса `QStackedLayout` можно следующим образом:

<Объект> = QStackedLayout([<Родитель>])

В необязательном параметре можно задать ссылку на родительский компонент или контейнер. Если параметр не указан, следует передать ссылку на контейнер в метод `setLayout()` родительского компонента.

Класс `QStackedLayout` поддерживает следующие методы:

- ◆ addWidget(<Компонент>) — добавляет компонент в конец контейнера. Метод возвращает индекс добавленного компонента;
- ◆ insertWidget(<Индекс>, <Компонент>) — добавляет компонент в указанную позицию контейнера. Метод возвращает индекс добавленного компонента;
- ◆ setCurrentIndex(<Индекс>) — делает видимым компонент с указанным индексом. Метод является слотом;
- ◆ currentIndex() — возвращает индекс видимого компонента;
- ◆ setCurrentWidget(<Компонент>) — делает видимым указанный компонент. Метод является слотом;
- ◆ currentWidget() — возвращает ссылку на видимый компонент;
- ◆ setStackingMode(<Режим>) — задает режим отображения компонентов. В параметре могут быть указаны следующие атрибуты из класса `QStackedLayout`:
 - StackOne — 0 — только один компонент видим (значение по умолчанию);
 - StackAll — 1 — видны все компоненты;

- ◆ `stackingMode()` — возвращает режим отображения компонентов;
- ◆ `count()` — возвращает количество компонентов внутри контейнера;
- ◆ `widget(<Индекс>)` — возвращает ссылку на компонент, который расположен по указанному индексу, или значение `None`.

Для удаления и замены компонентов следует пользоваться методами `removeWidget()` и `replaceWidget()`, описанными в разд. 20.2.

Класс `QStackedLayout` поддерживает следующие сигналы:

- ◆ `currentChanged(<Индекс>)` — генерируется при изменении видимого компонента. Через параметр внутри обработчика доступен целочисленный индекс нового компонента;
- ◆ `widgetRemoved(<Индекс>)` — генерируется при удалении компонента из контейнера. Через параметр внутри обработчика доступен целочисленный индекс удаленного компонента.

Класс `QStackedWidget` также реализует стек компонентов, но представляет собой компонент, а не контейнер. Иерархия наследования выглядит так:

`(QObject, QPaintDevice) — QWidget — QFrame — QStackedWidget`

Создать экземпляр этого класса можно следующим образом:

`<Объект> = QStackedWidget([<Родитель>])`

Класс `QStackedWidget` поддерживает методы `addWidget()`, `insertWidget()`, `removeWidget()`, `replaceWidget()`, `count()`, `currentIndex()`, `currentWidget()`, `widget()`, `setCurrentIndex()` и `setCurrentWidget()`, которые выполняют те же действия, что и одноименные методы в классе `QStackedLayout`. Кроме того, класс `QStackedWidget` наследует все методы из базовых классов и определяет два дополнительных:

- ◆ `indexOf(<Компонент>)` — возвращает индекс компонента, ссылка на который указана в параметре;
- ◆ `__len__()` — возвращает количество компонентов. Метод вызывается при использовании функции `len()`, а также для проверки объекта на логическое значение.

Чтобы отследить изменения внутри компонента, следует назначить обработчики сигналов `currentChanged` и `widgetRemoved`.

20.6. Класс `QSizePolicy`

Если в вертикальный контейнер большой высоты добавить надпись и кнопку, то кнопка займет пространство, совпадающее с рекомендуемыми размерами (которые возвращает метод `sizeHint()`), а под надпись будет выделено все остальное место. Управление размерами компонентов внутри контейнера определяется правилами, установленными с помощью класса `QSizePolicy`. Установить эти правила для компонента можно с помощью метода `setSizePolicy(<QSizePolicy>)` класса `QWidget`, а получить их — с помощью метода `sizePolicy()`.

Создать экземпляр класса `QSizePolicy` (он определен в модуле `QtWidgets`) можно следующим способом:

`<Объект> = QSizePolicy([<Правило для горизонтали>, <Правило для вертикали>, [<Тип компонента>]])`

Если параметры не заданы, размер компонента должен точно соответствовать размерам, возвращаемым методом `sizeHint()`. В первом и втором параметрах указывается один из следующих атрибутов класса `QSizePolicy`:

- ◆ `Fixed` — размер компонента должен точно соответствовать размерам, возвращаемым методом `sizeHint()`;
- ◆ `Minimum` — размер, возвращаемый методом `sizeHint()`, является минимальным для компонента и может быть увеличен при необходимости;
- ◆ `Maximum` — размер, возвращаемый методом `sizeHint()`, является максимальным для компонента и может быть уменьшен при необходимости;
- ◆ `Preferred` — размер, возвращаемый методом `sizeHint()`, является предпочтительным и может быть как увеличен, так и уменьшен;
- ◆ `Expanding` — размер, возвращаемый методом `sizeHint()`, может быть как увеличен, так и уменьшен. Компонент займет все свободное пространство в контейнере;
- ◆ `MinimumExpanding` — размер, возвращаемый методом `sizeHint()`, является минимальным для компонента. Компонент займет все свободное пространство в контейнере;
- ◆ `Ignored` — размер, возвращаемый методом `sizeHint()`, игнорируется. Компонент займет все свободное пространство в контейнере.

Изменить правила управления размерами уже после создания экземпляра класса `QSizePolicy` позволяют методы `setHorizontalPolicy(<Правило для горизонтали>)` и `setVerticalPolicy(<Правило для вертикали>)`.

С помощью методов `setHorizontalStretch(<Фактор для горизонтали>)` и `setVerticalStretch(<Фактор для вертикали>)` можно указать фактор растяжения. Чем больше указанное значение относительно значения, заданного в других компонентах, тем больше места будет выделяться под компонент. Этот параметр можно сравнить с жесткостью пружины.

Можно указать, что минимальная высота компонента зависит от его ширины. Для этого необходимо передать значение `True` в метод `setHeightForWidth(<Флаг>)`. Кроме того, следует в классе компонента переопределить метод `heightForWidth(<Ширина>)` — переопределенный метод должен возвращать высоту компонента для указанной в параметре ширины.

20.7. Объединение компонентов в группу

Состояние одних компонентов может зависеть от состояния других — например, из нескольких переключателей можно выбрать только один. Кроме того, некоторый набор компонентов может использоваться для ввода связанных данных — например, имени, отчества и фамилии пользователя. В этом случае компоненты объединяют в группу.

Группа компонентов отображается внутри рамки, на верхней границе которой выводится текст заголовка. Реализовать группу позволяет класс `QGroupBox`. Иерархия наследования выглядит так:

`(QObject, QPaintDevice) – QWidget – QGroupBox`

Создать экземпляр класса `QGroupBox` можно следующим образом:

`<Объект> = QGroupBox(<Родитель>)`

`<Объект> = QGroupBox(<Текст>[, <Родитель>])`

В необязательном параметре <Родитель> можно указать ссылку на родительский компонент. Параметр <Текст> задает текст заголовка, который отобразится на верхней границе рамки. Внутри текста заголовка символ &, указанный перед буквой, задает комбинацию клавиш быстрого доступа. В этом случае буква, перед которой указан символ &, будет в качестве подсказки пользователю подчеркнута. При одновременном нажатии клавиши <Alt> и подчеркнутой буквы первый компонент из группы окажется в фокусе ввода.

После создания экземпляра класса `QGroupBox` следует добавить компоненты в какой-либо контейнер, а затем передать ссылку на контейнер в метод `setLayout()` группы.

Типичный пример использования класса `QGroupBox` представлен кодом из листинга 20.5. Созданная им группа показана на рис. 20.4.

Листинг 20.5. Пример использования компонента `QGroupBox`

```
# -*- coding: utf-8 -*-
from PyQt5 import QtWidgets
import sys
app = QtWidgets.QApplication(sys.argv)
window = QtWidgets.QWidget()
window.setWindowTitle("QGroupBox")
window.resize(200, 80)
mainbox = QtWidgets.QVBoxLayout()
radio1 = QtWidgets.QRadioButton("&Да")
radio2 = QtWidgets.QRadioButton("&Нет")
box = QtWidgets.QGroupBox("&Вы знаете язык Python?") # Объект группы
hbox = QtWidgets.QHBoxLayout() # Контейнер для группы
hbox.addWidget(radio1) # Добавляем компоненты
hbox.addWidget(radio2)
box.setLayout(hbox) # Передаем ссылку на контейнер
mainbox.addWidget(box) # Добавляем группу в главный контейнер
window.setLayout(mainbox) # Передаем ссылку на главный контейнер в окно
radio1.setChecked(True) # Выбираем первый переключатель
window.show()
sys.exit(app.exec_())
```

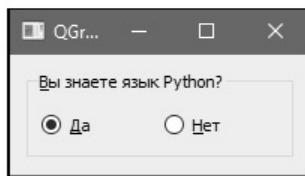


Рис. 20.4. Пример использования компонента `QGroupBox`

Класс `QGroupBox` поддерживает следующие методы (здесь приведены только основные полный их список смотрите на странице <https://doc.qt.io/qt-5/qgroupbox.html>):

- ◆ `setTitle(<Текст>)` задает текст заголовка;
- ◆ `title()` возвращает текст заголовка;

- ◆ `setAlignment(<Выравнивание>)` — задает горизонтальное выравнивание текста заголовка. В параметре указываются следующие атрибуты класса `QtCore.Qt`: `AlignLeft`, `AlignHCenter` или `AlignRight`:
- ```
box.setAlignment(QtCore.Qt.AlignRight)
```
- ◆ `alignment()` — возвращает горизонтальное выравнивание текста заголовка;
- ◆ `setCheckable(<Флаг>)` — если в параметре указать значение `True`, то перед текстом заголовка будет отображен флажок. Если флажок установлен, группа станет активной, а если флажок снят, все компоненты внутри группы окажутся неактивными. По умолчанию флажок не отображается;
- ◆ `isCheckable()` — возвращает значение `True`, если перед заголовком выводится флажок, и `False` — в противном случае;
- ◆ `setChecked(<Флаг>)` — если в параметре указать значение `True`, флажок, отображаемый перед текстом подсказки, будет установлен. Значение `False` сбрасывает флажок. Метод является слотом;
- ◆ `isChecked()` — возвращает значение `True`, если флажок, отображаемый перед текстом заголовка, установлен, и `False` — в противном случае;
- ◆ `setFlat(<Флаг>)` — если в параметре указано значение `True`, будет отображаться только верхняя граница рамки, а если `False` — то все границы рамки;
- ◆ `isFlat()` — возвращает значение `True`, если отображается только верхняя граница рамки, и `False` — если все границы рамки.

Класс `QGroupBox` поддерживает сигналы:

- ◆ `clicked(<Состояние флажка>)` — генерируется при щелчке мышью на флажке, выводимом перед текстом заголовка. Если состояние флажка изменяется с помощью метода `setChecked()`, сигнал не генерируется. Через параметр внутри обработчика доступно значение `True`, если флажок установлен, и `False` — если сброшен;
- ◆ `toggled(<Состояние флажка>)` — генерируется при изменении состояния флажка, выводимого перед текстом заголовка. Через параметр внутри обработчика доступно значение `True`, если флажок установлен, и `False` — если сброшен.

## 20.8. Панель с рамкой

Класс `QFrame` расширяет возможности класса `QWidget` за счет добавления рамки вокруг компонента. Этот класс, в свою очередь, наследуют некоторые компоненты, например надписи, многострочные текстовые поля и др. Иерархия наследования выглядит так:

```
(QObject, QPaintDevice) - QWidget - QFrame
```

Конструктор класса `QFrame` имеет следующий формат:

```
<Объект> = QFrame([parent=<Родитель>[, flags=<Тип окна>]])
```

В параметре `parent` указывается ссылка на родительский компонент. Если параметр не указан или имеет значение `None`, компонент будет обладать своим собственным окном. Если в параметре `flags` указан тип окна, компонент, имея родителя, будет обладать своим собственным окном, но окажется привязан к родителю. Это позволяет, например, создать модальное окно, которое будет блокировать только окно родителя, но не все окна приложения. Какие именно значения можно указать в параметре `flags`, мы уже рассматривали в разд. 18.2.

Класс `QFrame` поддерживает следующие методы (здесь приведены только основные — полный их список смотрите на странице <https://doc.qt.io/qt-5/qframe.html>):

- ◆ `setFrameShape(<Форма>)` — задает форму рамки. Могут быть указаны следующие атрибуты класса `QFrame`:
  - `NoFrame` — 0 — нет рамки;
  - `Box` — 1 — прямоугольная рамка;
  - `Panel` — 2 — панель, которая может быть выпуклой или вогнутой;
  - `WinPanel` — 3 — панель в стиле Windows 2000, которая может быть выпуклой или вогнутой. Ширина границы — 2 пикселя. Этот атрибут присутствует для совместимости со старыми версиями Qt;
  - `HLine` — 4 — горизонтальная линия. Используется как разделитель;
  - `VLine` — 5 — вертикальная линия без содержимого;
  - `StyledPanel` — 6 — панель, внешний вид которой зависит от текущего стиля. Она может быть выпуклой или вогнутой. Это рекомендуемая форма рамки для панелей;
- ◆ `setFrameShadow(<Тень>)` — задает стиль тени. Могут быть указаны следующие атрибуты класса `QFrame`:
  - `Plain` — 16 — нет тени;
  - `Raised` — 32 — панель отображается выпуклой;
  - `Sunken` — 48 — панель отображается вогнутой;
- ◆ `setFrameStyle(<Стиль>)` — задает форму рамки и стиль тени одновременно. В качестве значения через оператор `|` указывается комбинация приведенных ранее атрибутов класса `QFrame`:

```
frame.setFrameStyle(QtWidgets.QFrame.Panel | QtWidgets.QFrame.Raised)
```
- ◆ `setLineWidth(<Ширина>)` — задает ширину линий у рамки;
- ◆ `setMidLineWidth(<Ширина>)` — задает ширину средней линии у рамки. Средняя линия используется для создания эффекта выпуклости и вогнутости и доступна только для форм рамки `Box`, `HLine` и `VLine`.

## 20.9. Панель с вкладками

Для создания панели с вкладками («блокнота») предназначен класс `QTabWidget`. Панель состоит из области заголовка с ярлыками и набора вкладок с различными компонентами. В один момент времени показывается содержимое только одной вкладки. Щелчок мышью на ярлыке в области заголовка приводит к отображению содержимого соответствующей вкладки.

Иерархия наследования для класса `QTabWidget` выглядит так:

`(QObject, QPaintDevice) – QWidget – QTabWidget`

Конструктор класса `QTabWidget` имеет следующий формат:

`<Объект> = QTabWidget([<Родитель>])`

В параметре `<Родитель>` указывается ссылка на родительский компонент. Если он не указан, компонент будет обладать своим собственным окном.

Пример кода, создающего компонент `QTabWidget`, приведен в листинге 20.6. Сама панель с вкладками показана на рис. 20.5.

#### Листинг 20.6. Пример использования компонента `QTabWidget`

```
-*- coding: utf-8 -*-
from PyQt5 import QtWidgets
import sys
app = QtWidgets.QApplication(sys.argv)
window = QtWidgets.QWidget()
window.setWindowTitle("QTabWidget")
window.resize(400, 100)
tab = QtWidgets.QTabWidget()
tab.addTab(QtWidgets.QLabel("Содержимое вкладки 1"), "Вкладка &1")
tab.addTab(QtWidgets.QLabel("Содержимое вкладки 2"), "Вкладка &2")
tab.addTab(QtWidgets.QLabel("Содержимое вкладки 3"), "Вкладка &3")
tab.setCurrentIndex(0)
vbox = QtWidgets.QVBoxLayout()
vbox.addWidget(tab)
window.setLayout(vbox)
window.show()
window.show()
sys.exit(app.exec_())
```

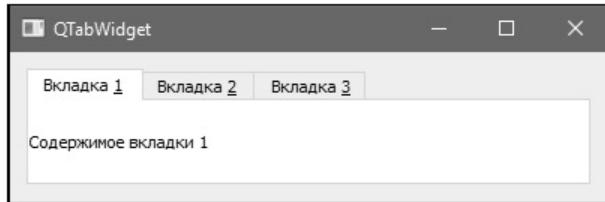


Рис. 20.5. Панель с вкладками `QTabWidget`

Класс `QTabWidget` поддерживает следующие методы (здесь приведены только основные, полное описание класса содержится на странице <https://doc.qt.io/qt-5/qtabwidget.html>):

- ◆ `addTab()` добавляет вкладку в конец контейнера и возвращает ее индекс. Форматы метода:

```
addTab(<Компонент>, <Текст заголовка>)
addTab(<Компонент>, <QIcon>, <Текст заголовка>)
```

В параметре `<Компонент>` указывается ссылка на компонент, который будет отображаться на вкладке. Чаще всего этот компонент является лишь родителем для других компонентов. Параметр `<Текст заголовка>` задает текст, который будет отображаться на ярлыке в области заголовка. Внутри текста заголовка символ `&`, указанный перед буквой, задает комбинацию клавиш быстрого доступа. В этом случае буква, перед которой указан символ `&`, будет в качестве подсказки пользователю подчеркнута. При одновременном нажатии клавиши `<Alt>` и подчеркнутой буквы откроется соответствующая

вкладка. Параметр `<QIcon>` позволяет указать значок (экземпляр класса `QIcon`), который отобразится перед текстом в области заголовка.

- Пример указания стандартного значка:

```
style = window.style()
icon = style.standardIcon(QtWidgets.QStyle.SP_DriveNetIcon)
tab.addTab(QtWidgets.QLabel("Содержимое вкладки 1"), icon,
 "Вкладка &l")
```

- Пример загрузки значка из файла:

```
icon = QtGui.QIcon("icon.png")
tab.addTab(QtWidgets.QLabel("Содержимое вкладки 1"), icon,
 "Вкладка &l")
```

◆ `insertTab()` — добавляет вкладку в указанную позицию и возвращает индекс этой вкладки. Форматы метода:

```
insertTab(<Индекс>, <Компонент>, <Текст заголовка>)
insertTab(<Индекс>, <Компонент>, <QIcon>, <Текст заголовка>)
```

◆ `removeTab(<Индекс>)` — удаляет вкладку с указанным индексом, при этом компонент, который отображался на вкладке, не удаляется;

◆ `clear()` — удаляет все вкладки, при этом компоненты, которые отображались на вкладках, не удаляются;

◆ `setTabText(<Индекс>, <Текст заголовка>)` — задает текст заголовка для вкладки с указанным индексом;

◆ `setElideMode(<Режим>)` — задает режим обрезки текста в названии вкладки, если он не помещается в отведенную область (в месте пропуска выводится троеточие). Могут быть указаны следующие атрибуты класса `QtCore.Qt`:

- `ElideLeft` — 0 — текст обрезается слева;
- `ElideRight` — 1 — текст обрезается справа;
- `ElideMiddle` — 2 — текст вырезается посередине;
- `ElideNone` — 3 — текст не обрезается;

◆ `tabText(<Индекс>)` — возвращает текст заголовка вкладки с указанным индексом;

◆ `setTabIcon(<Индекс>, <QIcon>)` — устанавливает значок перед текстом в заголовке вкладки с указанным индексом. Во втором параметре указывается экземпляр класса `QIcon`;

◆ `setTabPosition(<Позиция>)` — задает позицию области заголовка. Могут быть указаны следующие атрибуты класса `QTabWidget`:

- `North` — 0 — сверху;
- `South` — 1 — снизу;
- `West` — 2 — слева;
- `East` — 3 — справа.

Пример:

```
tab.setTabPosition(QtWidgets.QTabWidget.South)
```

- ◆ `setTabShape(<Форма>)` — задает форму углов ярлыка вкладки в области заголовка. Могут быть указаны следующие атрибуты класса `QTabWidget`:
  - `Rounded` — 0 — скругленные углы (значение по умолчанию);
  - `Triangular` — 1 — треугольная форма;
- ◆ `setTabsClosable(<Флаг>)` — если в качестве параметра указано значение `True`, то после текста заголовка вкладки будет отображена кнопка ее закрытия. По нажатию этой кнопки генерируется сигнал `tabCloseRequested`;
- ◆ `setMovable(<Флаг>)` — если в качестве параметра указано значение `True`, ярлыки вкладок можно перемещать с помощью мыши;
- ◆ `setDocumentMode(<Флаг>)` — если в качестве параметра указано значение `True`, область компонента не будет отображаться как панель;
- ◆ `setUsesScrollButtons(<Флаг>)` — если в качестве параметра указано значение `True`, то, если все ярлыки вкладок не помещаются в область заголовка панели, появятся две кнопки, с помощью которых можно прокручивать область заголовка, тем самым отображая только часть ярлыков. Значение `False` запрещает скрытие ярлыков;
- ◆ `setTabToolTip(<Индекс>, <Текст>)` — задает текст всплывающей подсказки для ярлыка вкладки с указанным индексом;
- ◆ `setTabWhatsThis(<Индекс>, <Текст>)` — задает текст справки для ярлыка вкладки с указанным индексом;
- ◆ `setTabEnabled(<Индекс>, <Флаг>)` — если вторым параметром передано значение `False`, вкладка с указанным в первом параметре индексом станет недоступной. Значение `True` делает вкладку доступной;
- ◆ `isTabEnabled(<Индекс>)` — возвращает значение `True`, если вкладка с указанным индексом доступна, и `False` — в противном случае;
- ◆ `setCurrentIndex(<Индекс>)` — делает видимой вкладку с указанным в параметре индексом. Метод является слотом;
- ◆ `currentIndex()` — возвращает индекс видимой вкладки;
- ◆ `setCurrentWidget(<Компонент>)` — делает видимой вкладку с указанным компонентом. Метод является слотом;
- ◆ `currentWidget()` — возвращает ссылку на компонент, расположенный на видимой вкладке;
- ◆ `widget(<Индекс>)` — возвращает ссылку на компонент, который расположен по указанному индексу, или значение `None`;
- ◆ `indexOf(<Компонент>)` — возвращает индекс вкладки, на которой расположен компонент. Если компонент не найден, возвращается значение `-1`;
- ◆ `count()` — возвращает количество вкладок. Получить количество вкладок можно также с помощью функции `len()`:

```
print(tab.count(), len(tab))
```

Класс `QTabWidget` поддерживает такие сигналы:

- ◆ `currentChanged(<Индекс>)` — генерируется при переключении на другую вкладку. Через параметр внутри обработчика доступен целочисленный индекс новой вкладки;

- ◆ `tabCloseRequested(<Индекс>)` генерируется при нажатии кнопки закрытия вкладки. Через параметр внутри обработчика доступен целочисленный индекс закрываемой вкладки.

## 20.10. Компонент «аккордеон»

Класс `QToolBox` позволяет создать «аккордеон» — компонент с несколькими вкладками, в котором изначально отображается содержимое только одной вкладки, а у остальных видимы лишь заголовки. После щелчка мышью на заголовке вкладки она открывается, а остальные сворачиваются. Иерархия наследования выглядит так:

`(QObject, QPaintDevice) — QWidget — QFrame — QToolBox`

Конструктор класса `QToolBox` имеет следующий формат:

`<Объект> = QToolBox([parent=<Родитель>[, flags=<Тип окна>]])`

В параметре `parent` указывается ссылка на родительский компонент. Если он не указан или имеет значение `None`, компонент будет обладать своим собственным окном. В параметре `flags` может быть указан тип окна.

Пример кода, создающего компонент класса `QToolBox`, представлен в листинге 20.7. Созданный им «аккордеон» показан на рис. 20.6.

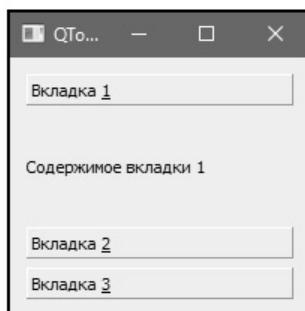


Рис. 20.6. Пример использования класса `QToolBox`

### Листинг 20.7. Пример использования класса `QToolBox`

```
-*- coding: utf-8 -*-
from PyQt5 import QtWidgets
import sys
app = QtWidgets.QApplication(sys.argv)
window = QtWidgets.QWidget()
window.setWindowTitle("QToolBox")
window.resize(200, 100)
toolBox = QtWidgets.QToolBox()
toolBox.addItem(QtWidgets.QLabel("Содержимое вкладки 1"), "Вкладка &1")
toolBox.addItem(QtWidgets.QLabel("Содержимое вкладки 2"), "Вкладка &2")
toolBox.addItem(QtWidgets.QLabel("Содержимое вкладки 3"), "Вкладка &3")
toolBox.setCurrentIndex(0)
```

```
vbox = QtWidgets.QVBoxLayout()
vbox.addWidget(toolBox)
window.setLayout(vbox)
window.show()
sys.exit(app.exec_())
```

Класс QToolBox поддерживает следующие методы (здесь приведены только основные — полный их список смотрите на странице <https://doc.qt.io/qt-5/qtoolbox.html>):

- ◆ `addItem()` — добавляет вкладку в конец контейнера. Метод возвращает индекс добавленной вкладки. Форматы метода:

```
addItem(<Компонент>, <Текст заголовка>
addItem(<Компонент>, <QIcon>, <Текст заголовка>)
```

В параметре `<Компонент>` указывается ссылка на компонент, который будет отображаться на вкладке. Чаще всего этот компонент является лишь родителем для других компонентов. Параметр `<Текст заголовка>` задает текст, который будет отображаться на ярлыке в области заголовка. Внутри текста заголовка символ &, указанный перед буквой, задает комбинацию клавиш быстрого доступа. В этом случае буква, перед которой указан символ &, будет — в качестве подсказки пользователю — подчеркнута. При одновременном нажатии клавиши `<Alt>` и подчеркнутой буквы откроется соответствующая вкладка. Параметр `<QIcon>` позволяет указать значок (экземпляр класса `QIcon`), который отобразится перед текстом в области заголовка;

- ◆ `insertItem()` — добавляет вкладку в указанную позицию. Метод возвращает индекс добавленной вкладки. Форматы метода:

```
insertItem(<Индекс>, <Компонент>, <Текст заголовка>
insertItem(<Индекс>, <Компонент>, <QIcon>, <Текст заголовка>)
```

- ◆ `removeItem(<Индекс>)` — удаляет вкладку с указанным индексом, при этом компонент, который отображался на вкладке, не удаляется;

- ◆ `setItemText(<Индекс>, <Текст заголовка>)` — задает текст заголовка для вкладки с указанным индексом;

- ◆ `itemText(<Индекс>)` — возвращает текст заголовка вкладки с указанным индексом;

- ◆ `setItemIcon(<Индекс>, <QIcon>)` — устанавливает значок перед текстом в заголовке вкладки с указанным индексом. Во втором параметре указывается экземпляр класса `QIcon`;

- ◆ `setItemToolTip(<Индекс>, <Текст>)` — задает текст всплывающей подсказки для ярлыка вкладки с указанным индексом;

- ◆ `setItemEnabled(<Индекс>, <Флаг>)` — если вторым параметром передается значение `False`, вкладка с указанным в первом параметре индексом станет недоступной. Значение `True` делает вкладку доступной;

- ◆ `isItemEnabled(<Индекс>)` — возвращает значение `True`, если вкладка с указанным индексом доступна, и `False` — в противном случае;

- ◆ `setCurrentIndex(<Индекс>)` — делает видимой вкладку с указанным индексом. Метод является слотом;

- ◆ `currentIndex()` — возвращает индекс видимой вкладки;

- ◆ `setCurrentWidget(<Компонент>)` — делает видимым вкладку с указанным компонентом. Метод является слотом;

- ◆ `currentWidget()` — возвращает ссылку на компонент, который расположен на видимой вкладке;
- ◆ `widget(<Индекс>)` — возвращает ссылку на компонент, который расположен по указанному индексу, или значение `None`;
- ◆ `indexOf(<Компонент>)` — возвращает индекс вкладки, на которой расположен компонент. Если компонент не найден, возвращается значение `-1`;
- ◆ `count()` — возвращает количество вкладок. Получить количество вкладок можно также с помощью функции `len()`:

```
print(toolBox.count(), len(toolBox))
```

При переключении на другую вкладку генерируется сигнал `currentChanged(<Индекс>)`. Через параметр внутри обработчика доступен целочисленный индекс вкладки, на которую было выполнено переключение.

## 20.11. Панели с изменяемым размером

Класс `QSplitter` позволяет изменять размеры добавленных компонентов, взявшись мышью за границу между компонентами. Иерархия наследования выглядит так:

```
(QObject, QPaintDevice) - QWidget - QFrame - QSplitter
```

Конструктор класса `QSplitter` имеет два формата:

```
<Объект> = QSplitter([parent=<Родитель>])
<Объект> = QSplitter(<Ориентация>, parent=<Родитель>)
```

В параметре `parent` указывается ссылка на родительский компонент. Если таковой не указан или имеет значение `None`, компонент будет обладать своим собственным окном. Параметр `<Ориентация>` задает ориентацию размещения компонентов. Могут быть заданы атрибуты `Horizontal` (по горизонтали) или `Vertical` (по вертикали) класса `QtCore.Qt`. Если параметр не указан, компоненты размещаются по горизонтали.

Пример использования класса `QSplitter` показан в листинге 20.8, а результат можно увидеть на рис. 20.7.

### Листинг 20.8. Пример использования класса `QSplitter`

```
-*- coding: utf-8 -*-
from PyQt5 import QtCore, QtWidgets
import sys
app = QtWidgets.QApplication(sys.argv)
window = QtWidgets.QWidget()
window.setWindowTitle("QSplitter")
window.resize(200, 200)
splitter = QtWidgets.QSplitter(QtCore.Qt.Vertical)
label1 = QtWidgets.QLabel("Содержимое компонента 1")
label2 = QtWidgets.QLabel("Содержимое компонента 2")
label1.setStyleSheet(QtWidgets.QFrame.Box | QtWidgets.QFrame.Plain)
label2.setStyleSheet(QtWidgets.QFrame.Box | QtWidgets.QFrame.Plain)
splitter.addWidget(label1)
splitter.addWidget(label2)
```

```
vbox = QtWidgets.QVBoxLayout()
vbox.addWidget(splitter)
window.setLayout(vbox)
window.show()
sys.exit(app.exec_())
```

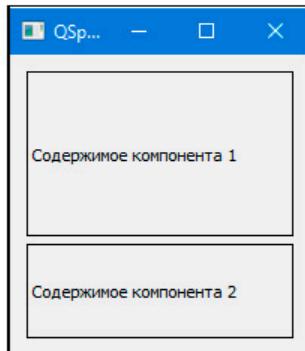


Рис. 20.7. Пример использования класса QSplitter

Класс `QSplitter` поддерживает следующие методы (здесь приведены только основные полный их список смотрите на странице <https://doc.qt.io/qt-5/qsplitter.html>):

- ◆ `addWidget(<Компонент>)` добавляет компонент в конец контейнера;
- ◆ `insertWidget(<Индекс>, <Компонент>)` добавляет компонент в указанную позицию. Если компонент был добавлен ранее, он будет перемещен в новую позицию;
- ◆ `setOrientation(<Ориентация>)` задает ориентацию размещения компонентов. Могут быть заданы атрибуты `Horizontal` (по горизонтали) или `Vertical` (по вертикали) класса `QtCore.Qt`;
- ◆ `setHandleWidth(<Ширина>)` задает ширину компонента-разделителя, взявшись за который мышью, можно изменить размер области;
- ◆ `saveState()` возвращает экземпляр класса `QByteArray` с размерами всех областей. Эти данные можно сохранить (например, в файл), а затем восстановить с помощью метода `restoreState()`;
- ◆ `restoreState(<QByteArray>)` восстанавливает размеры областей компонента из экземпляра класса, возвращенного методом `saveState()`;
- ◆ `setChildrenCollapsible(<Флаг>)` если в параметре указано значение `False`, пользователь не сможет уменьшить размеры всех компонентов до нуля. По умолчанию размер может быть нулевым, даже если для какого-либо компонента установлены минимальные размеры;
- ◆ `setCollapsible(<Индекс>, <Флаг>)` значение `False` в параметре `<Флаг>` запрещает уменьшение размеров до нуля для компонента с указанным индексом;
- ◆ `setOpaqueResize(<Флаг>)` если в качестве параметра указано значение `False`, размеры компонентов изменятся только после окончания перемещения границы и отпускания кнопки мыши. В процессе перемещения мыши вместе с ней будет перемещаться специальный компонент в виде линии;
- ◆ `setStretchFactor(<Индекс>, <Фактор>)` задает фактор растяжения для компонента с указанным индексом;

- ◆ `setSizes(<Список>)` — задает размеры всех компонентов. Для горизонтального контейнера указывается список со значениями ширины каждого компонента, а для вертикального контейнера — список со значениями высоты каждого компонента;
- ◆ `sizes()` — возвращает список с размерами (шириной или высотой):  

```
print(splitter.sizes()) # Результат: [308, 15]
```
- ◆ `count()` — возвращает количество компонентов. Получить количество компонентов можно также с помощью функции `len()`:  

```
print(splitter.count(), len(splitter))
```
- ◆ `widget(<Индекс>)` — возвращает ссылку на компонент, который расположен по указанному индексу, или значение `None`;
- ◆ `indexOf(<Компонент>)` — возвращает индекс области, в которой расположен компонент. Если таковой не найден, возвращается значение `-1`.

При изменении размеров генерируется сигнал `splitterMoved(<Позиция>, <Индекс>)`. Через первый параметр внутри обработчика доступна новая позиция, а через второй параметр — индекс перемещаемого разделителя; оба параметра целочисленные.

## 20.12. Область с полосами прокрутки

Класс `QScrollArea` реализует область с полосами прокрутки. Если компонент не помещается в размеры области, полосы прокрутки будут отображены автоматически. Иерархия наследования выглядит так:

```
(QObject, QPaintDevice) - QWidget - QFrame -
 QAbstractScrollArea - QScrollArea
```

Конструктор класса `QScrollArea` имеет следующий формат:

```
<Объект> = QScrollArea([<Родитель>])
```

Класс `QScrollArea` поддерживает следующие методы (здесь приведены только основные — полный их список смотрите на странице <https://doc.qt.io/qt-5/qscrollarea.html>):

- ◆ `addWidget(<Компонент>)` — помещает компонент в область прокрутки;
- ◆ `setWidgetResizable(<Флаг>)` — если в качестве параметра указано значение `True`, при изменении размеров области будут изменяться и размеры компонента. Значение `False` запрещает изменение размеров компонента;
- ◆ `setAlignment(<Выравнивание>)` — задает местоположение компонента внутри области, когда размеры области больше размеров компонента:  

```
scrollArea.setAlignment(Qt.AlignCenter)
```
- ◆ `ensureVisible(<X>, <Y>[, xMargin=50] [, yMargin=50])` — прокручивает область к точке с координатами (`<X>`, `<Y>`) и полями `xMargin` и `yMargin`;
- ◆ `ensureWidgetVisible(<Компонент>[, xMargin=50] [, yMargin=50])` — прокручивает область таким образом, чтобы `<Компонент>` был видим;
- ◆ `widget()` — возвращает ссылку на компонент, который расположен внутри области, или значение `None`;
- ◆ `takeWidget()` — удаляет компонент из области и возвращает ссылку на него. Сам компонент не удаляется.

Класс `QScrollArea` наследует следующие методы из класса `QAbstractScrollArea` (здесь перечислены только основные — полный их список смотрите на странице <https://doc.qt.io/qt-5/qabstractscrollarea.html>):

- ◆ `horizontalScrollBar()` — возвращает ссылку на горизонтальную полосу прокрутки (экземпляр класса `QScrollBar`);
- ◆ `verticalScrollBar()` — возвращает ссылку на вертикальную полосу прокрутки (экземпляр класса `QScrollBar`);
- ◆ `setHorizontalScrollBarPolicy(<Режим>)` — устанавливает режим отображения горизонтальной полосы прокрутки;
- ◆ `setVerticalScrollBarPolicy(<Режим>)` — устанавливает режим отображения вертикальной полосы прокрутки.

В параметре `<Режим>` могут быть указаны следующие атрибуты из класса `QtCore.Qt`:

- `ScrollBarAsNeeded` — 0 — полоса прокрутки отображается только в том случае, если размеры компонента больше размеров области;
- `ScrollBarAlwaysOff` — 1 — полоса прокрутки никогда не отображается;
- `ScrollBarAlwaysOn` — 2 — полоса прокрутки отображается всегда.