

Pemrograman Desktop

Pertemuan 4 : Layout
yoga@trunojoyo.ac.id

Pendahuluan

- Programmer biasanya ingin menambahkan lebih dari satu widget ke jendela, dan memiliki kendali atas di mana widget yang ditambahkan akan ditempatkan.
- Untuk menyusun widget di Qt bisa menggunakan tata letak.
- Ada 4 tata letak dasar yang tersedia di Qt, yang tercantum dalam tabel berikut.

Layout	Behavior
QHBoxLayout	Linear horizontal layout
QVBoxLayout	Linear vertical layout
QGridLayout	In indexable grid XxY
QStackedLayout	Stacked (z) in front of one another

Pendahuluan

- Ada tiga tata letak 2 dimensi yang tersedia di Qt. QVBoxLayout, QHBoxLayout dan QGridLayout.
- Selain itu ada juga QStackedLayout yang memungkinkan menempatkan widget satu di atas yang lain dalam ruang yang sama, namun hanya menampilkan satu widget dalam satu waktu.

Contoh Aplikasi

```
import sys

from PyQt6.QtCore import Qt
from PyQt6.QtWidgets import QApplication,
    QMainWindow

from layout_colorwidget import Color

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("My App")
```

```
widget = Color("red")
self.setCentralWidget(widget)
```

```
app = QApplication(sys.argv)

window = MainWindow()
window.show()

app.exec()
```

Contoh Aplikasi

Simpan Dengan Nama layout_colorwidget.py

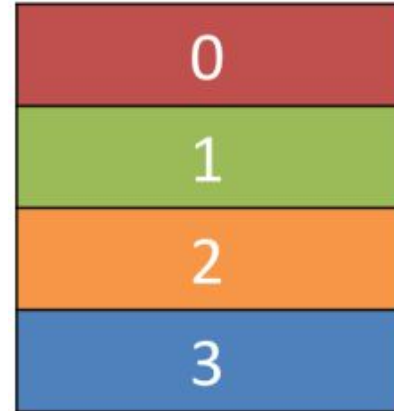
```
from PyQt6.QtGui import QColor, QPalette
from PyQt6.QtWidgets import QWidget
```

```
class Color(QWidget):
    def __init__(self, color):
        super().__init__()
        self.setAutoFillBackground(True)

        palette = self.palette()
        palette.setColor(
            QPalette.ColorRole.Window,
            QColor(color))
        self.setPalette(palette)
```

QVBoxLayout

- QVBoxLayout mengatur widget satu di atas yang lain secara linier.
- Menambahkan widget akan menambahkannya ke bagian bawah kolom.
- Untuk menambahkan tata letak ke QMainWindow kita bisa menerapkannya ke QWidget pada contoh sebelumnya.
- Memungkinkan untuk menggunakan `.setCentralWidget` untuk menerapkan widget (dan tata letak) ke jendela.



Contoh Aplikasi

```
import sys

from PyQt6.QtCore import Qt
from PyQt6.QtWidgets import (
    QApplication,
    QMainWindow,
    QVBoxLayout,
    QWidget,
)

from layout_colorwidget import Color

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("My App")
```

```
layout = QVBoxLayout()

layout.addWidget(Color("red"))

widget = QWidget()
widget.setLayout(layout)
self.setCentralWidget(widget)

app = QApplication(sys.argv)

window = MainWindow()
window.show()

app.exec()
```

QVBoxLayout

- Ketika menambahkan widget, widget tersebut akan berbaris secara vertikal sesuai urutan penambahannya.
- Setiap widget baru yang Anda tambahkan dengan `.addWidget()`, ditambahkan secara vertikal.
- Window akan menampilkan daftar vertikal dari widget.
- Setiap widget baru ditambahkan ke bagian bawah daftar.

Contoh Aplikasi

```
import sys

from PyQt6.QtCore import Qt
from PyQt6.QtWidgets import (
    QApplication,
    QMainWindow,
    QVBoxLayout,
    QWidget,
)

from layout_colorwidget import Color

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("My App")
```

```
layout = QVBoxLayout()

layout.addWidget(Color("red"))
layout.addWidget(Color("green"))
layout.addWidget(Color("blue"))

widget = QWidget()
widget.setLayout(layout)
self.setCentralWidget(widget)

app = QApplication(sys.argv)

window = MainWindow()
window.show()

app.exec()
```

QHBoxLayout

- QHBoxLayout mengatur widget ke arah horisontal.
- Menambahkan widget akan menambahkannya ke bagian samping.



Contoh Aplikasi

```
import sys

from PyQt6.QtCore import Qt
from PyQt6.QtWidgets import (
    QApplication,
    QHBoxLayout,
    QLabel,
    QMainWindow,
    QWidget,
)

from layout_colorwidget import Color
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("My App")
        layout = QHBoxLayout()
        layout.addWidget(Color("red"))
        layout.addWidget(Color("green"))
```

```
layout.addWidget(Color("blue"))
```

```
widget = QWidget()
widget.setLayout(layout)
self.setCentralWidget(widget)
```

```
app = QApplication(sys.argv)
```

```
window = MainWindow()
window.show()
```

```
app.exec()
```

QHBoxLayout

- Untuk tata letak yang lebih kompleks, kita dapat menyusun tata letak satu sama lain menggunakan `.addLayout` .
- Kita bisa menambahkan `QVBoxLayout` ke dalam `QHBoxLayout` utama. Jika kita menambahkan beberapa widget ke `QVBoxLayout`, widget tersebut akan disusun secara vertikal di slot pertama tata letak induk.

Contoh Aplikasi

```
import sys

from PyQt6.QtCore import Qt
from PyQt6.QtWidgets import (
    QApplication,
    QHBoxLayout,
    QLabel,
    QMainWindow,
    QVBoxLayout,
    QWidget,
)

from layout_colorwidget import Color

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("My App")
```

```
layout1 = QHBoxLayout()
layout2 = QVBoxLayout()
layout3 = QVBoxLayout()

layout2.addWidget(Color("red"))
layout2.addWidget(Color("yellow"))
layout2.addWidget(Color("purple"))

layout1.addLayout(layout2)

layout1.addWidget(Color("green"))
layout3.addWidget(Color("red"))
layout3.addWidget(Color("purple"))

layout1.addLayout(layout3)

widget = QWidget()
widget.setLayout(layout1)
self.setCentralWidget(widget)

app = QApplication(sys.argv)
window = MainWindow()
window.show()
app.exec()
```

QHBoxLayout

- Kita dapat mengatur jarak di sekitar tata letak menggunakan `.setContentMargins` atau mengatur jarak antar elemen menggunakan `.setSpacing`.

```
layout1.setContentMargins(0, 0, 0, 0)  
layout1.setSpacing(20)
```

- Tambahkan pada kode program sebelum nya

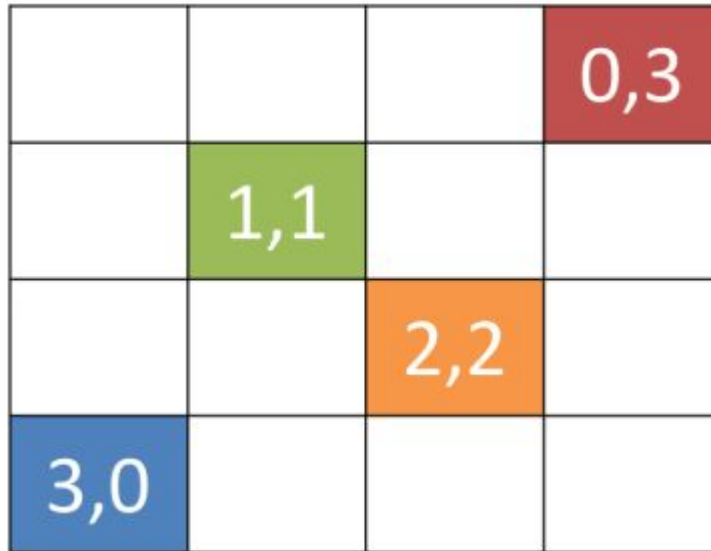
Widget QGridLayout

- Meskipun bermanfaat, jika kita mencoba dan menggunakan QVBoxLayout dan QHBoxLayout untuk meletakkan banyak elemen, mis. untuk formulir,
- Kita akan merasa sangat sulit untuk memastikan widget dengan ukuran berbeda berbaris. Solusi untuk ini adalah QGridLayout.
- QGridLayout memungkinkan kita memposisikan item secara spesifik dalam grid.

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

Widget QGridLayout

- Kita menentukan posisi baris dan kolom untuk setiap widget.
- Anda dapat melewati elemen, dan elemen tersebut akan dibiarkan kosong.



A 4x4 grid illustrating the QGridLayout widget. The grid contains four colored cells, each representing a widget at a specific row and column index. The coordinates are displayed in the center of each cell.

			0,3
	1,1		
		2,2	
3,0			

Contoh Aplikasi

```
import sys

from PyQt6.QtCore import Qt
from PyQt6.QtWidgets import (
    QApplication,
    QGridLayout,
    QLabel,
    QMainWindow,
    QWidget,
)

from layout_colorwidget import Color

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("My App")
```

```
layout = QGridLayout()

layout.addWidget(Color("red"), 0, 0)
layout.addWidget(Color("green"), 1, 0)
layout.addWidget(Color("blue"), 1, 1)
layout.addWidget(Color("purple"), 2, 1)

widget = QWidget()
widget.setLayout(layout)
self.setCentralWidget(widget)

app = QApplication(sys.argv)

window = MainWindow()
window.show()

app.exec()
```

QStackedLayout

- QStackedLayout memungkinkan memposisikan elemen tepat di depan satu sama lain.
- Kita kemudian dapat memilih widget mana yang ingin ditampilkan.
- Kita bisa menggunakan untuk menggambar lapisan dalam aplikasi grafis, atau untuk meniru antarmuka tab.

QStackedLayout

- QStackedWidget yang merupakan widget container yang bekerja dengan cara yang persis sama dengan QStackedLayout.
- Kita bisa menambahkan tumpukan langsung ke QMainWindow dengan `.setCentralWidget`.



Contoh Aplikasi

```
import sys

from PyQt6.QtCore import Qt
from PyQt6.QtWidgets import (
    QApplication,
    QLabel,
    QMainWindow,
    QStackedLayout,
    QWidget,
)

from layout_colorwidget import Color

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("My App")

        layout = QStackedLayout()
```

```
        layout.addWidget(Color("red"))
        layout.addWidget(Color("green"))
        layout.addWidget(Color("blue"))
        layout.addWidget(Color("yellow"))

        layout.setCurrentIndex(3)

        widget = QWidget()
        widget.setLayout(layout)
        self.setCentralWidget(widget)
```

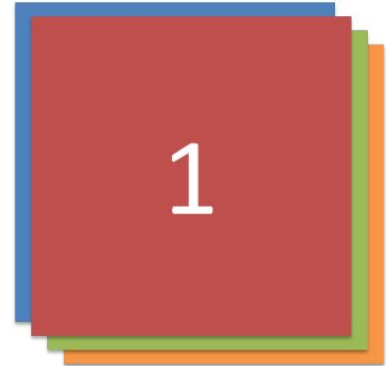
```
app = QApplication(sys.argv)

window = MainWindow()
window.show()

app.exec()
```

QStackedLayout

- Kita bisa mengatur widget mana saja yang diposisikan di depan
- contoh, jika memposisikan stack ke 2 ke depan
- Hanya satu tampilan ('tab') yang terlihat pada satu waktu.
- Kita dapat mengontrol widget mana yang akan ditampilkan kapan saja dengan menggunakan `.setCurrentIndex()` atau `.setCurrentWidget()` untuk menyetel item berdasarkan indeks (dalam urutan penambahan widget) atau berdasarkan widget itu sendiri.



Contoh Aplikasi

```
import sys

from PyQt6.QtCore import Qt
from PyQt6.QtWidgets import (
    QApplication,
    QHBoxLayout,
    QLabel,
    QMainWindow,
    QPushButton,
    QStackedLayout,
    QVBoxLayout,
    QWidget,
)

from layout_colorwidget import Color

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("My App")
```

```
        pagelayout = QVBoxLayout()
        button_layout = QHBoxLayout()
        self.stacklayout = QStackedLayout()

        pagelayout.addLayout(button_layout)
        pagelayout.addLayout(self.stacklayout)

        btn = QPushButton("red")
        btn.pressed.connect(self.activate_tab_1)
        button_layout.addWidget(btn)
        self.stacklayout.addWidget(Color("red"))

        btn = QPushButton("green")
        btn.pressed.connect(self.activate_tab_2)
        button_layout.addWidget(btn)
        self.stacklayout.addWidget(Color("green"))

        btn = QPushButton("yellow")
        btn.pressed.connect(self.activate_tab_3)
        button_layout.addWidget(btn)

        self.stacklayout.addWidget(Color("yellow"))

        widget = QWidget()
        widget.setLayout(pagelayout)
        self.setCentralWidget(widget)
```

Contoh Aplikasi

```
def activate_tab_1(self):  
    self.stacklayout.setCurrentIndex(0)  
  
def activate_tab_2(self):  
    self.stacklayout.setCurrentIndex(1)  
  
def activate_tab_3(self):  
    self.stacklayout.setCurrentIndex(2)
```

```
app = QApplication(sys.argv)
```

```
window = MainWindow()  
window.show()
```

```
app.exec()
```

QTabWidget

- Cara lebih elegan untuk berkerja dengan tab bisa menggunakan QTabWidget.
- Meskipun bukan jenis widget layout

Contoh Aplikasi

```
import sys

from PyQt6.QtCore import Qt
from PyQt6.QtWidgets import (
    QApplication,
    QLabel,
    QMainWindow,
    QPushButton,
    QTabWidget,
    QWidget,
)

from layout_colorwidget import Color

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("My App")
```

```
        tabs = QTabWidget()

        tabs.setTabPosition(QTabWidget.TabPosition.West)
        tabs.setMovable(True)

        for n, color in enumerate(["red", "green",
                                    "blue", "yellow"]):
            tabs.addTab(Color(color), color)

        self.setCentralWidget(tabs)

app = QApplication(sys.argv)

window = MainWindow()
window.show()

app.exec()
```