# Programowanie, część III

Rafał J. Wysocki

9 kwietnia 2010

# Biblioteki i podsystemy graficzne

## Biblioteki graficzne (*ang. graphics library*)

Są potrzebne, aby można było programować grafikę w C++, ponieważ (niestety) we współczesnych systemach komputerowych podsystemy graficzne (*ang. graphics subsystem*) są *bardzo* skomplikowane.

# Biblioteki i podsystemy graficzne

## Biblioteki graficzne (*ang. graphics library*)

Są potrzebne, aby można było programować grafikę w C++, ponieważ (niestety) we współczesnych systemach komputerowych podsystemy graficzne (*ang. graphics subsystem*) są *bardzo* skomplikowane.

## Przykład – GNU/Linux

- Sprzęt (*ang. hardware*) – wyświetlanie, przetwarzanie scen 3D, komunikacja z użytkownikiem
- Sterowniki w jądrze – operowanie sprzętem (niski poziom)
- Serwer X Windows – komunikacja ze sterownikami, system okien, udostępnianie przez sieć, zdarzenia
- Menedżer okien (*ang. window manager*) – dekoracje okien, komunikacja z aplikacjami i serwerem X Windows, zdarzenia

# Obsługa zdarzeń

**Zdarzenie (*ang. event*)**

Sytuacja potencjalnie wymagająca reakcji ze strony co najmniej jednego z programów korzystających z podsystemu graficznego.

# Obsługa zdarzeń

## Zdarzenie (*ang. event*)

Sytuacja potencjalnie wymagająca reakcji ze strony co najmniej jednego z programów korzystających z podsystemu graficznego.

## Źródła zdarzeń

1. Serwer X Windows
   - Sprzęt (klawiatura, mysz, ekran dotykowy)
   - Interakcje między oknami (zachodzenie na siebie itp.)
2. Menedżer okien
   - Komunikaty z menu itp.
   - Komunikaty od aplikacji

# Obsługa zdarzeń

## Zdarzenie (*ang. event*)

Sytuacja potencjalnie wymagająca reakcji ze strony co najmniej jednego z programów korzystających z podsystemu graficznego.

## Źródła zdarzeń

1. Serwer X Windows
   - Sprzęt (klawiatura, mysz, ekran dotykowy)
   - Interakcje między oknami (zachodzenie na siebie itp.)
2. Menedżer okien
   - Komunikaty z menu itp.
   - Komunikaty od aplikacji

## Sygnalizowanie zdarzenia

Sprzęt $\Leftrightarrow$ Serwer X Windows $\Leftrightarrow$ Menedżer okien $\Leftrightarrow$ Aplikacja

# Związki między zdarzeniami

Jedno zdarzenie może powodować wystąpienie kolejnych zdarzeń.

# Związki między zdarzeniami

Jedno zdarzenie może powodować wystąpienie kolejnych zdarzeń.

## Przykład: ukrywanie okna

1. Zdarzenie – kliniecie myszą w obszarze przycisku ukrywania okna.
2. Serwer X Windows przekazuje informację o tym zdarzeniu do menedżera okien.
3. Menedżer okien inicjuje operację ukrywania okna. W jej wyniku inne okna mogą stać się widoczne.
4. Serwer X Windows wykrywa te sytuacje i traktuje je jako zdarzenia. Przekazuje informacje o nich do menedżera okien.
5. Menedżer okien przekazuje informacje o zdarzeniach do aplikacji korzystających z „odsłoniętych” okien.

# Graficzny interfejs użytkownika

## GUI (*ang. Graphical User Interface*)

Umożliwia użytkownikowi sterowanie komputerem (systemem operacyjnym i aplikacjami) za pośrednictwem podsystemu graficznego.

# Graficzny interfejs użytkownika

## GUI (*ang. Graphical User Interface*)

Umożliwia użytkownikowi sterowanie komputerem (systemem operacyjnym i aplikacjami) za pośrednictwem podsystemu graficznego.

## Sterowanie z wykorzystaniem zdarzeń

Poczynania użytkownika „wytwarzają” zdarzenia (np. ruch myszą, wciśnięcie klawisza), które są interpretowane (na wielu poziomach) i skutkują przeprowadzeniem odpowiednich działań.

# Graficzny interfejs użytkownika

## GUI (*ang. Graphical User Interface*)

Umożliwia użytkownikowi sterowanie komputerem (systemem operacyjnym i aplikacjami) za pośrednictwem podsystemu graficznego.

## Sterowanie z wykorzystaniem zdarzeń

Poczynania użytkownika „wytwarzają” zdarzenia (np. ruch myszą, wciśnięcie klawisza), które są interpretowane (na wielu poziomach) i skutkują przeprowadzeniem odpowiednich działań.

Jeśli każda akcja w systemie komputerowym jest skutkiem jakiegoś zdarzenia (np. związanego z GUI), to mamy do czynienia ze środowiskiem sterowanym zdarzeniami (*ang. event-driven environment*).

# Komunikacja z użytkownikiem w GUI

Machanizm obsługi zdarzeń pozwala użytkownikowi wpływać na zachowanie się komputera (systemu operacyjnego i aplikacji), ale potrzebne jest też przekazywanie informacji zwrotnych (*ang. feedback*) użytkownikowi.

# Komunikacja z użytkownikiem w GUI

Machanizm obsługi zdarzeń pozwala użytkownikowi wpływać na zachowanie się komputera (systemu operacyjnego i aplikacji), ale potrzebne jest też przekazywanie informacji zwrotnych (*ang. feedback*) użytkownikowi.

W GUI do tego celu wykorzystywana jest grafika – grafika wyświetlana na ekranie reprezentuje informacje przeznaczone dla użytkownika.

# Komunikacja z użytkownikiem w GUI

Machanizm obsługi zdarzeń pozwala użytkownikowi wpływać na zachowanie się komputera (systemu operacyjnego i aplikacji), ale potrzebne jest też przekazywanie informacji zwrotnych (*ang. feedback*) użytkownikowi.

W GUI do tego celu wykorzystywana jest grafika – grafika wyświetlana na ekranie reprezentuje informacje przeznaczone dla użytkownika.

Może ona zawierać tekst, ale w GUI tekst jest wyświetlany „w towarzystwie" elementów graficznych (tzn. tekst jest traktowany jako część grafiki).

# Podstawowe elementy grafiki (*ang. graphics primitives*)

Każdy rysunek jest zbudowanych z podstawowych elementów, takich jak pojedyncze punkty, linie proste, łuki, prostokąty itp.

# Podstawowe elementy grafiki (*ang. graphics primitives*)

Każdy rysunek jest zbudowanych z podstawowych elementów, takich jak pojedyncze punkty, linie proste, łuki, prostokąty itp.

Zwykle sprzęt (tzw. karta graficzna) może być zaprogramowany do umieszczania takich podstawowych elementów w odpowiednich miejscach ekranu.

# Podstawowe elementy grafiki (*ang. graphics primitives*)

Każdy rysunek jest zbudowanych z podstawowych elementów, takich jak pojedyncze punkty, linie proste, łuki, prostokąty itp.

Zwykle sprzęt (tzw. karta graficzna) może być zaprogramowany do umieszczania takich podstawowych elementów w odpowiednich miejscach ekranu.

## Pamięć obrazu (*ang. video memory*)

Pamięć dostępna dla procesora karty graficznej (zwykle wchodzi w skład samej karty), zawierająca informacje o tym, co ma być wyświetlane.

# Podstawowe elementy grafiki (*ang. graphics primitives*)

Każdy rysunek jest zbudowanych z podstawowych elementów, takich jak pojedyncze punkty, linie proste, łuki, prostokąty itp.

Zwykle sprzęt (tzw. karta graficzna) może być zaprogramowany do umieszczania takich podstawowych elementów w odpowiednich miejscach ekranu.

**Pamięć obrazu (*ang. video memory*)**

Pamięć dostępna dla procesora karty graficznej (zwykle wchodzi w skład samej karty), zawierająca informacje o tym, co ma być wyświetlane.

Podstawowe elementy grafiki są zapisywane (w odpowiedni sposób i we właściwej kolejności) w pamięci obrazu, dzięki czemu (później) stają się widoczne na ekranie wyświetlacza (np. monitora).

# Bufor klatki i piksele

**Bufor klatki (*ang. frame buffer*)**

Część pamięci obrazu, której zawartość bezpośrednio określa wyświetlany obraz (tzn. jest odwzorowywana na ekran).

# Bufor klatki i piksele

## Bufor klatki (*ang. frame buffer*)

Część pamięci obrazu, której zawartość bezpośrednio określa wyświetlany obraz (tzn. jest odwzorowywana na ekran).

## Piksel (*ang. pixel*)

Odpowiada pojedynczemu punktowi o ustalonym kolorze na wyświetlanym obrazie. Informacje o pikselach są zapisywane w buforze klatki.

# Bufor klatki i piksele

## Bufor klatki (*ang. frame buffer*)

Część pamięci obrazu, której zawartość bezpośrednio określa wyświetlany obraz (tzn. jest odwzorowywana na ekran).

## Piksel (*ang. pixel*)

Odpowiada pojedynczemu punktowi o ustalonym kolorze na wyświetlanym obrazie. Informacje o pikselach są zapisywane w buforze klatki.

## RGB (*ang. Red, Green, Blue*)

Najczęściej wykorzystywany zapis pikseli w pamięci, w którym każdy piksel jest reprezentowany przez 3 B (24 b) danych określających natężenie poszczególnych składowych światła widzialnego (czerwonej, zielonej i niebieskiej). Dla każdej składowej można używać wartości od 0 do 255.

# Sprzętowe wspomaganie operacji na buforze klatki

## Wielkość bufora klatki

Zależy od rozdzielczości wyświetlacza i reprezentacji pikseli. Dla wyświetlacza 1280x1024 i pikseli w 24-bitowej reprezentacji RGB wynosi ona $3932160\,B = 3840\,KiB = 3,75\,MiB$.

# Sprzętowe wspomaganie operacji na buforze klatki

## Wielkość bufora klatki

Zależy od rozdzielczości wyświetlacza i reprezentacji pikseli. Dla wyświetlacza 1280x1024 i pikseli w 24-bitowej reprezentacji RGB wynosi ona $3932160\,B = 3840\,KiB = 3,75\,MiB$.

## Manipulowanie pikselami

Praktycznie każda karta graficzna umożliwia modyfikowanie pojedynczych pikseli bezpośrednio w buforze klatki i odczytywanie zawartości bufora klatki.

# Sprzętowe wspomaganie operacji na buforze klatki

## Wielkość bufora klatki

Zależy od rozdzielczości wyświetlacza i reprezentacji pikseli. Dla wyświetlacza 1280x1024 i pikseli w 24-bitowej reprezentacji RGB wynosi ona $3932160\,B = 3840\,KiB = 3,75\,MiB$.

## Manipulowanie pikselami

Praktycznie każda karta graficzna umożliwia modyfikowanie pojedynczych pikseli bezpośrednio w buforze klatki i odczytywanie zawartości bufora klatki.

## Przyspieszanie grafiki 2D (*ang. 2D graphics acceleration*)

Sprzętowe wspomaganie przez kartę graficzną operacji na zawartości bufora klatki, takich jak rysowanie linii prostych, łuków, prostokątów itp., wypełnianie wzorem (*ang. pattern*) ograniczonych obszarów, manipulowanie prostokątnymi „oknami”.

# Tworzenie rysunków poza buforem klatki

**Piksmapa (*ang. pixmap*)**

Odwzorowany w pamięci operacyjnej komputera fragment bufora klatki odpowiadający prostokątnemu obszarowi na ekranie.

# Tworzenie rysunków poza buforem klatki

**Piksmapa (*ang. pixmap*)**

Odwzorowany w pamięci operacyjnej komputera fragment bufora klatki odpowiadający prostokątnemu obszarowi na ekranie.

Piksmapy można wykorzystywać do tworzenia ryskunków, podobnie jak bufor klatki, ale będą one wyświetlane dopiero po skopiowaniu zawartości piksmapy do bufora klatki.

# Tworzenie rysunków poza buforem klatki

**Piksmapa (*ang. pixmap*)**

Odwzorowany w pamięci operacyjnej komputera fragment bufora klatki odpowiadający prostokątnemu obszarowi na ekranie.

Piksmapy można wykorzystywać do tworzenia ryskunków, podobnie jak bufor klatki, ale będą one wyświetlane dopiero po skopiowaniu zawartości piksmapy do bufora klatki.

Niektóre karty graficzne wspomagają sprzętowo kopiowanie piksmap, zarówno między buforem klatki i pamięcią główną, jak i w obrębie bufora klatki.

# Tworzenie rysunków poza buforem klatki

**Piksmapa (*ang. pixmap*)**

Odwzorowany w pamięci operacyjnej komputera fragment bufora klatki odpowiadający prostokątnemu obszarowi na ekranie.

Piksmapy można wykorzystywać do tworzenia ryskunków, podobnie jak bufor klatki, ale będą one wyświetlane dopiero po skopiowaniu zawartości piksmapy do bufora klatki.

Niektóre karty graficzne wspomagają sprzętowo kopiowanie piksmap, zarówno między buforem klatki i pamięcią główną, jak i w obrębie bufora klatki.

Piksmapy pochodzące z jednego komputera nie muszą być poprawnie wyświetlane po umieszczeniu ich zawartości w buforze klatki karty graficznej na innym komputerze.

# Grafika trójwymiarowa (3D)

Większość współczesnych kart graficznych jest kostruowana z myślą o grafice trójwymiarowej (3D).

# Grafika trójwymiarowa (3D)

Większość współczesnych kart graficznych jest kostruowana z myślą o grafice trójwymiarowej (3D).

Pamięć obrazu jest wykorzystywana do zapisu trójwymiarowej sceny (w przestrzennym układzie współrzędnych), na podstawie której generowana jest zawartość bufora klatki.

# Grafika trójwymiarowa (3D)

Większość współczesnych kart graficznych jest kostruowana z myślą o grafice trójwymiarowej (3D).

Pamięć obrazu jest wykorzystywana do zapisu trójwymiarowej sceny (w przestrzennym układzie współrzędnych), na podstawie której generowana jest zawartość bufora klatki.

**Renderowanie (*ang. rendering*)**

Operacja, w czasie której zawartość bufora klatki jest tworzona na podstawie sceny 3D zapisanej w pamięci obrazu. Obejmuje przekształcenia brył, nakładanie tekstur, wygładzanie krawędzi, dodawanie efektów oświetlenia, mgły, przezroczystości itp. oraz rzutowanie.

# Grafika 3D w GUI

**Przyspieszanie grafiki 3D (*ang. 3D graphics acceleration*)**

Wspomaganie sprzętowe przez kartę graficzną przetwarzania scen 3D podczas renderowania.

# Grafika 3D w GUI

**Przyspieszanie grafiki 3D (*ang. 3D graphics acceleration*)**

Wspomaganie sprzętowe przez kartę graficzną przetwarzania scen 3D podczas renderowania.

Współczesne karty graficzne sprawniej przyspieszają grafikę 3D, niż grafikę 2D, więc często bardziej opłaca się wykorzystywać pełne możliwości karty zamiast manipulować „tylko” zawartością bufora klatki.

# Grafika 3D w GUI

**Przyspieszanie grafiki 3D (*ang. 3D graphics acceleration*)**

Wspomaganie sprzętowe przez kartę graficzną przetwarzania scen 3D podczas renderowania.

Współczesne karty graficzne sprawniej przyspieszają grafikę 3D, niż grafikę 2D, więc często bardziej opłaca się wykorzystywać pełne możliwości karty zamiast manipulować „tylko” zawartością bufora klatki.

Okna wykorzystywane przez aplikacje do komunikacji z użytkownikiem są traktowane jako elementy sceny 3D i renderowane z wykorzystaniem sprzętowego wspomagania ze strony karty graficznej.

# Od czego zależy w jakim stopniu wykorzystujemy sprzęt

To, w jakim stopniu można będzie wykorzystywać możliwości sprzętu w zakresie „przyspieszania” tworzenia grafiki, zależy od sterowników w jądrze systemu i (dla GNU/Linuksa) serwera X Windows oraz menedżera okien.

# Od czego zależy w jakim stopniu wykorzystujemy sprzęt

To, w jakim stopniu można będzie wykorzystywać możliwości sprzętu w zakresie „przyspieszania" tworzenia grafiki, zależy od sterowników w jądrze systemu i (dla GNU/Linuksa) serwera X Windows oraz menedżera okien.

Sterowniki karty graficznej w jądrze systemu i w serwerze X Windows muszą „wiedzieć" jak zaprogramować sprzęt do przeprowadzania poszczególnych operacji.

# Od czego zależy w jakim stopniu wykorzystujemy sprzęt

To, w jakim stopniu można będzie wykorzystywać możliwości sprzętu w zakresie „przyspieszania" tworzenia grafiki, zależy od sterowników w jądrze systemu i (dla GNU/Linuksa) serwera X Windows oraz menedżera okien.

Sterowniki karty graficznej w jądrze systemu i w serwerze X Windows muszą „wiedzieć" jak zaprogramować sprzęt do przeprowadzania poszczególnych operacji.

Serwer X Windows udostępnia menedżerowi okien i, za jego pośrednictwem, aplikacjom, szereg standardowych funkcji, z których mogą one korzystać w celu przeprowadzania podstawowych operacji związanych z tworzeniem grafiki.

# Rola biblioteki graficznej

Tworząc aplikację z reguły nie wiemy z góry na jakich systemach będzie ona uruchamiana:

- Nie wiemy jakie będą możliwości sprzętu.
- Nie wiemy jakie funkcje będzie udostępniał serwer X Windows.
- Nie wiemy jaki menedżer okien będzie używany.
- Nie wiemy na co będzie pozwalał menedżer okien.

# Rola biblioteki graficznej

Tworząc aplikację z reguły nie wiemy z góry na jakich systemach będzie ona uruchamiana:

- Nie wiemy jakie będą możliwości sprzętu.
- Nie wiemy jakie funkcje będzie udostępniał serwer X Windows.
- Nie wiemy jaki menedżer okien będzie używany.
- Nie wiemy na co będzie pozwalał menedżer okien.

Biblioteka graficzna rozpoznaje możliwości podsystemu graficznego i dostosowuje do nich działanie funkcji oraz klas, z których mogą korzystać aplikacje.

# Rola biblioteki graficznej

Tworząc aplikację z reguły nie wiemy z góry na jakich systemach będzie ona uruchamiana:

- Nie wiemy jakie będą możliwości sprzętu.
- Nie wiemy jakie funkcje będzie udostępniał serwer X Windows.
- Nie wiemy jaki menedżer okien będzie używany.
- Nie wiemy na co będzie pozwalał menedżer okien.

Biblioteka graficzna rozpoznaje możliwości podsystemu graficznego i dostosowuje do nich działanie funkcji oraz klas, z których mogą korzystać aplikacje.

Biblioteka graficzna pozwala korzystać z różnych podsystemów graficznych w jednolity sposób.

# Biblioteki graficzne i zdarzenia

Biblioteki graficzne dostarczają narzędzi ułatwiających obsługę zdarzeń na poziomie aplikacji.

# Biblioteki graficzne i zdarzenia

Biblioteki graficzne dostarczają narzędzi ułatwiających obsługę zdarzeń na poziomie aplikacji.

**Widżet (*ang. widget, WIndow gaDGET*)**

Obiekt reprezentujący podstawowy komponent GUI, taki jak okno, przycisk, suwak, rozwijane menu itp. W środowiskach MS Windows takie komponenty GUI nazywane są kontrolkami (*ang. control*).

# Biblioteki graficzne i zdarzenia

Biblioteki graficzne dostarczają narzędzi ułatwiających obsługę zdarzeń na poziomie aplikacji.

**Widżet (*ang. widget, WIndow gaDGET*)**

Obiekt reprezentujący podstawowy komponent GUI, taki jak okno, przycisk, suwak, rozwijane menu itp. W środowiskach MS Windows takie komponenty GUI nazywane są kontrolkami (*ang. control*).

Każdemu widżetowi przypisuje się prostokątny obszar ekranu i zdarzenia związane z tym obszarem są interpretowane w standardowy sposób w kontekście funkcji spełnianej przez widżet (np. kliknięcie lewym przyciskiem myszy w obszarze przypisanym widżetowi reprezentującemu przycisk może być interpretowane jako wciśnięcie przycisku).

# Podstawowe klasy z biblioteki Qt

## `QApplication`

- Obiekt tej klasy reprezentuje program (aplikację).
- Inicjuje zasoby wykorzystywane przez bibliotekę.
- Do konstruktora przekazuje się argumenty funkcji `main()`.
- Wywołanie metody `exec()` uruchamia pętlę obsługi zdarzeń.

# Podstawowe klasy z biblioteki Qt

## `QApplication`

- Obiekt tej klasy reprezentuje program (aplikację).
- Inicjuje zasoby wykorzystywane przez bibliotekę.
- Do konstruktora przekazuje się argumenty funkcji `main()`.
- Wywołanie metody `exec()` uruchamia pętlę obsługi zdarzeń.

## Pętla obsługi zdarzeń (*ang. event loop*)

Pętla, w której zdarzenia sygnalizowane przez różne części podsystemu graficznego są przetwarzane i przekazywane odpowiednim widżetom.

# Podstawowe klasy z biblioteki Qt

## `QApplication`

- Obiekt tej klasy reprezentuje program (aplikację).
- Inicjuje zasoby wykorzystywane przez bibliotekę.
- Do konstruktora przekazuje się argumenty funkcji `main()`.
- Wywołanie metody `exec()` uruchamia pętlę obsługi zdarzeń.

## Pętla obsługi zdarzeń (*ang. event loop*)

Pętla, w której zdarzenia sygnalizowane przez różne części podsystemu graficznego są przetwarzane i przekazywane odpowiednim widżetom.

Obiekt klasy `QApplication` musi być utworzony przed jakimkolwiek innym obiektem związanym z GUI i może być tylko jeden.

# Podstawowe klasy z biblioteki Qt c. d.

## QObject

Od tej klasy pochodzi większość klas dostępnych w bibliotece Qt. Definiuje ona interfejs umożliwiający przekazywanie informacji o zdarzeniach od jednego komponentu GUI do drugiego (jego konstrukcja wykracza poza standard C++).

# Podstawowe klasy z biblioteki Qt c. d.

## QObject

Od tej klasy pochodzi większość klas dostępnych w bibliotece Qt. Definiuje ona interfejs umożliwiający przekazywanie informacji o zdarzeniach od jednego komponentu GUI do drugiego (jego konstrukcja wykracza poza standard C++).

## QPaintDevice

Obiekty tej klasy reprezentują prostokątne powierzchnie, na których można rysować (np. prostokątne wycinki ekranu lub piksmapy).

# Podstawowe klasy z biblioteki Qt c. d.

## QObject

Od tej klasy pochodzi większość klas dostępnych w bibliotece Qt. Definiuje ona interfejs umożliwiający przekazywanie informacji o zdarzeniach od jednego komponentu GUI do drugiego (jego konstrukcja wykracza poza standard C++).

## QPaintDevice

Obiekty tej klasy reprezentują prostokątne powierzchnie, na których można rysować (np. prostokątne wycinki ekranu lub piksmapy).

Z każdą taką powierzchnią jest związana prostokątna siatka współrzędnych (o wartościach całkowitych nieujemnych) i punktem o współrzędnych $[0, 0]$ w lewym górnym rogu. Każdy punkt tej siatki reprezentuje jeden piksel.

# Podstawowe klasy z biblioteki Qt c. d.

## `QWidget`

Klasa pochodna od `QObject` i `QPaintDevice`. Od tej klasy pochodzą wszystkie klasy reprezentujące typowe komponenty GUI. Własne komponenty GUI najłatwiej tworzy się poprzez definiowanie własnych klas pochodnych względem `QWidget`.

# Podstawowe klasy z biblioteki Qt c. d.

## QWidget

Klasa pochodna od `QObject` i `QPaintDevice`. Od tej klasy pochodzą wszystkie klasy reprezentujące typowe komponenty GUI. Własne komponenty GUI najłatwiej tworzy się poprzez definiowanie własnych klas pochodnych względem `QWidget`.

## QPainter

Obiekty tej klasy służą do przeprowadzania podstawowych operacji graficznych (takich, jak rysowanie podstawowych elementów grafiki) na „powierzchniach” reprezentowanych przez obiekty klasy `QPaintDevice`.

# Podstawowe klasy z biblioteki Qt c. d.

## QWidget

Klasa pochodna od `QObject` i `QPaintDevice`. Od tej klasy pochodzą wszystkie klasy reprezentujące typowe komponenty GUI. Własne komponenty GUI najłatwiej tworzy się poprzez definiowanie własnych klas pochodnych względem `QWidget`.

## QPainter

Obiekty tej klasy służą do przeprowadzania podstawowych operacji graficznych (takich, jak rysowanie podstawowych elementów grafiki) na „powierzchniach” reprezentowanych przez obiekty klasy `QPaintDevice`.

W celu narysowania czegoś tworzy się obiekt klasy `QPainter` związany z obiektem klasy `QPaintDevice` reprezentującym powierzchnię, na której ma powstać rysunek.

# Najprosztszy program wykorzystujący Qt

```
#include <QApplication>
#include <QLabel>

int main(int argc, char *argv[])
{
        QApplication app(argc, argv);

        QLabel hello("Hello world!");
        hello.resize(100, 30);

        hello.show();
        return app.exec();  // Start pętli obsługi zdarzeń
}
```

# Najprosztszy program wykorzystujący Qt

```
#include <QApplication>
#include <QLabel>

int main(int argc, char *argv[])
{
        QApplication app(argc, argv);

        QLabel hello("Hello world!");
        hello.resize(100, 30);

        hello.show();
        return app.exec();  // Start pętli obsługi zdarzeń
}
```

- Argumenty funkcji `main()` są przekazywane do konstruktora obiektu app (inicjalizacja struktur danych Qt).
- Obiekt `hello` reprezentuje etykietę (*ang. label*), czyli prostokątny wycinek ekranu z napisem.
- Metody `resize()` i `show()` pochodzą z klasy `QWidget`, która jest klasą bazową dla `QLabel`.

# Kompilowanie programów używających Qt

Interfejs obsługi zdarzeń wykorzystywany przez Qt wykracza poza standard C++, więc programy używające Qt muszą być kompilowane w specjalny sposób.

# Kompilowanie programów używających Qt

Interfejs obsługi zdarzeń wykorzystywany przez Qt wykracza poza standard C++, więc programy używające Qt muszą być kompilowane w specjalny sposób.

1. Tworzymy katalog o nazwie, która ma być nazwą pliku z programem.
2. W tym katalogu tworzymy (przynajmniej) plik o nazwie `main.cpp` zawierający funkcję `main()`. Kod źródłowy programu może być zapisany w wielu plikach (w tym katalogu).
3. W katalogu z plikami źródłowymi wykonujemy polecenia:

```
$ qmake -project
$ qmake
$ make
```

# Kompilowanie programów używających Qt c. d.

**`qmake -project`**

Tworzy plik z rozszerzeniem `.pro` zawierający informacje m. in. o tym z jakich plików składa się program.

# Kompilowanie programów używających Qt c. d.

## `qmake -project`

Tworzy plik z rozszerzeniem `.pro` zawierający informacje m. in. o tym z jakich plików składa się program.

## `qmake`

Przygotowuje plik `Makefile` dla polecenia `make`, zawierający instrukcje kompilacji programu.

# Kompilowanie programów używających Qt c. d.

## `qmake -project`

Tworzy plik z rozszerzeniem `.pro` zawierający informacje m. in. o tym z jakich plików składa się program.

## `qmake`

Przygotowuje plik `Makefile` dla polecenia `make`, zawierający instrukcje kompilacji programu.

## `make`

Wykonuje instrukcje zapisane w pliku `Makefile`. Poza uruchamianiem kompilatora C++ z odpowiednimi opcjami wykonuje programy przekształcające niestandardowy kod związany Qt na standardowy kod w C++.