Biblioteki graficzne, Qt – najważniejsze własności

Bogdan Kreczmer

bogdan.kreczmer@pwr.edu.pl

Katedra Cybernetyki i Robotyki
Wydziału Elektroniki
Politechniki Wrocławskiej

Kurs: Wizualizacja danych sensorycznych

Niniejsza prezentacja została wykonana przy użyciu systemu składu PDFLaTeX *oraz stylu* `beamer`, *którego autorem jest Till Tantau.*

Strona domowa projektu `Beamer`:
`http://latex-beamer.sourceforge.net`

Podstawowe cechy

Aplikacja tekstowa jednowątkowa bez obsługi sygnałów (w sensie Uniksa)

- Możemy dobrze określić jak ma wyglądać przepływ sterowania. W zależności od danych otrzymanych z wejścia przepływ ten może być modyfikowany. Niemniej o aplikacji takiej możemy myśleć jako o zbiorze sekwencyjnie wykonywanych funkcji lub procedur.
- Aplikacja *decyduje* kiedy rozpocząć interakcję z użytkownikiem.
- Jeśli wchodzimy w tryb interakcji, to aplikacja *"zawiesza"* się w oczekiwaniu na udostępnienie przez system danych wprowadzonych przez użytkownika.
- Aplikacja jawnie decyduje kiedy coś ma pojawić się na ekranie. Poza tymi momentami nie zmienia zawartości ekranu.

Podstawowe cechy

Aplikacja tekstowa jednowątkowa bez obsługi sygnałów (w sensie Uniksa)

- Możemy dobrze określić jak ma wyglądać przepływ sterowania. W zależności od danych otrzymanych z wejścia przepływ ten może być modyfikowany. Niemniej o aplikacji takiej możemy myśleć jako o zbiorze sekwencyjnie wykonywanych funkcji lub procedur.
- Aplikacja *decyduje* kiedy rozpocząć interakcję z użytkownikiem.
- Jeśli wchodzimy w tryb interakcji, to aplikacja *"zawiesza"* się w oczekiwaniu na udostępnienie przez system danych wprowadzonych przez użytkownika.
- Aplikacja jawnie decyduje kiedy coś ma pojawić się na ekranie. Poza tymi momentami nie zmienia zawartości ekranu.

Podstawowe cechy

Aplikacja tekstowa jednowątkowa bez obsługi sygnałów (w sensie Uniksa)

- Możemy dobrze określić jak ma wyglądać przepływ sterowania. W zależności od danych otrzymanych z wejścia przepływ ten może być modyfikowany. Niemniej o aplikacji takiej możemy myśleć jako o zbiorze sekwencyjnie wykonywanych funkcji lub procedur.
- Aplikacja *decyduje* kiedy rozpocząć interakcję z użytkownikiem.
- Jeśli wchodzimy w tryb interakcji, to aplikacja *"zawiesza"* się w oczekiwaniu na udostępnienie przez system danych wprowadzonych przez użytkownika.
- Aplikacja jawnie decyduje kiedy coś ma pojawić się na ekranie. Poza tymi momentami nie zmienia zawartości ekranu.

Podstawowe cechy

Aplikacja tekstowa jednowątkowa bez obsługi sygnałów (w sensie Uniksa)

- Możemy dobrze określić jak ma wyglądać przepływ sterowania. W zależności od danych otrzymanych z wejścia przepływ ten może być modyfikowany. Niemniej o aplikacji takiej możemy myśleć jako o zbiorze sekwencyjnie wykonywanych funkcji lub procedur.
- Aplikacja *decyduje* kiedy rozpocząć interakcję z użytkownikiem.
- Jeśli wchodzimy w tryb interakcji, to aplikacja *"zawiesza"* się w oczekiwaniu na udostępnienie przez system danych wprowadzonych przez użytkownika.
- Aplikacja jawnie decyduje kiedy coś ma pojawić się na ekranie. Poza tymi momentami nie zmienia zawartości ekranu.

Podstawowe cechy

Aplikacja tekstowa jednowątkowa bez obsługi sygnałów (w sensie Uniksa)

- Możemy dobrze określić jak ma wyglądać przepływ sterowania. W zależności od danych otrzymanych z wejścia przepływ ten może być modyfikowany. Niemniej o aplikacji takiej możemy myśleć jako o zbiorze sekwencyjnie wykonywanych funkcji lub procedur.
- Aplikacja *decyduje* kiedy rozpocząć interakcję z użytkownikiem.
- Jeśli wchodzimy w tryb interakcji, to aplikacja *"zawiesza"* się w oczekiwaniu na udostępnienie przez system danych wprowadzonych przez użytkownika.
- Aplikacja jawnie decyduje kiedy coś ma pojawić się na ekranie. Poza tymi momentami nie zmienia zawartości ekranu.

Podstawowe cechy

Aplikacja tekstowa jednowątkowa bez obsługi sygnałów (w sensie Uniksa)

- Możemy dobrze określić jak ma wyglądać przepływ sterowania. W zależności od danych otrzymanych z wejścia przepływ ten może być modyfikowany. Niemniej o aplikacji takiej możemy myśleć jako o zbiorze sekwencyjnie wykonywanych funkcji lub procedur.
- Aplikacja *decyduje* kiedy rozpocząć interakcję z użytkownikiem.
- Jeśli wchodzimy w tryb interakcji, to aplikacja *"zawiesza"* się w oczekiwaniu na udostępnienie przez system danych wprowadzonych przez użytkownika.
- Aplikacja jawnie decyduje kiedy coś ma pojawić się na ekranie. Poza tymi momentami nie zmienia zawartości ekranu.

Podstawowe cechy

Aplikacja tekstowa jednowątkowa z obsługą sygnałów (w sensie Uniksa)

- Zazwyczaj możemy dość dobrze określić przepływ sterowania. Odstępstwem jest wywołanie funkcji w odpowiedzi na pojawienie się sygnału.
- Zgłoszenie sygnału (w systemie Unix/Linux) ma charakter asynchroniczny. Może pojawić się w dowolnym momencie.
- Obsługę sygnału należy rozumieć tak samo jak obsługę przerwania. W tym też sensie realizacja odpowiedzi na sygnał nie powinna być czasochłonna.
- Obsługa sygnałów nie powinna zdominować działania całego programu. Dzięki temu pozostałe cechy programu są analogiczne jak te, które miała aplikacja bez obsługi sygnału.

Podstawowe cechy

Aplikacja tekstowa jednowątkowa z obsługą sygnałów (w sensie Uniksa)

- Zazwyczaj możemy dość dobrze określić przepływ sterowania. Odstępstwem jest wywołanie funkcji w odpowiedzi na pojawienie się sygnału.
- Zgłoszenie sygnału (w systemie Unix/Linux) ma charakter asynchroniczny. Może pojawić się w dowolnym momencie.
- Obsługę sygnału należy rozumieć tak samo jak obsługę przerwania. W tym też sensie realizacja odpowiedzi na sygnał nie powinna być czasochłonna.
- Obsługa sygnałów nie powinna zdominować działania całego programu. Dzięki temu pozostałe cechy programu są analogiczne jak te, które miała aplikacja bez obsługi sygnału.

Podstawowe cechy

Aplikacja tekstowa jednowątkowa z obsługą sygnałów (w sensie Uniksa)

- Zazwyczaj możemy dość dobrze określić przepływ sterowania. Odstępstwem jest wywołanie funkcji w odpowiedzi na pojawienie się sygnału.
- Zgłoszenie sygnału (w systemie Unix/Linux) ma charakter asynchroniczny. Może pojawić się w dowolnym momencie.
- Obsługę sygnału należy rozumieć tak samo jak obsługę przerwania. W tym też sensie realizacja odpowiedzi na sygnał nie powinna być czasochłonna.
- Obsługa sygnałów nie powinna zdominować działania całego programu. Dzięki temu pozostałe cechy programu są analogiczne jak te, które miała aplikacja bez obsługi sygnału.

Podstawowe cechy

Aplikacja tekstowa jednowątkowa z obsługą sygnałów (w sensie Uniksa)

- Zazwyczaj możemy dość dobrze określić przepływ sterowania. Odstępstwem jest wywołanie funkcji w odpowiedzi na pojawienie się sygnału.
- Zgłoszenie sygnału (w systemie Unix/Linux) ma charakter asynchroniczny. Może pojawić się w dowolnym momencie.
- Obsługę sygnału należy rozumieć tak samo jak obsługę przerwania. W tym też sensie realizacja odpowiedzi na sygnał nie powinna być czasochłonna.
- Obsługa sygnałów nie powinna zdominować działania całego programu. Dzięki temu pozostałe cechy programu są analogiczne jak te, które miała aplikacja bez obsługi sygnału.

Podstawowe cechy

Aplikacja tekstowa jednowątkowa z obsługą sygnałów (w sensie Uniksa)

- Zazwyczaj możemy dość dobrze określić przepływ sterowania. Odstępstwem jest wywołanie funkcji w odpowiedzi na pojawienie się sygnału.
- Zgłoszenie sygnału (w systemie Unix/Linux) ma charakter asynchroniczny. Może pojawić się w dowolnym momencie.
- Obsługę sygnału należy rozumieć tak samo jak obsługę przerwania. W tym też sensie realizacja odpowiedzi na sygnał nie powinna być czasochłonna.
- Obsługa sygnałów nie powinna zdominować działania całego programu. Dzięki temu pozostałe cechy programu są analogiczne jak te, które miała aplikacja bez obsługi sygnału.

Podstawowe cechy

Aplikacja tekstowa jednowątkowa z obsługą sygnałów (w sensie Uniksa)

- Zazwyczaj możemy dość dobrze określić przepływ sterowania. Odstępstwem jest wywołanie funkcji w odpowiedzi na pojawienie się sygnału.
- Zgłoszenie sygnału (w systemie Unix/Linux) ma charakter asynchroniczny. Może pojawić się w dowolnym momencie.
- Obsługę sygnału należy rozumieć tak samo jak obsługę przerwania. W tym też sensie realizacja odpowiedzi na sygnał nie powinna być czasochłonna.
- Obsługa sygnałów nie powinna zdominować działania całego programu. Dzięki temu pozostałe cechy programu są analogiczne jak te, które miała aplikacja bez obsługi sygnału.

Podstawowe cechy

Aplikacja graficzna w systemach okienkowych

- Program składa się z części, która inicjuje działanie aplikacji. Reszta zaś to zbiór funkcji, które wywoływane są w odpowiedzi na ***zdarzenia***. *Tak więc aplikacja staje się zbiorem procedur obsługi* ***zdarzeń***.
- Jawnie możemy określić przepływ sterowania tylko w niektórych partiach programu, które dotyczą funkcji wywoływanych w odpowiedzi na dane zdarzenie.
- Dobrze skonstruowana aplikacja powinna być cały czas w interakcji z otoczeniem programu, tzn. powinna realizować obsługę zdarzeń, np. odrysowania, aby nie powodować *zamrażania* aplikacji.
- Aby nie *zamrażać* aplikacji, w trakcie pracy czasochłonnej procedury/funkcji zaleca się co pewien czas *oddawać* sterowanie, tak aby pozostałe zdarzenia mogły być również obsługiwane.

Podstawowe cechy

Aplikacja graficzna w systemach okienkowych

- Program składa się z części, która inicjuje działanie aplikacji. Reszta zaś to zbiór funkcji, które wywoływane są w odpowiedzi na ***zdarzenia***. *Tak więc aplikacja staje się zbiorem procedur obsługi* ***zdarzeń***.
- Jawnie możemy określić przepływ sterowania tylko w niektórych partiach programu, które dotyczą funkcji wywoływanych w odpowiedzi na dane zdarzenie.
- Dobrze skonstruowana aplikacja powinna być cały czas w interakcji z otoczeniem programu, tzn. powinna realizować obsługę zdarzeń, np. odrysowania, aby nie powodować *zamrażania* aplikacji.
- Aby nie *zamrażać* aplikacji, w trakcie pracy czasochłonnej procedury/funkcji zaleca się co pewien czas *oddawać* sterowanie, tak aby pozostałe zdarzenia mogły być również obsługiwane.

Podstawowe cechy

Aplikacja graficzna w systemach okienkowych

- Program składa się z części, która inicjuje działanie aplikacji. Reszta zaś to zbiór funkcji, które wywoływane są w odpowiedzi na ***zdarzenia***. *Tak więc aplikacja staje się zbiorem procedur obsługi* ***zdarzeń***.
- Jawnie możemy określić przepływ sterowania tylko w niektórych partiach programu, które dotyczą funkcji wywoływanych w odpowiedzi na dane zdarzenie.
- Dobrze skonstruowana aplikacja powinna być cały czas w interakcji z otoczeniem programu, tzn. powinna realizować obsługę zdarzeń, np. odrysowania, aby nie powodować *zamrażania* aplikacji.
- Aby nie *zamrażać* aplikacji, w trakcie pracy czasochłonnej procedury/funkcji zaleca się co pewien czas *oddawać* sterowanie, tak aby pozostałe zdarzenia mogły być również obsługiwane.

Podstawowe cechy

Aplikacja graficzna w systemach okienkowych

- Program składa się z części, która inicjuje działanie aplikacji. Reszta zaś to zbiór funkcji, które wywoływane są w odpowiedzi na ***zdarzenia***. *Tak więc aplikacja staje się zbiorem procedur obsługi* ***zdarzeń***.
- Jawnie możemy określić przepływ sterowania tylko w niektórych partiach programu, które dotyczą funkcji wywoływanych w odpowiedzi na dane zdarzenie.
- Dobrze skonstruowana aplikacja powinna być cały czas w interakcji z otoczeniem programu, tzn. powinna realizować obsługę zdarzeń, np. odrysowania, aby nie powodować *zamrażania* aplikacji.
- Aby nie *zamrażać* aplikacji, w trakcie pracy czasochłonnej procedury/funkcji zaleca się co pewien czas *oddawać* sterowanie, tak aby pozostałe zdarzenia mogły być również obsługiwane.

Podstawowe cechy

Aplikacja graficzna w systemach okienkowych

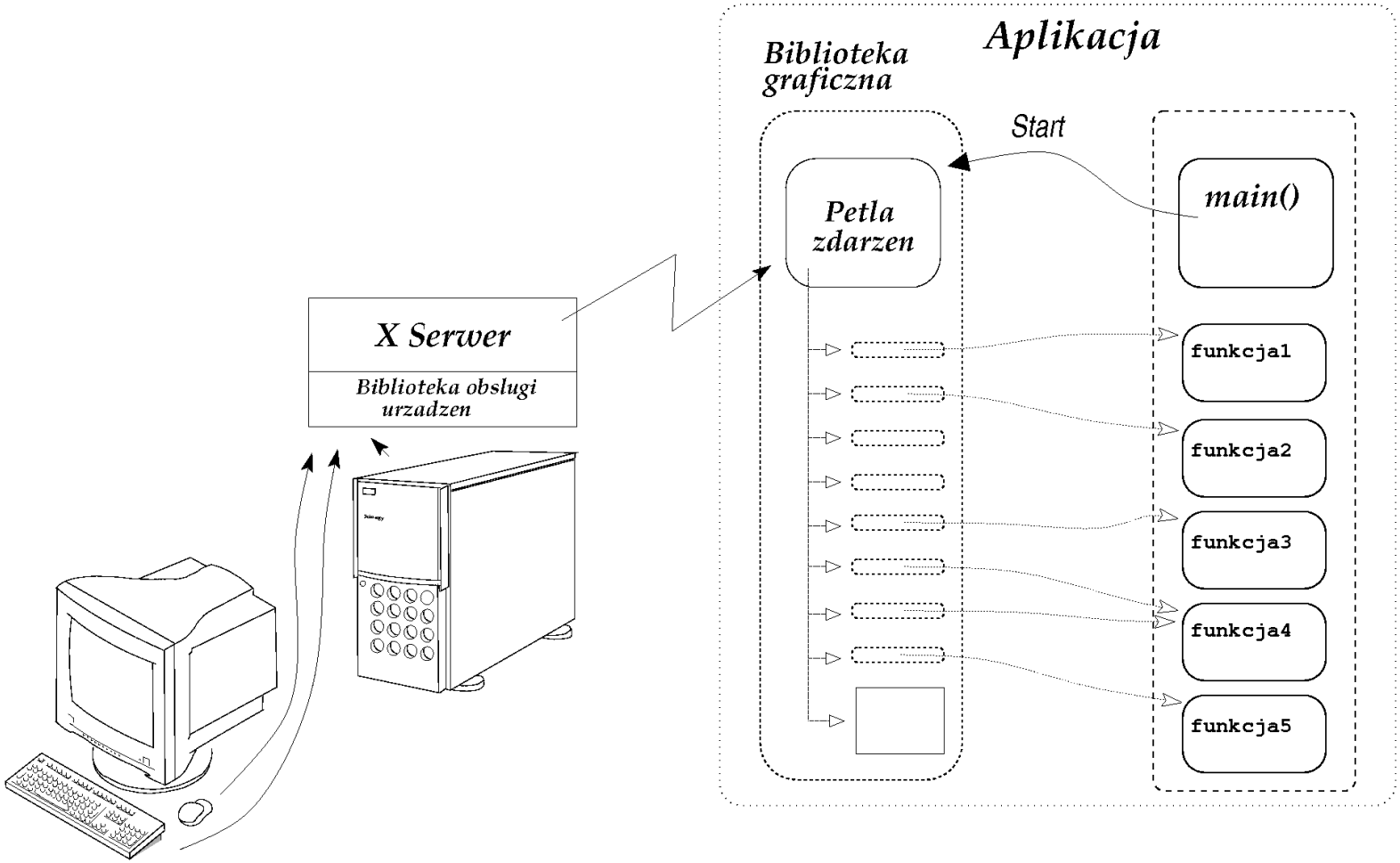
- Program składa się z części, która inicjuje działanie aplikacji. Reszta zaś to zbiór funkcji, które wywoływane są w odpowiedzi na ***zdarzenia***. *Tak więc aplikacja staje się zbiorem procedur obsługi* ***zdarzeń***.
- Jawnie możemy określić przepływ sterowania tylko w niektórych partiach programu, które dotyczą funkcji wywoływanych w odpowiedzi na dane zdarzenie.
- Dobrze skonstruowana aplikacja powinna być cały czas w interakcji z otoczeniem programu, tzn. powinna realizować obsługę zdarzeń, np. odrysowania, aby nie powodować *zamrażania* aplikacji.
- Aby nie *zamrażać* aplikacji, w trakcie pracy czasochłonnej procedury/funkcji zaleca się co pewien czas *oddawać* sterowanie, tak aby pozostałe zdarzenia mogły być również obsługiwane.

Podstawowe cechy

Aplikacja graficzna w systemach okienkowych

- Program składa się z części, która inicjuje działanie aplikacji. Reszta zaś to zbiór funkcji, które wywoływane są w odpowiedzi na ***zdarzenia***. *Tak więc aplikacja staje się zbiorem procedur obsługi* ***zdarzeń***.
- Jawnie możemy określić przepływ sterowania tylko w niektórych partiach programu, które dotyczą funkcji wywoływanych w odpowiedzi na dane zdarzenie.
- Dobrze skonstruowana aplikacja powinna być cały czas w interakcji z otoczeniem programu, tzn. powinna realizować obsługę zdarzeń, np. odrysowania, aby nie powodować *zamrażania* aplikacji.
- Aby nie *zamrażać* aplikacji, w trakcie pracy czasochłonnej procedury/funkcji zaleca się co pewien czas *oddawać* sterowanie, tak aby pozostałe zdarzenia mogły być również obsługiwane.

Aplikacja w graficznym systemie okienkowym



Podstawowe cechy

Aplikacja graficzna w systemach okienkowych c.d.

- Na ekranie nigdy jawnie niczego nie rysujemy.
- Do rysowania programista dostarcza procedurę/funkcję, która musi być gotowa do odrysowania zawartości okienka w każdym momencie działania aplikacji.
- Można jawnie zalecić uruchomienie procedury rysowania. Jednak oprócz tego system okienkowy może uruchomić procedurę odrysowania również w innych momentach.

Podstawowe cechy

Aplikacja graficzna w systemach okienkowych c.d.

- Na ekranie nigdy jawnie niczego nie rysujemy.
- Do rysowania programista dostarcza procedurę/funkcję, która musi być gotowa do odrysowania zawartości okienka w każdym momencie działania aplikacji.
- Można jawnie zalecić uruchomienie procedury rysowania. Jednak oprócz tego system okienkowy może uruchomić procedurę odrysowania również w innych momentach.

Podstawowe cechy

Aplikacja graficzna w systemach okienkowych c.d.

- Na ekranie nigdy jawnie niczego nie rysujemy.
- Do rysowania programista dostarcza procedurę/funkcję, która musi być gotowa do odrysowania zawartości okienka w każdym momencie działania aplikacji.
- Można jawnie zalecić uruchomienie procedury rysowania. Jednak oprócz tego system okienkowy może uruchomić procedurę odrysowania również w innych momentach.

Podstawowe cechy

Aplikacja graficzna w systemach okienkowych c.d.

- Na ekranie nigdy jawnie niczego nie rysujemy.
- Do rysowania programista dostarcza procedurę/funkcję, która musi być gotowa do odrysowania zawartości okienka w każdym momencie działania aplikacji.
- Można jawnie zalecić uruchomienie procedury rysowania. Jednak oprócz tego system okienkowy może uruchomić procedurę odrysowania również w innych momentach.

Podstawowe cechy

Aplikacja graficzna w systemach okienkowych c.d.

- Na ekranie nigdy jawnie niczego nie rysujemy.
- Do rysowania programista dostarcza procedurę/funkcję, która musi być gotowa do odrysowania zawartości okienka w każdym momencie działania aplikacji.
- Można jawnie zalecić uruchomienie procedury rysowania. Jednak oprócz tego system okienkowy może uruchomić procedurę odrysowania również w innych momentach.

Podstawowe cechy konstrukcji bibliotek okienkowych

- ⋆ Biblioteka dostarcza mechanizmy obsługi zdarzeń oraz mechanizmy tworzenia elementów graficznych aplikacji.
- ⋆ Biblioteka dostarcza funkcje, które realizują domyślną obsługę zdarzeń.
- ⋆ Programista pisze kod, który odpowiedzialny jest za utworzenie elementów graficznych oraz dostarcza funkcje, które modyfikują sposób obsługi zdarzeń. Istnienie tych funkcji trzeba odpowiednio zasygnalizować w danym systemie.

Podstawowe cechy konstrukcji bibliotek okienkowych

- ⋆ Biblioteka dostarcza mechanizmy obsługi zdarzeń oraz mechanizmy tworzenia elementów graficznych aplikacji.
- ⋆ Biblioteka dostarcza funkcje, które realizują domyślną obsługę zdarzeń.
- ⋆ Programista pisze kod, który odpowiedzialny jest za utworzenie elementów graficznych oraz dostarcza funkcje, które modyfikują sposób obsługi zdarzeń. Istnienie tych funkcji trzeba odpowiednio zasygnalizować w danym systemie.

Podstawowe cechy konstrukcji bibliotek okienkowych

- ⋆ Biblioteka dostarcza mechanizmy obsługi zdarzeń oraz mechanizmy tworzenia elementów graficznych aplikacji.
- ⋆ Biblioteka dostarcza funkcje, które realizują domyślną obsługę zdarzeń.
- ⋆ Programista pisze kod, który odpowiedzialny jest za utworzenie elementów graficznych oraz dostarcza funkcje, które modyfikują sposób obsługi zdarzeń. Istnienie tych funkcji trzeba odpowiednio zasygnalizować w danym systemie.

Podstawowe cechy konstrukcji bibliotek okienkowych

- ⋆ Biblioteka dostarcza mechanizmy obsługi zdarzeń oraz mechanizmy tworzenia elementów graficznych aplikacji.
- ⋆ Biblioteka dostarcza funkcje, które realizują domyślną obsługę zdarzeń.
- ⋆ Programista pisze kod, który odpowiedzialny jest za utworzenie elementów graficznych oraz dostarcza funkcje, które modyfikują sposób obsługi zdarzeń. Istnienie tych funkcji trzeba odpowiednio zasygnalizować w danym systemie.

Podstawowe cechy konstrukcji bibliotek okienkowych

- ⋆ Biblioteka dostarcza mechanizmy obsługi zdarzeń oraz mechanizmy tworzenia elementów graficznych aplikacji.
- ⋆ Biblioteka dostarcza funkcje, które realizują domyślną obsługę zdarzeń.
- ⋆ Programista pisze kod, który odpowiedzialny jest za utworzenie elementów graficznych oraz dostarcza funkcje, które modyfikują sposób obsługi zdarzeń. Istnienie tych funkcji trzeba odpowiednio zasygnalizować w danym systemie.

Tworzenie natywnych aplikacji okienkowych

Aplikacje natywne dla danego systemu okienkowego tworzy się z wykorzystaniem API (*Application Programming Interface*), które zazwyczaj dostarczane jest z danym systemem okienkowym.

Przykłady API:

- *Win32* (MS Windows) – jest to zbiór funkcji napisanych w języku C zawartych w bibliotekach DLL. Najważniejsze składniki: kernel32.dll, user32.dll, gdi32.dll
- *Motif* (X Window) – de facto system X Window nie posiada API, a jedynie zbiór pakietów narzędziowych (np. *Xt*). Jednak ze względu na to, że *Motif* zyskał powszechne uznanie i jest przez wielu producentów dostarczany wraz z systemem X Window, to zwykło uważać się, że wyznacza on API dla tego systemu.
- *Carbon* (MacOS X) – nie jest to czyste API, a raczej baza programowa (ang. *framework*).

Tworzenie natywnych aplikacji okienkowych

Aplikacje natywne dla danego systemu okienkowego tworzy się z wykorzystaniem API (*Application Programming Interface*), które zazwyczaj dostarczane jest z danym systemem okienkowym.

Przykłady API:

- *Win32* (MS Windows) – jest to zbiór funkcji napisanych w języku C zawartych w bibliotekach DLL. Najważniejsze składniki: kernel32.dll, user32.dll, gdi32.dll
- *Motif* (X Window) – de facto system X Window nie posiada API, a jedynie zbiór pakietów narzędziowych (np. *Xt*). Jednak ze względu na to, że *Motif* zyskał powszechne uznanie i jest przez wielu producentów dostarczany wraz z systemem X Window, to zwykło uważać się, że wyznacza on API dla tego systemu.
- *Carbon* (MacOS X) – nie jest to czyste API, a raczej baza programowa (ang. *framework*).

Tworzenie natywnych aplikacji okienkowych

Aplikacje natywne dla danego systemu okienkowego tworzy się z wykorzystaniem API (*Application Programming Interface*), które zazwyczaj dostarczane jest z danym systemem okienkowym.

Przykłady API:

- *Win32* (MS Windows) – jest to zbiór funkcji napisanych w języku C zawartych w bibliotekach DLL. Najważniejsze składniki: kernel32.dll, user32.dll, gdi32.dll
- *Motif* (X Window) – de facto system X Window nie posiada API, a jedynie zbiór pakietów narzędziowych (np. *Xt*). Jednak ze względu na to, że *Motif* zyskał powszechne uznanie i jest przez wielu producentów dostarczany wraz z systemem X Window, to zwykło uważać się, że wyznacza on API dla tego systemu.
- *Carbon* (MacOS X) – nie jest to czyste API, a raczej baza programowa (ang. *framework*).

Tworzenie natywnych aplikacji okienkowych

Aplikacje natywne dla danego systemu okienkowego tworzy się z wykorzystaniem API (*Application Programming Interface*), które zazwyczaj dostarczane jest z danym systemem okienkowym.

Przykłady API:

- *Win32* (MS Windows) – jest to zbiór funkcji napisanych w języku C zawartych w bibliotekach DLL. Najważniejsze składniki: kernel32.dll, user32.dll, gdi32.dll
- *Motif* (X Window) – de facto system X Window nie posiada API, a jedynie zbiór pakietów narzędziowych (np. *Xt*). Jednak ze względu na to, że *Motif* zyskał powszechne uznanie i jest przez wielu producentów dostarczany wraz z systemem X Window, to zwykło uważać się, że wyznacza on API dla tego systemu.
- *Carbon* (MacOS X) – nie jest to czyste API, a raczej baza programowa (ang. *framework*).

Tworzenie natywnych aplikacji okienkowych

Aplikacje natywne dla danego systemu okienkowego tworzy się z wykorzystaniem API (*Application Programming Interface*), które zazwyczaj dostarczane jest z danym systemem okienkowym.

Przykłady API:

- *Win32* (MS Windows) – jest to zbiór funkcji napisanych w języku C zawartych w bibliotekach DLL. Najważniejsze składniki: kernel32.dll, user32.dll, gdi32.dll
- *Motif* (X Window) – de facto system X Window nie posiada API, a jedynie zbiór pakietów narzędziowych (np. *Xt*). Jednak ze względu na to, że *Motif* zyskał powszechne uznanie i jest przez wielu producentów dostarczany wraz z systemem X Window, to zwykło uważać się, że wyznacza on API dla tego systemu.
- *Carbon* (MacOS X) – nie jest to czyste API, a raczej baza programowa (ang. *framework*).

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- *Nakładka na natywne API*
- *Emulacja wybranego API*
- *Emulacja GUI*

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API*
- ⋆ *Emulacja wybranego API*
- ⋆ *Emulacja GUI*

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API*
- ⋆ *Emulacja wybranego API*
- ⋆ *Emulacja GUI*

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API*
- ⋆ *Emulacja wybranego API*
- ⋆ *Emulacja GUI*

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API*
- ⋆ *Emulacja wybranego API*
- ⋆ *Emulacja GUI*

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API*
- ⋆ *Emulacja wybranego API*
- ⋆ *Emulacja GUI*

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API* – biblioteka odwołuje się do systemu okienkowego poprzez natywne API.
  - Zalety: stosunkowo łatwa implementacja biblioteki.
  - Wady:
    — mniejsza efektywność kodu (odwoływanie się do systemu okienkowego poprzez warstwę pośrednią)
    — aby zapewnić przenośność, elementy graficzne i dostępne mechanizmy muszą być ograniczone do tych, które występują na wszystkich platformach.

  Przykładem tego typu biblioteki było *wxWindows* (obecnie *wxWidgets*).
- ⋆ *Emulacja wybranego API*
- ⋆ *Emulacja GUI*

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API* – biblioteka odwołuje się do systemu okienkowego poprzez natywne API.
  - Zalety: stosunkowo łatwa implementacja biblioteki.
  - Wady:
    — mniejsza efektywność kodu (odwoływanie się do systemu okienkowego poprzez warstwę pośrednią)
    — aby zapewnić przenośność, elementy graficzne i dostępne mechanizmy muszą być ograniczone do tych, które występują na wszystkich platformach.

  Przykładem tego typu biblioteki było *wxWindows* (obecnie *wxWidgets*).
- ⋆ *Emulacja wybranego API*
- ⋆ *Emulacja GUI*

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API* – biblioteka odwołuje się do systemu okienkowego poprzez natywne API.
  - Zalety: stosunkowo łatwa implementacja biblioteki.
  - Wady:
    — mniejsza efektywność kodu (odwoływanie się do systemu okienkowego poprzez warstwę pośrednią)
    — aby zapewnić przenośność, elementy graficzne i dostępne mechanizmy muszą być ograniczone do tych, które występują na wszystkich platformach.

  Przykładem tego typu biblioteki było *wxWindows* (obecnie *wxWidgets*).
- ⋆ *Emulacja wybranego API*
- ⋆ *Emulacja GUI*

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API* – biblioteka odwołuje się do systemu okienkowego poprzez natywne API.
  - Zalety: stosunkowo łatwa implementacja biblioteki.
  - Wady:
    — mniejsza efektywność kodu (odwoływanie się do systemu okienkowego poprzez warstwę pośrednią)
    — aby zapewnić przenośność, elementy graficzne i dostępne mechanizmy muszą być ograniczone do tych, które występują na wszystkich platformach.

  Przykładem tego typu biblioteki było *wxWindows* (obecnie *wxWidgets*).
- ⋆ *Emulacja wybranego API*
- ⋆ *Emulacja GUI*

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API* – biblioteka odwołuje się do systemu okienkowego poprzez natywne API.
  - Zalety: stosunkowo łatwa implementacja biblioteki.
  - Wady:
    — mniejsza efektywność kodu (odwoływanie się do systemu okienkowego poprzez warstwę pośrednią)
    — aby zapewnić przenośność, elementy graficzne i dostępne mechanizmy muszą być ograniczone do tych, które występują na wszystkich platformach.

  Przykładem tego typu biblioteki było *wxWindows* (obecnie *wxWidgets*).
- ⋆ *Emulacja wybranego API*
- ⋆ *Emulacja GUI*

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API*
- ⋆ *Emulacja wybranego API*
- ⋆ *Emulacja GUI*

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API*
- ⋆ *Emulacja wybranego API* – przykładem jest Win32, które emulowane jest pod systemem UNIX. Pakietami narzędziowymi, które to umożliwiają są: MainWin (MS Windows), Wind/U (Bristol Technology).
  - Zalety: względnie łatwa przenośność oprogramowania, które zostało stworzone dla MS Windows.
  - Wady:
    — platformy graficzne MS Windows są istotnie różne. Powoduje to, że aplikacje, które pracują w sposób zadawalający pod MS Windows, w środowisku X Window będą mało efektywne.
    — Win32 ma pewne własności, które nie są zdokumentowane, mimo to są wykorzystywane. To istotnie ogranicza przenośność.
    — nie mogą być wykorzystane w pełni własności środowiska X Window. Aplikacja może jedynie korzystać z tego co jest w Win32.
- ⋆ *Emulacja GUI*

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API*
- ⋆ *Emulacja wybranego API* – przykładem jest Win32, które emulowane jest pod systemem UNIX. Pakietami narzędziowymi, które to umożliwiają są: MainWin (MS Windows), Wind/U (Bristol Technology).
  - Zalety: względnie łatwa przenośność oprogramowania, które zostało stworzone dla MS Windows.
  - Wady:
    — platformy graficzne MS Windows są istotnie różne. Powoduje to, że aplikacje, które pracują w sposób zadawalający pod MS Windows, w środowisku X Window będą mało efektywne.
    — Win32 ma pewne własności, które nie są zdokumentowane, mimo to są wykorzystywane. To istotnie ogranicza przenośność.
    — nie mogą być wykorzystane w pełni własności środowiska X Window. Aplikacja może jedynie korzystać z tego co jest w Win32.
- ⋆ *Emulacja GUI*

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API*
- ⋆ *Emulacja wybranego API* – przykładem jest Win32, które emulowane jest pod systemem UNIX. Pakietami narzędziowymi, które to umożliwiają są: MainWin (MS Windows), Wind/U (Bristol Technology).
  - Zalety: względnie łatwa przenośność oprogramowania, które zostało stworzone dla MS Windows.
  - Wady:
    — platformy graficzne MS Windows są istotnie różne. Powoduje to, że aplikacje, które pracują w sposób zadawalający pod MS Windows, w środowisku X Window będą mało efektywne.
    — Win32 ma pewne własności, które nie są zdokumentowane, mimo to są wykorzystywane. To istotnie ogranicza przenośność.
    — nie mogą być wykorzystane w pełni własności środowiska X Window. Aplikacja może jedynie korzystać z tego co jest w Win32.
- ⋆ *Emulacja GUI*

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API*
- ⋆ *Emulacja wybranego API* – przykładem jest Win32, które emulowane jest pod systemem UNIX. Pakietami narzędziowymi, które to umożliwiają są: MainWin (MS Windows), Wind/U (Bristol Technology).
  - Zalety: względnie łatwa przenośność oprogramowania, które zostało stworzone dla MS Windows.
  - Wady:
    — platformy graficzne MS Windows są istotnie różne. Powoduje to, że aplikacje, które pracują w sposób zadawalający pod MS Windows, w środowisku X Window będą mało efektywne.
    — Win32 ma pewne własności, które nie są zdokumentowane, mimo to są wykorzystywane. To istotnie ogranicza przenośność.
    — nie mogą być wykorzystane w pełni własności środowiska X Window. Aplikacja może jedynie korzystać z tego co jest w Win32.
- ⋆ *Emulacja GUI*

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API*
- ⋆ *Emulacja wybranego API*
- ⋆ *Emulacja GUI*

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API*
- ⋆ *Emulacja wybranego API*
- ⋆ *Emulacja GUI* – nie korzysta się z natywnych pakietów narzędziowych. Elementy graficzne tworzone są poprzez bezpośrednie wykorzystanie możliwie najniższe warstwy danego systemu w oparciu o własny emulowany toolkit.
  - Zalety: ujednolicony wygląd kontrolek, duża efektywność kodu
  - Wady: stworzenie takiej biblioteki jest bardzo pracochłonne, gdyż na każdej platformie trzeba zaimplementować indywidualny kod dla tworzenia poszczególnych elementów graficznych.

  Przykładem tego typu biblioteki jest *Qt*. W tym kierunku podąża obecnie również *wxWidgets*.

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API*
- ⋆ *Emulacja wybranego API*
- ⋆ *Emulacja GUI* – nie korzysta się z natywnych pakietów narzędziowych. Elementy graficzne tworzone są poprzez bezpośrednie wykorzystanie możliwie najniższe warstwy danego systemu w oparciu o własny emulowany toolkit.
  - Zalety: ujednolicony wygląd kontrolek, duża efektywność kodu
  - Wady: stworzenie takiej biblioteki jest bardzo pracochłonne, gdyż na każdej platformie trzeba zaimplementować indywidualny kod dla tworzenia poszczególnych elementów graficznych.

Przykładem tego typu biblioteki jest *Qt*. W tym kierunku podąża obecnie również *wxWidgets*.

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API*
- ⋆ *Emulacja wybranego API*
- ⋆ *Emulacja GUI* – nie korzysta się z natywnych pakietów narzędziowych. Elementy graficzne tworzone są poprzez bezpośrednie wykorzystanie możliwie najniższe warstwy danego systemu w oparciu o własny emulowany toolkit.
  - Zalety: ujednolicony wygląd kontrolek, duża efektywność kodu
  - Wady: stworzenie takiej biblioteki jest bardzo pracochłonne, gdyż na każdej platformie trzeba zaimplementować indywidualny kod dla tworzenia poszczególnych elementów graficznych.

Przykładem tego typu biblioteki jest *Qt*. W tym kierunku podąża obecnie również *wxWidgets*.

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API*
- ⋆ *Emulacja wybranego API*
- ⋆ *Emulacja GUI* – nie korzysta się z natywnych pakietów narzędziowych. Elementy graficzne tworzone są poprzez bezpośrednie wykorzystanie możliwie najniższe warstwy danego systemu w oparciu o własny emulowany toolkit.
  - Zalety: ujednolicony wygląd kontrolek, duża efektywność kodu
  - Wady: stworzenie takiej biblioteki jest bardzo pracochłonne, gdyż na każdej platformie trzeba zaimplementować indywidualny kod dla tworzenia poszczególnych elementów graficznych.

Przykładem tego typu biblioteki jest *Qt*. W tym kierunku podąża obecnie również *wxWidgets*.

Typy konstrukcji wieloplatformowych bibliotek okienkowych

- ⋆ *Nakładka na natywne API*
- ⋆ *Emulacja wybranego API*
- ⋆ *Emulacja GUI* – nie korzysta się z natywnych pakietów narzędziowych. Elementy graficzne tworzone są poprzez bezpośrednie wykorzystanie możliwie najniższe warstwy danego systemu w oparciu o własny emulowany toolkit.
  - Zalety: ujednolicony wygląd kontrolek, duża efektywność kodu
  - Wady: stworzenie takiej biblioteki jest bardzo pracochłonne, gdyż na każdej platformie trzeba zaimplementować indywidualny kod dla tworzenia poszczególnych elementów graficznych.

  Przykładem tego typu biblioteki jest *Qt*. W tym kierunku podąża obecnie również *wxWidgets*.

Funkcje obsługi zdarzeń

- ⋆ Biblioteka dostarcza mechanizmy obsługi zdarzeń oraz mechanizmy tworzenia elementów graficznych aplikacji.
- ⋆ Biblioteka dostarcza funkcje, które realizują domyślną obsługę zdarzeń.
- ⋆ Programista pisze kod, który odpowiedzialny jest za utworzenie elementów graficznych oraz dostarcza funkcje, które modyfikują sposób obsługi zdarzeń. Istnienie tych funkcji trzeba odpowiednio zasygnalizować w danym systemie.

Jak sygnalizowana jest obecność funkcji?

Funkcje obsługi zdarzeń

- ⋆ Biblioteka dostarcza mechanizmy obsługi zdarzeń oraz mechanizmy tworzenia elementów graficznych aplikacji.
- ⋆ Biblioteka dostarcza funkcje, które realizują domyślną obsługę zdarzeń.
- ⋆ Programista pisze kod, który odpowiedzialny jest za utworzenie elementów graficznych oraz dostarcza funkcje, które modyfikują sposób obsługi zdarzeń. Istnienie tych funkcji trzeba odpowiednio zasygnalizować w danym systemie.

Jak sygnalizowana jest obecność funkcji?

Funkcje obsługi zdarzeń

- ⋆ Biblioteka dostarcza mechanizmy obsługi zdarzeń oraz mechanizmy tworzenia elementów graficznych aplikacji.
- ⋆ Biblioteka dostarcza funkcje, które realizują domyślną obsługę zdarzeń.
- ⋆ Programista pisze kod, który odpowiedzialny jest za utworzenie elementów graficznych oraz dostarcza funkcje, które modyfikują sposób obsługi zdarzeń. Istnienie tych funkcji trzeba odpowiednio zasygnalizować w danym systemie.

Jak sygnalizowana jest obecność funkcji?

Funkcje obsługi zdarzeń

- ⋆ Biblioteka dostarcza mechanizmy obsługi zdarzeń oraz mechanizmy tworzenia elementów graficznych aplikacji.
- ⋆ Biblioteka dostarcza funkcje, które realizują domyślną obsługę zdarzeń.
- ⋆ Programista pisze kod, który odpowiedzialny jest za utworzenie elementów graficznych oraz dostarcza funkcje, które modyfikują sposób obsługi zdarzeń. Istnienie tych funkcji trzeba odpowiednio zasygnalizować w danym systemie.

Jak sygnalizowana jest obecność funkcji?

Funkcje obsługi zdarzeń

- *Wywołanie zwrotne* (callback functions) – wxWindows, GTK+
- *Metody wirtualne* – wxWidgets, Qt
- *Sygnały* – Qt

Wywołania zwrotne versus Sygnały

- Mechanizm sygnałów pozwala z danym zdarzeniem skojarzyć kilka funkcji, które dostarczają różnego typu reakcje na to samo zdarzenie np. wyświetlenie napisu, zablokowanie przycisku itd.

  Wywołanie zwrotne zawsze kojarzy jedną reakcję z danym zdarzeniem.

- Mechanizm sygnałów może być wykorzystany również przez samą aplikację. Może ona emitować sygnały, które będą uruchamiać zbiór reakcji.

  W przypadku wywołań zwrotnych biblioteki graficzne nie dostarczają dodatkowych mechanizmów, które pozwalałyby tworzyć w równie naturalny sposób wywołania zwrotne dla kodu nie związanego z samą biblioteką.

Wywołania zwrotne versus Sygnały

- Mechanizm sygnałów pozwala z danym zdarzeniem skojarzyć kilka funkcji, które dostarczają różnego typu reakcje na to samo zdarzenie np. wyświetlenie napisu, zablokowanie przycisku itd.

  Wywołanie zwrotne zawsze kojarzy jedną reakcję z danym zdarzeniem.

- Mechanizm sygnałów może być wykorzystany również przez samą aplikację. Może ona emitować sygnały, które będą uruchamiać zbiór reakcji.

  W przypadku wywołań zwrotnych biblioteki graficzne nie dostarczają dodatkowych mechanizmów, które pozwalałyby tworzyć w równie naturalny sposób wywołania zwrotne dla kodu nie związanego z samą biblioteką.

Wywołania zwrotne versus Sygnały

- Mechanizm sygnałów pozwala z danym zdarzeniem skojarzyć kilka funkcji, które dostarczają różnego typu reakcje na to samo zdarzenie np. wyświetlenie napisu, zablokowanie przycisku itd.

  Wywołanie zwrotne zawsze kojarzy jedną reakcję z danym zdarzeniem.

- Mechanizm sygnałów może być wykorzystany również przez samą aplikację. Może ona emitować sygnały, które będą uruchamiać zbiór reakcji.

  W przypadku wywołań zwrotnych biblioteki graficzne nie dostarczają dodatkowych mechanizmów, które pozwalałyby tworzyć w równie naturalny sposób wywołania zwrotne dla kodu nie związanego z samą biblioteką.

Wywołania zwrotne versus Sygnały

- Mechanizm sygnałów pozwala z danym zdarzeniem skojarzyć kilka funkcji, które dostarczają różnego typu reakcje na to samo zdarzenie np. wyświetlenie napisu, zablokowanie przycisku itd.

  Wywołanie zwrotne zawsze kojarzy jedną reakcję z danym zdarzeniem.
- Mechanizm sygnałów może być wykorzystany również przez samą aplikację. Może ona emitować sygnały, które będą uruchamiać zbiór reakcji.

  W przypadku wywołań zwrotnych biblioteki graficzne nie dostarczają dodatkowych mechanizmów, które pozwalałyby tworzyć w równie naturalny sposób wywołania zwrotne dla kodu nie związanego z samą biblioteką.

Biblioteka Qt

- Komponenty finalne `GUI`
- Komponenty bazowe `GUI`
- Komponenty pomocnicze (ang. Utilities)
- Pomocnicze szablony i kontenery
- Własne elementarne typy danych zapewniające przenośność

Biblioteka Qt

- Komponenty finalne `GUI`
- Komponenty bazowe `GUI`
- Komponenty pomocnicze (ang. Utilities)
- Pomocnicze szablony i kontenery
- Własne elementarne typy danych zapewniające przenośność

Biblioteka Qt

- Komponenty finalne `GUI`
- Komponenty bazowe `GUI`
- Komponenty pomocnicze (ang. Utilities)
- Pomocnicze szablony i kontenery
- Własne elementarne typy danych zapewniające przenośność

Biblioteka Qt

- Komponenty finalne `GUI`
- Komponenty bazowe `GUI`
- Komponenty pomocnicze (ang. Utilities)
- Pomocnicze szablony i kontenery
- Własne elementarne typy danych zapewniające przenośność

Biblioteka Qt

- Komponenty finalne `GUI`
- Komponenty bazowe `GUI`
- Komponenty pomocnicze (ang. Utilities)
- Pomocnicze szablony i kontenery
- Własne elementarne typy danych zapewniające przenośność

Biblioteka Qt

- Komponenty finalne `GUI`
- Komponenty bazowe `GUI`
- Komponenty pomocnicze (ang. Utilities)
- Pomocnicze szablony i kontenery
- Własne elementarne typy danych zapewniające przenośność

Biblioteka Qt

- Komponenty finalne `GUI`
  - Klasy aplikacji,
  - Kontrolki (Widgets),
  - Dialogi,
  - Zarządzanie geometrią obiektów.
- Komponenty bazowe `GUI`
- Komponenty pomocnicze (ang. Utilities)
- Pomocnicze szablony i kontenery
- Własne elementarne typy danych zapewniające przenośność

Biblioteka Qt

- Komponenty finalne GUI
  - Klasy aplikacji,
  - Kontrolki (Widgets),
  - Dialogi,
  - Zarządzanie geometrią obiektów.
- Komponenty bazowe GUI
- Komponenty pomocnicze (ang. Utilities)
- Pomocnicze szablony i kontenery
- Własne elementarne typy danych zapewniające przenośność

Biblioteka Qt

- Komponenty finalne `GUI`
- Komponenty bazowe `GUI`
- Komponenty pomocnicze (ang. Utilities)
- Pomocnicze szablony i kontenery
- Własne elementarne typy danych zapewniające przenośność

Biblioteka Qt

- Komponenty finalne GUI
- Komponenty bazowe GUI
  - Kontrolki abstrakcyjne,
  - Klasy operacji rysowania,
  - Klasy czcionek,
  - Jądro,
  - Inne przydatne klasy.
- Komponenty pomocnicze (ang. Utilities)
- Pomocnicze szablony i kontenery
- Własne elementarne typy danych zapewniające przenośność

Biblioteka Qt

- Komponenty finalne GUI
- Komponenty bazowe GUI
  - Kontrolki abstrakcyjne,
  - Klasy operacji rysowania,
  - Klasy czcionek,
  - Jądro,
  - Inne przydatne klasy.
- Komponenty pomocnicze (ang. Utilities)
- Pomocnicze szablony i kontenery
- Własne elementarne typy danych zapewniające przenośność

Biblioteka Qt

- Komponenty finalne `GUI`
- Komponenty bazowe `GUI`
- Komponenty pomocnicze (ang. Utilities)
- Pomocnicze szablony i kontenery
- Własne elementarne typy danych zapewniające przenośność

Biblioteka Qt

- Komponenty finalne `GUI`
- Komponenty bazowe `GUI`
- Komponenty pomocnicze (ang. Utilities)
  - Ogólne klasy narzędzi,
  - Klasy operacji I/O,
  - Klasy operacji na obrazach,
  - Klasy daty i czasu.
- Pomocnicze szablony i kontenery
- Własne elementarne typy danych zapewniające przenośność

Biblioteka Qt

- Komponenty finalne `GUI`
- Komponenty bazowe `GUI`
- Komponenty pomocnicze (ang. Utilities)
  - Ogólne klasy narzędzi,
  - Klasy operacji I/O,
  - Klasy operacji na obrazach,
  - Klasy daty i czasu.
- Pomocnicze szablony i kontenery
- Własne elementarne typy danych zapewniające przenośność

Biblioteka Qt

- Komponenty finalne `GUI`
- Komponenty bazowe `GUI`
- Komponenty pomocnicze (ang. Utilities)
- Pomocnicze szablony i kontenery
- Własne elementarne typy danych zapewniające przenośność

Biblioteka Qt

- Komponenty finalne `GUI`
- Komponenty bazowe `GUI`
- Komponenty pomocnicze (ang. Utilities)
- Pomocnicze szablony i kontenery
- Własne elementarne typy danych zapewniające przenośność

Najprostsza aplikacja

Najprostsza aplikacja

```
#include <QApplication>
#include <QMessageBox>

int  main( int  argc, char  **argv )
{
 QApplication  app( argc, argv );
 QMessageBox   Message("Wazne pytanie","Byc albo nie byc ?",
                       QMessageBox::Critical,
                       QMessageBox::Yes,
                       QMessageBox::No,0);
 return  Message.exec();
}
```

Komponenty związane z systemem operacyjnym

Biblioteka Qt dostarcza wiele komponentów, które umożliwiają uniezależnić aplikację do bibliotek systemowych. Komponenty te pozwalają na realizację operacji takich jak:

- komunikacja międzyprocesowa z wykorzystaniem mechanizmu gniazd (np. `QSocket`),
- tworzenie wątków (`QThread`),
- nawiązywanie połączenia od strony klienta (`QClient`),
- nawiązywanie połączenia od strony serwera (`QServer`),
- obsługę protokołu HTTP (np. `QHttp`, `QHttpHeader` itp.),
- uruchamianie zewnętrznych programów i realizację komunikacji z nimi (`QProcess`),
- dostęp do kartotek i plików (`QFile`, `QDir`).

Komponenty związane z systemem operacyjnym

Biblioteka Qt dostarcza wiele komponentów, które umożliwiają uniezależnić aplikację do bibliotek systemowych. Komponenty te pozwalają na realizację operacji takich jak:

- komunikacja międzyprocesowa z wykorzystaniem mechanizmu gniazd (np. `QSocket`),
- tworzenie wątków (`QThread`),
- nawiązywanie połączenia od strony klienta (`QClient`),
- nawiązywanie połączenia od strony serwera (`QServer`),
- obsługę protokołu HTTP (np. `QHttp`, `QHttpHeader` itp.),
- uruchamianie zewnętrznych programów i realizację komunikacji z nimi (`QProcess`),
- dostęp do kartotek i plików (`QFile`, `QDir`).

Komponenty związane z systemem operacyjnym

Biblioteka Qt dostarcza wiele komponentów, które umożliwiają uniezależnić aplikację do bibliotek systemowych. Komponenty te pozwalają na realizację operacji takich jak:

- komunikacja międzyprocesowa z wykorzystaniem mechanizmu gniazd (np. `QSocket`),
- tworzenie wątków (`QThread`),
- nawiązywanie połączenia od strony klienta (`QClient`),
- nawiązywanie połączenia od strony serwera (`QServer`),
- obsługę protokołu HTTP (np. `QHttp`, `QHttpHeader` itp.),
- uruchamianie zewnętrznych programów i realizację komunikacji z nimi (`QProcess`),
- dostęp do kartotek i plików (`QFile`, `QDir`).

Komponenty związane z systemem operacyjnym

Biblioteka Qt dostarcza wiele komponentów, które umożliwiają uniezależnić aplikację do bibliotek systemowych. Komponenty te pozwalają na realizację operacji takich jak:

- komunikacja międzyprocesowa z wykorzystaniem mechanizmu gniazd (np. `QSocket`),
- tworzenie wątków (`QThread`),
- nawiązywanie połączenia od strony klienta (`QClient`),
- nawiązywanie połączenia od strony serwera (`QServer`),
- obsługę protokołu HTTP (np. `QHttp`, `QHttpHeader` itp.),
- uruchamianie zewnętrznych programów i realizację komunikacji z nimi (`QProcess`),
- dostęp do kartotek i plików (`QFile`, `QDir`).

Komponenty związane z systemem operacyjnym

Biblioteka Qt dostarcza wiele komponentów, które umożliwiają uniezależnić aplikację do bibliotek systemowych. Komponenty te pozwalają na realizację operacji takich jak:

- komunikacja międzyprocesowa z wykorzystaniem mechanizmu gniazd (np. `QSocket`),
- tworzenie wątków (`QThread`),
- nawiązywanie połączenia od strony klienta (`QClient`),
- nawiązywanie połączenia od strony serwera (`QServer`),
- obsługę protokołu HTTP (np. `QHttp`, `QHttpHeader` itp.),
- uruchamianie zewnętrznych programów i realizację komunikacji z nimi (`QProcess`),
- dostęp do kartotek i plików (`QFile`, `QDir`).

Komponenty związane z systemem operacyjnym

Biblioteka Qt dostarcza wiele komponentów, które umożliwiają uniezależnić aplikację do bibliotek systemowych. Komponenty te pozwalają na realizację operacji takich jak:

- komunikacja międzyprocesowa z wykorzystaniem mechanizmu gniazd (np. `QSocket`),
- tworzenie wątków (`QThread`),
- nawiązywanie połączenia od strony klienta (`QClient`),
- nawiązywanie połączenia od strony serwera (`QServer`),
- obsługę protokołu HTTP (np. `QHttp`, `QHttpHeader` itp.),
- uruchamianie zewnętrznych programów i realizację komunikacji z nimi (`QProcess`),
- dostęp do kartotek i plików (`QFile`, `QDir`).

Komponenty związane z systemem operacyjnym

Biblioteka Qt dostarcza wiele komponentów, które umożliwiają uniezależnić aplikację do bibliotek systemowych. Komponenty te pozwalają na realizację operacji takich jak:

- komunikacja międzyprocesowa z wykorzystaniem mechanizmu gniazd (np. `QSocket`),
- tworzenie wątków (`QThread`),
- nawiązywanie połączenia od strony klienta (`QClient`),
- nawiązywanie połączenia od strony serwera (`QServer`),
- obsługę protokołu HTTP (np. `QHttp`, `QHttpHeader` itp.),
- uruchamianie zewnętrznych programów i realizację komunikacji z nimi (`QProcess`),
- dostęp do kartotek i plików (`QFile`, `QDir`).

Komponenty związane z systemem operacyjnym

Biblioteka Qt dostarcza wiele komponentów, które umożliwiają uniezależnić aplikację do bibliotek systemowych. Komponenty te pozwalają na realizację operacji takich jak:

- komunikacja międzyprocesowa z wykorzystaniem mechanizmu gniazd (np. `QSocket`),
- tworzenie wątków (`QThread`),
- nawiązywanie połączenia od strony klienta (`QClient`),
- nawiązywanie połączenia od strony serwera (`QServer`),
- obsługę protokołu HTTP (np. `QHttp`, `QHttpHeader` itp.),
- uruchamianie zewnętrznych programów i realizację komunikacji z nimi (`QProcess`),
- dostęp do kartotek i plików (`QFile`, `QDir`).

Komponety związane ze strukturami danych

Dostępne są również klasy i szablony, które definiują podstawowe struktury danych oraz ich obsługę. Do struktur tych należą:

- wektory (`QVector`),
- listy (`QList`),
- kolejki priorytetowe (`QQueue`),
- stosy (`QStack`),
- mapy i multimapy (`QMap` i `QMultiMap`).

Komponety związane ze strukturami danych

Dostępne są również klasy i szablony, które definiują podstawowe struktury danych oraz ich obsługę. Do struktur tych należą:

- wektory (`QVector`),
- listy (`QList`),
- kolejki priorytetowe (`QQueue`),
- stosy (`QStack`),
- mapy i multimapy (`QMap` i `QMultiMap`).

Komponety związane ze strukturami danych

Dostępne są również klasy i szablony, które definiują podstawowe struktury danych oraz ich obsługę. Do struktur tych należą:

- wektory (`QVector`),
- listy (`QList`),
- kolejki priorytetowe (`QQueue`),
- stosy (`QStack`),
- mapy i multimapy (`QMap` i `QMultiMap`).

Komponety związane ze strukturami danych

Dostępne są również klasy i szablony, które definiują podstawowe struktury danych oraz ich obsługę. Do struktur tych należą:

- wektory (`QVector`),
- listy (`QList`),
- kolejki priorytetowe (`QQueue`),
- stosy (`QStack`),
- mapy i multimapy (`QMap` i `QMultiMap`).

Komponety związane ze strukturami danych

Dostępne są również klasy i szablony, które definiują podstawowe struktury danych oraz ich obsługę. Do struktur tych należą:

- wektory (`QVector`),
- listy (`QList`),
- kolejki priorytetowe (`QQueue`),
- stosy (`QStack`),
- mapy i multimapy (`QMap` i `QMultiMap`).

Komponety związane ze strukturami danych

Dostępne są również klasy i szablony, które definiują podstawowe struktury danych oraz ich obsługę. Do struktur tych należą:

- wektory (`QVector`),
- listy (`QList`),
- kolejki priorytetowe (`QQueue`),
- stosy (`QStack`),
- mapy i multimapy (`QMap` i `QMultiMap`).

Kiedy używać pomocniczych komponentów Qt

Kiedy wykorzystywać komponenty związane mechanizmami, które dostępne są w bibliotekach systemowych?

- Na pewno wtedy, gdy chcemy zapewnić przenośność między różnymi platformami (np. `Unix`, `MS Windows`, `MacOS X`)

Kiedy nie jest zalecane wykorzystywanie tego typu komponentów?

- Jeżeli aplikacja nie będzie używana na innych platformach, to zdecydowanie lepiej jest wykorzystać natywne biblioteki zapewniające właściwe mechanizmy, takie jak tworzenie gniazd i realizację komunikacji międzyprocesowej.
  Biblioteki te są dobrze ustabilizowane i występują wszędzie w obrębie danej platformy. Natomiast biblioteka Qt wciąż dynamicznie rozwija się. Zmiany w bibliotece mogą utrudnić utrzymanie aplikacji.

Kiedy używać pomocniczych komponentów Qt

Kiedy wykorzystywać komponenty związane mechanizmami, które dostępne są w bibliotekach systemowych?

- Na pewno wtedy, gdy chcemy zapewnić przenośność między różnymi platformami (np. `Unix`, `MS Windows`, `MacOS X`)

Kiedy nie jest zalecane wykorzystywanie tego typu komponentów?

- Jeżeli aplikacja nie będzie używana na innych platformach, to zdecydowanie lepiej jest wykorzystać natywne biblioteki zapewniające właściwe mechanizmy, takie jak tworzenie gniazd i realizację komunikacji międzyprocesowej.
  Biblioteki te są dobrze ustabilizowane i występują wszędzie w obrębie danej platformy. Natomiast biblioteka Qt wciąż dynamicznie rozwija się. Zmiany w bibliotece mogą utrudnić utrzymanie aplikacji.

Kiedy używać pomocniczych komponentów Qt

Kiedy wykorzystywać komponenty związane mechanizmami, które dostępne są w bibliotekach systemowych?

- Na pewno wtedy, gdy chcemy zapewnić przenośność między różnymi platformami (np. `Unix`, `MS Windows`, `MacOS X`)

Kiedy nie jest zalecane wykorzystywanie tego typu komponentów?

- Jeżeli aplikacja nie będzie używana na innych platformach, to zdecydowanie lepiej jest wykorzystać natywne biblioteki zapewniające właściwe mechanizmy, takie jak tworzenie gniazd i realizację komunikacji międzyprocesowej.
  Biblioteki te są dobrze ustabilizowane i występują wszędzie w obrębie danej platformy. Natomiast biblioteka Qt wciąż dynamicznie rozwija się. Zmiany w bibliotece mogą utrudnić utrzymanie aplikacji.

Kiedy używać pomocniczych komponentów Qt

Kiedy wykorzystywać komponenty związane mechanizmami, które dostępne są w bibliotekach systemowych?

- Na pewno wtedy, gdy chcemy zapewnić przenośność między różnymi platformami (np. `Unix`, `MS Windows`, `MacOS X`)

Kiedy nie jest zalecane wykorzystywanie tego typu komponentów?

- Jeżeli aplikacja nie będzie używana na innych platformach, to zdecydowanie lepiej jest wykorzystać natywne biblioteki zapewniające właściwe mechanizmy, takie jak tworzenie gniazd i realizację komunikacji międzyprocesowej.
  Biblioteki te są dobrze ustabilizowane i występują wszędzie w obrębie danej platformy. Natomiast biblioteka Qt wciąż dynamicznie rozwija się. Zmiany w bibliotece mogą utrudnić utrzymanie aplikacji.

Kiedy używać pomocniczych komponentów Qt

Kiedy wykorzystywać komponenty oferujące definicje struktur danych, które dostępne są również w bibliotece STL język C++?

- W bibliotece Qt klasy i szablony klasy `QList`, `QVector`, `QMap` itd. konsekwentnie wykorzystywane są do przekazywania danych do poszczególnych komponentów graficznych. Jeżeli wykonywane operacje w programie są ściśle związane z tymi komponentami, to wówczas warto skorzystać ze wspomnianych struktur danych.

Kiedy używać pomocniczych komponentów Qt

Kiedy wykorzystywać komponenty oferujące definicje struktur danych, które dostępne są również w bibliotece STL język C++?

- W bibliotece Qt klasy i szablony klasy `QList`, `QVector`, `QMap` itd. konsekwentnie wykorzystywane są do przekazywania danych do poszczególnych komponentów graficznych. Jeżeli wykonywane operacje w programie są ściśle związane z tymi komponentami, to wówczas warto skorzystać ze wspomnianych struktur danych.

Kiedy używać pomocniczych komponentów Qt

Kiedy nie jest zalecane wykorzystywanie tego typu klas i szablonów?

- Jeżeli aplikacja nie realizuje operacji bezpośrednio związanych z komponentami graficznymi, to wówczas bardziej zalecanym jest korzystanie z biblioteki STL język C++.
  Biblioteki ta jest dobrze ustabilizowana i dostępna jest z każdym kompilatorem C++ spełniającym normę ANSI/ISO C++.
  Ze względu na dynamiczny rozwój biblioteki Qt, zmiany w niej mogą utrudnić utrzymanie aplikacji.
  W przypadku aplikacji, które nie korzystają w ogóle z interfejsu graficznego, zastosowanie biblioteki Qt może ograniczyć jej przenośność (nie każdy komputer w obrębie danej platformy systemowej musi być wyposażony w bibliotekę Qt).

Kiedy używać pomocniczych komponentów Qt

Kiedy nie jest zalecane wykorzystywanie tego typu klas i szablonów?

- Jeżeli aplikacja nie realizuje operacji bezpośrednio związanych z komponentami graficznymi, to wówczas bardziej zalecanym jest korzystanie z biblioteki STL język C++.
  Biblioteki ta jest dobrze ustabilizowana i dostępna jest z każdym kompilatorem C++ spełniającym normę ANSI/ISO C++.
  Ze względu na dynamiczny rozwój biblioteki Qt, zmiany w niej mogą utrudnić utrzymanie aplikacji.
  W przypadku aplikacji, które nie korzystają w ogóle z interfejsu graficznego, zastosowanie biblioteki Qt może ograniczyć jej przenośność (nie każdy komputer w obrębie danej platformy systemowej musi być wyposażony w bibliotekę Qt).

Kiedy używać pomocniczych komponentów Qt

Kiedy nie jest zalecane wykorzystywanie tego typu klas i szablonów?

- Jeżeli aplikacja nie realizuje operacji bezpośrednio związanych z komponentami graficznymi, to wówczas bardziej zalecanym jest korzystanie z biblioteki STL język C++.
  Biblioteki ta jest dobrze ustabilizowana i dostępna jest z każdym kompilatorem C++ spełniającym normę ANSI/ISO C++.
  Ze względu na dynamiczny rozwój biblioteki Qt, zmiany w niej mogą utrudnić utrzymanie aplikacji.
  W przypadku aplikacji, które nie korzystają w ogóle z interfejsu graficznego, zastosowanie biblioteki Qt może ograniczyć jej przenośność (nie każdy komputer w obrębie danej platformy systemowej musi być wyposażony w bibliotekę Qt).

Ogólne zalecenie co do konstrukcji aplikacji

- W aplikacji powinno zostać wydzielone *jądro* (rozumiane jako zbiór modułów), które nie zależy od zastosowanej biblioteki graficznej.
- Wykorzystanie biblioteki należy możliwie ograniczyć do modułów, które istotnie wykorzystują elementy graficzne oferowane przez bibliotekę, a w których konstrukcji i obsłudze biblioteka ta jest bardzo pomocna.
- Odzielenie GUI od zasadniczej części aplikacji nie tylko ułatwia tworzenie przenośnych aplikacji. Pozwala na jasne oddzielenie tego co zależy od interackji z użytkownikiem, od tego co związane jest z implementowaną metodą rozwiązywania danego problemu. Ułatwia to również uruchaminie całej aplikacji i jej późniejsze utrzymanie.

Ogólne zalecenie co do konstrukcji aplikacji

- W aplikacji powinno zostać wydzielone *jądro* (rozumiane jako zbiór modułów), które nie zależy od zastosowanej biblioteki graficznej.
- Wykorzystanie biblioteki należy możliwie ograniczyć do modułów, które istotnie wykorzystują elementy graficzne oferowane przez bibliotekę, a w których konstrukcji i obsłudze biblioteka ta jest bardzo pomocna.
- Odzielenie GUI od zasadniczej części aplikacji nie tylko ułatwia tworzenie przenośnych aplikacji. Pozwala na jasne oddzielenie tego co zależy od interackji z użytkownikiem, od tego co związane jest z implementowaną metodą rozwiązywania danego problemu. Ułatwia to również uruchaminie całej aplikacji i jej późniejsze utrzymanie.

Ogólne zalecenie co do konstrukcji aplikacji

- W aplikacji powinno zostać wydzielone *jądro* (rozumiane jako zbiór modułów), które nie zależy od zastosowanej biblioteki graficznej.
- Wykorzystanie biblioteki należy możliwie ograniczyć do modułów, które istotnie wykorzystują elementy graficzne oferowane przez bibliotekę, a w których konstrukcji i obsłudze biblioteka ta jest bardzo pomocna.
- Odzielenie GUI od zasadniczej części aplikacji nie tylko ułatwia tworzenie przenośnych aplikacji. Pozwala na jasne oddzielenie tego co zależy od interackji z użytkownikiem, od tego co związane jest z implementowaną metodą rozwiązywania danego problemu. Ułatwia to również uruchaminie całej aplikacji i jej późniejsze utrzymanie.

Koniec prezentacji