



NAJLEPSZE WZORCE DLA C++ I QT!

C++ i Qt

Wprowadzenie do
wzorców projektowych

Wydanie II



Alan Ezust · Paul Ezust

Tytuł oryginału: An Introduction to Design Patterns in C++ with Qt (2nd Edition)

Tłumaczenie: Justyna Walkowska

ISBN: 978-83-246-8246-1

Authorized translation from the English language edition, entitled: Introduction to Design Patterns in C++ with Qt, Second Edition; ISBN 0132826453; by Alan Ezust and Paul Ezust; published by Pearson Education, Inc; publishing as Prentice Hall.

Copyright © 2012 Alan and Paul Ezust.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

Polish language edition published by HELION S.A., Copyright © 2014.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/cppqtw>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Słowo wstępne	13
Przedmowa	17
Przedmowa do wydania drugiego	19
Podziękowania	21
O autorach	25
Część I. Wzorce projektowe i Qt	27
Rozdział 1. Wprowadzenie do C++	29
1.1. Przegląd języka C++	29
1.2. Krótka historia C++	30
1.3. Pierwszy przykład	30
1.4. Standardowe wejście i wyjście	33
1.5. Wprowadzenie do funkcji	36
1.6. qmake, pliki projektu i Makefile	41
1.7. Pomocne zasoby w internecie	46
1.8. Łańcuchy znaków	47
1.9. Strumienie	48
1.10. Strumienie do plików	50
1.11. Okienka dialogowe Qt	54
1.12. Identyfikatory, typy i literały	57
1.13. Typy proste	59
1.14. Słowo kluczowe const	69
1.15. Wskaźniki i dostęp do pamięci	70
1.16. Referencje	75
1.17. Różnica pomiędzy const* a *const	76
1.18. Powtórka	79

Rozdział 2. Pierwsza klasa	81
2.1. Na początku było struct	81
2.2. Definicje klas	82
2.3. Modyfikatory dostępu do składowych	85
2.4. Enkapsulacja	87
2.5. Wprowadzenie do UML	87
2.6. Przyjaciele klasy	89
2.7. Konstruktory	90
2.8. Destruktory	92
2.9. Słowo kluczowe static	93
2.10. Deklaracje i definicje klas	97
2.11. Konstruktory kopiujące i operatory przypisania	99
2.12. Konwersje	101
2.13. Funkcje składowe const	104
2.14. Podobiekty	105
2.15. Ćwiczenia: klasy	107
2.16. Pytania sprawdzające znajomość rozdziału	114
Rozdział 3. Wprowadzenie do Qt	119
3.1. Styl i konwencje nazewnicze	119
3.2. Moduł Qt Core	121
3.3. Qt Creator: zintegrowane środowisko programistyczne dla Qt	124
3.4. Ćwiczenia: wprowadzenie do Qt	125
3.5. Powtórka	126
Rozdział 4. Listy	127
4.1. Wprowadzenie do kontenerów	127
4.2. Iteratory	128
4.3. Relacje	133
4.4. Ćwiczenia: relacje	135
4.5. Powtórka	137
Rozdział 5. Funkcje	139
5.1. Przeladowywanie funkcji	139
5.2. Argumenty opcjonalne	142
5.3. Przeladowywanie operatorów	144
5.4. Przekazywanie parametrów przez wartość	148
5.5. Przekazywanie parametrów przez referencję	149
5.6. Referencje do const	152

5.7. Wartości zwracane przez funkcje	154
5.8. Zwracanie referencji przez funkcje	154
5.9. Przeladowywanie w oparciu o const	155
5.10. Funkcje inline	157
5.11. Funkcje z listą argumentów o zmiennej długości	161
5.12. Ćwiczenia: szyfrowanie	162
5.13. Powtórka	164
Rozdział 6. Dziedziczenie i polimorfizm	165
6.1. Proste dziedziczenie	165
6.2. Dziedziczenie i polimorfizm	172
6.3. Dziedziczenie z abstrakcyjnej klasy bazowej	177
6.4. Projektowanie dziedziczenia	182
6.5. Przeladowywanie, ukrywanie i przesłanianie	184
6.6. Konstruktory, destruktory i kopiujące operatory przypisania	186
6.7. Przetwarzanie argumentów z wiersza poleceń	190
6.8. Kontenery	195
6.9. Kontenery zarządzające, kompozycja i agregacja	197
6.10. Kontenery wskaźników	200
6.11. Powtórka	215
Rozdział 7. Biblioteki i wzorce projektowe	221
7.1. Budowanie i używanie bibliotek	222
7.2. Ćwiczenie: instalowanie bibliotek	229
7.3. Frameworki i komponenty	231
7.4. Wzorce projektowe	233
7.5. Powtórka	240
Rozdział 8. Klasy QObject i QApplication, sygnały i sloty	243
8.1. Wartości i obiekty	246
8.2. Rodzice i dzieci: wzorzec Kompozyt	247
8.3. Pętla zdarzeń w QApplication	253
8.4. QObject i moc w pigułce	255
8.5. Sygnały i sloty	255
8.6. Cykl życia QObject	257
8.7. QTestLib	258
8.8. Ćwiczenia: QObject, QApplication, sygnały i sloty	261
8.9. Powtórka	261

Rozdział 9. Widżety i projektowanie interfejsów	263
9.1. Kategorie widżetów	264
9.2. Designer: wprowadzenie	266
9.3. Okna dialogowe	269
9.4. Układ formularza	271
9.5. Ikonki, obrazki i zasoby	273
9.6. Układ widżetów	276
9.7. Integracja okienek z kodem	283
9.8. Ćwiczenia: formularze wejściowe	288
9.9. O pętli zdarzeń raz jeszcze	289
9.10. Zdarzenia rysowania	297
9.11. Powtórka	299
Rozdział 10. Główne okna i akcje	301
10.1. QAction, QMenu, QMenuBar	302
10.2. Obszary i dokowanie	309
10.3. QSettings: konfiguracja aplikacji	311
10.4. Schowek i transfer danych	314
10.5. Wzorzec Polecenie	315
10.6. Umiędzynarodawianie i funkcja tr()	322
10.7. Ćwiczenia: Główne okna i akcje	323
10.8. Powtórka	323
Rozdział 11. Uogólnienia i kontenery	325
11.1. Typy uogólnione i szablony	325
11.2. Typy uogólnione, algorytmy i operatory	331
11.3. Przykład sortowania mapy	333
11.4. Funktory i wskaźniki na funkcje	336
11.5. Wzorzec Pylek: klasy współdzielone przez domniemanie	338
11.6. Ćwiczenia: typy uogólnione	342
11.7. Powtórka	343
Rozdział 12. Metaobiekty, metawłaściwości i mechanizm refleksji	345
12.1. Wzorzec Metaobiekt i klasa QMetaObject	345
12.2. Identyfikacja typu i qobject_cast	347
12.3. Macro Q_PROPERTY: opis właściwości QObject	349
12.4. Klasa QVariant: dostęp do właściwości	352
12.5. Dynamiczne właściwości	355
12.6. Deklarowanie i rejestrowanie metatypów	358

12.7. Funkcja invokeMethod()	360
12.8. Ćwiczenia: refleksja	361
12.9. Powtórka	361
Rozdział 13. Modele i widoki	363
13.1. Model-View-Controller (MVC)	364
13.2. Modele i widoki Qt	365
13.3. Modele tabel	375
13.4. Modele drzewiaste	384
13.5. Inteligentne wskaźniki	387
13.6. Ćwiczenia: modele i widoki	390
13.7. Powtórka	391
Rozdział 14. Walidacja i wyrażenia regularne	393
14.1. Maski wprowadzania danych	393
14.2. Walidatory	396
14.3. Wyrażenia regularne	398
14.4. Walidacja wyrażeń regularnych	406
14.5. Podklasy QValidator	407
14.6. Ćwiczenia: walidacja i wyrażenia regularne	410
14.7. Powtórka	411
Rozdział 15. Parsowanie XML	413
15.1. Parsery XML w Qt	415
15.2. Parsowanie SAX	417
15.3. XML, struktury drzewiaste i DOM	421
15.4. Strumienie XML	429
15.5. Powtórka	431
Rozdział 16. Więcej wzorców projektowych	433
16.1. Wzorce konstrukcyjne	433
16.2. Wzorzec Pamiętka	442
16.3. Wzorzec Fasada	447
16.4. Powtórka	453
Rozdział 17. Współbieżność	455
17.1. QProcess i zarządzanie procesami	455
17.2. QThread i QtConcurrent	468
17.3. Ćwiczenia: QThread i QtConcurrent	480
17.4. Powtórka	481

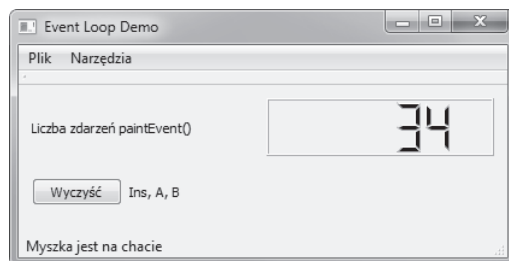
Rozdział 18. Programowanie baz danych	483
18.1. Dostęp do bazy danych z Qt: QSqlDatabase	485
18.2. Zapytania i zestawy wyników	488
18.3. Modele baz danych	490
18.4. Powtórka	491
Część II. Podstawy języka C++	493
Rozdział 19. Typy i wyrażenia	495
19.1. Operatory	495
19.2. Instrukcje i struktury kontroli	500
19.3. Ewaluacja wyrażen logicznych	506
19.4. Typy wyliczeniowe	507
19.5. Typy całkowite ze znakiem i bez znaku	509
19.6. Standardowa konwersja wyrażen	511
19.7. Konwersje jawne	514
19.8. Rzutuj bezpiecznie, używając rzutowania ANSI C++	515
19.9. Przeciążanie operatorów specjalnych	520
19.10. Identyfikacja typów w czasie wykonania programu	525
19.11. Operatory wyboru składowych	527
19.12. Ćwiczenia: typy i wyrażenia	529
19.13. Powtórka	530
Rozdział 20. Zakres identyfikatorów i klasa pamięci	531
20.1. Deklaracje i definicje	531
20.2. Zakres identyfikatorów	532
20.3. Klasa pamięci	540
20.4. Przestrzenie nazw	544
20.5. Powtórka	548
Rozdział 21. Dostęp do pamięci	549
21.1. Patologia wskaźników	550
21.2. Patologia wskaźników na sterce	552
21.3. Dostęp do pamięci: podsumowanie	554
21.4. Wprowadzenie do tablic	555
21.5. Arytmetyka wskaźników	556
21.6. Tablice, funkcje i wartości zwracane	557
21.7. Różne rodzaje tablic	559
21.8. Dozwolone operacje na wskaźnikach	559

21.9. Tablice i pamięć: ważne informacje	561
21.10. Ćwiczenia: dostęp do pamięci	562
21.11. Powtórka	563
Rozdział 22. Szczegółowo o dziedziczeniu	565
22.1. Wskaźniki wirtualne i tablice wirtualne	565
22.2. Polimorfizm i destruktory wirtualne	568
22.3. Wielokrotne dziedziczenie	571
22.4. Dziedziczenie publiczne, chronione i prywatne	577
22.5. Powtórka	579
Część III. Zadania programistyczne	581
Rozdział 23. Ćwiczenia: szafa grająca	583
23.1. Lista odtwarzania	584
23.2. Listy odtwarzania	585
23.3. Wybór źródła	586
23.4. Listy odtwarzania w bazie danych	587
23.5. Rozszerzenie przykładu Star Delegates	587
23.6. Sortowanie, filtrowanie i edycja list odtwarzania	588
Dodatek A. Słowa kluczowe języka C++	589
Dodatek B. Standardowe nagłówki	591
Dodatek C. Narzędzia pracy programisty	593
Dodatek D. Przewodnik Alana: Debian w pigułce dla programistów	615
Dodatek E. Konfiguracja C++/Qt	621
Bibliografia	629
Skorowidz	631

Główne okna i akcje

Większość aplikacji `QApplication` opiera się na pojedynczym oknie `QMainWindow`. Jak pokazuje rysunek 10.1, `QMainWindow` zawiera elementy niezbędne w większości aplikacji desktopowych:

- centralny widżet,
- pasek menu,
- paski narzędzi,
- pasek stanu,
- obszary łączenia.



RYСУNEK 10.1. Główne okno

W większości aplikacji `QMainWindow` jest rodzicem (lub dziadkiem) obiektów `QAction`, `QWidget` i `QObject` na sterście. Często stosowaną praktyką jest rozszerzanie tej klasy, jak na przykładzie 10.1.

PRZYKŁAD 10.1. `src/widgets/mainwindow/mymainwindow.h`

```
[...]  
class MyMainWindow : public QMainWindow {  
    Q_OBJECT  
public:
```

```

explicit QMainWindow(QWidget* parent=0);
void closeEvent(QCloseEvent* event);           ///1

protected slots:
    virtual void newFile();
    virtual void open();
    virtual bool save();
[...]
```

///**1** Metoda nadpisana, by przechwycić zdarzenie zamknięcia okna przez użytkownika.

10.1. QAction, QMenu, QMenuBar

Klasa `QAction`, dziedzicząca z `QObject`, jest klasą bazową dla akcji wybranych przez użytkownika. Ma ona bardzo bogaty interfejs, o czym wkrótce się przekonasz. Interfejs `QWidget` pozwala każdemu widżetowi na przechowywanie listy `QList<QAction*>`.

Każdemu widżetowi może towarzyszyć zestaw akcji. Niektóre widżety wyświetlają listę akcji w menu kontekstowym, inne na pasku menu. Jeśli chcesz zapoznać się z kompletem informacji na temat tworzenia menu kontekstowych dla widżetów, zajrzyj do dokumentacji `setContextMenuPolicy()`.

Obiekty `QMenu` to również widżety `QWidget`, oferujące szczególny widok na kolekcję akcji `QAction`. Obiekt `QMenuBar` to zbiór menu, najczęściej spotykany wewnątrz `QMainWindow`.

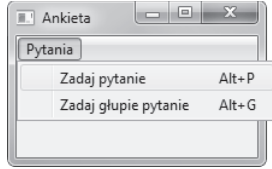
Jeśli rodzicem `QMenu` jest `QMenuBar`, `QMenu` jest widoczne jako menu rozwijane o znajomym interfejsie. Jeśli jego rodzicem nie jest `QMenuBar`, może ono wyskakiwać jak okno dialogowe — w takim przypadku jest nazywane **menu kontekstowym**¹. Rodzicem `QMenu` może również być inne `QMenu`. Wówczas mamy do czynienia z **podmenu**.

W celu ułatwienia użytkownikowi podjęcia wyboru każda akcja `QAction` jest wyposażona w następujące elementy:

- tekst lub ikonka pojawiające się w menu lub na przycisku;
- akcelerator, czyli klawisz skrót;
- odpowiedź „Co to jest?”;
- możliwość przedstawiania stanu akcji pomiędzy następującymi parami właściwości: widoczna/niewidoczna, dostępna/niedostępna, zaznaczona/niezaznaczona;
- sygnały `changed()` (zmiana), `hovered()` (najechnanie myszą), `toggled()` (zmiana właściwości), `triggered()` (uruchomienie).

Okno `QMainWindow` na rysunku 10.2 zawiera jeden pasek menu zawierający jedno menu z dwiema akcjami do wyboru.

¹ Menu kontekstowe najczęściej aktywuje się poprzez kliknięcie prawym przyciskiem myszy lub wciśnięcie przycisku menu. Nazywamy je *kontekstowym*, ponieważ jego zawartość zawsze jest zależna od kontekstu (który `QWidget` jest aktualnie zaznaczony lub aktywny).



RYSUNEK 10.2. QMenu

Przykład 10.2 przedstawia kod użyty do stworzenia paska narzędzi z rysunku 10.2. Funkcja `QMainWindow::menuBar()` zwraca wskaźnik na `QMenuBar` będący dzieckiem `QMainWindow`. Funkcja `menuBar()` tworzy i zwraca wskaźnik na pusty `QMenuBar`, jeśli pasek nie został jeszcze utworzony.

PRZYKŁAD 10.2. `src/widgets/dialogs/messagebox/dialogs.cpp`

[...]

```

/* Umieść menu na pasku */
QMenu* menu = new QMenu(tr("&Pytania"), this);

QMainWindow::menuBar()->addMenu(menu);

/* Kilka opcji w menu */
menu->addAction(tr("&Zadaj pytanie"),
               this, SLOT(askQuestion()), tr("Alt+Z"));
menu->addAction(tr("Zadaj &głupie pytanie"),
               this, SLOT(askDumbQuestion()), tr("Alt+G"));
}

```

Każde wywołanie `QMenu::addAction(text, target, slot, shortcut)` tworzy nienazwany obiekt `QAction` i dodaje go do `QMenu`. Następnie wywołuje ukrytą funkcję z klasy bazowej, `QWidget::addAction(QAction*)`. Dodaje ona nowo powstałą akcję do listy akcji, które zostaną użyte w menu kontekstowym. Akcja zostanie dodana do listy `QList<QAction*>` przechowywanej przez menu.

10.1.1. QAction, QToolBar, QActionGroup

Ponieważ aplikacje pozwalają użytkownikom na wydawanie tych samych komend na różne sposoby (np. poprzez menu, przyciski, skróty klawiszowe), enkapsulacja takiej komendy do postaci *akcji* jest dobrym sposobem zapewnienia spójnego zachowania w całej aplikacji. Akcje `QAction` mogą emitować sygnały i podłączać je do slotów, jeśli jest to potrzebne.

W aplikacjach Qt GUI akcje są z reguły „odpalane” na jeden z następujących sposobów:

- użytkownik klika opcję w menu,
- użytkownik wprowadza skrót klawiszowy,
- użytkownik klika przycisk na pasku narzędzi.

Istnieje kilka przeładowanych postaci funkcji `QMenu::addAction()`. W przykładzie 10.3 korzystamy z wersji odziedziczonej z `QWidget`, `addAction(QAction*)`.

Przykład 10.3 pokazuje, jak dodawać akcje do menu, grup akcji i pasek narzędzi. Na początku tworzymy klasę dziedziczącą z `QMainWindow` i dodajemy do niej kilka składowych `QAction`, a także grupę `QActionGroup` i pasek narzędzi `QToolBar`.

PRZYKŁAD 10.3. src/widgets/menus/study.h

```
[...]
class Study : public QMainWindow {
    Q_OBJECT
public:
    explicit Study(QWidget* parent=0);
public slots:
    void actionEvent(QAction* act);
private:
    QActionGroup* actionGroup;           ///<1
    QToolBar* toolbar;                   ///<2

    QAction* useTheForce;
    QAction* useTheDarkSide;
    QAction* studyWithObiWan;
    QAction* studyWithYoda;
    QAction* studyWithEmperor;
    QAction* fightYoda;
    QAction* fightDarthVader;
    QAction* fightObiWan;
    QAction* fightEmperor;
protected:
    QAction* addChoice(QString name, QString text);
};
[...]
```

`///<1` Do przechwytywania sygnałów.

`///<2` Do wyświetlania akcji jako przycisków.

Konstruktor naszej klasy tworzy wymagane menu i instaluje je wewnątrz paska `QMenuBar` będącego integralną częścią klasy bazowej. Odpowiedni kod został pokazany jako przykład 10.4.

PRZYKŁAD 10.4. src/widgets/menus/study.cpp

```
[...]
Study::Study(QWidget* parent) : QMainWindow(parent) {
    actionGroup = new QActionGroup(this);
    actionGroup->setExclusive(false);
    statusBar();

    QWidget::setWindowTitle( "zostać jedi chcesz?" );           ///<1

    QMenu* useMenu = new QMenu("&Użyj", this);
    QMenu* studyMenu = new QMenu("&Studiuj", this);
```

```

QMenu* fightMenu = new QMenu("&Walcz", this);

useTheForce = addChoice("useTheForce", "Użyj &Mocy");
useTheForce->setStatusTip("To początek podróży...");
useTheForce->setEnabled(true);
useMenu->addAction(useTheForce);                                     ///2
[...]

studyWithObiWan = addChoice("studyWithObiWan", "&Ucz się u Obi-Wana ");
studyMenu->addAction(studyWithObiWan);
studyWithObiWan->setStatusTip("Z pewnością otworzy przed Tobą "
                              "wiele drzwi...");

fightObiWan = addChoice("fightObiWan", "&Walcz z Obi-Wanem");
fightMenu->addAction(fightObiWan);
fightObiWan->setStatusTip("W ten sposób z pewnością "
                          "nauczysz się od niego wielu sztuczek!");
[...]

QMainWindow::menuBar()->addMenu(useMenu);
QMainWindow::menuBar()->addMenu(studyMenu);
QMainWindow::menuBar()->addMenu(fightMenu);

toolbar = new QToolBar("Pasek wyboru", this);                       ///3
toolbar->addAction(actionGroup->actions());

QMainWindow::addToolBar(Qt::LeftToolBarArea, toolbar);

QObject::connect(actionGroup, SIGNAL(triggered(QAction*)),
                 this, SLOT(actionEvent(QAction*)));                ///4

QWidget::move(300, 300);
QWidget::resize(300, 300);
}

```

///1 Niektóre z użytych tu prefiksów *NazwaKlasy::* nie są wymagane, ponieważ funkcje można wywołać na *this*. Nazw klas można użyć, by wywołać wersję funkcji zdefiniowaną w konkretnej nadklasie lub by pokazać czytelnikowi kodu, którą wersję wywołujemy.

///2 Akcja jest już w *QActionGroup*, ale dodajemy ją także do *QMenu*.

///3 W ten sposób uzyskujemy widoczne przyciski w dokowanym (przyłączanym) widżecie dla każdej z *QAction*.

///4 Zamiast podłączać sygnał każdej z akcji, łączymy się jedynie z zawierającą je wszystkie grupą *ActionGroup*.

Istnieje możliwość podłączenia poszczególnych sygnałów *triggered()* akcji do poszczególnych slotów. W przykładzie 10.5 grupujemy powiązane ze sobą akcje wewnątrz *QActionGroup*. Jeśli „odpalona” zostanie jedna z akcji z grupy, *QAction* wyemituje pojedynczy sygnał *triggered(QAction*)*, który pozwala na obsługę wszystkich akcji w ujednolicony sposób. Sygnał przekazuje wskaźnik na konkretną odpaloną akcję, zatem można wybrać odpowiednią dla niej reakcję.

PRZYKŁAD 10.5. src/widgets/menus/study.cpp

[...]

```
// Funkcja wytwórcza (fabryka) produkująca akcje QAction inicjalizowane w ujednolicony sposób
QAction* Study::addChoice(QString name, QString text) {
    QAction* retval = new QAction(text, this);
    retval->setObjectName(name);
    retval->setEnabled(false);
    retval->setCheckable(true);
    actionGroup->addAction(retval);           ///1
    return retval;
}
```

///**1** Dodaj każdą akcję do QActionGroup, dzięki czemu potrzebny będzie tylko jeden sygnał podłączony do jednego slotu.

Po stworzeniu akcji QAction jest ona dodawana (za pomocą addAction()) do trzech innych obiektów:

1. QActionGroup — w celu obsługi sygnałów.
2. QMenu — jednego z trzech menu rozwijanych na pasku QMenuBar.
3. QToolBar — gdzie zostanie udostępniona jako przycisk.

Dla urozmaicenia przykładu wprowadziliśmy pewne zależności logiczne pomiędzy opcjami w menu, odpowiadające fabułom kilku znanych filmów. Są one widoczne w funkcji actionEvent() z przykładu 10.6.

PRZYKŁAD 10.6. src/widgets/menus/study.cpp

[...]

```
void Study::actionEvent(QAction* act) {
    QString name = act->objectName();
    QString msg = QString();

    if (act == useTheForce) { // użyj Mocy
        studyWithObiWan->setEnabled(true); // ucz się u Obi-Wana
        fightObiWan->setEnabled(true); // walcz z Obi-Wanem
        useTheDarkSide->setEnabled(true); // użyj ciemnej strony Mocy
    }
    if (act == useTheDarkSide) { // użyj ciemnej strony Mocy
        studyWithYoda->setEnabled(false); // ucz się u Yody
        fightYoda->setEnabled(true); // walcz z Yodą
        studyWithEmperor->setEnabled(true); // ucz się u Imperatora
        fightEmperor->setEnabled(true); // walcz z Imperatorem
        fightDarthVader->setEnabled(true); // walcz z Darthem Vaderem
    }
    if (act == studyWithObiWan) { // ucz się u Obi-Wana
        fightObiWan->setEnabled(true); // walcz z Obi-Wanem
        fightDarthVader->setEnabled(true); // walcz z Darthem Vaderem
        studyWithYoda->setEnabled(true); // ucz się u Yody
    }
}
```

[...]


```

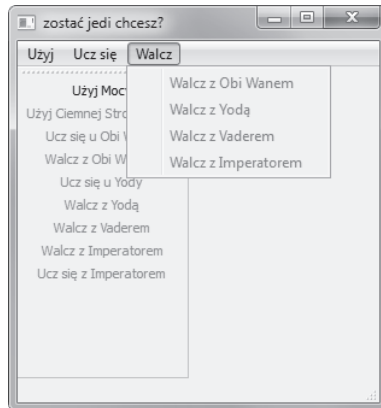
if (act == fightObiWan ) {
    if (studyWithEmperor->isChecked()) {
        msg = "Wygrałeś!";
    }
    else {
        msg = "Przegrałeś.";
        act->setChecked(false);
        studyWithYoda->setEnabled(false);
    }
}
[...]

if (msg != QString()) {
    QMessageBox::information(this, "Wynik", msg, "ok");
}
}

```

Ponieważ wszystkie akcje należą do QActionGroup, sygnał triggered(QAction*) powoduje wywołanie actionEvent().

Wszystkie opcje w menu poza jedną są początkowo nieaktywne. W miarę jak użytkownik wybiera spośród dostępnych opcji, inne stają się dostępne lub nie, co pokazuje rysunek 10.3. Zwróć uwagę na spójność pomiędzy przyciskami i opcjami w menu. Kliknięcie aktywnego przycisku powoduje zaznaczenie odpowiadającego mu elementu w menu. Akcja QAction przechowuje stan (dostępna/zaznaczona), a QMenu i QToolBar to widoki tej akcji.



RYSUNEK 10.3. Wybór akcji z menu i pasków narzędzi

10.1.2. Ćwiczenie: GUI dla gry w karty

Napisz grę oczko (blackjack). Możliwe są następujące akcje:

- nowa gra,
- rozdaj karty,
- tasuj,
- dobierz,

- zostać (nie dobieramy nowej karty),
- wyjść.

Akcje powinny być dostępne poprzez pasek menu oraz przez pasek narzędzi. Aplikacja powinna przypominać tę przedstawioną na rysunku 10.4. Przejdźmy do zwięzłego omówienia reguł gry.



RYSUNEK 10.4. Zrzut ekranu gry w oczko

Na początku gry użytkownik oraz rozdający otrzymują po dwie karty. Użytkownik może zero lub więcej razy dobrać karty. Każde dobranie powoduje dodanie do ręki użytkownika jednej karty. Użytkownik kończy dobieranie, wykonując akcję Zostań.

Celem gry jest uzyskanie największej liczby punktów, przy czym nie może ona przekraczać 21.

Zasady ewaluacji ręki są następujące: figury (walet, królowa, król) mają po 10 punktów, as to 1 lub 11 — w zależności od tego, co akurat odpowiada graczowi. Pozostałe karty mają wartość punktową równą ich numerowi. Jeśli masz na ręce asa i waleta, lepiej jest policzyć asa za 11 punktów, co łącznie daje upragnione 21. Natomiast jeśli masz na ręce 7 i 8, a wyciągniesz asa, lepiej przypisać mu wartość 1.

Jeśli gracz przekroczy 21 punktów, automatycznie przegrywa i rozdanie dobiega końca.

Jeśli gracz ma na ręce pięć kart, których suma nie przekracza 21 — wygrywa.

Po tym jak gracz wygra, przegra lub zostanie (nie dobiera karty) rozdający dobiera dowolnie wiele kart tak długo, dopóki nie przekroczy liczby 18. Gdy to nastąpi, rozdający musi zostać, a gra się kończy. Wygrywa gracz, którego wynik jest bliższy (lecz nie większy niż) 21. Jeśli obaj gracze mają po tyle samo punktów, wygrywa rozdający.

Po zakończeniu rozdania użytkownik może wybierać jedynie spośród akcji: Rozdaj, Nowa gra i Wyjść (Dobierz i Zostań są niedostępne).

Po wybraniu przez użytkownika akcji Rozdaj opcja ta staje się nieaktywna aż do zakończenia rozdania.

Aplikacja powinna pamiętać liczbę gier wygranych przez użytkownika i przez rozdającego, przyznając im jeden „duży” punkt za wygraną rozdania. Wyniki powinny być wyświetlane nad kartami gracza.

Rozdawaj karty bez resetowania talii do momentu, w którym karty się skończą lub użytkownik wybierze akcję Rozdaj.

Spróbuj wykorzystać w programie klasę `CardDeck` i inne klasy utworzone w ramach ćwiczeń i przykładów z sekcji 6.9.1. Dodaj do gry grafikę, wyświetlając wizualizację każdej z kart, na przykład w sposób przedstawiony w podrozdziałach 9.5 i 9.6.

Stwórz menu rozwijane i pasek narzędzi z akcjami. `Dobierz` i `Zostań` mają być dostępne dopiero po rozdaniu kart.

Pokaż, ile kart zostało jeszcze w talii, korzystając z przewijanego pola `QSpinBox` w wersji tylko do odczytu.

Akcja `Nowa gra` powinna zerować zapamiętaną liczbę gier wygranych i przegranych przez graczy oraz resetować talię.

Sugestie projektowe

Staraj się rozdzielić klasy modelu od klas widoku — nie wprowadzaj kodu GUI do klas modelu. Rozdział modelu od widoku to dobra praktyka programistyczna, ułatwiająca utrzymywanie projektu.

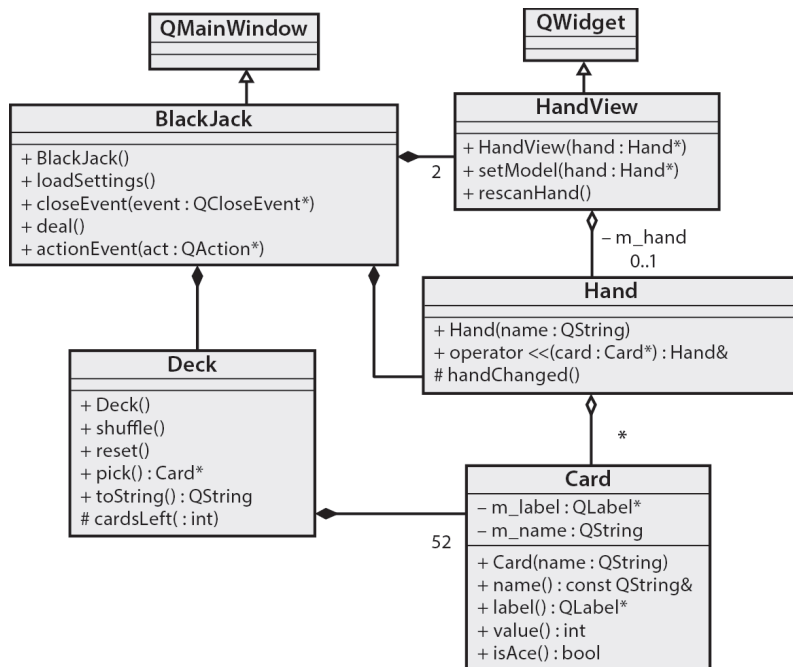
Rysunek 10.5 przedstawia jeden z możliwych projektów aplikacji.

10.2. Obszary i dokowanie

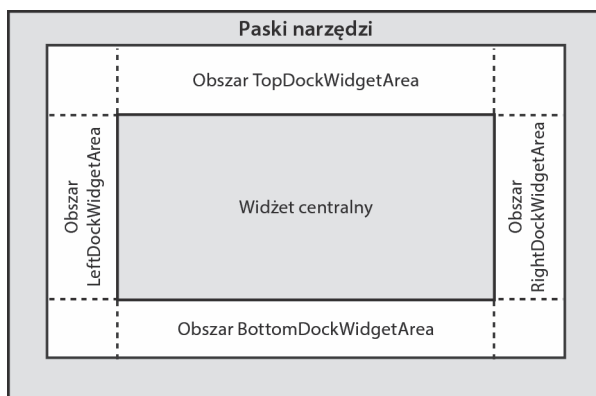
Każda klasa dziedzicząca z `QMainWindow` ma obszary dokowania okien, po jednym z każdej strony **widżetu centralnego**, jak pokazuje rysunek 10.6. Obszary te są używane do podłączania (**dokowania**) okien pomocniczych do widżetu centralnego.

O `QDockWidget` można myśleć jako o kopercie na inny widżet. Ma on pasek tytułu i obszar treści, w którym można umieścić inny widżet. W zależności od konfiguracji atrybutów `QDockWidget` może być odłączany (wówczas „pływa”), rozciągany, zamykany, przeciągany w inne miejsce lub podłączany do tego samego lub innego obszaru dokowania — na życzenie użytkownika końcowego. Dozwolona jest sytuacja, w której kilka takich widżetów zajmuje ten sam obszar dokowania.

Okno `QMainWindow` poprawnie tworzy przeciągalne obszary `QSplitter` pomiędzy widżetem centralnym a widżetami `QDockWidget`. Główne okno ma dwie podstawowe funkcje do zarządzania obszarami dokowania:



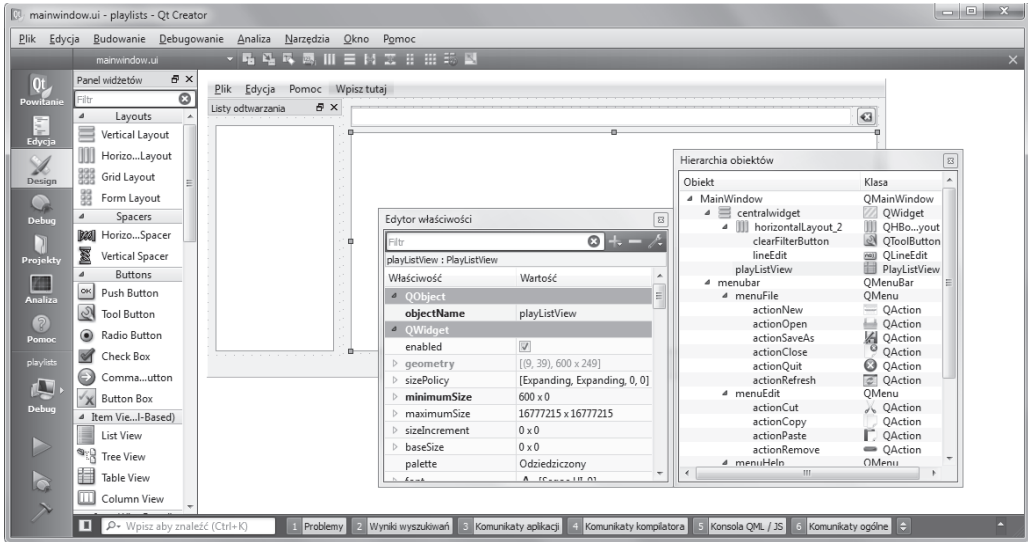
RYSUNEK 10.5. Diagram klas UML dla gry w oczko (klasy: Blackjack — oczko, Deck — talia, HandView — widok ręki, Hand — ręka, Card — karta)



RYSUNEK 10.6. Obszary dokowania QMainWindow

1. `setCentralWidget(QWidget*)` — ustawia widżet centralny;
2. `addDockWidget(Qt::DockWidgetAreas, QDockWidget*)` — dodaje podany `QDockWidget` do określonego obszaru dokowania.

Okna dokowane pełnią istotną rolę w zintegrowanych środowiskach programistycznych, ponieważ w różnych sytuacjach wymagane są różne zestawy narzędzi i widoki. Każde okno dokowane jest widżetem, który można dość łatwo „podpiąć” do głównego okna, jak pokazuje rysunek 10.7.



RYSUNEK 10.7. Pływające okna QDockWindow w Designerze

Jak większość aplikacji Qt, również Designer — narzędzie do projektowania i tworzenia interfejsów użytkownika — intensywnie korzysta z okien dokowanych. Designer składa się z Edytora widżetów (widżet centralny) i dokowanych widoków narzędzi takich jak:

- Panel widżetów,
- Inspektor obiektów,
- Edytor właściwości,
- Edytor sygnałów / slotów,
- Edytor akcji,
- Edytor zasobów.

Podłączane okna nie zawsze muszą być widoczne, dlatego można sterować ich wyświetlaniem za pomocą menu Widok (*View*). Jest ono zwracane przez funkcję `QMainWindow::createPopupMenu()`, co pozwala na odwoływanie się do niego z menu kontekstowych i dodawanie go do pasków narzędzi i menu rozwijanych.

10.3. QSettings: konfiguracja aplikacji

Większość aplikacji pozwala użytkownikom na konfigurację ustawień. Ustawienia (opcje, preferencje) powinny być trwałe. Właśnie w tym celu wprowadzono `QSettings`.

Obiekt `QSettings` (ustawienia) musi przed pierwszym użyciem zostać zainicjalizowany następującymi wartościami: nazwa aplikacji, nazwa organizacji, domena organizacji. Te wymagania chronią przed przypadkowym nadpisaniem ustawień jednej aplikacji przez ustawienia drugiej. Jeśli jednak informacje te są już w posiadaniu `QApplication`, jak w przykładzie 10.7, to konstruktor domyślny `QSettings` może ich użyć ponownie.

PRZYKŁAD 10.7. src/widgets/mainwindow/mainwindow-main.cpp

```

#include "mainwindow.h"
#include <QApplication>

int main( int argc, char ** argv ) {
    QApplication app( argc, argv );
    app.setOrganizationName("objectlearning");
    app.setOrganizationDomain("objectlearning.net");
    app.setApplicationName("mainwindow-test");
    MyMainWindow mw;
    mw.show();
    return app.exec();
}

```

Po takiej inicjalizacji `QApplication` możliwe jest utworzenie instancji `QSettings` z wykorzystaniem konstruktora domyślnego.

Kiedy tworzysz aplikację opartą na `QMainWindow`, jednymi z pierwszych ustawień, które będziesz chciał pamiętać, będą rozmiar i położenie okna. Być może zechcesz również zapamiętać nazwy ostatnich dokumentów otwieranych przez aplikację.

Obiekt `QSettings` przechowuje trwałe mapy par klucz i wartość. Jest on także obiektem `QObject` i korzysta z rozwiązania przypominającego interfejs właściwości `QObject`: `setValue()` ustawia wartość `value`, a `value()` ją zwraca. Za jego pomocą można składować dowolne dane, które mają być pamiętane pomiędzy uruchomieniami aplikacji.

Obiekt `QSettings` wymaga podania nazwy organizacji i nazwy aplikacji, ale jest w stanie w konstruktorze domyślnym pobrać te dane od aplikacji. Każda kombinacja nazw definiuje unikatową trwałą mapę, która nie wchodzi w konflikt z ustawieniami aplikacji Qt o innych nazwach.

Wzorzec Monostate

Klasa, która pozwala wielu instancjom na współdzielenie tego samego stanu, jest implementacją **wzorca Monostate** (pojedynczy stan). Dwie instancje `QSettings` o tej samej nazwie organizacji i aplikacji mogą odwoływać się do tych samych trwałych danych. W ten sposób aplikacje mogą łatwo uzyskiwać dostęp do wspólnych ustawień z różnych plików źródłowych.

Klasa `QSettings` implementuje wzorzec Monostate.

Właściwy mechanizm trwałego przechowywania danych `QSettings` jest zależny od implementacji i dość elastyczny. Możliwe jest wykorzystanie rejestru Win32 (w Windows) i katalogu `$HOME/.settings` (w Linuksie). Więcej informacji znajdziesz w dokumentacji API klasy `QSettings`.

Funkcja `QMainWindow::saveState()` (zapisz stan) zwraca tablicę `QByteArray` zawierającą informacje o paskach narzędzi i widżetach podłączonych do głównego okna. Wykorzystana jest do tego właściwość `objectName` każdego z podwidżetów, dlatego istotne jest, by nazwy te były unikatowe. Funkcja `saveState()` ma opcjonalny parametr `int versionNumber`. Obiekt `QSettings` przechowuje tablicę `QByteArray` pod kluczem "state".

Funkcja `QMainWindow::restoreState()` (przywróć stan) pobiera `QByteArray`, prawdopodobnie utworzoną przez `saveState()`, i w oparciu o zapisane w niej informacje odpowiednio rozmieszcza paski narzędzi i podłączone widżety. Ona również ma opcjonalny parametr `versionNumber`. Odczytaj ustawienia po skonstruowaniu obiektu, jak w przykładzie 10.8.

PRZYKŁAD 10.8. src/widgets/mainwindow/mymainwindow.cpp

```
[...]

void QMainWindow::readSettings() {
    QSettings settings;
    QPoint pos = settings.value("pos", QPoint(200, 200)).toPoint();
    QSize size = settings.value("size", QSize(400, 400)).toSize();
    QByteArray state = settings.value("state", QByteArray())
        .toByteArray();

    restoreState(state);
    resize(size);
    move(pos);
}
```

Ustawienia należy zapisać po tym, gdy użytkownik wyrazi chęć wyjścia z aplikacji, ale przed zamknięciem okna. Odpowiednim na to miejscem jest kod obsługi zdarzeń, gdyż można tam obsłużyć zdarzenie, zanim zrobi to sam widżet. Przykład 10.9 przedstawia funkcję obsługi zdarzenia zamknięcia okna, która zapisuje informacje konfiguracyjne w `QSettings`.

PRZYKŁAD 10.9. src/widgets/mainwindow/mymainwindow.cpp

```
[...]

void QMainWindow::closeEvent(QCloseEvent* event) {
    if (maybeSave()) {
        writeSettings();
        event->accept();
    } else {
        event->ignore();
    }
}

void QMainWindow::writeSettings() {
    /* zapisz położenie i rozmiar głównego okna */
    QSettings settings;
    settings.setValue("pos", pos());
    settings.setValue("size", size());
    settings.setValue("state", saveState());
}
```

10.4. Schowek i transfer danych

Czasami niezbędne jest pobranie danych z pewnego miejsca i przeniesienie ich gdzie indziej. Można w tym celu zastosować metodę „kopiuj i wklej” (schowek) lub „przeciągnij i upuść”. W obu przypadkach wykorzystywane są te same klasy.

Każda aplikacja Qt ma dostęp do **schowka** systemowego poprzez wywołanie `qApp->clipboard->board()`. Schowek przechowuje dane określonego typu (tekst, grafika, URL, typy zdefiniowane przez użytkownika). Jeśli chcesz umieścić dane w schowku, stwórz obiekt `QMimeType`, zakoduj dane i wywołaj `QClipboard->setMimeType()`. Do zapisywania danych poszczególnych typów służą następujące funkcje:

- `setText()` — dane tekstowe;
- `setHtml()` — tekst ze znacznikami;
- `setImageData()` — dane graficzne;
- `setUrls()` — lista adresów URL lub nazw plików.

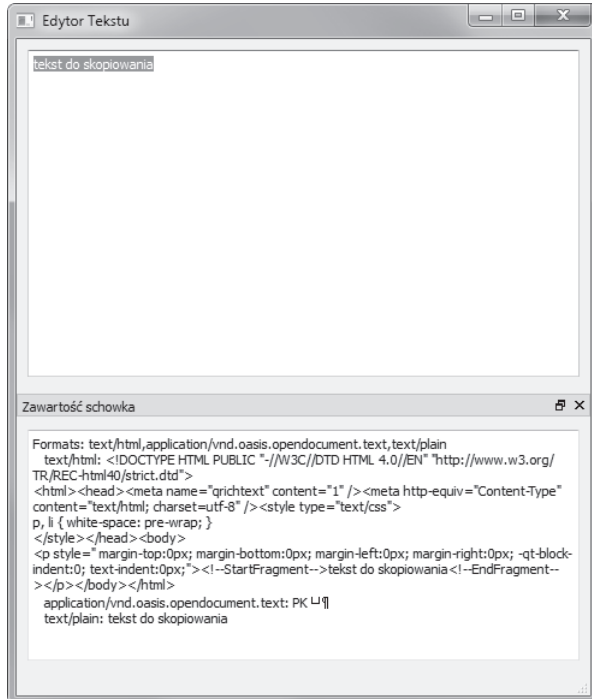
W przykładzie 10.10 podłączamy sygnał `changed` (zmieniony) schowka systemowego do slotu `MainWindow clipboardChanged()`. Sygnał jest emitowany za każdym razem, gdy dowolna aplikacja skopiuje coś do schowka. Możesz uruchomić przykład i sprawdzić, jakie typy danych są dostępne podczas kopiowania danych pochodzących z `QTextEdit` lub z innych aplikacji.

PRZYKŁAD 10.10. `src/clipboard/mainwindow.cpp`

```
[...]
MainWindow::MainWindow(QWidget* parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setUpUi(this);
    connect (qApp->clipboard(), SIGNAL(changed(QClipboard::Mode)),
            this, SLOT(clipboardChanged(QClipboard::Mode)));
}

void MainWindow::clipboardChanged(QClipboard::Mode) {
    QStringList sl;
    QClipboard *cb = qApp->clipboard();
    const QMimeType *md = cb->mimeType();
    sl << "Formaty: " + md->formats().join(",");
    foreach (QString format, md->formats()) {
        QByteArray ba = md->data(format);
        sl << "   " + format + ": " + QString(ba);
    }
    ui->clipboardContentsEdit->setText(sl.join("\n"));
}
[...]
```

Rysunek 10.8 pokazuje, co się dzieje, gdy tekst jest kopiowany z `QTextEdit`. W tym konkretnym przypadku dane w schowku zostały zakodowane na trzy sposoby: jako czysty tekst,



RYSUNEK 10.8. Demonstracja schowka systemowego

HTML oraz w otwartym formacie OASIS. Ostateczny wybór jest zależny od tego, jaki format danych jest oczekiwany w miejscu docelowym (tam gdzie tekst zostanie wklejony lub upuszczony).

10.5. Wzorzec Poleccie

Dodanie do programu akcji „cofnij” uspokaja użytkownika i zachęca go do eksperymentowania z możliwościami programu. Takie rozwiązanie może okazać się lepsze niż okienko wymuszające na użytkownika zatwierdzenie wprowadzonych zmian. Qt oferuje gotowe klasy ułatwiające tworzenie aplikacji pozwalających na cofanie zmian.

Wzorzec Poleccie

Wzorzec Poleccie (ang. *Command*), opisany w [Gamma95], opakowuje operacje w postaci obiektów o wspólnym interfejsie wykonawczym. Taka architektura umożliwia ustawianie operacji w kolejce, logowanie operacji oraz cofnięcie wyników operacji już wykonanej. Poprawna implementacja tego wzorca może współdzielić kod obsługi błędów i sytuacji wyjątkowych.

Klasy Qt, które w mniejszym lub większym stopniu implementują ten wzorzec, to `QUndoCommand`, `QRunnable` i `QAction`.

Implementacja wzorca Polecenie w Qt jest wyjątkowo prosta:

- Możesz stworzyć kilka poleceń i ustawić je w kolejce w ramach odpowiedniego kontenera (np. kolejki `QQueue`).
- Polecenie cofnięcia `QUndoCommand` można wykonać, wkładając je na stos `QUndoStack`.
- Jeśli polecenia mają być wykonywane równolegle, możesz dodać dziedzicznie z `QRunnable` i zaplanować ich uruchomienie w ramach puli wątków za pomocą `QtConcurrent::Run()`².
- Możesz zapisać polecenia w pliku (serializacja) i odczytać je później (może nawet na innej maszynie). Przydaje się to do pracy wsadowej i rozproszonej³.

10.5.1. Polecenie `QUndoCommand` i przetwarzanie obrazów

Kolejny przykład wiąże się z zastosowaniem polecenia cofnięcia `QUndoCommand`. Nasz przykładowy program służy do obróbki zdjęć⁴. Wykorzystuje on klasę `QImage`, niezależną od sprzętu reprezentację obrazu pozwalającą na manipulację na poziomie pojedynczych pikseli. Klasa `QImage` obsługuje wiele spośród najpopularniejszych formatów plików graficznych, w tym JPEG⁵, stratny system kompresji, którego użyjemy na potrzeby przykładu.

Na początku, co pokazuje przykład 10.11, wprowadzimy kilka typowych operacji na obrazach. Wszystkie będą dziedzyczyły z `QUndoCommand`. Pierwsza operacja zmienia kolor pikseli, mnożąc wartości składowych RGB (czerwony, zielony, niebieski) przez podane wartości `double`. Druga operacja zastępuje połowę obrazu lustrzanym odbiciem drugiej połowy, poziomo lub pionowo w zależności od wyboru użytkownika. Konstruktory obu operacji pobierają referencję do źródłowej grafiki `QImage` oraz tworzą pustą instancję `QImage` o tym samym rozmiarze i formacie.

PRZYKŁAD 10.11. `src/undo-demo/image-manip.h`

```
[...]
class AdjustColors : public QUndoCommand {
public:
    AdjustColors(QImage& img, double radj, double gadj, double badj)
        : m_Image(img), m_Saved(img.size(), img.format()), m_RedAdj(radj),
          m_GreenAdj(gadj), m_BlueAdj(badj)    {setText("dostosuj kolory"); }
    virtual void undo();
    virtual void redo();

private:
    QImage& m_Image;
    QImage m_Saved;
    double m_RedAdj, m_GreenAdj, m_BlueAdj;
};
```

² Wątkom poświęcony zostanie podrozdział 17.2.

³ Wzorec Serializator został omówiony w sekcji 7.4.1.

⁴ Inspiracją dla przykładu były prace Guzdiała i Ericsona [Guzdial07] i ich projekt MediaComp.

⁵ <http://jpeg.org/>

```

    void adjust(double radj, double gadj, double badj);
};

class MirrorPixels : public QUndoCommand {
public:
    virtual void undo();
    virtual void redo();
[...]
```

Każda z operacji tworzy kopię oryginalnej grafiki przed rozpoczęciem wprowadzania zmian. Przykład 10.12 przedstawia implementację operacji `AdjustColor` modyfikującej kolory. Jej konstruktor przechodzi przez wszystkie piksele `QImage` i wywołuje na każdym z nich funkcję `pixel()`. Funkcja ta zwraca kolor jako czwórkę `ARGB` zapisaną jako ośmiobajtowa wartość `int` (bez znaku) w postaci `AARRGGBB`, gdzie każda para bajtów reprezentuje odpowiednią składową koloru. Poszczególne składowe z czwórki (o aliasie `typedef QRgb`) możemy pobrać przy użyciu funkcji `qRed()` (czerwony), `qGreen()` (zielony) i `qBlue()` (niebieski). Wartości należą do przedziału od 0 do 255⁶. Następnie wartości `RGB` piksela zostają zastąpione wartościami zmodyfikowanymi przez mnożenie. Pamiętaj, że wynik mnożenia `double` i `int` zostanie zapisany w zmiennej typu `int`, co spowoduje jego obcięcie. Konsekwencją tego obcięcia jest nieodwracalność tej operacji — mnożenie wartości wynikowej przez odwrotność parametru nie przywróci stanu wyjściowego.

PRZYKŁAD 10.12. `src/undo-demo/image-manip.cpp`

```

[...]
```

```

void AdjustColors::adjust(double radj, double gadj, double badj) {
    int h(m_Image.height()), w(m_Image.width());
    int r, g, b;
    QRgb oldpix, newpix;
    m_Saved = m_Image.copy(QRect()); // zapisz kopię całego obrazka
    for(int y = 0; y < h; ++y) {
        for(int x = 0; x < w; ++x) {
            oldpix = m_Image.pixel(x,y);
            r = qRed(oldpix) * radj;
            g = qGreen(oldpix) * gadj;
            b = qBlue(oldpix) * badj;
            newpix = qRgb(r,g,b);
            m_Image.setPixel(x,y,newpix);
        }
    }
}

void AdjustColors::redo() {
    qDebug() << "AdjustColors::redo()";
    adjust(m_RedAdj, m_GreenAdj, m_BlueAdj);
}
```

⁶ Czwartą parą to komponent *alfa*, określający przezroczystość piksela. Jego domyślna wartość to `ff` (255) — nieprzezroczysty.

```
void AdjustColors::undo() {
    qDebug() << "AdjustColors::undo()";
    m_Image = m_Saved.copy(QRect());
}
```

Metoda `undo()` (cofnij) przywraca zapisaną kopię obrazu, a `redo()` (ponów) znów wywołuje funkcję modyfikującą piksele.

GUI zaprojektowaliśmy w Qt Creatorze. Grafika `QImage` jest konwertowana do postaci `QPixmap` i wyświetlana na ekranie wewnątrz `QLabel`. Rysunek 10.9 przedstawia oryginalne zdjęcie, przed jakimikolwiek przekształceniami.



RYSUNEK 10.9. Oryginalna grafika

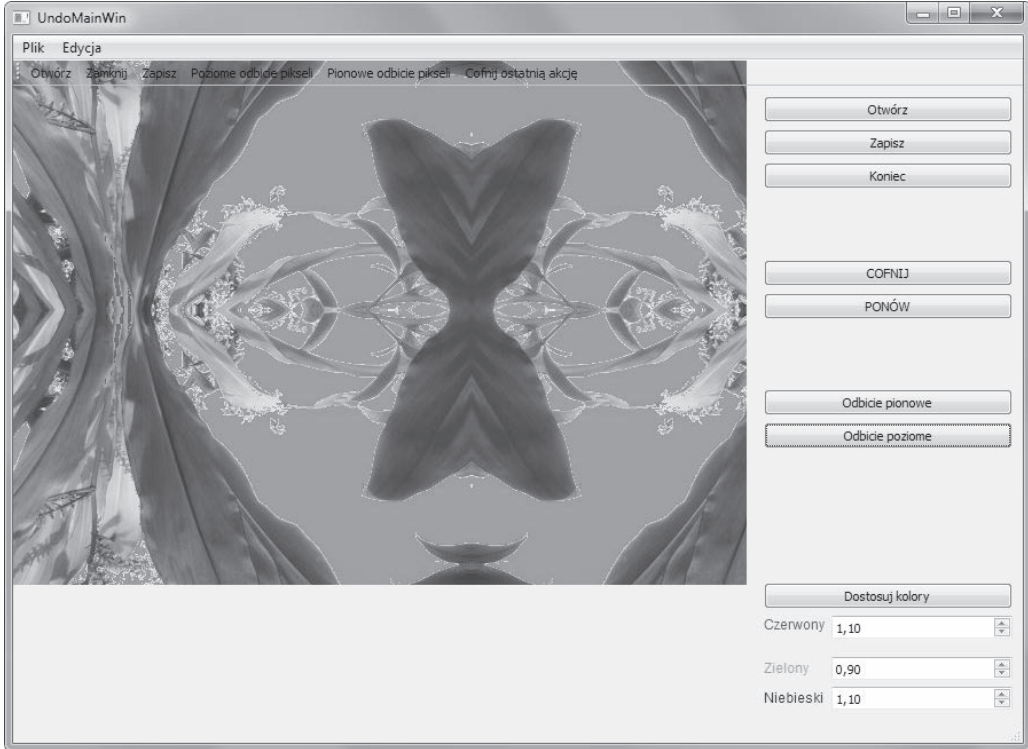
Rysunek 10.10 przedstawia przykry dla oka efekt operacji `AdjustColors` oraz obu wersji `MirrorPixels`.

Klasa `UndoMainWin` dziedziczy z `QMainWindow` i używa `QUndoStack`. Domyślnie Qt Creator osadza klasę `Ui` jako składową wskaźnikową `UndoMainWin`. W przykładzie 10.13 sloty prywatne początkowo były jedynie załączkami wygenerowanymi przez Qt Creator, gdy w Designerze wybraliśmy opcję *Idź do slotu*.

PRZYKŁAD 10.13. `src/undo-demo/undomainwin.h`

```
#ifndef UNDOMAINWIN_H
#define UNDOMAINWIN_H

#include <QMainWindow>
#include <QUndoStack>
```



RYSUNEK 10.10. Grafika po modyfikacji

```

class QWidget;
class QLabel;
class QImage;
class QEvent;
namespace Ui {
    class UndoMainWin;
}

class UndoMainWin : public QMainWindow {
    Q_OBJECT
public:
    explicit UndoMainWin(QWidget* parent = 0);
    ~UndoMainWin();

public slots:
    void displayImage(const QImage& img);

private:
    Ui::UndoMainWin* ui;
    QLabel* m_ImageDisplay;
    QImage m_Image;
    QUndoStack m_Stack;

private slots:
    void on_redoButton_clicked();
    void on_openButton_clicked();
    void on_actionAdjust_Color_triggered();

```

```

void on_actionUndo_The_Last_Action_triggered();
void on_actionHorizontal_Mirror_triggered();
void on_actionVertical_Mirror_triggered();
void on_actionQuit_triggered();
void on_actionSave_triggered();
void on_actionClose_triggered();
void on_saveButton_clicked();
void on_quitButton_clicked();
void on_adjustColorButton_clicked();
void on_undoButton_clicked();
void on_verticalMirrorButton_clicked();
void on_horizontalMirrorButton_clicked();
void on_actionOpen_triggered();
};

#endif // UNDOMAINWIN_H

```

W przykładzie 10.14 masz okazję zaobserwować styl implementacji stosowany przez Qt Creator w celu połączenia ze sobą poszczególnych elementów. Zwróć uwagę na związłe definicje prywatnych slotów z przykładu 10.13.

PRZYKŁAD 10.14. `src/undo-demo/undomainwin.cpp`

```

[...]
#include "image-manip.h"
#include "ui_undomainwin.h"
#include "undomainwin.h"

UndoMainWin::UndoMainWin(QWidget *parent)
: QMainWindow(parent), ui(new Ui::UndoMainWin),
  m_ImageDisplay(new QLabel(this)), m_Image(QImage()) {
    ui->setupUi(this);
    m_ImageDisplay->setMinimumSize(640,480);
}

UndoMainWin::~UndoMainWin() {
    delete ui; //1
}

void UndoMainWin::displayImage(const QImage &img) {
    m_ImageDisplay->setPixmap(QPixmap::fromImage(img));
}

void UndoMainWin::on_actionOpen_triggered() {
    m_Image.load(QFileDialog::getOpenFileName());
    displayImage(m_Image);
}

void UndoMainWin::on_horizontalMirrorButton_clicked() {
    MirrorPixels* mp = new MirrorPixels(m_Image, true);
    m_Stack.push(mp);
    displayImage(m_Image);
}

void UndoMainWin::on_adjustColorButton_clicked() {
    double radj(ui->redSpin->value()), gadj(ui->greenSpin->value()),

```

```

    badj(ui->blueSpin->value());
    AdjustColors* ac = new AdjustColors(m_Image, radj, gadj, badj);
    m_Stack.push(ac);
    displayImage(m_Image);
}
[...]
```

///*1* To nie QObject ani dziecko, więc musi zostać ręcznie usunięty.

Klasa QImage została zoptymalizowana pod kątem operacji na pikselach. Klasa QPixmap korzysta z pamięci wideo i jest używana przez widzety wyświetlające grafiki na ekranie. Jak wspomnieliśmy wcześniej, obraz QImage można przekształcić na QPixmap i wyświetlić na QLabel.

10.5.1.1. Ćwiczenia: polecenie QUndoCommand i przetwarzanie obrazów

Dodaj do kodu z przykładu 10.11 więcej operacji modyfikujących obraz. Oto kilka pomysłów:

- 1. Obraz monochromatyczny.** Przekształć obraz zdefiniowany z użyciem składowych RGB do postaci monochromatycznej, opartej na odcieniach szarości. Kolor szary uzyskasz, nadając wszystkim trzem składowym koloru tę samą wartość. Niestety, jeśli ograniczysz się do ustawienia dla nowego piksela wartości średniej z trzech składowych, wynik okaże się zbyt ciemny. Standardowy sposób radzenia sobie z tym problemem opiera się na założeniu, że niebieski jest postrzegany jako „ciemniejszy” niż czerwony. Lepszy efekt uzyskasz, przemnażając wartości kolorów przez współczynniki wagowe⁷:

```
czer *= 0.30; ziel *= 0.59; nieb *=0.11;
```

Po tej operacji możesz już wyznaczyć luminancję piksela. Luminancja (natężenie oświetlenia) to wartość int równa ważonej średniej trzech składowych koloru. W naszym przypadku wagi już zostały nadane, pozostaje więc tylko wyliczenie średniej:

```
luminancja = (czer + ziel + nieb) / 3
```

Na koniec zastąp wartość każdej składowej nowo wyznaczoną wartością.

- 2. Negatyw.** Przekształć obraz zdefiniowany z użyciem składowych RGB do postaci negatywu. Wystarczy zamienić wartość składowej v na $255 - v$.
- 3. Zamiana kolorów.** Przesuń składowe koloru tak, by składowa czerwona otrzymała wartość niebieskiej, zielona czerwonej, a niebieska zielonej.
- 4. Trzy kolory.** Wyznacz intensywność ci każdego piksela (w prosty sposób jako zwykłą średnią składowych). Jeśli wartość ci jest mniejsza niż 85, zmniejsz wartości

⁷ Zagadnienie luminancji zostało omówione np. tutaj: <http://nemesi.lonestar.org/reference/internet/web/color/luminance.html>.

składowych czerwonej i niebieskiej do 0. Jeśli jest większa od 85, ale niższa od 170, wyzeruj składowe niebieską i zieloną. Jeśli ci to 170 lub więcej, wyzeruj czerwony i zielony.

5. **Wyodrębnienie krawędzi.** Porównaj natężenie każdego piksela z natężeniem piksela leżącego dokładnie pod nim. Jeśli wartość absolutna różnicy przekracza wyznaczoną (przekazaną jako argument) wartość progową, zmień kolor górnego piksela na czarny (wszystkie składowe mają wartość 0), a w przeciwnym razie — na biały (same wartości 255).

10.6. Umiędzynarodawianie i funkcja tr()

Gdyby Twój program miał zostać przetłumaczony na inny język (**umiędzynarodowiony**), mógłbyś liczyć na wsparcie Qt Linguist i narzędzi tłumaczeniowych Qt, które rozwiązują problemy związane z organizacją i składowaniem przetłumaczonych fraz. Możesz przygotować kod do tłumaczenia, otaczając każdą frazę do przetłumaczenia wywołaniem funkcji `QObject::tr()`. Gdy zostanie użyta w sposób niestatyczny, użyje nazwy klasy obiektu (pobranej z `QObject`) jako kontekstu do grupowania przetłumaczonych fraz.

Funkcja `tr()` pełni dwie role:

1. Umożliwia narzędziu `lupdate` wydobycie wszystkich fraz, które mają być tłumaczone.
2. Jeśli dostępne jest tłumaczenie i wybrano język, zwraca przetłumaczoną frazę.

Jeśli nie podano tłumaczenia, funkcja zwróci oryginalny tekst.



UWAGA

Ważne jest, by każda fraza, która ma być tłumaczona, znajdowała się w całości wewnątrz funkcji `tr()` i by była dostępna na etapie kompilacji. W przypadku łańcuchów znaków z parametrami należy użyć funkcji `QString::arg()` w celu umieszczenia parametrów w wersji przetłumaczonej. Na przykład:

```
statusBar()->message(tr("Zakończono %1 z %2. Postęp:
%3%").arg(processed).arg(total).arg(percent));
```

Dzięki temu możliwa jest zmiana kolejności parametrów w tłumaczeniu, co bywa konieczne dla niektórych par języków.

Do tłumaczenia używane są następujące narzędzia:

1. `lupdate` — przeszukuje pliki `.ui` Designera i pliki C++ pod kątem fraz do przetłumaczenia. Generuje plik `.ts`.
2. `linguist` — edytuje pliki `.ts`, pozwala użytkownikowi na podanie tłumaczeń.
3. `lrelease` — czyta pliki `.ts` i na ich podstawie generuje pliki `.qm` używane przez aplikację do zastosowania tłumaczeń.

Więcej informacji na temat narzędzi do umiędzynarodawiania znajdziesz w dokumentacji⁸.

10.7. Ćwiczenia: Główne okna i akcje

1. Napisz aplikację do edycji tekstu. Niech `QTextEdit` pełni rolę widżetu centralnego.
 - Tytuł okna ma pokazywać nazwę pliku oraz informację o niezapisanych zmianach.
 - Menu *Plik*: dodaj akcje *Otwórz*, *Zapisz jako* i *Zakończ*.
 - Menu *Pomoc*: dodaj akcje *O programie* i *O Qt*.
 - Jeśli są niezapisane zmiany, przed opuszczeniem programu poproś użytkownika o potwierdzenie.

2. Napisz aplikację, która umożliwia użytkownikowi otwarcie pliku tekstowego (lub pliku ze znacznikami) i przeglądanie oraz przewijanie jego zawartości. Widok powinien pokazywać co najmniej 10 linii naraz.

Użytkownik może wyszukać w pliku interesujący go łańcuch znaków. Jeśli łańcuch zostanie odnaleziony, plik powinien zostać przewinięty do odpowiedniej linii, by użytkownik widział kontekst odnalezionego fragmentu. Jeśli plik nie zawiera danego tekstu, pokaż komunikat na pasku statusu.

Daj użytkownikowi możliwość przechodzenia do następnych i poprzednich wystąpień szukanego łańcucha znaków.

Dodaj przycisk *Zamknij*, którego użycie usuwa z widoku dotychczas wyświetlany plik i zachęca użytkownika do otwarcia innego lub wyjścia z aplikacji.

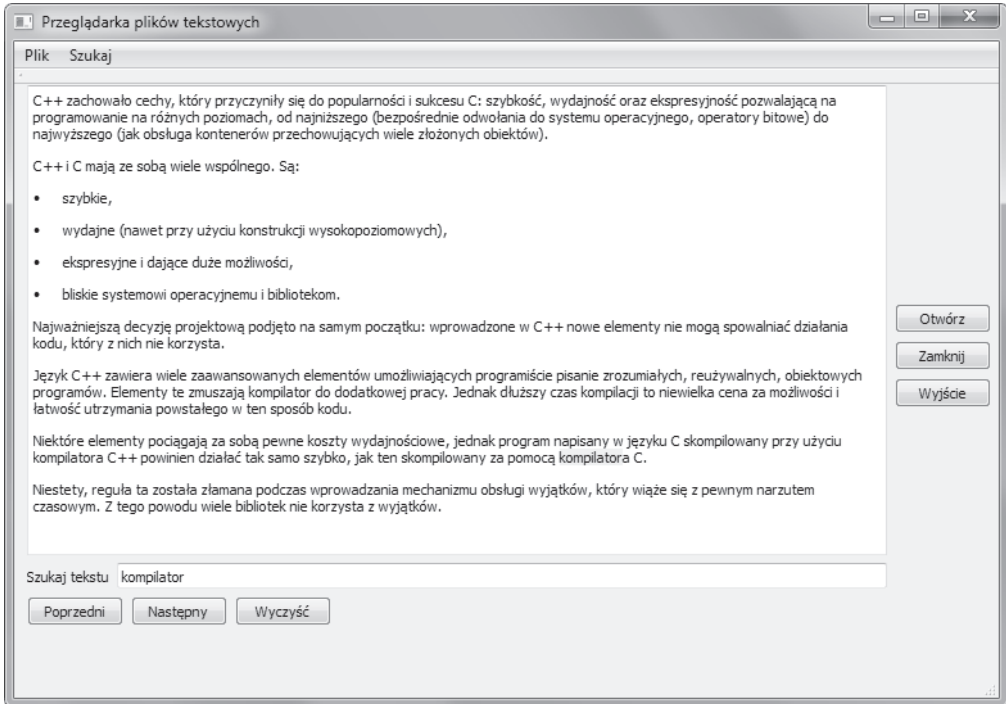
Rysunek 10.11 przedstawia zrzut ekranu z analogicznego programu. Dwa menu zawierają akcje duplikujące akcje dostępne w postaci przycisków. Wyjątkiem jest *Wyczyść*, która jedynie usuwa tekst z pola wejściowego.

3. Widżet `QTextEdit` ma metody pozwalające na cofanie i przywracanie. Przyjrzyj się im i znajdź odpowiedź na pytanie, co dokładnie można cofnąć lub przywrócić.

10.8. Powtórka

1. Jakie są główne funkcje `QMainWindow`?
2. Jak można zainstalować pasek `QMenuBar` w oknie `QMainWindow`?
3. W jaki sposób można zapisać, a później przywrócić rozmiar, położenie i układ widżetów w aplikacji GUI?
4. Czym jest widżet centralny?
5. Jak sprawić, by dany widżet został widżetem centralnym?
6. Do czego służą widżety dokowalne?

⁸ <http://qt-project.org/doc/qt-5.1/qtdoc/index.html>



RYSUNEK 10.11. Przeglądarka plików tekstowych

7. Ile takich widżetów może mieć aplikacja?
8. Jak się ich używa?
9. Co to jest akcja?
10. Jak używać akcji?
11. Czym jest grupa akcji? Po co miałbyś dodawać ją do aplikacji?
12. Jak napisać aplikację dostępną dla użytkowników posługujących się różnymi językami?

Skorowidz

A

- abstrakcyjna klasa bazowa,
 - 177, 178
- adapter, 448
- agregacja, 133, 197, 286
- AJAX, 232
- akcje, 301, 323
- akcje QAction, 303
- algorytm, 331, 332
- algorytm mapReduce(), 477
- aliasy operatorów, 496
- alternatywa, 66
- anonimowe przestrzenie nazw,
 - 546
- ANSI, 30, 515
- antywzorce, 237
- antywzorce metodyczne, 238
- API DOM, 417
- API SAX, 416
- aplikacje QtSQL, 483
- apt, 617, 619
- argumenty
 - opcjonalne, 142
 - operatora, 495
 - wiersza poleceń, 62
- arytmetyka wskaźników, 556
- asocjacja, 134
- asocjacja dwukierunkowa, 134
- atak wstrzyknięcia SQL, 487
- atrybuty, 415

B

- bazy danych, 483
- biblioteka, 202, 214, 221, 226, 230
 - boost, 232
 - Customer, 437
 - DataObjects, 437
 - libgtk, 232
 - libgtk++, 232
- biblioteki
 - DLL, 228
 - programistyczne, 221
 - rozszerzeń, 611
 - szablonów, 591
 - zamknięte, 222
- bit znaku, 509
- bitowe przesunięcie, 332
- blok zagnieżdżony, 114
- bloki, 86
- błędy
 - czasu wykonania, 603
 - konsolidacji, 600, 603
 - logiczne, 35
 - makr, 159
 - ochrony, 73
 - pamięci, 606
 - parsowania, 487
- budowanie bibliotek, 229, 230

C

- charakterystyka operatorów, 496
- czas życia
 - obiektu statycznego, 540
 - QObject, 257
- czyszczenie plików, 595

D

- dane
 - graficzne, 265
 - składowe, 81, 82
- daty, 122
- DDL, Data Definition Language, 485
- Debian, 615
- debugowanie, 603
- definicja
 - funkcji, 531
 - klasy, 82, 531
 - obiektu, 531
- deklaracja
 - funkcji, 37
 - metatypu, 359
 - nazwy, 531
 - zmiennych, 33
- delegaty, 364, 371
- delegaty z gwiazdkami, 372
- dereferencja, 71, 190
- destruktor, 92
- destruktor wirtualny, 568
- diagram
 - dziedziczenia, 177
 - klas, 94, 108, 112, 166, 199
- dokowanie, 309
- dokument XML, 415
- DOM, 421
- dostęp do
 - bazy danych, 485
 - pamięci, 70, 74, 549, 554, 562
 - pól klasy, 121
 - składowych, 85, 349, 596
 - właściwości, 352
 - zasobu, 274

drzewo DOM, 421
 duplikacja danych, 378
 dynamiczne

- łączenie bibliotek, 228
- sprawdzenie typu, 348
- wiązanie, 174
- właściwości, 355

 dyrektywa #include, 33, 596
 dystrybucje Linuksa, 616
 dziedziczenie, 165, 170, 174, 192, 565

- chronione, 577
- proste, 165
- prywatne, 572, 577
- publiczne, 572, 577
- wielokrotne, 571, 573
- wirtualne, 575

E

edycja

- interfejsów, 266
- sygnałów i slotów, 268
- zasobów, 274

 edytory XML, 416
 eksport danych, 443
 element QStandardItem, 378
 enkapsulacja kodu, 87, 365
 etykieta, 534
 ewaluacja wyrażeń logicznych, 506

F

fabryka, 306, 439

- abstrakcyjna, 434, 437, 444
- singletonowa, 438

 filtr zdarzeń, 465
 filtrowanie, 383
 folder, 131
 format Docbook, 427
 formularz, 271

- adresowy, 410
- wejściowy, 288

 framework, 231

- MVC, 365
- Qt, 119
- QTestLib, 258
- Wt, 232

funkcja

- exec(), 486
- invokeMethod(), 360
- main(), 62
- push(), 329
- qRegisterMetaType(), 359
- qSort(), 331
- setGeometry(), 276
- start(), 456
- tr(), 322

 funkcje, 139

- argumenty opcjonalne, 142
- inline, 157, 158
- prototyp, 37
- przekazywanie parametrów, 148
- przeładowywanie, 38, 139, 155, 184
- przesłanianie, 184
- składowe, 82
- składowe const, 104
- skrótów, hash functions, 467
- sygnatura, 38, 139
- ukrywanie, 185
- wirtualne, 173, 179
- wytwórcze, 306
- wywoływanie, 139
- zmienna lista argumentów, 161
- zwracanie referencji, 154
- zwracanie wartości, 154
- zwrotne, 419

 funktry, 336

G

generowanie XML, 425
 generyki, 128
 główne okna, 301, 323
 gospodarz, 104
 gra

- w karty, 307, 310
- w życie, 298, 480

 graficzny interfejs użytkownika, GUI, 307, 462
 grafika, 318

H

hierarchia

- dziedziczenia, 571, 572
- procesów, 460
- typów podstawowych, 512

 HTML, Hypertext Markup Language, 413

I

IDE, 45, 124, 266, 611
 identyfikacja typu, 347, 525
 identyfikator, 57, 84, 531
 ikonki, 273
 implementacja, 87
 import danych, 444
 informacje o typie obiektu, 347
 inicjalizacja

- domyślna obiektu, 90
- obiektów statycznych, 95
- składowych, 169
- zmiennych, 33

 instalacja

- bibliotki, 227, 600
- frameworku Qt, 622

 instancja funkcji, 327
 instancja klasy, 83
 instrukcja, 500

- break, 505
- continue, 505
- switch, 501, 565

 instrukcje

- iteracji, 503
- wyboru, 33, 501, 503
- złożone, 500

 integracja okienek z kodem, 283
 inteligentne wskaźniki, 387
 interfejs

- GUI, 266
- pasywny, 254, 418
- programistyczny, API, 483

 iteracja, 33
 iterator, 128

J

język

- DDL, 485
- SQL, 483
- UML, 87

K

kanał dla błędów, 34

katalog, 131

kategorie

- typów, 325
- widżetów, 264

KDE, 615

klasa, 81, 107

- AbstractMetaLoader, 450
- ArgumentList, 192
- CustomerFactory, 437
- GradTeachingFellow, 577
- MainWindow, 294
- MetaDataTable, 488
- MetaDataValue, 449
- pamięci, 542
- Person, 88
- QAbstractItemModel, 366, 376, 384
- QAction, 302
- QApplication, 253
- QDialog, 269
- QIcon, 276
- QImage, 276, 321
- QLayout, 279
- QMenu, 302
- QMenuBar, 302
- QMetaObject, 345, 347
- QMetaType, 358
- QObject, 243, 255, 346, 541
- QObjectReader, 445
- Qonsole, 462
- QPicture, 276
- QPixmap, 276, 477
- QProcess, 455–68
- QRegExpValidator, 406
- QSqlQuery, 486
- QStandardItemModel, 376, 384

QtConcurrent, 468, 480

QThread, 468, 480

QTreeWidgetItem, 384

QTreeWidgetItem, 384

QValidator, 408

QVariant, 352

QWidget, 263

QXmlContentHandler, 419

QXmlSimpleReader, 419

QXmlStreamReader, 429

QXmlStreamWriter, 429

Slacker, 423, 424

Ui, 285

klasy

- abstrakcyjne, 177
- bazowe, 165, 203
- inline, 603
- konkretne, 178
- kontrolera, 365
- modeli, 366
- pamięci, 540
- poходne, 165
- SAX, 418
- testowe, 258
- WidgetItem, 384
- widoku, 366
- współdzielone przez domniemanie, 338
- wytwórcze, 437

kod

- definiujący klasę, 85
- implementujący klasę, 85
- kliencki, 81, 85
- serializacyjny, 238
- uzupełnień do dwóch, 509
- uzupełnień do jedności, 509
- współbieżny, 468
- wyjścia, 63

kolaż, 481

kolejność inicjalizacji, 186

klas bazowych, 573

składowych, 573

kolory, 316

komentarze, 32

kompilacja, 32, 41, 623

kompilator, 221

kompilator gcc, 275

komponent, 231, 247

kompozycja, 88, 133, 197

kompozyt, 197, 247

komunikacja z bazą danych, 483

konfiguracja

aplikacji, 311

C++/Qt, 621

koniunkcja, 66

konsola bash, 459

konsolidacja, 41

konsolidator, 222, 598

konstruktor, 90, 186

domyślny, 90, 91

explicit, 244

konwertujący, 101

kopiujący, 99, 101, 187, 205

kontener, 127, 325, 549

agregujący, 198

asocjacyjny, 196

na pliki, 131

Qt, 195

wskaźników, 200, 206, 212

zarządzający, 197

kontroler, 266, 462

konwersja, 101, 204

automatyczna, 512

jawna, 514

typów, 66, 519

wskaźników, 519

wyrażeń, 511, 514

kopiowanie kontenerów, 198

kopiujący operator przypisania, 99, 187, 205

kotwice, 400

krotność, 134

kwantyfikatory, 399

L

liczby

8-bitowe, 510

zakres blokowy, 538

licznik

argumentów, 62

czasu, 292

linki symboliczne, 131, 230

Linuks, 616

lista, 127

- inicjalizacyjna składowych, 90
- odtworzenia, 584, 587
- QStringList, 129, 210, 488

literał, 57, 507

literał wyliczeniowy, 508

Ł

łańcuch znaków, 47, 129

M

Mac OS X, 626

makro, 158

- BADABS, 160
- BADCUBE, 160
- METADATAEXPORT, 234
- Q_DECLARE_METATYPE, 358
- Q_ENUMS, 352, 358
- Q_INVOKABLE, 436
- QEXPECT_FAIL, 260
- QTEST_MAIN, 258, 260

manipulatory, 35

mapa QMap, 333

maski wprowadzania danych, 393, 395

mechanizm

- qobject_cast, 347
- refleksji, 345
- RTTI, 348

menu kontekstowe, 302

metadane, 448

metaobiekty, 345, 346

metasymbole maskujące, 394

metatypyw, 358

metawłaściwości, 345

metody, 82, 173

migawki pamięci, 78

model, 265, 363, 390

- QAbstractItemModel, 384
- systemu, 135
- systemu plików, 368
- wątków, 468

modele

- baz danych, 490
- drzewiaste, 384
- edytowalne, 381
- tabel, 375, 490

modelowanie, 612

model-widok-kontroler, 363

moduł

- Qt Core, 121
- Qt Multimedia, 584
- Qt XML, 416

modyfikator

- dostępu, 572
- typu, 76, 82, 85

modyfikowanie grafiki, 319

muteks, 474

MVC, Model-View-Controller, 363

N

nadklasa, 166

nagłówek klasy, 167

nagłówki, 42, 83

nakładka, Wrapper, 448

narzędzia

- debugujące, 293
- do modelowania, 612

narzędzie

- apt, 617
- cinclud2dot, 227
- lupdate, 322
- make, 41
- Makedep, 227
- qmake, 223, 593, 626
- top, 457
- Trolltech, 386
- QObjectBrowser, 293
- Qt Creator, 274
- Umbrello, 88, 612
- valgrind, 606
- walidatory, 398
- wxWidgets, 232

nawigowalność, 134

nazwa znacznika, 508

notacja infiksowa, 65

numer telefonu, 400, 407

O

obiekt, 71, 83

- QFileSystemModel, 368
- QObject, 246, 257
- QSettings, 311
- QTimer, 293
- QWidget, 263

obiekty statyczne, 547

obliczenia matematyczne, 64

obraz monochromatyczny, 321

obrazki, 273

obsługa

- baz danych, 483
- łańcuchów, 129
- sygnałów, 306
- wskaźników, 550
- zdarzeń, 479
- zdarzeń parsowania, 418

obszar statyczny, 540

obszary dokowania, 310

ochrona pamięci, 73

odczyt z pliku, 209

odnajdywanie błędów pamięci, 606

odtworacz plików MP3, 583

- lista odtwarzania, 584, 587
- wyбір źródła, 586

okna

- dialogowe, 54, 265, 269
- dokowane, 310
- plywające, 311

okno QMainWindow, 309

open source, 222, 621

operacje

- atomowe, 471
- na wskaźnikach, 559

operator, 144, 147, 331, 495-499

- ::, 538

<<(), 332

dekrementacji, 65

delete, 73, 552

ekstrakcji, 34

indeksu, 521

inkrementacji, 65

konwersji, 520

modulo, 65

new, 73
 pobrania adresu, 71
 przesunięcia, 34
 przypisania, 99, 103
 qobject_cast, 347
 rzutowania, 347
 static_cast, 515
 typeid, 527
 wstawienia, 34, 89
 wyluskania, 71
 wyrażenia warunkowego, 66
 wywołania funkcji, 523
 zakresu, 84, 538
 operatory
 globalne, 146
 przypisania, 99
 QDataStream, 236
 QTextStream, 235
 rzutowania, 515
 unarne, 71
 wyboru składowych, 527
 opis właściwości QObject, 349
 otwarte przestrzenie nazw, 546

P

pakiet
 dev, 222
 open source, 222
 pamięć, 553, 561
 parametr, 190
 referencyjny, 150
 szablonowy, 325
 parametry przełącznikowe, 190
 parsowanie, 422, 430
 SAX, 417
 sterowane zdarzeniami, 419
 XML, 413, 415, 418
 pełnomocnik filtra sortującego,
 383
 pętla
 do..while, 504
 for, 504
 while, 503
 zdarzeń, 253, 289
 platforma, 221
 Win32, 626
 open source, 621
 plik Makefile, 41, 593
 pliki
 .cpp, 42, 84
 .dll, 600
 .gdbinit, 606
 .h, 42
 .mp3, 583
 .pro, 43
 .qrc, 274
 .ui, 266, 415
 .xmi, 415
 .xml, 416
 binarne, 236
 implementacyjne, 83, 598
 multimedialne, 448
 nagłówkowe, 83, 592, 598
 obiektów, 222
 projektu, 41, 56
 pobieranie danych, 270
 podgląd widżetów, 268
 podklasa, 166
 podłączanie sygnału, 269
 podmenu, 302
 podobiekt, 88, 105
 pola, 81, 82
 pole obrotowe, spin box, 56
 polecenie
 gdb, 605
 make, 41, 594
 qmake, 41, 42, 595
 QUndoCommand, 316
 update-alternatives, 620
 valgrind, 609
 where, 605
 polimorfizm, 165, 172, 439, 568
 połączenie z bazą danych, 485
 POSIX, 621
 powłoka konsolowa, 462
 późne wiązanie, 174, 569
 preprocesor, 596
 procesy potomne, 456
 profiler, 606
 program Designer
 edytory, 311

integracja z Qt Creatorem,
 287
 projektowanie formularzy, 284
 tworzenie interfejsów, 266
 programowanie baz danych, 483
 projekt Model-Widok-
 Kontroler, MVC, 283
 projektowanie
 dziedziczenia, 182
 interfejsów, 263
 prototyp funkcji, 37
 przechodniość, 342
 przeciążanie operatorów
 specjalnych, 520
 przeglądarka obiektów, 293
 przekazywanie
 parametrów przez
 referencje, 149
 parametrów przez wartość,
 148
 przeładowywanie, overload, 38
 funkcji, 139, 184
 operatorów, 144, 147
 przełączniki, 190
 przenoszenie biblioteki, 227
 przepływ sterowania, 253
 przesłanianie funkcji, 184
 przestrzeń nazw, 33, 544, 546
 przestrzeń nazw otwarta, 546
 przetwarzanie
 argumentów, 190, 195
 grafik, 276, 316
 przyciski, 264
 przyjaciele klasy, 89
 publiczny interfejs, 87
 punkty wstrzymania, 606

Q

Qt, 119
 Qt Core, 121
 Qt Creator, 120, 124, 267, 274,
 612
 Qt SDK, 622
 Qt XML, 416
 QSql, 225

R

refaktoryzacja, 166
 referencje, 75, 601
 referencje do const, 152
 refleksja, 345, 361
 reguły konstrukcyjne, 434
 rejestr, 541
 rekord, 485
 relacja, 88

- agregacji, 133
- asocjacji, 134
- całość – część, 248
- dwukierunkowa, 97
- dziedziczenia, 245
- implementacji, 578
- jeden do jednego, 133
- jeden do wielu, 133
- kompozycji, 133
- krotność, 134
- nawigowalność, 134
- rodzic – dziecko, 245
- typu jest, 578

 relacje

- dwukierunkowe, 598
- między klasami, 231

 relacyjne bazy danych, 483
 responsywność GUI, 451
 reużywanie kodu, 231
 rodzaje tablic, 559
 role, 364
 role danych, 380
 rozmiar obiektów, 60
 rozmieszczanie widżetów, 276
 rozszerzanie funkcjonalności, 170
 równoległe

- kalkulatory, 471
- obliczenia, 477

 RTTI, Run Time Type Information, 347, 525
 rysowanie, 297
 rzutowanie, 514

- ANSI, 347, 515
- const_cast, 517
- dynamic_cast, 525
- odrzucające const, 516

reinterpret_cast, 518
 static_cast, 515,
 w dół, 347, 525

S

SAX, Simple API for XML, 413
 schowek, 314
 serializacja, 239, 316, 357
 składnia

- łańcuchowa, 34
- wielokrotnego dziedziczenia, 571
- wyrażeń regularnych, 398

 składowe

- modyfikowalne, 105
- prywatne, 88
- publiczne, 88
- statyczne, 94

 skróty klawiszowe, 381
 slot, 256
 słowo kluczowe, 589

- const, 69, 152, 155
- explicit, 103
- extern, 547
- static, 93, 541
- struct, 81
- virtual, 570
- volatile, 77, 471

 sortowanie, 333, 383
 sortowanie przez kopcowanie, 331
 specyfikacja projektu, 41
 SQL, 483
 SQLite, 483
 stałe, 69
 stałe globalne, 542
 stan

- pamięci, 553
- wskaźnika, 552

 standardowe

- nagłówki, 591
- wejście, 33
- wyjście, 33

 standardy kodowania, 121
 statyczna klasa pamięci, 93

statyczne

- składowe klas, 93
- zmienne globalne, 96
- zmienne lokalne, 93, 352

 statyczny obszar pamięci, 540
 sterownik, 483
 sterownik SQL, 486
 sterta, 73, 540, 552
 STL, Standard Template Library, 591
 stos, 328, 540
 stosowanie

- bibliotek, 222
- fabryk, 439
- referencji, 152

 struktury drzewiaste, 421
 strumienie, 48, 122, 235

- do plików, 50, 54
- XML, 429

 sygnał, 255
 sygnatura funkcji, 38, 139
 symbole maskujące, 394
 symetryczność, 342
 synchronizacja współdzielonych danych, 480
 system

- apt, 617
- plików, 131

 szablony, 325, 330

- funkcji, 326
- klas, 327

 szyfrowanie, 162

Ś

ścieżka

- target.path, 594
- złączeń, 45

 ślad stosu, 605
 śledzenie błędów, 603
 środowisko

- okienkowe, 459
- programistyczne Qt Creator, 120

T

tabele, 485
 tablice, 127, 555, 561
 asocjacyjne, 333
 dynamiczne, 340, 559
 wirtualne, 565
 tagi, 415
 technologia .NET, 232
 tester wyrażeń regularnych, 402
 testowanie modelu, 386, 387
 testy, 258
 tłumaczenie, 322
 transfer danych, 314
 tryb
 edycji sygnałów, 268
 projektowania, 266
 tworzenie
 formularzy, 272
 interfejsów, 266
 obiektów, 434
 programu okienkowego, 583
 projektu, 41
 typ
 bool, 513
 QVariant, 246
 typy, 57, 495, 529
 argumentów, 190
 całkowite, 509
 danych, 195
 kontenerów, 195
 niekompletne, 597
 obiektywne, 246
 polimorficzne, 565
 proste, 59, 67
 relacji, 133
 uogólnione, 325, 330, 342
 wartości, 246
 wylizeniowe, 203, 507

U

układ, 265, 277
 formularza, 271
 QLayout, 276
 widżetów, 276

ukrywanie funkcji, 185
 UML, Unified Modeling
 Language, 87
 unarny obiekt funkcyjny, 478
 unia, 352
 uogólnienia, 128, 325
 uruchamianie procesu, 456
 uszkodzanie pamięci, 551

W

walidacja, 393, 410
 walidacja wyrażeń regularnych,
 406
 walidator numerów telefonów,
 407
 walidatory, 396
 wartości zwracane, 154, 557
 wątek potomny, 473
 wątki, 468
 czyszczenie, 474
 robocze, 480
 zatrzymywanie, 474
 wczesne wiązanie, 174
 wektor argumentów, 62
 weryfikacja instalacji, 625
 węzły, 415
 widoki, 265, 363, 390
 widoki drzew XML, 425
 widżet, 263
 centralny, 309
 ProductForm, 283
 QObjectBrowser, 293
 widżety
 dla aplikacji technicznych, 611
 odstępny, 279
 rozciąganie, 279
 rozpórki, 279
 układ, 276, 282
 ustawienia rozmiaru, 280
 wielokrotne dziedziczenie, 286
 wiersz poleceń, 62
 przetwarzanie argumentów,
 190, 195
 Win32, 626
 wirtualne klasy bazowe, 576

właściwości
 dynamiczne, 355
 Q_PROPERTY, 349
 wprowadzanie danych, 270
 wskaźnik, 70, 74, 200, 550, 556,
 559
 pusty, null pointer, 71
 void*, 518
 wskaźniki
 na funkcje, 336
 wirtualne, 565
 współbieżność, 455
 współczynnik przyspieszenia, 475
 współdzielenie zasobów, 201, 338
 wtyczka, plugin, 440
 wybór akcji, 307
 wyciek pamięci, 552, 553, 606
 wyjątek, 73
 wyluskiwanie wskaźników, 596
 wyrażenia, 495, 529
 wyrażenia
 logiczne, 66, 506
 regularne, 398, 405, 410
 walidacja, 406
 wyszukiwanie dzieci, 250
 wywołania bezpośrednie, 174
 wywoływanie funkcji, 139
 wyzwalacze, 374
 wzorce
 konstrukcyjne, 433, 441
 projektowe, 221, 232
 wzorzec
 Fabryka abstrakcyjna, 434
 Fasada, 447
 Iterator, 129
 Kompozyt, 247
 Metaobiekt, 345
 Monostate, 312
 Obserwator, 254, 294
 Pamiętka, 442, 443
 Polecenie, 315
 Pylek, 338, 339
 Serializator, 234
 Singleton, 434, 438

X

XHTML, 414
 XML, Extensible Markup
 Language, 413, 421

Z

zakres
 blokowy, 86, 533, 538
 domyślny identyfikatorów,
 533
 funkcji, 534
 globalny, 537
 identyfikatorów, 531
 instrukcji switch, 535
 klasy, 84, 537
 pliku, 537
 przestrzeni nazw, 536
 zależność
 cykliczna, 598
 kompilacyjna, 225
 konsolidacyjna, 225
 załączanie
 nagłówków, 45
 plików, 596
 zapis do pliku, 209
 zapowiedź klasy, 97, 597

zapytania, 488
 DDL, 485
 przygotowane, prepared
 statements, 486
 zarządzanie
 dziećmi, 250, 253
 procesami, 455, 467
 zależnościami, 224
 zasoby, 273
 zbiory znaków, 399
 zdarzenia, 253, 255
 klawiatury, 464
 rysowania, 297
 zestawy wyników, 488
 zintegrowane środowisko
 programistyczne, IDE, 45,
 124, 266, 611
 zliczanie referencji, 338
 zmienna, 71
 zmienna środowiskowa
 CPPLIBS, 222
 HOME, 231
 INCLUDEPATH, 223
 LD_LIBRARY_PATH, 230
 LIBS, 223
 PATH, 230

zmiennie, 33
 globalne, 541
 lokalne, 93, 352
 qmake, 44
 statyczne, 541
 środowiskowe, 458, 625
 znaczniki, 415
 znak
 dwukropka, 534
 przecinka, 572
 średnika, 504
 znaki
 grupujące, 399
 przechwytyjące, 399
 specjalne, 399
 zwracanie
 referencji, 154
 wartości, 154
 zwrotność, 342

Ź

źródła, 42

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

Wykorzystaj moc duetu C++ i Qt!

Język C++ przez długie lata był liderem wśród języków programowania i wciąż jest rozwijany. Qt to zestaw bibliotek pozwalający na tworzenie atrakcyjnego interfejsu użytkownika, działającego w różnych systemach operacyjnych — Windows, Mac OS X oraz Linux. Projekt jest intensywnie rozwijany od 1992 roku, a najnowsza wersja pozwala nawet na tworzenie aplikacji internetowych i mobilnych. Wykorzystanie możliwości C++ oraz potencjału Qt może dać niezwykle efekty!

W trakcie lektury nauczysz się podstaw języka C++ i zgłębisz tajniki bibliotek Qt. Zdobędziesz interesujące informacje na temat wykorzystania kontenerów, metaobektów, metawłaściwości i mechanizmu refleksji. Dowiesz się, jak wykorzystać wzorzec MVC (ang. Model-View-Controller) oraz w jaki sposób opanować problemy związane z programowaniem współbieżnym. Ponadto nauczysz się panować nad wskaźnikami i unikać typowych problemów związanych z dostępem do pamięci. Książka ta jest doskonałą pozycją dla wszystkich programistów C++, którzy chcą wzbogacić swój warsztat o bibliotekę Qt.

Dzięki tej książce:

- poznasz podstawy C++ oraz Qt
- opanujesz programowanie współbieżne
- zrozumiesz wzorzec MVC
- zbudujesz interesującą aplikację przy użyciu C++ i Qt

helion.pl
księgarnia
internetowa

Nr katalogowy: 18259



Księgarnia internetowa:
<http://helion.pl>



Zamówienia telefoniczne:
0 801 339900



0 601 339900

**PRENTICE
HALL**



Helion

Sprawdź najnowsze promocje:
• <http://helion.pl/promocje>
Książki najchętniej czytane:
• <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
• <http://helion.pl/nowosci>

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

ISBN 978-83-246-8246-1



9 788324 682461

cena: 89,00 zł

Informatyka w najlepszym wydaniu