

## IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

## KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

## TWÓJ KOSZYK

DODAJ DO KOSZYKA

## CENNIK I INFORMACJE

ZAMÓW INFORMACJE  
O NOWOŚCIACH

ZAMÓW CENNIK

## CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

# Java. Wzorce projektowe

Autor: James William Cooper

Tłumaczenie: Piotr Badarycz

ISBN: 83-7197-529-5

Tytuł oryginału: [Java Design Patterns](#)

Format: B5, stron: około 400



Jest to praktyczna książka, która mówi o tym jak pisać programy w języku Java z użyciem standardowych wzorców projektowych. Książka składa się z serii krótkich rozdziałów, z których każdy opisuje jeden wzorec i zawiera przynajmniej jeden wizualny, kompletny i działający przykładowy program. Każdy rozdział zawiera również diagramy UML pokazujące zależności występujące pomiędzy klasami programu. Podczas lektury niniejszej książki czytelnik dowie się, że wzorce projektowe są powszechnie stosowanym sposobem organizacji obiektów w programach. Dzięki wykorzystaniu wzorców łatwiej jest pisać programy i później je modyfikować. Zapoznanie się z wzorcami pozwoli zdobyć słownictwo i zestaw pojęć, za pomocą których można łatwiej opisać konstrukcję swoich programów.



# Spis treści

	<b>Przedmowa .....</b>	<b>9</b>
	<b>Podziękowania .....</b>	<b>11</b>
<b>Część I</b>	<b>Czym są wzorce projektowe? .....</b>	<b>13</b>
<b>Rozdział 1.</b>	<b>Wprowadzenie .....</b>	<b>15</b>
	Definicja wzorca projektowego.....	17
	Proces uczenia się.....	18
	Studiowanie wzorców projektowych .....	19
	Uwagi na temat projektowania zorientowanego obiektowo .....	19
	Klasy JFC .....	20
	Wzorce projektowe w języku Java.....	20
<b>Rozdział 2.</b>	<b>Diagramy UML .....</b>	<b>21</b>
	Dziedziczenie .....	22
	Interfejsy .....	23
	Kompozycja.....	23
	Adnotacje.....	24
	JVISION.....	24
	Visual SlickEdit.....	24
<b>Część II</b>	<b>Wzorce konstrukcyjne.....</b>	<b>25</b>
<b>Rozdział 3.</b>	<b>Factory (fabryka) .....</b>	<b>27</b>
	Jak działa fabryka .....	27
	Przykładowy kod .....	28
	Teraz dwie klasy pochodne .....	28
	Tworzenie Simple Factory .....	29
	Wzorzec Factory w obliczeniach matematycznych .....	30
	Zagadnienia do przemyślenia .....	31
<b>Rozdział 4.</b>	<b>Factory Method (metoda fabrykująca) .....</b>	<b>33</b>
	Klasa Swimmer .....	35
	Klasa Event.....	35
	Rozstawienie bezpośrednie .....	36
	Program rozstawiający .....	38
	Inne fabryki .....	38
	Kiedy używać Factory Method .....	38
	Zagadnienia do przemyślenia .....	39

<b>Rozdział 5.</b>	<b>Abstract Factory (fabryka abstrakcji).....</b>	<b>41</b>
	Abstract Factory w projektowaniu ogrodów .....	42
	Jak działa interfejs użytkownika .....	43
	Dodawanie nowych klas .....	44
	Konsekwencje stosowania wzorca Abstract Factory .....	45
	Zagadnienia do przemyślenia .....	45
<b>Rozdział 6.</b>	<b>Singleton .....</b>	<b>47</b>
	Tworzymy Singleton używając metody statycznej .....	47
	Wyjątek .....	48
	Zgłaszanie wyjątku .....	48
	Tworzenie instancji klasy .....	49
	Dostarczenie globalnego punktu dostępu dla wzorca Singleton .....	49
	Pakiet javax.comm jako przykład użycia wzorca Singleton .....	50
	Konsekwencje stosowania wzorca Singleton .....	53
	Zagadnienia do przemyślenia .....	53
<b>Rozdział 7.</b>	<b>Builder (budowniczy) .....</b>	<b>55</b>
	Program do śledzenia inwestycji .....	56
	Nazywanie budowniczych .....	57
	Budowniczy listy wyboru .....	59
	Budowniczy pól wyboru .....	60
	Konsekwencje stosowania wzorca Builder .....	61
	Zagadnienia do przemyślenia .....	61
<b>Rozdział 8.</b>	<b>Prototype (prototyp).....</b>	<b>63</b>
	Klonowanie obiektów w języku Java .....	64
	Używanie prototypu .....	64
	Stosowanie wzorca Prototypu .....	66
	Menedżer prototypów .....	69
	Klonowanie z wykorzystaniem serializacji .....	69
	Konsekwencje stosowania wzorca Prototypu .....	70
	Zagadnienia do przemyślenia .....	71
	Podsumowanie wzorców konstrukcyjnych .....	71
<b>Część III</b>	<b>Wzorce strukturalne.....</b>	<b>73</b>
<b>Rozdział 9.</b>	<b>Adapter .....</b>	<b>75</b>
	Przenoszenie danych pomiędzy listami .....	75
	Korzystanie z klasy JList z biblioteki JFC .....	76
	Adaptory uniwersalne .....	81
	Adaptory dynamiczne .....	81
	Adaptory w języku Java .....	82
	Zagadnienia do przemyślenia .....	83
<b>Rozdział 10.</b>	<b>Bridge (most) .....</b>	<b>85</b>
	Diagram klas .....	87
	Rozbudowa mostu .....	87
	Java Bean jako przykład wzorca Bridge .....	89
	Konsekwencje stosowania wzorca Bridge .....	89
	Zagadnienia do przemyślenia .....	90
<b>Rozdział 11.</b>	<b>Composite (kompozyt) .....</b>	<b>91</b>
	Implementacja kompozytu .....	92
	Obliczanie wynagrodzeń .....	92

	Klasa Employee .....	93
	Klasa Boss .....	94
	Tworzenie drzewa pracowników .....	96
	Awans .....	97
	Lista dwukierunkowa .....	97
	Konsekwencje stosowania wzorca Composite .....	98
	Prosty Composite .....	98
	Kompozyty w Javie .....	99
	Inne kwestie dotyczące implementacji .....	99
	Zagadnienia do przemyślenia .....	99
<b>Rozdział 12.</b>	<b>Decorator (dekorator) .....</b>	<b>101</b>
	Dekorowanie przycisku .....	101
	Użycie dekoratora .....	103
	Diagram klas .....	104
	Dekorowanie obwódek w Javie .....	105
	Dekoratory niewizualne .....	106
	Dekoratory, adaptory i kompozyty .....	108
	Konsekwencje stosowania wzorca Dekoratora .....	108
	Zagadnienia do przemyślenia .....	109
<b>Rozdział 13.</b>	<b>Facade (fasada) .....</b>	<b>111</b>
	Tworzenie klas fasady .....	112
	Konsekwencje stosowania wzorca Facade .....	115
	Uwagi dotyczące instalacji i uruchamiania programu dbFrame .....	115
	Zagadnienia do przemyślenia .....	116
<b>Rozdział 14.</b>	<b>Flyweight (waga piórkowa) .....</b>	<b>117</b>
	Omówienie .....	118
	Przykładowy kod .....	118
	Wzorce Flyweight w Javie .....	122
	Współdzielone obiekty .....	122
	Obiekty „kopiowane podczas zapisu” .....	123
	Zagadnienia do przemyślenia .....	123
<b>Rozdział 15.</b>	<b>Proxy (pośrednik) .....</b>	<b>125</b>
	Przykładowy kod .....	126
	Kopiowanie podczas zapisu .....	128
	Enterprise Java Beans .....	128
	Porównanie z innymi wzorcami .....	128
	Zagadnienia do przemyślenia .....	128
	Podsumowanie wzorców strukturalnych .....	128
<b>Część IV</b>	<b>Wzorce czynnościowe .....</b>	<b>131</b>
<b>Rozdział 16.</b>	<b>Chain of Responsibility (łańcuch odpowiedzialności) .....</b>	<b>133</b>
	Zastosowania .....	134
	Przykładowy kod .....	134
	Wizualne komponenty pola listy .....	137
	Implementacja systemu pomocy .....	139
	Łańcuch czy drzewo? .....	142
	Rodzaje żądań .....	143
	Przykłady w Javie .....	143
	Konsekwencje stosowania wzorca Chain of Responsibility .....	143
	Zagadnienia do przemyślenia .....	144

<b>Rozdział 17. Command (polecenie)</b>	<b>145</b>
Motywacja	145
Obiekt polecenia	146
Używanie obiektów polecenia	147
Wzorzec Command	148
Wzorzec Command w języku Java	150
Konsekwencje stosowania wzorca Command	151
Wycofywanie operacji	152
Zagadnienia do przemyślenia	155
<b>Rozdział 18. Interpreter</b>	<b>157</b>
Motywacja	157
Zastosowania	157
Prosty przykład raportowania	158
Interpretowanie języka	159
Obiekty używane podczas parsowania	160
Redukowanie parsowanego stosu	162
Implementowanie wzorca Interpretera	163
Konsekwencje stosowania wzorca Interpretera	166
Zagadnienia do przemyślenia	167
<b>Rozdział 19. Iterator</b>	<b>169</b>
Motywacja	169
Wyliczenia w Javie	170
Przykładowy kod	170
Iteratory filtrowane	171
Konsekwencje stosowania wzorca Iteratora	173
Iteratory i kompozyty	174
Iteratory w Javie 1.2	174
Zagadnienia do przemyślenia	174
<b>Rozdział 20. Mediator</b>	<b>175</b>
Przykładowy system	175
Interakcje pomiędzy komponentami	176
Przykładowy kod	177
Mediatorzy i obiekty poleceń	180
Konsekwencje stosowania wzorca Mediatora	180
Mediator z pojedynczym interfejsem	181
Kwestie implementacyjne	181
<b>Rozdział 21. Memento</b>	<b>183</b>
Motywacja	183
Implementacja	184
Przykładowy kod	184
Konsekwencje stosowania wzorca Memento	188
Zagadnienia do przemyślenia	188
<b>Rozdział 22. Observer (obserwator)</b>	<b>189</b>
Obserwowanie zmian kolorów	190
Inne rodzaje komunikatów	193
Klasa JList jako obserwator	193
Architektura model-widok-kontroler jako wzorzec Observer	194
Interfejs Observer i klasa Observable	195
Konsekwencje stosowania wzorca Observer	195
Zagadnienia do przemyślenia	196

<b>Rozdział 23. State (stan)</b> .....	<b>197</b>
Przykładowy kod .....	197
Przełączanie pomiędzy stanami .....	201
Interakcje mediatora z klasą StateManager .....	202
Przejścia pomiędzy stanami .....	204
Mediator — „klasa Bóg” .....	204
Konsekwencje stosowania wzorca State .....	204
Zagadnienia do przemyślenia .....	205
<b>Rozdział 24. Strategy (strategia)</b> .....	<b>207</b>
Motywacja .....	207
Przykładowy kod .....	208
Klasa Context .....	209
Polecenia programu .....	210
Strategia dla wykresu liniowego i wykresu słupkowego .....	210
Rysowanie wykresów w Javie .....	211
Konsekwencje stosowania wzorca Strategy .....	213
Zagadnienia do przemyślenia .....	214
<b>Rozdział 25. Template (szablon)</b> .....	<b>215</b>
Motywacja .....	215
Rodzaje metod w klasach szablonych .....	216
Wzorce metod szablonych w Javie .....	217
Przykładowy kod .....	217
Szablony i wywołania zwrotne .....	221
Konsekwencje stosowania wzorca Template .....	222
Zagadnienia do przemyślenia .....	222
<b>Rozdział 26. Visitor (wizytator)</b> .....	<b>223</b>
Motywacja .....	223
Kiedy używać wzorca Visitor .....	224
Przykładowy kod .....	225
Wizytowanie klas .....	226
Wizytowanie wielu klas .....	227
Kierownicy są również pracownikami .....	228
Operacje wizytatora wyłapujące wszystkie klasy .....	229
Podwójne wywołania .....	230
Przemierzanie wielu klas .....	230
Konsekwencje stosowania wzorca Visitor .....	230
Zagadnienia do przemyślenia .....	231
<b>Część V Wzorce projektowe i Java Foundation Classes</b> .....	<b>233</b>
<b>Rozdział 27. JFC czyli Swing</b> .....	<b>235</b>
Instalacja i korzystanie z klas Swing .....	235
Koncepcje, na których oparta jest biblioteka Swing .....	236
Hierarchia klas Swing .....	236
<b>Rozdział 28. Pisanie prostego programu z wykorzystaniem JFC</b> .....	<b>237</b>
Ustawianie definicji wyglądu i zachowania .....	237
Obsługa zdarzenia zamknięcia okna .....	238
Klasa JFrame .....	238
Prosty program z dwoma przyciskami .....	239
Więcej o klasie JButton .....	240

<b>Rozdział 29. Przyciski radiowe i paski narzędziowe .....</b>	<b>241</b>
Przyciski radiowe .....	241
Klasa JToolBar .....	242
JToggleButton .....	242
Przykładowy program z różnymi przyciskami.....	243
<b>Rozdział 30. Komponenty menu i obiekty akcji .....</b>	<b>245</b>
Obiekty akcji .....	245
Wzorce projektowe i obiekty akcji.....	248
<b>Rozdział 31. Klasa JList .....</b>	<b>249</b>
Zaznaczanie elementów listy i zdarzenia .....	250
Dynamiczna zmiana wyświetlanej zawartości listy .....	250
Posortowana lista wykorzystująca komponent JList i obiekt ListModel .....	252
Sortowanie bardziej skomplikowanych obiektów.....	253
Otrzymywanie klucza bazy danych.....	255
Dodawanie ikon do komponentu JList .....	256
<b>Rozdział 32. Klasa JTable .....</b>	<b>259</b>
Prosty program z komponentem JTable .....	259
Interpretatory wyglądu komórek .....	262
Interpretacja innych klas.....	263
Zaznaczanie komórek tabeli .....	265
Wzorce użyte w powyższym przykładzie .....	266
<b>Rozdział 33. Klasa JTree .....</b>	<b>269</b>
Interfejs TreeModel .....	270
Podsumowanie.....	271
<b>Część VI Studium przypadków .....</b>	<b>273</b>
<b>Rozdział 34. Sandy i mediator .....</b>	<b>275</b>
<b>Rozdział 35. Problemy Herba z przetwarzaniem tekstu .....</b>	<b>279</b>
<b>Rozdział 36. Dylemat Mary .....</b>	<b>281</b>
<b>Dodatki .....</b>	<b>283</b>
<b>Bibliografia .....</b>	<b>285</b>
<b>Skorowidz .....</b>	<b>287</b>

## Rozdział 13.

# Facade (fasada)

W niniejszym rozdziale będziemy zajmować się wzorcem *Facade*. Wzorec ten jest używany do obudowywania zbioru złożonych klas i dostarcza dla nich prostszego interfejsu.

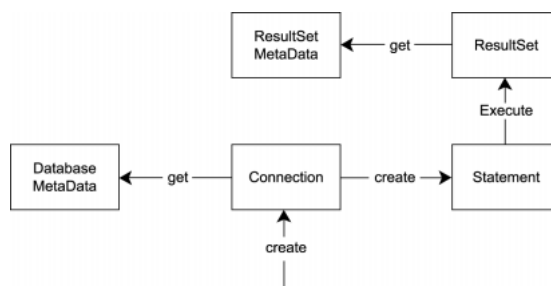
Często program podczas tworzenia ewoluuje i rośnie stopień jego komplikacji. Zachwycając się korzyściami płynącymi ze stosowania wzorców projektowych, zauważamy też ich ujemną cechę: czasami generują one bardzo wiele dodatkowych klas, przez co trudniej jest zrozumieć działanie programu. Poza tym programy często składają się z szeregu podsystemów, z których każdy posiada swój własny skomplikowany interfejs.

*Fasada* pozwala uprościć tę złożoność dostarczając uproszczonego interfejsu do tych podsystemów. Takie uproszczenie może czasami zmniejszyć elastyczność przykrywanym klas, lecz często dostarcza wszystkich funkcji niezbędnych każdemu użytkownikowi. Oczywiście przykrywane klasy i ich metody mogą być w dalszym ciągu dostępne.

Na szczęście, aby przedstawić przykład zastosowania wzorca *Facade*, nie będziemy musieli tworzyć złożonego systemu. Java dostarcza zbioru klas, które pozwalają na łączenie się z bazami danych poprzez interfejs zwany JDBC. Można połączyć się z każdą bazą danych, dla której producent dostarczył sterownik JDBC (odpowiedni zestaw klas), czyli z prawie każdą bazą danych dostępną na rynku. Niektóre bazy danych pozwalają na połączenie bezpośrednie, do innych dostęp jest zapewniony poprzez klasę mostu JDBC-ODBC.

Klasy obsługujące bazy danych z pakietu *java.sql* stanowią doskonały przykład niskopoziomowych klas, które komunikują się ze sobą w bardzo zawiły sposób (rysunek 13.1).

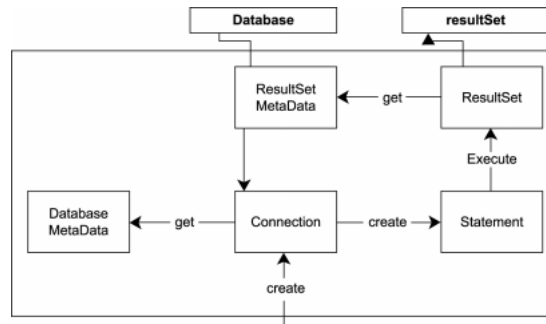
**Rysunek 13.1.**  
Zarys interakcji  
pomiędzy klasami  
pakietu *java.sql*\*,  
używanych do połączeń  
z bazą danych



Do połączenia z bazą danych wykorzystywana jest instancja klasy `Connection`. Do określania nazw tabel bazy danych i ich pól korzysta się z klasy `DatabaseMetadata` poprzez klasę `Connection`. Do konstruowania zapytań w języku SQL, czyli obiektu typu `String`, wykorzystuje się klasę `Statement`. Poprzez wykonanie zapytania (klasa `Statement`) otrzymuje się wynik — obiekt klasy `ResultSet`. Do określenia nazw kolumn trzeba jeszcze uzyskać instancję klasy `ResultSetMetadata`. Operowanie wszystkimi tymi klasami może być bardzo trudne, większość wywołań może zwracać wyjątki, przez co kod jest bardzo zagmatwany.

Poprzez zbudowanie *fasady* składającej się z klasy `Database` i klasy `Results`, możemy zbudować łatwiejszy w użyciu system.

**Rysunek 13.2.**  
Fasada zakrywająca  
wiele klas pakietu  
`java.sql`.\*



## Tworzenie klas fasady

Przyjrzyjmy się, jak nawiązuje się połączenie z bazą danych. Najpierw trzeba załadować sterownik bazy danych.

```

try {
    Class.forName(driver); // load the Bridge driver
}
catch (Exception e) {
    System.out.println(e.getMessage());
}

```

Następnie używamy klasy `Connection` do łączenia się z bazą danych. Pobieramy również metadane, aby dowiedzieć się więcej o bazie danych.

```

try {
    con = DriverManager.getConnection(url);
    dma = con.getMetaData(); //get the meta data
} catch (Exception e) {
    System.out.println(e.getMessage());
}

```

Jeśli chcemy sporządzić listę wszystkich nazw tabel bazy danych, musimy wywołać metodę `getTables` klasy `Metadata`, która zwróci obiekt `ResultSet`. Aby uzyskać listę nazw musimy przejść przez wszystkie elementy tego obiektu, i wyłuskać tylko tabele użytkownika, odrzucając tabele systemowe.

```

Vector tname = new Vector();
//add the table names to a Vector
//since we don't know how many there are
try {
    results = new Results(dma.getTables(catalog, null, "%", types));
} catch (Exception e) {
    System.out.println(e);
}

while (results.hasMoreElements())
    tname.addElement(results.getColumnValue("TABLE_NAME"));

```

Jak widać, już teraz bardzo trudno jest tym wszystkim zarządzać, a nie wykonaliśmy nawet jeszcze żadnego zapytania.

Możemy przyjąć pewne założenie upraszczające: wyjątki, które są zgłaszane przez metody klas nie wymagają skomplikowanej obsługi. W przeważającej części metody będą pracowały bezbłędnie, dopóki prawidłowo będzie działać połączenie sieciowe z serwerem bazy danych. Więc możemy obudować wszystkie te metody, tak by błędy, które wystąpią były wypisywane bez podejmowania żadnych dodatkowych akcji.

Możliwe jest teraz napisanie prostych klas zawierających wszystkie ważne metody klas `Connection`, `ResultSet`, `Statement` i `MetaData`. Tak będą wyglądały metody klasy `Database`:

```

class Database {
    public Database(String driver); // constructor
    public void      Open(String url, String cat);
    public String[]  getTableNames();
    public String[]  getColumnNames(String table);
    public String    getColumnValue(String table,
                                       String columnName);
    public String    getNextValue(String columnName);
    public ResultSet Execute(String sql);
}

```

A tak, metody klasy `Results`.

```

class Results {
    public Results(ResultSet rset); // constructor
    public String[]  getMetaData();
    public boolean   hasMoreElements();
    public String[]  nextElement();
    public String    getColumnValue(String columnName);
    public String    getColumnValue(int i)
}

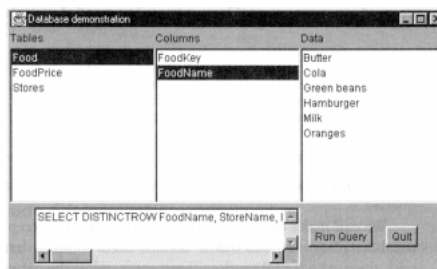
```

Te proste klasy pozwalają nam napisać program otwierający połączenie z bazą danych i wyświetlający nazwy jej tabel, kolumn i zawartość. Program umożliwia również wykonanie prostego zapytania SQL.

Nasz przykładowy program wykorzystujący *fasadę* daje dostęp do bazy danych zawierającej ceny żywności w trzech okolicznych supermarketach.

**Rysunek 13.3.**

Program *dbFrame* pokazuje dostęp do nazw tabel, kolumn i ich zawartości



Kliknięcie nazwy tabeli wyświetli nazwy kolumn tej tabeli, a kliknięcie nazwy kolumny wyświetli zawartość tej kolumny. Naciśnięcie przycisku „Run Query” wyświetli posortowane ceny pomarańczy we wszystkich supermarketach (rysunek 13.4).

**Rysunek 13.4.**

Wynik zapytania wykonanego przez program *dbFrame*

FoodName	StoreName	Price
Oranges	Village Market	0.2900
Oranges	Stop and Shop	0.3600
Oranges	Waldbaum's	0.4700

Po uruchomieniu program łączy się z bazą danych i pobiera listę tabel.

```
db = new Database("sun.jdbc.odbc.JdbcOdbcDriver");
db.Open("jdbc:odbc:Grocery prices". null);
String tnames[] = db.getTableNames();
loadList(Tables, tnames);
```

Kliknięcie w obszarze listy powoduje wykonanie zapytania dla nazw kolumn lub wartości.

```
public void itemStateChanged(ItemEvent e) {
    Object obj = e.getSource();
    if (obj == Tables)
        showColumns();
    if (obj == Columns)
        showData();
}
//-----
private void showColumns() {
    String cnames[] = db.getColumnNames(Tables.getSelectedItem());
    loadList(Columns, cnames);
}
//-----
private void showData() {
    String colname = Columns.getSelectedItem();
    String colval = db.getColumnValue(Tables.getSelectedItem(), colname);

    Data.setVisible(false);
    Data.removeAll();
    Data.setVisible(true);
}
```

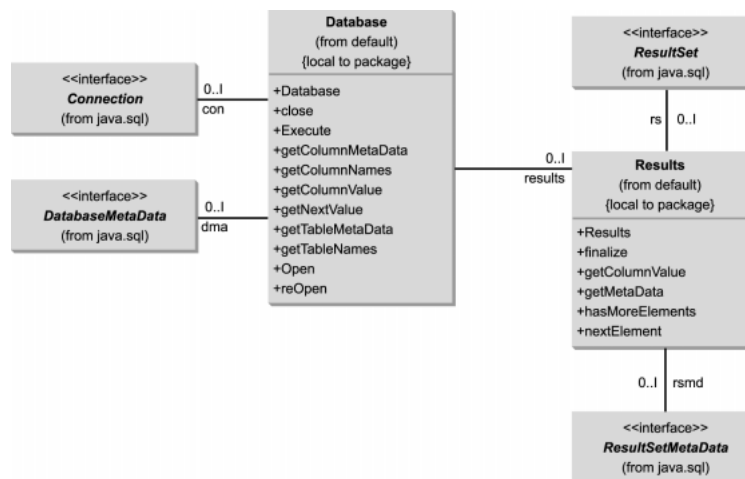
```

colval = db.getNextValue(Columns.getSelectedItem());
while (colval.length()>0) {
    Data.add(colval);
    wartosc = db.getNextValue(Columns.getSelectedItem());
}
}

```

Diagram przedstawia kompletną prezentację klas *fasady*.

**Rysunek 13.5.**  
Klasy *fasady*,  
z których może  
korzystać użytkownik



Zauważmy, że klasa `Database` zawiera instancję klasy `Connection`, `DatabaseMetaData` i `Results`. Z kolei klasa `Results` zawiera instancje klas `ResultSet` i `ResultSetMetaData`.

## Konsekwencje stosowania wzorca Facade

Wzorec *Facade* izoluje klienta od skomplikowanych komponentów podsystemów i dostarcza do nich prostszy interfejs do ogólnego użytku. Jednak nie ogranicza zaawansowanemu użytkownikowi dostępu do złożonych klas znajdujących się głębiej.

Dodatkowo *Facade* umożliwia dokonywanie zmian w przykrywanych podsystemach bez potrzeby modyfikacji kodu klienta i redukuje liczbę zależności podczas kompilacji.

## Uwagi dotyczące instalacji i uruchamiania programu dbFrame

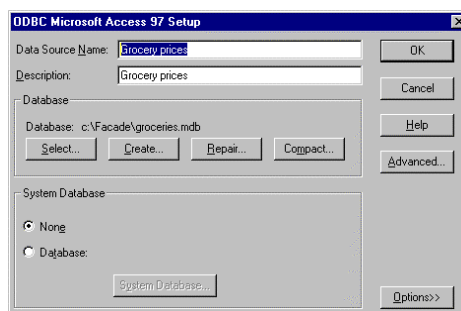
Aby umożliwić komunikację z bazą danych, należy zainstalować sterowniki ODBC i JDBC-ODBC dla Javy. Ten przykładowy program będzie działał tylko w środowisku Windows, ponieważ wymaga bazy danych Microsoft Access — pliku *groceries.mdb*. Sterownik JDBC-ODBC jest wbudowany w Javę 2 (wersja 1.2 lub wyższa). Gdy używa się wersji wcześniejszych, należy zaopatrzyć się w ten sterownik. Jest dostępny na stronie WWW: [java.sun.com](http://java.sun.com).

Sterownik ODBC Data Access jest dostępny na stronach firmy Microsoft. Po zainstalowaniu pojawi się folder ODBC w panelu sterowania.

Program z foldera *Facade*, który znajduje się na serwerze ftp wydawnictwa Helion (<ftp://ftp.helion.pl/przyklady/javawz.zip>), należy przekopiować na twardy dysk. Następnie należy uruchomić program ODBC z panelu sterowania i zarejestrować plik *groceries.mdb*, nacisnąć przycisk „Add” i wypełnić panel jak pokazano to na rysunku 13.6. Aby ustawić lokalizację pliku *groceries.mdb* należy nacisnąć „Select” i wskazać na plik w katalogu, do którego został wkopiowany.

### Rysunek 13.6.

*Sposób skonfigurowania dostępu ODBC/JDBC do przykładowej bazy danych*



Jeśli źródło danych zostanie nazwane inaczej niż „Grocery Prices”, trzeba będzie zmienić 20. linię w programie *dbFrame.java*.

## Zagadnienia do przemyślenia

1. Przypuśćmy, że mamy napisany program z polem menu *File/Open* i przyciskami pozwalającymi kontrolować rodzaj czcionki (pogrubiona i kursywa). Teraz przypuśćmy, że potrzebujemy, aby program dało się uruchamiać z linii komend przez podanie argumentów. Proszę zastanowić się, jak można wykorzystać wzorzec *Facade*, aby to osiągnąć.