

Spis treści (skrótowy)

1	Dobrze zaprojektowane aplikacje są super. <i>Tu zaczyna się wspaniałe oprogramowanie</i>	31
2	Gromadzenie wymagań. <i>Daj im to, czego chcą</i>	83
3	Wymagania ulegają zmianom. <i>Kocham cię, jesteś doskonały... A teraz — zmień się</i>	137
4	Analiza. <i>Zaczynamy używać naszych aplikacji w rzeczywistym świecie</i>	169
5	Część 1. Dobry projekt = elastyczne oprogramowanie. <i>Nic nie pozostaje wечно takie samo</i>	221
	Przerwywnik. Obiektowa katastrofa	245
	Część 2. Dobry projekt = elastyczne oprogramowanie. <i>Zabierz swoje oprogramowanie na 30-minutowy trening</i>	257
6	Rozwiązywanie naprawdę dużych problemów. <i>„Nazywam się Art Vandelay... jestem Architektem”</i>	301
7	Architektura. <i>Porządkowanie chaosu</i>	343
8	Zasady projektowania. <i>Oryginalność jest przereklamowana</i>	395
9	Powtarzanie i testowanie. <i>Oprogramowanie jest wciąż przeznaczone dla klienta</i>	441
10	Proces projektowania i analizy obiektowej. <i>Scalając to wszystko w jedno</i>	499
A	Pozostałości	571
B	Witamy w Obiektowie	589
	Skorowidz	603

Spis treści (z prawdziwego zdarzenia)



Wprowadzenie

Twój mózg koncentruje się na analizie i projektowaniu obiektowym.

Podczas gdy Ty starasz się czegoś **nauczyć**, Twój mózg robi Ci przysługę i dba o to, abyś przez przypadek **nie zapamiętał** zdobywanych informacji. Myśli sobie: „Lepiej zostawić trochę miejsca na bardziej istotne sprawy, na przykład jakich zwierząt unikać albo czy jazda na snowboardzie nago jest dobrym pomysłem”. W jaki zatem sposób możesz oszukać swój mózg i przekonać go, że Twoje życie zależy od znajomości analizy i projektowania obiektowego?

Dla kogo jest ta książka?	20
Wiemy, co sobie myślisz	21
Metapoznanie: myślenie o myśleniu	23
Zmusz swój mózg do posłuszeństwa	25
Ważne uwagi	26
Zespół techniczny	28
Podziękowania	29

Dobrze zaprojektowane aplikacje są super

1

Tu zaczyna się wspaniałe oprogramowanie

A zatem, w jaki sposób w praktyce pisze się wspaniałe oprogramowanie?

Zawsze bardzo trudno jest określić, od czego należy zacząć. Czy aplikacja faktycznie robi to, co powinna robić i czego od niej oczekujemy? A co z takimi problemami jak powtarzający się kod — przecież to nie może być dobre ani właściwe rozwiązanie, prawda? Zazwyczaj trudno jest określić, które z wielu problemów należy rozwiązać w pierwszej kolejności, a jednocześnie mieć pewność, że podczas wprowadzania poprawek nie popuszymy innych fragmentów aplikacji. Bez obaw. Po zakończeniu lektury tego rozdziału będziesz już dokładnie **wiedział, jak pisać doskonałe oprogramowanie**, i pewnie podążał w kierunku trwałego poprawienia sposobu tworzenia programów. I w końcu zrozumiesz, dlaczego OOA&D to czteroliterowy skrót (pochodzący od angielskich słów: **Object-Oriented Analysis and Design**, analiza i projektowanie obiektowe), który Twoja matka **chciałaby**, byś poznał.

Rock-and-roll jest wieczny!	32
Nowa elegancka aplikacja Ryśka...	33
Co przede wszystkim zmieniłbyś w aplikacji Ryśka?	38
Dośkonałe oprogramowanie...	
ma więcej niż jedną z wymienionych już cech	42
Wspaniałe oprogramowanie w trzech prostych krokach	43
W pierwszej kolejności skoncentruj się na funkcjonalności	48
Test	53
Szukamy problemów	55
Analiza metody search()	56
Stosuj proste zasady projektowania obiektowego	61
Projekt po raz pierwszy, projekt po raz drugi	66
Jak łatwo można wprowadzać zmiany w Twojej aplikacji?	68
Poddawaj hermetyzacji to, co się zmienia	71
Delegowanie	73
Nareszcie doskonałe oprogramowanie (jak na razie)	76
OOA&D ma na celu tworzenie wspaniałego oprogramowania, a nie dodanie Ci papierkowej roboty	79
Celne spostrzeżenia	80

Niby skąd mam wiedzieć, od czego należy zacząć? Mam wrażenie, że ilekroć zaczynam pracę nad nowym projektem, każdy ma inne zdanie odcień tego, co należy zrobić w pierwszej kolejności. Czasami zrobię coś dobre, lecz czasami kofczy się na tym, że muszę przerobić całą aplikację od początku, bo zacząłem w złym miejscu. A ja chcę jedynie pisać świetne oprogramowanie! A zatem, od czego powinienem zacząć pisanie aplikacji dla Ryśka?

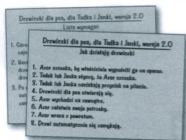


Gromadzenie wymagań

2 Daj im to, czego chcą

Każdy lubi zadowolonych klientów. Już wiesz, że pierwszy krok w pisaniu doskonałego oprogramowania polega na upewnieniu się, czego chce klient. Ale jak się dowiedzieć, **czego klient oczekuje?** Co więcej — skąd mieć pewność, że klient w ogóle **wie**, czego tak naprawdę chce? Właśnie wówczas na arenę wkraczają „**dobre wymagania**”. W tym rozdziale dowiesz się, w jaki sposób **zadowolić klientów**, upewniając się, że dostarczysz im właśnie to, czego chcą. Kiedy skończysz lekturę, wszystkie swoje projekty będziesz mógł opatrzyć etykietą „Satisfakcja gwarantowana” i posuniesz się o kolejny krok na drodze do tworzenia doskonałego oprogramowania... i to za każdym razem.

Nadszedł czas na kolejny pokaz	84
Twój programistyczny umiętności	84
Test programu	87
Nieprawidłowe zastosowanie (coś w tym stylu)	89
Czym jest wymaganie?	90
Tworzenie listy wymagań	92
Zaplanuj, co może się popsuć w systemie	96
Problemy w działaniu systemu są obsługiwane przez ścieżki alternatywne	98
(Ponowne) przedstawienie przypadku użycia	100
Jeden przypadek użycia, trzy części	102
Porównaj wymagania z przypadkami użycia	106
Twój system musi działać w praktyce	113
Poznajemy Szczęśliwą Ścieżkę	120
Przybornik projektanta	134



Drzwiczki dla psa oraz pilot stanowią elementy systemu, bądź też znajdują się wewnątrz niego.

Wymagania ulegają zmianom

3

Kocham cię, jesteś doskonały... A teraz — zmień się

Sądziś, że dowiedziałeś się już wszystkiego o tym, czego chciał klient? Nie tak szybko.. A zatem przeprowadziłeś rozmowy z klientem, zgromadziłeś wymagania, napisałeś przypadki użycia, napisałeś i dostarczyłeś klientowi odlotową aplikację. W końcu nadszedł czas na miłą, relaksującego drinka, nieprawdaż? Pewnie... aż do momentu gdy klient uzna, że tak naprawdę chce czegoś innego niż to, co Ci powiedział. Bardzo podoba mu się to, co zrobiłeś — poważnie! — jednak obecnie **nie jest już w pełni usatysfakcjonowany**. W rzeczywistym świecie **wymagania zawsze się zmieniają**, to Ty musisz sobie z tymi zmianami poradzić i pomimo nich zadbać o zadowolenie klienta.

Jesteś bohaterem!	138
Jesteś patalachem!	139
Jedyny pewnik analizy i projektowania obiektowego	141
Ścieżka oryginalna? Ścieżka alternatywna? Kto to wie?	146
Przypadki użycia muszą być zrozumiałe przede wszystkim dla Ciebie	148
Od startu do mety: jeden scenariusz	150
Wyznanie Ścieżki Alternatywnej	152
Uzupełnienie listy wymagań	156
Powielanie kodu jest bardzo złym pomysłem	164
Ostateczny test drzwiczek	166
Napisz swoją własną zasadę projektową!	167
Przybornik projektanta	168

```
public void pressButton() {
    System.out.println("Naciśnięto przycisk na pilocie...");
    if (door.isOpen()) {
        door.close();
    } else {
        door.open();
    }

    final Timer timer = new Timer();
    timer.schedule(new TimerTask() {
        public void run() {
            door.close();
            timer.cancel();
        }
    }, 5000);
}
```



Remote.java

Analiza

4

Zaczynamy używać naszych aplikacji
w rzeczywistym świecie

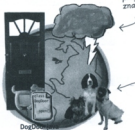
Czas zdać ostatnie egzaminy i zacząć stosować nasze aplikacje w rzeczywistym świecie. Twoje aplikacje muszą robić nieco więcej, niż jedynie działać prawidłowo na komputerze, którego używasz do ich tworzenia — komputerze o dużej mocy i doskonale skonfigurowanym; Twoje aplikacje muszą działać w takich warunkach, w jakich **rzeczywiści klienci będą ich używali**. W tym rozdziale zastanowimy się, jak zyskać pewność, że nasze aplikacje będą działać w **rzeczywistym kontekście**. Dowiesz się w nim, w jaki sposób analiza tekstowa może przekształcić stworzony wcześniej przypadek użycia w klasy i metody, które na pewno będą działać zgodnie z oczekiwaniami klienta. A kiedy skończysz lekturę tego rozdziału, także i Ty będziesz mógł powiedzieć: „Dokonałem tego! Moje oprogramowanie **jest gotowe do zastosowania w rzeczywistym świecie!**”.

Kiedy już określiłam, jakich klas i operacji będę potrzebować, odpowiednio zaktualizowałam diagram klas.

Jeden pies, dwa psy, trzy psy, cztery...	170
Twoje oprogramowanie ma kontekst	171
Określ przyczynę problemu	172
Zaplanuj rozwiązanie	173
Opowieść o dwóch programistach	180
Delegowanie w kodzie Szymka — analiza szczegółowa	184
Potęga aplikacji, których elementy są ze sobą luźno powiązane	186
Zwracaj uwagę na rzeczowniki występujące w przypadku użycia	191
Od dobrej analizy do dobrych klas...	204
Diagramy klas bez tajemnic	206
Diagramy klas to nie wszystko	211
Celne spostrzeżenia	215



W tym kontekście rzeczy przybierają zły obrót znacznie częściej.



W rzeczywistym świecie spotykamy inne psy, koty, gryzonie oraz całą masę innych problemów; a wszystkie te czynniki mają tylko jeden cel — doprowadzić do awarii naszego oprogramowania.

Rzeczywisty Świat

Dobry projekt = elastyczne oprogramowanie

5

(część 1.)

Nic nie pozostaje wiecznie takie samo

Zmiany są nieuniknione. Niezależnie od tego, jak bardzo podoba Ci się Twoje oprogramowanie w jego obecnej postaci, to najprawdopodobniej jutro zostanie ono **zmodyfikowane**. A im bardziej utrudnisz wprowadzanie modyfikacji w aplikacji, tym trudniej będzie Ci w przyszłości reagować na **zmiany potrzeb klienta**. W tym rozdziale mamy zamiar odwiedzić naszego starego znajomego oraz spróbować poprawić projekt istniejącego oprogramowania. Na tym przykładzie przekonamy się, jak **niewielkie zmiany mogą doprowadzić do poważnych problemów**. Prawdę mówiąc, jak się okaże, odkryte przez nas kłopoty będą tak poważne, że ich rozwiązanie będzie wymagało rozdziału składającego się aż z DWÓCH części!

Firma Instrumenty Strunowe Ryśka rozwija się	222
Klasy abstrakcyjne	225
Diagramy klas bez tajemnic (ponownie)	230
Ściągawka z UML-a	231
Porady dotyczące problemów projektowych	237
Trzy kroki tworzenia wspaniałego oprogramowania (po raz kolejny)	239

5

(przerywnik)

OBIEKTOWA KATASTROFA!

Najbardziej popularny quiz w Obiektowie

Unikanie ryzyka	Stawanie projektanci	Konstrukcje używane w kodzie	Utrzymanie i wielokrotne użycie	Nierwca oprogramowania
\$100	\$100	\$100	\$100	\$100
\$200	\$200	\$200	\$200	\$200
\$300	\$300	\$300	\$300	\$300
\$400	\$400	\$400	\$400	\$400

Dobry projekt = elastyczne oprogramowanie

5

(część 2.)

Zabierz swoje oprogramowanie na 30-minutowy trening

Czy kiedykolwiek marzyłeś o tym, by być nieco bardziej elastycznym w działaniu? Jeśli kiedykolwiek wpadłeś w kłopoty podczas prób wprowadzania zmian w aplikacji, to zazwyczaj oznacza to, że Twoje oprogramowanie powinno być nieco **bardziej elastyczne i odporne**. Aby pomóc swojej aplikacji, będziesz musiał przeprowadzić odpowiednią analizę, zastanowić się nad niezbędnymi zmianami w projekcie i dowiedzieć się, w jaki sposób **rozluźnić zależności pomiędzy jej elementami**. I w końcu, w wielkim finale, przekonasz się, że **większa spójność może pomóc w rozwiązaniu problemu powiązań**. Birzmi interesująco? A zatem przewróć kartkę — przystępujemy do poprawiania nieelastycznej aplikacji.

Wróćmy do aplikacji wyszukiwawczej Ryska	258
Dokładniejsza analiza metody search()	261
Korzyści, jakie dała nam analiza	262
Dokładniejsza analiza klas instrumentów	265
Śmierć projektu (decyzja)	270
Zmieńmy złe decyzje projektowe na dobre	271
Zastosowanie „podwójnej hermetyzacji” w aplikacji Ryska	273
Nigdy nie obawiaj się wprowadzania zmian	279
Elastyczna aplikacja Ryska	282
Testowanie dobrze zaprojektowanej aplikacji Ryska	285
Jak łatwo można zmodyfikować aplikację Ryska?	289
Wielki konkurs łatwości modyfikacji	290
Spójna klasa realizuje jedną operację naprawdę dobrze	293
Przegląd zmian wprowadzanych w oprogramowaniu dla Ryska	296
Doskonałe oprogramowanie to zazwyczaj takie, które jest „wystarczająco dobre”	298
Przybornik projektanta	300

Rozwiązywanie naprawdę dużych problemów

6

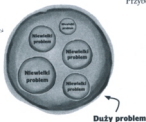
„Nazywam się Art Vandelay... jestem Architektem”

Nadszedł czas, by zbudować coś NAPRAWDĘ DUŻEGO. Czy jesteś gotów?

Zdobyłeś już wiele narzędzi do swojego projektanckiego przybornika, jednak w jaki sposób z nich skorzystasz, kiedy będziesz musiał napisać coś **naprawdę dużego**? Cóż, może jeszcze nie zdajesz sobie z tego sprawy, ale **disponujesz wszystkimi narzędziami, jakie mogą być potrzebne** do skutecznego rozwiązywania poważnych problemów. Niebawem poznasz kilka nowych narzędzi, takich jak **analiza dziedziny** oraz **diagramy przypadków użycia**, jednak nawet one bazują na wiadomościach, które już zdobyłeś, takich jak uważne słuchanie klienta oraz dokładne zrozumienie, co trzeba napisać, zanim jeszcze przystąpimy do faktycznego pisania kodu. Przygotuj się... nadszedł czas, byś sprawdził, jak sobie radzisz w roli architekta.

Rozwiązywanie dużych problemów	302
Wszystko zależy od sposobu spojrzenia na duży problem	303
Wymagania i przypadki użycia to dobry punkt wyjściowy...	308
Potrzebujemy znacznie więcej informacji	309
Określanie możliwości	312
Możliwość czy wymaganie	314
Przypadki użycia nie zawsze pomagają ujrzeć ogólny obraz tworzonego oprogramowania	316
Diagramy przypadków użycia	318
Mały aktor	323
Aktorzy to także ludzie (no dobrze... nie zawsze)	324
A zatem zabawmy się w analizę dziedziny!	329
Dziel i rządź	331
Nie zapominaj, kim tak naprawdę jest klient	335
Czym jest wzorzec projektowy?	337
Potęga OOA&D (i trochę zdrowego rozsądku)	340
Przybornik projektanta	342

W rzeczywistości **DUŻY PROBLEM** jest jedynie zbiorem mniejszych elementów funkcjonalnych, a każdy z nich reprezentuje stosujący problem.

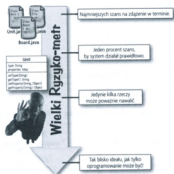


Architektura

7

Porządkowanie chaosu

Gdzieś musisz zacząć, jednak uważaj, żeby wybrać właściwe „gdzieś”! Już wiesz, jak podzielić swoją aplikację na wiele małych problemów, jednak oznacza to tylko i wyłącznie tyle, iż obecnie nie masz jednego dużego, lecz **WIELE** małych problemów. W tym rozdziale spróbujemy pomóc Ci w określeniu, **gdzie należy zacząć**, i upewnimy się, że nie będziesz marnował czasu na zajmowanie się tym, co trzeba. Nadeszła pora, by pozbierać te wszystkie **drobne kawałki** na Twoim biurku i zastanowić się, jak można je przekształcić w **uporządkowaną i dobrze zaprojektowaną aplikację**. W tym czasie poznasz niesłychanie ważne „trzy P dotyczące architektury” i dowiesz się, że Risk to znacznie więcej niż jedynie słynna gra wojenna z lat 80.



Czy czujesz się nieco przytłoczony?	344
Potrzebujemy architektury	346
Zacznijmy od funkcjonalności	349
Co ma znaczenie dla architektury	351
Trzy „P” dotyczące architektury	352
Wszystko sprowadza się do problemu ryzyka	358
Scenariusze pomagają zredukować ryzyko	361
Koncentruj się na jednej możliwości w danej chwili	369
Architektura jest strukturą Twojego projektu	371
Podobieństwa po raz kolejny	375
Analiza podobieństw: ścieżka do elastycznego oprogramowania	381
Co to znaczy? Zapytaj klienta	386
Zmniejszanie ryzyka pomaga pisać wspólnie oprogramowanie	391
Celne spostrzeżenia	392

Zasady projektowania

8

Originalność jest przereklamowana

Powielanie jest najlepszą formą unikania głupoty. Nic chyba nie daje większej satysfakcji niż opracowanie całkowicie nowego i oryginalnego rozwiązania problemu, który męczy nas od wielu dni... aż do czasu gdy okaże się, że ktoś **rozwiązał ten sam problem** już wcześniej, a co gorsza — zrobił to znacznie lepiej niż my. W tym rozdziale przyjrzymy się kilku **zasadom projektowania**, które udało się sformułować podczas tych wszystkich lat stosowania komputerów, i dowiemy się, w jaki sposób mogą one sprawić, że staniesz się lepszym programistą. Porzuć ambitne myśli o „zrobieniu tego lepiej” — lektura tego rozdziału udowodni Ci, jak pisać programy **sprytniej i szybciej**.



Zasada projektowania — w skrócie	396
Zasada otwarte-zamknięte	397
OCP, krok po kroku	399
Zasada nie powtarzaj się	402
Zasada DRY dotyczy obsługi jednego wymagania w jednym miejscu	404
Zasada jednej odpowiedzialności	410
Wykrywanie wielu odpowiedzialności	412
Przechodzenie od wielu do jednej odpowiedzialności	415
Zasada podstawienia Liskov	420
Studium błędnego sposobu korzystania z dziedziczenia	421
LSP ujawnia ukryte problemy związane ze strukturą dziedziczenia	422
Musi istnieć możliwość zastąpienia typu bazowego jego typem pochodnym	423
Naruszenia LSP sprawiają, że powstający kod staje się mylący	424
Deleguj funkcjonalność do innej klasy	426
Użyj kompozycji, by zebrać niezbędne zachowania z kilku innych klas	428
Agregacja — kompozycja bez nagłego zakończenia	432
Agregacja a kompozycja	433
Dziedziczenie jest jedynie jedną z możliwości	434
Celne spostrzeżenia	437
Przyborek projektanta	438

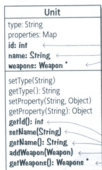
Powtarzanie i testowanie

9

Oprogramowanie jest wciąż przeznaczone dla klienta

Czas pokazać klientowi, jak bardzo Ci na nim zależy. Niekąją Cię szefowie? Klienci są zmartwieni? Udziałowcy wciąż zadają pytanie: „Czy wszystko będzie zrobione na czas?”. Żadna ilość nawet wspaniale zaprojektowanego kodu nie zadowoli Twoich klientów; musisz **pokazać im coś działającego**. Teraz, kiedy dysponujesz już solidnym przybornikiem z narzędziami do programowania obiektowego, nadszedł czas, byś **udowodnił swoim klientom**, że pisane przez Ciebie oprogramowanie naprawdę działa. W tym rozdziale poznasz dwa sposoby pracy nad implementacją możliwości funkcjonalnych tworzonego oprogramowania — dzięki nim Twoi klienci poczują się bezpiecznie, które sprawi, że powiedzą o Tobie: „**O tak, nie ma co do tego wątpliwości, jest właściwą osobą do napisania naszej aplikacji!**”.

Twój przyborek narzędziowy powoli się wypełnia	442
Wspaniale oprogramowanie tworzy się iteracyjnie	444
Schodzenie w głąb: dwie proste opcje	445
Programowanie w oparciu o możliwości	446
Programowanie w oparciu o przypadki użycia	447
Dwa podejścia do tworzenia oprogramowania	448
Analiza możliwości	452
Pisanie scenariuszy testowych	455
Programowanie w oparciu o testy	458
Podobieństwa po raz wtóry	460
Kładziemy nacisk na podobieństwa	464
Hermetyzujemy wszystko	466
Dopasuj testy do projektu	470
Testy bez tajemnic...	472
Udowodnij klientowi, że wszystko idzie dobrze	478
Jak dotąd używaliśmy programowania w oparciu o kontrakt	480
Tak naprawdę programowanie w oparciu o kontrakt dotyczy zaufania	481
Programowanie defensywne	482
Podziel swoją aplikację na mniejsze fragmenty funkcjonalności	491
Celne spostrzeżenia	493
Przyborek projektanta	496



Wskazanie właściwości, które są wpisane do wskaźnika jednokrotności, zostały przeobrażone w klasie Unit w formie odwołanych armacji.

Signum oznacza do obiektu, że identyfikator jednokrotności będzie obrotowy w konstruktorze klasy Unit, a jeżeli nie ma potrzeby definiowania odwrócić metody setAll().

Dla każdej z opcji wskaźnika klasę Signum odwołuje odpowiedni parę metod.

Pozostałości

A

**Dziesięć najważniejszych tematów
(których nie poruszyliśmy)**

Możesz nam wierzyć albo nie, ale to jeszcze nie jest koniec. Owszem, wyobraź sobie, że nawet po przeczytaniu tych 600 stron wciąż możesz jeszcze znaleźć tematy, o których nawet nie wspomnieliśmy. Choć dziesięć zagadnień, jakie mamy zamiar przedstawić w tym dodatku, nie zasługuje na wiele więcej niż krótką wzmiankę, to jednak nie chcieliśmy, byś opuszczał Obiektów bez informacji na ich temat. Teraz będziesz miał nieco więcej tematów do rozmów podczas firmowej imprezy z okazji wygrania telewizyjnego quizu Obiektowa Katastrofa... poza tym ktoś, od czasu do czasu, nie kocha stymulujących rozmów o analizie i projektowaniu?

Kiedy już skończymy, pozostanie jeszcze... następny dodatek... no i oczywiście indeks, i może kilka reklam... ale później dotrzesz wreszcie do końca książki. Obiecujemy.

Nr 1. JEST i MA	572
Nr 2. Sposoby zapisu przypadków użycia	574
Nr 3. Antywzorce	577
Nr 4. Karty CRC	578
Nr 5. Metryki	580
Nr 6. Diagramy sekwencji	581
Nr 7. Diagramy stanu	582
Nr 8. Testowania jednostkowe	584
Nr 9. Standardy kodowania i czytelny kod	586
Nr 10. Refaktoryzacja	588

Antywzorce

Antywzorce są przeciwieństwem wzorców projektowych. Stanowią one często słowniki ZŁ rozwiązań pewnych problemów. Powinny być w stanie rozpoznawać i niedopuszczalne pójścia i uśmiałek.

Klasa: DogDoor

Opis: Reprezentuje fizyczne drzwi dla psa. Stanowi interfejs zapewniający możliwość korzystania z urządzeń sprzętowych kontrolujących działanie drzwiczek dla psa.

Odpowiedzialności:

Nazwa	Współpracownik
Obecność drzwiczek	
Zamknięcie drzwiczek	

Zwróć uwagę, by zapisać tu zarówno operacje, które sama klasa realizuje samodzielnie, jak i te, które wykonuje przy udziale innych klas.

Do odpowiedzialności nie są wliczone metody dostępnego.



Witamy w Obiektywie

B Stosowanie języka obiektowego

Przygotuj się na zagraniczną wycieczkę. Czas odwiedzić Obiektów — miejsce, gdzie obiekty robią to, co powinny, aplikacje są dobrze hermetyzowane (już wkrótce dowiesz się, co to znaczy), a projekty oprogramowania pozwalają na ich wielokrotne stosowanie i rozbudowę. Musisz jeszcze poznać kilka dodatkowych zagadnień i poszerzyć swoje umiejętności językowe. Nie przejmuj się jednak, nie zajmie Ci to wiele czasu i zanim się obejrzyysz, już będziesz rozmawiał w języku obiektowym, jakbyś mieszkał w Obiektywie od wielu lat.

UML i diagramy klas	591
Dziedziczenie	593
Polimorfizm	595
Hermetyzacja	596
Celne spostrzeżenia	600

Oto jak przedstawiamy klasy na fakcie zwanym **diagramie klas**. To właśnie w taki sposób UML pozwala na przedstawienie szczegółowych informacji o klasach tworzących aplikację.

To są zmienne składowe klasy. Każda ma pewną nazwę, a po dwukropku określony jest jej typ.

To są metody klasy. Każda metoda ma nazwę, za nią w nawiasach podawane są parametry metody, a następnie, po dwukropku, typ wartości wynikowej.

To jest nazwa klasy. Zawsze jest umieszczane na samej górze diagramu i zapisywane pogrubioną czcionką.



Ta linia oddziela zmienne składowe klasy od jej metod.

Diagram klasy pozwala wyobrazić sobie ogólną postać klasy: bez trudu, na pierwszy rzut oka, można powiedzieć, co ona robi. W razie potrzeby w diagramie można nawet pominąć fragment zawierający zmienne składowe lub metody, jeśli to pomoże w przekazaniu niezbędnych informacji.

S Skorowidz

603