W niekonwencjonalny sposób poznaj techniki analizy i projektowania obiektowego

Rusz głową!

Analiza i projektowanie obiektowe

Popraw swoje umiejętności komunikacji dzięki zastosowaniu diagramów UML i przypadków użycia

Zmuś swój mózg do wysiłku, rozwiązując dziesiątki obiektowych ćwiczeń i zagadek

Nie pozwól, by Twoi klienci byli niezadowoleni

Przekształć opracowane wymagania i projekty w poważne oprogramowanie

Załaduj ważne zasady projektowania obiektowego prosto do swojego mózgu

Dowiedz się, w jaki sposób agregacja, wyodrębnianie i delegowanie pomogły Marii rozpocząć błyskotliwą karierę w Obiektowie

O'REILLY®

Brett D. McLaughlin,
Gary Pollice, David West

Helion

helion.pl

Idź do

- Spis treści
- Przykładowy rozdział

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

Analiza i projektowanie obiektowe. Rusz głową!

Autorzy: Brett D. McLaughlin, Gary Pollice, Dave West
Tłumaczenie: Piotr Rajca
ISBN: 978-83-246-2802-5
Tytuł oryginału: Head First Object-Oriented Analysis and Design
Format: 200×230, stron: 624

Współczesne systemy informatyczne mają niewiele wspólnego z tymi sprzed kilkunastu lat. Są skomplikowane, nafaszerowane wieloma technologiami, bywa też, że mają (zbyt) wielu autorów. Jak zapanować nad tym wszystkim? Jak projektować systemy szybko oraz bezbłędnie? Czujesz się zagubiony? Nic się nie martw! Po prostu...

Otwórz swój umysł! Teraz dzięki nowatorskim metodom nauczania możesz błyskawicznie opanować wszystkie elementy projektowania obiektowego. Charakterystyczna dla serii „Rusz głową!" cecha to wymieszana w odpowiednich proporcjach wiedza, humor oraz wszystko wyjaśniające grafiki. Informacje zawarte w książce obejmują pełny zakres tematyki związanej z analizą i projektowaniem obiektowym. Tylko kilkaset stron dzieli Cię od opanowania metod zbierania wymagań, tworzenia przypadków użycia czy też projektowania diagramów klas. A to tylko początek – sprawdź spis treści i przekonaj się, jak szeroki materiał zawiera ta książka.

Naprzód, głowo!

Nikt ci tego nie potrafił wytłumaczyć? Wydaje Ci się, że to problem nie na Twoją głowę? Nie potrzebujesz elektrowstrząsów, żeby pobudzić swój mózg do aktywnego działania. Tylko żadnych gwałtownych gestów! Usiądź wygodnie, otwórz książkę, dopiero teraz się zacznie. Na początek – rusz głową!

Precz z nudnymi wykładami i zakuwaniem bez zrozumienia!

Nauka to znacznie więcej niż tylko czytanie suchego tekstu. Twój mózg jest niczym głodny rekin, cały czas pręcy naprzód w poszukiwaniu nowej, apetycznej przekąski.

Jak karmimy Twój wygłodniały umysł?

Używamy rysunków, bo obraz wart jest 1024 słów. Stosujemy powtórzenia, by zakodować na stałe dane w Twojej chłonnej głowie. Oddziałujemy na emocje, jesteśmy nieprzewidywalni, zaskakujący i zabawni. Stawiamy przed Tobą wyzwania i zadajemy pytania, które angażują Cię w proces studiowania przedstawianych zagadnień. Cały czas pobudzamy Twój umysł do aktywnego działania, zmuszamy go do posłuszeństwa... a za ciężką pracę nagrodzimy go smakowitym ciasteczkiem w postaci wiedzy – wisienka gratis!

Rozgryź to sam!

- Zasady i cele projektowania obiektowego
- Metody zbierania wymagań
- Przypadki użycia i ich analiza
- Graficzna prezentacja systemu i zasad jego działania – diagramy UML
- Wzorce projektowe – sprawy skomplikowane stają się proste, a proste jeszcze prostsze
- Projektowanie architektury systemu
- Testowanie

Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 32 230 98 63
e-mail: helion@helion.pl
© Helion 1991–2010

Spis treści (skrócony)

1	Dobrze zaprojektowane aplikacje są super. *Tu zaczyna się wspaniałe oprogramowanie*	31
2	Gromadzenie wymagań. *Daj im to, czego chcą*	83
3	Wymagania ulegają zmianom. *Kocham cię, jesteś doskonały… A teraz — zmień się*	137
4	Analiza. *Zaczynamy używać naszych aplikacji w rzeczywistym świecie*	169
5	Część 1. Dobry projekt = elastyczne oprogramowanie. *Nic nie pozostaje wiecznie takie samo*	221
	Przerywnik. Obiektowa katastrofa	245
	Część 2. Dobry projekt = elastyczne oprogramowanie. *Zabierz swoje oprogramowanie na 30-minutowy trening*	257
6	Rozwiązywanie naprawdę dużych problemów. *„Nazywam się Art Vandelay... jestem Architektem”*	301
7	Architektura. *Porządkowanie chaosu*	343
8	Zasady projektowania. *Oryginalność jest przereklamowana*	395
9	Powtarzanie i testowanie. *Oprogramowanie jest wciąż przeznaczone dla klienta*	441
10	Proces projektowania i analizy obiektowej. *Scalając to wszystko w jedno*	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
Zmuś 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 rozwikłać w pierwszej kolejności, a jednocześnie mieć pewność, że podczas wprowadzania poprawek nie popsujemy 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.

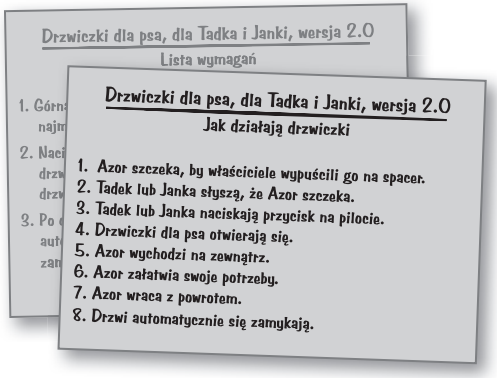
Rock-and-roll jest wieczny!	32
Nowa elegancka aplikacja Ryśka…	33
Co przede wszystkim zmieniłbyś w aplikacji Ryśka?	38
Doskonałe oprogramowanie… ma więcej niż jedną z wymienionych już cech	42
Wspaniałe oprogramowanie w trzech prostych krokach	43
W pierwszej kolejności skoncentruj się na funkcjonalności	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

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 „Satysfakcja gwarantowana" i posuniesz się o kolejny krok na drodze do tworzenia doskonałego oprogramowania... i to za każdym razem.

Nadszedł czas na kolejny pokaz Twych programistycznych umiejętności	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

Sądzisz, 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 miłego, 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 patałachem!	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);
  }
```

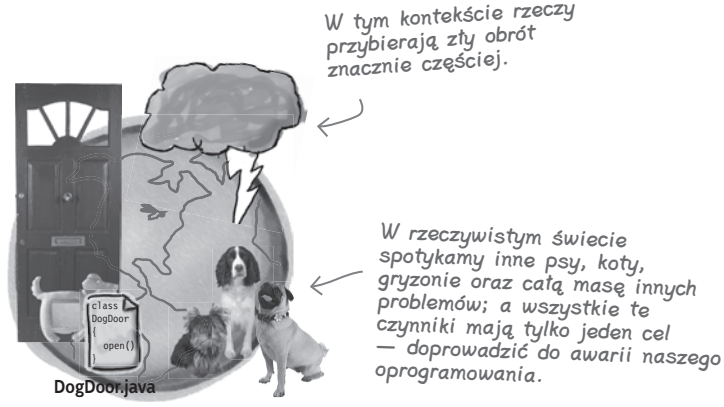
Remote.java

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**!".

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



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	Sławni projektanci	Konstrukcje używane w kodzie	Utrzymanie i wielokrotne użycie	Nerwica 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**. Brzmi interesująco? A zatem przewróć kartkę — przystępujemy do poprawiania nieelastycznej aplikacji.

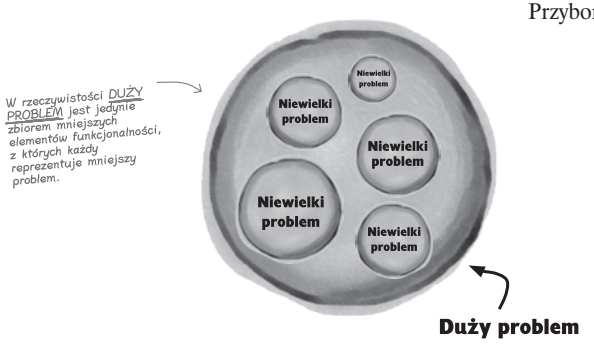
Wróćmy do aplikacji wyszukiwawczej Ryśka	258
Dokładniejsza analiza metody search()	261
Korzyści, jakie dała nam analiza	262
Dokładniejsza analiza klas instrumentów	265
Śmierć projektu (decyzja)	270
Zmieńmy złe decyzje projektowe na dobre	271
Zastosowanie „podwójnej hermetyzacji” w aplikacji Ryśka	273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

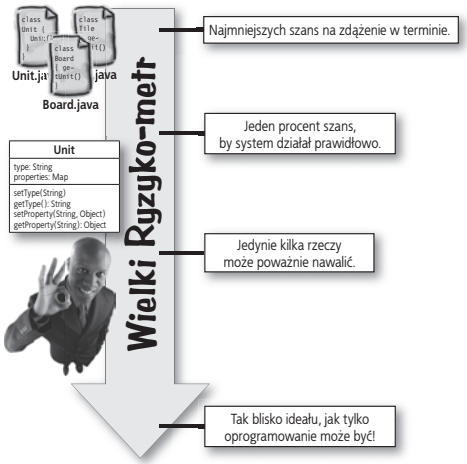
Rozwiązywanie dużych problemów	302
Wszystko zależy od sposobu spojrzenia na duży problem	303
Wymagania i przypadki użycia to dobry punkt wyjściowy…	308
Potrzebujemy znacznie więcej informacji	309
Określanie możliwości	312
Możliwość czy wymaganie	314
Przypadki użycia nie zawsze pomagają ujrzeć ogólny 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



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 nie 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.

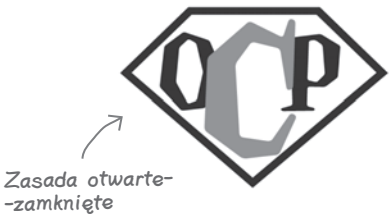


Czy czujesz się nieco przytłoczony?	344
Potrzebujemy architektury	346
Zacznijmy od funkcjonalności	349
Co ma znaczenie dla architektury	351
Trzy „P” dotyczące architektury	352
Wszystko sprowadza się do problemu ryzyka	358
Scenariusze pomagają zredukować ryzyko	361
Koncentruj się na jednej możliwości 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 wspaniałe oprogramowanie	391
Celne spostrzeżenia	392

Zasady projektowania

8 Oryginalność 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 **rozwikłał 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 otwarte-zamknięte

Zasada nie powtarzaj się

Zasada jednej odpowiedzialności

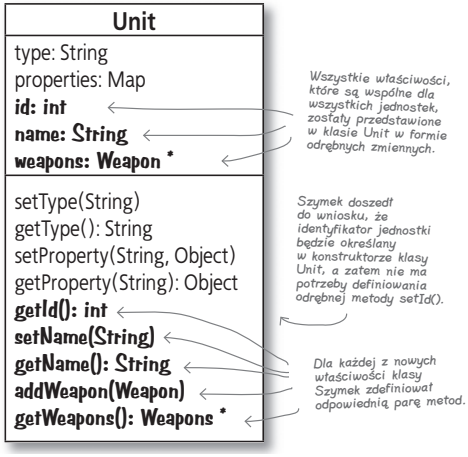
Zasada podstawienia Liskov

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
Przybornik projektanta	438

Powtarzanie i testowanie

9 Oprogramowanie jest wciąż przeznaczone dla klienta

Czas pokazać klientowi, jak bardzo Ci na nim zależy. Nękają 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 błogie ciepło, które sprawi, że powiedzą o Tobie: **„O tak, nie ma co do tego wątpliwości, jest właściwą osobą do napisania naszej aplikacji!".**



Twój przybornik narzędziowy powoli się wypełnia	442
Wspaniałe 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
Przybornik projektanta	496

Proces projektowania i analizy obiektowej

10 Scalając to wszystko w jedno

Czy dotarliśmy już do celu? Poświęciliśmy sporo czasu i wysiłku, by poznać wiele różnych sposobów pozwalających poprawić jakość tworzonego oprogramowania; teraz jednak nadeszła pora, by **połączyć i podsumować wszystkie zdobyte informacje**. Na to właśnie czekałeś: mamy zamiar zebrać **wszystko**, czego się nauczyłeś, i pokazać Ci, że wszystkie te informacje stanowią części jednego procesu, którego możesz wielokrotnie używać, by **tworzyć wspaniałe oprogramowanie**.

Tworzenie oprogramowania w stylu obiektowym	500
Trans-Obiektów	504
Mapa metra w Obiektowie	506
Lista możliwości	509
Przypadki użycia odpowiadają zastosowaniu, możliwości odpowiadają funkcjonalności	515
A teraz zacznij powtarzać te same czynności	519
Dokładniejsza analiza sposobu reprezentacji sieci metra	521
Używać klasy Line czy też nie używać... oto jest pytanie	530
Najważniejsze sprawy związane z klasą Subway	536
Ochrona własnych klas	539
Czas na przerwę	547
Wróćmy znowu do etapu określania wymagań	549
Koncentruj się na kodzie, a potem na klientach	551
Powtarzanie sprawia, że problemy stają się łatwiejsze	555
Jak wygląda trasa?	560
Samemu sprawdź Przewodnik Komunikacyjny po Obiektowie	564
Ktoś chętny na trzeci cykl prac?	567
Podróż jeszcze nie dobiegła końca…	569

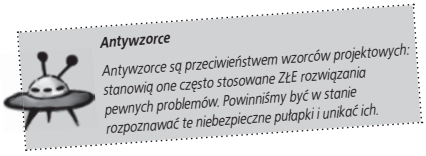
Pozostałości

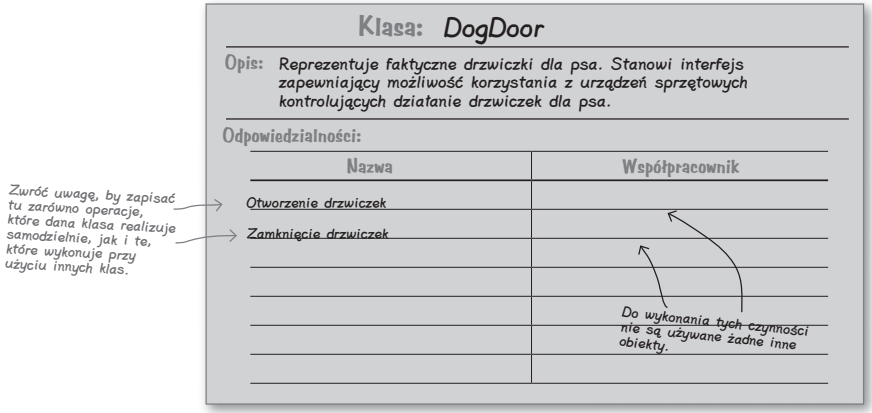
A Dziesięć najważniejszych tematów (których nie poruszyliśmy)

Możesz nam wierzyć albo i 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 któż, 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



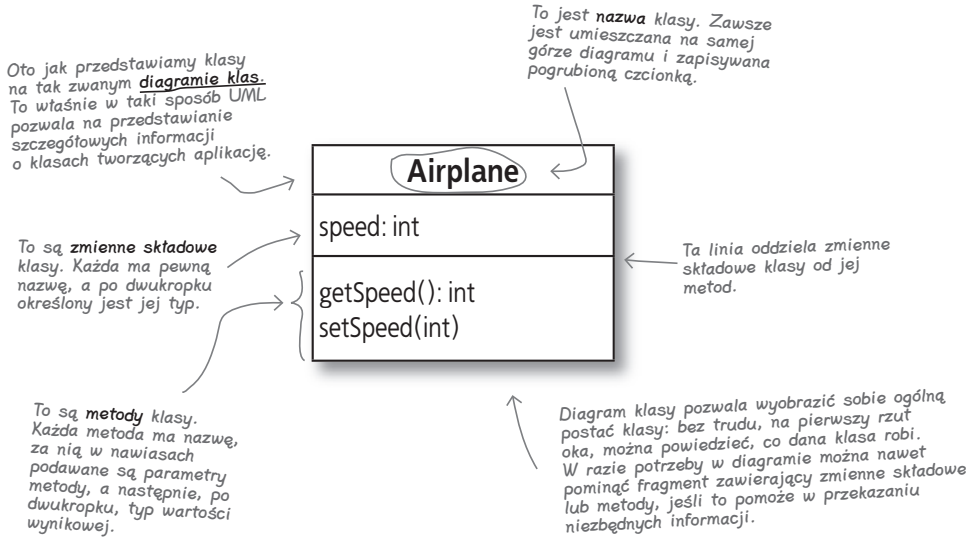


Witamy w Obiektowie

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 obejrzysz, już będziesz rozmawiał w języku obiektowym, jakbyś mieszkał w Obiektowie od wielu lat.

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



S Skorowidz 603

Kocham cię, jesteś doskonały… A teraz — zmień się

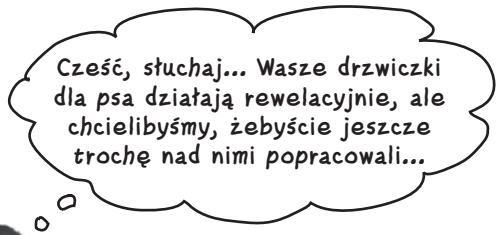
Sądzisz, że dowiedziałeś się już wszystkiego o tym, czego chciał klient? Nie tak szybko… A zatem przeprowadziłeś rozmowy z klientem, zgromadziłeś wymagania, napisałeś przypadki użycia, napisałeś i dostarczyłeś klientowi odlotową aplikację. W końcu nadszedł czas na miłego, relaksującego drinka, nieprawdaż? 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!

Pyszna pina colada do picia, słońce przyjemnie ogrzewające Twoje ciało, zwitek studolarowych banknotów wciśnięty za kąpielówki… właśnie tak wygląda życie programisty, który sprawił, że interesy firmy PsieOdrzwia kwitną. Drzwiczki, które stworzyliście dla Tadka i Janki, okazały się niesamowitym sukcesem i obecnie Darek sprzedaje je klientom na całym świecie.

Darek zarabia na Twoim kodzie naprawdę duże pieniądze.

Lecz wtem dzwoni telefon...



Tadek i Janka, którzy beztrosko przerywają Twoje wakacje.

Ty: O rany, czy coś jest nie tak z drzwiczkami?

Tadek i Janka: Nie, absolutnie nie. Drzwiczki działają dokładnie tak, jak opisałeś.

Ty: No, ale musi być jakiś problem, prawda? Może drzwiczki nie zamykają się odpowiednio szybko? A może nie działa przycisk na pilocie?

Tadek i Janka: Nie, co ty… Wszystko działa równie dobrze jak tego dnia, kiedy zainstalowałeś cały system i pokazałeś nam, jak wszystko funkcjonuje.

Ty: No to może Azor przestał szczekać, żebyście go wypuścili na zewnątrz? A… a sprawdziliście baterie w pilocie?

Tadek i Janka: Słuchaj, naprawdę z drzwiczkami jest wszystko w porządku. Po prostu mamy kilka pomysłów na modyfikacje, które chcielibyśmy wprowadzić…

Ty: Ale skoro wszystko dobrze działa, to w czym problem?

Męczy nas ciągła konieczność nasłuchiwania, czy Azor szczeka... Czasami nawet go nie słyszymy i Azor załatwia swoje potrzeby w kuchni...

No i ciągle się nam gubi pilot do drzwiczek albo zostawiamy go w innym pokoju. Męczy mnie już to ciągłe naciskanie przycisku, by otworzyć drzwiczki.

Drzwiczki dla psa, dla Tadka i Janki, wersja 2.0

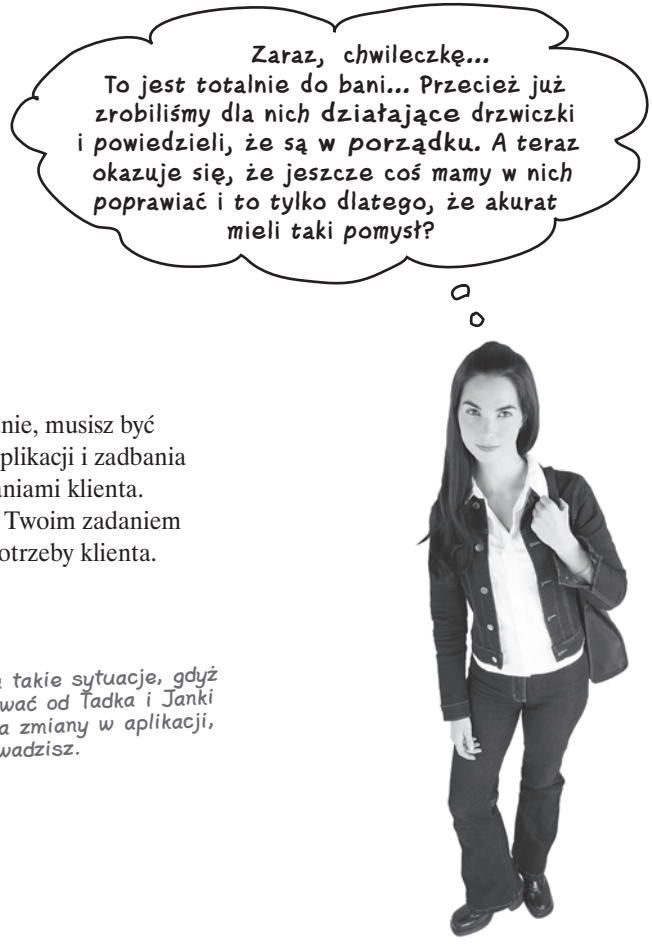
Jak (obecnie) działają drzwiczki

1. Azor szczeka, by właściciele wypuścili go na spacer.
2. **Tadek lub Janka słyszą, że Azor szczeka.**
3. **Tadek lub Janka naciskają przycisk na pilocie.**
4. Drzwiczki dla psa otwierają się.
5. Azor wychodzi na zewnątrz.
6. Azor załatwia swoje potrzeby.
   - 6.1 Drzwi zamykają się automatycznie.
   - 6.2. Azor szczeka, by właściciele wpuścili go do domu.
   - 6.3. Tadek lub Janka słyszą szczekanie Azora (znowu).
   - 6.4. Tadek lub Janka naciskają przycisk na pilocie.
   - 6.5. Drzwiczki dla psa otwierają się (znowu).
7. Azor wraca z powrotem.
8. Drzwi automatycznie się zamykają.

Czy nie dałoby się zrobić tak, żeby drzwiczki otwierały się automatycznie, kiedy Azor zaszczeka? W takim przypadku nie musielibyśmy w ogóle niczego robić, żeby go wypuścić. Rozmawialiśmy na ten temat i oboje uważamy, że to jest DOSKONAŁY pomysł!

No i wracamy do tablicy

A zatem nadszedł czas, by zabrać się do pracy nad poprawieniem drzwiczek dla psa zamówionych przez Tadka i Jankę. Musimy określić, w jaki sposób należy otwierać drzwiczki za każdym razem, gdy Azor zaszczeka. Zacznijmy od…



Klient ma zawsze rację

Nawet jeśli wymagania ulegną zmianie, musisz być przygotowany do zaktualizowania aplikacji i zadbania o to, by działała zgodnie z oczekiwaniami klienta. Kiedy klient wymyśli nową funkcję, Twoim zadaniem będzie zmienić aplikację i spełnić potrzeby klienta.

Darek uwielbia takie sytuacje, gdyż może zainkasować od Tadka i Janki nową sumkę za zmiany w aplikacji, które Ty wprowadzisz.

WYSIL SZARE KOMÓRKI

Właśnie poznałeś jedyny pewnik występujący w analizie i projektowaniu obiektowym. Jak myślisz, co nim jest?

Jedyny pewnik analizy i projektowania obiektowego*

W porządku, zatem co jest tym jedynym pewnikiem, na który zawsze możesz liczyć, pisząc oprogramowanie?

Niezależnie od tego, gdzie pracujesz, jakie oprogramowanie tworzysz oraz jakiego języka programowania używasz, jaka jest jedyna stała, która zawsze będzie Ci towarzyszyć?

ANAIMS

(użyj lusterka, by odczytać odpowiedź)

Niezależnie od tego, jak dobrze zaprojektujesz swoją aplikację, to wraz z upływem czasu zawsze będzie się ona rozwijać i zmieniać. Poznasz nowe rozwiązania występujących w niej problemów, używane języki programowania będą ewoluować, Twoi mili klienci wymyślą nowe, zwariowane wymagania, które Ty będziesz musiał „poprawiać".

Zaostrz ołówek

Wymagania cały czas ulegają zmianom... czasami w połowie realizacji projektu, a czasami w chwili, gdy sądziłeś, że wszystko już jest gotowe. Poniżej zapisz kilka potencjalnych przyczyn, dla których mogą ulec zmianie wymagania w aktualnie pisanej przez Ciebie aplikacji.

Mój klient zdecydował, że chciałby, by aplikacja działała w inny sposób.

Mój szef uznał, że aplikacja spisywałaby się lepiej, gdyby została napisana jako aplikacja internetowa, a nie tradycyjna.

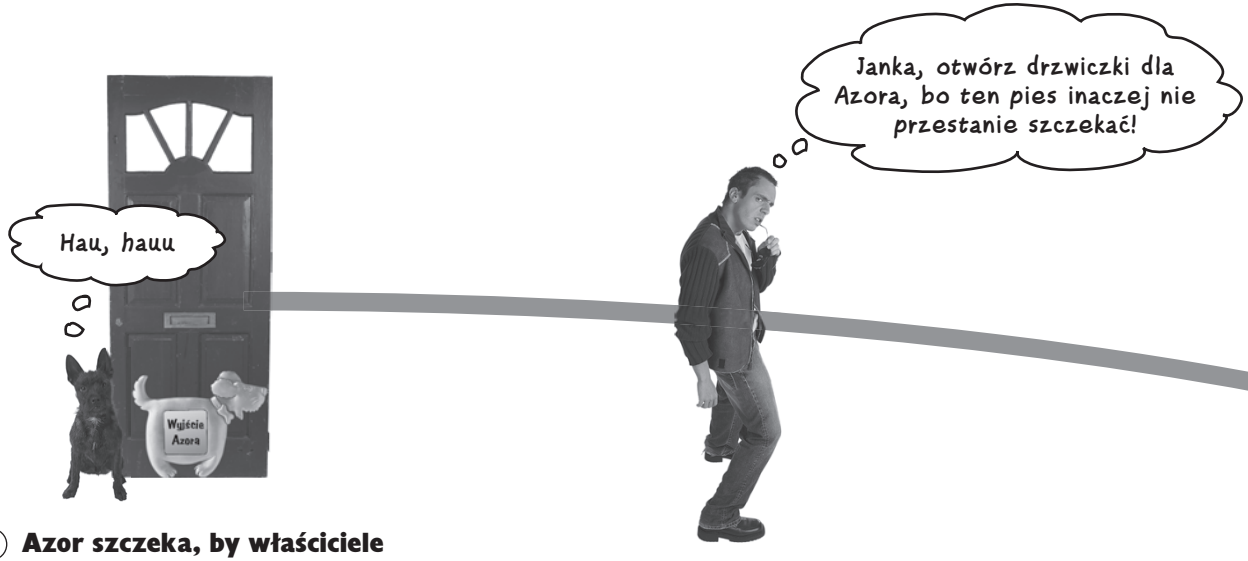
Wymagania zawsze ulegają zmianom. Jeśli jednak stworzyłeś dobre przypadki użycia, to zazwyczaj będziesz w stanie szybko zmodyfikować aplikację i dostosować ją do nowych wymagań.

* *Jeśli czytałeś książkę Wzorce projektowe. Rusz głową!, to zapewne ta strona będzie wyglądała znajomo. Autorzy tej książki w tak doskonały sposób opisali modyfikacje, że postanowiliśmy „pożyczyć" ich pomysły i jedynie ZMIENIĆ kilka słów tu i ówdzie. Dziękujemy wam, Beth i Ericu!*

Ćwiczenie

Rozszerzyć możliwości drzwiczek Tadka i Janki o rozpoznawanie szczekania.

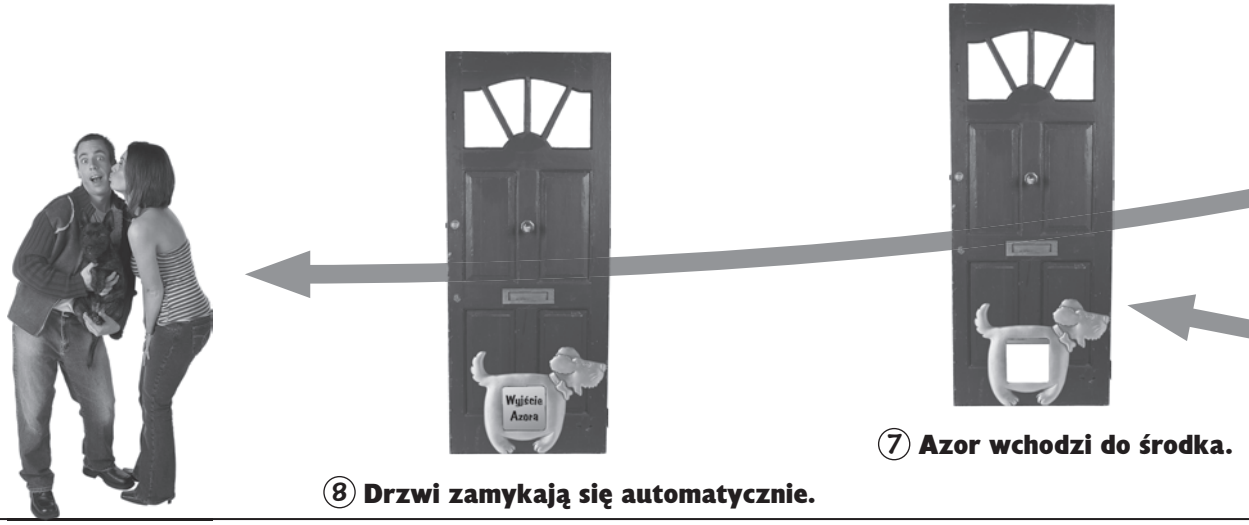
Zaktualizuj diagram i dodaj do niego ścieżkę alternatywną przedstawiającą sytuację, w której Azor zaczyna szczekać. Nowy system rozpoznawania dźwięków, oferowany przez Darka, rozpoznaje szczekanie i drzwiczki otwierają się automatycznie. Pilot do drzwiczek wciąż powinien funkcjonować, dlatego z diagramu nie powinieneś niczego usuwać; po prostu dodaj do niego nową ścieżkę prezentującą, jak system działa w przypadku rozpoznania szczekania Azora.



① **Azor szczeka, by właściciele wypuścili go na spacer.**

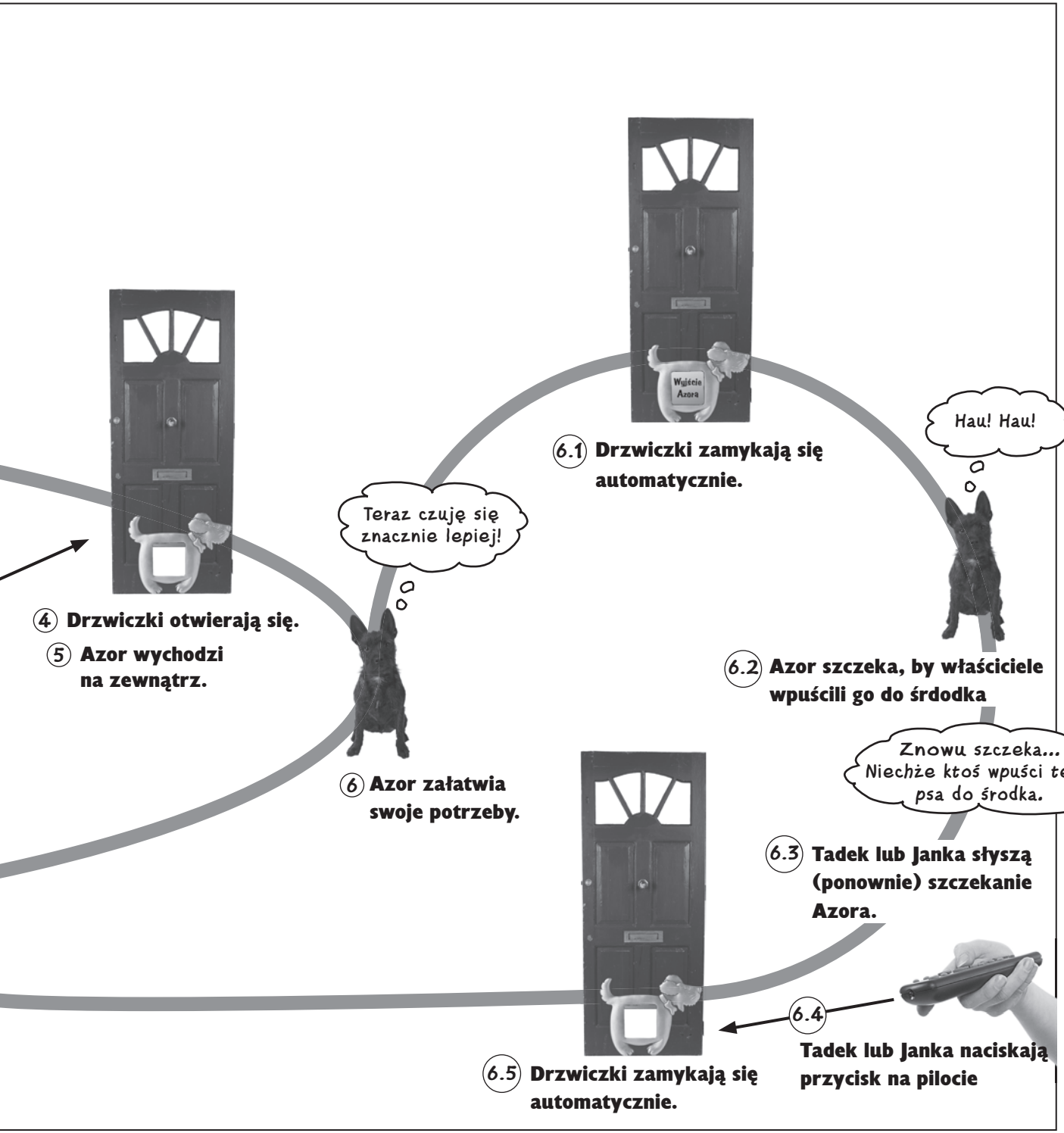
② **Tadek lub Janka słyszą, że Azor szczeka.**

③ **Tadek lub Janka naciskają przycisk na pilocie.**



⑦ **Azor wchodzi do środka.**

⑧ **Drzwi zamykają się automatycznie.**



6.1 **Drzwiczki zamykają się automatycznie.**

6.2 **Azor szczeka, by właściciele wpuścili go do środka**

6.3 **Tadek lub Janka słyszą (ponownie) szczekanie Azora.**

6.4 **Tadek lub Janka naciskają przycisk na pilocie**

6.5 **Drzwiczki zamykają się automatycznie.**

6 **Azor załatwia swoje potrzeby.**

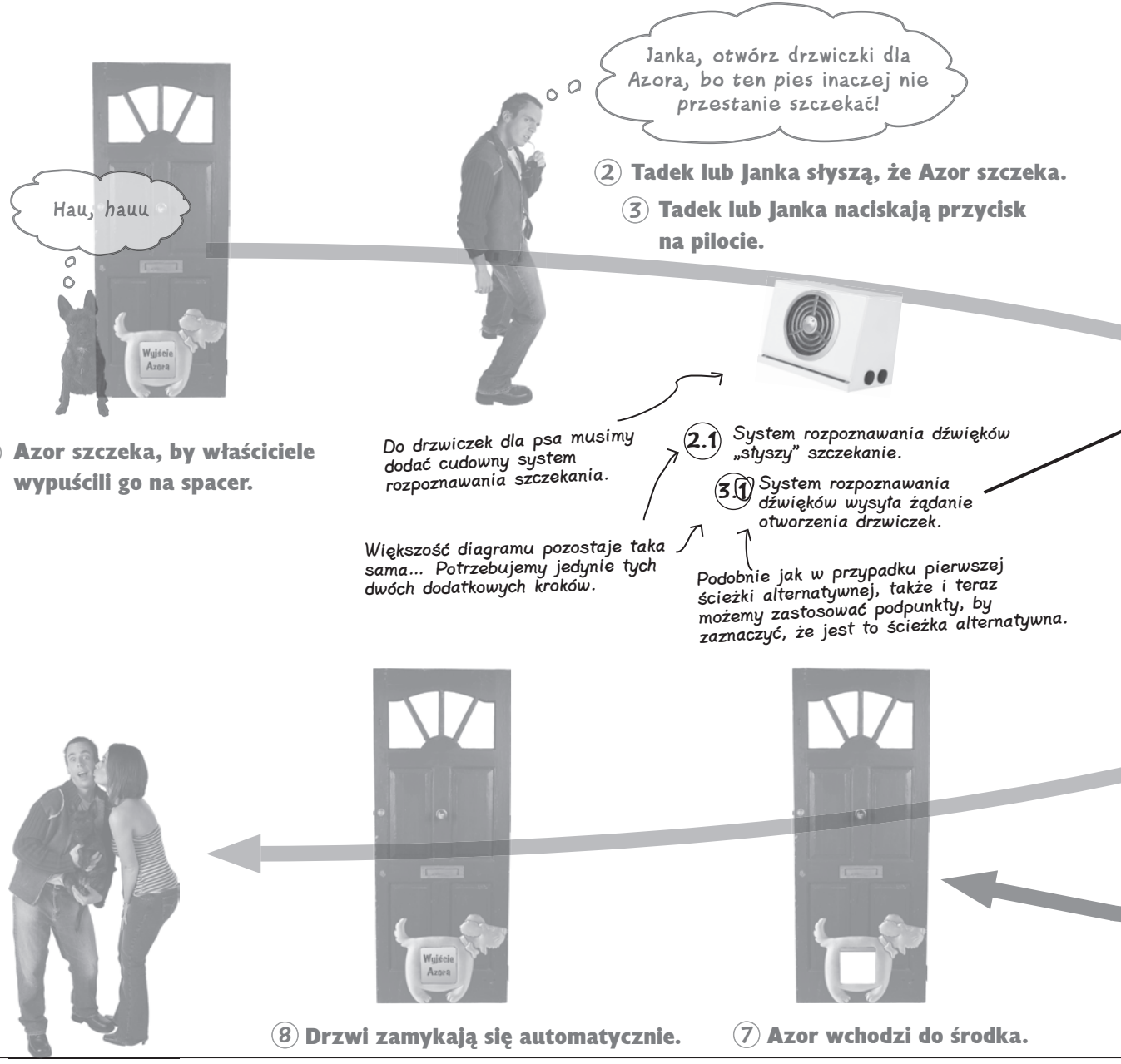
4 **Drzwiczki otwierają się.**

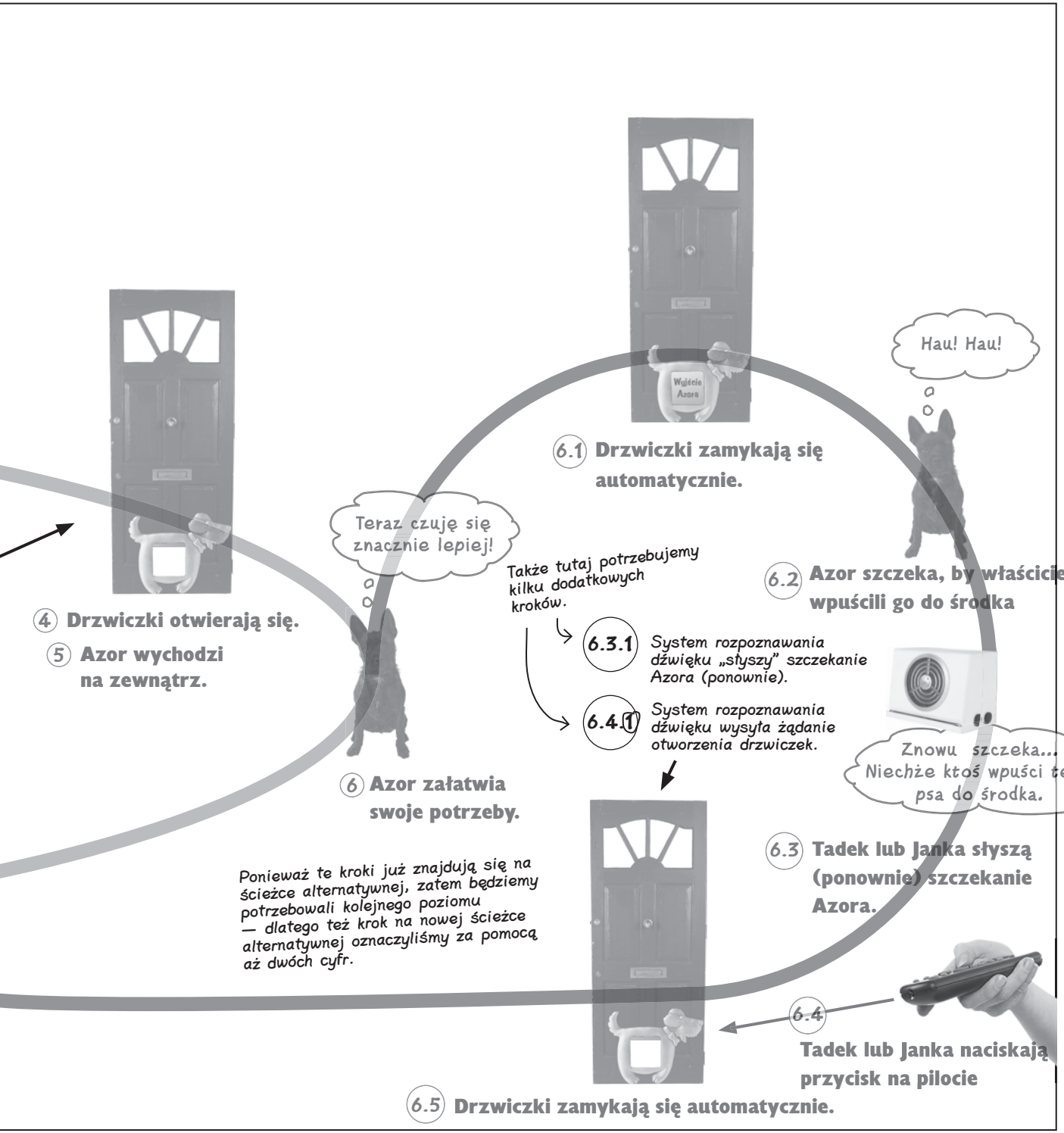
5 **Azor wychodzi na zewnątrz.**

Rozwiązania ćwiczeń

Rozszerzyć możliwości drzwiczek Tadka i Janki o rozpoznawanie szczekania.

Zaktualizuj diagram i dodaj do niego ścieżkę alternatywną przedstawiającą sytuację, w której Azor zaczyna szczekać. Nowy system rozpoznawania dźwięków, oferowany przez Darka, rozpoznaje szczekanie i drzwiczki otwierają się automatycznie. Pilot do drzwiczek wciąż powinien funkcjonować, dlatego z diagramu nie powinieneś niczego usuwać; po prostu dodaj do niego nową ścieżkę prezentującą, jak system działa w przypadku rozpoznania szczekania Azora.





6.1 **Drzwiczki zamykają się automatycznie.**

6.2 **Azor szczeka, by właściciele wpuścili go do środka**

6.3 **Tadek lub Janka słyszą (ponownie) szczekanie Azora.**

6.4 **Tadek lub Janka naciskają przycisk na pilocie**

6.5 **Drzwiczki zamykają się automatycznie.**

6 **Azor załatwia swoje potrzeby.**

4 **Drzwiczki otwierają się.**

5 **Azor wychodzi na zewnątrz.**

Ale teraz mój przypadek użycia jest totalnie pogmatwany i trudny do zrozumienia. Wszystkie te ścieżki alternatywne sprawiają, że określenie, co się właściwie dzieje, przysparza poważnych problemów.

Ścieżką oryginalną? Ścieżką alternatywną? Kto to wie?

Drzwiczki dla psa, dla Tadka i Janki, wersja 2.1

Jak (obecnie) działają drzwiczki

1. Azor szczeka, by właściciele wypuścili go na spacer.
2. Tadek lub Janka słyszą, że Azor szczeka.
   - **2.1. System rozpoznawania dźwięków „słyszy" szczekanie Azora.**
3. Tadek lub Janka naciskają przycisk na pilocie.
   - **3.1. System rozpoznawania dźwięków wysyła żądanie otworzenia drzwiczek.**
4. Drzwiczki dla psa otwierają się.
5. Azor wychodzi na zewnątrz.
6. Azor załatwia swoje potrzeby.
   - 6.1 Drzwi zamykają się automatycznie.
   - 6.2. Azor szczeka, by właściciele wpuścili go do domu.
   - 6.3. Tadek lub Janka słyszą szczekanie Azora (znowu).
     - **6.3.1. System rozpoznawania dźwięków „słyszy" szczekanie Azora (ponownie).**
   - 6.4. Tadek lub Janka naciskają przycisk na pilocie.
     - **6.4.1. System rozpoznawania dźwięków wysyła żądanie otworzenia drzwiczek.**
   - 6.5. Drzwiczki dla psa otwierają się (znowu).
7. Azor wraca z powrotem.
8. Drzwi automatycznie się zamykają.

Teraz pojawiły się alternatywne kroki dla oryginalnych kroków numer 2. i 3.

Przedstawiliśmy je jako podpunkty listy, trzeba jednak pamiętać, że w rzeczywistości stanowią one zupełnie nową ścieżkę alternatywną.

Te podpunkty stanowią dodatkowy zestaw kroków, które mogą być wykonane...

Teraz mamy już nawet kroki alternatywne do wcześniejszych kroków alternatywnych.

... a te podpunkty, stanowią w rzeczywistości zupełnie inną ścieżkę przejścia przez przypadek użycia.

Ciągle uważam, że ten przypadek użycia jest trudny do zrozumienia i nieczytelny. Wygląda na to, że Tadek i Janka zawsze słyszą, kiedy Azor szczeka, ale urządzenie do rozpoznawana dźwięków słyszy go tylko czasami. Ale to nie jest to, czego by chcieli Tadek i Janka.

Czy rozumiesz, o czym mówi Gerard? Pomysł Tadka i Janki polegał na tym, by już więcej nie musieli nasłuchiwać, czy Azor szczeka, czy nie.

Drzwiczki dla psa, dla Tadka i Janki, wersja 2.1

Jak (obecnie) działają drzwiczki

1. Azor szczeka, by właściciele wypuścili go na spacer.
2. Tadek lub Janka słyszą, że Azor szczeka.
   - 2.1. System rozpoznawania dźwięków „słyszy" szczekanie Azora.
3. Tadek lub Janka naciskają przycisk na pilocie.
   - 3.1. System rozpoznawania dźwięków wysyła żądanie otworzenia drzwiczek.
4. Drzwiczki dla psa otwierają się.
5. Azor wychodzi na zewnątrz.
6. Azor załatwia swoje potrzeby.
   - 6.1 Drzwi zamykają się automatycznie.
   - 6.2. Azor szczeka, by właściciele wpuścili go do domu.
   - 6.3. Tadek lub Janka słyszą szczekanie Azora (znowu).
     - 6.3.1. System rozpoznawania dźwięków „słyszy" szczekanie Azora (ponownie).
   - 6.4. Tadek lub Janka naciskają przycisk na pilocie.
     - 6.4.1. System rozpoznawania dźwięków wysyła żądanie otworzenia drzwiczek.
   - 6.5. Drzwiczki dla psa otwierają się (znowu).
7. Azor wraca z powrotem.
8. Drzwi automatycznie się zamykają.

W rzeczywistości w nowym przypadku użycia chodzi nam o to, by pokazać, że może być wykonany krok 2. lub krok 2.1...

...a następnie krok 3. lub krok 3.1.

Tutaj może być wykonany krok 6.3 lub krok 6.3.1...

...a tu krok 6.4 lub 6.4.1.

Przypadki użycia muszą być zrozumiałe i użyteczne przede wszystkim dla Ciebie

Jeśli masz problemy ze zrozumieniem przygotowanego przypadku użycia, *to po prostu zapisz go w jakiś inny sposób*. Istnieją setki różnych sposobów zapisywania przypadków użycia, jednak najważniejsze jest to, by był on pojmowalny dla Ciebie, Twojego zespołu oraz osób, którym musisz go wytłumaczyć. Spróbujmy zatem zapisać przypadek użycia ze strony 147 w bardziej przejrzysty i zrozumiały sposób.

Wszystkie kroki, które mogą zostać wykonane zamiast jakichś kroków ścieżki głównej, umieściliśmy z prawej strony.

Teraz dodaliśmy nagłówek informujący, że wszystkie kroki umieszczone w lewej kolumnie należą do ścieżki głównej.

Drzwiczki dla psa, dla Tadka i Janki, wersja 2.2

Jak działają drzwiczki

Ścieżka główna

1. Azor szczeka, by właściciele wypuścili go na spacer.
2. Tadek lub Janka słyszą, że Azor szczeka.
3. Tadek lub Janka naciskają przycisk na pilocie.
4. Drzwiczki dla psa otwierają się.
5. Azor wychodzi na zewnątrz.
6. Azor załatwia swoje potrzeby.
   - 6.1 Drzwi zamykają się automatycznie.
   - 6.2. Azor szczeka, by właściciele wpuścili go do domu.
   - 6.3. Tadek lub Janka słyszą szczekanie Azora (znowu).
   - 6.4. Tadek lub Janka naciskają przycisk na pilocie.
   - 6.5. Drzwiczki dla psa otwierają się (znowu).
7. Azor wraca z powrotem.
8. Drzwi automatycznie się zamykają.

Ścieżki alternatywne

2.1. System rozpoznawania dźwięków „słyszy" szczekanie Azora.

3.1. System rozpoznawania dźwięków wysyła żądanie otworzenia drzwiczek.

6.3.1. System rozpoznawania dźwięków „słyszy" szczekanie Azora (ponownie).

6.4.1. System rozpoznawania dźwięków wysyła żądanie otworzenia drzwiczek.

Ten sposób zapisu jest nieco bardziej przejrzysty i zrozumiały: możemy wykonać krok 2. LUB krok 2.1; a następnie krok 3. LUB krok 3.1.

Jeśli istnieje tylko jeden krok, to podczas realizacji sekwencji czynności opisanych przez przypadek użycia krok ten zawsze zostanie wykonany.

Kroki podrzędne są opcjonalne... Możesz ich użyć, choć nie jest to wcale wymagane. Pomimo to umieściliśmy je w lewej kolumnie, gdyż nie stanowią one zamienników dla innych kroków ścieżki głównej.

Niezależnie od tego, w jaki sposób zostanie wykonany przypadek użycia, zawsze jego realizacja zakończy się na kroku 8. należącym do ścieżki głównej.

Kroki przedstawione z prawej strony mogą zastąpić krok 6.3 oraz 6.4. Podczas realizacji procesu opisywanego przez przypadek użycia można wykonać tylko jeden z wariantów: krok zapisany z lewej LUB krok zapisany z prawej strony.

Jeśli naprawdę możemy tworzyć przypadki użycia w dowolny sposób, to czy możemy zmodyfikować je w taki sposób, by system rozpoznawania szczekania stał się elementem głównej ścieżki? Bo przecież chcemy, by w większości przypadków system działał właśnie w taki sposób, nieprawdaż?

Doskonały pomysł!

Ścieżka główna reprezentuje ten sposób działania systemu, jaki ma być realizowany w większości przypadków. Tadek i Janka zapewne życzyliby sobie, by system rozpoznawania dźwięku otwierał drzwiczki dla Azora częściej niż oni przy użyciu pilota, dlatego też takie zmodyfikowanie ścieżki głównej jest dobrym rozwiązaniem:

Drzwiczki dla psa, dla Tadka i Janki, wersja 2.3

Jak działają drzwiczki

Ścieżka główna

1. Azor szczeka, by właściciele wypuścili go na spacer.
2. System rozpoznawania dźwięków „słyszy" szczekanie Azora.
3. System rozpoznawania dźwięków wysyła żądanie otworzenia drzwiczek.
4. Drzwiczki dla psa otwierają się.
5. Azor wychodzi na zewnątrz.
6. Azor załatwia swoje potrzeby.
   - 6.1. Drzwi zamykają się automatycznie.
   - 6.2. Azor szczeka, by właściciele wpuścili go do domu.
   - 6.3. System rozpoznawania dźwięków „słyszy" szczekanie Azora (ponownie).
   - 6.4. System rozpoznawania dźwięków wysyła żądanie otworzenia drzwiczek.
   - 6.5. Drzwiczki dla psa otwierają się (znowu).
7. Azor wraca z powrotem.
8. Drzwi automatycznie się zamykają.

Ścieżki alternatywne

- 2.1. Tadek lub Janka słyszą, że Azor szczeka.
- 3.1. Tadek lub Janka naciskają przycisk na pilocie.
- 6.3.1. Tadek lub Janka słyszą szczekanie Azora (znowu).
- 6.4.1. Tadek lub Janka naciskają przycisk na pilocie.

Teraz kroki związane z wykorzystaniem systemu rozpoznawania dźwięku zostały usunięte ze ścieżki alternatywnej i dołączone do ścieżki głównej.

Obecnie Tadek i Janka będą używali pilota raczej sporadycznie, dlatego umieszczenie wszystkich kroków związanych z pilotem w ścieżkach alternatywnych będzie prawidłowym rozwiązaniem.

Od startu do mety: jeden scenariusz

Uwzględniając wszystkie ścieżki alternatywne zamieszczone w nowym przypadku użycia, istnieje wiele sposobów na wypuszczenie Azora na spacer i późniejsze wpuszczenie go do domu. Poniżej przedstawiliśmy jeden z możliwych scenariuszy.

Wykorzystajmy tę ścieżkę alternatywną i pozwólmy Tadkowi i Jance otworzyć drzwiczki przy użyciu pilota.

Wszystkie możliwe ścieżki realizacji tego przypadku użycia rozpoczynają się od kroku 1.

Drzwiczki dla psa, dla Tadka i Janki, wersja 2.3

Jak działają drzwiczki

Ścieżka główna

1. Azor szczeka, by właściciele wypuścili go na spacer.
2. System rozpoznawania dźwięków „słyszy" szczekanie Azora.
3. System rozpoznawania dźwięków wysyła żądanie otworzenia drzwiczek
4. Drzwiczki dla psa otwierają się.
5. Azor wychodzi na zewnątrz.
6. Azor załatwia swoje potrzeby.
   - 6.1. Drzwi zamykają się automatycznie.
   - 6.2. Azor szczeka, by właściciele wpuścili go do domu.
   - 6.3. System rozpoznawania dźwięków „słyszy" szczekanie Azora (ponownie).
   - 6.4. System rozpoznawania dźwięków wysyła żądanie otworzenia drzwiczek.
   - 6.5. Drzwiczki dla psa otwierają się (znowu).
7. Azor wraca z powrotem.
8. Drzwi automatycznie się zamykają.

Ścieżki alternatywne

- 2.1. Tadek lub Janka słyszą, że Azor szczeka.
- 3.1. Tadek lub Janka naciskają przycisk na pilocie.
- 6.3.1. Tadek lub Janka słyszą szczekanie Azora (znowu).
- 6.4.1. Tadek lub Janka naciskają przycisk na pilocie.

W tym przypadku zastosujemy tę oto ścieżkę, reprezentującą sytuację, w której Azor pozostaje na zewnątrz po automatycznym zamknięciu drzwiczek.

Także teraz zastosujemy ścieżkę alternatywną, w której Tadek i Janka otwierają drzwiczki pilotem.

Poruszanie się zgodnie z kolejnością wyznaczaną przez strzałki pozwala na przejście danego przypadku użycia konkretną ścieżką. Taka ścieżka jest nazywana **scenariuszem**. W jednym przypadku użycia można zazwyczaj wyróżnić kilka scenariuszy.

Realizacja przypadku użycia zawsze zakończy się na kroku 8. — Azor zawsze bezpiecznie wróci do domu.

Nie istnieją głupie pytania

P: Rozumiem ścieżkę główną naszego przypadku użycia, ale czy możecie mi jeszcze raz wytłumaczyć, czym jest ścieżka alternatywna?

O: Ścieżka alternatywna to jeden lub kilka kroków, które mogą, lecz nie muszą, zostać wykonane lub które tworzą alternatywny sposób przejścia przez przypadek użycia. Ścieżki alternatywne mogą zawierać *dodatkowe* kroki dołączane do ścieżki głównej bądź też kroki pozwalające na dotarcie do celu w sposób *całkowicie odmienny* niż ten, jaki zapewnia ścieżka główna.

P: A zatem, kiedy Azor wyjdzie na zewnątrz i utknie tam, będzie to element ścieżki alternatywnej?

O: Owszem. W naszym przypadku użycia kroki 6.1, 6.2, 6.3, 6.4 oraz 6.5 tworzą ścieżkę alternatywną. Są to kroki *dodatkowe*, które system może wykonać; są one potrzebne wyłącznie w przypadku, gdy Azor zostanie na zewnątrz po automatycznym zamknięciu się drzwiczek. Jednak jest to ścieżka alternatywna, gdyż Azor *nie zawsze* załatwia swoje potrzeby aż tak długo — system może przejść bezpośrednio z kroku 6. do kroku 7.

P: I do tego celu używamy kroków podrzędnych, oznaczonych jako 6.1 i 6.2?

O: Dokładnie. Dzieje się tak, gdyż ścieżka alternatywna zawierająca kroki dodatkowe jest po prostu zbiorem czynności, które mogą zostać wykonane jako *fragmenty* innego kroku ścieżki głównej. Kiedy Azor zostaje na zewnątrz zbyt długo, to na ścieżce głównej zostają wykonane kroki 6. i 7., dlatego też ścieżka alternatywna zaczyna się od kroku o numerze 6.1 i kończy krokiem 6.5. Wszystkie one są opcjonalnymi elementami kroku 6.

P: A zatem jak nazwać sytuację, w której będą istnieć aż dwie różne ścieżki prowadzące przez pewien fragment przypadku użycia?

O: Cóż, tak naprawdę to jest to tylko inny rodzaj ścieżki alternatywnej. Kiedy Azor zacznie szczekać, to jedna ścieżka reprezentuje sytuację, w której Tadek lub Janka usłyszą go i otworzą drzwiczki, a druga — sytuację, w której drzwiczki automatycznie otworzy system rozpoznawania szczekania. Jednak system jest zaprojektowany w taki sposób, iż może zostać wykonana tylko jedna ścieżka, czyli drzwiczki dla psa zostaną otworzone albo przy użyciu pilota, albo przez system rozpoznawania dźwięków, lecz nigdy przez oba te zdarzenia jednocześnie.

P: Czy w jednym przypadku użycia może być więcej niż jedna ścieżka alternatywna?

O: Oczywiście. W jednym przypadku użycia może być kilka ścieżek alternatywnych udostępniających dodatkowe kroki oraz wiele różnych ścieżek od warunku rozpoczęcia do warunku zakończenia. Może także istnieć ścieżka alternatywna prowadząca do szybszego zakończenia przypadku użycia... Jednak w przypadku drzwiczek dla psa zamówionych przez Tadka i Jankę nie musimy uciekać się aż do tak złożonych rozwiązań.

Kompletna ścieżka prowadząca przez cały przypadek użycia, od jego pierwszego do ostatniego kroku, jest nazywana scenariuszem.

Większość przypadków użycia posiada kilka różnych scenariuszy, jednak wszystkie one realizują ten sam cel użytkownika.

Przypadki użycia bez tajemnic

W tym tygodniu: Wyznanie Ścieżki Alternatywnej

Rusz głową!: Witamy, Ścieżko Alternatywna. Słyszeliśmy, że ostatnio nie jesteś zbyt szczęśliwa. Powiedz nam, co takiego się dzieje?

Ścieżka Alternatywna: Po prostu czasami mam wrażenie, że nie jestem dość często włączana w bieg zdarzeń. Chodzi mi o to, że trudno beze mnie stworzyć dobry przypadek użycia, jednak wygląda na to, że niemal zawsze jestem ignorowana.

Rusz głową!: Ignorowana? Ale sama właśnie powiedziałaś, że znajdujesz się niemal w każdym przypadku użycia. Zabrzmiało to tak, jakbyś była naprawdę ważna!

Ścieżka Alternatywna: Może i *zabrzmiało*. Jednak nawet jeśli jestem fragmentem przypadku użycia, to i tak mogę zostać pominięta i zastąpiona jakimś innym zbiorem kroków. To naprawdę paskudne… To tak, jakby mnie tam w ogóle nie było!

Rusz głową!: Czy możesz nam to wytłumaczyć na jakimś przykładzie?

Ścieżka Alternatywna: Właśnie kilka dni temu byłam fragmentem przypadku użycia opisującego kupowanie płyt CD w nowym internetowym sklepie muzycznym — Muzykologia. Byłam tym tak bardzo poruszona… A okazało się, że obsługuję sytuacje, w których karta kredytowa klienta została odrzucona.

Rusz głową!: Hm… ale to chyba naprawdę ważne zadanie! Zatem w czym problem?

Ścieżka Alternatywna: Cóż… może. Sądzę, że to faktycznie istotne zadanie, jednak okazuje się, że zawsze jestem pomijana. Wygląda to tak, jak gdyby wszyscy składali zamówienia, a ich karty kredytowe były zawsze akceptowane. ***Chociaż byłam częścią przypadku użycia, to nie należałam do najczęściej realizowanych scenariuszy.***

Rusz głową!: Aha, rozumiem. Czyli jeśli czyjaś karta kredytowa nie została odrzucona, to w ogóle nie byłaś wykonywana.

Ścieżka Alternatywna: Właśnie! A specjaliści od finansów i bezpieczeństwa wprost mnie uwielbiali; wciąż zachwycali się, jak bardzo jestem ważna dla firmy… Ale kto by chciał siedzieć cały czas na stołku i próżnować?

Rusz głową!: Chyba zaczynam rozumieć. Niemniej jednak wciąż pomagasz temu przypadkowi użycia, prawda? Nawet jeśli nie jesteś nieustannie używana, to jednak od czasu do czasu musisz wkroczyć do akcji.

Ścieżka Alternatywna: To prawda, wszyscy mamy ten sam cel. Po prostu nie zdawałam sobie sprawy z tego, że mogę być tak ważna dla przypadku użycia, a jednocześnie niemal całkowicie ignorowana.

Rusz głową!: Cóż, tylko pomyśl… Przypadek użycia nigdy nie byłby kompletny bez ciebie.

Ścieżka Alternatywna: No tak… Kroki 3.1 i 4.1 wciąż mi to powtarzają. Oczywiście, one są częścią ścieżki alternatywnej wykonywanej wtedy, gdy klienci mają już założone konto w naszym sklepie, i dlatego są wykonywane cały czas. Łatwo im mówić!

Rusz głową!: Trzymaj się. Wszyscy wiemy, że jesteś bardzo ważną częścią przypadku użycia.

Zaostrz ołówek

Ile scenariuszy można wyróżnić w przypadku użycia opisującym drzwiczki dla Tadka i Janki?

Na ile sposobów można przejść przypadek użycia opisujący działanie drzwiczek zamówionych przez Tadka i Jankę? Pamiętaj, że czasami konieczne jest wykorzystanie jednej z istniejących ścieżek alternatywnych, a niekiedy wszystkie ścieżki alternatywne można w ogóle pominąć.

Drzwiczki dla psa, dla Tadka i Janki, wersja 2.3

Jak działają drzwiczki

Ścieżka główna

1. Azor szczeka, by właściciele wypuścili go na spacer.
2. System rozpoznawania dźwięków „słyszy" szczekanie Azora.
3. System rozpoznawania dźwięków wysyła żądanie otworzenia drzwiczek.
4. Drzwiczki dla psa otwierają się.
5. Azor wychodzi na zewnątrz.
6. Azor załatwia swoje potrzeby.
   - 6.1 Drzwi zamykają się automatycznie.
   - 6.2. Azor szczeka, by właściciele wpuścili go do domu.
   - 6.3. System rozpoznawania dźwięków „słyszy" szczekanie Azora (ponownie).
   - 6.4. System rozpoznawania dźwięków wysyła żądanie otworzenia drzwiczek.
   - 6.5. Drzwiczki dla psa otwierają się (znowu).
7. Azor wraca z powrotem.
8. Drzwi automatycznie się zamykają.

Ścieżki alternatywne

- 2.1. Tadek lub Janka słyszą, że Azor szczeka.
- 3.1. Tadek lub Janka naciskają przycisk na pilocie.
- 6.3.1. Tadek lub Janka słyszą szczekanie Azora (znowu).
- 6.4.1. Tadek lub Janka naciskają przycisk na pilocie.

Sprawdź nasze odpowiedzi podane na następnej stronie.

Aby pomóc Ci w rozwiązaniu tego zadania, zapisaliśmy wszystkie kroki, jakie są wykonywane w scenariuszu przedstawionym na powyższym rysunku.

1. 1, 2.1, 3.1, 4, 5, 6.1, 6.2, 6.3.1, 6.4.1, 6.5, 7, 8
2. _____
3. _____
4. _____
5. _____
6. _____
7. _____
8. _____

Być może nie będziesz potrzebował wszystkich tych pustych miejsc.

Zaostrz ołówek

Rozwiązanie Ile scenariuszy można wyróżnić w przypadku użycia opisującym drzwiczki dla Tadka i Janki?

Na ile sposobów można przejść przypadek użycia opisujący działanie drzwiczek zamówionych przez Tadka i Jankę? Pamiętaj, że czasami konieczne jest wykorzystanie jednej z istniejących ścieżek alternatywnych, a niekiedy wszystkie ścieżki alternatywne można w ogóle pominąć.

Drzwiczki dla psa, dla Tadka i Janki, wersja 2.3

Jak działają drzwiczki

Ścieżka główna

1. Azor szczeka, by właściciele wypuścili go na spacer.
2. System rozpoznawania dźwięków „słyszy" szczekanie Azora.
3. System rozpoznawania dźwięków wysyła żądanie otworzenia drzwiczek.
4. Drzwiczki dla psa otwierają się.
5. Azor wychodzi na zewnątrz.
6. Azor załatwia swoje potrzeby.
   - 6.1 Drzwi zamykają się automatycznie.
   - 6.2. Azor szczeka, by właściciele wpuścili go do domu.
   - 6.3. System rozpoznawania dźwięków „słyszy" szczekanie Azora (ponownie).
   - 6.4. System rozpoznawania dźwięków wysyła żądanie otworzenia drzwiczek.
   - 6.5. Drzwiczki dla psa otwierają się (znowu).
7. Azor wraca z powrotem.
8. Drzwi automatycznie się zamykają.

Ścieżki alternatywne

- 2.1. Tadek lub Janka słyszą, że Azor szczeka.
- 3.1. Tadek lub Janka naciskają przycisk na pilocie.
- 6.3.1. Tadek lub Janka słyszą szczekanie Azora (znowu).
- 6.4.1. Tadek lub Janka naciskają przycisk na pilocie.

To jest ścieżka główna przypadku użycia.

Te dwa scenariusze nie wykorzystują ścieżki alternatywnej reprezentującej sytuację, gdy Azor zostaje na zewnątrz po zamknięciu drzwiczek.

1. 1, 2.1, 3.1, 4, 5, 6.1, 6.2, 6.3.1, 6.4.1, 6.5, 7, 8
2. 1, 2, 3, 4, 5, 6, 7, 8
3. 1, 2.1, 3.1, 4, 5, 6, 7, 8
4. 1, 2.1, 3.1, 4, 5, 6, 6.1, 6.2, 6.3, 6.4, 6.5, 7, 8

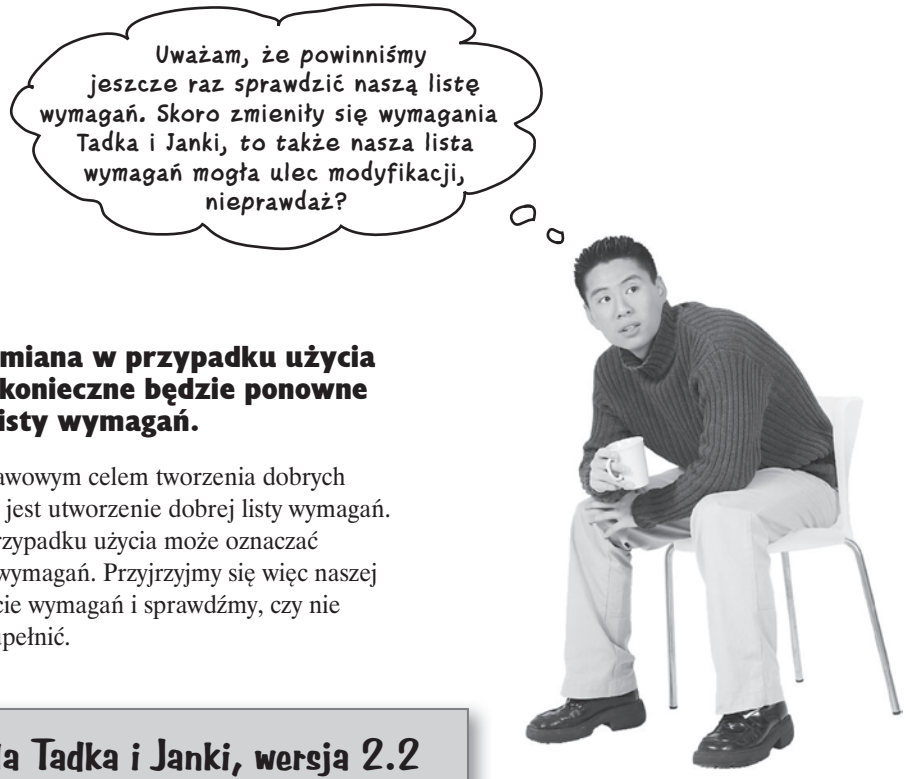
Jeśli wykonasz krok 2.1, to zawsze będziesz musiał wykonać także krok 3.1.

Jeśli wykonasz krok 6.3.1, to będziesz także musiał wykonać krok 6.4.1.

5. 1, 2, 3, 4, 5, 6, 6.1, 6.2, 6.3.1, 6.4.1, 6.5, 7, 8
6. 1, 2.1, 3.1, 4, 5, 6, 6.1, 6.2, 6.3.1, 6.4.1, 6.5, 7, 8
7. 1, 2, 3, 4, 5, 6, 6.1, 6.2, 6.3, 6.4, 6.5, 7, 8
8. <ten punkt jest pusty>

Przygotujmy się do kodowania...

Teraz, kiedy dokończyliśmy prace nad przypadkiem użycia i określiliśmy wszystkie możliwe scenariusze korzystania z drzwiczek dla psa, jesteśmy już gotowi, by napisać kod, który spełni wszystkie nowe wymagania Tadka i Janki. Zastanówmy się, co ten kod powinien robić...



Jakakolwiek zmiana w przypadku użycia powoduje, że konieczne będzie ponowne sprawdzenie listy wymagań.

Pamiętaj, że podstawowym celem tworzenia dobrych przypadków użycia jest utworzenie dobrej listy wymagań. A zatem zmiana przypadku użycia może oznaczać także zmianę listy wymagań. Przyjrzyjmy się więc naszej dotychczasowej liście wymagań i sprawdźmy, czy nie powinniśmy jej uzupełnić.

Drzwiczki dla psa, dla Tadka i Janki, wersja 2.2

Lista wymagań

1. Górna krawędź otworu drzwiczek musi być umieszczona co najmniej na wysokości 30 centymetrów.
2. Naciśnięcie przycisku na pilocie powoduje otworzenie drzwiczek, jeśli te są zamknięte, lub ich zamknięcie, jeśli są otwarte.
3. Po otworzeniu drzwiczek powinny one zostać automatycznie zamknięte, jeśli wcześniej nie zostaną zamknięte przez użytkownika.

Dopisz tu wszelkie dodatkowe wymagania, które określiłeś podczas analizy różnych scenariuszy działania drzwiczek dla psa, przedstawionych na stronie 154.

Uzupełnienie listy wymagań

A zatem musimy obsłużyć kilka nowych ścieżek alternatywnych, co będzie wymagało dodania do naszej dotychczasowej listy wymagań kilku nowych punktów. Z przedstawionego poniżej przypadku użycia wykreśliliśmy wszystkie punkty, które już są obsługiwane przez wymagania znajdujące się na liście. Wygląda na to, że będziemy musieli dodać do listy wymagań kilka nowych punktów.

Drzwiczki dla psa, dla Tadka i Janki, wersja 2.3

Jak działają drzwiczki

Ścieżka główna

~~1. Azor szczeka, by właściciele wypuścili go na spacer.~~

2. System rozpoznawania dźwięków „słyszy" szczekanie Azora.
3. System rozpoznawania dźwięków wysyła żądanie otworzenia drzwiczek.

~~4. Drzwiczki dla psa otwierają się.~~
~~5. Azor wychodzi na zewnątrz.~~
~~6. Azor załatwia swoje potrzeby.~~
~~6.1 Drzwi zamykają się automatycznie.~~
~~6.2. Azor szczeka, by właściciele wpuścili go do domu.~~
6.3. System rozpoznawania dźwięków „słyszy" szczekanie Azora (ponownie).
6.4. System rozpoznawania dźwięków wysyła żądanie otworzenia drzwiczek.
~~6.5. Drzwiczki dla psa otwierają się (znowu).~~
~~7. Azor wraca z powrotem.~~
~~8. Drzwi automatycznie się zamykają.~~

Ścieżki alternatywne

~~2.1. Tadek lub Janka słyszą, że Azor szczeka.~~
~~3.1. Tadek lub Janka naciskają przycisk na pilocie.~~

~~6.3.1. Tadek lub Janka słyszą szczekanie Azora (znowu).~~
~~6.4.1. Tadek lub Janka naciskają przycisk na pilocie.~~

Te punkty wiążą się tak naprawdę z dwoma wymaganiami: „usłyszeniem" szczekania oraz otworzeniem drzwiczek dla psa.

To są zupełnie inne kroki niż 2. i 3., jednak wiążą się z nimi dokładnie takie same wymagania.

Wymagania związane z większością kroków znajdujących się na ścieżce głównej określiliśmy już w poprzednim rozdziale.

Pamiętaj, że te kroki, znajdujące się obecnie na ścieżce alternatywnej, w przypadku użycia przedstawionym w poprzednim rozdziale znajdowały się na ścieżce głównej...

... dlatego związane z nimi wymagania już są na naszej liście.

Drzwiczki dla psa, dla Tadka i Janki, wersja 2.3

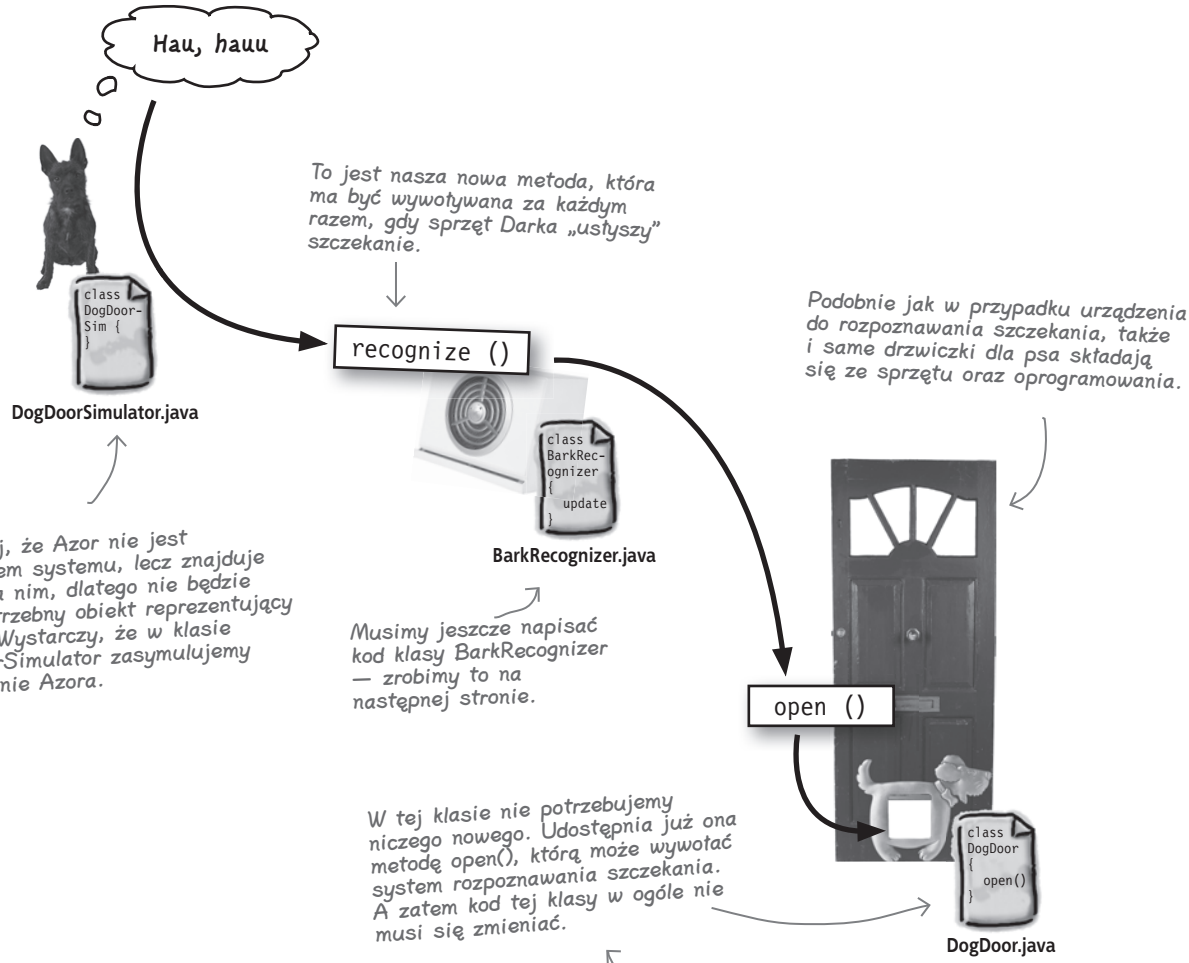
Lista wymagań

1. Górna krawędź otworu drzwiczek musi być umieszczona co najmniej na wysokości 30 centymetrów.
2. Naciśnięcie przycisku na pilocie powoduje otworzenie drzwiczek, jeśli te są zamknięte, lub ich zamknięcie, jeśli są otwarte.
3. Po otworzeniu drzwiczek, powinny one zostać automatycznie zamknięte, jeśli wcześniej nie zostaną zamknięte przez użytkownika.
4. System rozpoznawania dźwięków musi być w stanie określić, kiedy pies szczeka.
5. System rozpoznawania dźwięków musi być w stanie otworzyć drzwiczki, kiedy pies zacznie szczekać.

Oto dwa nowe wymagania, które musimy dodać do listy.

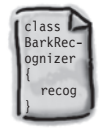
W końcu ponownie możemy zacząć pisać kod obsługujący drzwiczki

Wraz z nowymi wymaganiami musi się pojawić nowy kod. Potrzebne nam będzie szczekanie Azora, system rozpoznawania dźwięków, który będzie nasłuchiwać i rozpoznawać to szczekanie oraz w odpowiedzi na nie otwierać drzwiczki.



Czy nie słyszałem jakiegoś „hau"?

Potrzebujemy pewnego kodu, który mógłby zostać wykonany, kiedy sprzęt Darka „usłyszy" szczekanie. Utwórzmy zatem klasę **BarkRecognizer**, a w niej metodę, która będzie odpowiadać na szczekanie.



BarkRecognizer.java

```
public class BarkRecognizer {

  private DogDoor door;

  public BarkRecognizer(DogDoor door) {
    this.door = door;
  }

  public void recognize(String bark) {
    System.out.println("   BarkRecognizer: Usłyszano '" +
        bark + "'");
    door.open();
  }
}
```

W tej zmiennej przechowamy obiekt drzwiczek, z którymi będzie współpracować ten egzemplarz systemu rozpoznawania szczekania.

Konstruktor klasy BarkRecognizer musi wiedzieć, jakie drzwiczki należy otwierać.

Za każdym razem gdy sprzęt usłyszy szczekanie, wywoła tę metodę, przekazując do niej usłyszane dźwięki.

W tym przypadku musimy jedynie wyświetlić komunikat, by system wiedział, że zarejestrowaliśmy szczekanie...

...a następnie otworzyć drzwiczki dla psa.

Nie istnieją głupie pytania

P: Tylko tyle? Faktycznie, wygląda na to, że klasa BarkRecognizer nie ma zbyt wiele do roboty.

O: W obecnej chwili faktycznie nie ma. Wymagania aplikacji są bardzo proste — jak usłyszysz szczekanie, otwórz drzwiczki — więc także kod jest prosty. Za każdym razem gdy układy sprzętowe systemu rozpoznawania dźwięku usłyszą szczekanie, zostanie wywołana metoda **recognize()** klasy **BarkRecognizer**, która z kolei otworzy drzwiczki. Pamiętaj, by starać się zachować jak największą prostotę: nie komplikuj rozwiązania, jeśli nie jest to konieczne.

P: Ale co się stanie, jeśli zacznie szczekać inny pies niż Azor? Czy przed otworzeniem drzwiczek klasa BarkRecognizer nie powinna upewnić się, że szczeka Azor, a nie jakieś inne zwierzę?

O: Bardzo interesujące pytanie! Klasa **BarkRecognizer** słyszy *każde* szczekanie, ale my raczej nie chcemy, by otwierała drzwiczki, wpuszczając do domu *każdego* psa, nieprawdaż? Może wrócimy do tego problemu i rozwiążemy go później. A może powinieneś dokładniej wszystko przemyśleć podczas testowania systemu?

Myślę, że po dodaniu tej nowej klasy BarkRecognizer mamy już wszystko, czego nam potrzeba. Przetestujmy ją i sprawdźmy, czy będziemy w stanie ponownie uszczęśliwić Tadka i Jankę.

W pierwszej kolejności upewnijmy się, czy zadbaliśmy o spełnienie wszystkich wymagań, jakie Tadek i Janka postawili przed nową wersją drzwiczek dla psa.

Drzwiczki dla psa, dla Tadka i Janki, wersja 2.3

Lista wymagań

1. Górna krawędź otworu drzwiczek musi być umieszczona co najmniej na wysokości 30 centymetrów.
2. Naciśnięcie przycisku na pilocie powoduje otworzenie drzwiczek, jeśli te są zamknięte, lub ich zamknięcie, jeśli są otwarte.
3. Po otworzeniu drzwiczek powinny one zostać automatycznie zamknięte, jeśli wcześniej nie zostaną zamknięte przez użytkownika.
4. **System rozpoznawania dźwięków musi być w stanie określić, kiedy pies szczeka.**
5. **System rozpoznawania dźwięków musi być w stanie otworzyć drzwiczki, kiedy pies zacznie szczekać.**

To jest wymaganie sprzętowe przeznaczone raczej dla Darka. Na razie możemy użyć symulatora, by wygenerować szczekanie, które system rozpoznawania dźwięku mógłby wykryć. Takie rozwiązanie pozwoli nam przetestować nową wersję systemu.

To jest kod, który właśnie napisaliśmy... Za każdym razem gdy system rozpoznawania dźwięku usłyszy szczekanie, otworzy drzwiczki.

Hmm... tak naprawdę, to nasz system rozpoznawania dźwięku nie „rozpoznaje" szczekania, nieprawdaż? Otwiera drzwiczki po usłyszeniu KAŻDEGO szczekania. Być może później trzeba będzie zająć się tym problemem.

Podłączamy nowe drzwiczki do prądu

Oto efekt finalny napisania nowego przypadku użycia i nowego kodu. Sprawdźmy, czy wszystko działa tak, jak powinno.

DogDoorSimulator.java

1 Aktualizacja kodu źródłowego klasy DogDoorSimulator:

```
public class DogDoorSimulator {

  public static void main(String[] args) {
    DogDoor door = new DogDoor();
    BarkRecognizer recognizer = new BarkRecognizer(door);
    Remote remote = new Remote(door);

    // Symulujemy, że system sprzętowy słyszy szczekanie
    System.out.println("Azor szczeka, by wyjść na zewnątrz...");
    recognizer.recognize("Hau");

    System.out.println("\nAzor wyszedł na zewnątrz...");
    System.out.println("\nAzor załatwił swoje potrzeby...");

    try {
      Thread.currentThread().sleep(10000);
    } catch (InterruptedException e) { }

    System.out.println("...ale był na zewnątrz zbyt długo!");

    // Symulujemy, że system sprzętowy słyszy szczekanie (ponownie)
    System.out.println("\nAzor zaczyna szczekać...");
    recognizer.recognize("Hau");

    System.out.println("\nAzor z powrotem wszedł do domu...");
  }
}
```

Tworzymy obiekt BarkRecognizer, kojarzymy z obiektem drzwiczek dla psa i pozwalamy na nasłuchiwanie szczekania.

Nie dysponujemy faktycznym sprzętem, zatem jedynie symulujemy, że sprzęt usłyszy i rozpozna szczekanie*.

To właśnie w tym miejscu nasz nowy obiekt BarkRecognizer wkracza do akcji.

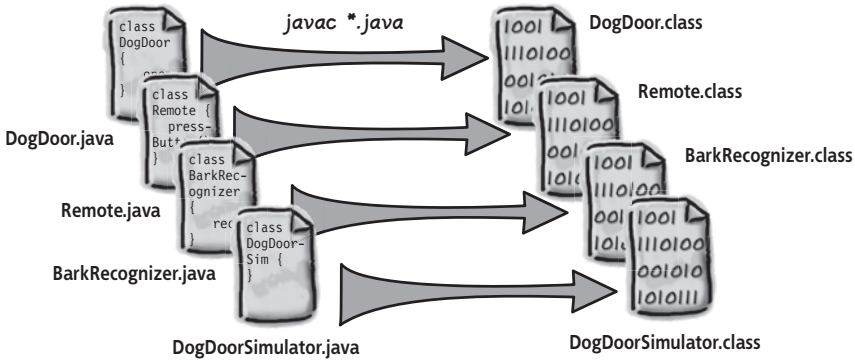
W tym miejscu symulujemy upływ dłuższego okresu czasu.

Testujemy proces, w którym Azor pozostaje dłużej na zewnątrz; chcemy bowiem upewnić się, że wszystko działa zgodnie z naszymi oczekiwaniami.

Zauważ, że ani Tadek, ani Janka nie muszą już naciskać przycisku na pilocie.

*Autorzy niniejszej książki naprawdę chcieli dołączyć do niej odpowiednie urządzenie sprzętowe, które byłoby w stanie usłyszeć szczekanie psa... Jednak goście z działu marketingu upierali się, że nikt by nie kupił tej książki za cenę 900 zł. Ciekawe, czy mieli rację!

❷ Ponownie skompiluj wszystkie pliki źródłowe aplikacji.



❸ Uruchom aplikację i obserwuj, jak drzwiczki dla psa działają bez żadnej interwencji ze strony człowieka.

```
Wiersz polecenia

T:\>java FindGuitarTester
Azor szczeka, by wyjść na zewnątrz...
   BarkRecognizer: Usłyszano 'Hau'
Drzwiczki otwierają się.

Azor wyszedł na zewnątrz...

Azor załatwił swoje potrzeby...
...ale był na zewnątrz zbyt długo!

Azor zaczyna szczekać...
   BarkRecognizer: Usłyszano 'Hau'
Drzwiczki otwierają się.

Azor z powrotem wszedł do domu...

T:\>
```

Pomiędzy pojawieniem się tych napisów mija kilka sekund, w czasie których Azor załatwia swoje potrzeby.

WYSIL SZARE KOMÓRKI

W naszym kodzie występuje pewien poważny problem, który uwidocznił się po uruchomieniu symulatora. Czy potrafisz go wskazać? Co byś zrobił, by go rozwiązać?

Zaostrz ołówek

Który scenariusz testujemy?

Czy jesteś w stanie określić, który scenariusz naszego przypadku użycia aktualnie testujemy? Zapisz wszystkie kroki wykonywane przez symulator (zajrzyj na stronę 149, by przypomnieć sobie, jak wygląda nasz aktualnie używany przypadek użycia).

Zaostrz ołówek

Rozwiązanie Który scenariusz testujemy?

Czy jesteś w stanie określić, który scenariusz naszego przypadku użycia aktualnie testujemy? Oto wykonane przez nas kroki przypadku użycia przedstawionego na stronie 161:
1, 2, 3, 4, 5, 6, 6.1, 6.2, 6.3, 6.4, 5.6, 7, 8

WYSIL SZARE KOMÓRKI

W naszym kodzie występuje pewien poważny problem, który uwidocznił się po uruchomieniu symulatora. Czy potrafisz go wskazać? Co byś zrobił, by go rozwiązać?

Czy określiłeś, co było nie w porządku z ostatnią wersją naszego systemu?

W naszej nowej wersji systemu drzwiczki nie zamykają się automatycznie!

Poniżej przedstawiliśmy fragment kodu z poprzedniej wersji systemu — w którym drzwiczki otwierało naciśnięcie przycisku — odpowiadający za automatyczne zamykanie drzwiczek po upływie określonego czasu:

```
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);
  }
}
```

Kiedy Tadek lub Janka nacisną przycisk na pilocie, to ten fragment kodu uruchamia także licznik czasu, który automatycznie zamknie drzwiczki.

Pamiętasz zapewne, że ten licznik czasu czeka 5 sekund, a następnie wysyła żądanie zamknięcia drzwiczek.

Remote.java

Ale w klasie **BarkRecognizer** otwieramy drzwiczki i nigdy ich nie zamykamy:

```
public void recognize(String bark) {
  System.out.println(„   BarkRecognizer:
Usłyszano '" +
      bark + "'");
  door.open();
}
```

Otwieramy drzwiczki, lecz ich nie zamykamy.

BarkRecognizer.java

Darek, szef firmy PsieOdrzwia, zdecydował, że dokładnie wie, co masz zrobić.

Nawet ja to potrafię wymyślić. Wystarczy dodać do klasy BarkRecognizer licznik czasu, — taki sam, jakiego użyliśmy w klasie Remote. To powinno wystarczyć, by system ponownie zaczął działać poprawnie. W końcu wiesz... Tadek i Janka czekają!

A co TY myślisz o pomyśle Darka?

Według mnie Darek to lamer. Nie mam zamiaru umieszczać tego samego kodu w dwóch miejscach — w klasie obsługującej pilota i systemie rozpoznawania dźwięków.

Powielanie kodu jest bardzo złym pomysłem. Ale w takim razie, gdzie należałoby umieścić fragment kodu odpowiadający za automatyczne zamykanie drzwi?

Cóż, za zamykanie drzwiczek powinny chyba odpowiadać same drzwiczki, a nie jakiś pilot lub system rozpoznawania dźwięków. Dlaczego zatem nie umieścimy tego kodu w klasie DogDoor?

Niech zatem drzwiczki zawsze zamykają się automatycznie.

Ponieważ Janka nie chce, by drzwiczki dla psa w ogóle zostawały otwarte, możemy je *zawsze* zamykać automatycznie. A zatem możemy przenieść kod z licznikiem odpowiadającym za automatyczne zamykanie drzwiczek do klasy **DogDoor**. Dzięki temu niezależnie od tego, *kto lub co* otworzy drzwiczki, zawsze zamkną się one automatycznie.

Choć jest to decyzja projektowa, to jednak możemy ją zaliczyć do grupy zadań związanych z doprowadzeniem oprogramowania do takiego stanu, by działało zgodnie z oczekiwaniami klienta. Pamiętaj, że nie ma niczego złego w używaniu dobrego projektu podczas prac nad funkcjonalnością systemu.

Aktualizacja klasy drzwiczek

DogDoor.java

A zatem skopiujmy kod odpowiadający za automatyczne zamykanie drzwiczek z klasy **Remote** i przenieśmy go do klasy **DogDoor**:

```
public void open() {
    System.out.println("Drzwiczki otwierają się.");
    open = true;

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

  public void close() {
    System.out.println("Drzwiczki zamykają się.");
    open = false;
  }
}
```

To jest dokładnie ten sam kod, który wcześniej był używany w klasie Remote.java.

Teraz drzwi zamykają się same... nawet jeśli dodamy nowe urządzenia, które będą mogły je otwierać. Super!

Nie możesz zapomnieć o dodaniu instrukcji importujących klasy java.util.Timer oraz java.util.TimerTask.

Uproszczenie kodu obsługującego pilota

Teraz musisz usunąć kod odpowiadający za automatyczne zamknięcie z klasy **Remote**, gdyż te możliwości funkcjonalne zostały przeniesione do klasy **DogDoor**.

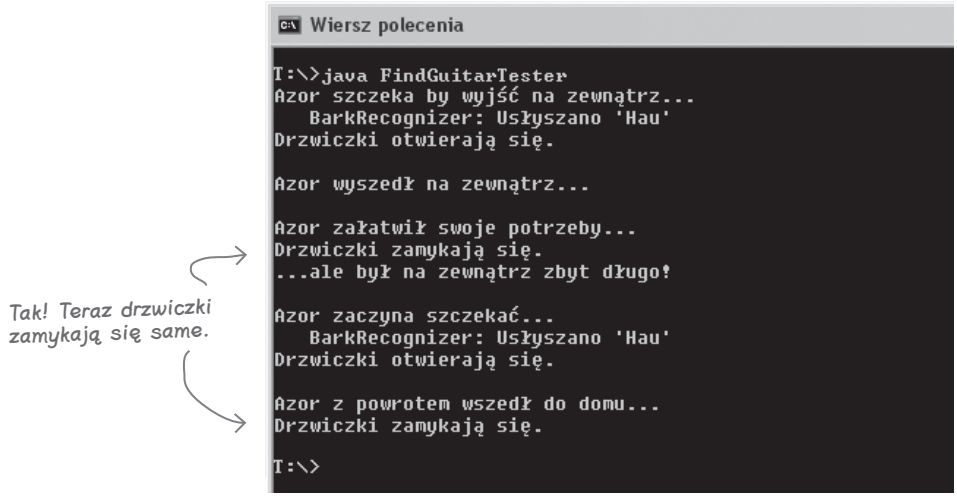
```
public void pressButton() {
  System.out.println("Naciśnięto przycisk na pilocie...");
  if (door.isOpen()) {
    door.close();
  } else {
    door.open();

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

Remote.java

Ostateczny test drzwiczek

Wprowadziłeś naprawdę sporo zmian do drzwiczek dla psa zamówionych przez Tadka i Jankę. Nadszedł czas, by przetestować, czy wszystko działa, jak należy. Wprowadź ostatnie zmiany w plikach **`Remote.java`** oraz **`DogDoor.java`**, tak by drzwiczki zamykały się automatycznie, skompiluj wszystkie pliki źródłowe, po czym uruchom symulator:



WYSIL SZARE KOMÓRKI

A co by się stało, gdyby Tadek i Janka zdecydowali, że chcą drzwiczek, w których okres poprzedzający automatyczne zamknięcie drzwiczek jest dłuższy? Albo krótszy? Sprawdź, czy potrafisz wyobrazić sobie, w jaki sposób należałoby zmienić klasę DogDoor, tak by to klient mógł określać ilość czasu, po jakim drzwiczki mają się zamknąć.

Czasami zmiana wymagań może uwidocznić problemy występujące w systemie, których istnienia nawet nie podejrzewałeś.

Zmiany są pewnikiem, a Twój system powinien być coraz lepszy, ilekroć go poprawiasz.

Zaostrz ołówek

Napisz swoją własną zasadę projektową!

W tym rozdziale wykorzystałeś bardzo ważną zasadę projektową, związaną z powielającym się kodem i samoczynnym zamykaniem się drzwiczek dla psa. Zastanów się, co to mogła być za zasada, i postaraj się ją podsumować w kilku słowach.

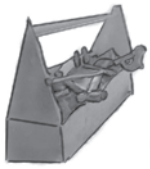
Zasada projektowa

_____

_____

_____

W tym rozdziale nie znajdziesz odpowiedzi na tę zagadkę, jednak mamy zamiar wrócić do tego problemu nieco później. Pomimo to spróbuj zgadnąć!



Kolejne narzędzia do naszego projektanckiego przybornika

W tym rozdziale zdobyłeś naprawdę sporo nowej wiedzy, a teraz nadszedł czas, by dodać zgromadzone informacje do Twojego projektanckiego przybornika. Przyjrzyj się wszystkim informacjom zamieszczonym na tej stronie i dobrze je zapamiętaj.

Wymagania

Dobre wymagania gwarantują, że system będzie działał zgodnie z oczekiwaniami klienta.

Upewnij się, że wymagania obejmują wszystkie kroki przypadku użycia opracowanego dla tworzonego systemu.

Wykorzystaj przypadki użycia, by dowiedzieć się o wszystkich rzeczach, o których klient zapomniał Ci powiedzieć.

Przypadki użycia ujawnią wszystkie niekompletne lub brakujące wymagania, które zapewne będziesz musiał dodać do tworzonego systemu.

Wraz z upływem czasu Twoje wymagania zawsze będą się zmieniać (i powiększać).

W tym rozdziale poznałeś tylko jedną nową zasadę związaną z wymaganiami, niemniej jednak jest to bardzo ważna zasada!

Zasady projektowania obiektowego

Hermetyzuj to, co się zmienia.

Hermetyzacja pozwala nam uświadomić sobie, że drzwiczki dla psa same powinny obsługiwać własne zamykanie. Oddzieliliśmy zachowanie drzwi od reszty kodu aplikacji.

CELNE SPOSTRZEŻENIA

- Wraz z postępem prac nad projektem wymagania zawsze będą się **zmieniać**.
- Wraz ze zmianami wymagań system musi ewoluować tak, by spełniać nowe wymagania.
- Jeśli system będzie musiał pracować w nowy lub zmieniony sposób, zacznij od aktualizacji przypadku użycia.
- **Scenariusz** jest konkretną ścieżką pozwalającą na przejście całego przypadku użycia od jego początku aż do końca.
- Jeden przypadek użycia może posiadać więcej scenariuszy, o ile tylko każdy z nich będzie spełniać ten sam cel klienta.
- **Ścieżki alternatywne** mogą składać się z kroków wykonywanych tylko czasami, mogą także pozwalać na przejście fragmentów przypadku użycia w całkowicie odmienny sposób.
- Jeśli pewien krok przypadku użycia jest opcjonalny bądź jeśli stanowi alternatywną ścieżkę przejścia przez przypadek użycia, to do jego oznaczenia należy użyć podpunktów, np.: 3.1, 4.1 i 5.1 lub 2.1.1, 2.2.1 i 2.3.1.
- Niemal zawsze należy starać się unikać **powielania kodu**. Może ono bowiem ogromnie utrudnić utrzymanie i rozwijanie oprogramowania, a samo jego wystąpienie sygnalizuje problemy lub usterki w projekcie aplikacji.