helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

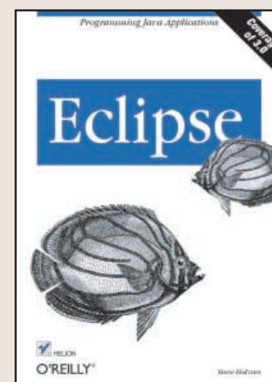
TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

Eclipse



Autor: Steve Holzner
Tłumaczenie: Adam Bochenek
ISBN: 83-7361-648-9
Tytuł oryginału: Eclipse
Format: B5, stron: 304
Przykłady na ftp: 2031 kB

Poznaj możliwości Eclipse

Eclipse to dostępne nieodpłatnie zintegrowane środowisko programistyczne do tworzenia aplikacji w języku Java. W rozwój projektu Eclipse zaangażowało się wiele firm informatycznych. Zaowocowało to powstaniem doskonałego narzędzia, którego zalety docenią zarówno początkujący, jak i doświadczeni programiści. Bogactwo funkcji, możliwość integracji z narzędziem Ant i kontenerem aplikacji Tomcat oraz współpraca z systemem CVS to tylko niektóre zalety środowiska Eclipse, wpływające na ciągły wzrost jego popularności.

Książka „Eclipse" to przewodnik po tym środowisku. Przedstawia wszystkie wiadomości związane z wykorzystaniem środowiska Eclipse w procesie tworzenia aplikacji w języku Java. Polecenia menu, tworzenie kodu źródłowego, kompilacja z wykorzystaniem narzędzia Ant, zastosowanie zewnętrznych modułów i wiele innych rzeczy — wszystko to opisano w przykładach na stronach niniejszej książki.

- Podstawowe wiadomości o Eclipse
- Tworzenie programów w środowisku Eclipse
- Testowanie i usuwanie błędów
- Praca zespołowa z wykorzystaniem środowiska CVS
- Korzystanie z narzędzia Ant
- Tworzenie interfejsów użytkownika
- Korzystanie z biblioteki SWT
- Środowisko Struts
- Tworzenie modułów rozszerzających Eclipse

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Wstęp ... 7

1. Podstawy Eclipse ... 13
Eclipse i Java 13
Jak wejść w posiadanie Eclipse? 16
Architektura Eclipse 18
Widoki i perspektywy 21
Pierwszy program 23
Quick Fix 33
Słowo o zarządzaniu projektami 36

2. Tworzenie aplikacji w języku Java ... 39
Podstawy kodowania 39
Kompilacja i uruchamianie 49
Tworzenie dokumentacji Javadoc 56
Refaktoring 57
Inne cechy środowiska 63
Konfiguracja środowiska Eclipse 66

3. Testowanie i debugowanie ... 73
Czym jest JUnit? 73
Debugowanie 82

4. Praca zespołowa ... 101
Jak działa system kontroli wersji? 101
System CVS 101
Instalacja serwera CVS 103
Dodanie projektu do repozytorium CVS 104

5. Tworzenie projektów przy użyciu narzędzia Ant ... 123
Jak działa Ant? 123
Tworzenie archiwum JAR 126
Konfiguracja Anta w ramach Eclipse 131
Obsługa błędów w skryptach 135

6. Programowanie interfejsu użytkownika — od apletów po Swing 137
Aplikacje AWT 140
Aplikacje Swing 142
Instalacja wtyczki Eclipse 148
Wtyczka V4ALL 149

7. Biblioteka SWT — część pierwsza.. 155
Graficzny interfejs użytkownika w Javie 155
SWT — pierwszy przykład 156
Obsługa zdarzeń 163
Menedżery układu 167
Listy 169
Edytor V4ALL i SWT 171

8. Biblioteka SWT — część druga 175
Menu 175
Listwy narzędziowe 180
Suwaki 183
Drzewa 187
Okna dialogowe 189
Korzystanie z Internet Explorera w ramach okna SWT 193

9. Tworzenie aplikacji sieciowych 197
Serwer Tomcat — instalacja 197
Pierwsza strona JSP 199
Pierwszy serwlet 201
Serwlet w katalogu serwera 204
Współpraca z komponentami JavaBeans 207
Wtyczka Sysdeo Tomcat 209
Dystrybucja aplikacji sieciowych 216

10. Eclipse i Struts... 219
Przykład aplikacji Struts 219
Widok 222
Kontroler 225
Model 228
Wtyczka Easy Struts 232

11. Tworzenie wtyczek — Plug-in Development Environment 239
Niezbędny plik plugin.xml 240
Środowisko tworzenia wtyczek (PDE) 241
Platforma testowa 245
Wtyczka typu Standard Plug-in 247

12. Tworzenie wtyczek — edytory i widoki 257
Edytor wielostronicowy 257
Tworzenie widoku 265
Dystrybucja wtyczki 270

13. Co nowego w Eclipse 3.0? 273
Pierwszy rzut oka 273
Tworzenie projektu 273
Zmiany dotyczące platformy Eclipse 279
Zmiany w JDT 282
Inne modyfikacje 288

Skorowidz 289

ROZDZIAŁ 1.

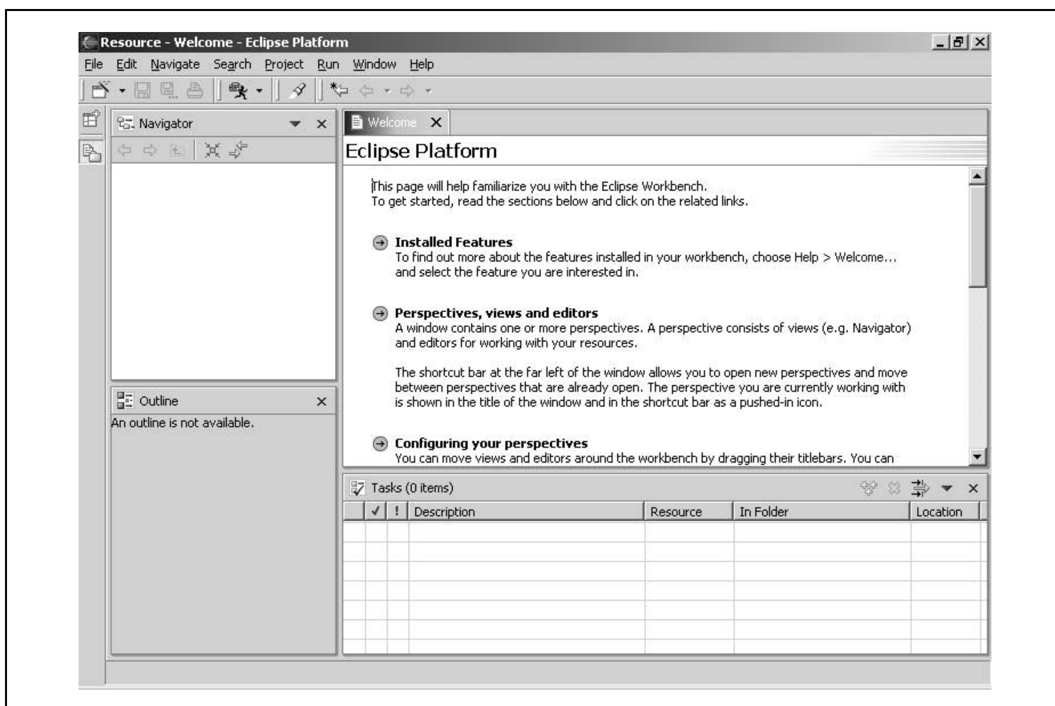
Podstawy Eclipse

Zakładamy, że sięgnąłeś po tę książkę, gdyż jesteś programistą znającym język Java. W takim razie, oprócz wielu zalet, doświadczyłeś najprawdopodobniej również pewnych drobnych uciążliwości, które skutecznie potrafią uprzykrzać nam życie, gdy tworzymy programy w tym niewątpliwie doskonałym języku. Chodzi tu o pomijanie pakietów, które chcemy zaimportować, zapominanie o deklaracji zmiennych, przeoczone średniki, drobne błędy syntaktyczne czy wręcz „literówki". Słowem wszystko co powoduje, że standardowy kompilator wywoływany z poziomu wiersza poleceń, czyli `javac`, niepokoi nas różnego rodzaju komunikatami o błędach. Drażni nas, że są to najczęściej drobne pomyłki i zastanawiamy się, dlaczego kompilator nie jest w stanie sam ich zneutralizować. Skoro brakuje np. średnika, dlaczego sam `javac` nie może go dodać, tylko przerywa nam proces twórczej pracy, komunikując o nieistotnych drobiazgach?

Sam kompilator `javac` nie jest w stanie nam pomóc, nie wymagajmy od niego funkcjonalności edytora. Ale ekrany wypełnione po brzegi komunikatami o błędach składniowych nie są przyjemnym doświadczeniem. Pozostawiają bowiem uczucie, że nakłada się na nas zbyt duże restrykcje. By to zmienić, należy skorzystać ze zintegrowanego środowiska projektowego typu IDE (ang. *Integrated Development Environment*), które nie tylko zidentyfikuje i poinformuje nas o potencjalnych błędach jeszcze przed przystąpieniem do kompilacji, ale również zasugeruje sposób ich poprawienia. Java jako język zdecydowanie wymaga środowiska IDE, nic więc dziwnego, że na rynku istnieje bogata oferta aplikacji tego typu. Spośród nich na pozycję lidera wysuwa się bohater naszej książki — Eclipse. Zaraz po uruchomieniu wygląda tak jak na rysunku 1.1.

Eclipse i Java

Chociaż Eclipse traktować możemy jak uniwersalne środowisko IDE do tworzenia aplikacji w wielu różnych językach programowania — takich jak np. C/C++ czy nawet Cobol — najczęściej kojarzymy je Javą. Co więcej, obsługa Javy jest w nie wbudowana. Mamy więc do czynienia z aplikacją, która powstała jako *uniwersalna platforma* do tworzenia oprogramowania i w praktyce wywiązuje się z tego zadania znakomicie. Zajmiemy się jednak połączeniem z punktu widzenia Eclipse najbardziej naturalnym i jednocześnie wśród programistów najpopularniejszym: Eclipse + Java.



Rysunek 1.1. Okno główne Eclipse zaraz po uruchomieniu

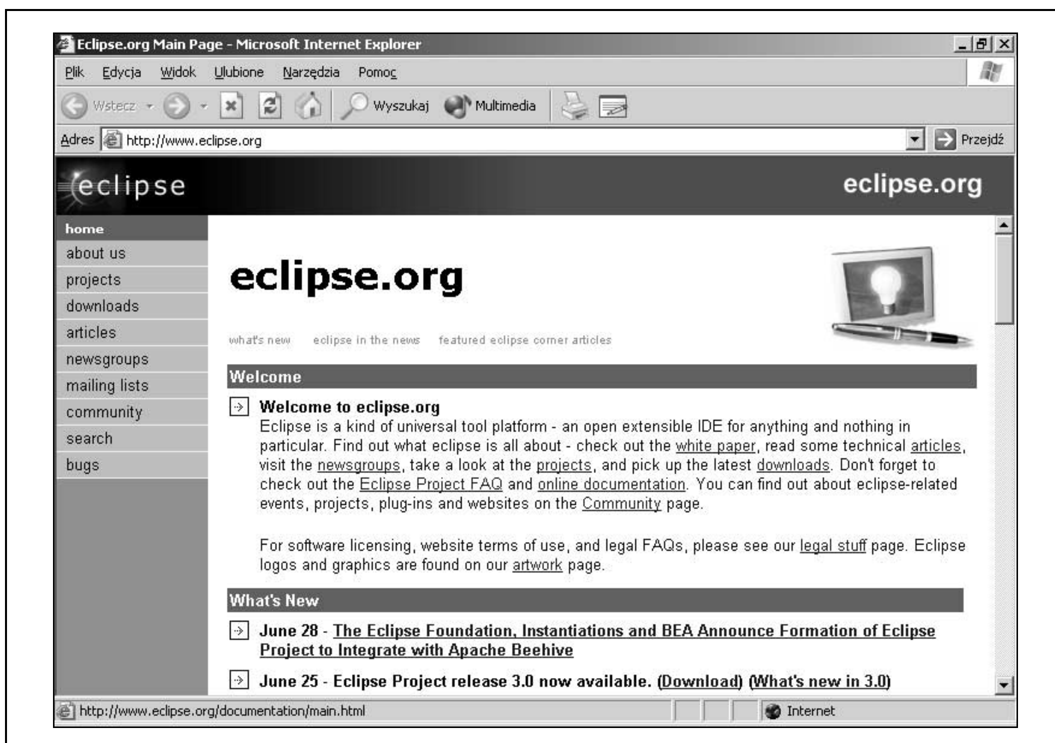
Magia Eclipse bierze się stąd, że niweluje większość niedogodności pisania w Javie w sposób, o jakim do niedawna mogliśmy tylko marzyć. Błędy, o które potykał się kompilator *javac*, wychwytywane są, zanim jeszcze pomyślimy o kompilacji, co więcej, jesteśmy informowani o najwłaściwszym sposobie wybrnięcia z kłopotu. Wszystko, co w takiej chwili robimy, sprowadza się do wskazania kursorem myszy odpowiedniego miejsca i kliknięcia — niczego więcej się od nas nie wymaga. Większość programistów Javy bardzo docenia takie rozwiązanie.

Odrobina historii

Eclipse to środowisko, które można pobrać i używać za darmo. Ktoś powie: podobnie jak wiele innych środowisk typu IDE dla Javy. Ale Eclipse ma nad nimi znaczną przewagę — zapewnia ją potęga firmy IBM, która podobno wydała na jego rozwój 40 milionów dolarów. Pierwsza wersja o numerze 1.0 ujrzała światło dzienne w listopadzie 2001 roku i stopniowo zaczęła zdobywać popularność (chociaż — jak w przypadku wielu narzędzi przeznaczonych dla programistów — nie obyło się bez usterek i dyskusji na ich temat).

Od tego czasu Eclipse dorosło i rozwinęło się, wersja opisywana w niniejszej książce ma numer 2.1.1 i jest narzędziem zbierającym wiele pochwał. Największy przełom nastąpił wraz z opublikowaniem wersji 2.1. Przez pierwszych kilka dni od momentu premiery serwer o adresie *http://www.eclipse.org* był tak obciążony, że pobranie kopii graniczyło z cudem.

Eclipse to projekt zapoczątkowany przez firmę IBM. Obecnie ma on status open source, ciągle jednak pozostaje pod dużym wpływem programistów tej firmy. Oficjalnie jednak jest produktem konsorcjum o nazwie „eclipse.org". Główna jego strona znajduje się pod adresem *http://www.eclipse.org*, widzimy ją na rysunku 1.2.



Rysunek 1.2. Strona główna konsorcjum eclipse.org

Początkowo konsorcjum Eclipse składało się z zależnej od IBM firmy Object Technologies International (OTI) — pierwszego twórcy Eclipse — a także firm: Borland, IBM, MERANT, QNX Software Systems, Rational Software3, Red Hat, SuSE, TogetherSoft3 i Webgain2. Taki był stan w listopadzie roku 2001. Obecnie jest to w sumie 45 przedsiębiorstw, włączając w to firmy Sybase, Hitachi, Oracle, Hewlett-Packard, Intel i wiele innych.

OTI nie jest nowym graczem na rynku narzędzi dla programistów, choć nazwa ta nie jest wyjątkowo popularna. Firma zapoczątkowała bardzo znaną i cenioną linię produktów o nazwie IBM Visual Age (IBM wchłonął OTI w roku 1996). Jakiś czas temu OTI stworzyło narzędzie do programowania w Javie napisane w języku Smalltalk. Znamy je jako Visual Age for Java (w skrócie VA4J) — rynek zareagował na ten produkt bardzo przychylnie. Można w pewnym przybliżeniu powiedzieć, że Eclipse to VA4J napisane od nowa w Javie. Usunięto pewne cechy pierwotnego środowiska, które uważane były za zbyt specyficzne, dodając w ich miejsce wiele nowych funkcji. Prawdziwym będzie więc stwierdzenie, że Eclipse jest narzędziem stosunkowo nowym, ale o bardzo bogatym rodowodzie.

Projekt Eclipse podzielony jest na trzy podprojekty:

- Właściwa platforma Eclipse stanowiąca filar całej aplikacji.
- Java Development Toolkit (JDT).
- Środowisko tworzenia wtyczek (ang. *plug-in*) o nazwie PDE (ang. *Plug-in Development Environment*), które służy do budowania własnych modułów rozszerzających możliwości Eclipse (zwanych wtyczkami).

Wymienione projekty również są podzielone na kolejne podprojekty — JDT, na przykład, składa się części odpowiadającej za interfejs użytkownika, podzbiór odpowiedzialny ze debugowanie itp. Całą ich listę możemy zobaczyć, zerkając na stronę znajdującą się pod adresem *http://www.eclipse.org/eclipse/* lub (co zdecydowanie zalecamy) oddając się lekturze niniejszej książki.

Parę słów na temat licencji typu CPL (ang. Common Public License)

Eclipse to oprogramowanie typu open source, ale pojęcie to pozostawia pewne wątpliwości dotyczące legalności używania aplikacji tego typu. Wyjaśnijmy więc, że ten rodzaj software'u pozwala użytkownikowi na dostęp do kodu źródłowego i prawa do jego modyfikacji. Wolno nam również oprogramowanie to dystrybuować. Z drugiej jednak strony licencje typu open source często nie zezwalają na dystrybucję programu w postaci zmodyfikowanej, chyba że końcowy odbiorca otrzyma odpowiednie prawa.

Pewne licencje wymagają także, aby aplikacje korzystające z rozwiązań typu open source również były rozpowszechniane na tych samych prawach. Widzimy więc, że open source jest pojęciem dosyć rozległym i opisuje pewną klasę programów. Dlatego wprowadzono bardziej szczegółowy podział licencji. Eclipse udostępniane jest na zasadach CPL (ang. *Common Public License*), co oznacza, że oprogramowanie korzystające z naszego środowiska bądź je rozszerzające może być rozpowszechniane na licencjach bardziej restrykcyjnych. Mówiąc prościej, możemy używać go w celach komercyjnych.

Natomiast jeśli zamierzasz modyfikować i udostępniać nowe wersje samego Eclipse, koniecznie zapoznaj się ze szczegółami dotyczącymi licencji typu CPL (pod adresem *http://www.opensource.org*, a dokładnie *http://www.opensource.org/licenses/cpl.php*). Pewien fragment opisu tej licencji mówi, że „celem jej jest ułatwienie wykorzystania programu do celów komercyjnych".

Tyle o licencjach, teraz zajmiemy się samym programem.

Jak wejść w posiadanie Eclipse?

W jaki sposób możemy otrzymać i zainstalować Eclipse? Bardzo prosto — wchodzimy na stronę *http://www.eclipse.org/downloads* i wybieramy jeden z dostępnych tam mirrorów. Następnie wskazujemy i pobieramy (bezpłatnie) interesującą nas wersję. Ponieważ jest ich sporo, kilka słów komentarza.

Istnieją cztery wersje wydań (ang. *build*) środowiska Eclipse:

Wersje gotowe do rozpowszechniania (ang. Release builds)
: Zespół tworzący Eclipse zaleca korzystanie z wersji tego typu. Są to kolejne oficjalne wydania, odpowiednio przetestowane i zapewniające poprawną, bezproblemową pracę. Gdyby Eclipse był sprzedawany w pudełkach, znaleźlibyśmy w nich właśnie kolejne *release builds*.

Wydania stabilne (ang. Stable builds)
: Możemy porównać je do wersji *beta*. Wersja stabilna jest kandydatem na wersję gotową *release*, nie jest jednak jeszcze w pełni przetestowana i może sprawiać różne (najczęściej drobne) kłopoty. Korzystając z niej mamy wszakże okazję zapoznać się z funkcjonalnością nowych wydań przed ich oficjalną prezentacją.

Wersje integracyjne (ang. Integration builds)

Tutaj mamy do czynienia z wersjami, które są w fazie scalania składników projektu. Mimo że każdy z nich osobno został w pełni przetestowany, prawidłowa współpraca nie jest jeszcze przesądzona i może powodować pewne „zgrzyty". Dopiero po upewnieniu się, że wszystko jest OK, program przechodzi do grupy wydań stabilnych.

Wersje robocze (ang. Nightly builds)

Najbardziej eksperymentalne spośród dostępnych publicznie wydań Eclipse. Są to wersje na bieżąco rozwijane przez zespół, nie ma żadnej gwarancji, że będą pracować poprawnie. Doświadczenia z nimi mogą obfitować w poważne problemy, ale jest to najlepszy sposób, by zorientować się, w jakim kierunku zmierza projekt i co nowego w nim się pojawi.

W normalnych warunkach zalecamy pobranie najświeższej wersji gotowej (*release*). Nie zapominajmy też, że koniecznie należy wskazać, jaki system operacyjny nas interesuje. A później już tylko klikamy i pobieramy.

Jeśli interesują nas szczegóły dotyczące obecnej i planowanych wersji, możemy zerknąć tutaj: *http://www.eclipse.org/eclipse/development/main.html*.

Instalacja Eclipse jest wyjątkowo nieskomplikowana — cały proces obejmuje rozpakowanie i skopiowanie plików (pobrany plik, w zależności od systemu operacyjnego, ma format *.zip* lub *.tar*). Po zakończeniu tych operacji znajdziemy plik wykonywalny (w wersji Windows jest to *eclipse.exe*), który jest gotowy do uruchomienia.

Miła wiadomość dla użytkowników systemu Windows — Eclipse nie korzysta z rejestru systemowego, tak więc nie ma problemów z deinstalacją bądź ponowną instalacją.

Przygodę z Eclipse rozpoczynamy od uruchomienia pliku wykonywalnego (np. *eclipse.exe*). Za pierwszym razem musimy uzbroić się w odrobinę cierpliwości — Eclipse wymaga paru sekund na przeprowadzenie pewnych czynności instalacyjnych, takich jak utworzenie katalogów roboczych itp. Nie trwa to jednak długo, dlatego też chwilę później witani jesteśmy ekranem zaprezentowanym na rysunku 1.1.

Do prawidłowej pracy Eclipse wymaga zainstalowanej wcześniej maszyny wirtualnej Java. Jeśli warunek ten nie zostanie spełniony, zobaczymy komunikat „A Java Runtime Environment (JRE) or Java Development Kit (JDK) must be available in order to run Eclipse". Jeśli nie zainstalowaliśmy Javy wcześniej, musimy to zrobić teraz, inaczej Eclipse nie zadziała. Javę pobieramy spod adresu *http://java.sun.com/j2se*.

W następnym rozdziale zaprezentujemy, jak Eclipse obsługuje sytuację, gdy mamy więcej niż jedną wersję Javy, i w jaki sposób, na przykład, przepiąć się z zainstalowanego wraz z przeglądarką JRE na właśnie pobraną najnowszą wersję JDK.

By maksymalnie uprościć uruchamianie Eclipse, proponujemy stworzenie skrótu do pliku wykonywalnego i umieszczenie go gdzieś pod ręką. W Windows wystarczy w tym celu kliknąć prawym przyciskiem myszy plik *eclipse.exe*, a następnie użyć polecenia Utwórz skrót (ang. *Create Shortcut*). Skrót możemy przenieść np. na pulpit. W systemie Linux bądź Unix dołączamy katalog Eclipse do domyślnej ścieżki bądź korzystamy z polecenia `ln -s` w celu stworzenia skrótu.

Architektura Eclipse

Czym właściwie jest Eclipse? Większość z nas uważa, że Eclipse to zintegrowane środowisko do tworzenia aplikacji w języku Java, czyli kolejne Java IDE. W sądzie takim utwierdza nas fakt, że po pobraniu i uruchomieniu aplikacji rzeczywiście otrzymujemy zestaw narzędzi do pisania w Javie (który nosi nazwę Java Development Toolkit, czyli JDT) oraz środowisko do tworzenia wtyczek (ang. *Plug-in Development Environment, w skrócie PDE*). Tak więc, jeśli zamierzasz używać Javy, możesz traktować Eclipse właśnie w ten sposób.

Jednak Eclipse to znacznie więcej niż się na początku wydaje. Jest to *uniwersalna platforma* do tworzenia aplikacji. A JDT — odpowiednik Java IDE — jest tak naprawdę jedynie wtyczką, czyli jednym z dostępnych modułów (tyle, że najpopularniejszym). Musimy więc jasno wyjaśnić, że Eclipse samo w sobie jest rodzajem platformy, szkieletu, który umożliwia podłączenie do niego wtyczek o bardzo różnym charakterze i zastosowaniu. W ramach tego szkieletu funkcjonują moduły zwane wtyczkami realizujące konkretne zadania. Eclipse samo w sobie jest pakietem stosunkowo niewielkim.

Platforma udostępnia możliwość uruchamiania wtyczek. Tak więc, jeśli chcemy programować w Javie, korzystamy z JDT, modułu, który domyślnie jest zawarty w środowisku. Gdy interesują nas inne języki, używamy innych wtyczek, takich jak np. CDT, czyli modułu służącego do tworzenia aplikacji w C/C++. Instalowanie kolejnych wtyczek jest bardzo łatwe, o czym wkrótce się przekonamy — polega na skopiowaniu odpowiednich plików do wskazanego katalogu i ponownym uruchomieniu Eclipse. W trakcie restartu Eclipse sam wykryje zmiany i zarejestruje nowy moduł, choć ze względu na racjonalne gospodarowanie pamięcią załaduje go dopiero w momencie, gdy będzie potrzebny.

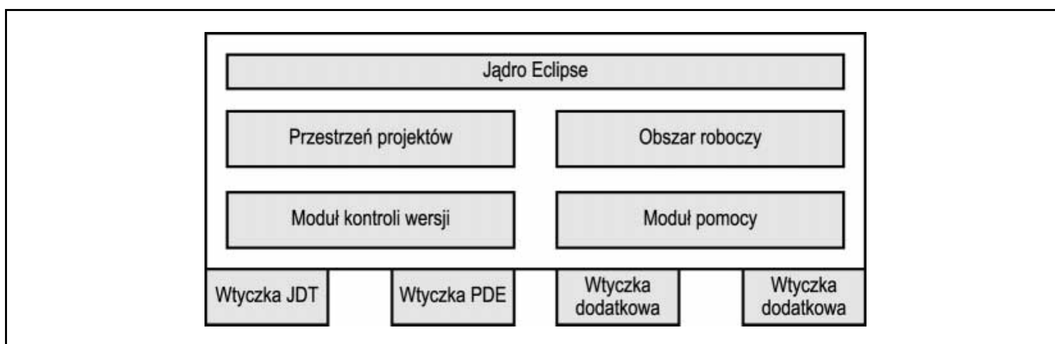
Warto zdać sobie sprawę, że Eclipse, chociaż zostało napisane w Javie, jest środowiskiem niezależnym od konkretnego języka programowania. Do tworzenia aplikacji w dowolnym języku wymagana jest jedynie odpowiednia wtyczka. Co więcej, samo środowisko nie musi bazować na języku angielskim. Taką zmianę również realizuje się poprzez wymianę wtyczek. Firma OTI oferuje moduły obsługujące język japoński, koreański, niemiecki, francuski, włoski, portugalski, hiszpański, a nawet chiński (w wersji tradycyjnej i uproszczonej).

Platforma Eclipse

Platforma Eclipse składa się z kilku podstawowych komponentów, którymi są: jądro, obszar roboczy, przestrzeń projektów, moduł kontroli wersji i moduł pomocy. Widzimy je wszystkie na rysunku 1.3.

Jądro (ang. Platform Kernel)

Głównym zadaniem jądra jest uruchomienie całości i załadowanie poszczególnych wtyczek. Jądro jest modułem, który startuje jako pierwszy i ładuje te komponenty, które widzimy na ekranie i utożsamiamy z Eclipse.



Rysunek 1.3. Architektura Eclipse

Obszar roboczy (ang. Workbench)

Spójrzmy jeszcze raz na rysunek 1.1 — widzimy na nim okno główne Eclipse, czyli nasz obszar roboczy. Zawiera ono wszystkie potrzebne menu i listwy narzędziowe, a jego zadaniem jest wyświetlanie i zarządzanie wieloma wewnętrznymi oknami, które nazywać będziemy widokami.

Obszar roboczy, obok jądra, należy do najbardziej podstawowych modułów. Widzimy go podczas uruchamiania Eclipse, on też wyświetla ekran powitalny. Dopiero później przechodzimy do innego IDE, takiego jak JDT, które przejmuje sterowanie.

Obszar roboczy wywołuje pewne kontrowersje. W każdym systemie operacyjnym wygląda jak rodzima aplikacja, a nie jak typowy program napisany w Javie. Interfejs użytkownika Eclipse został stworzony bowiem przy użyciu biblioteki SWT (ang. *Standard Widget Toolkit*) oraz zbudowanej na jej podstawie biblioteki JFace. SWT współpracuje bardzo blisko z systemem operacyjnym i wykorzystuje do maksimum jego możliwości, dlatego też aplikacje mają wygląd odpowiadający platformie, w ramach której działają. Jest to koncepcja zupełnie inna od tej, która przyświecała twórcom biblioteki Swing, czyli podstawowego zbioru klas służących do budowy interfejsu użytkownika w Javie.

Biblioteka SWT musiała zostać napisana oddzielnie dla każdego systemu operacyjnego, w którym może działać Eclipse. To właśnie było powodem sporu wewnątrz społeczności zaangażowanej w tworzenie narzędzia; część osób twierdziła, że traci się w ten sposób tak ważną dla Javy niezależność od systemu operacyjnego. Zdecydowano jednak inaczej, ale nie jest to żaden problem, ponieważ Eclipse dostępny jest na wszystkie znaczące platformy, takie jak Windows, Solaris, Mac OS X, Linux/Motif, Linux/GTK2, HP-UX i wiele innych.

W dalszej części książki zaprezentujemy biblioteki SWT i JFace, które pozwalają nam tworzyć programy o wyglądzie naturalnym dla używanego przez nas systemu operacyjnego. SWT zapewnia podstawowe operacje dotyczące grafiki i klasy bazowe obsługujące elementy interfejsu użytkownika, JFace funkcjonalność tę znacząco rozszerza. Bardzo istotne jest to, że z obu możemy korzystać również przy tworzeniu własnych aplikacji.

Przestrzeń projektów (ang. Workspace)

Przestrzeń projektów (ang. *workspace*) zarządza naszymi zasobami — tzn. wszystkimi plikami wchodzącymi w skład projektów, które przechowujemy na dysku bądź współdzielimy z innymi komputerami. Aplikacje tworzone w Eclipse nazywamy *projektami*. Każdy projekt (pliki

wchodzące w jego skład) stanowi osobny folder, który znajduje się w katalogu stanowiącym przestrzeń projektów (np. *c:\eclipse211\workspace*). Dzięki temu łatwo do niego dotrzeć. Poszczególne katalogi projektów mogą zawierać wiele podkatalogów. Zwykle wszystkie katalogi projektów są podkatalogami w ramach przestrzeni projektów, ale nie musi tak być — do projektów możemy dołączać katalogi znajdujące się w dowolnym miejscu na dysku naszego komputera lub w sieci.

Gdy pracujemy z kodem, komponent przestrzeni projektów odpowiedzialny jest za zarządzanie wszelkimi zasobami, które stanowią projekt. Praca ta polega na zapisywaniu zmian, prowadzeniu ich historii i zapewnieniu możliwości powrotu do wersji poprzednich.

Przechowywanie projektów w tym samym katalogu ma wiele zalet. Na przykład, jeśli zainstalujemy nową wersję Eclipse, przeniesienie wszystkich projektów z wersji poprzedniej nie stanowi żadnego problemu i polega na skopiowani zawartości katalogu będącego przestrzenią projektów. (Na wszelki wypadek jednak w komentarzu do nowej wersji koniecznie należy sprawdzić, czy taka operacja jest dozwolona. Na przykład przypomnimy, że nie da się przenieść projektów stworzonych w wersji 1.0 do wersji 2.0 i późniejszych. Nie powinno być jednak żadnych problemów przy przechodzeniu np. z 2.1.1 do 2.1.2).

Moduł kontroli wersji (ang. Team Component)

Moduł kontroli wersji (ang. *team component*) to wtyczka, która odpowiada w Eclipse za zarządzanie kolejnymi wersjami plików z kodem źródłowym. Mechanizm ten polega na tym, że dany plik źródłowy jest na czas edycji pobierany (ang. *check out*) z repozytorium, a po wprowadzeniu zmian zostaje tam zwrócony (ang. *check in*). Tak więc kolejne zmiany mogą być śledzone. Co więcej, ponieważ każdy plik może być w danym momencie pobrany i edytowany tylko przez jednego członka zespołu, nie ma możliwości, by zmiany wprowadzone przez jedną osobę ulegały zatarciu bądź były pominięte na etapie scalania.

Komponent śledzenia wersji pełni rolę klienta systemu CVS (ang. *Concurrent Version System*), który odwołuje się do serwera CVS. Jeśli narzędzie o nazwie CVS nie jest Czytelnikowi dobrze znane, nie musi się obawiać. Zajmiemy się tym zagadnieniem dokładnie w rozdziale czwartym. Dzięki systemowi kontroli wersji jesteśmy w stanie śledzić wszystkie zmiany dokonywane w projekcie, co jest cechą niezwykle istotną w pracy zespołowej. Ale również kiedy aplikacje są pisane w pojedynkę, możliwość zobaczenia historii zmian w kodzie może okazać się cechą bardzo pomocną.

Moduł pomocy (ang. Help Component)

Moduł pomocy to komponent, którego zadaniem jest zarządzanie i udostępnianie użytkownikowi plików pomocy. Jest to w rzeczywistości rozszerzalny system zarządzający dokumentacją. Dołączane wtyczki dostarczają dokumenty pomocy w formacie HTML wraz z ich definicją zapisaną w formacie XML, dzięki czemu nowe pliki mogą zostać dołączone do istniejącego systemu pozwalającemu na nawigację.

Powyżej przedstawiliśmy podstawowe składniki architektury Eclipse. Ale do skutecznego posługiwania się aplikacją niezbędna jest jeszcze prezentacja niezwykle ważnych pojęć, jakimi są: widok i perspektywa.

Widoki i perspektywy

Kiedy pracujemy z Eclipse, korzystamy z obszaru roboczego, który podzielony jest na wiele zawartych w nim paneli nazywanych widokami (ang. *view*). Te widoki pozwalają nam kontrolować różne składniki projektu i jego strukturę. Na przykład jeden widok pokazuje klasy zawarte w projekcie, w innym widzimy pakiety, a kolejny to kod źródłowy wybranej klasy. Zerkając na rysunek 1.1, w lewym górnym rogu dostrzec można widok o nazwie *Navigator* — dzięki niemu możemy zobaczyć dostępne w danym momencie projekty i wybrać jeden z nich.

Ponieważ miejsca na ekranie nigdy nie jest tyle, ile byśmy chcieli, widoki często są na siebie nałożone. Do przechodzenia między nimi korzystamy z zakładek, które znajdują się zawsze przy którejś z krawędzi.

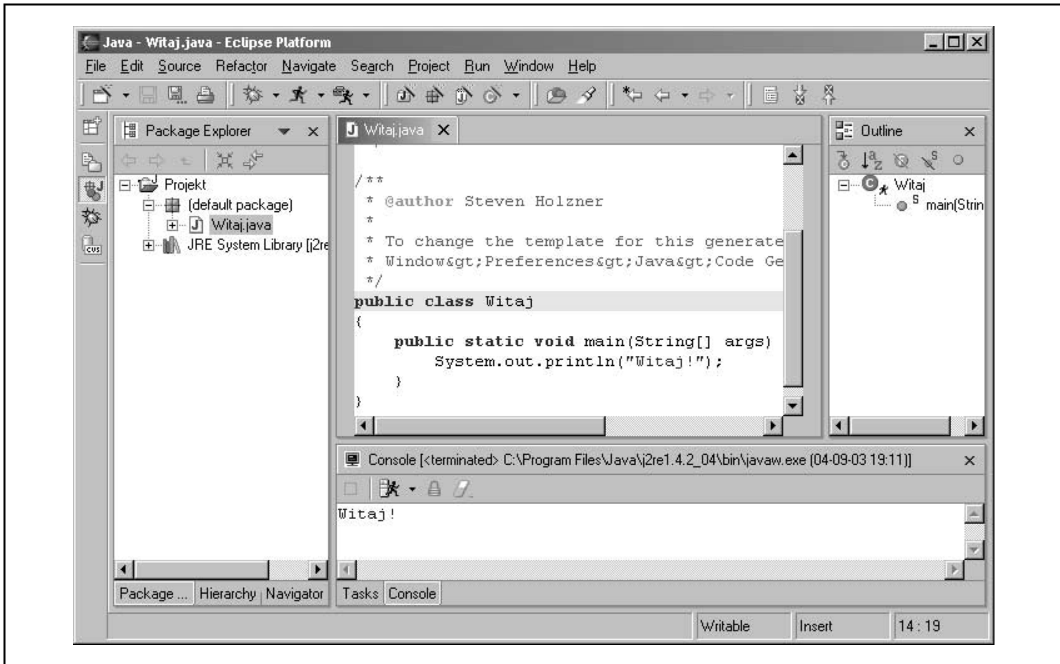
Zdarza się czasami, że chcemy powrócić do widoku, który wcześniej zamknęliśmy. W tym celu należy użyć polecenia *Window/Show View* i wybrać interesujący panel.

Edytor jest specjalnym typem widoku, który pojawia się w centralnej części obszaru roboczego. Zawartość otwieranych dokumentów, plików z kodem źródłowym czy też innych zasobów pojawia się właśnie w edytorze. Eclipse automatycznie dopasowuje rodzaj edytora do typu edytowanego dokumentu: dla plików z kodem w języku Java jest to edytor Javy, dla zasobów definiujących wygląd interfejsu aplikacji jest to odpowiedni edytor GUI, zdefiniowany i zawarty w odpowiedniej wtyczce. Możemy także otworzyć plik zapisany w formacie Microsoft Word (Eclipse pozwala na edycję takiego pliku dzięki obsłudze mechanizmu OLE). Zaraz po uruchomieniu Eclipse wyświetla w edytorze tekst powitalny (rysunek 1.1).

Okno edytora to miejsce, w którym wykonujemy największą pracę przy tworzeniu aplikacji — to właśnie tutaj wprowadzamy bądź modyfikujemy kod źródłowy. Edytor zawarty w module JDT (czyli naszym Java IDE) ma bardzo dużo cech podnoszących komfort pracy i wydajność. Mamy tu na myśli sprawdzanie i kolorowanie składni, rozwijanie listy dostępnych metod, parametrów itp. Jednocześnie możemy pracować z kilkoma otwartymi dokumentami; do przełączania pomiędzy nimi służą zakładki znajdujące się przy górnej krawędzi widoku (można w tym celu użyć polecenia *Window/Switch to Editor…* — zobaczymy wtedy listę otwartych plików). By zakończyć edycję dowolnego dokumentu, wystarczy kliknąć na symbol *X* znajdujący się na każdej z zakładek. Widok całego edytora (niezależnie od liczby otwartych tam dokumentów) możemy schować za pomocą opcji *Window/ Hide Editors*. Powrót do poprzedniego stanu uzyskamy natomiast, korzystając z polecenia *Window/Show Editors*. Podsumowując: widoki to sposoby prezentacji wybranych aspektów projektu. Edytor to miejsce, w którym tworzymy i rozwijamy kod bądź inne zasoby.

Drugim niezwykle istotnym pojęciem jest perspektywa (ang. *perspective*). Perspektywy są to grupy odpowiednio skonfigurowanych widoków. Wiadomo, że inny układ widoków będzie nam odpowiadał w momencie pisania kodu w Javie, a zupełnie inny chcemy mieć podczas debugowania aplikacji. Dzięki perspektywom nie musimy za każdym razem zmieniać układu okien — wystarczy przełączyć się na wybraną perspektywę.

Każda perspektywa jest zestawem odpowiednio rozmieszczonych widoków i edytorów; po jej wybraniu ekran automatycznie przyjmuje zdefiniowaną postać. Spójrzmy, na przykład, na perspektywę o nazwie *Java*: by ją wybrać, użyjemy polecenia *Window/Open Perspective*, a z podmenu, które się pokaże, pozycję o nazwie *Java*. Ekran przebierze wtedy postać pokazaną na rysunku 1.4.

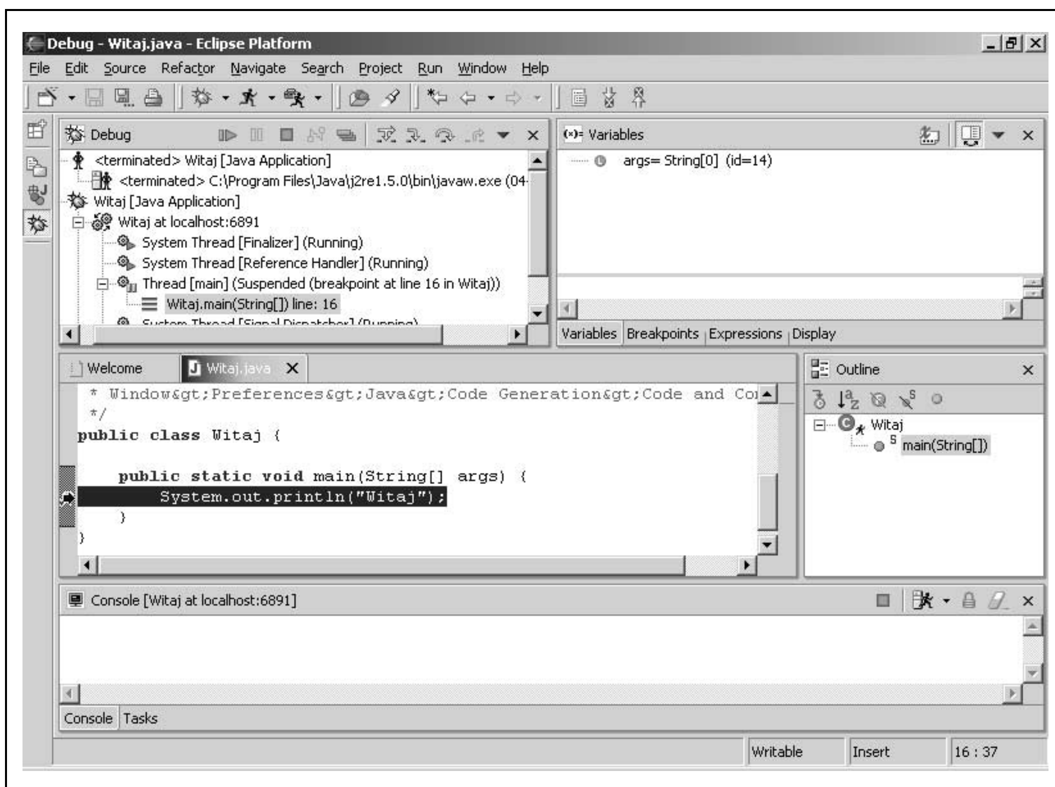


Rysunek 1.4. Perspektywa Java

Na perspektywę składa się przede wszystkim edytor widoczny w środku obszaru roboczego. To w nim będziemy wpisywać kod źródłowy. Obecne są również inne widoki, wśród nich przeglądarka pakietów (ang. *Package Explorer*) widoczna po lewej stronie. Służy ona do nawigowania wśród pakietów i klas dostępnych w projekcie. Po prawej stronie znajduje się widok przedstawiający szczegóły aktualnie wskazanej klasy. Kiedy zechcemy rozwijaną aplikację debugować, obszar roboczy przełączamy na perspektywę *Debug*, którą widzimy na rysunku 1.5. W tym wariancie edytor z kodem nie musi mieć już tak dużych rozmiarów, ważna jest natomiast możliwość podglądania wartości zmiennych i wyrażeń czy też stosu wywołań.

Koncepcja widoków, edytorów i perspektyw jest dla użytkownika bardzo przejrzysta i intuicyjna. Tak bardzo, że w codziennej pracy często się o niej zapomina. Jednak dla nas, którzy bacznie przyglądamy się Eclipse, są to pojęcia bardzo ważne, i dlatego będziemy je w dalszej części książki nazywać po imieniu.

A teraz, wzmocnieni bliską znajomością architektury i pojęć podstawowych, rozpoczynamy właściwą pracę. Mimo iż Eclipse to platforma uniwersalna, duża jej popularność wiąże się przede wszystkim z jej zastosowaniem do tworzenia aplikacji w Javie. Tak też będzie w naszym przypadku. Zaraz po uruchomieniu widzimy Eclipse z perspektywy o nazwie *Resource* — jest to domyślna, ogólna perspektywa dotycząca tworzenia dowolnych zasobów. My jednak przełączymy się do trybu o nazwie *Java* i z tego miejsca rozpoczniemy pisanie przykładowego kodu.



Rysunek 1.5. Perspektywa Debug

Pierwszy program

Będziemy korzystać z JDT (ang. *Java Development Tools*). Pod nazwą tą kryje się zestaw sześciu ściśle współpracujących ze sobą wtyczek, które wspólnie tworzą środowisko do tworzenia aplikacji w języku Java. I nawet jeśli programujesz w tym języku od lat, przygotuj się na nowe, wspaniałe doświadczenia. Poznając zasady posługiwania się JDT, często zadajemy sobie pytanie, dlaczego na tak dobre rozwiązanie trzeba było czekać tak długo.

Eclipse to narzędzie, w którym największy nacisk położono na komfortowe tworzenie kodu. Dlatego najlepszym sposobem na odkrycie jego zalet jest próba samodzielnego napisania aplikacji. W naszym przypadku użyjemy JDT do przygotowania wielce użytecznego programu, którego zadaniem będzie wypisanie krzepiącego komunikatu o treści „Bez obaw!”. Pełen kod źródłowy widzimy na listingu 1.1.

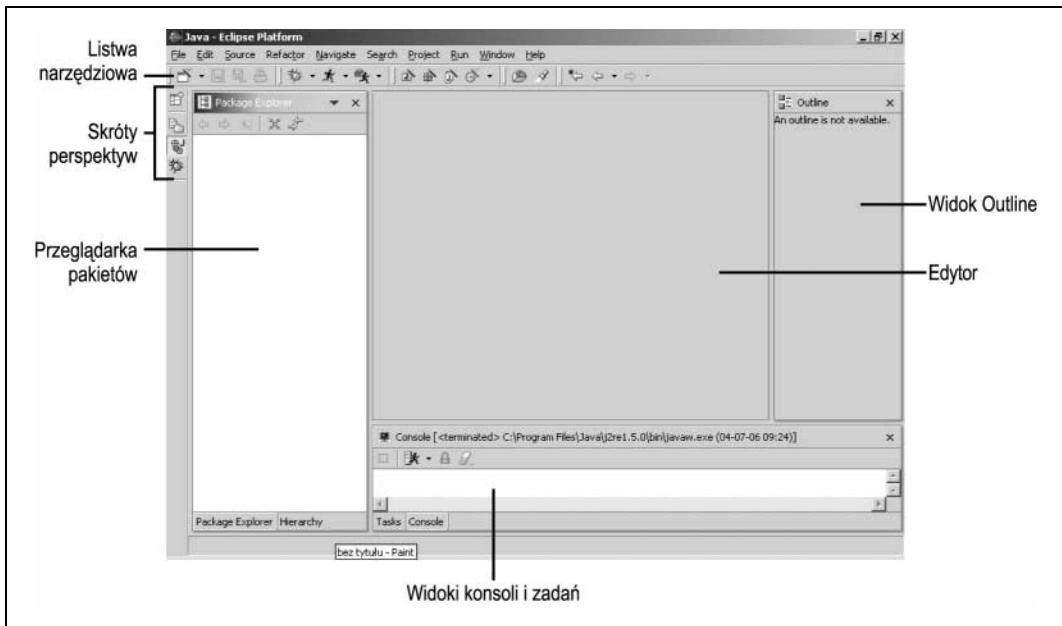
Listing 1.1. Przykład Ch01_01.java

```
public class Ch01_01
{
    public static void main(String[] args) {
        System.out.println("Bez obaw!");
    }
}
```

Jak taki program napisać za pomocą Eclipse? I w jaki sposób Eclipse usprawni proces tworzenia aplikacji? Właśnie teraz udzielimy odpowiedzi na tak postawione pytania. Zaczniemy od tego, że wszystkie pliki źródłowe w języku Java muszą być zawarte w projekcie. Tak więc pierwszym naszym krokiem będzie stworzenie nowego projektu.

Tworzenie projektu

By przywołać perspektywę o nazwie *Java*, a następnie wprowadzić kod naszego pierwszego przykładu, *Ch01_01.java*, zaraz po uruchomieniu Eclipse musimy wywołać opcję *Window/Open Perspective/Java*. Spowoduje to uruchomienie modułu JDT i przejście do ekranu skonfigurowanego w sposób widoczny na rysunku 1.6. To perspektywa, z której w swojej pracy z Eclipse korzystamy najczęściej.



Rysunek 1.6. Eclipse wraz z JDT w perspektywie Java

Zanim zaczniemy cokolwiek pisać, warto zerknąć na układ widoków. Na samej górze okna głównego (czyli obszaru roboczego) znajduje się menu główne i listwy narzędziowe (z pozycjami standardowymi Eclipse oraz tymi charakterystycznymi dla JDT). Poznamy je bardziej szczegółowo już wkrótce.

Po lewej stronie ekranu widzimy panel, na którym znajdują się dwa widoki: przeglądarka pakietów (ang. *Package Explorer*) oraz widok hierarchii (ang. *Hierarchy*). Przełączamy się pomiędzy nimi przy wykorzystaniu zakładek widocznych w dolnej części panelu. Pierwszy panel pozwala nam poruszać się wśród pakietów składających się na nasz projekt, a także przechodzić do innych projektów i ich składników. Możemy również wskazywać za jego pomocą pliki, które chcemy otworzyć w edytorze. Widok hierarchii pozwala nam sprawdzić, jak dany element (klasa) wygląda na tle hierarchii klas — wskazujemy interesującą nas pozycję w edytorze kodu, klikamy prawym przyciskiem myszy i z menu kontekstowego wybieramy polecenie

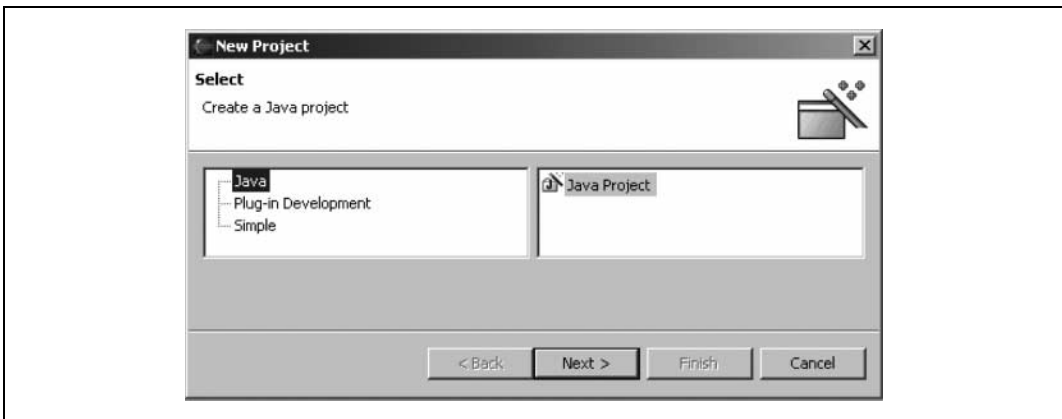
Open Type Hierarchy. Po chwili w opisywanym widoku mamy wszystkie informacje dotyczące dziedziczenia, klas bazowych itp. Widok nie ma charakteru statycznego — możemy swobodnie poruszać się po drzewie klas i ich polach oraz metodach. To świetny (i chyba najszybszy) sposób na zorientowanie się, jakie metody (wraz z parametrami) zawiera wskazana klasa.

Widoczny po prawej stronie ekranu panel z widokiem *Outline* prezentuje uporządkowany, hierarchiczny opis zawartości edytowanego właśnie pliku. Co więcej, mamy możliwość nawigacji — wystarczy wskazać na dowolny element widoku, a zostanie on również pokazany w edytorze. Ta cecha powinna zadowolić przede wszystkim osoby, które dotychczas pisały kod Javy za pomocą prostych edytorów tekstu. Już nie ma konieczności przewijania długich plików w poszukiwaniu interesującej nas metody, teraz wystarczy kliknąć pozycję drzewa.

Dolna część okna perspektywy *Java* zawiera panel z dwoma widokami: zadania (ang. *Tasks*) oraz konsola (ang. *Console*). Jak zwykle przełączamy się między nimi przy użyciu zakładek. Widok zadań zawiera listę problemów, które wymagają rozstrzygnięcia; są to przede wszystkim zgłoszone błędy. Widok konsoli stanowi odpowiednik standardowego urządzenia wyjścia podczas uruchamiania programów z poziomu Eclipse. To tu pojawiać się będą komunikaty takie jak „Bez obaw!".

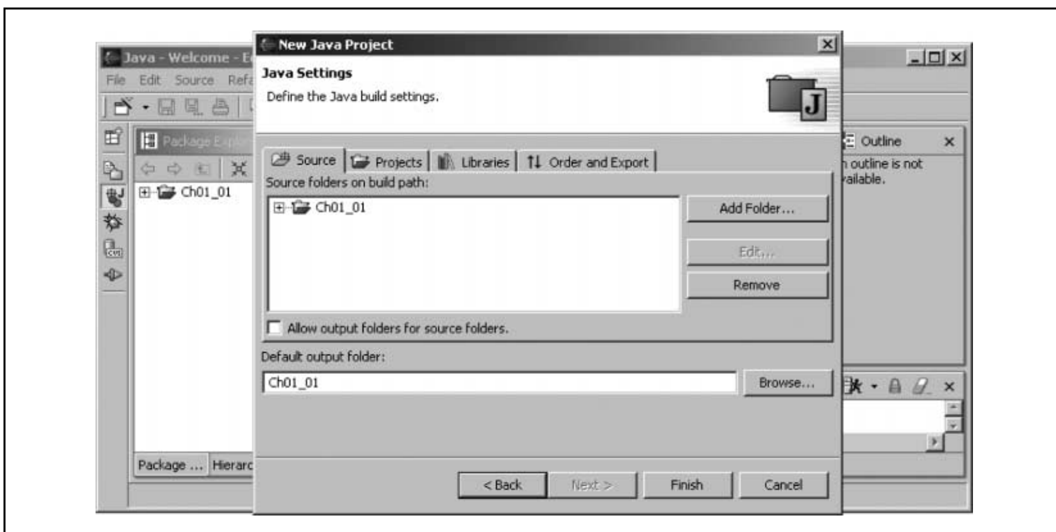
Panel zawierający widoki edytowanych plików znajduje się w środku ekranu. Zakładki do przełączania między nimi znajdują się w górnej jego części, zaczynając od lewej strony. Edytory zawarte w JDT oferują przebogatą listę funkcji, znacznie wykraczając poza możliwości typowego edytora służącego do wprowadzanie tekstu. Większość ich atutów początkowo nie rzuca się w oczy, zaskoczeni jesteśmy dopiero w trakcie pisania.

A w praktyce działa to tak: w celu utworzenia nowego projektu po wybraniu perspektywy *Java* wywołujemy polecenie *File/New/Project*. Można w tym celu posłużyć się również prawym przyciskiem myszy użytym w obszarze przeglądarki pakietów — wybieramy wtedy opcję *New/Project*. W obu przypadkach na ekranie pojawi się okno dialogowe o nazwie *New Project* pokazane na rysunku 1.7



Rysunek 1.7. Kreator tworzący nowy projekt

Wskazujemy pozycję *Java* (po lewej stronie) oraz *Java Project* (po prawej) i klikamy przycisk *Next*, dzięki czemu przechodzimy do drugiego kroku pracy kreatora. Tutaj w polu o nazwie *Project Name* wpisujemy nazwę tworzonego projektu: `Ch01_01`, a następnie klikamy ponownie przycisk *Next*. Pojawia się kolejne okienko kreatora (rysunek 1.8), w którym widzimy, że za



Rysunek 1.8. Trzeci etap działania kreatora projektów

chwilę utworzony zostanie projekt o nazwie `Ch01_01` w katalogu o tej samej nazwie. Przechodząc na zakładkę *Projects*, możemy do ścieżki naszego nowego projektu dołączyć inne, istniejące już projekty. Będziemy to robić przy tworzeniu nieco bardziej zaawansowanych aplikacji. Zakładka *Libraries* pozwala przeglądać oraz dodawać zewnętrzne biblioteki w postaci plików JAR. Standardowo dołączone są biblioteki systemowe JRE. Zakładka *Order and Export* służy do ustalenia kolejności klas na ścieżce przeszukiwania. Tutaj też możemy zdecydować, czy projekt ma być dostępny dla innych aplikacji. W naszym przykładzie nie modyfikujemy żadnego z opisanych parametrów, klikamy jedynie przycisk *Finish*, kończąc w ten sposób działanie kreatora.

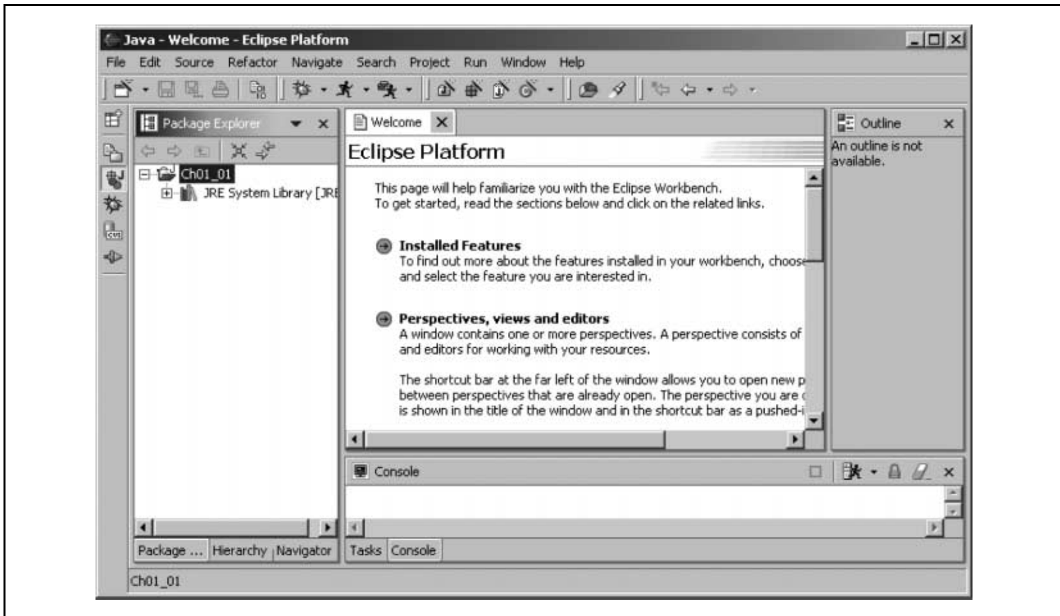
Po wykonaniu powyższych kroków w przeglądarce pakietów pojawia się nowa pozycja, czyli stworzony przed chwilą projekt `Ch01_01`, tak jak na rysunku 1.9. Reprezentowany jest on jako katalog o tej samej nazwie w przestrzeni projektów i w tej chwili nie zawiera nic poza dołączonymi bibliotekami systemowymi JRE.

Projekty składają się z plików, klas i dołączonych bibliotek. Ponieważ my nic takiego jeszcze nie mamy, kolejnym krokiem będzie stworzenie pierwszej klasy w języku Java.

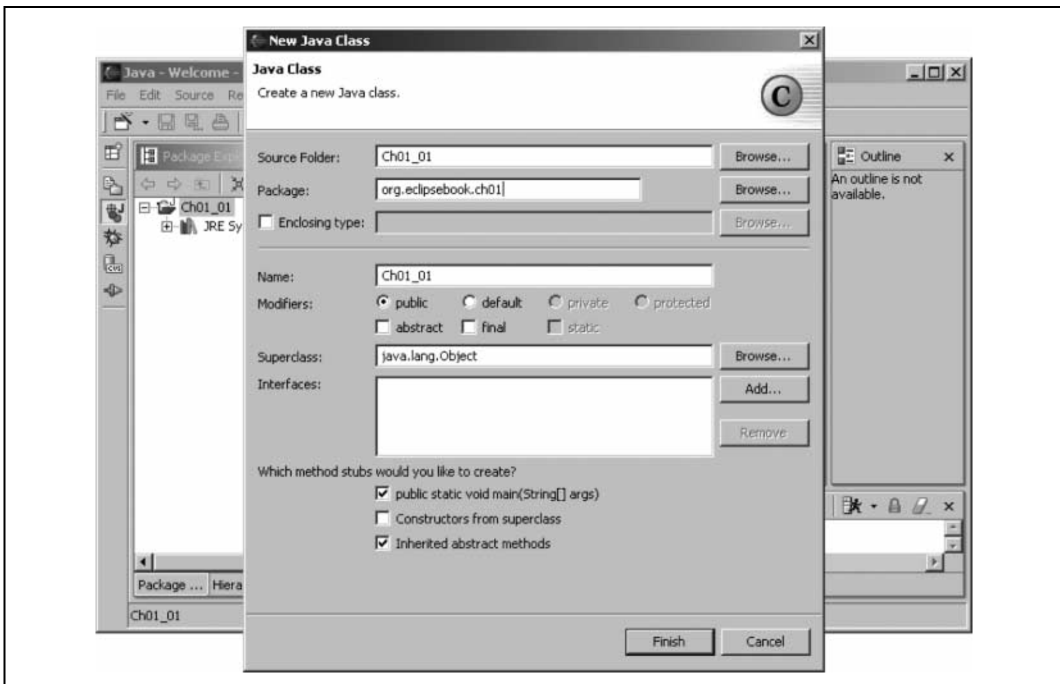
Tworzenie klasy

W naszym przykładzie mamy jedną publiczną klasę o nazwie `Ch01_01`, której kod źródłowy zostanie przez Eclipse zapisany w pliku o tej samej nazwie i rozszerzeniu *.java*. Sposobów dodania nowej klasy do projektu jest wiele: można użyć ikony z okrągłym *C* wewnątrz z listwy narzędziowej, można skorzystać z polecenia menu *File/New/Class* lub też kliknąć prawym przyciskiem myszy w obrębie przeglądarki pakietów i wybrać opcję *New/Class*. Niezależnie od wariantu, na który się zdecydujemy, efekt zawsze będzie taki sam: pojawi się okno dialogowe o nazwie *New Java Class* (rysunek 1.10).

Spójrzmy na opcje zawarte w dialogu. Dla nowo tworzonej klasy określić możemy modyfikator dostępu — `public`, `private` bądź `protected`; wolno nam także wskazać, że klasa jest typu abstract bądź final. Podać możemy też klasę, po której chcemy dziedziczyć, oraz interfejsy,



Rysunek 1.9. Nowy projekt w przeglądarce pakietów

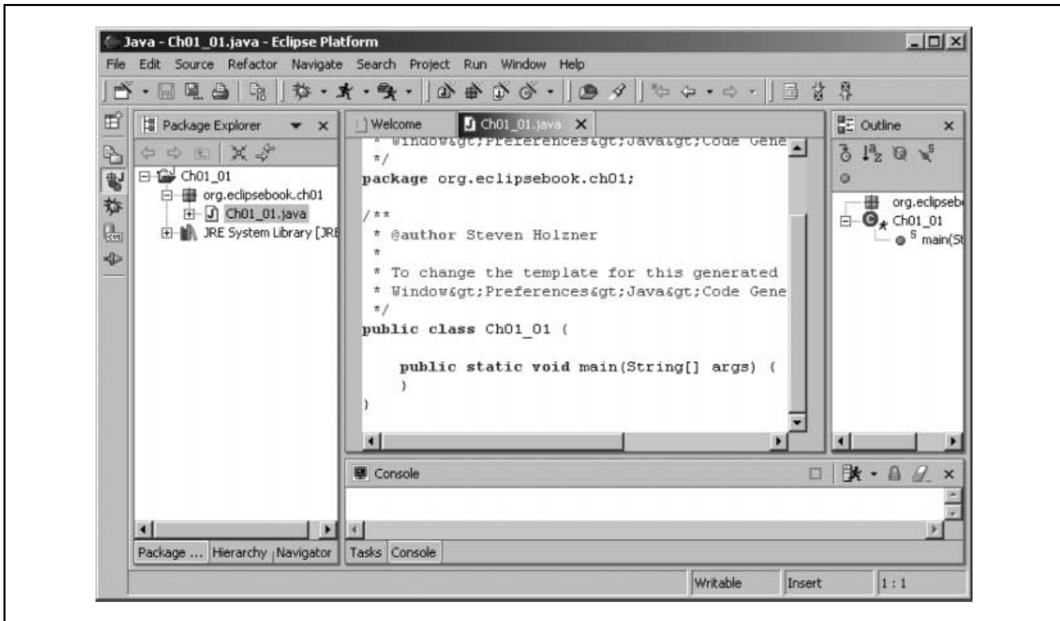


Rysunek 1.10. Kreator klas

które będziemy implementować. Istnieje również swoboda wskazania pakietu, do którego klasa ma należeć, po to, być uniknąć ewentualnych konfliktów nazw. Umawiamy się, że nazwy pakietów odpowiadać będą kolejnym rozdziałom książki, tak więc w polu *Package* wpisujemy

org.eclipsebook.ch01. Nie wolno oczywiście zapomnieć o wpisaniu nazwy samej klasy; do pola *Name* wstawiamy Ch01_01. Akceptujemy pozostałe parametry i wciskamy przycisk *Finish*. Zwrócić również należy uwagę, że włączona została opcja *public static void main(String[] args)* w polu określającym funkcje, których szkielety należy automatycznie wygenerować. Dzięki temu Eclipse sam stworzy metodę main.

Ułamek sekundy po kliknięciu przycisku *Finish* jesteśmy już w edytorze i widzimy kod źródłowy właśnie wygenerowanej klasy Ch01_01 (rysunek 1.11). Zwróć uwagę, że znajduje się ona w pakiecie org.eclipsebook.ch01. Plik źródłowy o nazwie *Ch01_01.java* pojawił się w katalogu *workspace\ Ch01_01\org\eclipsebook\ch01*, czyli w folderze projektu rozszerzonym o podkatalogi odpowiadające pakietom.



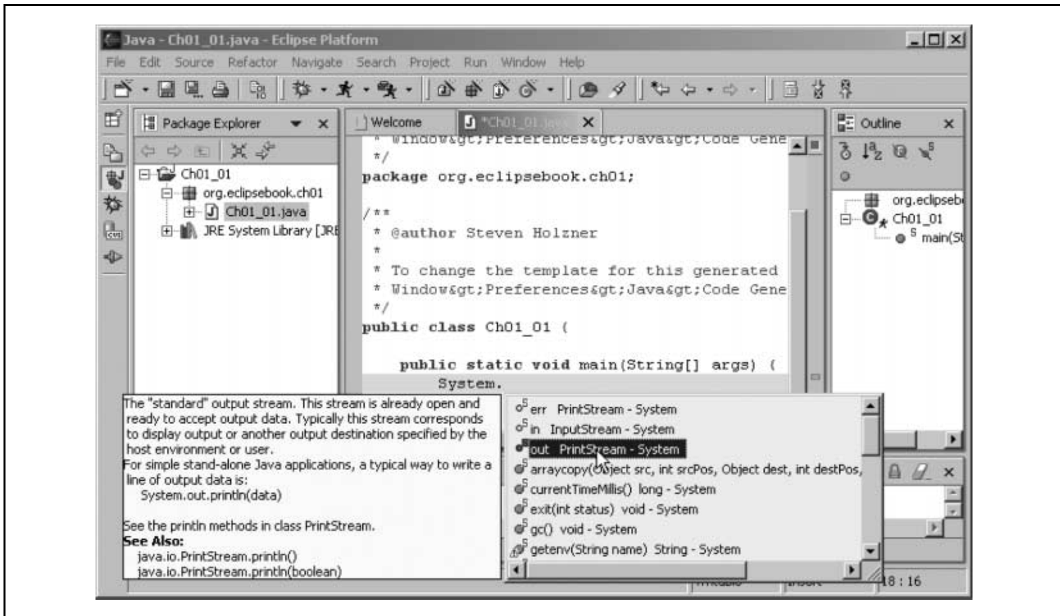
Rysunek 1.11. Wygenerowana klasa Ch01_01

Jak dotąd, całkiem nieźle — mamy już klasę oraz metodę main. Uzupełnijmy ją teraz o zaprezentowany wcześniej kod.

Asystent wprowadzania kodu

Edytor kodu JDT nie odbiega wyglądem od innych edytorów służących do wpisywania kodu źródłowego. Zawiera jednak wiele cech, które czynią go dużo bardziej funkcjonalnym. Na przykład JDT posiada funkcję asystenta wprowadzania kodu (ang. *code assist*), której zadaniem jest pomoc w uzupełnianiu kodu, który właśnie wpisujemy. Zalety tego rozwiązania widoczne są już po kilku chwilach od rozpoczęcia pisania.

Załóżmy, że chcemy wpisać wiersz: System.out.println("Bez obaw!") wewnątrz metody main. Przenosimy kursor do jej wnętrza i wpisujemy System. , a następnie przerywamy na moment. Asystent już po chwili wyświetli listę pól i metod dostępnych w ramach pakietu System, tak jak widać to na rysunku 1.12.



Rysunek 1.12. *Asystent wprowadzania kodu*

Jeśli wskażemy na tej liście pole `out`, po chwili zobaczymy opis tego elementu, a podwójne kliknięcie spowoduje wstawienie pola do naszego kodu źródłowego. Teraz dopisujemy kropkę (mamy już `System.out.`) i ponownie czekamy przez chwilę — asystent wprowadzania znowu podpowiada listę metod, które mogą być w tym momencie zastosowane. Klikamy dwa razy `println(String arg0)` i już mamy prawie gotowy wiersz kodu, który chcieliśmy wstawić do metody `main`:

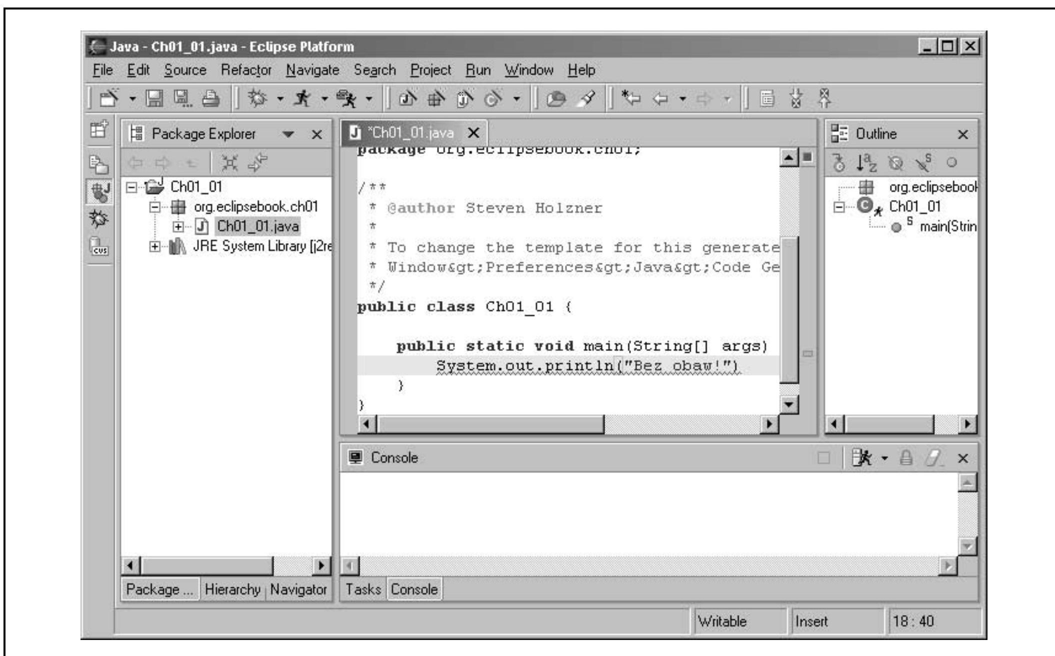
```
public class Ch01_01
{
    public static void main(String[] args) {
        System.out.println()
    }
}
```

Pozostaje nam już tylko wpisać tekst `"Bez obaw!"` (nawet tutaj asystent dopisze za nas drugi cudzysłów):

```
public class Ch01_01
{
    public static void main(String[] args) {
        System.out.println("Bez obaw!")
    }
}
```

Jednak edytowana przez nas linia cały czas jest podkreślona na czerwono, a to oznacza, że nie wszystko jest w porządku. By się dowiedzieć, w czym rzecz, umieszczamy kursor myszy na wskazanym wierszu, co powoduje, że pojawi się opis błędu. Rysunek 1.13 pokazuje, że w naszym przypadku chodzi o brak średnika na końcu linii.

Gdy brak średnika zostanie uzupełniony, czerwone podkreślenie zniknie — linia została uznana za poprawną systaktycznie:



Rysunek 1.13. Wskazówka mówiąca o braku średnika na końcu linii

```
public class Ch01_01
{
    public static void main(String[] args) {
        System.out.println("Bez obaw!");
    }
}
```

Nasz kod jest już kompletny. Spójrzmy na przeglądarkę pakietów z rysunku 1.13 — widzimy tam pełną strukturę naszego projektu: poczynając od nazwy pakietu i pliku z kodem źródłowym, poprzez nazwę klasy `Ch01_01`, a na metodzie `main` tej klasy kończąc. W ten sposób przeglądarka pakietów daje nam najłatwiejszy dostęp do wszystkich elementów projektu, a przeniesienie ich do edytora odbywa się za pomocą podwójnego kliknięcia danego obiektu.

Innym poręcznym sposobem na odszukanie i wyświetlenie wszystkich składników klasy bądź obiektu jest wskazanie go w edytorze, wywołanie menu kontekstowego prawym przyciskiem myszy i wybranie z niego opcji *Open Type Hierarchy*. W odpowiedzi na to w widoku *Hierarchy* zobaczymy hierarchię klas, z których wskazany element się wywodzi, a poniżej wszystkie jego składniki, czyli pola i metody wraz z listą parametrów. Poruszając się w ramach tej struktury, edytor pokazywał będzie również kod źródłowy poszczególnych elementów (chyba, że kod źródłowy nie jest dostępny, gdy dany element znajduje się w archiwum JAR, na przykład obiekt `System.out` zawarty jest w bibliotece `rt.jar`, nie mamy więc dostępu do kodu źródłowego. Istnieją jednak sposoby, by skojarzyć z danym archiwum JAR odpowiedni katalog zawierający kody źródłowe).

Widzimy więc, że kodowanie jest wyraźnie uproszczone dzięki asystentowi wprowadzania kodu, który znał wszystkie metody dostępne z poziomu `System.out`. Nasza rola sprowadziła się do wyboru jednej z propozycji. Asystent automatycznie wywoływany jest po wpisaniu znaku kropki (`.`). Możemy go również wywołać na żądanie, służy do tego kombinacja klawiszy *Ctrl+Spacja*.

Istnieje również możliwość wyłączenia opcji podpowiadania kodu. Jeśli dojdziemy do wniosku, że asystent jest zbędny, należy skorzystać z polecenia *Window/Preferences*. W drzewie znajdującym się w lewej części okienka dialogowego wskazujemy element *Java/Editor*, a nastepnie przechodzimy na zakładkę *Code Assist*. Tam znajdziemy wszystkie opcje dotyczące asystenta wprowadzania kodu.

Podczas edycji pliku źródłowego, takiego jak nasz *Ch01_01.java*, możemy zauważyć symbol gwiazdki, który widoczny jest na zakładce zaraz przed nazwą pliku (rysunek 1.13). Oznacza to, że ostatnio wprowadzone zmiany nie zostały jeszcze zapisane na dysku. Istnieje wiele sposobów na zapisanie pliku. Można w tym celu kliknąć ikonę z symbolem dyskietki (*Save*) bądź tę znajdującą się obok niej (*Save As*); można wywołać z poziomu edytora jego menu kontekstowe i wybrać polecenie *Save* bądź też skorzystać z menu *File* aplikacji (polecenia *Save*, *Save As* oraz *Save All*).

Jesteśmy już w momencie, w którym mamy gotowy i zapisany na dysku kod źródłowy. A jak go uruchomić?

Uruchamianie programu

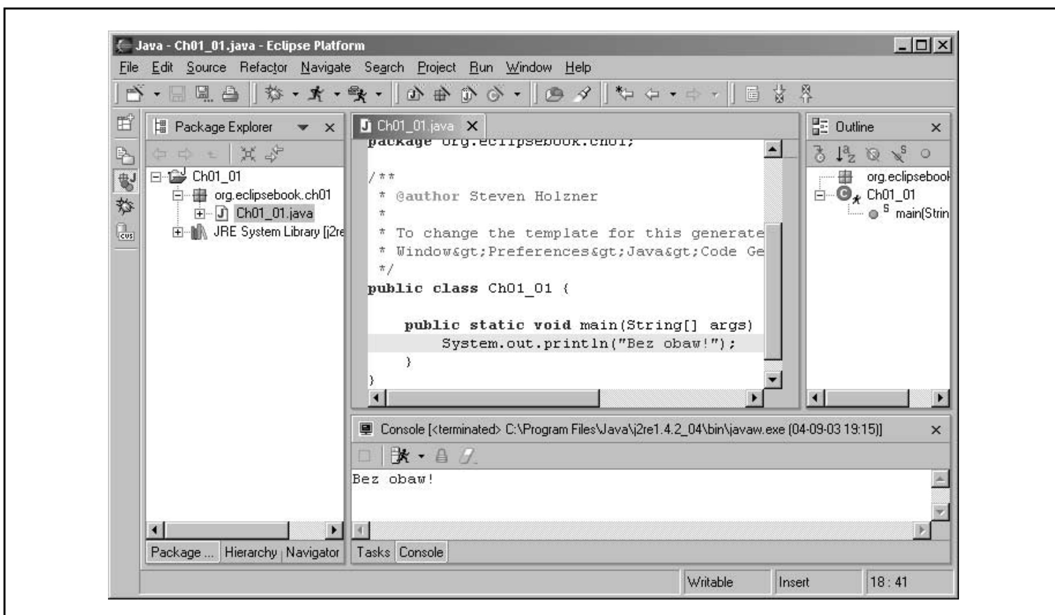
Przyjrzyjmy się dokładnie przeglądarce pakietów pokazanej na rysunku 1.13. Przy klasie `Ch01_01` widać symbol będący kółkiem zawierającym literę *C*, a w dolnym, prawym jego rogu zobaczymy niewielką, biegnącą postać. W ten sposób Eclipse wskazuje klasy, które można uruchomić, czyli takie, które zawierają publiczną, statyczną metodę `main` o odpowiedniej sygnaturze. Aby uruchomić nasz przykładowy program, należy albo wybrać polecenie menu *Run/Run As/Java Application*, albo na listwie narzędziowej kliknąć strzałkę znajdującą się obok ikonki z biegnącą postacią i z menu, które się w odpowiedzi na tę operację pojawi, wybrać opcję *Run As/Java Application*. Jeśli plik źródłowy nie został wcześniej zapisany, Eclipse pokaże stosowną informację. Gdy plik jest zapisany, program zostanie uruchomiony, a wyświetlony przez niego komunikat pojawi się na konsoli. Wynik działania metody `System.out.println("Bez obaw!");` zobaczymy w widoku o nazwie *Console* u dołu ekranu, tak jak na rysunku 1.14.

Gratulujemy — właśnie dołączyłeś do rodziny programistów Eclipse!

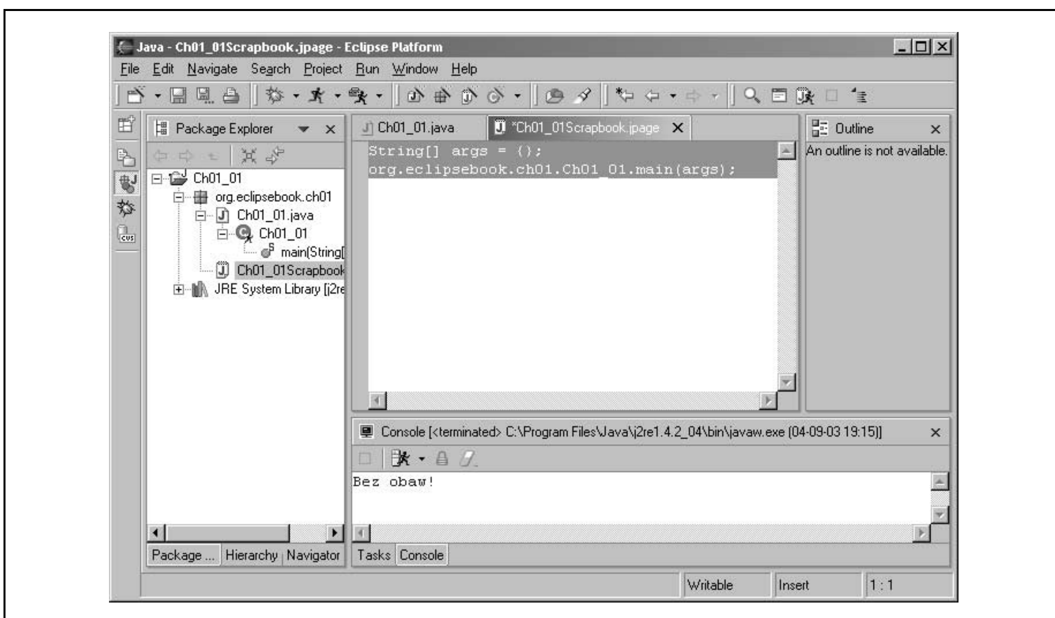
Strona uruchamiania fragmentów kodu (ang. Scrapbook)

Istnieje jeszcze jeden, bardzo interesujacy sposób na uruchamianie programów, a co więcej, nie jest w nim wymagana metoda `main`. Zamiast tego możemy użyć strony uruchamiania fragmentów kodu. Pozwala nam ona na uruchomienie kodu w inny niż dotychczas sposób, co więcej, możemy uruchamiać wskazane fragmenty aplikacji — jest to bardzo przydatne w procesie tworzenia oprogramowania. I choć nie jest to funkcja podstawowa i niezbędna, w pewnych sytuacjach może okazać się bardzo pomocna.

Tworzenie nowej strony uruchamiania fragmentów kodu rozpoczynamy od użycia polecenia menu o nazwie *File/New/Scrapbook Page*. Wyświetli się wtedy okno dialogowe zatytułowane *New Scrapbook Page*. W jego polu *File name* wpisujemy `Ch01_01Scrapbook`, a następnie klikamy przycisk *Finish*. W ten sposób stworzony został plik o nazwie *Ch01_01Scrapbook.jpage*. Pojawił się w przeglądarce pakietów, został też automatycznie otwarty na nowej zakładce edytora (rysunek 1.15).



Rysunek 1.14. Program Ch01_01 w działaniu



Rysunek 1.15. Strona uruchamiania wybiórczego

Teraz możemy wpisać kod, który pozwoli nam uruchomić i sprawdzić wybrany fragment projektu (ma to oczywiście sens przy dużych, złożonych aplikacjach; teraz jedynie prezentujemy ten mechanizm). Na przykład wpiszmy na stronie przedstawione poniżej dwa wiersze kodu — zaznaczamy, że konieczne jest podanie pełnej nazwy pakietu, do którego testowana klasa należy:

```
String[] args = {};
org.eclipsebook.ch01.Ch01_01.main(args);
```

By wskazać, który kod należy uruchomić, musimy zaznaczyć wprowadzone dwa wiersze (tak jak na rysunku 1.15), a nastepnie kliknąć prawym przyciskiem myszy, by z menu kontekstowego wybrać opcję *Execute* (bądź skorzystac z menu: *Run/Execute*). Wynik działania testowanego programu powinien pojawić się w widoku konsoli. W ten sposób, za pomocą strony uruchamiania jesteśmy w stanie sprawdzić pewne fragmenty tworzonego kodu, a rezultaty testów obserwować na konsoli. Zamknięcia strony dokonujemy, podobnie jak w przypadku innych zakładek widocznych w edytorze, za pomocą symbolu X znajdującego się po prawej stronie nazwy pliku.

Jeśli zamiast polecenia *Execute* z menu kontekstowego wybierzemy opcję *Display*, zobaczymy wartość zwróconą przez zaznaczony i uruchomiony kod. Jest to bardzo wygodny i szybki sposób testowania pojedynczych metod, bez konieczności uruchamiania całego programu. Co więcej, kod wpisywany na strony uruchamiania fragmentów kodu może być bardziej skomplikowany niż w naszym przykładzie. Możemy w nim korzystać z zewnętrznych pakietów dołączonych za pomocą opcji *Set Imports* znajdującej się w menu kontekstowym.

Kilka pierwszych przykładów powinno już Cię przekonać, że JDT to narzędzie bardzo zaawansowane. Pamiętaj, to dopiero początek. Za chwilę pokażemy, jak Eclipse radzi sobie z błędami.

Quick Fix

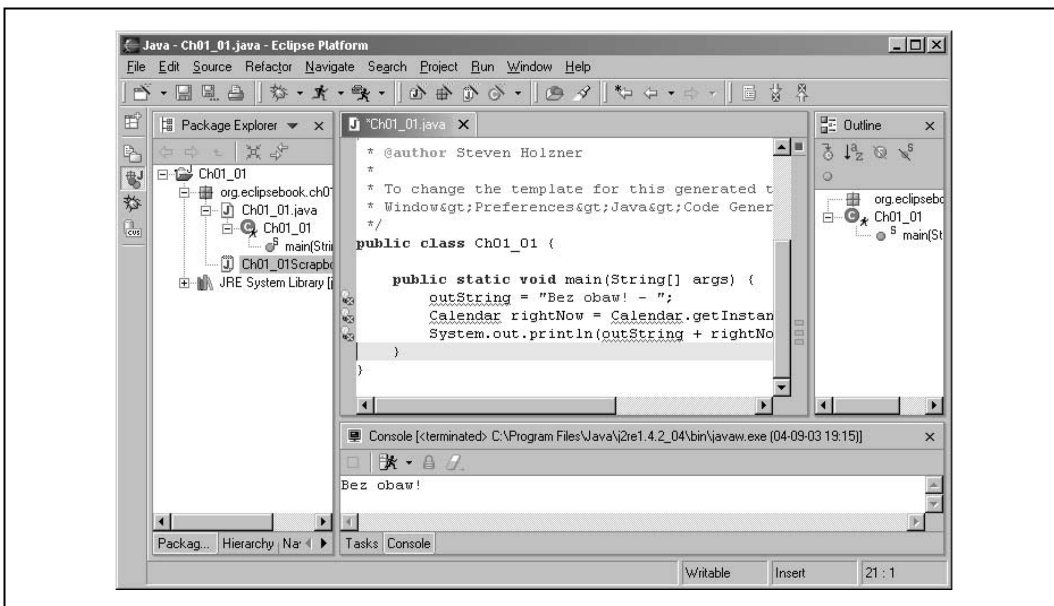
Quick Fix to mechanizm, który pomaga nam radzić sobie z błędami. Jest to jeden z tych elementów, który zdaniem wielu programistów powinien być dostępny już dawno temu. Załóżmy teraz, że chcemy nasz przykład rozszerzyć o wyświetlanie bieżącej daty i czasu:

```
public class Ch01_01
{
    public static void main(String[] args) {
        outString = "Bez obaw! - ";
        Calendar rightNow = Calendar.getInstance();
        System.out.println(outString + rightNow.getTime());
    }
}
```

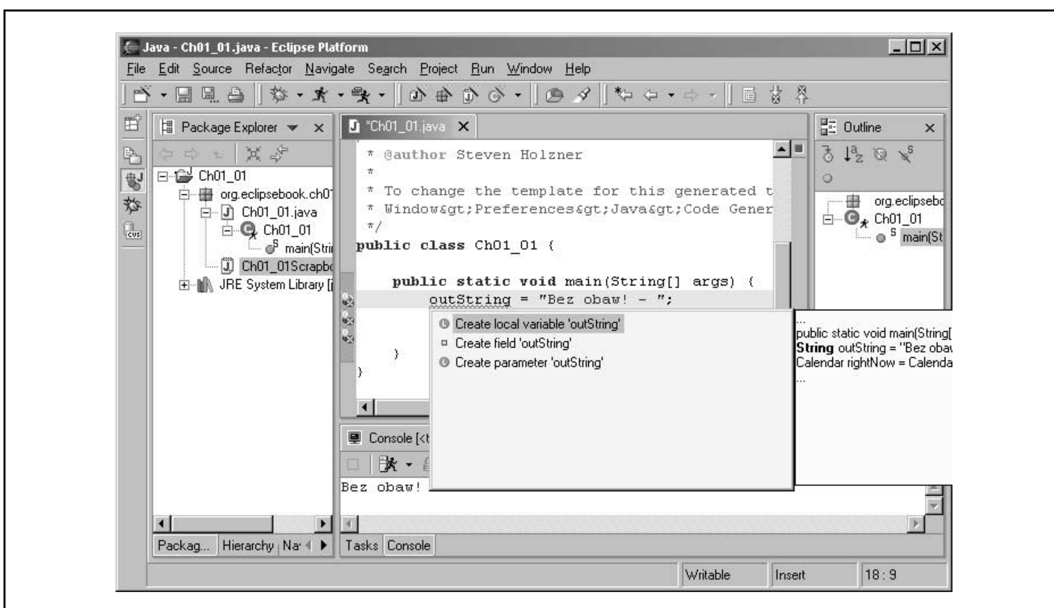
Nie jest tajemnicą, że powyższy fragment zawiera kilka błędów. Zmienna `outString` nie została zadeklarowana, co powoduje, że linie pierwsza i trzecia metody `main` są niepoprawne. Klasa `Calendar` nie jest zaimportowana, zatem drugi wiersz kodu również jest błędny. Gdybyśmy korzystali z kompilatora *javac*, musielibyśmy zakończyć edycję pliku i czekać na werdykt kompilatora, który zwróciłby nam listę błędów. Eclipse natomiast, podkreślając nieprawidłowe fragmenty, informuje nas o błędach natychmiast (rysunek 1.16).

Co więcej, Eclipse nie umywa rąk, mówiąc jedynie: „błąd", ale próbuje wskazać nam sposób jego eliminacji. Zwróć uwagę na symbole przedstawiające żółtą żarówkę i czerwony krzyżyk po lewej stronie wierszy zawierających błędy. Oznacza to, że Quick Fix jest w stanie nam w tych miejscach pomóc. Po prawej stronie, na wysokości tych wierszy, widać czerwone prostokąty — one z kolei pomogą nam nawigować wśród miejsc, które wymagają interwencji.

Jeśli zatrzymamy na chwilę kursor myszy nad pierwszą z pokazanych żarówek, wyświetlony zostanie komentarz opisujący rodzaj błędu, taki jak na rysunku 1.17.



Rysunek 1.16. Działanie mechanizmu Quick Fix



Rysunek 1.17. Quick Fix — komentarz dotyczący błędu

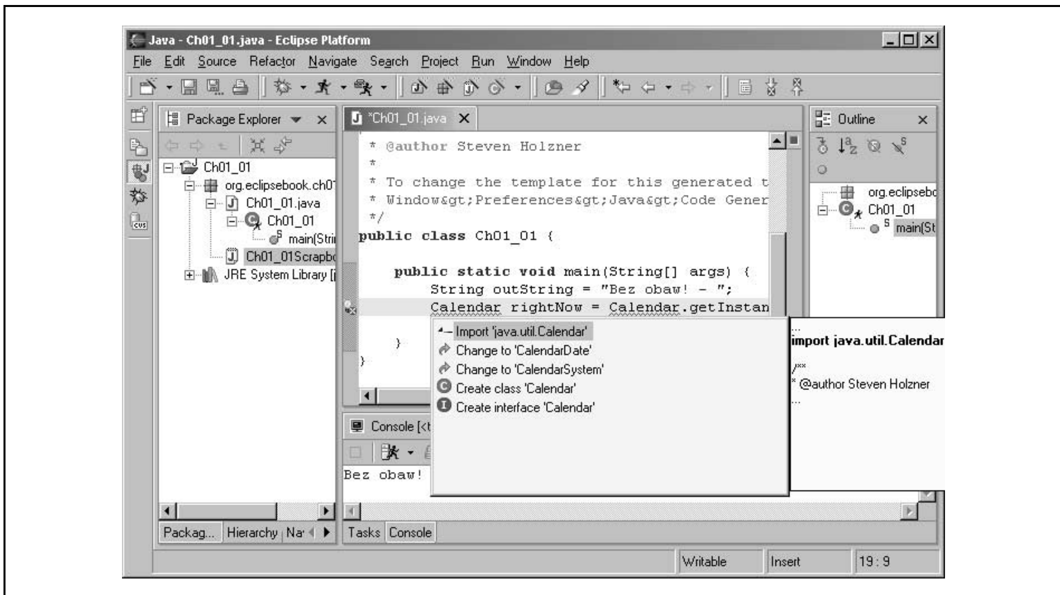
Ale to nie wszystko. Kliknij teraz w tym samym miejscu, a JDT zasugeruje sposób rozwiązania problemu. W naszym przypadku propozycji jest kilka. Pierwsza z nich mówi o konieczności zadeklarowania zmiennej lokalnej `outString`. Na rysunku 1.17 widać, że pokazany jest nawet fragment naszego kodu po uwzględnieniu tej modyfikacji. Najlepsze jest jednak to, że zmiany nie musimy wprowadzać ręcznie, wystarczy podwójne kliknięcie i poprawka jest już naniesiona!

Kod wygląda następująco:

```
public class Ch01_01
{
    public static void main(String[] args) {
        String outString = "Bez obaw! - ";
        Calendar rightNow = Calendar.getInstance();
        System.out.println(outString + rightNow.getTime());
    }
}
```

Ponieważ deklaracja zmiennej `outString` miała wpływ również na ostatnią linię kodu, pozbyliśmy się za jednym zamachem dwóch żarówek.

Pozostał nam już tylko jeden błąd dotyczący klasy `Calendar`. Znajduje się on w drugim wierszu metody `main`. Rysunek 1.18 pokazuje, co tym razem proponuje mechanizm Quick Fix — znowu bezbłędnie sygnalizuje konieczność zaimportowania klasy `Calendar` z pakietu `java.util`. Jak wiele znaczy ten mechanizm, doceni każdy programista, który niejednokrotnie musiał przedzierać się przez hierarchię pakietów Javy. Często bowiem pamiętamy dobrze nazwę klasy, ale nie mamy pojęcia, w jakim pakiecie się ona znajduje.

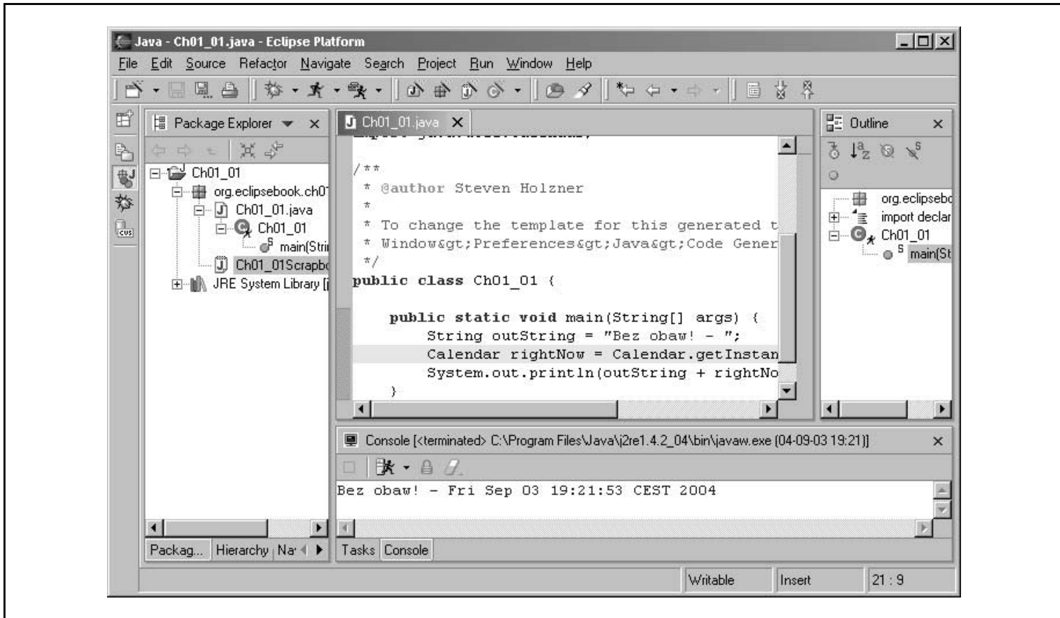


Rysunek 1.18. Automatyczne dołączanie klauzuli importu

I znowu wystarczyło kliknąć myszą, by w kodzie pojawił się wiersz `import java.util.Calendar`. Nasz kod wygląda teraz tak:

```
import java.util.Calendar;
    .
    .
    .
public class Ch01_01
{
    public static void main(String[] args) {
        String outString = "Bez obaw! - ";
        Calendar rightNow = Calendar.getInstance();
        System.out.println(outString + rightNow.getTime());
    }
}
```

Przed chwilą poradziliśmy sobie z ostatnim błędem, tak więc zniknęła ostatnia żarówka. Możemy teraz uruchomić program i zobaczyć na konsoli, jak wygląda zmodyfikowany komunikat (rysunek 1.19).



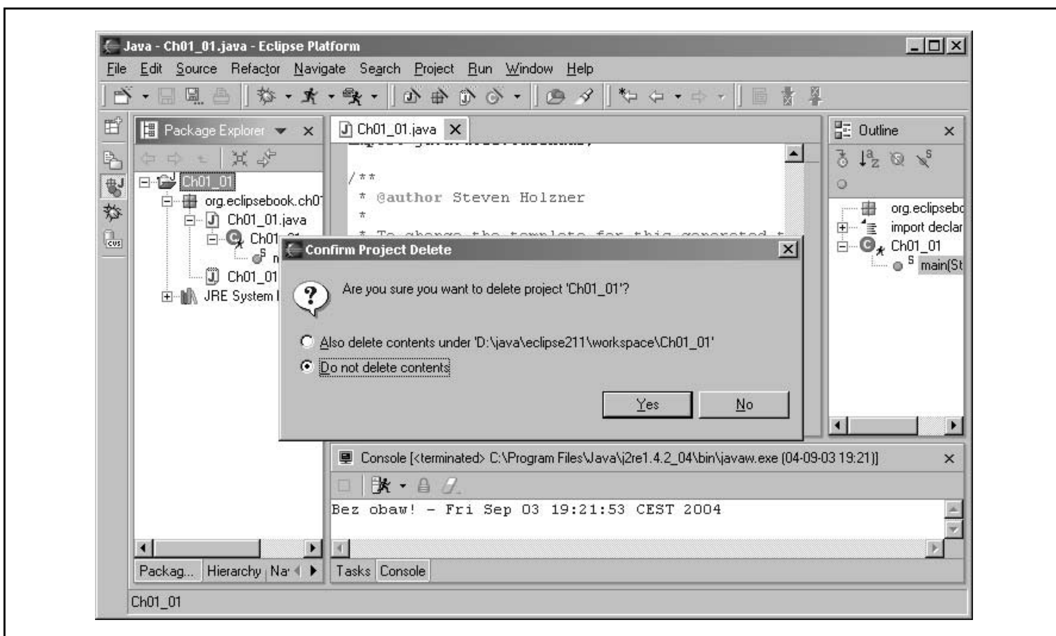
Rysunek 1.19. Nowa wersja programu w działaniu

Słowo o zarządzaniu projektami

Gdy stworzymy więcej projektów takich jak `Ch01_01`, zacznie się okazywać, że Eclipse staje się coraz bardziej zatłoczone. Dzieje się tak, ponieważ wszystkie projekty wyświetlone są w widoku przeglądarki pakietów (*Package Explorer*) i nawigatora (*Navigator*). Przypominamy, że widok nawigatora służy do przeglądania projektów i różni się nieco o przeglądarki pakietów. Tak więc przy 30 projektach zacznie być już ciasno. Jest kilka sposobów na rozwiązanie takiej sytuacji (opiszemy je w rozdziale 2., „Tworzenie aplikacji w języku Java"), teraz zaprezentuję wariant najprostszy.

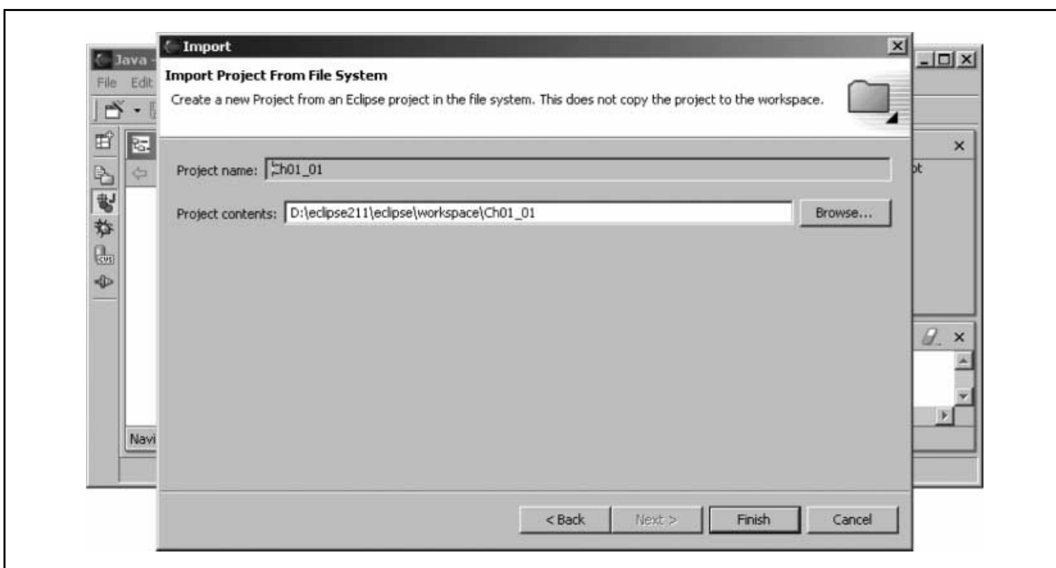
Aby usunąć projekt z przeglądarki pakietów oraz widoku nawigatora, wystarczy go po prostu stamtąd skasować (polecenie *Delete*). Operacja ta nie usuwa jednak fizycznych plików składających się na projekt. Możemy w każdej chwili do niego powrócić. Na przykład, chcąc pozbyć się projektu `Ch01_01`, klikamy prawym przyciskiem myszy odpowiedni element widoku i z menu kontekstowego wybieramy polecenie *Delete*. Eclipse wyświetli okno dialogowe o nazwie *Confirm Project Delete* (rysunek 1.20).

Jeżeli chcemy zastosować wariant opisany powyżej, musimy się upewnić, że zaznaczona jest opcja *Do not delete contents*, dzięki czemu mamy pewność, że żaden istniejący plik nie zostanie fizycznie usunięty. Po potwierdzeniu naszego zamiaru kliknięciem przycisku *Yes* wskazany projekt zniknie z widoków przeglądarki pakietów i nawigatora. Oczywiście, jeśli zdecydujemy się zaznaczyć pierwszą opcję: *Also delete contents under…*, cały nasz dorobek stracimy bezpowrotnie.



Rysunek 1.20. Usunięcie projektu

Po usunięciu projektu z widoków możemy do niego w każdej chwili powrócić, czyli go zaimportować. W tym celu ze znanego nam już menu kontekstowego wybieramy polecenie *Import* (możemy również skorzystać z menu — opcja *File/Import*). W wyniku tej operacji wyświetlony zostanie dialog *Import*. Wskazujemy element *Existing Project into Workspace* i wciskamy przycisk *Next*. Na następnym ekranie korzystamy z przycisku *Browse…* i wskazujemy katalog *Ch01_01* znajdujący się w przestrzeni projektów (rysunek 1.21).



Rysunek 1.21. Import istniejącego projektu

Klikając *Finish*, doprowadzamy do stanu, w którym `Ch01_01` ponownie jest widoczny i gotowy do użycia. Sprawdzimy to, przechodząc do widoku przeglądarki pakietów i wywołując polecenie *Run/Run As/Java Application*. Opisany sposób radzenia sobie z nadmiarem projektów nie jest może wyszukany, ale za to skuteczny. Inne zaprezentujemy podczas dalszej lektury.

Ciekawą opcję stanowi polecenie *Close* znajdujące się w menu kontekstowym przeglądarki pakietów. Służy ono do zamknięcia projektu. Nie usuwa ono projektu z widoku, ale zwija wszystkie jego składniki i cały projekt sprowadza do pojedynczej ikonki i jego nazwy. Zaletą tego jest szybszy start Eclipse, bowiem środowisko podczas uruchamiania nie inicjalizuje zamkniętych projektów.

Rozdział pierwszy dobiega końca. To dopiero początek naszej znajomości z Eclipse. W następnym rozdziale poznamy kolejne szczegóły związane z używaniem JDT do tworzenia aplikacji w języku Java.