Technologia i rozwiązania

Eclipse 4
Programowanie wtyczek na przykładach

Rozszerz możliwości środowiska Eclipse!

Dr Alex Blewitt

Helion

[PACKT] open source* PUBLISHING

Tytuł oryginału: Eclipse 4 Plug-in Development by Example: Beginner's Guide

Tłumaczenie: Rafał Jońca

ISBN: 978-83-246-8754-1

First published in the English language under the title „Eclipse 4 Plug-in Development by Example: Beginner's Guide".

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: http://helion.pl (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/eclip4
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

3

ROZDZIAŁ

Tworzenie widoków w JFace

W poprzednim rozdziale przyjrzeliśmy się podstawowym elementom SWT, które stanowią pomost między elementami systemu operacyjnego a Javą. W tym rozdziale poznamy JFace, który korzysta z SWT w celu zapewnienia architektury MVC, a także dostarczenia wielu typowych widgetów używanych przez Eclipse.

W tym rozdziale:

- utworzymy widok do przedstawiania hierarchicznych danych,
- użyjemy zasobów obrazu, czcionki lub koloru,
- wygenerujemy stylizowany tekst,
- posortujemy i przefiltrujemy wpisy w widokach,
- dodamy akcje dla podwójnych kliknięć,
- zaznaczymy i obsłużymy właściwości,
- utworzymy widok dla danych tabelarycznych.

Dlaczego JFace?

Choć SWT zapewnia podstawową implementację prostych widgetów (na przykład drzew, przycisków i etykiet), wszystko działa na bardzo podstawowym poziomie, bo wykorzystywane są teksty i indeksy zaznaczeń. Aby łatwiej wyświetlać strukturyzowane dane, JFace udostępnia kilka zaawansowanych widoków, które stanowią połączenie widgetów SWT i menedżerów zdarzeń, co zapewnia wygodną obsługę interfejsu użytkownika dla strukturyzowanych treści.

Istnieje wiele rodzajów zaawansowanych widoków nazywanych *viewer* (wszystkie dziedziczą po klasie `Viewer`), ale najczęściej stosowanymi są te, które należą do `ContentViewer`, na przykład `TreeViewer` i `TableViewer`. Istnieją również wersje bazujące na tekście (`TextViewer` ma podklasy dla `SourceViewer`), a także widoki operacyjne (`ConsoleViewer` dla widoku *Console* lub `DetailedProgressViewer` dla widoku *Progress*). W tym rozdziale wykonamy widoki bazujące na klasach `TreeViewer` i `TableViewer`. Ponieważ JFace bazuje na SWT, wiedza na temat szczegółów działania SWT jest niezbędna do prawidłowego użytkowania JFace.

Tworzenie widoków TreeViewer

Wiele widgetów w Eclipse bazuje na widoku przypominającym drzewo — jest to zarówno nawigator plików, jak i okno wyświetlania zawartości klas. Framework JFace udostępnia klasę `TreeViewer` realizującą wszystkie niezbędne funkcjonalności. Użyjemy jej do wykonania widoku `TimeZoneTreeView`.

Kroki do wykonania — tworzenie obiektu TreeViewer

Podobnie jak miało to miejsce w poprzednim rozdziale, nowy widok `TimeZoneTreeView` utworzymy przy użyciu edytora *plugin.xml*. Widok wyświetli strefy czasowe ułożone hierarchicznie względem regionów.

1. Kliknij prawym przyciskiem myszy projekt *com.packtpub.e4.clock.ui* i wybierz polecenie *Plug-in Tools/Open Manifest*, by otworzyć plik *plugin.xml*, jeśli jeszcze nie jest otwarty.
2. Otwórz zakładkę *Extensions* i znajdź element *org.eclipse.ui.views*. Kliknij go prawym przyciskiem myszy i z menu wybierz polecenie *New/View*. Wypełnij pola w sposób opisany poniżej.
   - W polu *ID* wpisz **`com.packtpub.e4.clock.ui.views.TimeZoneTreeView`**.
   - W polu *Name* wpisz **`Widok drzewa stref czasowych`**.
   - W polu *Class* wpisz **`com.packtpub.e4.clock.ui.views.TimeZoneTreeView`**.
   - W polu *Category* wpisz **`com.packtpub.e4.clock.ui`**.
   - W polu *Icon* wpisz **`icons/sample.gif`**.
3. Zapisz plik. Konfigurator umieścił w pliku *plugin.xml* następujący wpis.

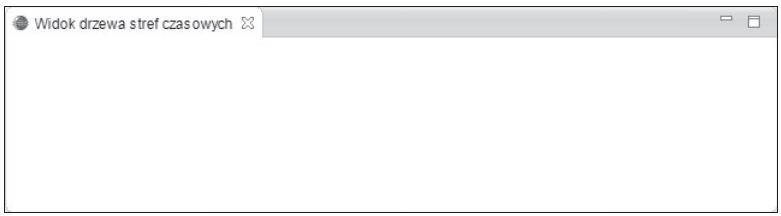
```
<view
   category="com.packtpub.e4.clock.ui"
   class="com.packtpub.e4.clock.ui.views.TimeZoneTreeView"
   icon="icons/sample.gif"
   id="com.packtpub.e4.clock.ui.views.TimeZoneTreeView"
   name="Widok drzewa stref czasowych"
   restorable="true">
</view>
```

4. Podobnie jak wcześniej, utwórz klasę `TimeZoneTreeView`, która rozszerza klasę `ViewPart`.
5. W metodzie `createPartControl()` utwórz instancję `TreeViewer` z opcjami `V_SCROLL`, `H_SCROLL` i `MULTI`. Zapamiętaj obiekt w polu klasy. Zaimplementuj metodę `setFocus()`, by ustawiała widok drzewa jako aktywny element.

```
public class TimeZoneTreeView extends ViewPart {
   private TreeViewer treeViewer;
   public void createPartControl(Composite parent) {
      treeViewer = new TreeViewer(parent, SWT.MULTI | SWT.H_SCROLL |
      ↪SWT.V_SCROLL );
   }
   public void setFocus() {
      treeViewer.getControl().setFocus();
   }
}
```

☞ E4: Choć Eclipse 4 zostanie szczegółowo omówione w rozdziale 7., warto wspomnieć, że w Eclipse 4 nad metodą `createPartControl()` niezbędna jest adnotacja `@Inject` (by zapewnić przekazanie obiektu `Composite`), a nad metodą `setFocus()` — adnotacja `@Focus`.

6. Uruchom testową wersję Eclipse i przejdź do widoku, wybierając polecenie *Window/ShowView/Other/Śledzenie czasu/Widok drzewa stref czasowych*.



7. W odróżnieniu od Swing, gdzie oczekuje się otrzymywania danych w klasie bazującej na konkretnym interfejsie, JFace nie wymaga żadnej konkretnej klasy. W zamian oczekuje obiektu wartości do wyświetlenia (wejście), interfejsu, który czyta dane (dostawca treści), i interfejsu do wyświetlania danych (dostawca etykiet).
8. Utwórz nową klasę o nazwie `TimeZoneLabelProvider` dziedzicząca po `LabelProvider` (z pakietu `org.eclipse.jface.viewers`). Będzie zawierała metodę o nazwie `getText()`, która otrzymuje obiekt i zamienia go na reprezentację tekstową. Zamiast wywoływać `toString()`, zwróć odpowiednią wartość związaną z `Map.Entry` lub `TimeZone`.

```
public class TimeZoneLabelProvider extends LabelProvider {
   public String getText(Object element) {
      if (element instanceof Map) {
         return "Strefy czasowe";
```

```
            } else if (element instanceof Map.Entry) {
               return ((Map.Entry) element).getKey().toString();
            } else if (element instanceof TimeZone) {
               return ((TimeZone) element).getID().split("/")[1];
            } else {
               return "Nieznany typ: " + element.getClass();
            }
         }
      }
```

Ponieważ obiekt `TreeViewer` może mieć wiele korzeni, test `instanceof Map` służy do sprawdzenia, czy to wierzchołek drzewa, który powinien mieć nazwę `Strefy czasowe`.

9. Warto zapewnić wartość domyślną — nawet jeśli jest to pusty tekst — bo otrzymanie nieznanego typu można łatwo wyśledzić i naprawić.
10. Utwórz nową klasę `TimeZoneContentProvider` implementującą interfejs `ITreeContentProvider`. Interfejs wymaga implementacji trzech metod z sześciu (pozostałe mogą pozostać puste). Oto one.
    - `hasChildren()` — zwraca `true`, jeśli węzeł ma potomków.
    - `getChildren()` — zwraca potomków konkretnego węzła.
    - `getElements()` — zapewnia główne korzenie.
11. Metoda `hasChildren()` zwróci wartość `true`, jeśli zostanie do niej przekazany obiekt typu `Map` lub `Collection`, który nie będzie pusty. Przekazanie `Map.Entry` spowoduje wywołanie rekurencyjne. Dla drzew bazujących na zagnieżdżonych `Map` lub `Collection` metoda `hasChildren()` będzie wyglądała identycznie.

```
public boolean hasChildren(Object element) {
   if (element instanceof Map) {
      return !((Map) element).isEmpty();
   } else if (element instanceof Map.Entry) {
      return hasChildren(((Map.Entry)element).getValue());
   } else if (element instanceof Collection) {
      return !((Collection) element).isEmpty();
   } else {
      return false;
   }
}
```

12. Implementacja `getChildren()` rekurencyjnie wchodzi do obiektów typu `Map`, `Collection` lub `Map.Entry`, stosując przy tym opisany wcześniej wzorzec. Ponieważ metoda wymaga zwrócenia typu `Object[]`, kod używa funkcjonalności wbudowanej w klasę `Map`, by zamienić zawartość `entrySet()` na tablicę.

```
public Object[] getChildren(Object parentElement) {
   if (parentElement instanceof Map) {
      return ((Map) parentElement).entrySet().toArray();
```

```
    } else if (parentElement instanceof Map.Entry) {
       return getChildren(((Map.Entry)parentElement).getValue());
    } else if (parentElement instanceof Collection) {
       return ((Collection) parentElement).toArray();
    } else {
       return new Object[0];
    }
  }
```

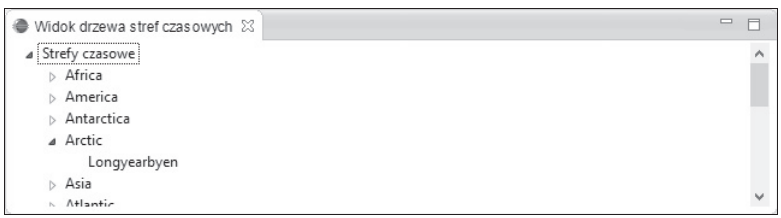
13. Kluczem przy implementacji `ITreeContentProvider` jest zapewnienie stałej synchronizacji kodu metod `getChildren()` i `hasChildren()`. Jednym ze sposobów zapewnienia takiej sytuacji jest użycie w metodzie `hasChildren()` metody `getChildren()` i sprawdzanie, czy tablica jest pusta, ale wydajność takiej operacji nie będzie najlepsza, jeśli `getChildren()` to operacja bardzo złożona obliczeniowo.
14. Ponieważ `TreeViewer` może mieć wiele korzeni, istnieje metoda pobierająca tablicę korzeni z elementu wejściowego. Błąd we frameworku JFace uniemożliwia argumentowi `getElements()` posiadanie własnej wartości. Z tego powodu przyjęło się, że najlepiej przekazać tablicę (zawierającą tylko jeden element) i następnie ją zwrócić. Metoda przedstawiona poniżej będzie najprawdopodobniej wyglądała tak samo dla każdej klasy `TreeContentProvider`, którą kiedykolwiek napiszesz.

```
public Object[] getElements(Object inputElement) {
   if (inputElement instanceof Object[]) {
      return (Object[]) inputElement;
   } else {
      return new Object[0];
   }
}
```

15. Po zakończeniu tworzenia odpowiednich klas dostawców danych przejdź do metody `createPartControl()` klasy `TimeZoneTreeView`, by połączyć dostawców z obiektem widoku i ustalić obiekt będący źródłem danych.

```
treeViewer.setLabelProvider(new TimeZoneLabelProvider());
treeViewer.setContentProvider(new TimeZoneContentProvider());
treeViewer.setInput(new Object[]{TimeZoneComparator.getTimeZones()});
```

16. Uruchom testową wersję Eclipse i otwórz widok poleceniem *Window/ShowView/Other/Śledzenie czasu/Widok drzewa stref czasowych*, by zobaczyć efekt końcowy.



Co się stało?

Dane do `TreeViewer` przekazaliśmy przy użyciu metody `setInput()`. Metoda prawie zawsze otrzymuje tablicę obiektów, nawet jeśli jest to tylko jeden element.

Aby zapewnić rozpakowanie struktury danych, interfejs `ITreeContentProvider` udostępnia dwie kluczowe metody — `hasChildren()` i `getChildren()`. Umożliwiają one przechodzenie przez strukturę danych na żądanie, gdy użytkownik zwija lub rozwija gałęzie drzewa. Powodem istnienia dwóch metod jest fakt, iż obliczenia w metodzie `getChildren()` mogą być bardzo kosztowne. Metoda `hasChildren()` służy do sprawdzenia, czy należy wyświetlić ikonę rozwinięcia węzła. Wywołanie metody `getChildren()` jest opóźnione aż do momentu faktycznego otwarcia węzła.

> W strukturach danych, które to zapewniają, warto również zaimplementować metodę `getParent()`; umożliwia ona dostęp do obiektu. Jeśli jest zaimplementowana, wywołanie `viewer.reveal(Object)` powoduje rozwinięcie węzłów w hierarchii, by odsłonić wskazany obiekt.

Do wyświetlenia etykiet na drzewie służy klasa `LabelProvider`. Dostarcza ona etykietę (i opcjonalny obrazek) dla każdego elementu. Dla każdego typu obiektu można użyć innej ikony. Z rozwiązania tego skorzystano w widoku *Package* z perspektywy *Java*, który wyświetla ikonę klasy dla klas, ikonę pakietu dla pakietów i tak dalej.

Klasa `LabelProvider` może wyświetlać komunikaty na różne sposoby. Nic nie stoi na przeszkodzie, by dodać do etykiety informację o przesunięciu czasowym (różnicę między konkretną strefą czasową i czasem GMT).

Kroki do wykonania — JFace i obrazy

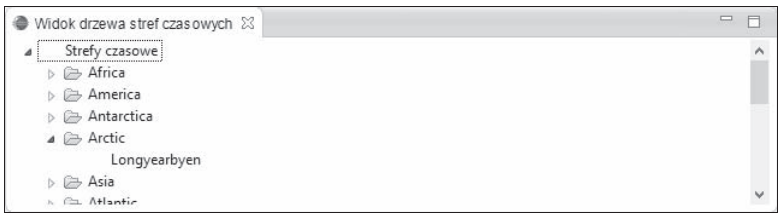
Klasa `TimeZoneLabelProvider` może zwrócić obiekt `Image` będący standardowym widgetem SWT. Choć obraz (obiekt `Image`) można wczytać w sposób podobny jak w poprzednim rozdziale, JFace oferuje rejestry zasobów służące do zarządzania zestawami zasobów aplikacji. Rejestry obsługują klasy `ImageRegistry`, `FontRegistry` i `ColorRegistry`. Rejestr zasobów ma za zadanie przechowywać listę obiektów `Resource` i zwalniać je we właściwy sposób, ale tylko wtedy, gdy nie są już potrzebne.

JFace posiada rejestry globalne, ale istnieją również rejestry bardziej szczegółowe używane przez IDE na przykład do przechowywania list ikon folderów i plików. W rejestrze tego typu korzysta się z deskryptorów do określania konkretnych zasobów, więc po przekazaniu deskryptora otrzymuje się odpowiadającą mu instancję zasobu. Zwróconym zasobem zarządza rejestr, więc kod, który go otrzyma, nie powinien go zwalniać.

1. W `TimeZoneLabelProvider` dodaj metodę `getImage()`, w której używa się rejestru obrazów `ImageRegistry`, by pobrać ikonę folderu. Oto kod metody.

```
public Image getImage(Object element) {
   if(element instanceof Map.Entry) {
      return
PlatformUI.getWorkbench().getSharedImages().getImage(ISharedImages.IMG_OBJ_FOLDER);
   } else {
      return super.getImage(element);
   }
}
```

2. Uruchom testową wersję Eclipse i otwórz *Widok drzewa stref czasowych*. Obok tekstu z nazwą regionu pojawi się ikona folderu. Instancji `Image` nie trzeba niszczyć samodzielnie, ponieważ należy do wtyczki `PlatformUI` (zasób obrazu zostanie zwolniony w momencie wyłączania `PlatformUI`).



☞ E4: W Eclipse 4 instancję `ISharedImages` można otrzymać poprzez wstrzyknięcie.

```
@Inject
private ISharedImages images;
images.getImage(ISharedImages.IMG_OBJ_FOLDER);
```

3. By otrzymać inny obraz, użyj globalnych instancji `ImageRegistry` lub `JFaceRegistry` lub utwórz własną kopię. Zastosowanie globalnej wersji oznacza, że obraz `Image` nigdy nie zostanie zniszczony, ponieważ `JFaceRegistry` istnieje przez cały czas życia instancji Eclipse.

   Zamiast tego utwórz obiekty `LocalResourceManager` i `ImageRegistry` powiązane z cyklem życia kontrolki. Gdy kontrolka nadrzędna będzie usuwana, automatycznie usunięte zostaną również obrazy. Umieść w metodzie `CreatePartControl` klasy `TimeZoneTreeView` poniższy kod.

```
public void createPartControl(Composite parent) {
   ResourceManager rm = JFaceResources.getResources();
   LocalResourceManager lrm = new LocalResourceManager(rm,parent);
```

4. Używając obiektu `LocalResourceManger`, utwórz instancję `ImageRegistry` i za pomocą metody `createFromURL()` dodaj obiekt `ImageDescriptor` na podstawie adresu URL.

```
ImageRegistry ir = new ImageRegistry(lrm);
URL sample = getClass().getResource("/icons/sample.gif");
ir.put("sample", ImageDescriptor.createFromURL(sample));
```

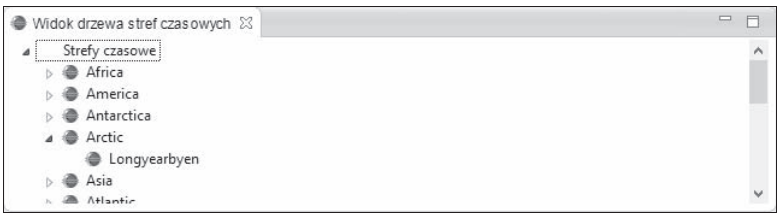
5. Po wypełnieniu obiektu `ImageRegistry` trzeba go powiązać z obiektem `LabelProvider`, by ten mógł użyć właściwego obrazu, jeśli będzie trzeba. Przekaż rejestr obrazów do konstruktora klasy `TimeZoneLabelProvider`.

```
~~treeViewer.setLabelProvider(new TimeZoneLabelProvider());~~
treeViewer.setLabelProvider(new TimeZoneLabelProvider(ir));
```

6. Zaimplementuj w `TimeZoneLabelProvider` konstruktor, który zapamięta obiekt `ImageRegistry`. Następnie użyj go do pobrania obrazu w wywołaniu `getImage()`.

```
private final ImageRegistry ir;
public TimeZoneLabelProvider(ImageRegistry ir) {
   this.ir = ir;
}
public Image getImage(Object element) {
   if(element instanceof Map.Entry) {
      return ir.get("sample");
   } else if(element instanceof TimeZone) {
      return ir.get("sample");
   } else {
      return super.getImage(element);
   }
}
```

7. Ponownie uruchom testową wersję Eclipse. W drzewie pojawi się przykładowa ikona wtyczki.



Co się stało?

Początkowo użyliśmy standardowych obrazów znajdujących się w obiekcie `PlatformUI`. Predefiniowane deskryptory pochodziły z interfejsu `ISharedImages`. Nazwy deskryptorów zaczynają się od `IMG`; zastosowano w nich następujący wzorzec:

- `etool` lub `dtool` — włączone lub wyłączone ikony paska narzędziowego,
- `elcl` lub `dlcl` — włączone lub wyłączone ikony lokalnego paska narzędziowego,
- `dec` — dekorator,
- `obj` i `objs` — obiekty (pliki, foldery i tym podobne).

Inne wtyczki zawierają własne zestawy obrazów, na przykład interfejs JDT dodaje ikony związane z pakietami, klasami, metodami i polami.

W celu użycia własnych obrazów utworzyliśmy obiekt `ImageRegistry` obsługiwany przez obiekt `LocalResourceManager`. Jeśli do konstruktora trafi obiekt `Control`, klasa rejestruje w nim obiekt `DisposeListener`. W ten sposób, gdy kontrola będzie niszczona, podobnie stanie się z powiązanymi z nią obrazami. Dzięki temu cały kod jest bardziej przejrzysty, gdyż `ImageRegistry` można bez większych problemów przekazać do klasy `TimeZoneContentProvider`.

Obiekt `ImageRegistry` inicjalizujemy zestawem obiektów `ImageDescriptor` — w tym przypadku plikiem *icons/sample.gif* pochodzącym z kreatora wtyczek. Ten sam klucz służy do inicjalizacji i dostępu do obrazu. Niektóre projekty Eclipse trzymają się konwencji z interfejsem `ISharedImages` z zestawem stałych.

Kroki do wykonania — style w dostawcy etykiet

Interfejs `IStyledLabelProvider` służy do zmiany domyślnego stylu w widoku drzewa. Użyto go w widoku treści klas, który wyświetla typ zwracany przez metody, lub też w dekoratorze zespołów, który wyświetla informację o zmianach.

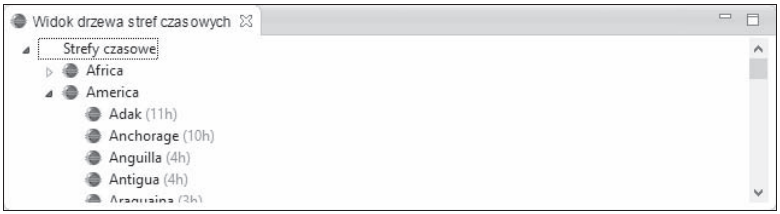
1. Dodaj interfejs `IStyledLabelProvider` do `TimeZoneLabelProvider` i utwórz metodę `getStyledText()`. Jeśli zaznaczonym elementem jest `Map.Entry` zawierający `TimeZone`, w nawiasach wskaż przesunięcie czasowe.

```
public class TimeZoneLabelProvider extends LabelProvider implements
IStyledLabelProvider {
   public StyledString getStyledText(Object element) {
      String text = getText(element);
      StyledString ss = new StyledString(text);
      if (element instanceof TimeZone) {
         int offset = -((TimeZone) element).getOffset(0);
         ss.append(" (" + offset / 3600000 + "h)", StyledString.DECORATIONS_
         ↪STYLER);
      }
      return ss;
   }
}
```

2. By użyć dostawcy etykiet ze zmienionym stylem, trzeba go otoczyć klasą `DelegatingStyledCellLabelProvider`. Zmodyfikuj konstruktor wywoływany w metodzie `createPartControl()` metody `TimeZoneTreeView`.

```
treeViewer.setLabelProvider(
   new DelegatingStyledCellLabelProvider(
      new TimeZoneLabelProvider(ir)));
```

3. Uruchom testową wersję Eclipse i otwórz widok, by zobaczyć przesunięcia czasowe zapisane innym kolorem.



4. By zmienić czcionkę używaną w widoku, klasa `TimeZoneLabelProvider` musi implementować interfejs `IFontProvider`. Klasa `FontRegistry` z JFace umożliwia pobranie domyślnej czcionki z włączoną kursywą. Dodaj parametr `FontRegistry` do konstruktora `TimeZoneLabelProvider` i zaimplementuj metodę `getFont()`.

```
public class TimeZoneLabelProvider extends LabelProviderimplements
➥IStyledLabelProvider, IFontProvider {
   private final FontRegistry fr;
   public TimeZoneLabelProvider(ImageRegistry ir, FontRegistry fr){
      this.ir = ir;
      this.fr = fr;
   }
   public Font getFont(Object element) {
      Font italic = fr.getItalic(JFaceResources.DEFAULT_FONT);
      return italic;
   }
}
```

5. Zmodyfikuj klasę `TimeZoneTreeView`, by utworzyć i przekazać globalny obiekt `FontRegistry` pobrany z klasy `JFaceResources`.

```
FontRegistry fr = JFaceResources.getFontRegistry();
~~treeViewer.setLabelProvider(~~
   ~~new DelegatingStyledCellLabelProvider(~~
      ~~new TimeZoneLabelProvider(ir)));~~
treeViewer.setLabelProvider(
   new DelegatingStyledCellLabelProvider(
      new TimeZoneLabelProvider(ir, fr)));
```

6. Ponownie uruchom testową wersję Eclipse, by przekonać się, że strefy czasowe są teraz pisane kursywą.

Co się stało?

Implementując interfejs `IStyledLabelProvider` i otaczając go klasą `DelegatingStyledCellLabelProvider`, można zmieniać styl poszczególnych elementów drzewa, włącznie ze zmianami czcionki i koloru. Klasa `StyledText` umożliwia wyświetlanie tekstu różnymi stylami.

Choć w przykładzie pojawiła się klasa `DecorationsStyler`, dodatkowe style można także zdefiniować przy użyciu wywołania `StyledString.createColorRegistryStyler("czcionka", "tło")`, gdzie oba teksty to klucze w globalnym obiekcie `ColorRegistry` z JFace.

Choć kolory można zmieniać dla poszczególnych znaków, czcionka (obiekt `Font`) jest jedna dla całego tekstu. Wynika to z faktu, iż wyliczenia rozmiaru etykiety zakładają, że cały tekst jest pisany identyczną czcionką.

Jako dobrą praktykę uważa się używanie przez dostawców treści i etykiet menedżerów zasobów przekazywanych do konstruktorów. Dzięki temu kod łatwo sprawdzić za pomocą testów automatycznych i pozorowanych zasobów. Niezależnie od tego, czy stosuje się model programistyczny Eclipse 3.x, czy Eclipse 4.x, oddzielenie użycia zasobów od miejsca ich tworzenia to klucz do wygodnego testowania.

Quiz — podstawy JFace

P1. Jakie metody zawiera `LabelProvider`?

P2. Jaka jest różnica między metodami `hasChildren()` i `getChildren()` z `ContentProvider`?

P3. Do czego służy klasa `ImageRegistry`?

P4. W jaki sposób zmienić styl elementów widoku drzewa?

Sprawdź się — dodanie obrazów dla regionów

Po poznaniu podstaw postaraj się rozszerzyć przykład o kilka elementów.

- Popraw klasę `TimeZoneLabelProvider`, by podawała przesunięcie w godzinach i minutach względem GMT.
- Uaktualnij wtyczkę, dodając ikony flag i tworząc wpisy w rejestrze obrazów (nazwa strefy czasowej może być kluczem, co ułatwi całą obsługę).
- Wyświetl nazwę regionu kursywą, ale same nazwy stref czasowych pogrubioną czcionką.

Sortowanie i filtracja

Jedną z cech JFace jest to, że za sortowanie danych może odpowiadać widok, co odciąża strukturę danych od odpowiedzialności za właściwe przetwarzanie materiałów. W ten sposób bardzo łatwo utworzyć widoki z filtracją, w których użytkownik szuka określonej frazy lub też sortuje wyniki zgodnie ze swym zapotrzebowaniem. Filtry są powszechnie używane w IDE Eclipse. Przykładem są chociażby opcje *Hide libraries from external* lub *Hide closed projects* znajdujące się w opcjach wielu widoków.

Kroki do wykonania — sortowanie elementów w widoku

Widok drzewa wyświetla obecnie dane w sposób posortowany, ale za sortowanie nie odpowiada widok. Ponieważ dane znajdują się w obiekcie `TreeMap`, wykonuje on automatyczne sortowanie elementów na podstawie wartości zwracanych przez metodę `toString()`. By użyć innego sposobu sortowania (na przykład bazującego na przesunięciu czasu), można albo zmodyfikować obiekt `TreeMap`, dodając nowy komparator i sortując dane przy ich tworzeniu, albo sortować na poziomie widoku drzewa. Pierwszy wybór jest dobry tylko w sytuacji, gdy z danych korzysta jeden widok lub gdy dane pochodzą z dużego, zewnętrznego magazynu danych, który przeprowadzi sortowanie zdecydowanie bardziej efektywnie (takiego jak na przykład relacyjna baza danych). W mniejszych zbiorach danych sortowaniem może zająć się widok.

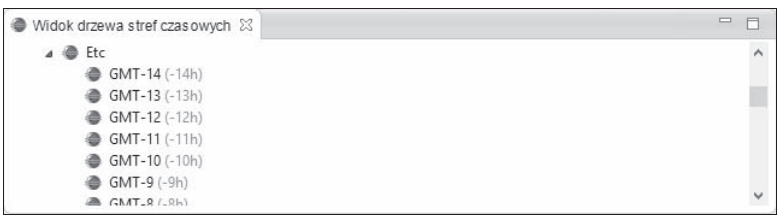
1. Widoki strukturyzowane JFace umożliwiają sortowanie przy użyciu klasy `ViewerComparator`. Utwórz nową klasę — `TimeZoneViewerComparator` — w pakiecie `com.packtpub.e4.clock.ui.internal` i zaimplementuj metodę `compare()`.

```
public class TimeZoneViewerComparator extends ViewerComparator {
   public int compare(Viewer viewer, Object o1, Object o2) {
      int compare;
      if (o1 instanceof TimeZone && o2 instanceof TimeZone) {
         long time= System.currentTimeMillis();
         compare=((TimeZone)o2).getOffset(time) - ((TimeZone)o1).getOffset(time);
      } else {
         compare = o1.toString().compareTo(o2.toString());
      }
      return compare;
   }
}
```

2. Podepnij nową klasę porównywania do widoku.

```
treeViewer.setComparator(new TimeZoneViewerComparator());
```

3. Uruchom testową wersję Eclipse i otwórz *Widok drzewa stref czasowych*. Strefy czasowe powinny być posortowane najpierw po przesunięciu czasu, a następnie alfabetycznie.



4. Aby dodać sortowanie specyficzne dla widoku, zmodyfikuj metodę `compare()` z `TimeZoneViewerComparator`, by otrzymać klucz `REVERSE` z danych widoku. Użyj go do odwrócenia sortowania wyników.

```
return compare;
boolean reverse =
Boolean.parseBoolean(String.valueOf(viewer.getData("REVERSE")));
return reverse ? -compare : compare;
```

5. Aby zobaczyć efekt działania nowego sortowania, ustaw klucz `REVERSE` tuż przed wywołaniem `setComparator()` na końcu metody `createPartControl()` z `TimeZoneTreeView`.

```
treeViewer.setData("REVERSE",Boolean.TRUE);
treeViewer.setComparator(new TimeZoneViewerComparator());
```

6. Ponownie uruchom testową wersję Eclipse, by przekonać się, że wyniki posortowane są odwrotnie niż poprzednio.

Co się stało?

Dodając obiekt `ViewerComparator` do obiektu `Viewer`, możemy określić sposób sortowania danych w konkretnym widoku. Oczywiście, najczęściej będzie to powiązane z wyborem odpowiedniej opcji w widoku — może to być opcja odwracająca sortowanie lub też zmieniająca sortowanie między nazwą i przesunięciem czasu.

Implementując obiekt komparatora, warto upewnić się, że metoda będzie obsługiwała różne typy obiektów (włączając te, których się nie oczekuje). Dane w widoku mogą się zmieniać lub być inne w trakcie działania aplikacji. Korzystaj z `instanceof`, by upewnić się, że typ jest właściwy.

Aby zapamiętać właściwości specyficzne dla widoku, użyj metod `setData()` i `getData()` z widoku. Dzięki temu można użyć ogólnego komparatora w wielu różnych widokach przy jednoczesnym respektowaniu ustawień filtracji i sortowania dla konkretnego widoku.

Przedstawiony przykład zawiera dane sortowania ustawione na stałe, co wymaga ponownego uruchomienia Eclipse, by zobaczyć efekt zmian. Po zmianie właściwości widoku, które wpływają na sortowanie lub filtrację, wywołaj metodę `refresh()` widoku, by zaktualizować wyświetlane dane zgodnie z nowymi ustawieniami.

Kroki do wykonania — filtrowanie elementów w widoku

Inną, często wykorzystywaną w widokach funkcją jest filtracja. Służy ona do ręcznego wyszukiwania konkretnego elementu lub też poszukiwania konkretnych elementów widoku. Bardzo często filtrację wiąże się z menu widoku, czyli menu rozwijanym po kliknięciu trójkąta w prawym górnym rogu widoku. Najczęściej stosuje się nazwę *Filters* (filtry). Klasa `ViewerFilter` zawiera metodę dotyczącą filtracji nazywaną `select()`. (Istnieją metody `filter()`, ale służą one do filtracji całej tablicy; metoda `select()` używana jest do określenia, czy należy wyświetlić konkretny element, czy też go pominąć).

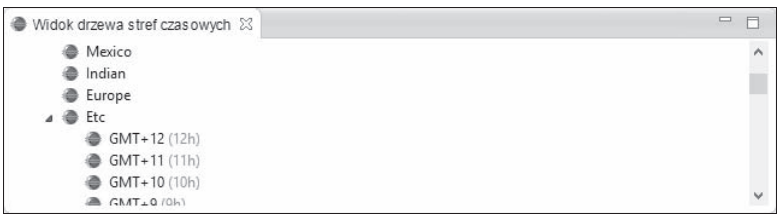
1. Utwórz klasę `TimeZoneViewerFilter` w pakiecie `com.packtpub.e4.clock.ui.internal`, która dziedziczy po klasie `ViewerFilter`. Konstruktor powinien przyjmować wzorzec typu `String`. Metoda `select()` musi zwracać `true`, jeśli element jest typu `TimeZone` i zawiera w swej nazwie wzorzec.

```
public class TimeZoneViewerFilter extends ViewerFilter {
   private String pattern;
   public TimeZoneViewerFilter(String pattern) {
      this.pattern = pattern;
   }
   public boolean select(Viewer v, Object parent, Object element) {
      if(element instanceof TimeZone) {
         TimeZone zone = (TimeZone)element;
         return zone.getDisplayName().contains(pattern);
      } else {
         return true;
      }
   }
}
```

2. Filtr ustawia się na poziomie widoku. Ponieważ widoki mogą mieć kilka filtrów, przekazuje się je do widoku jako tablicę. W tym przypadku wzorzec filtru ustawiamy w konstruktorze, ale w rzeczywistości zostałby pobrany od użytkownika. Zmodyfikuj klasę `TimeZoneTreeView` na końcu metody `createPartControl()`.

```
treeViewer.setFilters(new ViewerFilter[] {
   new TimeZoneViewerFilter("GMT")});
```

3. Uruchom testową wersję Eclipse i otwórz widok. Wyświetlane są tylko strefy czasowe z regionu *Etc*.



4. Aby usunąć ikony rozwijania węzłów przy pozostałych elementach, można włączyć w widoku drzewa automatyczne wykonywanie testów rozwinięć węzłów.

```
treeViewer.setExpandPreCheckFilters(true);
```

5. Ponownie uruchom testową wersję Eclipse i otwórz *Widok drzewa stref czasowych*. Zauważ, że puste grupy nadal są wyświetlane, ale nie ma już obok nich ikon rozwijania, bo po filtracji nie mają już elementów potomnych.

Co się stało?

Klasa `TimeZoneViewerFilter` powstała jako podklasa klasy `ViewerFilter` i jest przekazywana do klasy `TreeViewer`. W trakcie wyświetlania i filtracji danych filtr jest wywoływany dla każdego elementu drzewa (włącznie z korzeniem).

Domyślnie, gdy metoda `hasChildren()` zwróci wartość `true`, pojawi się ikona rozwinięcia gałęzi. Po kliknięciu algorytm przejdzie przez wszystkie potomki i wykona dla nich operację filtracji. Jeżeli okaże się, że po filtracji nie pozostał ani jeden element, algorytm usunie ikonę rozwinięcia.

Włączenie opcji `setExpandPreCheckFilters(true)` dla widoku spowoduje, że widok już na samym początku sprawdzi, czy po filtracji w gałęzi pozostanie choć jeden potomek. Opcja nie ma żadnych negatywnych konsekwencji, jeśli w ogóle nie ustawiono w niej filtrów. Jeśli filtry są ustawione, a danych w zbiorze jest dużo, wykonanie operacji sprawdzenia może zająć sporo czasu.

Aby domyślnie wyświetlić wszystkie elementy drzewa lub też zwinąć je do pojedynczego elementu, użyj metod `expandAll()` i `collapseAll()`. Najczęściej metody te wywołuje się z poziomu ikon typu *[+]* i *[-]* umieszczonych w lokalnym pasku narzędziowym widoku (patrz widoki *Synchronize* i *Package Explorer*).

Jeśli dane mają strukturę drzewiastą, która domyślnie powinna wyświetlić tylko część poziomów, warto zastosować metody `expandToLevel()` i `collapseToLevel()` przyjmujące wartość całkowitą i obiekt (użyj `getRoot()` dla korzenia, jeśli obiekt nie jest jawnie określony). Spowodują one rozwinięcie lub zwinięcie wszystkich elementów do zadanego poziomu. Metoda `expandAll()` to skrót wywołujący metodę `expandToLevel(getRoot(), ALL_LEVELS)`.

W odpowiedzi na zdarzenie wyboru, które zawiera ukryty obiekt, warto wykonać wcześniej operację `reveal()`, by konkretny element stał się widoczny. Pamiętaj, że `reveal()` działa tylko wtedy, gdy metoda `getParent()` jest poprawnie zaimplementowana, co w prezentowanym przykładzie nie ma miejsca.

Quiz — sortowanie i filtracja

P1. Jak posortować elementy drzewa w sposób inny niż domyślny?

P2. Jaka metoda służy do filtracji elementów?

P3. W jaki sposób połączyć kilka filtrów?

Sprawdź się — rozwijanie gałęzi i filtracja

Po poznaniu zasad dotyczących sortowania i filtracji rozbuduj przykład o kilka nowych elementów.

- Dodaj drugi filtr, który usuwa wszystkie strefy czasowe z ujemnym przesunięciem czasu.
- Po otwarciu widoku wykonaj operację `expandAll()`.

- Zastosuj sortowanie, w którym regiony są ułożone w odwrotnej kolejności alfabetycznej, ale strefy czasowe — w porządku alfabetycznym.
- Dodaj okno dialogowe pozwalające na zmianę wzorca filtru. Dodatkowo użyj pustego ciągu znaków jako wartości stosowanej do usunięcia filtracji.

Interakcje i właściwości

Możliwość wyświetlenia danych to jedna rzecz, ale w większości widoków najważniejsza jest interaktywność. Niezależnie od tego, czy dotyczy to funkcjonalności sortowania i filtracji z wcześniejszej części rozdziału, czy wyboru konkretnych elementów, widoki muszą być elementami interaktywnymi, by można ich użyć nie tylko do przeglądania danych, ale również do ich edycji.

Kroki do wykonania — dodanie procedury obsługi podwójnego kliknięcia

Widok drzewa najczęściej służy do wyświetlania treści w sposób hierarchiczny. Niestety, drzewo nie jest odpowiednią strukturą, by wyświetlić wszystkie szczegóły obiektu. Po podwójnym kliknięciu konkretnego elementu warto wyświetlić jego szczegóły.

1. Na końcu metody `createPartControl()` klasy `TimeZoneTreeView` zarejestruj anonimową klasę wewnętrzną implementującą interfejs `IDoubleClickListener` i dodaj ją metodą `addDoubleClickListener()` do obiektu `treeViewer`. Podobnie jak w rozdziale 1., otwórz okno dialogowe, by sprawdzić, czy wszystko zadziałało prawidłowo.

```
treeViewer.addDoubleClickListener(new IDoubleClickListener() {
   public void doubleClick(DoubleClickEvent event) {
      Viewer viewer = event.getViewer();
      Shell shell = viewer.getControl().getShell();
      MessageDialog.openInformation(shell, "Podwójne kliknięcie", "Wykryto
      ↪podwójne kliknięcie");
   }
});
```

2. Uruchom testową wersję Eclipse i otwórz widok. Podwójnie kliknij drzewo, a pojawi się komunikat *Wykryto podwójne kliknięcie*. Okno jest typu modalnego, co zapobiega wybraniu innych elementów interfejsu aż do momentu zamknięcia okna.
3. By znaleźć wybrane obiekty, Eclipse udostępnia interfejs `ISelection` (który zapewnia jedynie metodę `isEmpty()`) oraz interfejs `IStructuredSelection` (zapewnia iterator i inne metody dostępowe). Istnieje również kilka wyspecjalizowanych podtypów, na przykład `ITreeSelection`, który potrafi prześledzić ścieżkę prowadzącą do aktualnie wybranego elementu drzewa. W metodzie dotyczącej podwójnego kliknięcia znajdującej się w metodzie `createPartControl()` klasy `TimeZoneTreeView` zastąp fragment `MessageDialog` poniższym kodem.

```
MessageDialog.openInformation(shell, Podwójne kliknięcie", "Wykryto podwójne
➥kliknięcie");
ISelection sel = viewer.getSelection();
Object selectedValue;
if (!(sel instanceof IStructuredSelection) || sel.isEmpty()) {
   selectedValue = null;
} else {
   selectedValue = ((IStructuredSelection)sel).getFirstElement();
}
if (selectedValue instanceof TimeZone) {
   TimeZone timeZone = (TimeZone)selectedValue;
   MessageDialog.openInformation(shell, timeZone.getID(), timeZone.toString());
}
```

4. Uruchom Eclipse i otwórz widok. Dwukrotnie kliknij element drzewa, a pojawi się okno informacyjne z tekstem zawierającym nazwę strefy czasowej.
5. Aby wyświetlić więcej informacji na temat obiektu `TimeZone`, utwórz podklasę klasy `MessageDialog` o nazwie w `TimeZoneDialog` i umieść ją w pakiecie `com.packtpub.e4.clock.ui.internal`. Implementacja ma następującą postać.

```
public class TimeZoneDialog extends MessageDialog {
   private TimeZone timeZone;
   public TimeZoneDialog(Shell parentShell, TimeZone timeZone) {
      super(parentShell, timeZone.getID(), null, "Strefa czasowa " + timeZone.
      ➥getID(), INFORMATION, new String[] { IDialogConstants.OK_LABEL }, 0);
      this.timeZone = timeZone;
   }
}
```

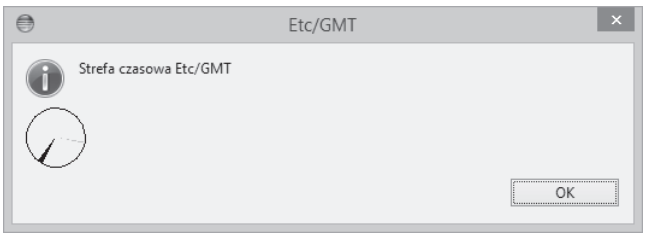
6. Treść okna zapewnia metoda `CustomArea()` używana do budowania zawartości widoku. Dodaj do klasy `TimeZoneDialog` metodę `createCustomArea()`.

```
protected Control createCustomArea(Composite parent) {
   ClockWidget clock = new ClockWidget(parent,SWT.NONE, new RGB(128,255,0));
   clock.setOffset((TimeZone.getDefault().getOffset(System.currentTimeMillis())
   ➥- timeZone.getOffset(System.currentTimeMillis()))/3600000);
   return parent;
}
```

7. Na końcu zmodyfikuj wywołanie `MessageDialog.open()` z klasy `TimeZoneTreeView`, by korzystało z nowej implementacji.

```
if (selectedValue instanceof TimeZone) {
   TimeZone timeZone = (TimeZone) selectedValue;
   MessageDialog.openInformation(shell, timeZone.getID(), timeZone.
   ➥toString());
   new TimeZoneDialog(shell, timeZone).open();
}
```

8. Uruchom testową wersję Eclipse i dwukrotnie kliknij strefę czasową, by zobaczyć okno dialogowe.



Co się stało?

Dodaliśmy procedurę obsługi podwójnego kliknięcia i zarejestrowaliśmy ją za pomocą metody `addDoubleClickListener()`. Początkowo wyświetlaliśmy standardowe okno informacyjne, ale później utworzyliśmy własną podklasę `MessageDialog`, która korzysta z klasy `ClockWidget`. By otrzymać odpowiednią strefę czasową (obiekt `TimeZone`), pobraliśmy aktualnie zaznaczony obiekt z `TreeViewer`.

Za zaznaczanie odpowiada interfejs `ISelection`. Metoda `getSelection()` powinna zawsze zwrócić wartość inną niż `null`, ale czasem uzyskana wartość spowoduje zwrócenie `true` po użyciu jej w metodzie `isEmpty()`. Istnieją jednak dwa interesujące interfejsy pochodne — `IStructuredSelection` i `ITreeSelection`.

Interfejs `ITreeSelection` jest podtypem `IStructuredSelection` i dodaje metody specyficzne dla drzew. Umożliwia otrzymanie informacji o aktualnie zaznaczonych elementach i ich elementach nadrzędnych (w strukturze drzewa).

Interfejs `IStructuredSelection` jest chyba najczęściej stosowanym interfejsem, gdy chodzi o systemy wyboru. Jeśli wybór nie jest pusty, praktycznie zawsze jest instancją implementującą `IStructuredSelection`. Z tego powodu bardzo często można zobaczyć poniższy fragment kodu.

```
ISelection sel = viewer.getSelection();
Object selectedValue;
if (!(sel instanceof IStructuredSelection) || sel.isEmpty()) {
   selectedValue = null;
} else {
   selectedValue = ((IStructuredSelection)sel).getFirstElement();
}
```

Fragment pobiera zaznaczenie z widoku. Jeśli wynik nie jest instancją `IStructuredSelection` lub jest pusty, przypisuje zmiennej `selectedValue` wartość `null`. W pozostałych sytuacjach rzutuje otrzymany obiekt na interfejs `IStructuredSelection` i wywołuje metodę `getFirstElement()`, by pobrać pojedynczą wartość zaznaczenia.

Zaznaczeniu mogło ulec więcej elementów, co oznacza, że metoda `getFirstElement()` zwraca jedynie pierwszy z nich. Klasa implementująca `IStructuredSelection` musi zapewnić iterator umożliwiający pobranie wszystkich zaznaczonych obiektów.

☞ E4: W Eclipse 4 zaznaczony obiekt można wstrzyknąć do metody za pomocą adnotacji.

```
@Inject @Optional
void setTZ(@Named(IServiceConstants.ACTIVE_SELECTION) TimeZone timeZone) {
}
```

Kroki do wykonania — wyświetlanie właściwości

IDE Eclipse, zamiast wymuszać tworzenie coraz to nowych okien dialogowych dla każdego obiektu, udostępnia ogólny widok właściwości (znajdujący się we wtyczce `org.eclipse.ui.views`), który służy do wyświetlania informacji o aktualnie zaznaczonym obiekcie. Właściwości są odkrywane w sposób uogólniony, a dostęp do nich zapewnia interfejs `IPropertySource`. Dzięki temu obiekt może wprowadzić abstrakcję w kwestii wyliczania wartości pól pokazywanych w oknie właściwości.

Najprostszym sposobem utworzenia źródła właściwości jest zapewnienie, by obiekt sam zaimplementował interfejs `IPropertySource`. Oczywiście, jest to możliwe tylko w sytuacji, gdy kod źródłowy można zmienić, ale w wielu sytuacjach (na przykład w przypadku obiektu `TimeZone` lub `Map.Entry` zawierającego klucz typu `String` i obiekt `TimeZone`) kod źródłowy nie jest dostępny.

1. Otwórz plik *MANIFEST/META-INF.MF* i dodaj `org.eclipse.ui.views` jako zależność w zakładce *Dependencies* lub jako paczkę w *Require-Bundle*. W przeciwnym razie `IPropertySource` nie będzie odnajdywane.
2. Utwórz w pakiecie `com.packtpub.e4.clock.ui.internal` klasę `TimeZonePropertySource` implementującą interfejs `IPropertySource`. W konstruktorze przyjmij pojedynczą instancję `TimeZone`.

```
public class TimeZonePropertySource implements IPropertySource {
   private TimeZone timeZone;
   public TimeZonePropertySource(TimeZone timeZone) {
      this.timeZone = timeZone;
   }
}
```

3. Jedynymi metodami, które trzeba zaimplementować, są `getPropertyValue()` i `getPropertyDescriptors()`. (Pozostałe metody, takie jak `getEditableValue()` i `isPropertySet()`, można zignorować, bo używa się ich tylko w operacjach edycji. Powinny pozostać puste lub zwracać `null` albo `false`. Metody `getPropertyValue()` i `isPropertySet()` wywołuje się z identyfikatorem. Pozostałe metody zwracają tablice obiektów `PropertyDescriptors` łączące identyfikator i nazwę właściwości do wyświetlenia w interfejsie graficznym. Dodaj poniższy kod do klasy `TimeZonePropertySource`.

```
private static final Object ID = new Object();
private static final Object DAYLIGHT = new Object();
private static final Object NAME = new Object();
public IPropertyDescriptor[] getPropertyDescriptors() {
   return new IPropertyDescriptor[] {
      new PropertyDescriptor(ID, "Strefa czasowa"),
      new PropertyDescriptor(DAYLIGHT, "Czas letni"),
      new PropertyDescriptor(NAME, "Nazwa")
   };
}
public Object getPropertyValue(Object id) {
   if (ID.equals(id)) {
      return timeZone.getID();
   } else if(DAYLIGHT.equals(id)) {
      return timeZone.inDaylightTime(new Date());
   } else if (NAME.equals(id)) {
      return timeZone.getDisplayName();
   } else {
      return null;
   }
}
```

4. Powiązanie źródła właściwości z oknem właściwości wymaga użycia adaptera. Można go wskazać za pomocą interfejsu `IAdaptable`, który umożliwia klasie wirtualną implementację interfejsu. Ponieważ `TimeZone` nie może zaimplementować `IAdaptable` w sposób bezpośredni, potrzebujemy `IAdapterFactory`.

5. Utwórz w pakiecie `com.packtpub.e4.clock.ui.internal` klasę `TimeZoneAdapterFactory` implementującą interfejs `IAdapterFactory`.

```
public class TimeZoneAdapterFactory implements IAdapterFactory {
   public Class[] getAdapterList() {
      return new Class[] { IPropertySource.class };
   }
   public Object getAdapter(Object o, Class type) {
      if(type == IPropertySource.class && o instanceof TimeZone) {
         return new TimeZonePropertySource((TimeZone)o);
      } else {
         return null;
      }
   }
}
```

6. Aby zarejestrować fabrykę adapterów w Eclipse, dodaj odpowiedni wpis w pliku *plugin.xml*.

```
<extension point="org.eclipse.core.runtime.adapters">
   <factory adaptableType="java.util.TimeZone"
      class="com.packtpub.e4.clock.ui.internal.TimeZoneAdapterFactory">
      <adapter type="org.eclipse.ui.views.properties.IPropertySource"/>
   </factory>
</extension>
```

7. Uruchom testową wersję Eclipse, wybierz strefę czasową w widoku drzewa i otwórz okno właściwości poleceniem *Window/Show View/Other/General/Properties* z menu. Nie pojawią się żadne informacje. By mieć pewność, że adapter jest podpięty prawidłowo, dodaj na końcu metody `createPartControl()` z `TimeZoneTreeView` następujący wpis.

```
System.out.println("Adapterem jest " + Platform.getAdapterManager().
➥getAdapter(TimeZone.getDefault(),IPropertySource.class));
```

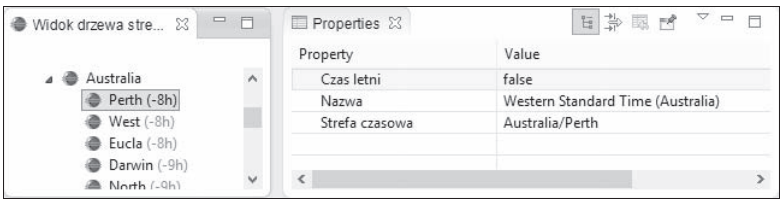
8. Uruchom testową wersję Eclipse, otwórz *Widok drzewa stref czasowych* i sprawdź widok *Console* głównego Eclipse. Konsola powinna zawierać wpis podobny do poniższego.

```
Adapterem jest com.packtpub.e4.clock.ui.internal.TimeZonePropertySource
➥@7f8a6fb0
```

9. Czego więc brakuje? Okazuje się, że okno *Properties* nie otrzymuje informacji o zmianie zaznaczenia. By rozwiązać problem, dodaj w metodzie `createPartControl()` z `TimeZoneTreeView` następujący wpis.

```
~~System.out.println("Adapterem jest " + Platform.getAdapterManager().~~
~~➥getAdapter(TimeZone.getDefault(),IPropertySource.class));~~
getSite().setSelectionProvider(treeViewer);
```

10. Teraz zmiany zaznaczenia będą przekazywane do IDE, by inne widoki mogły zaktualizować swoje dane. Po wybraniu strefy czasowej okno *Properties* będzie aktualizowało się automatycznie. Uruchom testową wersję Eclipse, otwórz *Widok drzewa stref czasowych*, wybierz strefę czasową i otwórz widok *Properties*.



☞ E4: W celu powiązania widoku z dostawcą zaznaczania należy użyć kodu podobnego do poniższego.

```
@Inject
ESelectionService selectionService;
ISelectionChangedListener selectionListener;
@PostConstruct
public void postConstruct() {
   selectionListener = new ISelectionChangedListener() {
      public void selectionChanged(SelectionChangedEvent e) {
         if (selectionService != null)
            selectionService.setSelection(e.getSelection());
      }
   };
   treeViewer.addSelectionChangedListener(selectionListener);
```

```
@PreDestroy
public void preDestroy() {
   if(selectionListener != null)
      treeViewer.removeSelectionChangedListener(selectionListener);
   selectionListener = null;
}
```

Co się stało?

Aby zaktualizować stan zaznaczenia IDE, musieliśmy powiązać dostawcę zaznaczenia widoku z tym, który dotyczy IDE (metoda `getSite()`). Gdy zmieni się zaznaczenie elementu w widoku, widok wyśle komunikat do wszystkich nasłuchujących obiektów, by te mogły odpowiednio zaktualizować swoje dane.

> ☞ E4: Procedurę obsługi zaznaczenia trzeba zarejestrować (i wyrejestrować) ręcznie, by zapewnić właściwe powiązanie między widokiem i usługą zaznaczenia. Zamiast `ISelectionService` używa się `ESelectionService`. Interfejs jest nieco inny, ponieważ `ISelectionService` jest powiązany z klasą `IWorkbenchPart`, a `ESelectionService` nie posiada podobnego powiązania.

W celu zapewnienia informacji dla widoku *Properties* utworzyliśmy dla `TimeZone` klasę bazującą na interfejsie `IPropertySource` i powiązaliśmy ją z `IAdapterManager` obiektu `Platform` poprzez deklarację w pliku *plugin.xml*.

Powiązania znacznie wygodniej tworzyć w sposób deklaratywny w pliku *plugin.xml*, bo nie trzeba stosować metod aktywacyjnych `start()` i `stop()`. Wynika to z faktu, iż metoda startowa z `Activator` nie może zostać wywołana aż do momentu wczytania pierwszej klasy z paczki; w przypadku adaptera rejestracja deklaratywna zapewnia odpowiednią informację niezależnie od kolejności wczytywania.

Fabryka adapterów zapewnia metodę `getAdapter()`, która odpowiada za otoczenie lub konwersję przekazanego obiektu na obiekt pożądanego typu. Jeśli obiekt jest już instancją docelowego typu, zostanie po prostu zwrócony — w przeciwnym razie metoda zwraca POJO, pośrednika lub otoczkę implementującą pożądany interfejs. Często zdarza się, że posiadamy klasę (na przykład `TimeZonePropertySupport`), której jedynym zadaniem jest implementacja pożądanego interfejsu. Klasa tego typu stanowi otoczkę dla obiektu (`TimeZone`) w celu zapewnienia wymaganej funkcjonalności.

Interfejs `IPropertySupport` zapewnia podstawowe metody do pobierania właściwości obiektu. Do identyfikacji właściwości używa identyfikatorów. Identyfikator może być obiektem dowolnego typu. W prezentowanym przykładzie były to instancje `new Object`. Choć można użyć obiektów typu `String` (co można zobaczyć w wielu przykładach), nie jest to podejście zalecane, ponieważ wartość obiektu `String` nie ma znaczenia, ale zajmuje miejsce w przestrzeni `PermGen` pamięci maszyny wirtualnej. Co więcej, obiekt `Object` umożliwia porównywanie instancji za

pomocą operacji == bez narażania się na ostrzeżenia automatycznych testów stylu lub pytania przy ocenie kodu. (Inne przykłady stosują metodę `equals()`, by zachęcić do jej użycia, gdy nie są stosowane obiekty `Object`, ale dobry JIT i tak wykona optymalizację, szczególnie wtedy, gdy kod wysyła wiadomość do instancji typu `static final`).

Quiz — działanie właściwości

P1. Jak instancje `TableViewer` mogą reagować na kliknięcie?

P2. Dlaczego tworzy się podklasy klasy `Dialog`?

P3. Czym są deskryptory właściwości?

P4. Jak wyświetlić właściwości w widoku *Properties*?

Dane tabelaryczne

Widok drzewa pojawia się w Eclipse bardzo często, ale czasem trzeba wyświetlić dodatkowe informacje związane z pojedynczym elementem. JFace zapewnia klasę `TableViewer` podobną do `TreeViewer`, ale zamiast pojedynczych etykiet można wyświetlać wiele kolumn z danymi. Istnieje również klasa `TableTreeViewer`, która łączy funkcjonalność obu klas.

Kroki do wykonania — przeglądanie stref czasowych w tabeli

Aby wyświetlać strefy czasowe w postaci tabelarycznej, utworzymy nowy widok o nazwie *Widok tabeli stref czasowych*.

1. Kliknij prawym przyciskiem myszy projekt *com.packtpub.e4.clock.ui* i wybierz polecenie *Plug-in Tools/Open Manifest*. Otwórz zakładkę *Extensions*, kliknij prawym przyciskiem myszy *org.eclipse.ui.views* i wybierz *New/View*. Wypełnij pola w następujący sposób.
   - W polu *ID* wpisz **`com.packtpub.e4.clock.ui.views.TimeZoneTableView`**.
   - W polu *Name* wpisz **`Widok tabeli stref czasowych`**.
   - W polu *Class* wpisz **`com.packtpub.e4.clock.ui.views.TimeZoneTableView`**.
   - W polu *Category* wpisz **`com.packtpub.e4.clock.ui`**.
   - W polu *Icon* wpisz **`icons/sample.gif`**.
2. Plik *plugin.xml* powinien po tej operacji zawierać następujący fragment.

```
<view
   category="com.packtpub.e4.clock.ui"
   class="com.packtpub.e4.clock.ui.views.TimeZoneTableView"
```

```
      icon="icons/sample.gif"
      id="com.packtpub.e4.clock.ui.views.TimeZoneTableView"
      name="Widok tabeli stref czasowych"
      restorable="true">
   </view>
```

3. Utwórz nową klasę `TimeZoneTableView` rozszerzającą `ViewPart` na podstawie skrótu z edytora lub skorzystaj z nowego kreatora klas. Po utworzeniu widoku dodaj pusty obiekt `TableViewer` i użyj klasy `ArrayContentProvider` z listą dostępnych stref czasowych.

```
public class TimeZoneTableView extends ViewPart {
   private TableViewer tableViewer;
   public void createPartControl(Composite parent) {
      tableViewer=new TableViewer(parent,SWT.H_SCROLL|SWT.V_SCROLL);
      tableViewer.getTable().setHeaderVisible(true);
      tableViewer.setContentProvider(ArrayContentProvider.getInstance());
      tableViewer.setInput(TimeZone.getAvailableIDs());
   }
   public void setFocus() {
      tableViewer.getControl().setFocus();
   }
}
```

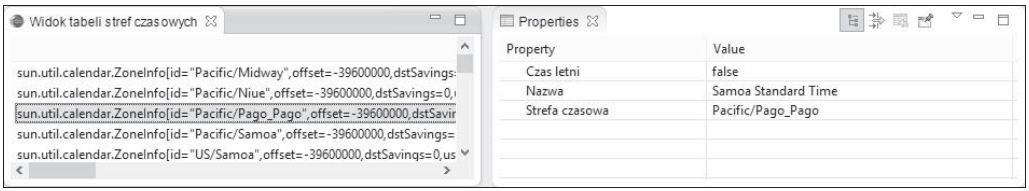
☞ E4: Tworząc część aplikacji dla Eclipse 4, trzeba pamiętać o dodaniu adnotacji `@Inject` dla konstruktora i adnotacji `@Focus` dla metody `setFocus()`.

4. Uruchom testową wersję Eclipse, a w widoku o nazwie *Widok listy stref czasowych* pojawi się jednowymiarowa lista wszystkich stref czasowych.

5. Skonwertuj tablicę obiektów `String` na tablicę obiektów `TimeZone` i ustaw ją jako dane wejściowe.

```
tableViewer.setInput(TimeZone.getAvailableIDs());
String[] ids = TimeZone.getAvailableIDs();
TimeZone[] timeZones = new TimeZone[ids.length];
for(int i=0;i<ids.length;i++) {
   timeZones[i] = TimeZone.getTimeZone(ids[i]);
}
tableViewer.setInput(timeZones);
getSite().setSelectionProvider(tableViewer);
```

6. Dostawca zaznaczenia został wskazany w ten sam sposób jak w przykładzie z klasą `TimeZoneTreeView`. Zaznacz element tabeli i sprawdź zawartość widoku *Properties*.



7. Tabela zawiera listę obiektów `ZoneInfo`. Wynika to z faktu, iż brakuje obiektu `LabelProvider`, więc do wyświetlania elementów widok używa wartości zwróconej przez `toString()`. Ponieważ tabela ma wiele kolumn, obiekt `TableViewer` wykorzystuje wiele instancji `TableViewerColumn`. Każda z nich reprezentuje kolumnę tabeli i posiada własny rozmiar, tytuł i dostawcę opisów. Tworzenie nowej kolumny oznacza najczęściej ustalenie standardowego wyglądu (na przykład szerokości) i wskazanie danych do wyświetlenia.

   By ułatwić wielokrotne użycie kodu, utwórz abstrakcyjną podklasę `ColumnLabelProvider` o nazwie `TimeZoneColumn` (w pakiecie `com.packtpub.e4.clock.ui.internal`) z abstrakcyjnymi metodami `getText()` i `getTitle()` oraz konkretną metodą `getWidth()`.

```
public abstract class TimeZoneColumn extends ColumnLabelProvider {
   public abstract String getText(Object element);
   public abstract String getTitle();
   public int getWidth() {
      return 250;
   }
}
```

8. Dodaj do klasy `TimeZoneColumn` metodę pomocniczą, która ułatwi dołączanie klasy do widoku.

```
public TableViewerColumn addColumnTo(TableViewer viewer) {
   TableViewerColumn tableViewerColumn = new TableViewerColumn(viewer,SWT.NONE);
   TableColumn column = tableViewerColumn.getColumn();
   column.setMoveable(true);
   column.setResizable(true);
   column.setText(getTitle());
   column.setWidth(getWidth());
   tableViewerColumn.setLabelProvider(this);
   return tableViewerColumn;
}
```

9. Utwórz w tym samym pakiecie podklasę `TimeZoneIDColumn` rozszerzającą `TimeZoneColumn` i zwracającą kolumnę identyfikatora.

```
public class TimeZoneIDColumn extends TimeZoneColumn {
   public String getText(Object element) {
      if (element instanceof TimeZone) {
```

```
            return ((TimeZone) element).getID();
         } else {
            return "";
         }
      }
      public String getTitle() {
         return "ID";
      }
   }
```

10. Zmodyfikuj klasę `TimeZoneTableView` — na końcu metody `createPartControl()` utwórz obiekt kolumny i wywołaj metodę `addColumnTo()` przed użyciem metody `setInput()`.

```
new TimeZoneIDColumn().addColumnTo(tableViewer);
tableViewer.setInput(timeZones);
```

> Pamiętaj, że kolumny trzeba utworzyć przed wywołaniem metody `setInput()`. W przeciwnym razie nie wyświetlą się poprawnie.

11. Uruchom testową wersję Eclipse i otwórz *Widok tabeli stref czasowych*. Kolumna *ID* powinna być jedyną wyświetlaną kolumną.

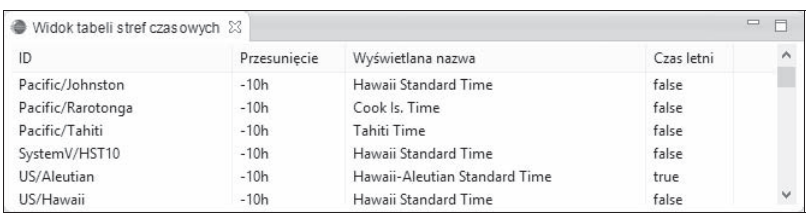
12. Aby dodać kolejne kolumny, skopiuj klasę `TimeZoneIDColumn`, a następnie zmień tytuł i zwracaną właściwość obiektu `TimeZone`. Przykładowo utwórz kopię `TimeZoneIDColumn` o nazwie `TimeZoneDisplayNameColumn`. Zmodyfikuj tytuł i pobieraną metodę `get`.

```
~~return ((TimeZone) element).getID();~~
return ((TimeZone) element).getDisplayName();
~~return "ID";~~
return "Wyświetlana nazwa";
```

13. Opcjonalnie wykonaj te same zadania dla innych właściwości `TimeZone`, na przykład przesunięcia czasu (metoda `getOffset()`) lub użycia czasu letniego (`useDaylightTime()`). Kolumny można następnie dodać do tabeli.

```
new TimeZoneOffsetColumn().addColumnTo(tableViewer);
new TimeZoneDisplayNameColumn().addColumnTo(tableViewer);
new TimeZoneSummerTimeColumn().addColumnTo(tableViewer);
```

14. Uruchom instancję Eclipse i przejdź do widoku, by zobaczyć dodatkowe kolumny.



ID	Przesunięcie	Wyświetlana nazwa	Czas letni
Pacific/Johnston	-10h	Hawaii Standard Time	false
Pacific/Rarotonga	-10h	Cook Is. Time	false
Pacific/Tahiti	-10h	Tahiti Time	false
SystemV/HST10	-10h	Hawaii Standard Time	false
US/Aleutian	-10h	Hawaii-Aleutian Standard Time	true
US/Hawaii	-10h	Hawaii Standard Time	false

Co się stało?

Utworzyliśmy obiekt `TableViewer`, a następnie dodaliśmy do niego wiele obiektów `ColumnLabelProvider`, by wyświetlić poszczególne kolumny. Tworzenie podklas `ColumnLabelProvider` zapobiega uciekaniu się do anonimowych klas wewnętrznych i ułatwia wykonanie metody pomocniczej. Podklasy ułatwiają tworzenie i podpinanie kolumn (o określonym tytule i szerokości) przy jednoczesnym pamiętaniu konkretnych podklas, takich jak `TimeZoneIDColumn`. W ten sposób unika się śledzenia kolumn za pomocą identyfikatorów.

By dostosować kolumny do własnych potrzeb, używamy klasy `Column` z SWT, włączając takie właściwości jak przesuwanie kolumn (`setMovable(true)`) lub zmianę ich rozmiaru (`setResizable(true)`). Dopuszczalne są również operacje związane z tabelą (klasa `Table` z SWT), takie jak wyświetlenie nagłówka (`setHeaderVisible(true)`).

Warto pamiętać, że kolumny widoku tabeli są obliczane w momencie wywołania metody `setInput()`, więc kolumny dodane po tym wywołaniu mogą nie wyświetlać się prawidłowo. Najlepiej wywołać metodę `setInput()` po zakończeniu innych prac związanych z tabelą.

Nic nie stoi na przeszkodzie, by przenieść do widoku funkcjonalności zaimplementowane w widoku drzewa. Przykładowo podpięcie logiki wyboru umożliwia wyświetlanie w widoku *Properties* właściwości wybranej strefy czasowej.

Kroki do wykonania — synchronizacja wyboru

Widoki `TimeZoneTableView` i `TimeZoneTreeView` mogą przekazywać wybór widokowi *Properties*. Reakcja na wybór zapewnia poczucie jednolitości, choć widoki są niezależnymi bytami.

Możliwe jest dodatkowe powiązanie widoków, by strefa czasowa (`TimeZone`) wybrana w jednym z nich automatycznie została podświetlona w drugim. W tym celu trzeba dodać nasłuchiwanie zdarzenia wyboru i jeśli wybrany zostanie obiekt `TimeZone`, wyświetlić go w widoku (przy użyciu metod `reveal()` i `setSelection()`).

1. Utwórz w pakiecie `com.packtpub.e4.clock.ui.internal` klasę `TimeZoneSelectionListener` implementującą interfejs `ISelectionListener`. Konstruktor przyjmie widok i obiekt części IDE. Trzeba też dodać metodę `selectionChanged()`.

```
public class TimeZoneSelectionListener implements ISelectionListener {
   private Viewer viewer;
   private IWorkbenchPart part;
   public TimeZoneSelectionListener(Viewer v, IWorkbenchPart p) {
      this.viewer = v;
      this.part = p;
   }
   public void selectionChanged(IWorkbenchPart p, ISelection sel) {
   }
}
```

2. Metoda `selectionChanged()` wykonuje kilka zadań. Oto one.
   - Ignorowanie zdarzenia, jeśli zostało wysłane przez tę samą część IDE.
   - Pobranie zaznaczonego obiektu ze zdarzenia i porównanie go z aktualnym zaznaczeniem.
   - Uaktualnienie widoku, jeśli obiekty są różne i zaznaczonym obiektem jest `TimeZone`.
3. Implementacja ma następującą postać.

```
public void selectionChanged(IWorkbenchPart p, ISelection sel) {
   if (p != this.part) {
      IStructuredSelection selected = ((IStructuredSelection)sel).
      ↪getFirstElement();
      Object current = ((IStructuredSelection)viewer.getSelection()).
      ↪getFirstElement();
      if(selected != current && selected instanceof TimeZone) {
         viewer.setSelection(sel);
         if(viewer instanceof StructuredViewer) {
            ((StructuredViewer) viewer).reveal(selected);
         }
      }
   }
}
```

4. Obiekt nasłuchiwania wyboru trzeba zarejestrować w widokach. Otwórz klasę `TimeZoneTableView` i na dole metody `createPartControl()` dodaj następujący kod.

```
selectionListener = new TimeZoneSelectionListener(tableViewer,
↪getSite().getPart());
getSite().getWorkbenchWindow().getSelectionService().addSelectionListener
↪(selectionListener);
```

5. Obiekt `selectionListener` trzeba dodać jako pole, ponieważ niezbędne jest jego usunięcie z listy procedur nasłuchiwania w momencie usuwania widoku.

```
private TimeZoneSelectionListener selectionListener;
public void dispose() {
   if (selectionListener != null) {
      getSite().getWorkbenchWindow().getSelectionService()
         .removeSelectionListener(selectionListener);
      selectionListener = null;
   }
   super.dispose();
}
```

6. Bardzo podobną zmianę (inna jest tylko nazwa zmiennej widoku) wykonaj w klasie `TimeZoneTreeView`.

```
selectionListener = new TimeZoneSelectionListener(treeViewer,
↪getSite().getPart());
getSite().getWorkbenchWindow().getSelectionService().addSelectionListener
↪(selectionListener);
```

7. Metoda `dispose()` w klasie `TimeZoneTreeView` powinna być taka sama jak w klasie `TimeZoneTableView`.
8. Uruchom instancję Eclipse, wybierz strefę czasową w widoku o nazwie *Widok tabeli stref czasowych*, a w widoku nazywanym *Widok drzewa stref czasowych* pojawi się ten sam wybór. Tym razem zmień zaznaczenie w widoku o nazwie *Widok drzewa stref czasowych*, by zobaczyć, czy w widoku nazwanym *Widok tabeli stref czasowych* pojawi się ten sam wpis.

Co się stało?

Zdarzenia wyboru są w Eclipse zgłaszane bardzo często, więc warto zatroszczyć się, by kod nasłuchiwania wyboru działał wydajnie. Filtrując zdarzenia pochodzące z tej samej części lub nieistotne typy, uzyskujemy większą wydajność. W przedstawionym kodzie sprawdzamy, czy zaznaczenie składa się z przynajmniej jednego elementu typu `TimeZone`, zanim poprosimy o aktualizację UI.

Wybór widoku można zsynchronizować z wywołaniem `setSelection()`. W ten sposób oszczędzamy na nowym obiekcie zaznaczenia i ustawiamy dane we właściwy sposób. Samo ustawienie wyboru nie wystarcza — wywołanie metody `reveal()` jest niezbędne do właściwego podświetlenia zaznaczonego elementu. W sytuacji, gdy zaznaczono wiele elementów, podświetli tylko pierwszy z nich.

Metoda `reveal()` dostępna jest jedynie dla `StructuredViewers`, więc trzeba rzutować obiekt wyboru na `IStructuredSelection` w przypadku obiektów typu `StructuredViewers`.

Na końcu rejestrujemy procedury obsługi zdarzeń w momencie tworzenia widoku i usuwamy je w momencie niszczenia widoku. W tym celu pobieramy obiekt typu `ISelectionService` z części IDE i wywołujemy metodę `addSelectionListener()`, by dodać procedurę obsługi, lub metodę `removeSelectionListener()`, by ją usunąć.

> ☞ E4: W Eclipse 4 zamiast `ISelectionService` stosuje się `ESelectionService`. To podobny, ale nie identyczny interfejs, ponieważ nie zapewnia przekazania obiektu `WorkbenchPart`. Najczęściej `ESelectionService` wstrzykuje się do widoku, a procedura obsługi jest dodawana w `@PostConstruct` i usuwana w `@PreDestroy`.

Quiz — działanie tabel

P1. W jaki sposób włączyć obsługę kolumn w `TableViewer`?

P2. Do czego służy `TableViewerColumn`?

P3. W jaki sposób synchronizować wybór elementu między dwoma widokami?

Podsumowanie

W tym rozdziale opisaliśmy użycie JFace do tworzenia widoków dla ustrukturyzowanych danych. Przedstawiliśmy zarówno widoki bazujące na drzewach (klasa `TreeViewer`), jak i widoki bazujące na tabelach (klasa `TableViewer`). Przedstawiliśmy również kilka wbudowanych w JFace funkcjonalności związanych z czcionkami i obrazami.

By synchronizować dane między widokami Eclipse, użyliśmy obiektów `ISelectionService` (w Eclipse 4 używa się obiektów `ESelectionService`). Umożliwienie widokom zgłaszania i odbierania zdarzeń wyboru zapewnia wizualną spójność IDE nawet wtedy, kiedy widoki znajdują się w różnych wtyczkach.

Skorowidz

G

I

J

K

M

N

O

P

R

S

Ś

T

U

V

W

Z

Eclipse 4
Programowanie wtyczek na przykładach

Eclipse to przede wszystkim darmowe i popularne środowisko programistyczne, używane głównie przez programistów języka Java. Dzięki elastyczności oraz możliwości tworzenia wtyczek Eclipse przydaje się także programistom wielu innych języków, między innymi C, C++, PHP. Platformę tę można wykorzystać również do tworzenia aplikacji. Dzięki tej książce przekonasz się, że to wcale nie musi być trudne!

Już w trakcie lektury początkowych rozdziałów wykonasz swoją pierwszą wtyczkę. Nauczysz się tworzyć widoki w SWT oraz JFace, pobierać dane od użytkownika oraz korzystać z zasobów. Ponadto dowiesz się, jak grupować wtyczki oraz je aktualizować. Z pewnością Twoją uwagę zwróci rozdział poświęcony automatycznym testom tworzonych rozszerzeń. Dzięki nim będziesz zawsze pewien, że rozszerzenia działają dokładnie tak, jak zaplanowałeś! Książka ta jest doskonałą lekturą dla wszystkich programistów chcących wykorzystać potencjał platformy Eclipse!

Dzięki tej książce:

- poznasz platformę Eclipse
- zrozumiesz model Eclipse w wersji 4
- pozwolisz użytkownikom dostosować aplikację do ich potrzeb
- błyskawicznie zbudujesz aplikację z użyciem Eclipse

Odkryj nieznane możliwości środowiska Eclipse!

[PACKT] open source PUBLISHING

helion.pl księgarnia internetowa

Nr katalogowy: 20681

Księgarnia internetowa: http://helion.pl

Zamówienia telefoniczne: 0 801 339900 0 601 339900

Helion

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

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

Informatyka w najlepszym wydaniu

sięgnij po WIĘCEJ KOD KORZYŚCI

ISBN 978-83-246-8754-1

Cena: 59,00 zł