Zajęcia 3. Klasy pochodne `Informatyk`, `Lekarz`, `Nauczyciel`. Rozbudowa aplikacji o operacje na plikach

3.1. Wstęp

Zaprojektowana do tej pory kolekcja typu `List<Pracownik>` służy do dokonywania różnych operacji na liście pracowników. Przechowywane obiekty typu `Pracownik` są jednak elementami ogólnymi (w pewnym sensie abstrakcyjnymi) z niewyszczególnionymi cechami typowymi dla konkretnych pracowników. Opisana w niniejszym rozdziale rozbudowa projektu typu **Class Library** umożliwi zdefiniowanie bardziej precyzyjnych obiektów. Będą one reprezentowane przez typy: `Informatyk`, `Lekarz` oraz `Nauczyciel`, pochodne po klasie `Pracownik`, które będzie można zarówno dodawać, usuwać, sortować, wyszukiwać na liście jak i dokonywać przy ich użyciu operacji wejścia-wyjścia. Aby udostępnić taką funkcjonalność aplikacji, konieczne będzie poszerzenie projektu o wyżej wymienione klasy. Muszą one zostać umieszczone w projekcie **BibliotekaKlas**.

Jak dotąd aplikacja umożliwia przeprowadzenie operacji wejścia-wyjścia na liście pracowników poprzez wczytanie i wypisanie ich danych w oknie konsoli (odpowiedzialne są za to statyczne metody `WriteLine()` i `ReadLine()` klasy `Console`). Teraz program zostanie wzbogacony o możliwość odczytu i zapisu danych pracowników do pliku. W tym celu w projekcie **BibliotekaKlas** projektu zostanie zdefiniowana kolejna klasa o nazwie `FormatDanych`. Operacje na pliku będą przeprowadzane w formacie Xml przy użyciu metod i właściwości klasy `DataSet`, głównego komponentu biblioteki **ADO.NET**.

Rozszerzenie aplikacji o nowe typy, udostępnienie możliwości odczytu/zapisu w formacie Xml pociągnie za sobą rozbudowę klasy `Lista` poprzez dodanie i modyfikację odpowiednich metod.

3.2. Dodanie klasy `Informatyk` do biblioteki klas

Klasa o nazwie `Informatyk` ma być klasą pochodną po klasie `Pracownik`. Dla tej klasy należy zdefiniować następujące składowe:

- Prywatne pola:
  - `string adresEmail;`
  - `string stronaInternetowa;`
- Publiczne właściwości:
  - `AdresEmail` zwracającą i ustawiającą wartość pola `adresEmail`.
  - `StronaInternetowa` zwracającą i ustawiającą wartość pola `stronaInternetowa`.
  - `Zawod` zwracającą składową `Informatyk` wyliczenia `Zawody`. Właściwość ma nadpisywać tę samą właściwość z klasy podstawowej (słowo kluczowe `override`).
- Publiczne metody:
  - bezparametrowy konstruktor inicjalizujący pola składowe zerami. W konstruktorze, na liście inicjalizacyjnej należy wywołać bezargumentowy konstruktor klasy bazowej.
  - konstruktor inicjalizujący wszystkie pola składowe na podstawie argumentów. Konstruktor ma posiadać te same argumenty co konstruktor klasy `Pracownik` i dodatkowe parametry potrzebne do zainicjalizowania pól składowych obiektu typu `Informatyk`).
  - konstruktor kopiujący inicjalizujący wszystkie pola składowe na podstawie argumentu wzorcowego typu `Informatyk`.
    Uwaga: W każdym konstruktorze, na liście inicjalizacyjnej, należy wywołać odpowiedni konstruktor klasy bazowej w celu ustawienia niedostępnych pól klasy bazowej.
  - `override Pracownik Clone()`, zwracającą instancję nowostworzonego obiektu typu `Informatyk` na wzór danego (wywołanie konstruktora kopiującego).
  - `override string SzczegolyZawodu()` zwracającą łańcuch w formie: adres email _s_ strona internetowa, gdzie _s_ jest znakiem separatora `'\t'`.
  - `override string ToString()` zwracającą łańcuch opisujący dane pracownika w formie: imię nazwisko dzień miesiąc rok ulica numer domu miasto adres email strona internetowa.

- `override string FormatWyjsciowy()` zwracającą łańcuch opisujący dane pracownika w formie:
  Zawód: Informatyk
  Imię i nazwisko: imię nazwisko
  Data urodzenia: dzień miesiąc rok
  Adres zamieszkania: ulica numer domu miasto
  Dane dodatkowe: adres email strona internetowa
  Definicja tej metody wymaga odpowiedniego sformatowania informacji oraz, ze względu na odniesienie się do danych z klasy bazowej, wywołania metody `FormatWyjsciowy` z klasy `Pracownik`. Implementacja zaprezentowana jest na Listingu 3.1.

```
public override string FotrmatWyjsciowy()
{
   return String.Format("Zawód: {0}\n{1}Dane dodatkowe: {2}, {3}",
          this.Zawod, base.FotrmatWyjsciowy(), adresEmail, stronaInternetowa);
}
```

Listing 3.1: Definicja metody formatującej dane informatyka do wymaganej postaci.

- `override void OdczytConsole()`, której zadaniem jest wczytanie z klawiatury wszystkich danych dla informatyka. Obiekt typu `Informatyk` przechowuje również dane zawarte w typie `Pracownik` (odnoszące się do imienia, nazwiska, daty urodzenia i adresu zamieszkania) w związku z tym w definicji metody konieczne będzie wywołanie metody `OdczytConsole` z klasy bazowej. Metoda `OdczytConsole` zdefiniowana dla klasy `Informatyk` przesłania tę samą metodę w klasie `Pracownik` co umożliwia zastosowanie polimorfizmu.
- `override void ZapisConsole()`, której zadaniem jest wypisanie na ekran wszystkich danych informatyka. W przypadku definicji tej metody ma miejsce taka sama sytuacja jak podczas odczytu danych z konsoli: w trakcie wypisania na ekran konieczne jest uwzględnienie informacji o danych podstawowych informatyka (wywołanie metody `OdczytConsole` z klasy `Pracownik`.)
- `override void OdczytXml(DataRow dr)` – ciało metody należy na chwilę obecną zostawić puste.
  Uwaga: Ze względu na to, iż argumentem metody `OdczytXml` jest obiekt typu `DataRow`, w pliku definicji klasy (**Informatyk.cs**) konieczne jest dołączenie przestrzeni nazw `System.Data` za pomocą dyrektywy `using`.

3.3. Dodanie klasy `Lekarz` do biblioteki klas

Klasa o nazwie `Lekarz` ma być klasą pochodną po klasie `Pracownik`. Dla tej klasy należy zdefiniować następujące składowe:

- Prywatne pola:
  - `string specjalizacja;`
  - `string tytul;`
- Publiczne właściwości:
  - `Specjalizacja` zwracającą i ustawiającą wartość pola `specjalizacja`.
  - `Tytul` zwracającą i ustawiającą wartość pola `tytul`.
  - `Zawod` zwracającą składową `Lekarz` wyliczenia `Zawody`. Właściwość ma nadpisywać tę samą właściwość z klasy podstawowej (słowo kluczowe `override`).
- Publiczne metody:
  - bezparametrowy konstruktor inicjalizujący pola składowe zerami. W konstruktorze, na liście inicjalizacyjnej należy wywołać bezargumentowy konstruktor klasy bazowej.

- konstruktor inicjalizujący wszystkie pola składowe na podstawie argumentów. Konstruktor ma posiadać te same argumenty co konstruktor klasy `Pracownik` i dodatkowe parametry potrzebne do zainicjalizowania pól składowych obiektu typu `Lekarz`).
- konstruktor kopiujący inicjalizujący wszystkie pola składowe na podstawie argumentu wzorcowego typu `Lekarz`.
  Uwaga: W każdym konstruktorze, na liście inicjalizacyjnej, należy wywołać odpowiedni konstruktor klasy bazowej w celu ustawienia niedostępnych pól klasy bazowej.
- `override Pracownik Clone()`, zwracającą instancję nowostworzonego obiektu typu `Lekarz` na wzór danego (wywołanie konstruktora kopiującego).
- `override string SzczegolyZawodu()` zwracającą łańcuch w formie: specjalizacja _s_ tytuł, gdzie _s_ jest znakiem separatora `'\t'`.
- `override string ToString()` zwracającą łańcuch opisujący dane pracownika w formie: imię nazwisko dzień miesiąc rok ulica numer-domu miasto specjalizacja tytuł.
- `override string FormatWyjsciowy()` zwracającą łańcuch opisujący dane pracownika w formie:
  Zawód: Lekarz
  Imię i nazwisko: imię nazwisko
  Data urodzenia: dzień miesiąc rok
  Adres zamieszkania: ulica numer domu miasto
  Dane dodatkowe: specjalizacja tytuł
- `override void OdczytConsole()`, której zadaniem jest wczytanie z klawiatury wszystkich danych dla lekarza. Uwaga: W przypadku pobrania z klawiatury informacji o lekarzu, należy uwzględnić jego dane podstawowe (podrozdział 3.3).
- `override void ZapisConsole()`, której zadaniem jest wypisanie na ekran wszystkich danych lekarza. Uwaga: W przypadku wyświetlenia informacji o lekarzu, należy uwzględnić jego dane podstawowe (podrozdział 3.3).
- `override void OdczytXml(DataRow dr)` – ciało metody należy na chwilę obecną zostawić puste.
  Uwaga: W pliku definicji klasy (**Lekarz.cs**) konieczne jest dołączenie przestrzeni nazw `System.Data`.

3.4. Dodanie klasy `Nauczyciel` do biblioteki klas

Klasa o nazwie `Nauczyciel` ma być klasą pochodną po klasie `Pracownik`. Dla tej klasy należy zdefiniować następujące składowe:

- Prywatne pola:
  - `string przedmiot;`
  - `string tytul;`
- Publiczne właściwości:
  - `Przedmiot` zwracającą i ustawiającą wartość pola `przedmiot`.
  - `Tytul` zwracającą i ustawiającą wartość pola `tytul`.
  - `Zawod` zwracającą składową `Nauczyciel` wyliczenia `Zawody`. Właściwość ma nadpisywać tę samą właściwość z klasy podstawowej (słowo kluczowe `override`).
- Publiczne metody:
  - bezparametrowy konstruktor inicjalizujący pola składowe zerami. W konstruktorze, na liście inicjalizacyjnej należy wywołać bezargumentowy konstruktor klasy bazowej.
  - konstruktor inicjalizujący wszystkie pola składowe na podstawie argumentów. Konstruktor ma posiadać te same argumenty co konstruktor klasy `Pracownik` i dodatkowe parametry potrzebne do zainicjalizowania pól składowych obiektu typu `Nauczyciel`).

- konstruktor kopiujący inicjalizujący wszystkie pola składowe na podstawie argumentu wzorcowego typu `Nauczyciel`.
  Uwaga: W każdym konstruktorze, na liście inicjalizacyjnej, należy wywołać odpowiedni konstruktor klasy bazowej w celu ustawienia niedostępnych pól klasy bazowej.
- `override Pracownik Clone()`, zwracającą instancję nowostworzonego obiektu typu `Nauczyciel` na wzór danego (wywołanie konstruktora kopiującego).
- `override string SzczegolyZawodu()` zwracającą łańcuch w formie: przedmiot _s_ tytuł, gdzie _s_ jest znakiem separatora `'\t'`.
- `override string ToString()` zwracającą łańcuch opisujący dane pracownika w formie: imię nazwisko dzień miesiąc rok ulica numer-domu miasto przedmiot tytuł.
- `override string FormatWyjsciowy()` zwracającą łańcuch opisujący dane pracownika w formie:
  Zawód: Nauczyciel
  Imię i nazwisko: imię nazwisko
  Data urodzenia: dzień miesiąc rok
  Adres zamieszkania: ulica numer domu miasto
  Dane dodatkowe: przedmiot tytuł
- `override void OdczytConsole()`, której zadaniem jest wczytanie z klawiatury wszystkich danych dla nauczyciela. Uwaga: W przypadku pobrania z klawiatury informacji o nauczycielu, należy uwzględnić jego dane podstawowe (podrozdział 3.3, 3.4).
- `override void ZapisConsole()`, której zadaniem jest wypisanie na ekran wszystkich danych nauczyciela. Uwaga: W przypadku wyświetlenia informacji o nauczycielu, należy uwzględnić jego dane podstawowe (podrozdział 3.3, 3.4).
- `override void OdczytXml(DataRow dr)` – ciało metody należy na chwilę obecną zostawić puste.
  Uwaga: W pliku definicji klasy (**Nauczyciel.cs**) konieczne jest dołączenie przestrzeni nazw `System.Data`.

3.5. Dodanie do biblioteki klas klasy `FormatDanych`

Wstęp

Przed definicją klasy `FormatDanych`, konieczne jest zrozumienie zasady przechowywania danych w formie tabel w środowisku .NET. Poniższe podrozdziały prezentują krótki opis koniecznych do tego celu typów, metod oraz właściwości wraz z przykładami, które ułatwią implementację klasy.

Klasa `DataSet` i jej wybrane składowe

Klasa `DataSet` reprezentuje w środowisku .NET pamięć podręczną. Jest głównym komponentem w bibliotece **ADO.NET** zawartym w przestrzeni `System.Data`, który udostępnia klasy do operacji na bazach danych. Jedną z właściwości typu `DataSet` jest kolekcja obiektów `DataTable`. Klasa ta reprezentuje pojedynczą tabelę w bibliotece **ADO.NET**. Udostępnia szereg właściwości i metod, dzięki czemu może stanowić odrębne źródło danych lub kolekcję tabel w obiekcie typu `DataSet`. Do najważniejszych właściwości klasy `DataTable` należą składowe `Columns` oraz `Rows`. Właściwość `Columns` udostępnia zbiór kolumn (obiekty typu `DataColumn`) danej tabeli. Wywołując na rzecz tej właściwość metodę `Add` można w bardzo prosty sposób dodać do tabeli kolumnę o określonym nagłówku i o typie obiektu, jaki będzie przechowywany w wybranej kolumnie. Z kolei właściwość `Rows` przechowuje wiersze (rekordy) tabeli w formie obiektów `DataRow`, które należy ustawiać poprzez zastosowanie indeksera (operatora `[]`) na obiekcie podając jako

argument nazwę kolumny. Tak jak właściwość `Columns`, również składowa `Rows` posiada metodę `Add` służącą do dodawania wierszy do tabeli.
Klasa `DataSet` oferuje również bardzo przejrzysty i intuicyjny sposób zapisywania i odczytywania danych zaprezentowanych w formie tabeli w formacie Xml. Zostanie on wykorzystany do operacji wejścia-wyjścia w tworzonej aplikacji.
Na sam koniec krótkiego omówienia klasy `DataSet` warto dodać, że obiekt tej klasy może zostać użyty jako łącznik z bazą danych i pobrać z niej wymaganą informację. Do tego celu może posłużyć klasa `SqlDataAdapter`, która po nawiązaniu połączenia ze źródłem, potrafi wypełnić komponent `DataSet` (wywołanie metody `Fill`).

Tworzenie i wypełnianie tabeli

Stworzenie jednej lub więcej tabel w ramach obiektu typu `DataSet` sprowadza się do wykonania krok po kroku kilku wymaganych czynności. Na samym początku, konieczne jest zdefiniowanie obiektu typu `DataSet` oraz `DataTable` za pomocą konstruktorów, których argumenty są reprezentowane poprzez łańcuchy nadające nazwę całemu komponentowi jak i danej tabeli. Listing 3.2 ilustruje definicje tych obiektów.

```
DataSet ds = new DataSet();
DataTable dt = new DataTable(nazwa);
```

Listing 3.2: Egzemplarze klasy `DataSet` i `DataTable` wymagane do przechowywania danych w formie tabeli.

gdzie `nazwa` jest dowolnym łańcuchem znaków. Następnie należy stworzyć poszczególne kolumny tabeli poprzez wywołanie metody `Add` na rzecz właściwości `Columns`. Dwie pierwsze linijki Listingu 3.3. przedstawiają dodanie do tabeli `dt` dwóch kolumn o nagłówku `"Wydział"` oraz `"Rok"` przechowujących odpowiednio obiekty typu `String` oraz `Int32`.

```
dt.Columns.Add("Wydział", typeof(String));
dt.Columns.Add("Rok", typeof(Int32));
ds.Tables.Add(dt);
```

Listing 3.3: Dodanie do tabeli `dt` kolumn o nagłówku `"Wydział"` i `"Rok"` do przechowywania obiektów typu `String` i `Int32`.

Ostatnia instrukcja kodu źródłowego dodaje tabelę do komponentu danych. Listing 3.3 zamyka etap tworzenia struktury pojedynczej tabeli w ramach obiektu `ds`. Kolejny krok polega na wypełnieniu tabeli poszczególnymi wierszami. Do tego celu, wymagane jest stworzenie obiektu typu `DataRow` i jego inicjalizacja za pomocą metody `NewRow`, składowej klasy `DataTable`. Ze względu na to, że tabela będzie się składała z określonej liczby wierszy, każdy taki wiersz musi być nową instancją typu `DataRow`. Do każdej komórki wiersza można odnieść się za pomocą indeksera podając jako argument nazwę kolumny i przypisać wartość wyspecyfikowaną w typie kolumny. Na sam koniec, stworzony i wypełniony wierz należy dodać do tabeli za pomocą metody `Add` wywołanej na rzecz właściwości `Rows` obiektu `dt`. Listing 3.4 przedstawia kod realizujący omówione kroki.

```
DataRow dr = dt.NewRow();
dr["Wydział"] = "Chemiczny";
dr["Rok"] = 2;
dt.Rows.Add(dr);
```

Listing 3.4: Stworzenie, wypełnienie i dodanie wiersza do tabeli.

Zapis i odczyt danych w formacie Xml

Zapis zawartości obiektu typu `DataSet` do pliku w formacie Xml obsługuje metoda `WriteXml`, która jako argument przyjmuje ścieżkę docelowego pliku. Jest to metoda składowa klasy `DataSet`, którą należy wywołać na rzecz komponentu przechowującego tabelę. Do odczytu danych z tabeli służy metoda `ReadXml`, której argument stanowi źródło odczytu informacji w formie łańcucha. Metoda ta jest również składową klasy

DataSet, zatem pobranie danych z pliku wymaga jej wywołania na rzecz komponentu. Listing 3.5 prezentuje sposób wywołania wyżej omówionych metod w celu zapisu i odczytu danych z pliku w formacie Xml.

```
ds.WriteXml("plik.xml");
ds.ReadXml("plik.xml");
```

Listing 3.5: Zapis i odczyt danych w formacie Xml przy użyciu metod składowych komponentu typu DataSet.

gdzie "plik.xml" jest łańcuchem specyfikującym pełną ścieżkę do pliku.

Definicja klasy FormatDanych

Klasa FormatDanych, w ramach aplikacji, odpowiedzialna będzie za zapis i odczyt danych pracowników z pliku. Dane te, zawarte w obiektach typu Pracownik i pochodnych, przechowywane są w kolekcji generycznej List<Pracownik>, natomiast w pliku zostaną ujęte w formie tabeli wewnątrz komponentu biblioteki **ADO.NET**, jakim jest omówiona w poprzednim podrozdziale klasa DataSet. Wykorzystanie tego właśnie komponentu pozwoli na sformatowanie danych do postaci Xml i dokonanie operacji wejścia-wyjścia w tym formacie.

Aby umożliwić przejrzyste sformatowanie danych do postaci Xml, w klasie FormatDanych należy zdefiniować następujące składowe:

- Prywatne pola:
  - string sciezka;
  - DataSet ds;
  - DataTable dt;
- Publiczną właściwość Sciezka typu string ustawiającą i zwracającą wartość pola składowego sciezka.
- Publiczne metody:
  - void InicjalizujKomponent(), w której należy zainicjalizować pola składowe. Pole ds należy ustawić poprzez wywołanie bezparametrowego konstruktora klasy DataSet, natomiast pole dt wywołując konstruktor klasy DataTable z parametrem, którym jest łańcuch "Pracownicy". Listing 3.2 przedstawia sposób wywoływania omawianych konstruktorów.
  - bezargumentowy konstruktor domyślny, w którym należy wywołać wcześniej zdefiniowaną metodę InicjalizujKomponent.
  - void StworzKolumny(), której zadaniem jest dodanie do tabeli dt kolumn: "Imię", "Nazwisko", "Data urodzenia", "Adres zamieszkania", "Dane szczegółowe" oraz "Zawód", wszystkie przechowujące obiekty typu String. Utworzoną tabelę należy dodać do komponentu ds. w sposób zaprezentowany na Listingu 3.3.
    Uwaga: Metoda StworzKolumny musi być wywołana w metodzie InicjalizujKomponent po inicjalizacji obiektów ds i dt.
  - void ZapisXml(List<Pracownik> lista), zapisującą zawartość komponentu ds do pliku przy pomocy metody WriteXml (Listing 3.5). Ze względu na to, iż metoda Zapisz ma za zadanie umieścić w pliku tabelę wchodzącą w skład obiektu ds, tabela ta musi najpierw zostać wypełniona obiektami listy. Dobrym zwyczajem jest umieszczenie kodu źródłowego implementującego wypełnienie tabeli w osobnej metodzie i wywołanie tej metody przed zapisem danych do pliku.
    W tym celu, do klasy FormatDanych trzeba dodać kolejną metodę o sygnaturze void WypełnijTabele(List<Pracownik> lista), mającą za zadanie wypełnić tabelę dt obiektami kolekcji generycznej lista. Metoda może być zdefiniowana jako publiczna. Listing 3.6 przedstawia fragment tej metody. Warto zwrócić na nim uwagę na kilka fragmentów kodu. Po pierwsze, na samym początku, kolekcja rzędów danej tabeli jest czyszczona (wywołanie metody Clear na rzecz właściwości Rows obiektu tabeli). Zabieg ten wykonany jest po to, aby przy każdorazowym wypełnianiu tabeli

usuwać jej starą zawartość. Po drugie, dodawanie elementów do tabeli przebiega po wszystkich elementach kolekcji `lista`, a poszczególne jej komórki wypełniane są łańcuchami zwróconymi poprzez właściwości iteratora pętli `foreach` (obiekt `p`). Po trzecie, i co najważniejsze, dochodzi do wypełnienia kolumny `"Dane szczegółowe"` w zależności od szczegółów zawodu pracownika, a ujmując ściślej – na podstawie wartości zwracanej przez metodę `SzczegolyZawodu`, która jest oznaczona jako wirtualna w klasie `Pracownik` oraz przesłonięta (słowo kluczowe `override`) we wszystkich klasach pochodnych. Elementy kolekcji `lista` mogą być instancjami typu `Pracownik`, `Informatyk`, `Lekarz` lub `Nauczyciel`, zatem w zależności od typu obiektu oraz wyżej wspomnianego przesłonięcia metody, dojdzie do polimorfizmu (późnego wiązania), i odpowiednie dane szczegółowe zawodu zostaną wpisane do komórki tabeli. Będzie to miało miejsce w trakcie działania programu, dlatego nie jest możliwe określenie tego typu w chwili obecnej ani nawet w trakcie kompilacji. Istnieje jednak pewność, że obiekt `p` jest zainicjalizowany odpowiednią instancją: podstawową bądź pochodną. Inicjalizacja ta miała miejsce w trakcie dodawania obiektów do listy w metodzie `Dodaj` (Listing 2.5) i w sposób jednoznaczny określiła typ obiektu. Identyczna sytuacja zachodzi w przypadku wypełniania komórki związanej z zawodem pracownika z tą różnicą, że w tym wypadku właściwość `Zawod` jest określona jako wirtualna iw klasie podstawowej `Pracownik` przesłonięta we wszystkich klasach pochodnych: `Informatyk`, `Lekarz` i `Nauczyciel`.

```
public void WypełnijTabele(List<Pracownik> lista)
{
   dt.Rows.Clear();
   DataRow dr;
   foreach (Pracownik p in lista)
   {
      dr = dt.NewRow();
      dr["Imię"] = p.Imie;
      dr["Nazwisko"] = p.Nazwisko;
      //Pozostałe dane: data urodzenia i adres zamieszkania
      dr["Dane szczegółowe"] = p.SzczegolyZawodu();
      dr["Zawód"] = p.Zawod.ToString();;
      dt.Rows.Add(dr);
   }
}
```

Listing 3.6: Wypełnienie tabeli wchodzącej w skład komponentu `ds` zawartością listy generycznej.

Uwaga 1: Metoda `ZapisXml` będzie wywoływana w trakcie zapisu danych do pliku w metodzie `ZapisXml` klasy `Lista`. Z tego powodu, podczas każdej próby przesłania danych, komponent `DataSet` musi być uaktualniany: inicjalizacja kontenera, powtórne ustawienie kolumn i wypełnienie rekordów danymi. Aby ten proces zawsze wykonywał się poprawnie, na samym początku metody `ZapisXml` klasy `FormatDanych` konieczne jest umieszczenie instrukcji wywołania metody `InicjalizujKomponent`.

Uwaga 2: Dopiero po wywołaniu metody `WypełnijTabele`, w metodzie `ZapisXml` klasy `FormatDanych` można dokonać ostatecznej operacji zapisu do pliku za pomocą instrukcji: `ds.WriteXml(sciezka);`

- `List<Pracownik> OdczytXml()`, której zadaniem jest: a) stworzenie lokalnego obiektu listy generycznej, b) odczytanie danych z pliku do komponentu `ds` za pomocą metody `ReadXml` (Listing 3.5) c) wypełnienie listy odczytanymi danymi i zwrócenie obiektu listy. Na Listingu 3.7 przedstawiony jest fragment metody implementującej wyżej wymienione czynności. Można w nim zauważyć, że po utworzeniu obiektu listy, do którego będą dodawane obiekty odpowiednich klas, inicjalizowana jest składowa klasy `FormatDanych`, czyli komponent typu `DataSet` i wczytane są do niego dane z pliku w formacie Xml. Następnie tworzony jest lokalny obiekt typu `Pracownik` – posłuży on do pobrania

danych w zależności od wykonywanego zawodu. Kolejny fragment kodu jest kluczowy: występują tutaj dwie pętle (`foreach`): pierwsza – po wszystkich tabelach w kolekcji tabel komponentu `ds` oraz druga – po każdym rzędzie tabeli. W drugiej pętli sprawdzany jest cyklicznie zawód pracownika, który jest umieszczony w komórce `dr["Zawód"]`. Jeżeli komórka ta przechowuje łańcuch `"Informatyk"`, instancja `p` inicjalizowana jest adresem obiektu klasy `Informatyk`. Dochodzi tutaj zatem do standardowej konwersji typów w hierarchii dziedziczenia – typ `Informatyk` jest pochodnym po klasie `Pracownik`, w związku z czym rzutowanie jest prawidłowe. Można się łatwo domyślić, że w zależności od zwracanego łańcucha zawartego w polu `dr["Zawód"]`, obiekt `p` będzie zawsze inicjalizowany jedną z trzech instancji typów pochodnych. Jest to bardzo istotne z punktu widzenia dwóch ostatnich instrukcji zewnętrznej pętli omawianego Listingu 3.7. Pierwsza z nich to wywołanie metody `OdczytXml` na rzecz obiektu `p`. Pełne wyjaśnienie tej instrukcji zaprezentowane jest w następnej sekcji, gdyż wymaga ona definicji kilku metod. Natomiast druga linijka przedstawia wywołanie metody `Add` na rzecz lokalnego obiektu listy generycznej `lista`. Należy pamiętać, że obiekt ten ma przechowywać egzemplarze klasy `Pracownik` a także po niej pochodne. Przekazanie do metody `Add` argumentu w postaci instancji `p`, spowoduje dodanie obiektu do listy. Ze względu na to, że obiekt `p`, w zależności od zawartości komórki tabeli, będzie zainicjalizowany konstruktorem klasy `Informatyk`, `Lekarz`, lub `Nauczyciel`, metoda `Add` doda do listy egzemplarz różnego typu. Dzięki wcześniejszemu wywołaniu metody `OdczytXml`, do obiektu `p` zostanie wczytana odpowiednia zawartość z pliku, z wyszczególnieniem danych podstawowych oraz szczegółowych pracownika. Przez to, kolekcja będzie przechowywała pracowników o różnych zawodach.

```
public List<Pracownik> OdczytXml()
{
   List<Pracownik> lista = new List<Pracownik>();
   ds = new DataSet();
   ds.ReadXml(sciezka);
   Pracownik p = new Pracownik();
   foreach (DataTable dt in ds.Tables)
   {
       foreach (DataRow dr in dt.Rows)
       {
          switch ((dr["Zawód"]).ToString())
          {
              case "Informatyk":
                 {
                    p = new Informatyk();
                    break;
                 }
              //Przypadki obsługujące pozostałe zawody
          }
          p.OdczytXml(dr);
          lista.Add(p);
       }
   }
   return lista;
}
```

Listing 3.7: Odczytanie danych z pliku w formacie Xml i wypełnienie listy odpowiednimi pracownikami.

Implementacja metody `OdczytXml` w klasie `Pracownik`

Jak wspomniano we wcześniejszym podrozdziale, wywołanie metody `OdczytXml` na rzecz obiektu `p`, ma umożliwić pobranie z pliku danych podstawowych oraz szczegółowych pracowników. Instancja `p`, w zależności od zawodu, inicjalizowana jest adresem obiektów pochodnych. W związku z tym, wywołanie

omawianej metody jest zależne od tymczasowego adresu obiektu `p`. Musi to być zatem metoda wirtualna zdefiniowana w klasie bazowej `Pracownik` oraz przesłonięta we wszystkich klasach pochodnych: `Informatyk`, `Lekarz`, `Nauczyciel`. Aplikacja zaprojektowana do tej pory posiada szkielety tych metody, jednak bez implementacji (podrozdział 1.3 oraz 3.2, 3.3 i 3.4). Listing 3.8 prezentuje fragment definicji metody `OdczytXml` zdefiniowanej dla klasy bazowej `Pracownik`.

```
public virtual void OdczytXml(DataRow dr)
{
   this.imie = dr.ItemArray[0].ToString();
   this.nazwisko = dr.ItemArray[1].ToString();
   string[] data = dr.ItemArray[2].ToString().Split(' ');
   this.dataUrodzenia.Dzien = Int32.Parse(data[0]);
   this.dataUrodzenia.Miesiac = data[1];
   this.dataUrodzenia.Rok = Int32.Parse(data[2]);
   //Pobranie danych adresowych
}
```

Listing 3.8: Definicja wirtualnej metody `OdczytXml` w klasie `Pracownik`.

Jak można zaobserwować, argumentem metody jest obiekt `dr` typu `DataRow`. Wykorzystanie tego typu w pliku, w którym zdefiniowana jest metoda (**Pracownik.cs**), wymaga przed definicją klasy dołączenia przestrzeni `System.Data` za pomocą dyrektywy `using`. Metoda `OdczytXml` wywołana jest w metodzie `Odczytaj`, składowej klasy `FormatDanych`, zatem argument `dr` przechowuje informacje zawarte we wszystkich rzędach tabeli `dt` kontenera `ds`, na rzecz którego wywołana została metoda `ReadXml` (Listing 3.7). Do danych poszczególnych wierszy obiektu `dr` można się dostać w bardzo łatwy sposób. Klasa `DataRow` udostępnia właściwość `ItemArray` zwracającą zawartość każdego rekordu w formie tablicy typu `Object`, której elementy stanowią kolejne komórki danego wiersza. Ich ilość równa jest zatem liczbie kolumn tabeli. Aby wydobyć zawartość takiej komórki, należy się do niej odwołać przy pomocy operatora `[]` właściwości `ItemArray` i odpowiedniego indeksu. Warto pamiętać, że, w zależności od typu zmiennej, do której zostaną przypisywane dane z komórki z wiersza, konieczne może być zastosowania odpowiedniego rzutowania. Na Listingu 3.8 na uwagę zasługuje instrukcja pobierająca dane z trzeciej komórki wiersza tabeli. Najpierw zwrócona zawartość tej komórki (`dr.temArray[2]`) konwertowana jest na łańcuch znakowy. Następnie łańcuch ten dzielony jest na *n* podłańcuchów na podstawie separatora `' '`, argumentu metody `Split`. Na sam koniec, wszystkie pod-łańcuchy zwrócone przez metodę `Split` tworzą i od razu inicjalizują lokalną tablicę `data` typu `string`. Konwerrsja elementu `dr.temArray[2]` na typ `string` wydobywa z trzeciej komórki każdego wiersza dane na temat daty w formie: dzień miesiąc rok, dlatego też poszczególne elementy tablicy `data` przechowują: wartość liczbową dnia, miesiąc oraz reprezentację liczbową roku jako łańcuch znaków. Przykładowo, tablica `data` może przechowywać następujące trzy elementy `data[0]="29"`, `data[1]="lutego"` oraz `data[2]="2012"`. Aby przypisać wartości dnia oraz roku (na Listingu 3.8 realizowane jest to przy użyciu odpowiednio właściwości `Dzien` oraz `Rok`) do składowych klasy `Pracownik`, należy dodatkowo skonwertować dane łańcuchy do typu `Int32` (alias `int`). Operacja ta przedstawiona jest na Listingu 3.8 przy użyciu metody `Parse`. Metoda `Parse` jako argument przyjmuje typ `string`, dlatego konieczne jest, aby elementy `data[0]` i `data[2]` zwracały wartości niezerowe (różne od `null`). Jeżeli, przez przypadek, komórki te zwróciłyby referencję zerową, wygenerowany zostałby wyjątek typu `FormatException`. Należy temu zapobiec i zastosować obsługę sytuacji wyjątkowej. Kod obsługujący taką sytuację powinien zostać umieszczony w metodzie odpowiedzialnej za interfejs komunikacji z użytkownikiem (np.: metoda `Main` dla aplikacji konsolowej lub metoda obsługująca zdarzenie odczytu z pliku dla aplikacji okienkowej).

Implementacje metod `OdczytXml` w klasach pochodnych

Metoda `OdczytXml` zdefiniowana jest w klasie `Pracownik` jako wirtualna i przesłonięta jest we wszystkich klasach pochodnych. Takie podejście umożliwia zastosowanie polimorfizmu podczas pobierania danych z pliku, gdyż nie będzie wówczas istotne to, jaki jest typ obiektu, tylko do której instancji się on odwołuje (Listing 3.7). Należy jednak pamiętać, że pełne odczytanie danych dla obiektów pochodnych wymaga najpierw pobrania informacji wspólnej dla każdego zawodu, a to wiąże się z wywołaniem metody `OdczytXml` dla klasy bazowej. Instrukcja za to odpowiedzialna musi być zatem umieszczona przed wczytaniem danych szczegółowych dla poszczególnych pracowników. Tak jak to przedstawiono w podrozdziałach 3.2, 3.3 i 3.4, argumentem każdej metody `OdczytXml` klasy pochodnej, jest obiekt typu `DataRow`. Obiekt ten przechowuje wszystkie dane zawarte w pliku Xml przez co, tak jak w przypadku metody `OdczytXml` klasy `Pracownik`, będzie możliwe pobranie danych szczegółowych do obiektu pochodnego. Listing 3.9 ilustruje implementację metody `OdczytXml` dla klasy `Nauczyciel`.

```
public override void OdczytXml(DataRow dr)
{
   base.OdczytXml(dr);
   string[] szczegoly = dr.ItemArray[4].ToString().Split('\t');
   this.przedmiot = szczegoly[0];
   this.tytul = szczegoly[1];
}
```

Listing 3.10: Definicja metody `OdczytXml` dla klasy `Naucyzciel`.

Pierwsza linijka kodu tej metody odpowiedzialna jest właśnie za pobranie wspólnej informacji dla wszystkich zawodów – do pracy rusza metoda `OdczytXml` klasy `Pracownik`. Wykonane zostaną zatem wszystkie instrukcje z Listingu 3.8 (wczytanie danych o imieniu, nazwisku, dacie urodzenia i zawodzie). Na Listingu 3.10 warto również zwrócić uwagę na inicjalizację lokalnej tablicy `szczegoly`: tablica wypełniana jest dwoma łańcuchami, w których umieszczone są dane szczegółowe pracowników. Separatorem rozdzielającym dane szczegółowe jest znak tabulatora `'\t'`, gdyż dane pracowników mogą być kilkuczłonowe. Przykładowo, podczas zapisu do pliku, dodatkowe informacje o nauczycielu mogą być następujące: przedmiot `"Metodyki i techniki programowania"`, tytuł: `"dr inż."`. Gdyby separatorem był znak spacji `' '`, elementy tablicy `szczegoly` (a przez pola składowe klasy) zostałyby niepełnie zainicjalizowane (miałoby miejsce wczytanie do pierwszej napotkanej spacji).

Modyfikacja klasy `Lista`

Zdefiniowana wcześniej klasa `FormatDanych` precyzuje format przechowywania danych pracowników oraz sposób przeprowadzania operacji odczytu i zapisu informacji do pliku. Implementacja operacji wejścia/wyjścia związana jest jednak z kolekcją obiektów, a mechanizm działania tej kolekcji obsługuje klasa `Lista`. W związku z tym, w klasie tej konieczne jest umieszczenie definicji pola składowego przechowujące informacje o formacie danych. Nazwa tego pola może być dowolna (np.: `df`). Pole to musi zostać zainicjalizowane w konstruktorze klasy `Lista`, poprzez wywołanie konstruktora klasy `FormatDanych`.

Przypisanie wartości początkowej dla obiektu `df` utworzy kontener przechowujący tabele z odpowiednio sformatowanymi kolumnami. Odczyt i zapis danych do pliku będzie tylko wymagał wywołania na rzecz obiektu `df` metod odczytujących zapisujących informacje o pracownikach. W tym celu do klasy `Lista`, należy dodać dwie metody `OdczytXml` i `ZapisXml` a w nich z kolei wywołać metody odczytujące i zapisujące dane na rzecz instancji `dsf`. Listing 3.10 ilustruje sposób realizacji tych czynności.

```
public void OdczytXml()
{
   lista = df.OdczyXml();
}
public void ZapisXml()
{
   df.ZapisXml(lista);
}
```

Listing 3.10: Definicja metody `OdczytXml` i `ZapisXml` dla klasy `Lista`.

W ciele metody `OdczytXml`, składowej klasy `Lista` (pole `lista`) przypisywany jest obiekt zwracany przez metodę `Odczytaj`. Jest to kolekcja generyczna typu `<Pracownik>` wypełniona instancjami klas `Informatyk`, `Lekarz` i `Nauczyciel`, których dane pobrane zostały z pliku w formacie Xml (Listing 3.7).

Uwaga: Ze względu na to, że metody `ZapisXml` oraz `OdczytXml` klasy `FormatDanych` wymagają sprecyzowanej ścieżki dostępu do pliku, w klasie `Lista` konieczne jest zdefiniowanie publicznej właściwości ustawiającej ścieżkę zapisu/odczytu danych. Listing 3.11 przedstawia definicję tej właściwości.

```
public string SciezkaDoPliku
{
   get { return df.Sciezka; }
   set { df.Sciezka = value; }
}
```

Listing 3.11: Definicja właściwości `SciezkaDostepu` dla klasy `Lista`.

Właściwość ta zostanie użyta w trakcie pobierania ścieżki dostępu zarówno z linii wiersza poleceń jak i z okna dialogowego zapisu lub odczytu z pliku.

W podrozdziale 2.2, w ramach tworzenia klasy `Lista`, należało zdefiniować metodę `OdczytConsole`, wczytującą z klawiatury dane do nowostworzonego obiektu klasy `Pracownik` i dodającą obiekt do listy. Na tym etapie rozbudowy projektu, program umożliwia pobieranie danych nie tylko dla pracowników, ale również informatyków, lekarzy oraz nauczycieli. Dlatego też, metoda `OdczytConsole` musi zostać zmodyfikowana tak, aby zaoferować użytkownikowi wybór dowolnego typu pracownika i dodanie go do listy. Dokonanie tego wyboru powinno być uzależnione od wielowariantowej opcji. Listing 3.12 prezentuje przykładowe rozwiązanie. Jak można zauważyć, na samym początku metody tworzony jest lokalny obiekt `p` klasy `Pracownik`. Następnie, za pomocą kilku instrukcji, na ekran wypisywana jest informacja z prośbą o podanie znaku reprezentującego dany zawód. W przypadku naciśnięcia przez użytkownika znaku `'i'`, instancji `p` przypisywany jest adres obiektu klasy pochodnej (`Informatyk`), przez co dochodzi do tzw. konwersji standardowej w dziedziczeniu. Pod koniec, na rzecz obiektu `p` wywoływana jest wirtualna metoda `OdczytConsole`, skladowa klasy `Pracownik`, która jest przesłonięta we wszystkich klasach pochodnych (podrozdział 3.2, 3.3 i 3.4). To spowoduje, że w trakcie działania programu (późne wiązanie) uruchomi się odpowiednia metoda klasy pochodnej (w omawianym przypadku będzie to metoda z klasy `Informatyk`). Wreszcie, w ostaniej instrukcji, do listy, za pomocą metody `Add`, zostanie dodany odpowiednio zainicjalizowany i ustawiony obiekt pochodny. Metoda ta przyjmuje argument typu `Pracownik`, więc zgodność typów będzie zachowana.

```
public void OdczytConsole()
{
   Pracownik p = new Pracownik();
   Console.WriteLine("Podaj typ pracownika");
   Console.WriteLine("i - informatyk");
   Console.WriteLine("l - lekarz");
   Console.WriteLine("n - nauczyciel");
   char typ = char.Parse(Console.ReadLine());
   switch (typ)
   {
      case 'i':
         {
            p = new Informatyk();
            break;
         }
   }
   p.OdczytConsole();
   lista.Add(p);
}
```

Listing 3.12: Definicja metody `OdczytConsole` dla klasy `Lista` umożliwiająca wprowadzanie danych z klawiatury dla różnych pracowników (w tym wypdaku tylko dla informatyka; pozostałe zawody należy zaimplementować analogicznie).