**Wojskowa akademia techniczna**

**im. jarosława dąbrowskiego w warszawie**

wydział cybernetyki



**Systemy baz danych**

**Sprawozdanie z Laboratorium nr 2**

**Obiektowe bazy danych**

Prowadzący: **mgr inż. Maciej Szymczyk**

Wykonane przez: **Joanna Mizera, Monika Owczarek**

Grupa: **I8E2S4**

Data laboratorium: **15.05.2019 r.**

# Identyfikacja klas, metod, związków między klasami dla wybranego obszaru dziedzinowego

Dziedziną naszej bazy danych jest przyszpitalna przychodnia specjalistyczna. Zawiera najważniejsze informacje związane z wizytami pacjentów.

## Klasy

Baza danych składa się z 6 klas.

* Osoba

Jest to klasa abstrakcyjna. Jej atrybutami są imię i nazwisko. Zawiera jedną metodę – jakSieNazywasz.

* Pacjent

Jest to klasa, która dziedziczy po klasie Osoba. Jest reprezentacją pacjenta leczącego się w przychodni. Posiada atrybuty takie jak: identyfikator, adres, miejscowość, dataUrodzenia oraz metodę umowWizyte. Ponadto nadpisuje metodę jakSieNazywasz, którą dziedziczy z klasy bazowej. Klasa powiązana jest z klasą Wizyta.

* Lekarz

Kolejna klasa dziedzicząca po klasie Osoba. Reprezentuje lekarza pracującego w przychodni. Posiada swój identyfikator oraz pozostałe atrybuty takie jak: tytuł, staż pracy. Tak jak w przypadku Pacjenta nadpisuje metodę z klasy bazowej – jakSieNazywasz uzupełniając zwracane informacje o swój tytuł. Klasa powiązana jest z klasą Wizyta oraz z klasą Oddział.

* Wizyta

Klasa reprezentująca wizytę – spotkanie lekarza z pacjentem. Posiada atrybuty dataWizyty, numerGabinetu, wizytaAnulowana oraz metody anulujWizyte (umożliwia anulowanie wizyty), umowWizyte. Klasa ta powiązana jest z klasami Pacjent i Lekarz.

* Oddział

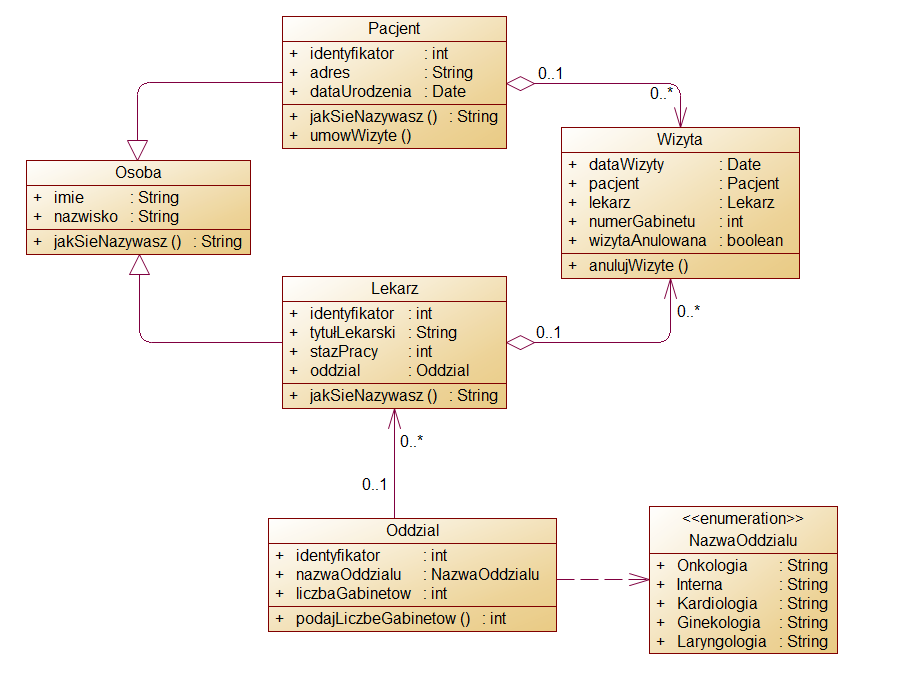
Klasa Oddział reprezentuje oddział w przychodni, do której należą lekarze w niej pracujący. Posiada swój identyfikator oraz atrybut liczbaGabinetow. Ma również jedną metodę podajLiczbeGabinetow zwracającą liczbą gabinetów na danym oddziale. Klasa powiązana jest z klasami Lekarz oraz NazwaOddziału.

* NazwaOddzialu

Klasa NazwaOddzialu jest wyliczeniowym typem danych. Zawiera listę składającą się z 5 nazw oddziałów (onkologia, interna, kardiologia, ginekologia, laryngologia) znajdujących się w przychodni. Powiązana zależnością z klasą Oddział.

## Diagram klas

Poniższy diagram przedstawia model bazy danych. Zawiera opisane wyżej klasy, atrybuty, związki (dziedziczenie, agregacja, asocjacja, zależność). Zaprojektowany został za pomocą narzędzia PowerDesigner.



# Implementacja

## Osoba

Class Przychodnia.Osoba [ Abstract ]  
{  
Property imie As %String [ Required ];  
  
Property nazwisko As %String [ Required ];  
  
Method jakSieNazywasz() As %String  
{  
 Set nazwa = ..imie\_" "\_..nazwisko  
 Return nazwa  
}  
}

## Pacjent

Class Przychodnia.Pacjent Extends (%Persistent, %Populate, %XML.Adaptor, %ZEN.DataModel.Adaptor, Przychodnia.Osoba)  
{  
Property identyfikator As %Integer [ Required ];  
  
Property adres As %String [ Required ];  
  
Property dataUrodzenia As %Date [ Required ];  
  
Relationship wizytyPacjenta As Przychodnia.Wizyta [ Cardinality = many, Inverse = pacjent ];  
  
Method jakSieNazywasz() As %String  
{  
 Set nazwa = "Pacjent"\_" "\_..imie\_" "\_ ..nazwisko  
 Return nazwa  
}  
  
Method umowWizyte(ByRef lekarz As Lekarz) As %String  
{  
 Set wizyta = ##class(Przychodnia.Wizyta).%New()  
 Set wizyta.dataWizyty = ##class(%Library.UTC).NowUTC()  
 Set wizyta.numerGabinetu = $Random(10)  
 Set wizyta.wizytaAnulowana = 0  
 Set wizyta.pacjent = $this  
 Set wizyta.lekarz = lekarz  
 Do wizyta.%Save()  
  
 Return "Wizyta umowiona"  
}  
}

## Lekarz

Class Przychodnia.Lekarz Extends (%Persistent, %Populate, %XML.Adaptor, %ZEN.DataModel.Adaptor, Przychodnia.Osoba)  
{  
Property identyfikator As %Integer [ Required ];  
  
Property tytulLekarski As %String [ Required ];  
  
Property stazPracy As %Integer [ Required ];  
  
Relationship oddzial As Przychodnia.Oddzial [ Cardinality = one, Inverse = lekarze ];  
  
Relationship wizytyLekarza As Przychodnia.Wizyta [ Cardinality = many, Inverse = lekarz ];  
  
Method jakSieNazywasz() As %String  
{  
 Set nazwa = ..tytulLekarski\_" "\_..imie\_" "\_..nazwisko  
 Return nazwa  
}  
}

## Wizyta

Class Przychodnia.Wizyta Extends (%Persistent, %Populate, %XML.Adaptor, %ZEN.DataModel.Adaptor)  
{  
Property dataWizyty As %DateTime [ Required ];  
  
Property numerGabinetu As %Integer [ Required ];  
  
Property wizytaAnulowana As %Boolean [ Required ];  
  
Relationship pacjent As Przychodnia.Pacjent [ Cardinality = one, Inverse = wizytyPacjenta ];  
  
Relationship lekarz As Przychodnia.Lekarz [ Cardinality = one, Inverse = wizytyLekarza ];  
  
Method anulujWizyte() As %String  
{  
 IF ..wizytaAnulowana = 1{  
 Return "Nie można anulować wizyty"  
 }  
 ELSE  
 {  
 Set ..wizytaAnulowana = 1  
 Do %Save()  
 Return "Wizyta anulowana"  
 }  
}  
}

## Oddzial

Class Przychodnia.Oddzial Extends (%Persistent, %Populate, %XML.Adaptor, %ZEN.DataModel.Adaptor)  
{  
Property identyfikator As %Integer [ Required ];  
  
Property liczbaGabinetow As %Integer [ Required ];  
  
Property nazwaOddzialu As Przychodnia.NazwaOddzialu [ Required ];  
  
Relationship lekarze As Przychodnia.Lekarz [ Cardinality = many, Inverse = oddzial ];  
  
Method podajLiczbeGabinetow() As %Integer  
{  
 set liczbaGabinetow = ..liczbaGabinetow  
 return liczbaGabinetow  
}

}

## NazwaOddzialu

Class Przychodnia.NazwaOddzialu Extends %EnumString  
{  
Parameter VALUELIST = ",Onkologia, Interna, Kardiologia, Ginekologia, Laryngologia";  
}

# Wypełnienie bazy danych danymi testowymi

W celu wypełnienia bazy danych danymi testowymi wykorzystana została auto-populacja. W ten sposób utworzone zostały obiekty klas Pacjent, Lekarz, Oddzial.

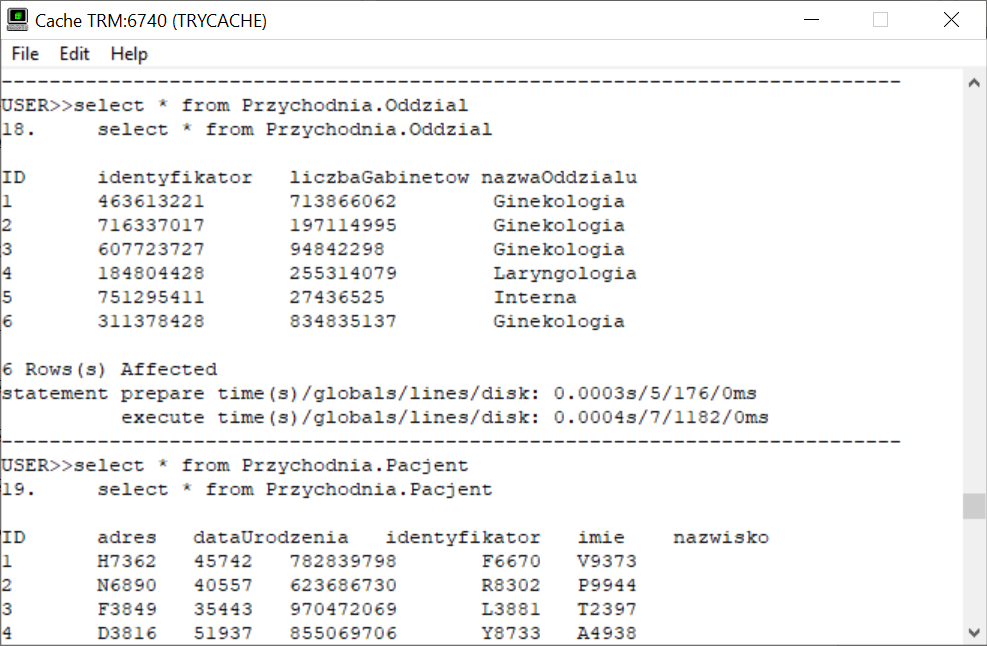
do ##class(Przychodnia.Pacjent).Populate(30)

do ##class(Przychodnia.Lekarz).Populate(30)

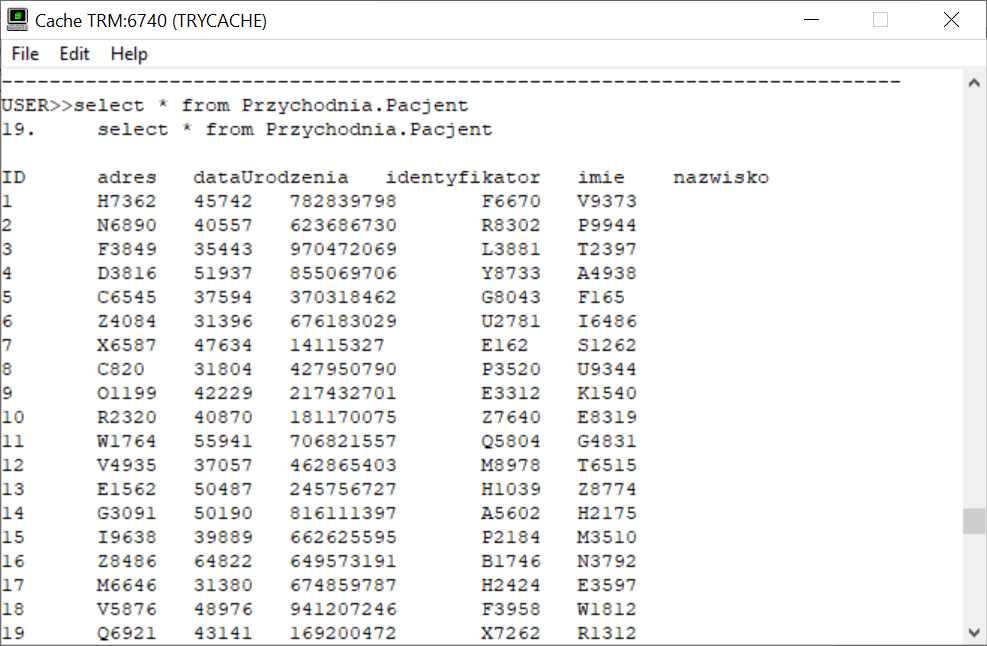
do ##class(Przychodnia.Oddzial).Populate(6)

Aby sprawdzić, czy obiekty zostały utworzone wykorzystano proste zapytania SQL.

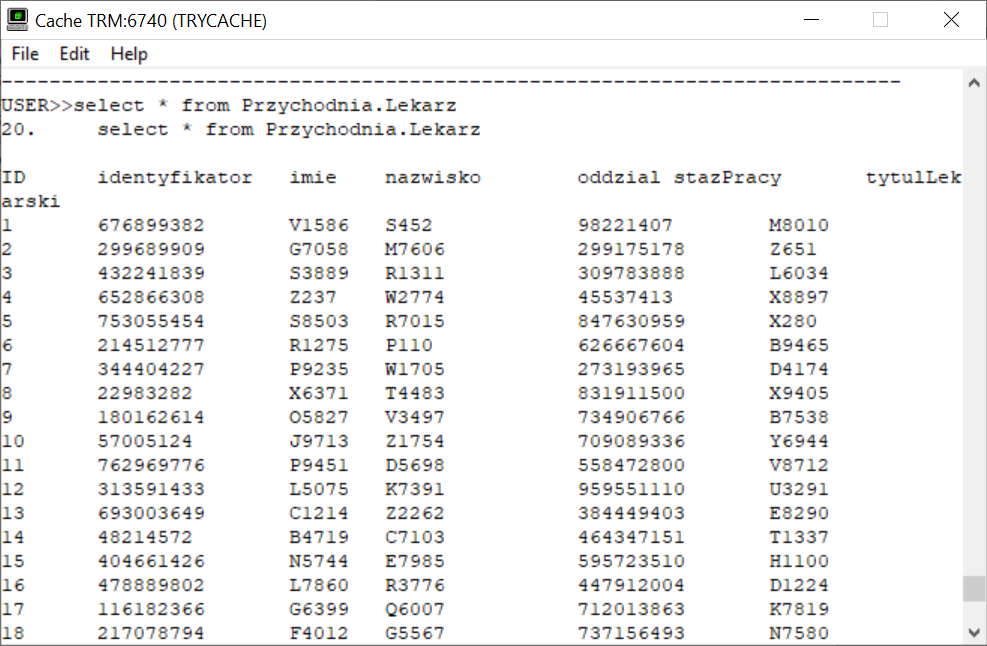
* Dla klasy Oddzial



* Dla klasy Pacjent



* Dla klasy Lekarz

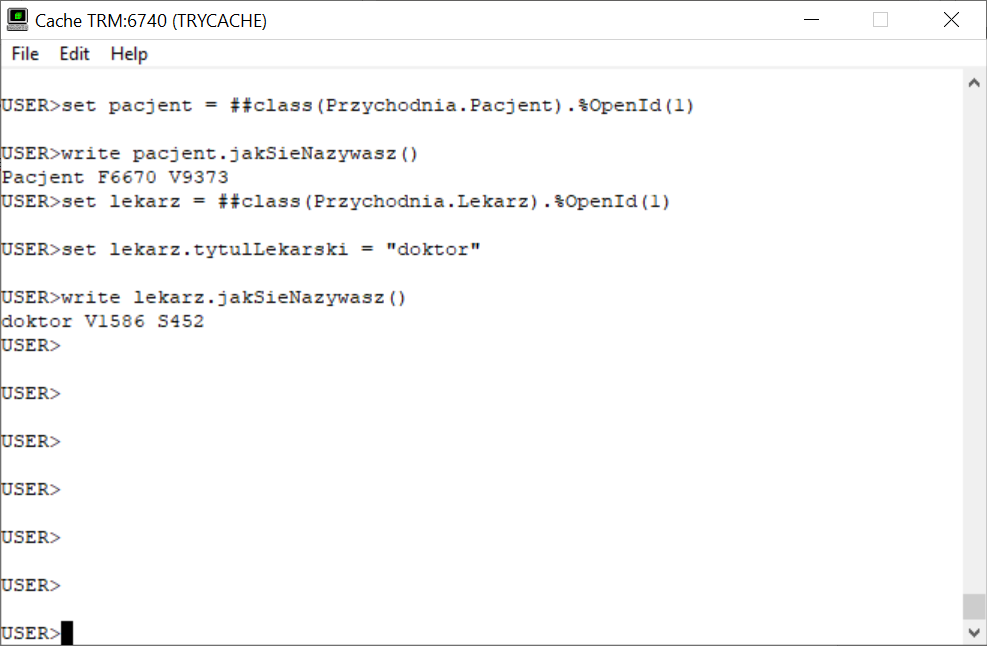


# Wykonanie metod obiektów

* jakSieNazywasz()

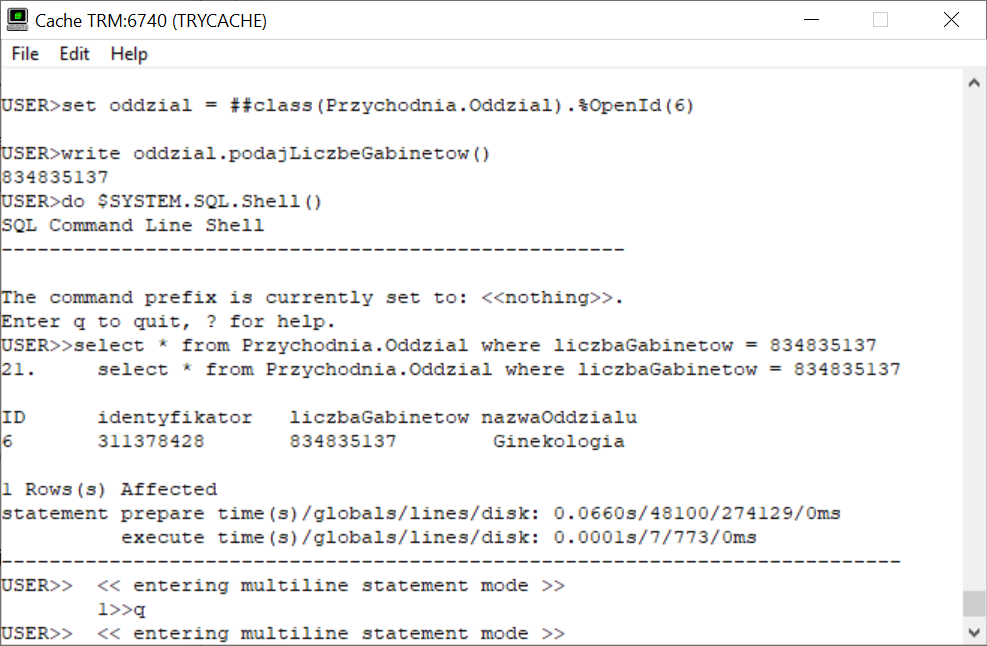
W celu wywołania konkretnej metody należy zacząć od załadowania wybranego obiektu z bazy. Użyta została do tego intrukcja *set pacjent = ##class (Przychodnia.Pacjent).%OpenId(1).* Pobrany został obiekt pacjenta o ID = 1. Następnie wykonana została metoda obiektu jakSieNazywasz() dzięki instrukcji *write pacjent.jakSieNazywasz().* Zgodnie z implementacją metody dla klasy Pacjent, zwróciła nam ona wygenerowane imię i nazwisko pacjenta razem z przedrostkiem „Pacjent”.

W taki sam sposób wykonana została metoda obiektu lekarz. W tym przypadku manualnie zmieniono wartość jednego z atrybutów – tytulLekarski, aby można było stwierdzić prawidłowe działanie metody. Wynikiem jej wykonania jest tytuł lekarski konkretnego lekarza oraz wygenerowane imię i nazwisko.



* podajLiczbeGabinetow()

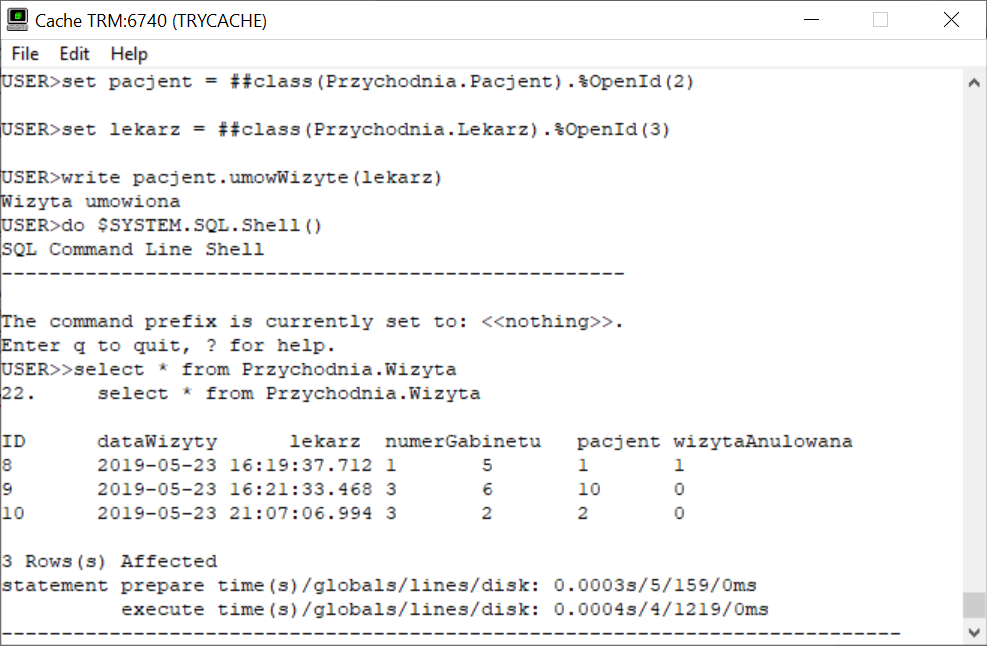
Kolejna metoda zwraca liczbę gabinetów konkretnego oddziału przychodni. Instrukcją *set oddzial = ##class(Przychodnia.Oddzial).%OpenId(6)* wybrano jeden z oddziałów. Następnie wykonano metodę podajLiczbeGabinetow().Po tej operacji sprawdzona została poprawność działania metody za pomocą zapytania w języku SQL.



* umowWizyte()

Wykonanie metody umowWizyte() skutkuje utworzeniem obiektu klasy Wizyta. Przed wykonaniem metody należy pobrać jeden obiekt klasy Pacjent (o ID = 2) i jeden obiekt klasy Lekarz (o ID = 3). Następnie wykonana została metoda z parametrem lekarz dzięki instrukcji *write pacjent.umowWizyte(lekarz)*.

W tym przypadku również sprawdzono działanie metody wyświetlając zawartość bazy danych prostym zapytaniem SQL. Widać, że utworzony został obiekt klasy Wizyta zawierający atrybuty lekarz o ID = 3 i pacjenta o ID = 2.

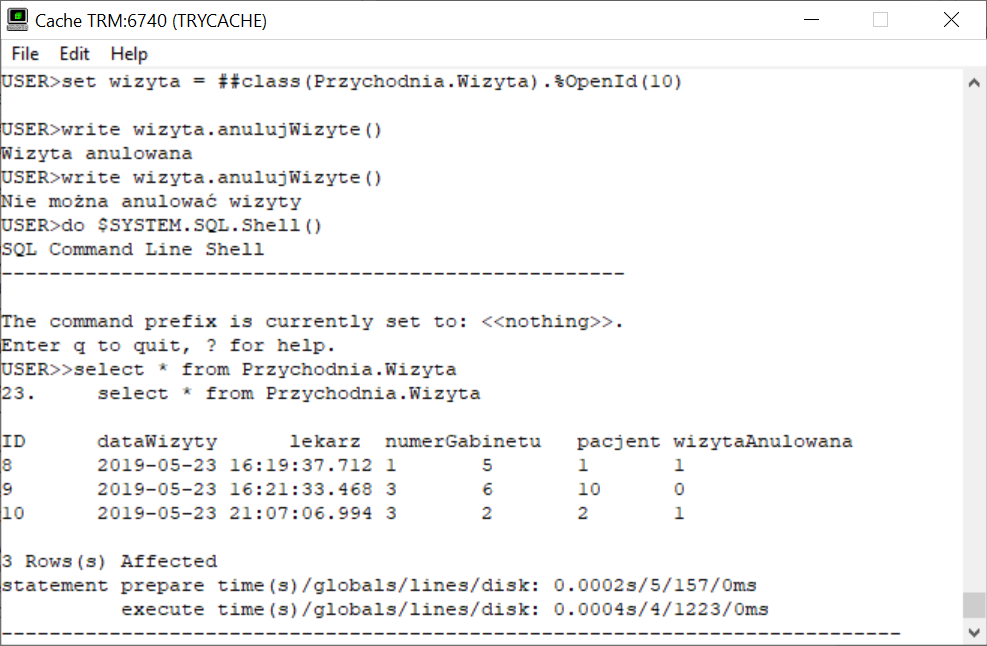


* anulujWizyte()

Wizyta może zostać również anulowana. Skutkuje to zmianą stanu atrybutu wizytaAnulowana.

Aby tego dokonać należy wybrać wizytę, którą chce się anulować i wykonać na niej metodę anulujWizyte(). Poniżej przedstawiono działanie tej metody. Po jednokrotnym wywołaniu tej metody pojawiła się informacja „Wizyta anulowana”, a wartość atrybutu wizytaAnulowana zmieniła się z 0 na 1. Po powtórzeniu tej operacji wyświetlił się komunikat „Nie można anulować wizyty”, ponieważ wizyta została już anulowana.

Tak jak w poprzednich przypadkach sprawdzono poprawność wykonania metody wyświetlając dane zapisane w bazie.



# Repozytorium git

<https://github.com/mi00j/obd-przychodnia.git>

# Filmik z działania aplikacji

# Podsumowanie pracy

## Napotkane problemy i ich rozwiązania

W trakcie tworzenia projektu napotkano kilka problemów.

Pierwszym z nich była konkatenacja wartości typu String. Konstrukcja łączenia ciągów tekstowych w Cache znacząco różni się od znanych już nam konstrukcji w innych językach. Ponadto nie występuje tutaj funkcja dla konkatenacji (jak w SQL) a jedynie podanie do zmiennej ciągu wyrazów. Rozwiązaniem tego problemu było użycie operatora \_.

Kolejnym napotkanym problemem był brak wyświetlenia błędu w przypadku niezapisania obiektu do bazy. Podczas implementacji atrybutów klasy użyty był inny typ danych niż w metodzie, która miała do tego atrybutu przypisać nową wartość. Atrybut dataWizyty posiadał typ Date, natomiast metoda umowWizyte korzystała z LibraryUTC, która wprowadzała obecną datę i czas. Metoda wykonywała się, ale dane nie zapisywane były w bazie i nie wyświetlał się żaden komunikat z tym związany. Rozwiązaniem była oczywiście zamiana typu Date na DateTime.

## Pomysły

Baza obiektowa proponowana przez Cache Intersystems jest dosyć rozbudowanym narzędziem. Oprócz możliwości manualnego implementowania klas, pozwala również na tworzenie ich z wykorzystaniem interfejsu graficznego. Nie przyspiesza to jednak pracy, ponieważ wyklikanie całej klasy wymaga sporej ilości poświęconego czasu. Ponadto interfejs ten nie należy do najnowszych, wygląda nieco przestarzale. Aby usprawnić i uprzyjemnić korzystanie z narzędzia należałoby poprawić interfejs graficzny.

Podobnie jest w przypadku Terminala proponowanego przez Cache. Korzystanie z niego, chociaż jest intuicyjne, jest dość męczące. Aby poprawnie wpisać dane polecenie trzeba dobrze znać instrukcje, ponieważ narzędzie ich nie podpowiada i nie sugeruje co zmienić, aby nie występowały błędy. Dodatkowym minusem jest chociażby fakt, że nie można zmienić rozmiaru okna terminala. W związku z tym proponowanym pomysłem jest ulepszenie Terminala pod względem wypisanych wyżej wad narzędzia.

## Ocena środowiska

|  |  |
| --- | --- |
| Jak środowisko realizuje zagadnienia związane z: | Odpowiedź |
| Obiektowością - Metodami | Metody dodawane są za pomocą wpisanego ręcznie kodu lub opcji Class -> Add -> Method. Zawiera ona nazwę, właściwości, typy przyjmowane i zwracane oraz kod źródłowy. Implementacja jest dość intuicyjna |
| Obiektowością – Dziedziczeniem, typami abstrakcyjnymi | Klasy dziedziczą po innych klasach dzięki dodaniu nazwy klasy bazowej do Extends |
| Obiektowością – Związkami między klasami (asocjacja, kompozycja) | Związki tworzone są przy pomocy wyrażenia Relationship definiując w parametrach Cardinality – typ połączenia (one, many) oraz Inverse – nazwa odwrotnego połączenia |
| Obiektowością – Typy danych – proste, złożone | Możliwe jest wykorzystanie zarówno prostych danych (String, Integer) jak i złożonych (np. zdefiniowanie utworzonej klasy jako złożony typ danych) |
| Obiektowością - Polimorfizmem | Możliwe jest nadpisywanie metod klasy bazowej przez klasy dziedziczące |
| Obiektowością – Tożsamością danych | Każdy obiekt posiada swój OID (object identifier) – unikalny identyfikator obiektu |
| Obiektowością – Enkapsulacją | Enkapsulację można zrealizować wybierając/dodając modyfikator Private |
| Obiektowością – Trwałością danych | Dzięki użyciu opcji Persistent zapewniamy że dane zostaną zapisane w bazie danych |
| Administracja – Zarządzanie środowiskiem | Środowiskiem można zarządzać z poziomu interfejsu lub z poziomu terminala |
| Interfejs – Czy narzędzie posiada API? Dla jakich języków? | Narzędzie posiada API m.in. dla języków Java, .NET, C++ |
| Środowisko – Czy narzędzie zawiera w sobie środowisko programistyczne? | Narzędzie zawiera środowisko programistyczne, jest nim Cache Studio |
| Skalowalność – Czy narzędzie umożliwia horyzontalne skalowanie środowiska? (rozproszone przetwarzanie, magazynowanie, replikacja) | Podstawową technologią jest Enterprise Cache Protocol (ECP) – architektura rozproszonego buforowania danych, która zarządza dystrybucją danych i blokad między heterogeniczną siecią systemów serwerowych.  Dzięki Caché Database Mirroring (Mirroring) możliwe jest tworzenie kopii lustrzanych systemu.  Shadowing natomiast umożliwia komputerom dodatkowym utrzymanie „cienia” kopii wybranych baz danych, ponieważ są one aktualizowane na maszynie podstawowej.  Shadowing i mirroring umożliwiają synchronizację tabel i obiektów. |
| Multi-model – Czy narzędzie zapewnia inne rodzaje bazy danych? | Przechowywanie danych odbywa się również za pomocą tabeli relacyjnych do których ma się dostęp dzięki zapytaniom SQL |