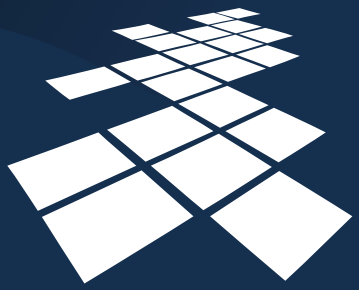
ZSBD – ćwiczenie 5

Obiektowe systemy zarządzania bazą danych. Podstawy pracy z OSZBD db4o.



UCZELNIA
ONLINE

Celem piątego ćwiczenia z Zaawansowanych systemów baz danych jest zapoznanie Państwa z zaletami obiektowych systemów baz danych. W tym celu wykorzystane zostanie przygotowane wcześniej przez Państwa środowisko pozwalające na tworzenie, kompilowanie i uruchamianie programów składujących obiekty w obiektowej bazie danych zarządzanej przez OSZBD db4o. W trakcie ćwiczenia dowiedzą się Państwo jakie architektury aplikacji można zbudować w oparciu o OZSBD db4o, a następnie poznacie państwo interfejsy i metody bibliotek OSZBD db4o pozwalające na wykonywanie podstawowych operacji na obiektach składowanych w bazie danych.

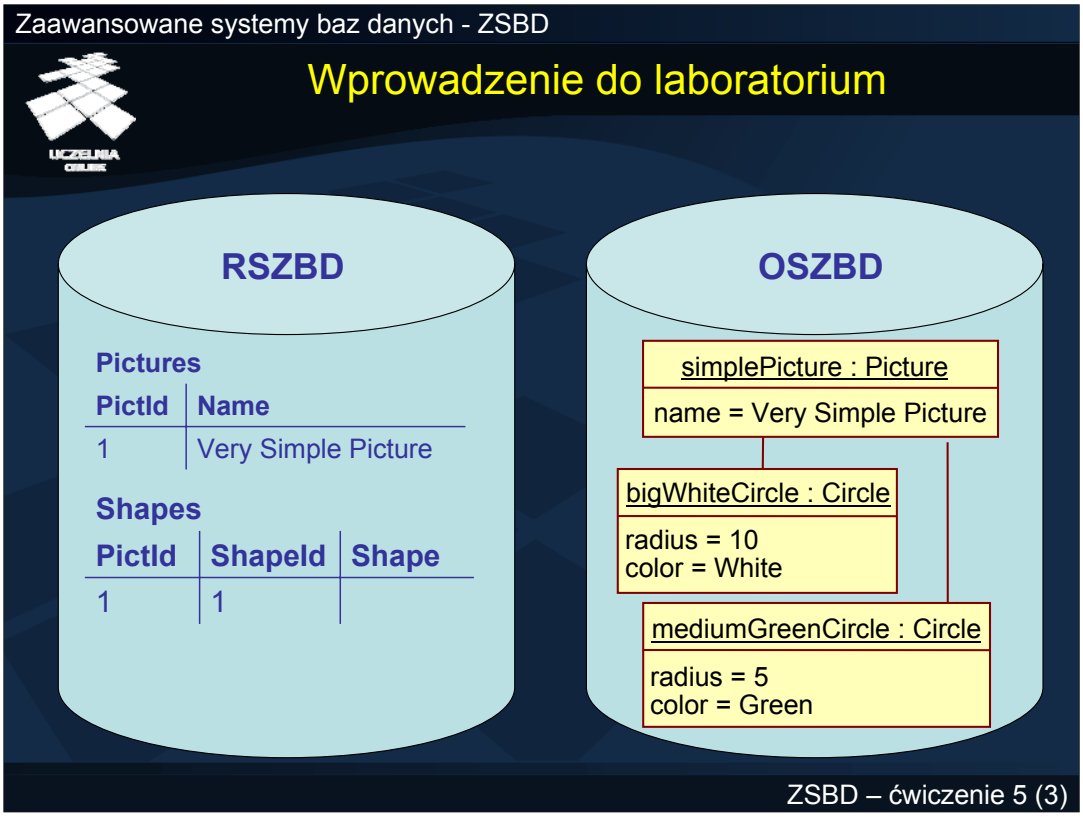
Wymagania:

Do wykonania ćwiczenia konieczna jest podstawowa znajomość środowiska Eclipse (pokazanego na poprzednim ćwiczeniu) i wiedza podstawowa z zakresu programowania w języku Java .

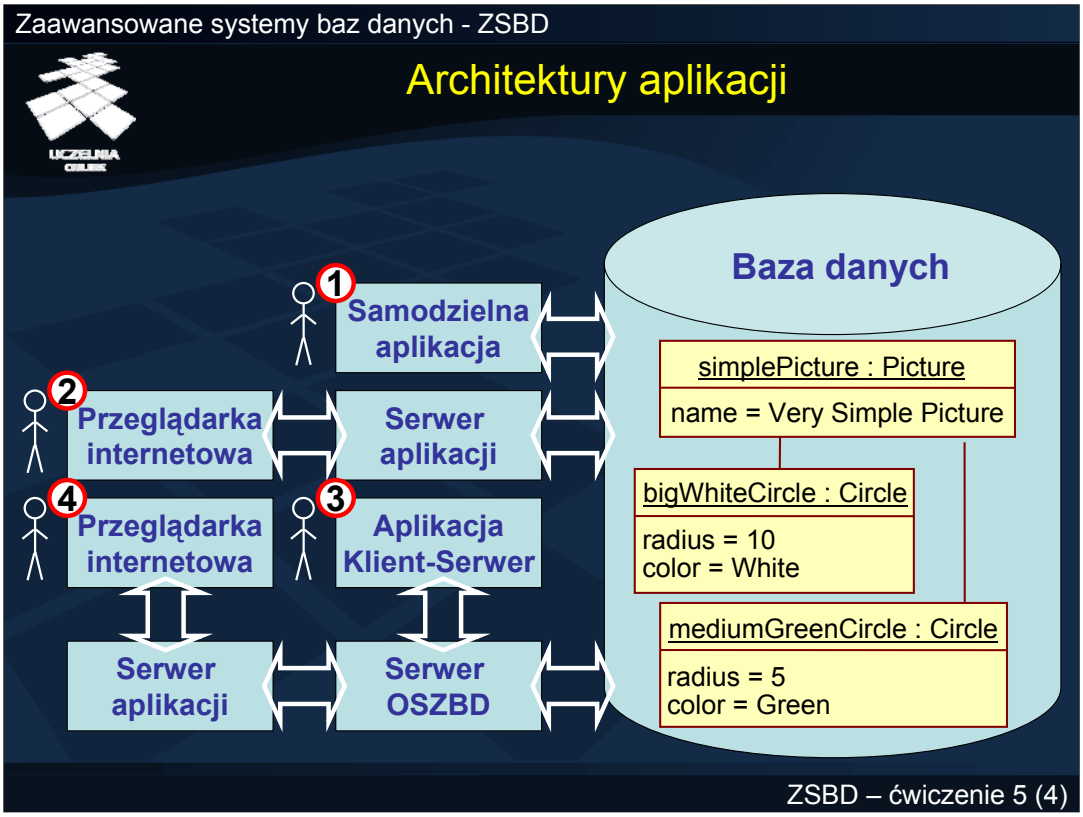
Plan ćwiczenia

- Wprowadzenie do laboratorium.
- Architektury aplikacji.
- Szkielet programu.
- Zapisywanie obiektów do bazy danych.
- Odszukiwanie i odczytywanie obiektów z bazy danych.
- Modyfikacja obiektów w bazie danych.
- Usuwanie obiektów z bazy danych.
- Zadania.
- Podsumowanie

Ćwiczenie rozpocznie się od przedstawienia motywacji stojącej za stworzeniem obiektowych baz danych oraz od przedstawienia podstawowej różnicy pomiędzy obiektowymi systemami zarządzania bazą danych, a relacyjnymi systemami zarządzania bazą danych (wprowadzenie do laboratorium). Następnie, zostaną opisane różne architektury aplikacji, które mogą zostać zbudowane za pomocą bibliotek OSZBD db4o. Po przedstawieniu architektur aplikacji, zostanie omówiony prosty szkielet programu wykorzystującego OSZBD db4o do składowania obiektów w bazie danych. Zapoznacie się tutaj państwo z metodami pozwalającymi na otwieranie i zamykanie dostępu do bazy danych. Podstawowy program będzie następnie rozbudowywany w celu demonstracji sposobów zapisywania, odszukiwania i odczytywania, modyfikacji i usuwania obiektów z bazy danych. Wykonywanie przedstawionych w tych punktach fragmentów programów demonstrujących sposoby wykonywania różnych operacji na obiektach nie jest konieczne, ale silnie zalecane, gdyż poprawi zrozumienie przez Państwa opisywanej w tych punktach tematyki. Na końcu zajęć zostaną przedstawione zadania do samodzielnego wykonania. Ćwiczenie zamknie slajd podsumowujący przedstawiony materiał.



Na zajęciach poświęconych systemom baz danych zapoznaliście się Państwo z relacyjnymi systemami zarządzania bazą danych (RSZBD). Jak zapewne już zauważyliście, pisanie aplikacji składujących dane w relacyjnych bazach danych wymaga implementacji warstw tłumaczących „świat relacyjny” (dane w tabelach) na „świat obiektowy" (wewnętrzne struktury aplikacji) oraz „świat obiektowy” na „świat relacyjny” (np. za pomocą JDBC). Napisane tych warstw potrafi często zająć dużą ilość czasu i wymaga rozwiązania wielu problemów (problem niezgodności reprezentacji danych w tych dwóch światach, określany jest, w języku angielskim, terminem *impedance mismatch*). Obiektowe systemy zarządzania bazą danych są odpowiedzią na ten problem. Pozwalają one na składowanie i odczytywanie obiektów bezpośrednio z bazy danych, bez konieczności transformacji ich do postaci relacyjnej. Rysunek na slajdzie demonstruje różnicę pomiędzy relacyjnymi systemami zarządzania bazą danych (dane składowane w bazach danych w postaci tabel), a obiektowymi systemami zarządzania bazą danych (dane składowane w bazach danych w postaci obiektów).

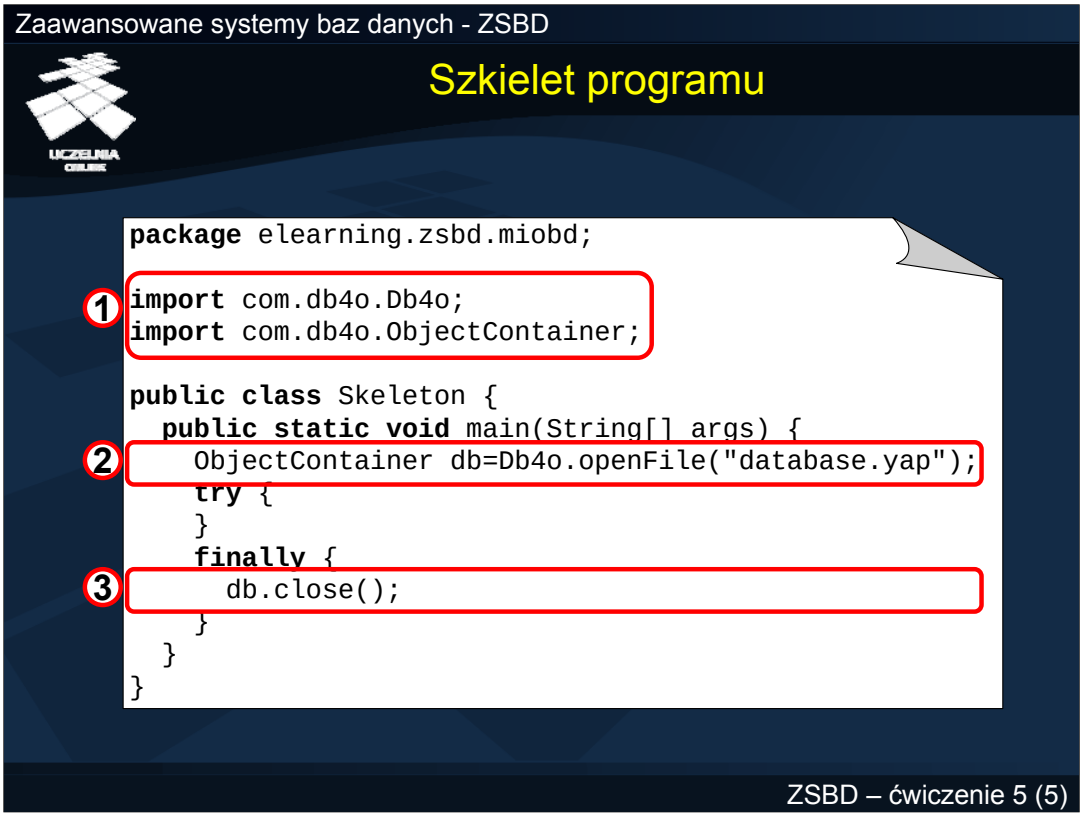


Obiektowy system zarządzania bazą danych db4o jest dostarczany w postaci zbioru bibliotek pozwalających na zbudowanie wszystkich popularnych architektur aplikacji wykorzystujących bazy danych. Możliwa jest konstrukcja samodzielnych aplikacji (1), które wykorzystują biblioteki db4o do tworzenia plików składujących ich własne dane. Taka architektura jest stosowana w sytuacji, w której dane nie są współdzielone z innymi użytkownikami poprzez sieć, a co najwyżej wymieniane jedynie poprzez przekopiowanie pliku. Programami tego typu mogą być programy służące do tworzenia diagramów, edytory tekstu, arkusze kalkulacyjne, gry komputerowe itp. Operowanie na plikach bazy danych odbywa się wówczas bezpośrednio poprzez metody bibliotek db4o i nie jest możliwe współbieżne operowanie na tych danych przez różne aplikacje (w danej chwili może być aktywna tylko jedna transakcja).

Biblioteki db4o zawierają również metody pozwalające na postawienie serwera dostępowego do bazy danych umożliwiającego współbieżne wykonywanie wielu transakcji. Serwer ten może być uruchomiony lokalnie i komunikować się z aplikacją poprzez pamięć operacyjną. Taki serwer pozwala na synchronizację i współbieżną pracę wielu aplikacji i wątków działających wewnątrz jednej maszyny wirtualnej. Możliwe jest zatem utworzenie wielowątkowej aplikacji działającej na tych samych danych, np. serwera aplikacji (2), który posiada prywatną bazę danych, z której nie korzystają inne aplikacje.

Możliwe jest również uruchomienie samodzielnego serwera dostępowego, z którym możliwa jest komunikacja poprzez sieć. Pozwala to na stworzenie aplikacji działających zarówno w architekturze Klient – Serwer (3) jak i architekturze trójwarstwowej (4).

Na ćwiczeniach poświęconych obiektowym systemom zarządzania bazą danych poznacie państwo jedynie najprostszy przypadek, czyli sposób tworzenia samodzielnych aplikacji, które bezpośrednio odwołują się do plików bazy danych. Należy jednak zwrócić uwagę, że tworzenie bardziej zaawansowanych aplikacji nie różni się prawie wcale od tworzenia aplikacji samodzielnych.

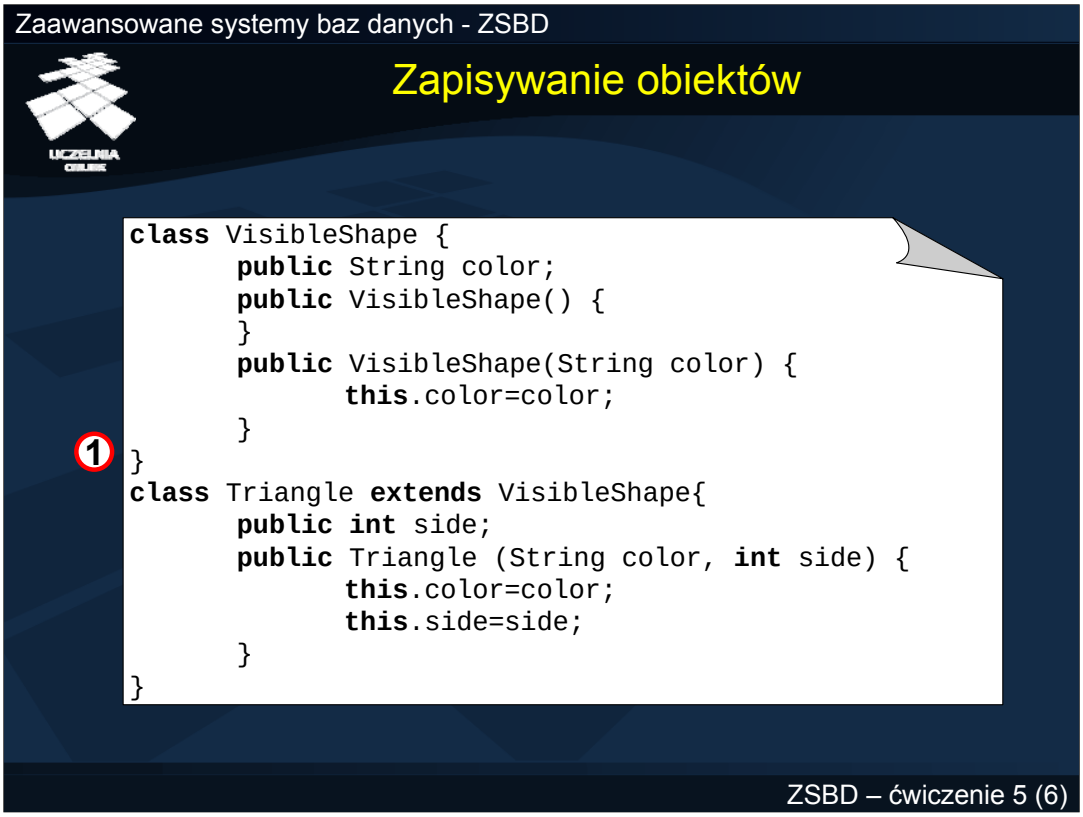


Slajd przedstawia szkielet programu współpracującego z OSZBD db4o. Można w nim wyróżnić trzy ważne fragmenty. Pierwszy fragment (1) dotyczy zaimportowania klas potrzebnych do otwarcia bazy danych. Importowane są tutaj klasa Db4o i interfejs ObjectContainer. Klasa Db4o jest klasą posiadającą statyczne metody służące do tworzenia połączeń z bazami danych oraz stawiania serwera systemu zarządzania bazą danych. Obiekty klas implementujących interfejs ObjectContainer reprezentują połączenia z bazą danych. Metody tego interfejsu pozwalają na zapis, odczyt i usuwanie obiektów z bazy danych oraz na zatwierdzenie bądź wycofanie transakcji w ramach której operacje te są wykonywane.

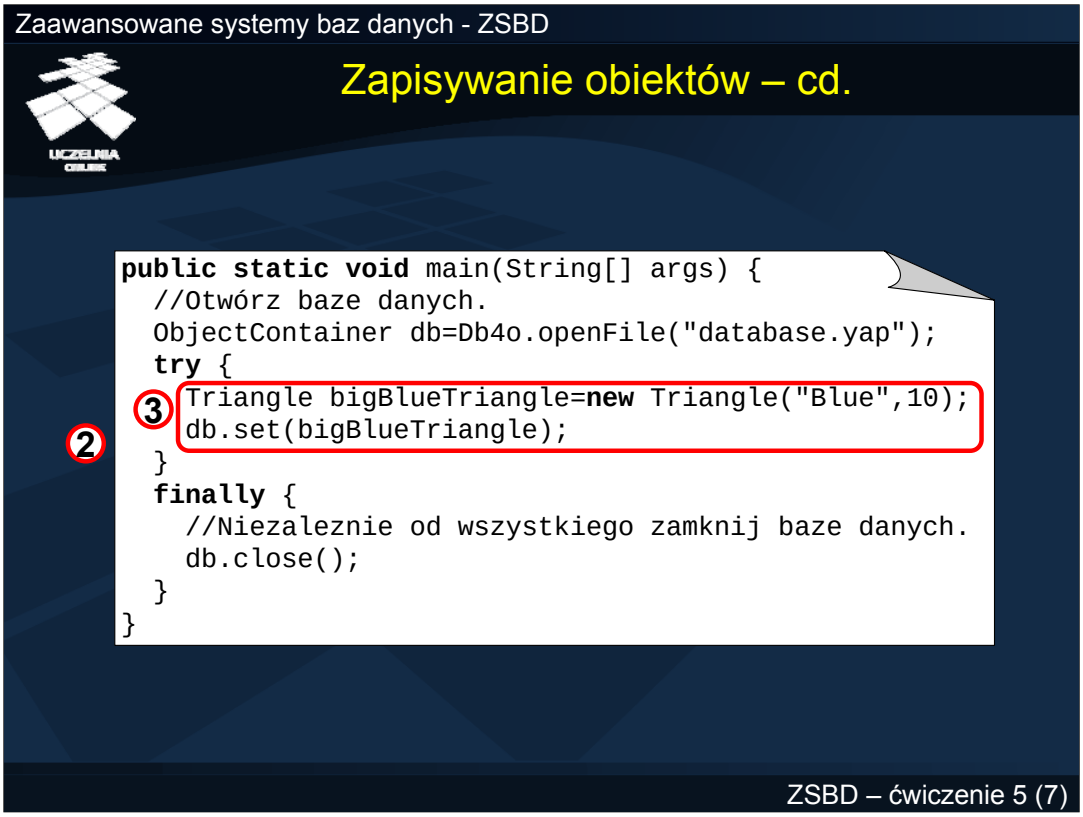
Drugi fragment programu, na który warto zwrócić uwagę (2) to fragment, w którym tworzone jest połączenie z bazą danych. Pierwszą czynnością, jaka musi być wykonana, jest deklaracja zmiennej typu ObjectContainer (tutaj jest to zmienna db). W następnym kroku, zmiennej tej przypisywany jest wynik działania statycznej metody klasy Db4o o nazwie openFile. Metoda openFile służy do otwarcia pliku z danymi bazy danych bezpośrednio, z pominięciem serwera dostępowego. Takie otwarcie pliku jest możliwe jedynie w sytuacji, gdy plik znajduje się w systemie plików maszyny, na której działa również aplikacja. Wielokrotne wywołanie metody openFile na tym samym pliku zwraca ten sam wskaźnik na obiekt reprezentujący połączenie, a zatem jednocześnie możliwe jest wykonywanie tylko jednej transakcji. Metoda openFile blokuje dostęp do pliku bazy danych innym procesom (innym maszynom wirtualnym), gwarantując tym samym zabezpieczenie przed współbieżną, niezsynchronizowaną, modyfikacją pliku bazy danych.

Trzeci fragment (3) pokazuje sposób zamknięcia połączenia z bazą danych. Jak łatwo zauważyć wiąże się on z aktywacją bezparametrowej metody close.

Kompletny kod szkieletu programu, który pokazano na slajdzie załączono do kursu w postaci pliku: Skeleton-lab5.1.java

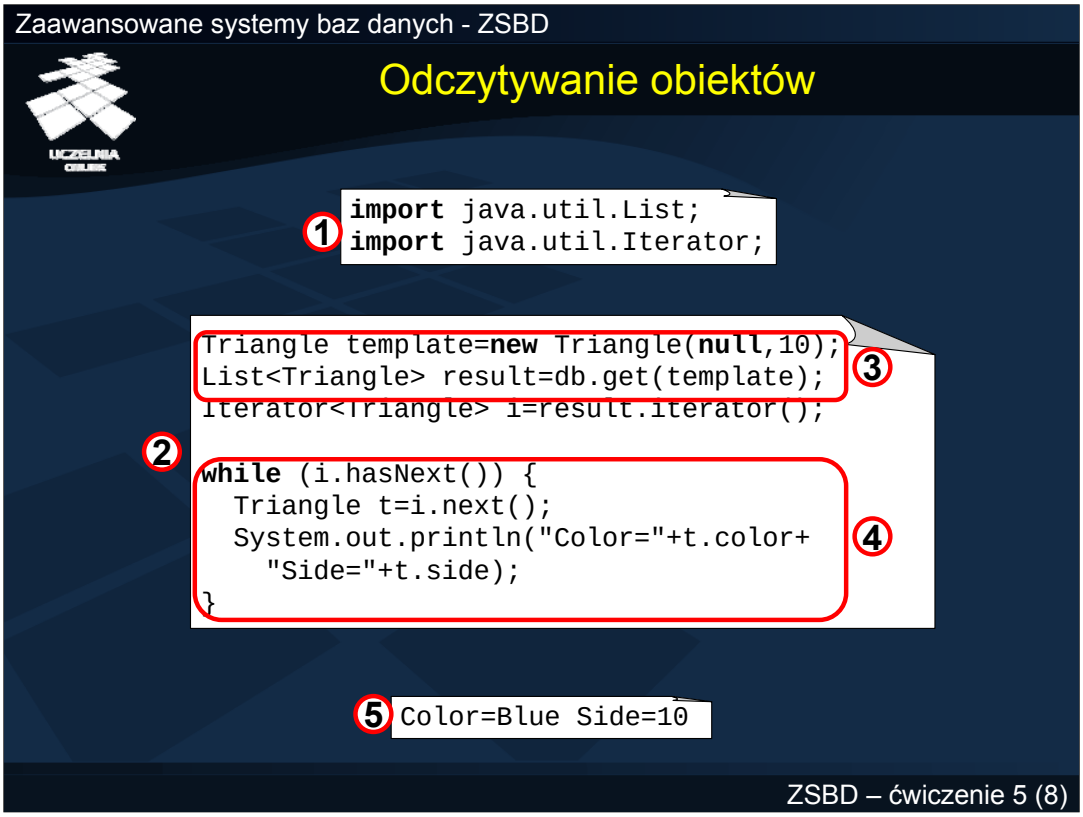


W celu demonstracji sposobu zapisu obiektów do bazy danych należy wprowadzić do szkieletu programu dwie modyfikacje. Pierwsza modyfikacja (1) polega na zdefiniowaniu w pliku szkieletu dwóch prywatnych klas: abstrakcyjnej VisibleShape i Triangle których instancje będą zapisywane do bazy danych. ...



... Druga modyfikacja (2) obejmuje dopisanie dwóch instrukcji we fragmencie programu oznaczonym przez (3), z których pierwsza tworzy nową instancję klasy Triangle, a druga zapisuje go ją bazy danych za pomocą metody set zdefiniowanej w interfejsie ObjectContainer. Jedynym parametrem metody set jest referencja do obiektu, który ma zostać zapisany w bazie danych. Po wykonaniu programu, w bazie danych zostanie zapisany obiekt opisujący niebieski trójkąt o bokach długości 10 jednostek.

Kompletny kod programu, którego fragmenty pokazano na tym i poprzednim slajdzie, załączono do kursu w postaci pliku: Skeleton-lab5.2.java



W celu pokazania jak można odczytywać obiekty z bazy danych, załóżmy, że poprzedni program został uruchomiony i do bazy danych został zapisany obiekt klasy Triangle, którego wartości pól color i side to odpowiednio „Blue” i 10.

Pierwszą czynnością, jaką należy wykonać, przed odczytaniem obiektów z bazy danych, jest zaimportowanie dwóch klas, które będą wykorzystywane (List i Iterator) (1). W celu odczytania obiektów z bazy danych należy wykonać do niej zapytanie. Biblioteki db4o pozwalają na wykonywanie zapytań na trzy sposoby: zapytanie przez przykład (ang. *Query By Example*, QBE), zapytanie natywne (ang. *Native Query*) i za pomocą SODA API. QBE jest najprostszym sposobem wykonywania zapytań, ale posiadającym niestety dużo ograniczeń. Zapytania natywne są głównym sposobem wykonywania zapytań w bazie danych db4o. SODA API jest wewnętrznym interfejsem bibliotek db4o, a jego używanie do wykonywania zapytań jest odradzane. Slajd pokazuje sposób wykonania zapytania za pomocą metody QBE. Zapytania natywne i interfejs SODA API są poza zakresem ćwiczeń.

Drugą modyfikacją poprzedniego programu, jaką należy wykonać, jest zastąpienie fragmentu programu oznaczonego przez (3) na poprzednim slajdzie kawałkiem kodu, który oznaczono na tym slajdzie jako (2). Właściwe zapytanie jest wykonywane za pomocą dwóch instrukcji oznaczonych na slajdzie przez (3). Pierwsza z nich tworzy obiekt klasy Triangle, który jest „przykładem” według którego mają być wyszukiwane obiekty w bazie danych. Druga linijka wywołuje metodę get interfejsu ObjectContainer, która powoduje odszukanie wszystkich obiektów „pasujących” do przykładu. Pasowanie obiektów do przykładu można zdefiniować następująco. Wszystkie pola obiektów, które mają zostać zwrócone w wyniku zapytania, muszą mieć takie same wartości, jak pola obiektu - przykładu, pod warunkiem, że wartość danego pola przykładu nie jest domyślna. Przykładowo, wartością domyślną dla pola typu String jest null, a dla pola typu int jest 0. Ponieważ utworzono przykład, który polu kolor (String) miał przypisane null, a polu side wartość 10, to zapytanie miało odszukać wszystkie obiekty, klasy Trójkąt, których bok miał długość 10 (kolor pomijamy, ponieważ przykład miał w polu kolor wartość domyślną, czyli null). W wyniku realizacji metody get zwracana jest kolekcja obiektów (interfejs List<E>) spełniających warunki zapytania. Zwrócona kolekcja jest przeglądana za pomocą iteratora w pętli (4). Jeżeli przeglądana baza danych jest wynikiem działania programu z poprzednich slajdów, to wynik działania tego programu będzie wyglądał tak, jak to przedstawiono na fragmencie slajdu oznaczonym przez (5).

Należy zwrócić tutaj uwagę na jedną, ważną własność db4o. Otóż, jeżeli w wyniku zapytania zostanie zwrócony obiekt, który został już wcześniej odczytany z bazy danych, to nie zostanie utworzona w pamięci jego kopia, a zapytanie zwróci jedynie referencję na obiekt znajdujący się już w pamięci. Nie ma zatem obaw, że w wyniku wielokrotnego wykonywania zapytań w bazie danych jakieś obiekty zduplikują się.

Poniżej przedstawiono kilka przykładowych zapytań typu QBE:

1. Znajdź wszystkie niebieskie trójkąty (pole side ma wartość domyślną równą 0).

```
Triangle template=new Triangle(„Blue”,0);
List<Triangle> result=db.get(template);
```

2. Znajdź wszystkie trójkąty (zarówno kolor jak i długość boku mają wartość domyślną).

```
Triangle template=new Triangle(null,0);
List<Triangle> result=db.get(template);
```

3. Znajdź wszystkie trójkąty.

```
List<Triangle> result=db.get(Triangle.class);
```

Powyższy przykład wymaga wytłumaczenia. Otóż, metoda get posiada również wersję, w której zamiast obiektu – przykładu, jako parametr podaje się obiekt reprezentujący klasę obiektów, które mają zostać odszukane (obiekt ten można uzyskać np. za pomocą operatora class). Ponieważ tutaj, jako parametr podajemy obiekt reprezentujący klasę Triangle, to wszystkie obiekty klasy trójkąt zostaną odczytane.

4. Znajdź wszystkie kształty.

```
List<VisibleShape> result=db.get(VisibleShape.class);
```

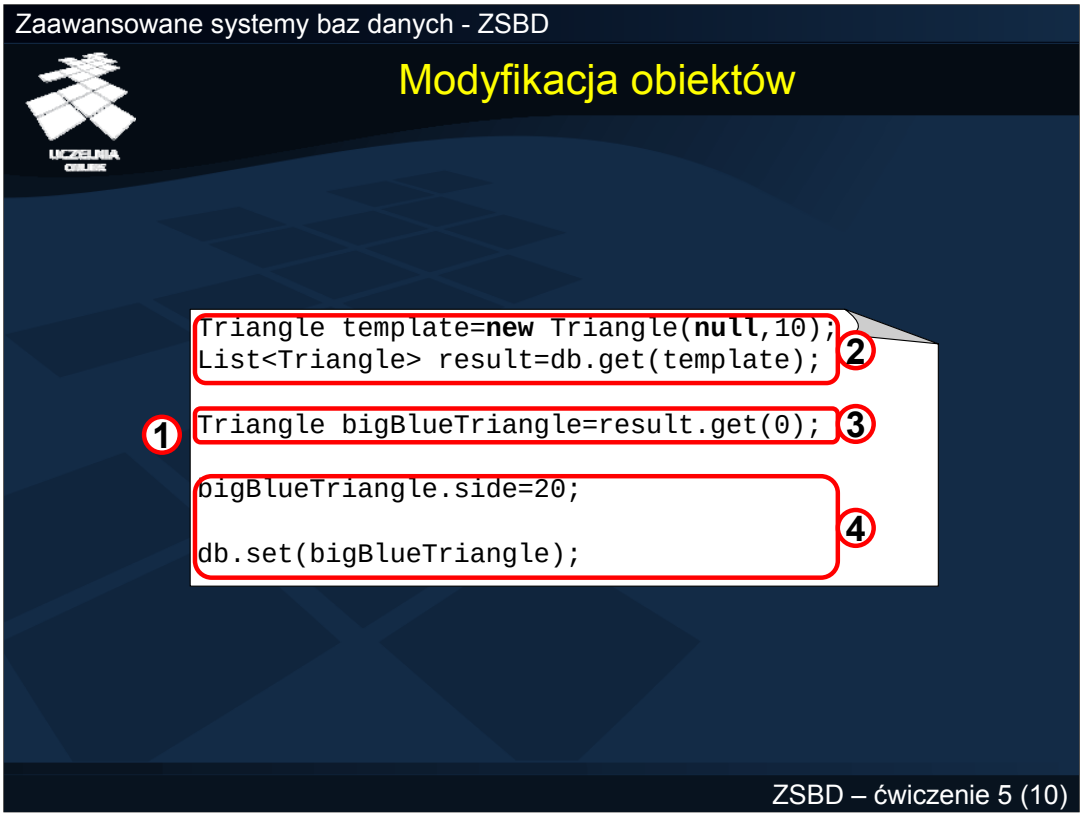
Powyższy przykład jest bardzo podobny do poprzedniego. Zapytaniem tym odszukujemy wszystkie obiekty klasy VisibleShape. Ponieważ klasa Triangle dziedziczy z klasy VisibleShape, to obiekty klasy Triangle również znajdą się w wynikowej kolekcji.

5. Odczytaj wszystkie obiekty.

```
List<Object> result=db.get(null);
```

Jeżeli metodzie get zostanie przekazany parametr null, to odczytywane są wszystkie obiekty z bazy danych. Dlatego też, powyższe zapytanie zwraca wszystkie obiekty zapisane w bazie danych.

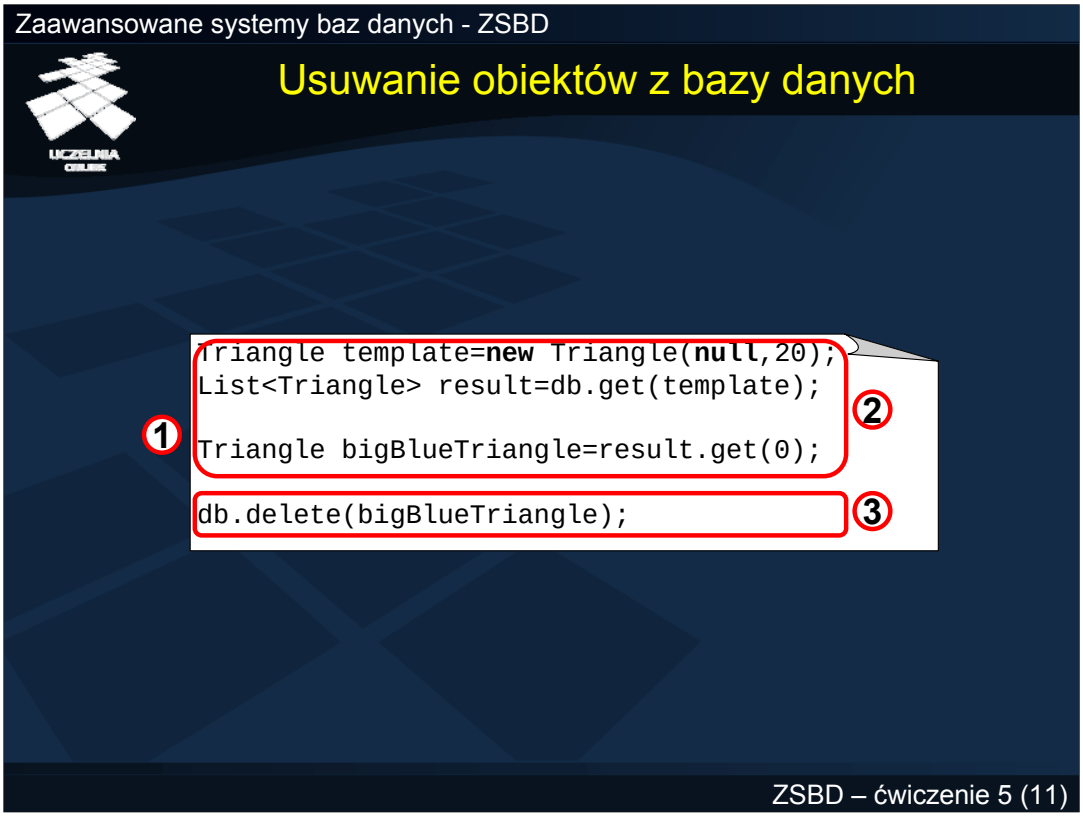
Kompletny kod programu, którego fragmenty pokazano na slajdzie załączono do kursu w postaci pliku: Skeleton-lab5.3.java



Wykorzystamy obecnie poznane dotychczas metody zapisu i odczytu obiektów do przeprowadzenia modyfikacji obiektu zapisanego w bazie danych. W celu pokazania jak można modyfikować obiekty zapisane już w bazie danych ciągle będziemy zakładać, że w bazie danych umieszczony jest obiekt reprezentujący niebieski trójkąt o boku długości 10 jednostek.

Kod przedstawiony na (1) powinien zastąpić kod oznaczony jako (2) na poprzednim slajdzie. W celu przeprowadzenia modyfikacji obiektu należy go najpierw odczytać z bazy danych. W tym celu wykonywane jest zapytanie (2). W kolejnym kroku referencja na obiekt jest wyciągana z kolekcji i zapisywana do zmiennej bigBlueTriangle (3). Ponieważ wiemy, że w bazie znajduje się tylko jeden obiekt reprezentujący trójkąt, to można otrzymać odpowiednią referencję z kolekcji za pomocą wywołania metody get interfejsu List<E> (Uwaga! To jest zupełnie inna metoda, niż metoda get interfejsu ObjectContainer!). Ostatecznie obiekt jest modyfikowany za pomocą standardowej operacji przypisania i zapisywany do bazy danych za pomocą metody set (4). Po wyżej opisanych operacjach obiekt w bazie danych powinien reprezentować niebieski trójkąt o długości boku równej 20 jednostek. Należy zwrócić szczególną uwagę na fakt, iż obiekt musi być najpierw odczytany z bazy danych, a potem zmodyfikowany, i zapisany. Jeżeli, przykładowo, zostałby utworzony nowy obiekt koloru niebieskiego, o długości boku 20 i został zapisany za pomocą metody set, to zostałby on zapisany jako drugi obiekt w bazie danych, a pierwszy obiekt nie uległby modyfikacji. Jeżeli chodzi o sposoby modyfikacji obiektu odczytanego z bazy danych, to można stosować dowolne sposoby dostępne w języku Java: poprzez przypisanie, aktywację metody zmieniającej stan obiektu, jak również wykorzystanie dowolnych innych operatorów zmieniających stan pól obiektu.

Kompletny kod programu, którego fragmenty pokazano na slajdzie załączono do kursu w postaci pliku: Skeleton-lab5.4.java



Usuwanie obiektów jest bardzo podobne do modyfikacji. W celu pokazania sposobu usuwania obiektów z bazy danych zakładamy, że w bazie danych znajduje się obiekt reprezentujący niebieski trójkąt o długości boku równej 20 (wynik działania programu z poprzedniego slajdu).

Fragment programu pokazany na slajdzie (1) powinien zastąpić fragment programu, który został oznaczony przez (1) na poprzednim slajdzie. Usunięcie obiektu z bazy danych odbywa się w dwóch krokach. W pierwszym kroku obiekt powinien zostać odczytany z bazy danych. Odpowiedzialny za to kod oznaczono na slajdzie przez (2). Warto zwrócić uwagę, że tutaj obiekt jest odczytywany w taki sam sposób jak przy modyfikacji, z tą jednak różnicą, że zapytanie odczytuje wszystkie obiekty reprezentujące trójkąt o długości boku 20 a nie 10 jak poprzednio, gdyż obiekt ten został zmodyfikowany za pomocą programu przedstawionego na poprzednim slajdzie. Kiedy uzyskamy referencję na obiekt, który chcemy usunąć, należy aktywować metodę delete interfejsu ObjectContainer przekazując jako parametr aktualny tej metodzie referencję na usuwany obiekt. W rezultacie działania metody delete obiekt jest usuwany z bazy danych.

Kompletny kod programu, którego fragmenty pokazano na slajdzie załączono do kursu w postaci pliku: Skeleton-lab5.5.java

Zadanie (1)

- Do zadania wykorzystaj szkielet programu i klasy zdefiniowane na wcześniejszych slajdach.

A. Zdefiniuj dwie dodatkowe klasy reprezentujące figury geometryczne: Rectangle i Circle wzorując się na klasie Triangle. Obie te klasy powinny dziedziczyć z klasy VisibleShape i posiadać pola:
   - Rectangle side1 i side2 typu *int*
   - Circle radius typu *int*

Zadanie (1) – cd.

B. Zapisz do bazy danych następujące obiekty:
   - Czerwony trójkąt o boku 5
   - Czerwone kółko o promieniu 4
   - Zielony prostokąt o bokach 6 i 7
   - Niebieskie kółko o promieniu 4
   - Czerwony trójkąt o boku 10

Zadanie (2)

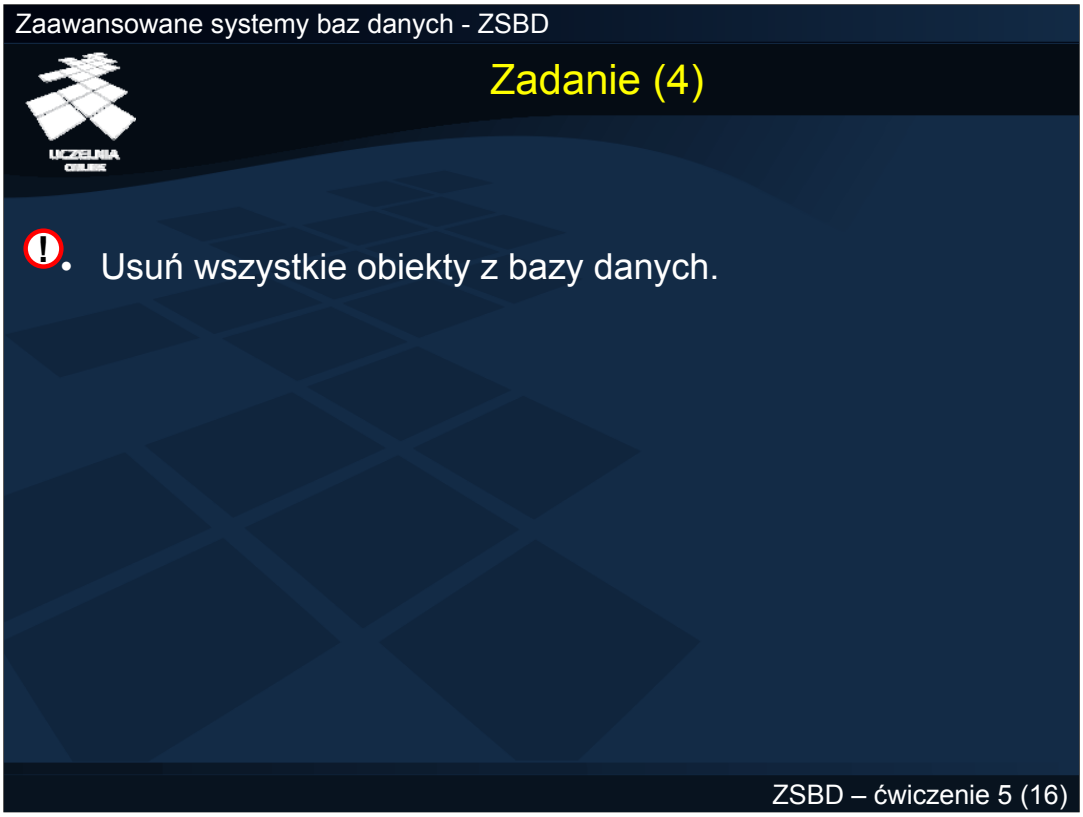
- Wykorzystując zapytania typu QBE odnajdź w bazie danych, utworzonej w poprzednim zadaniu, następujące obiekty:
  A. Wszystkie kółka o promieniu 4
  B. Wszystkie czerwone trójkąty
  C. Wszystkie czerwone figury

(!) Wykorzystaj fakt, że pole *color* jest polem klasy VisibleShape. Wykorzystaj operator instanceof aby zapewnić odpowiednie wyświetlanie wyników w zależności od klasy zwróconego obiektu, bądź przesłoń w każdej z klas reprezentujących figury metodę toString().

Zadanie (3)

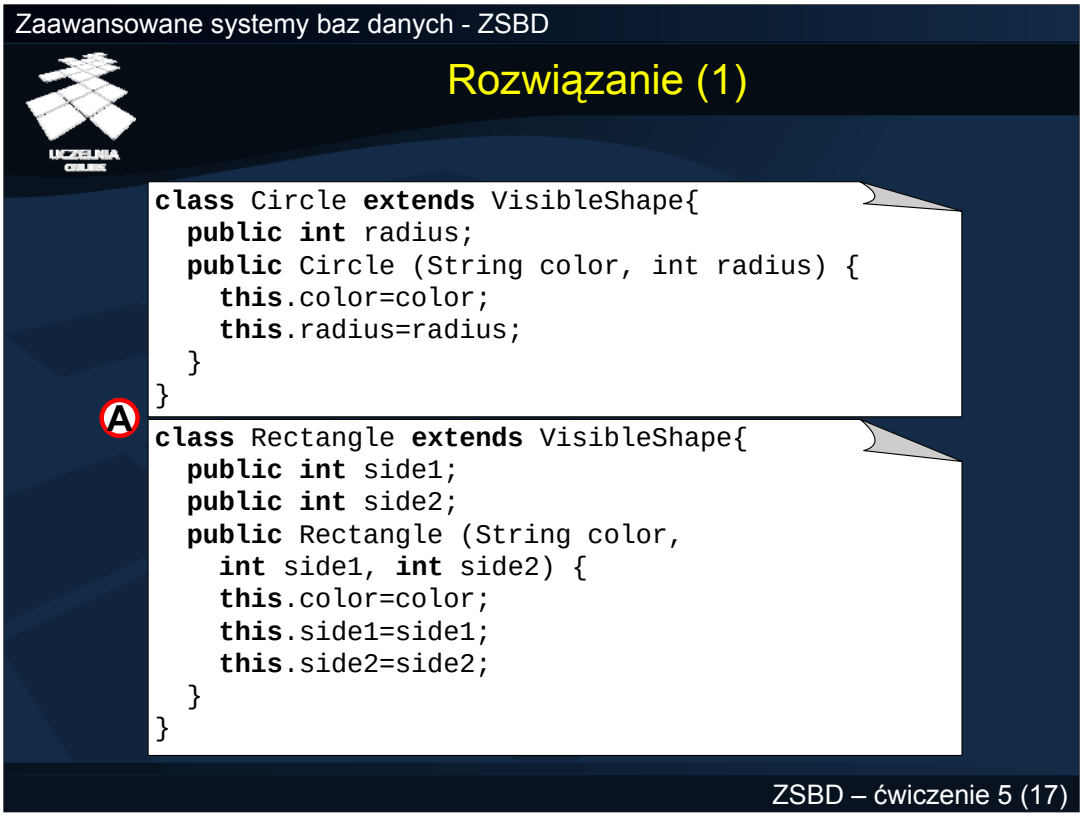
A. Zmień promień niebieskiego kółka w bazie danych na 5 i zapisz zmiany.
B. Odczytaj zmodyfikowany obiekt, żeby sprawdzić czy modyfikacja została zapisana.

(!) Modyfikację i odczyt obiektu należy wykonać w dwóch osobnych uruchomieniach programu, aby zmodyfikowany obiekt rzeczywiście został odczytany z bazy danych.



(!) Wykorzystaj możliwość odczytania wszystkich obiektów z bazy danych za pomocą metody get interfejsu ObjectConstraints.

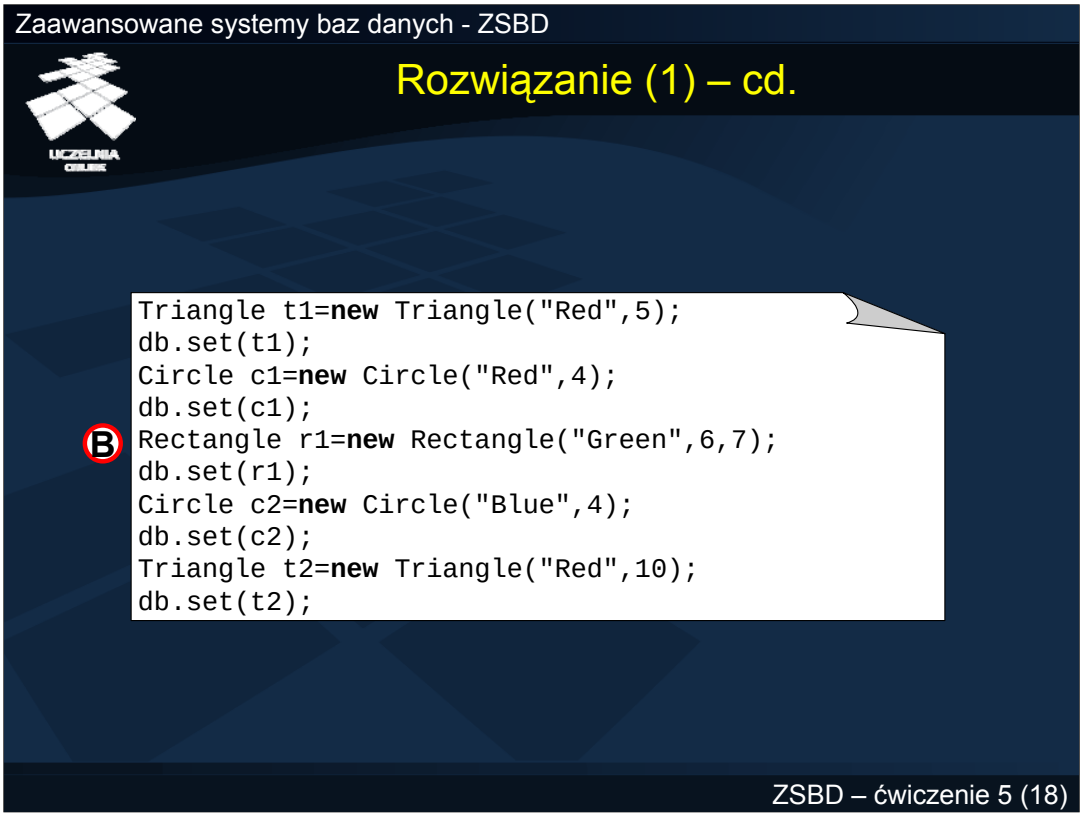
Kiedy wykonasz zadania możesz je porównać z rozwiązaniami przedstawionymi na kolejnych slajdach.



Slajd pokazuje rozwiązanie zadania (1A), którego treść przytoczono poniżej. Ciąg dalszy rozwiązania zadania (1) jest przedstawiony na kolejnym slajdzie.

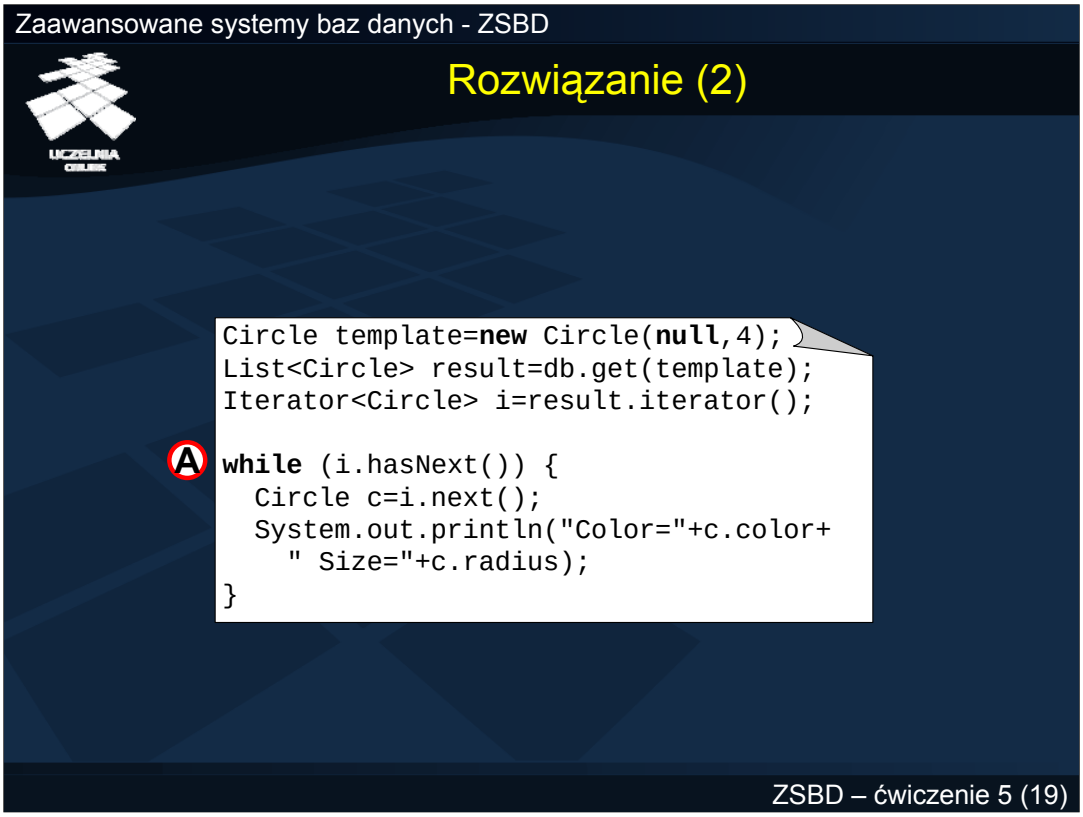
(A) Zdefiniuj dwie dodatkowe klasy reprezentujące figury geometryczne: Rectangle i Circle wzorując się na klasie Triangle. Obie te klasy powinny dziedziczyć z klasy VisibleShape i posiadać pola:

- Rectangle side1 i side2 typu int
- Circle radius typu int



Slajd pokazuje rozwiązanie zadania (1B), którego treść przytoczono poniżej.

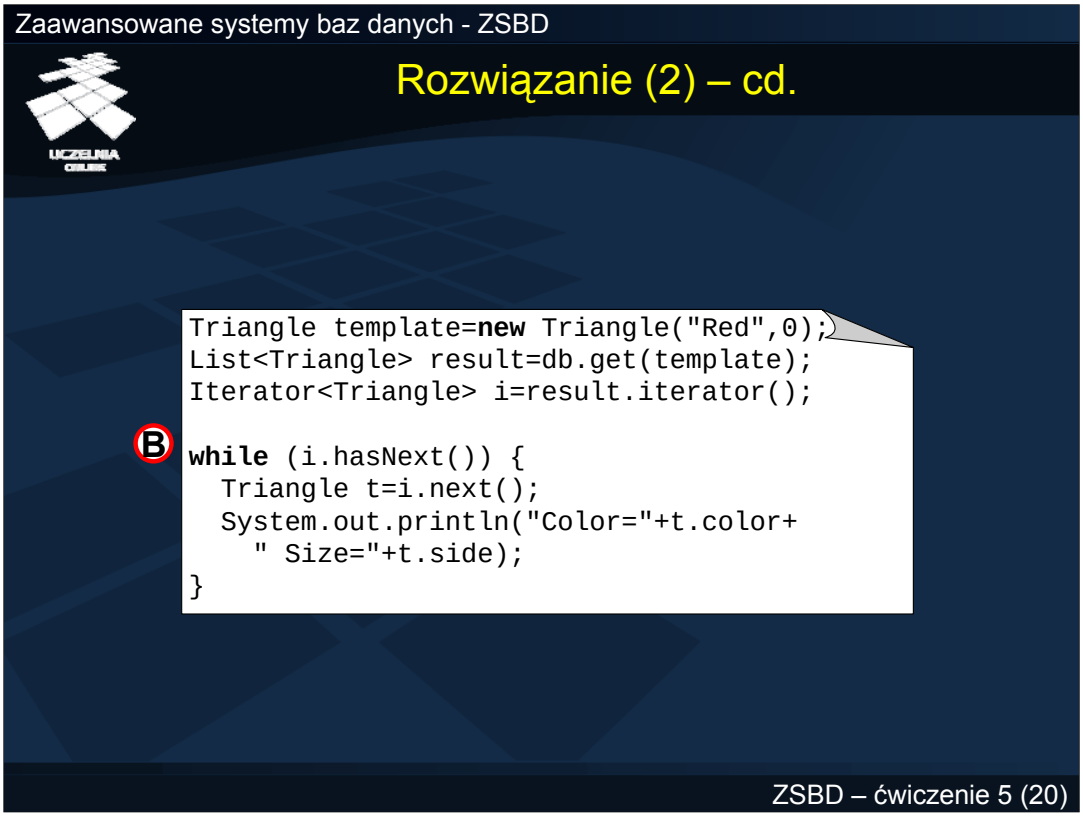
(B) Zapisz do bazy danych następujące obiekty:
- Czerwony trójkąt o boku 5
- Czerwone kółko o promieniu 4
- Zielony prostokąt o bokach 6 i 7
- Niebieskie kółko o promieniu 4
- Czerwony trójkąt o boku 10



Slajd pokazuje rozwiązanie zadania (2A), którego treść przytoczono poniżej.
Rozwiązanie jest kontynuowane na kolejnym slajdzie.

Wykorzystując zapytania typu QBE odnajdź następujące obiekty:

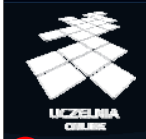
- (A) Wszystkie kółka o promieniu 4



Slajd pokazuje rozwiązanie zadania (2B), którego treść przytoczono poniżej.
Rozwiązanie jest kontynuowane na kolejnym slajdzie.

Wykorzystując zapytania typu QBE odnajdź następujące obiekty:

- (B) Wszystkie czerwone trójkąty



Rozwiązanie (2) – cd.

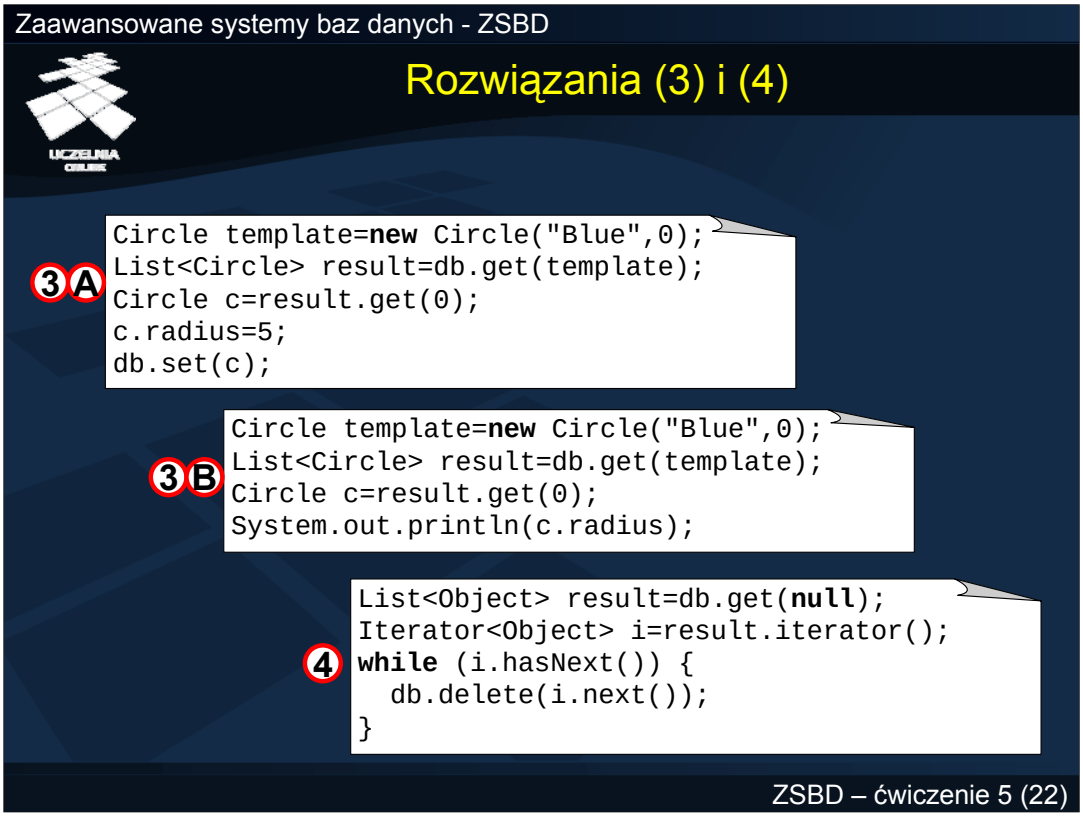
C

```
VisibleShape template=new VisibleShape("Red");
List<VisibleShape> result=db.get(template);
Iterator<VisibleShape> i=result.iterator();
while (i.hasNext()) {
  VisibleShape s=i.next();
  if (s instanceof Triangle) {
    System.out.println("(Triangle)Color="+((Triangle)s).color+
      "Side="+((Triangle)s).side);}
  if (s instanceof Circle) {
    System.out.println("(Circle)Color="+((Circle)s).color+
      "Radius="+((Circle)s).radius);}
  if (s instanceof Rectangle) {
    System.out.println("(Rectangle)Color="+((Rectangle)s).color+
      " Side1="+((Rectangle)s).side1+
      " Side2="+((Rectangle)s).side2);}
}
```


Slajd pokazuje rozwiązanie zadania (2C), którego treść przytoczono poniżej.

Wykorzystując zapytania typu QBE odnajdź następujące obiekty:

- (C) Wszystkie czerwone figury



Slajd pokazuje rozwiązania zadań (3) i (4), których treść przytoczono poniżej.

- (3)(A) Zmień promień niebieskiego kółka w bazie danych na 5 i zapisz zmiany.
- (3)(B) Odczytaj zmodyfikowany obiekt, żeby sprawdzić czy modyfikacja została zapisana.
- (4) Usuń wszystkie obiekty z bazy danych.

Podsumowanie

- W trakcie zajęć poznaliście Państwo podstawy pracy w obiektowym systemie zarządzania bazą danych db4o.
- Dowiedzieliście się jakie architektury aplikacji można stworzyć za pomocą obiektowego systemu zarządzania bazą danych db4o.
- Poznaliście sposoby zapisywania, wyszukiwania, odczytywania, modyfikowania i usuwania pojedynczych obiektów z bazy danych.