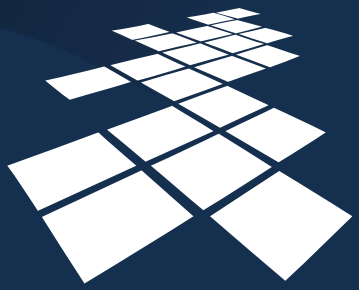
ZSBD – ćwiczenie 6

Obiektowe systemy zarządzania bazą danych. Praca ze złożonymi strukturami danych w OSZBD db4o.



UCZELNIA
ONLINE

Dotychczasowa praca z obiektowym systemem zarządzania bazą danych db4o nie różniła się specjalnie od pracy z relacyjnym systemem zarządzania bazą danych. W wyniku wykonania zapytania, otrzymywaliśmy obiekty jednego typu, które spełniały warunki zapytania. Obiekty te nie różniły się również w szczególny sposób od krotek. Celem obecnych zajęć jest demonstracja możliwości obiektowych systemów zarządzania bazą danych, w sytuacji kiedy aplikacja korzysta z bardziej złożonych, dynamicznych struktur danych. W ramach ćwiczenia poznacie państwo sposoby zapisu, wyszukiwania, odczytu, modyfikacji i usuwania złożonych struktur danych. Dowiecie się również jakie problemy są z tym związane i jak je rozwiązywać.

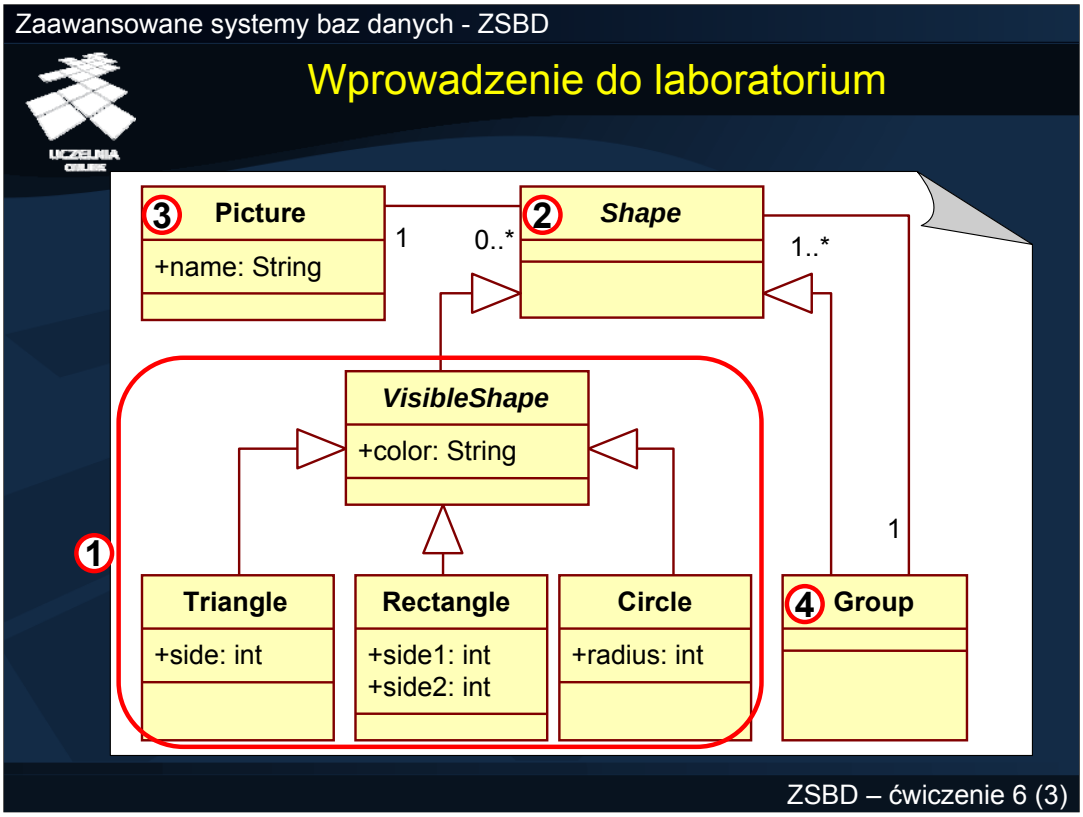
Wymagania:

Do wykonania ćwiczenia konieczna jest podstawowa znajomość środowiska Eclipse (pokazanego na poprzednich ćwiczeniach) i wiedza podstawowa z zakresu programowania w języku Java, oraz wykonanie piątego ćwiczenia z ZSBD.

Plan ćwiczenia

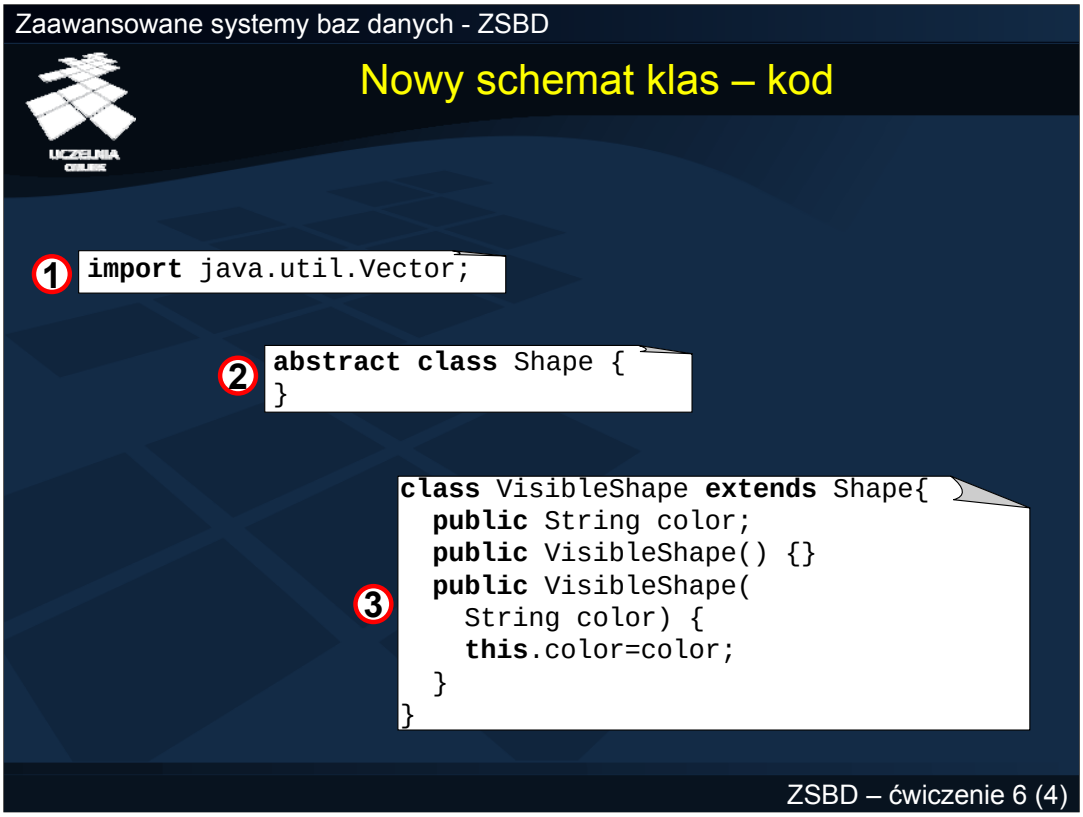
- Wprowadzenie do laboratorium i nowy schemat klas.
- Przykładowa złożona struktura danych.
- Zapisywanie złożonych struktur danych do baz danych.
- Odczytywanie złożonych struktur danych z baz danych.
- Modyfikacja złożonych struktur danych w bazie danych.
- Usuwanie złożonych struktur danych z bazy danych.
- Zadania.
- Podsumowanie.

Ćwiczenie zostanie rozpoczęte od przedstawienia nowego schematu klas, który będzie wykorzystywany w przy omawianiu nowej tematyki. Dla nowego schematu zostanie również przedstawiona przykładowa struktura danych, która będzie wykorzystana demonstracji mechanizmów obsługi złożonych struktur danych w OSZBD db4o. Następnie zostaną opisane mechanizmy db4o pozwalające na zapisywanie, odszukiwanie i odczytywanie, modyfikację, i usuwanie złożonych struktur danych. Nie jest konieczne wykonywanie fragmentów programów demonstrujących poszczególne mechanizmy, które zostaną przedstawione na slajdach. Jest to jednak silnie zalecane, gdyż umożliwi państwu lepsze zrozumienie omawianej tematyki. Na końcu zajęć zostaną przedstawione zadania do samodzielnego wykonania. Ćwiczenie zakończymy slajdem podsumowującym przedstawiony materiał.

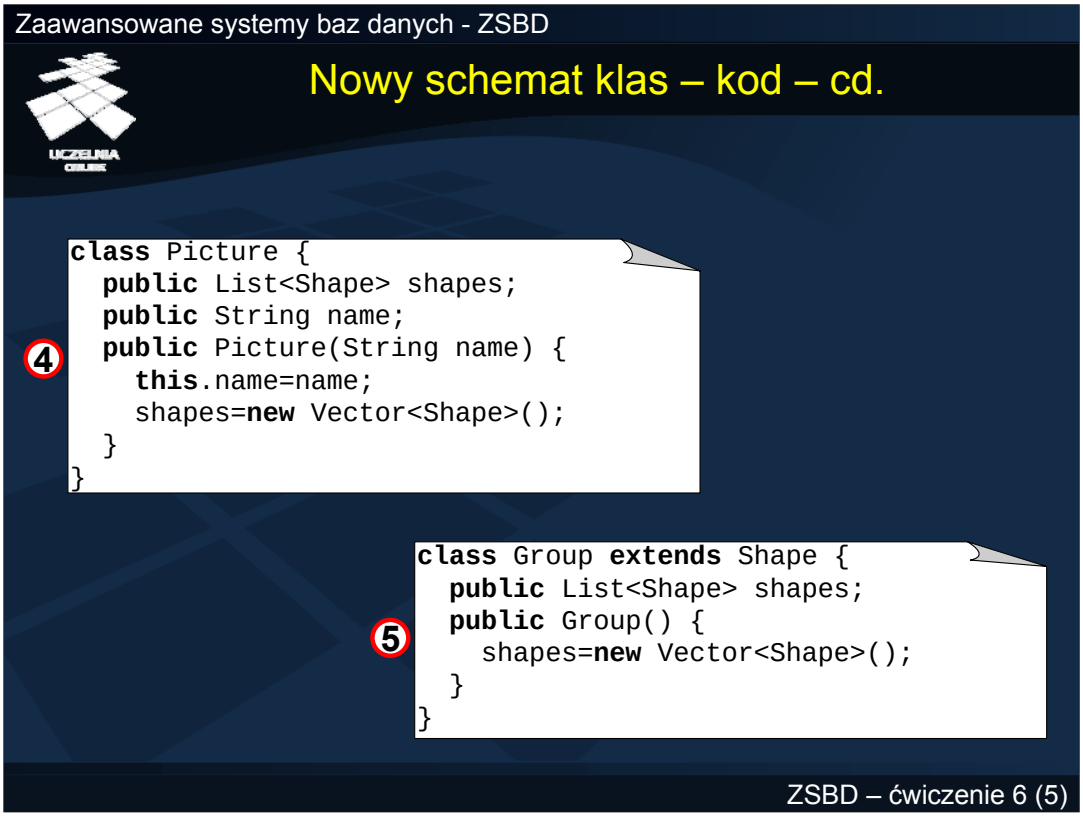


Tym co stanowi o sile obiektowych systemów zarządzania bazą danych jest nie tylko możliwość zapisywania pojedynczych obiektów do bazy danych, ale również możliwość łatwego i szybkiego zapisywania całych, złożonych struktur danych zbudowanych z obiektów powiązanych ze sobą powiązaniami referencyjnymi, oraz ich późniejszego odczytywania, modyfikacji i usuwania. Taką funkcjonalność posiada również OSZBD db4o.

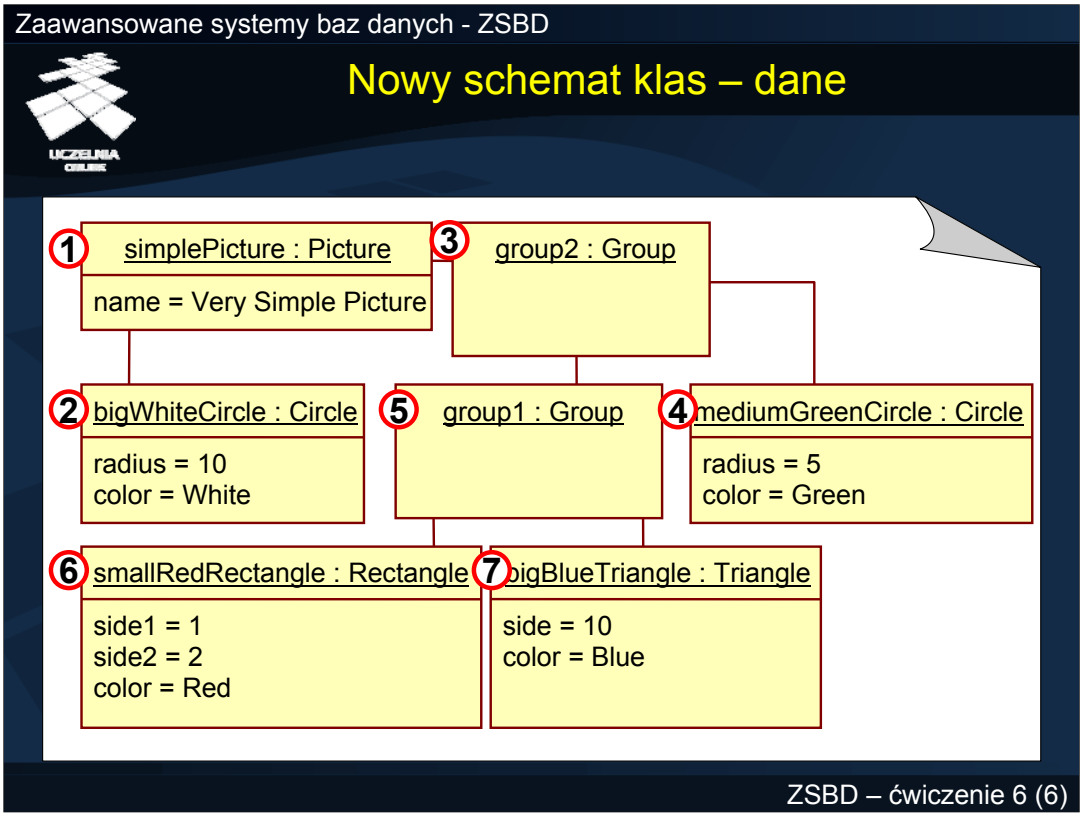
W celu demonstracji sposobu obsługi złożonych struktur danych przez OSZBD db4o, należy wprowadzić kilka zmian do schematu klas, który powstał w czasie wykonywania poprzedniego ćwiczenia. W poprzednim ćwiczeniu zostały zaimplementowane klasy: VisibleShape, Triangle, Rectangle i Circle (1). Pierwszą modyfikacją, jaką należy wprowadzić, jest implementacja abstrakcyjnej klasy Shape (2), z której powinna dziedziczyć klasa VisibleShape. W kolejnym kroku należy zaimplementować klasę Picture, która posiada pole name określające nazwę rysunku, i kolekcję referencji na obiekty klasy Shape. Klasa Picture reprezentuje zatem zbiór figur geometrycznych tworzących rysunek. Ostatnią klasą, jaką należy dodać, jest klasa Group, która dziedziczy z klasy Shape. Klasa Group powinna również mieć pole, które jest kolekcją referencji na obiekty klasy Shape. Klasa ta reprezentuje zatem grupę figur geometrycznych tworzących razem pewną logiczną całość. Ponieważ klasa Group dziedziczy z klasy Shape, to możliwe jest tworzenie rekursywnych struktur danych, w których grupy figur znajdują się w innych grupach.



Na tym i kolejnym slajdzie przedstawiono implementacje klas, które będą potrzebne do pokazania sposobu zapisywania i odczytywania złożonych struktur danych do bazy danych db4o. Ponieważ będziemy wykorzystywać kolekcje (klasy Picture i Group), musimy zaimportować jedną z klas, która reprezentuje taką kolekcje (1). Pierwszą klasą jaką należy zaimplementować, podczas wprowadzania zmian, jest abstrakcyjna klasa Shape (2). Z klasy tej dziedziczą wszystkie klasy, które reprezentują figury na rysunku. Należy również wprowadzić modyfikację do klasy VisibleShape polegającą na uczynieniu klasy Shape nadklasą klasy VisibleShape (3). ...

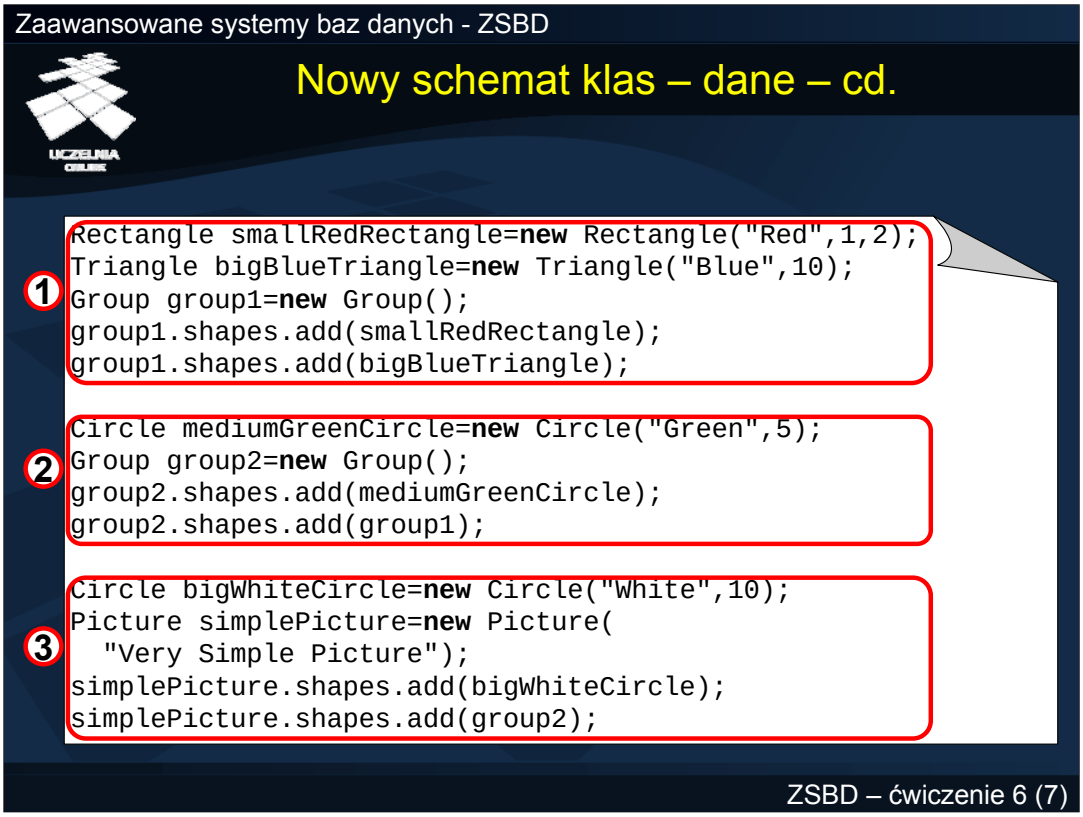


... Następnie, należy zaimplementować klasę Picture, która reprezentuje pojedynczy rysunek w bazie danych i przechowuje kolekcję referencji na obiekty klasy Shape (4). Ostatecznie, implementowana jest klasa Group, która służy do logicznego grupowania figur na rysunku (5).



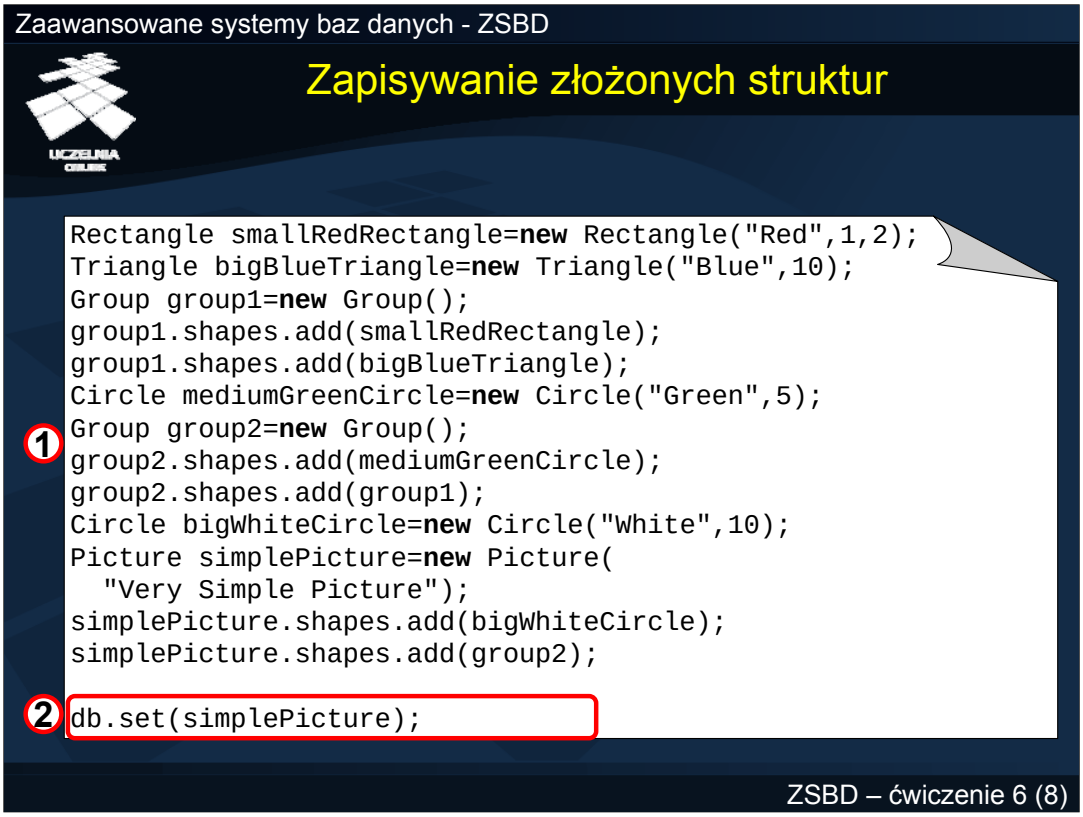
Na slajdzie przedstawiono przykładową strukturę danych, którą można zbudować wykorzystując klasy zdefiniowane na poprzednich slajdach. Będzie stanowić ona punkt wyjściowy do dalszego omawiania materiału. Struktura zawiera obiekty składające się na jeden rysunek o nazwie ”Very Simple Picture” (1). Na rysunek składa się obiekt reprezentujący białe kółko (2), oraz grupa figur (3) zawierająca dwa obiekty: zielone kółko (4) i grupę (5) z dwoma figurami: czerwonym prostokątem (6) i niebieskim trójkątem (7).

Kod tworzący taką strukturę został przedstawiony na kolejnym slajdzie.



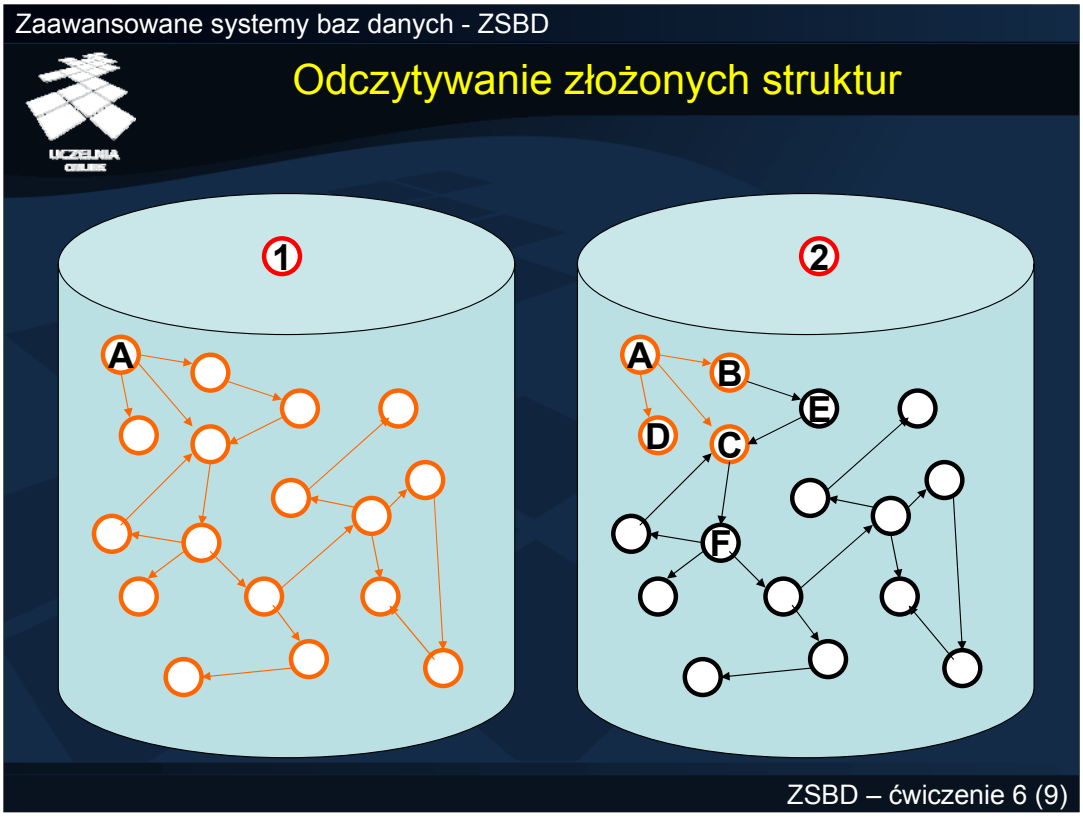
Program pokazany na tym slajdzie tworzy w pamięci strukturę danych pokazaną w postaci diagramu na poprzednim slajdzie. Najpierw tworzone są czerwony prostokąt i niebieski trójkąt i zapisywane do pierwszej grupy figur (1). Następnie tworzone jest zielone kółko, które, wraz z grupą pierwszą, jest zapisywane do grupy drugiej (2). Ostatecznie tworzone jest białe kółko i wraz z drugą grupą figur jest zapisywane do obiektu reprezentującego cały rysunek (3).

Kompletny kod programu, którego fragmenty pokazano na poprzednich i obecnym slajdzie załączono do kursu w postaci pliku: Skeleton-lab6.1.java



Na slajdzie pokazano kawałek kodu tworzący przykładowy rysunek w pamięci, opisany na poprzednim slajdzie (1). Został on uzupełniony tylko o jedną linijkę (2), którą jest aktywacja metody set interfejsu ObjectConstraints, której jako parametr przekazano referencję na obiekt simplePicture. To jedno wywołanie wystarczy, aby zapisać cały rysunek do bazy danych. Metoda set zapisuje obiekt przekazany jako parametr i sprawdza, czy pola zapisywanego obiektu wskazują na jakieś niezapisane obiekty. Jeżeli tak jest, to wszystkie niezapisane obiekty są również zapisywane w bazie danych. Procedura jest wywoływana rekursywnie tak długo, jak ciągle napotykane są jakieś obiekty do zapisania. Jak łatwo zauważyć, przykładowy rysunek ma w pamięci strukturę drzewa, a zatem wystarczy zapisać jego korzeń (obiekt klasy Picture), żeby cała struktura została zapisana do bazy danych.

Kompletny kod programu, którego fragmenty pokazano na slajdzie załączono do kursu w postaci pliku: Skeleton-lab6.2.java



Odczytanie obiektu z bazy danych może wymagać odczytania również wszystkich obiektów osiągalnych poprzez referencje z odczytywanego obiektu. Odczytanie tychże obiektów może prowadzić do odczytania kolejnych obiektów. W skrajnym przypadku, odczytanie jednego obiektu może prowadzić do wczytania do pamięci komputera całej bazy danych. Przykładowo, obiekt A na przykładzie (1) jest powiązany bezpośrednio, lub pośrednio z każdym obiektem w bazie danych. W związku z tym, jeżeli obiekt A zostałby odczytany, to wszystkie inne obiekty w bazie danych musiałyby również zostać odczytane. Taki efekt jest najczęściej niepożądany i dlatego db4o wprowadza kilka mechanizmów pozwalających na ograniczenie tego problemu. Podstawowym rozwiązaniem problemu jest ustalenie liczby kolejnych powiązań referencyjnych wzdłuż których db4o będzie nawigować w celu odczytywania obiektów powiązanych z obiektem odczytanym w wyniku realizacji zapytania, przy czym pierwszym powiązaniem referencyjnym jest powiązanie do odczytywanego obiektu. Jest to tzw. głębokość aktywacji. Przykładowo, dla głębokości aktywacji równej dwa, przy odczytywaniu obiektu A (przykład (2)) odczytane zostaną również obiekty osiągalne z obiektu A wzdłuż jednego powiązania referencyjnego (pierwsze powiązanie do obiektu A i drugie powiązanie do kolejnych). Na przykładzie (2) są to obiekty B, C i D. Obiekty, które zostały odczytane z bazy danych i znajdują się pamięci, nazywa się obiektami aktywnymi. Obiekty aktywne zaznaczono na rysunku kolorem pomarańczowym. Pozostałe obiekty, które nie zostały odczytane z bazy danych (zaznaczone na czarno) nazywa się obiektami nieaktywnymi. Niech zmienna refA zawiera referencję na obiekt A odczytany przy poziomie aktywacji 2. Niech obiekt A zawiera pola, które stanowią referencje na obiekty B, C i D o nazwach refB, refC i refD odpowiednio. Niech obiekt B posiada pole stanowiące referencję na obiekt E o nazwie refE, a obiekt C posiada pole stanowiące referencję na obiekt F o nazwie refF. Przy tych założeniach, następujące wyrażenia ścieżkowe są poprawne:

```
refA.refB.refE
refA.refC.refF
refA.refD
```

Nawigowanie poza obiekty E i F nie jest możliwe, gdyż obiekty te nie są aktywne. Jeżeli obiekt jest nieaktywny, to wszystkie jego pola zawierają wartości domyślne (null, 0, false). W przypadku referencji jest to wartość null. Próba nawigacji poprzez referencję o wartości null zakończyłaby się wyjątkiem NullPointerException.

Jeżeli zajdzie konieczność odczytania nieaktywnego obiektu, na którego jest znana referencja (np. refF), Możliwa jest aktywacja takiego obiektu za pomocą metody activate interfejsu ObjectContainer. Metoda ta przyjmuje dwa parametry: referencję na obiekt i głębokość aktywacji. W wyniku wywołania tej metody aktywowane są wszystkie obiekty, począwszy od obiektu przekazanego jako parametr, wzdłuż pewnej liczby kolejnych powiązań referencyjnych równych głębokości aktywacji.

Przykładowo, aktywacja obiektu wskazywanego przez referencję refF i wszystkich obiektów osiągalnych poprzez ten obiekt (głębokość aktywacji równa 1) wyglądałaby następująco:

```
db.activate(refA.refC.refF,1);
```

Istnieje również metoda o działaniu odwrotnym do metody activate, o nazwie deactivate, która powoduje deaktywację i zwolnienie pamięci zajmowanej przez obiekty. Ponieważ jednak metoda ta nie wchodzi w zakres ćwiczeń, pominiemy ją.

Istnieje również metoda pozwalająca na sprawdzenie, czy obiekt wskazywany przez referencję jest aktywny. Jest to metoda isActive interfejsu ExtObjectContainer. Metoda isActive jako parametr przyjmuje jedynie referencję na sprawdzany obiekt. Przykładowo, sprawdzenie, czy obiekt wskazywany przez referencję refF jest aktywny wyglądałoby następująco:

```
if (db.ext().isActive(refA.refC.refF)) {
            //aktywny
} else {
            //nieaktywny
}
```

(Uwaga! W przyszłych wersjach db4o, metoda isActive może zostać przeniesiona do interfejsu ObjectContainer. Wówczas aktywacja tej metody będzie wyglądać następująco:

```
 db.isActive(refA.refC.refF)
```

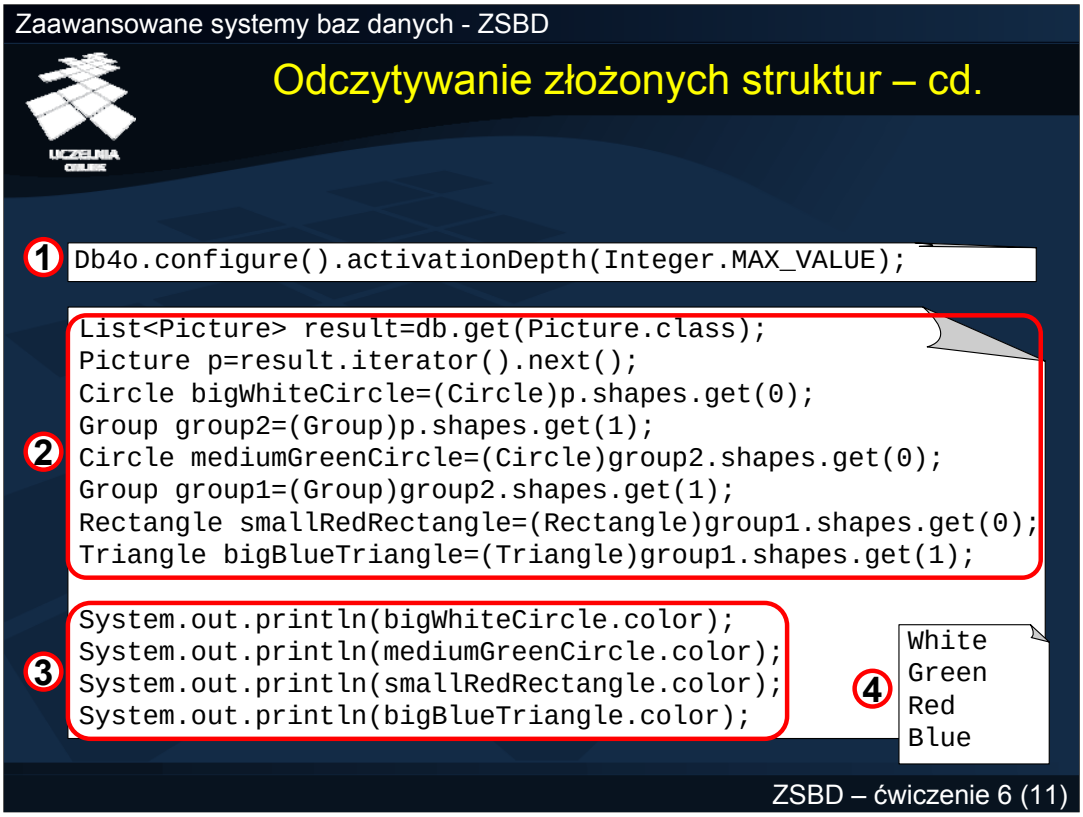
).

Głębokość aktywacji może być ustalony zarówno globalnie dla wszystkich klas, jak i dla każdej z klas z osobna oraz dla każdego pola każdej klasy z osobna. Domyślna globalna głębokość aktywacji wynosi 5, ale można ją zmienić za pomocą metody activationDepth zdefiniowanej w interfejsie Configuration. W celu uzyskania referencji na obiekt implementujący ten interfejs należy wykorzystać statyczną metodę klasy Db4o o nazwie configure. Przykładowo, ustalenie poziomu aktywacji na 10 wygląda następująco:

```
Db4o.configure().activationDepth(10);
```

Operacja ta musi zostać wykonana przed otwarciem połączenia, czyli przed utworzeniem obiektu implementującego interfejs ObjectContainer (np. metodą openFile klasy Db4o). Konfiguracja poziomu aktywacji dla poszczególnych klas z osobna jest poza zakresem ćwiczeń. Osoby zainteresowane powinny zajrzeć do dokumentacji, bądź tutoriala db4o.

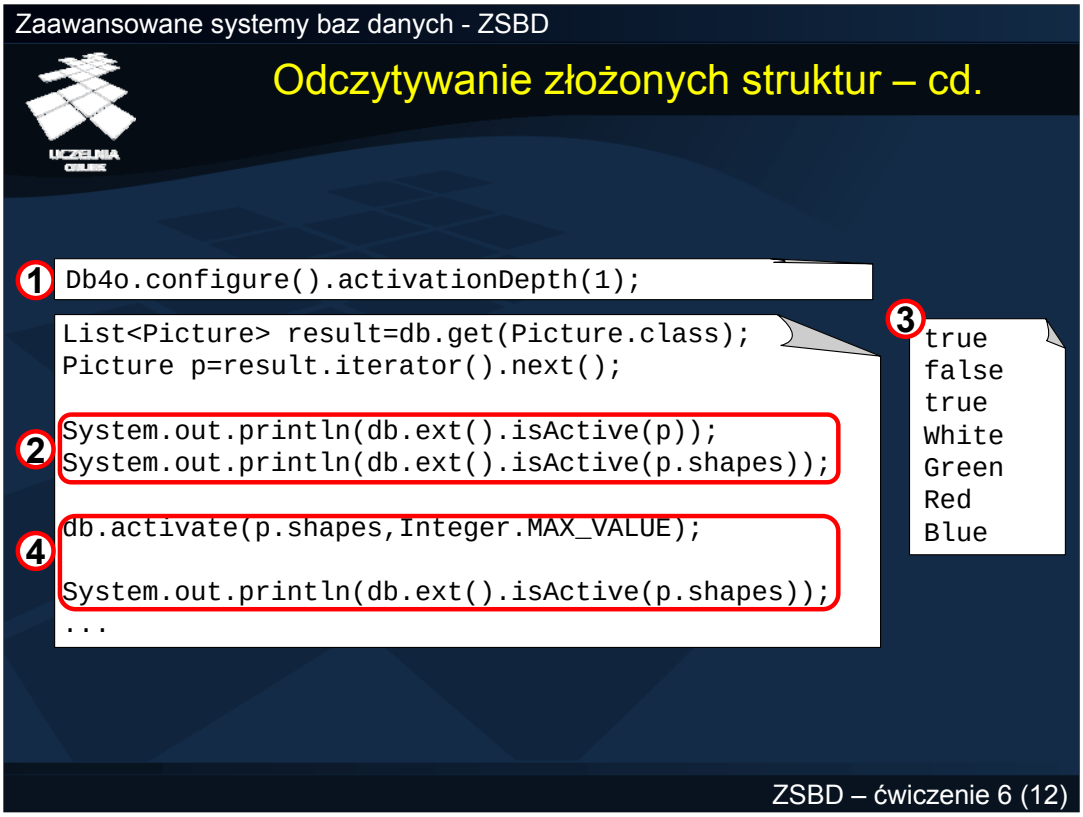
Prócz wyżej opisanych metod istnieje również metoda pozwalająca na takie skonfigurowanie db4o, że odczytanie obiektów konkretnych klas powoduje automatyczne odczytanie obiektów z nich osiągalnych z pominięciem ograniczenia na głębokość aktywacji. Nie wchodzi to jednak w zakres ćwiczeń.



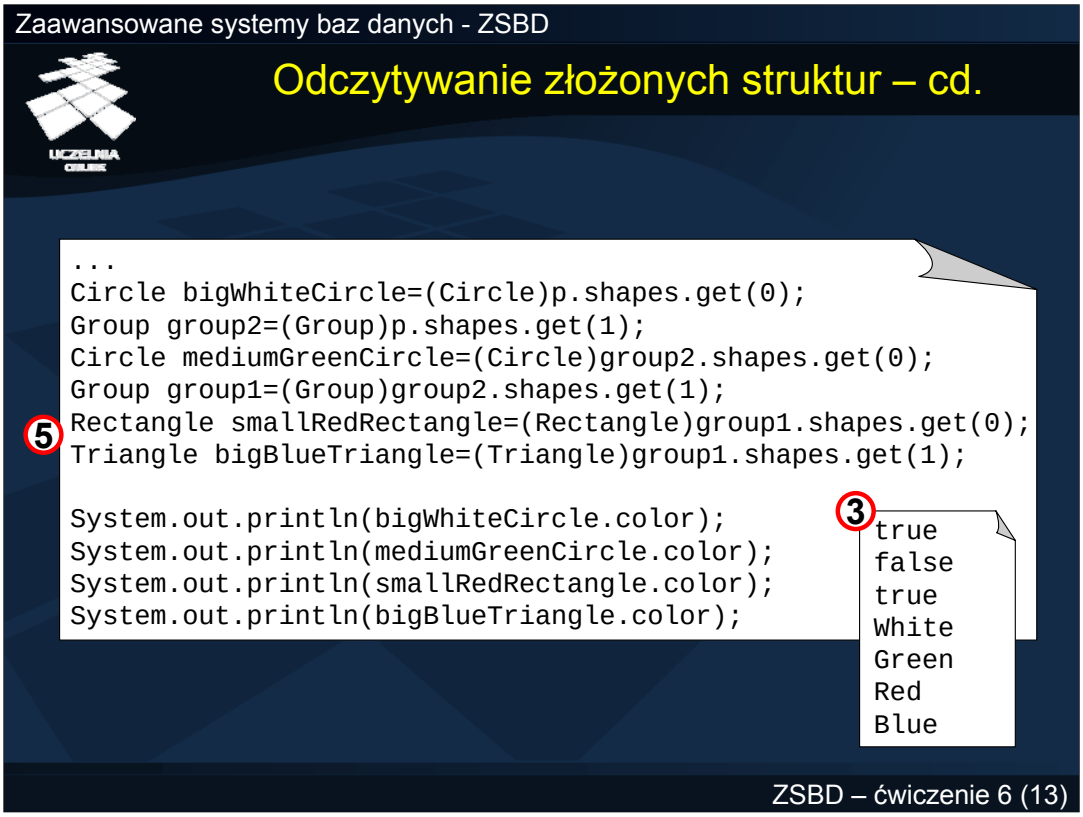
W celu demonstracji działania metod przedstawionych na poprzednim slajdzie będziemy zakładać, że przykładowy rysunek przedstawiony na slajdzie „Złożone struktury danych, dane” został zapisany do bazy danych.

Przykład przedstawiony na slajdzie ustawia bardzo wysoką głębokość aktywacji (1), która gwarantuje odczytanie wszystkich obiektów osiągalnych z obiektu odczytywanego z bazy danych. Po otwarciu bazy danych wykonywane jest zapytanie odczytujące przykładowy rysunek z bazy danych, a referencje na obiekty reprezentujące poszczególne figury są przypisywane do zmiennych (2). Ostatecznie sprawdzane jest, czy obiekty reprezentujące figury został odczytane poprawnie poprzez wypisanie na konsoli kolorów tychże figur (3). Wynikiem działania programu jest (4).

Kompletny kod programu, którego fragmenty pokazano na slajdzie załączono do kursu w postaci pliku: Skeleton-lab6.3.java

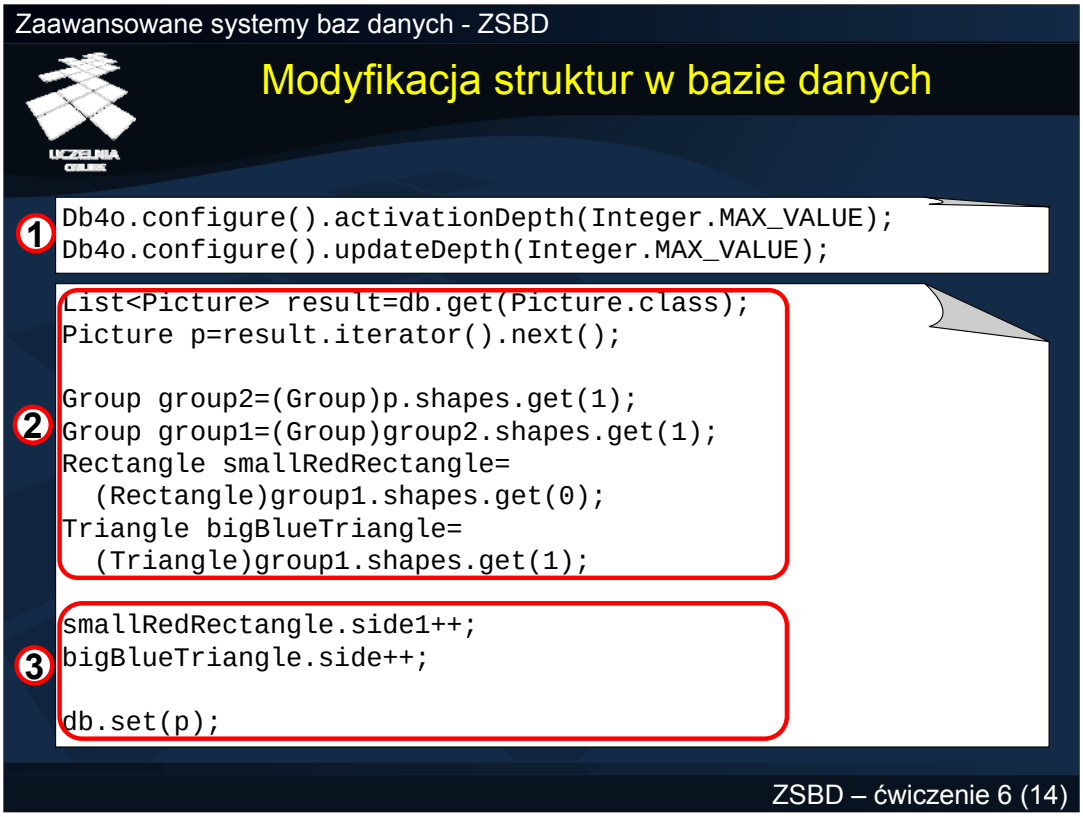


Przykład przedstawiony na tym i kolejnym slajdzie demonstruje użycie metod isActive i activate. W przeciwieństwie do poprzedniego przykładu, tutaj głębokość aktywacji ustawiona jest na 1 (1). Wynika z tego, że tylko obiekt spełniający warunki zapytania jest odczytywany. Wszelkie osiągalne z niego obiekty pozostaną nieaktywne. Pokazuje to fragment programu oznaczony przez (2). W wyniku działania tych dwóch linijek kodu na konsoli pojawia się napis „true” a potem „false” (3), co znaczy, że obiekt reprezentujący rysunek jest aktywny, ale kolekcja (która również jest obiektem) już nie. Dlatego też, w kolejnych linijkach kodu (4) kolekcja, oraz wszystkie obiekty z niej osiągalne, jest aktywowana. Fakt, że kolekcja jest aktywna jest sprawdzany za pomocą metody isActive, której wynik jest wypisywany na konsoli (3). ...



... Ostatni fragment programu (5) jest taki sam jak na poprzednim przykładzie i wypisuje kolory wszystkich figur na rysunku na konsoli (3).

Kompletny kod programu, którego fragmenty pokazano na tym i poprzednim slajdzie załączono do kursu w postaci pliku: Skeleton-lab6.4.java



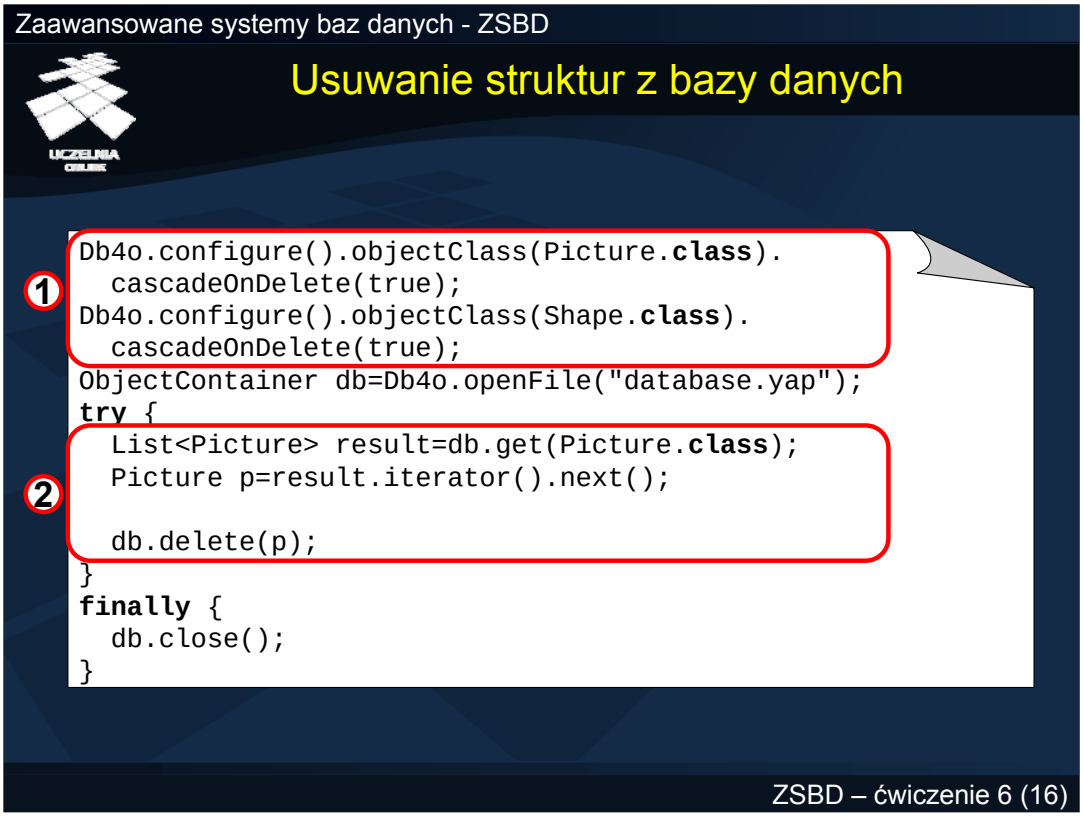
Jak już wspominano wcześniej, modyfikacja obiektów zapisanych w bazie danych polega na wczytaniu ich do pamięci z bazy danych, modyfikacji i ponownym zapisie za pomocą metody set. W przypadku złożonych struktur danych należy również postępować według tego samego schematu. Problem zaczyna się w sytuacji, kiedy, modyfikacji było bardzo dużo. Wywoływanie metody set dla każdego zmodyfikowanego obiektu złożonej struktury danych byłoby bardzo uciążliwe, a implementacja zapisu modyfikacji mogłaby zająć czas, który raczej warto poświęcić na implementację logiki aplikacji. Metoda set zawiera mechanizm pozwalający na aktualizację całych struktur zmodyfikowanych w pamięci. Przy okazji omawiania zapisywania złożonych struktur do bazy danych wspomniano, że metoda set zapisując obiekt sprawdza, czy jego pola nie wskazują na niezapisane do bazy danych obiekty, i jeżeli tak jest to te obiekty są również zapisywane do bazy danych. A co jeśli pola zapisywanego obiektu wskazują na obiekty już zapisywane wcześniej w bazie danych, bądź referencja na taki obiekt zostanie przekazana jako parametr aktualny metody set? W takiej sytuacji zaczyna działać mechanizm podobny do używanego przy odczycie obiektów. Polega on na wykorzystaniu tzw. głębokości aktualizacji, to jest liczby powiązań referencyjnych, wzdłuż których nawiguje db4o w poszukiwaniu obiektów do aktualizacji. Domyślna głębokość aktualizacji wynosi 1, a zatem aktualizowany jest jedynie obiekt przekazany jako parametr metodzie set. Głębokość aktualizacji można ustalić zarówno globalnie, jak i dla każdej z klas z osobna, jak również wyłączyć sprawdzanie głębokości aktualizacji dla obiektów konkretnych klas, albo pól obiektów konkretnych klas. W zakres ćwiczeń wchodzi jedynie ustalanie globalnej głębokości aktywacji. Globalną głębokość aktualizacji można zmodyfikować za pomocą metody updateDepth interfejsu Configuration). Podobnie jak w przypadku głębokości aktywacji, ten parametr musi również zostać ustalony przed nawiązaniem połączenia z bazą danych.

Przykładowy program pokazany na slajdzie ustala globalną głębokość aktualizacji na bardzo wysoką wartość, co w praktyce oznacza, że wywołanie metody set spowoduje aktualizację wszystkich obiektów osiągalnych z obiektu na którego referencję podano jako parametr tej metody (1). Prócz tego, głębokość aktywacji jest również ustawiana na wysoką wartość w celu ułatwienia wczytywania rysunku z bazy danych. Następnie, rysunek jest odczytywany z bazy danych, a referencje na dwie figury z grupy pierwszej są zapisywane do zmiennych (2). Ostatecznie, wykonywana jest modyfikacja obiektów, które są następnie zapisywane do bazy danych za pomocą wywołania metody set na obiekcie reprezentującym cały rysunek (3). Ponieważ głębokość aktywacji jest wysoka, to wszelkie zmiany wprowadzone do obiektów składowych rysunku zostaną zapisane.

Kompletny kod programu, którego fragmenty pokazano na slajdzie załączono do kursu w postaci pliku: Skeleton-lab6.5.java

Przykładowy program pokazany na slajdzie ustala globalną głębokość aktualizacji na bardzo wysoką wartość, co w praktyce oznacza, że wywołanie metody set spowoduje aktualizację wszystkich obiektów osiągalnych z obiektu na którego referencję podano jako parametr tej metody (1). Prócz tego, głębokość aktywacji jest również ustawiana na wysoką wartość w celu ułatwienia wczytywania rysunku z bazy danych. Następnie, rysunek jest odczytywany z bazy danych, a referencje na dwie figury z grupy pierwszej są zapisywane do zmiennych (2). Ostatecznie, wykonywana jest modyfikacja obiektów, które są następnie zapisywane do bazy danych za pomocą wywołania metody set na obiekcie reprezentującym cały rysunek (3). Ponieważ głębokość aktywacji jest wysoka, to wszelkie zmiany wprowadzone do obiektów składowych rysunku zostaną zapisane.

Kompletny kod programu, którego fragmenty pokazano na slajdzie załączono do kursu w postaci pliku: Skeleton-lab6.5.java



Dowolne obiekty, niezależnie od tego, czy są samodzielne, czy też są częścią jakiejś struktury, usuwa się za pomocą metody delete opisywanej wcześniej. Metoda ta usuwa jedynie obiekt, na który referencja została przekazana jako parametr. Nie ma globalnego parametru, podobnego do głębokości aktywacji, bądź głębokości aktualizacji, pozwalającego na usunięcie całej struktury danych za jednym wywołaniem metody. Możliwe jest jednak włączenie automatycznego usuwania obiektów bezpośrednio osiągalnych z obiektów konkretnej klasy, bądź obiektów wskazywanych przez określone pola określonych klas. Na ćwiczeniach pokazany zostanie jedynie sposób automatycznego usuwania obiektów bezpośrednio osiągalnych z obiektów konkretnej klasy. Automatyczne usuwanie włącza się za pomocą metody cascadeOnDelete interfejsu ObjectClass. Obiekty implementujące interfejs ObjectClass pozwalają na ustawianie parametrów konfiguracyjnych dotyczących pojedynczych klas. Oby uzyskać taki obiekt należy skorzystać z metody objectClass interfejsu Configuration. Przykładowo, aby włączyć kaskadowe usuwanie obiektów klasy Shape, należy, przed otwarciem połączenia z bazą danych, wykonać następującą instrukcję:

Db4o.configuration().objectClass(Shape.class).cascadeOnDelete(true);

Przy usuwaniu obiektów należy zwrócić szczególną uwagę na to, by żadne inne obiekty w bazie danych nie zawierały referencji na usuwane obiekty, gdyż db4o tego nie sprawdza (przynajmniej jest tak w wersji 5.2, dla której opracowano ćwiczenia), a zatem można łatwo doprowadzić do niespójności w bazie danych. Dodatkowym problemem występującym, przy włączonym automatycznym usuwaniu jest automatyczne usuwanie obiektów podczas aktualizacji, jeżeli aktualizowany obiekt stracił na nie referencje.

Dokumentacja podaje tutaj następujący przykład:

```
ObjectContainer con;
Bar bar1 = new Bar();
Bar bar2 = new Bar();
foo.bar = bar1;
con.set(foo); // do bazy danych zapisywane są obiekty foo i bar1
foo.bar = bar2;
con.set(foo); // do bazy danych zapisywany jest obiekt bar2, a
usuwany jest obiekt bar1
```

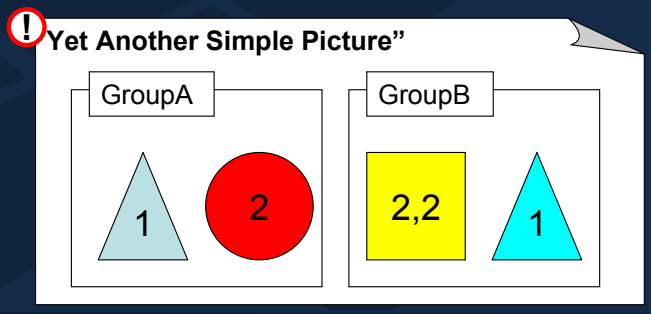
Obiekt bar1 zostanie usunięty z bazy danych niezależnie od tego, czy inne obiekty posiadają referencje na niego, czy nie.

Przykładowy program pokazany na slajdzie włącza automatyczne usuwanie obiektów, na które referencje zapisano w obiektach klas Picture i Shape (1), Następnie odczytuje obiekt reprezentujący przykładowy rysunek i usuwa go z bazy danych (2). Ponieważ włączone jest automatyczne usuwanie obiektów, ta jedna instrukcja spowoduje usunięcie całego rysunku z bazy danych.

Kompletny kod programu, którego fragmenty pokazano na slajdzie załączono do kursu w postaci pliku: Skeleton-lab6.6.java

Zadanie (1)

A. Utwórz i zapisz do bazy danych obiekty reprezentujące przykładowy rysunek, który był omawiany na poprzednich slajdach.
B. Utwórz w pamięci operacyjnej obiekty poniższego rysunku i zapisz go do bazy danych.



(!) Liczby zapisane wewnątrz poszczególnych figur to długości boków, lub promienia w zależności od figury.

Zadanie (2)

- Ustaw globalną głębokość aktywacji na 20 i odczytaj rysunek „Yet Another Simple Picture”. Sprawdź , czy wszystkie figury na rysunku są aktywne (metoda isActive).

Zadanie (3)

- Ustaw globalną głębokość aktywacji na 1 i odczytaj rysunek „Yet Another Simple Picture”. Sprawdź, czy kolekcja przechowująca figury rysunku jest aktywna (metoda isActive). Użyj metody activate do aktywacji kolekcji i pozostałych obiektów rysunku.

Zadanie (4)

A. Wczytaj rysunek „Yet Another Simple Picture” do pamięci. Zmodyfikuj obiekt reprezentujący zielony trójkąt. Spróbuj zapisać zmiany poprzez wywołanie metody *set* na obiekcie klasy Picture bez uprzedniego ustawiania głębokości aktualizacji.

B. Sprawdź czy zmiany zostały zapisane.

C. Zrób to samo co w punkcie A, ale ustaw głębokość aktualizacji na 20.

D. Sprawdź jeszcze raz, czy tym razem zmiany zostały zapisane.

(!) Każdy punkt zadania (4) powinien zostać wykonany jako osobne uruchomienie programu.

(!!) Do podpunktów B i D możesz do tego wykorzystać ten sam program.

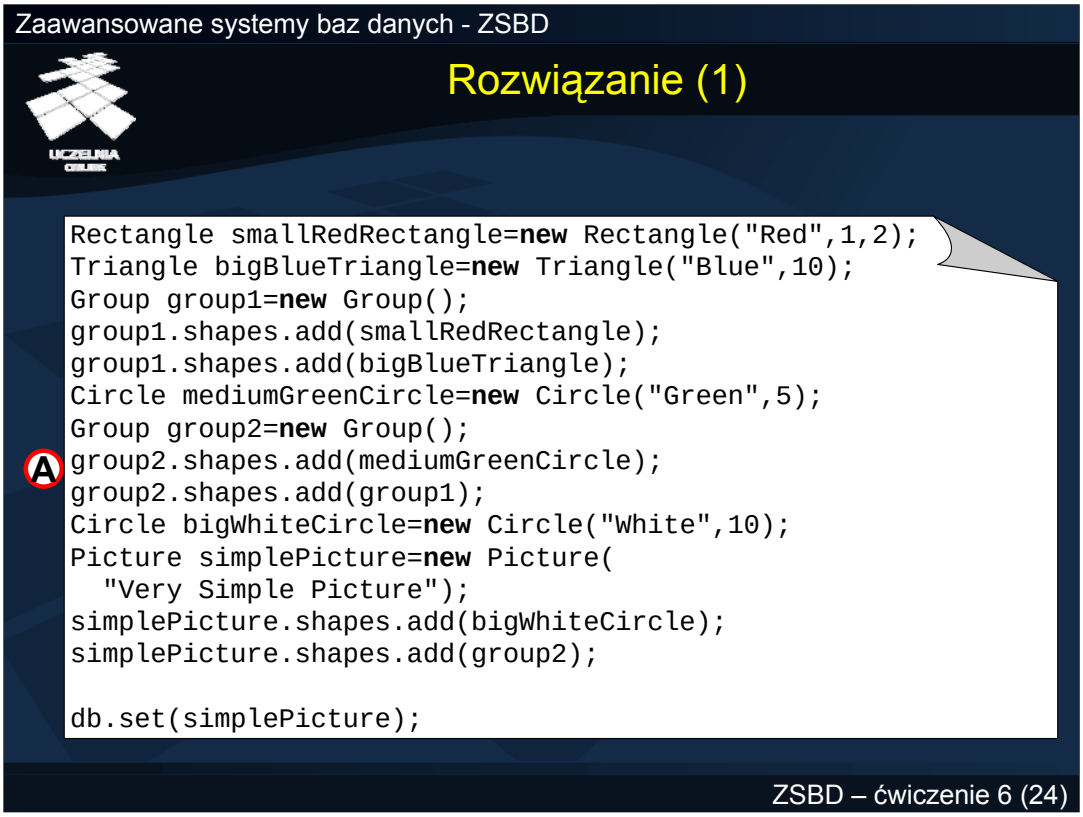
Zadanie (5)

- Usuń rysunek „Yet Another Simple Picture” z bazy danych, pozostawiając przykładowy rysunek nietknięty.

Zadanie (6)

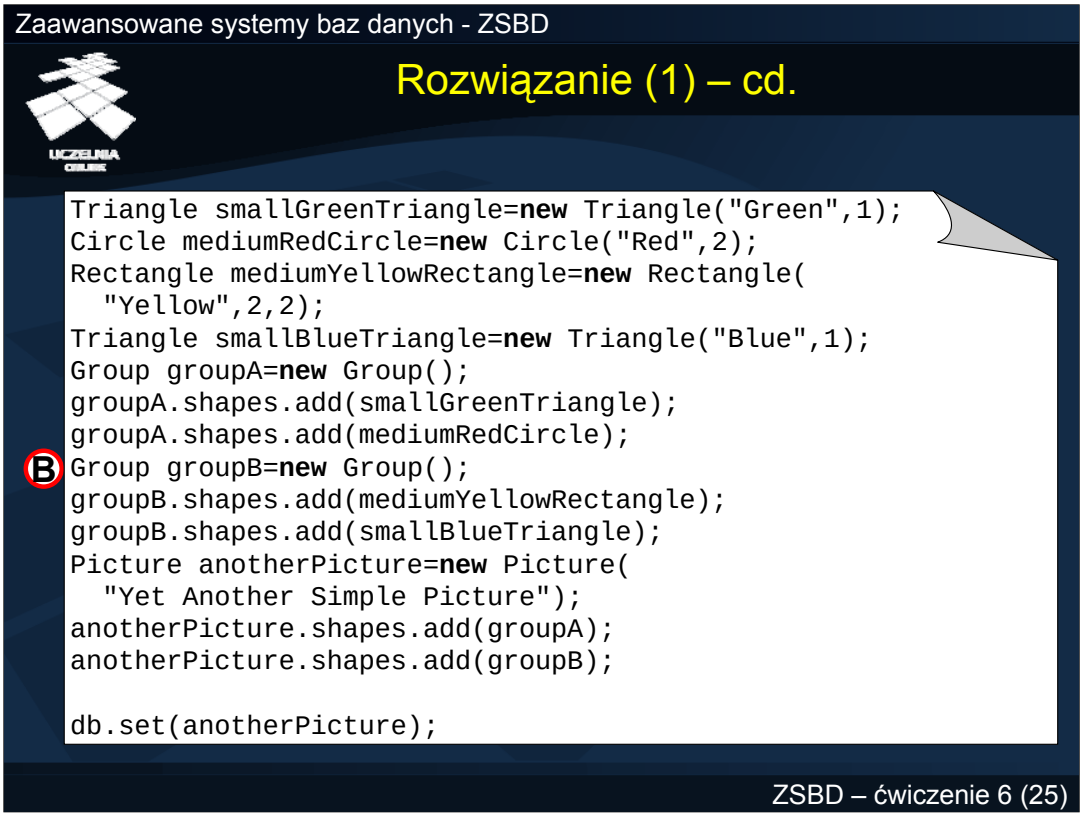
- Zaprojektuj schemat tabel pozwalający na przechowywanie takich samych rysunków jak schemat klas omawiany na ćwiczeniu (ze wszystkimi możliwościami grupowania itp.). Wyciągnij wnioski co do zalet i wad obiektowych baz danych w porównaniu do relacyjnych baz danych.

Kiedy wykonasz zadania możesz je porównać z rozwiązaniami przedstawionymi na kolejnych slajdach. Rozwiązanie do ostatniego zadania będzie obejmować jedynie schemat tabel. Wnioski powinno się wyciągnąć samemu ;).



Slajd pokazuje rozwiązanie zadania (1A), którego treść przytoczono poniżej. Rozwiązanie zadania (1) kontynuowane jest na kolejnym slajdzie.

(A) Utwórz i zapisz do bazy danych obiekty reprezentujące przykładowy rysunek, który był omawiany na poprzednich slajdach.



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

(B) Utwórz w pamięci obiekty rysunku „Yet Another Simple Picture” i zapisz go do bazy danych.

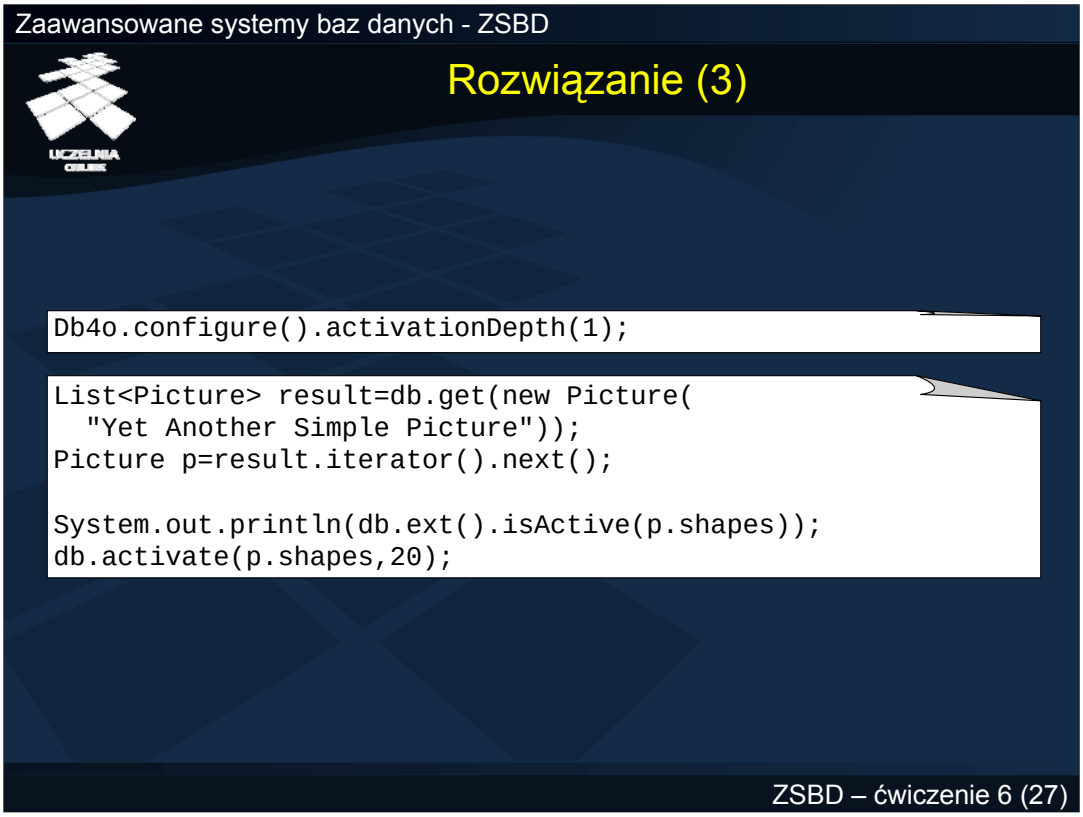
Rozwiązanie (2)

```
Db4o.configure().activationDepth(20);
```

```
List<Picture> result=db.get(new Picture(
  "Yet Another Simple Picture"));
Picture p=result.iterator().next();
Group groupA=(Group)p.shapes.get(0);
Group groupB=(Group)p.shapes.get(1);
Triangle smallGreenTriangle=(Triangle)groupA.shapes.get(0);
Circle mediumRedCircle=(Circle)groupA.shapes.get(1);
Rectangle mediumYellowRectangle=
  (Rectangle)groupB.shapes.get(0);
Triangle smallBlueTriangle=(Triangle)groupB.shapes.get(1);
System.out.println(db.ext().isActive(smallGreenTriangle));
System.out.println(db.ext().isActive(mediumRedCircle));
System.out.println(db.ext().isActive(
  mediumYellowRectangle));
System.out.println(db.ext().isActive(smallBlueTriangle));
```

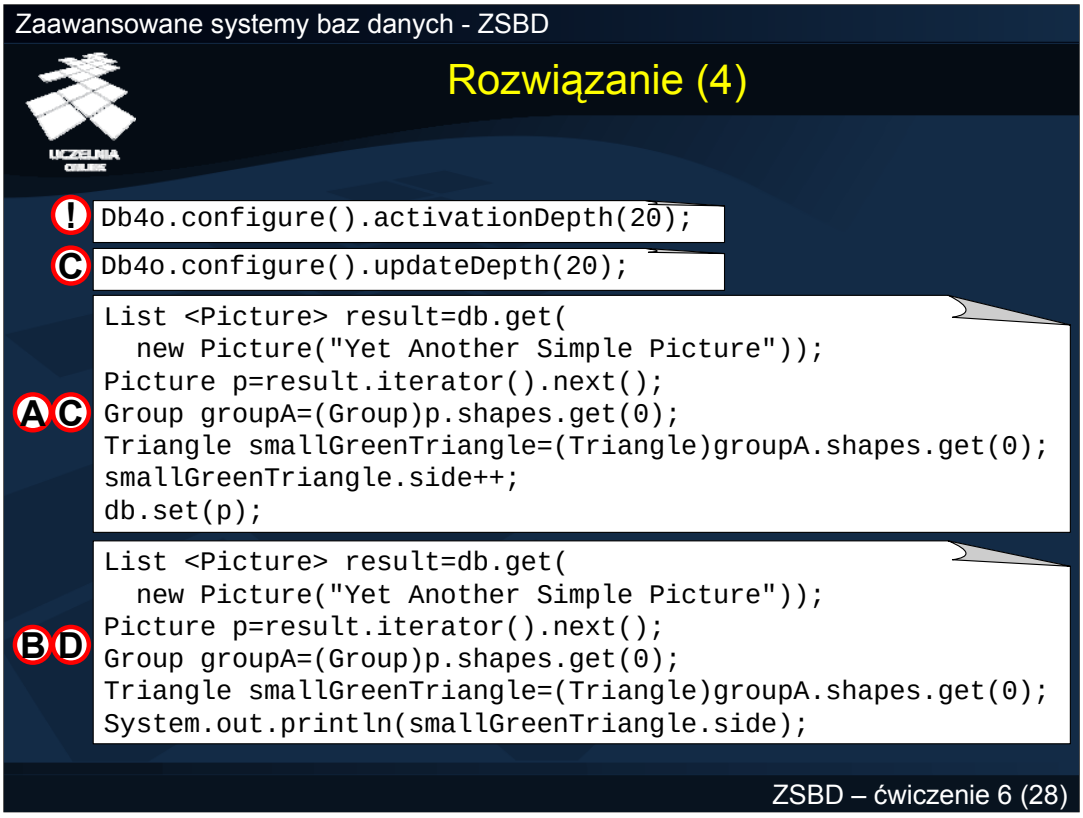

Slajd pokazuje rozwiązanie zadania (2), którego treść przytoczono poniżej.

Ustaw globalną głębokość aktywacji na 20 i odczytaj rysunek „Yet Another Simple Picture”. Sprawdź , czy wszystkie figury na rysunku są aktywne (metoda isActive).



Slajd pokazuje rozwiązanie zadania (3), którego treść przytoczono poniżej.

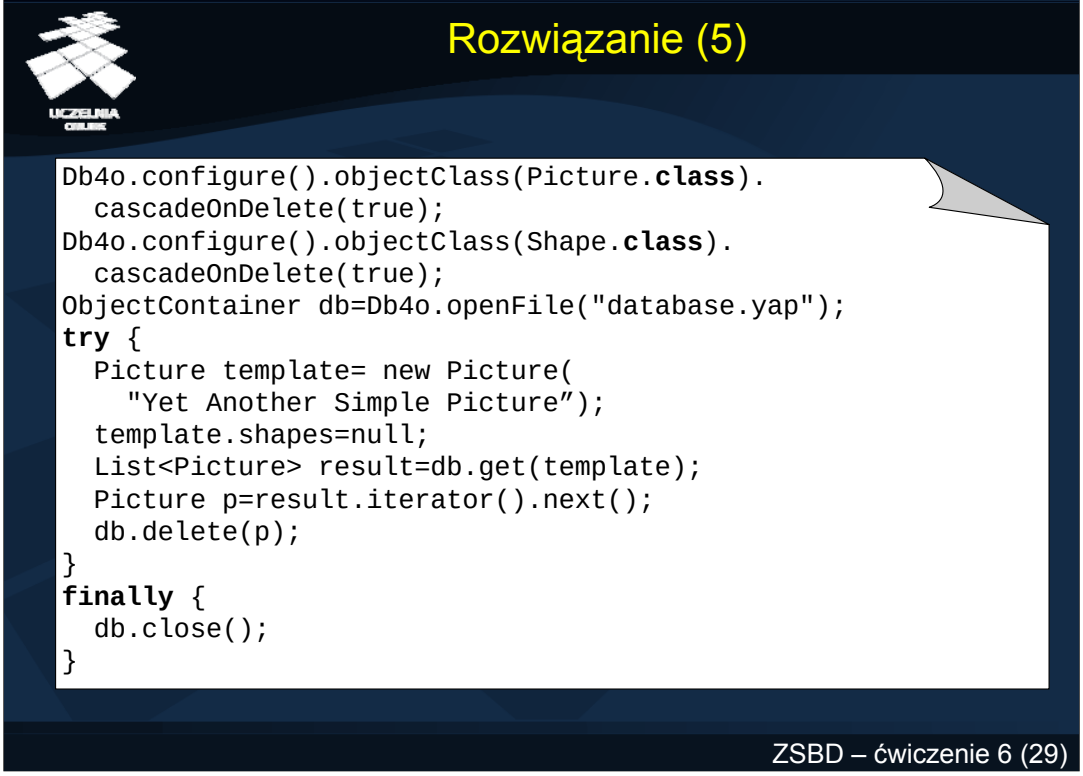
Ustaw globalną głębokość aktywacji na 1 i odczytaj rysunek „Yet Another Simple Picture”. Sprawdź, czy kolekcja przechowująca figury rysunku jest aktywna (metoda isActive). Użyj metody activate do aktywacji kolekcji i pozostałych obiektów rysunku.



Slajd pokazuje rozwiązanie zadania (4), którego treść przytoczono poniżej.

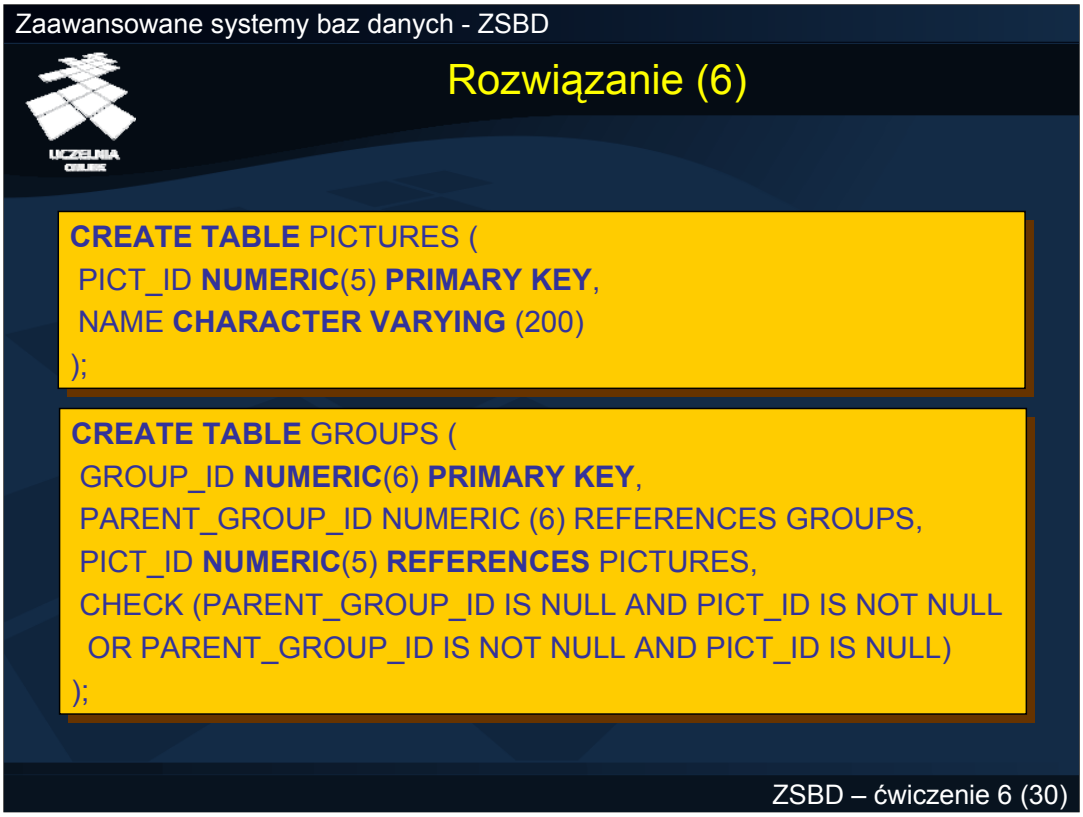
(!) ten fragment kodu powinien się znaleźć we wszystkich programach rozwiązujących kolejne podpunkty zadania.

- (A) Wczytaj rysunek „Yet Another Simple Picture” do pamięci. Zmodyfikuj obiekt reprezentujący zielony trójkąt. Spróbuj zapisać zmiany poprzez wywołanie metody set na obiekcie klasy Picture bez uprzedniego ustawiania głębokości aktualizacji.
- (B) Sprawdź czy zmiany zostały zapisane.
- (C) Zrób to samo co w punkcie A, ale ustaw głębokość aktualizacji na 20.
- (D) Sprawdź jeszcze raz, czy tym razem zmiany zostały zapisane.



Slajd pokazuje rozwiązanie zadania (5), którego treść przytoczono poniżej.

Usuń rysunek „Yet Another Simple Picture” z bazy danych, pozostawiając przykładowy rysunek nietknięty.



Slajd pokazuje rozwiązanie zadania (6), którego treść przytoczono poniżej. Rozwiązanie jest kontynuowane na kolejnych slajdach.

Zaprojektuj schemat tabel pozwalający na przechowywanie takich samych rysunków jak schemat klas omawiany na ćwiczeniu (ze wszystkimi możliwościami grupowania itp.). Wyciągnij wnioski co do zalet i wad obiektowych baz danych w porównaniu do relacyjnych baz danych.

Rozwiązanie (6) – cd.

```
CREATE TABLE SHAPES (
 SHAPE_ID NUMERIC(7) PRIMARY KEY,
 SHAPE CHARACTER VARYING (20) NOT NULL,
 COLOR CHARACTER VARYING (20) NOT NULL,
 PARAM1 NUMERIC(10) NOT NULL,
 PARAM2 NUMERIC(10),
 GROUP_ID NUMERIC(4) REFERENCES GROUPS,
 ...
```

Rozwiązanie (6) – cd.

```
...
 PICT_ID NUMERIC(5) REFERENCES PICTURES,
 CHECK (SHAPE='RECTANGLE' AND PARAM2 IS NOT NULL OR
   SHAPE<>'RECTANGLE' AND PARAM2 IS NULL),
 CHECK (SHAPE IN ('TRIANGLE','RECTANGLE','CIRCLE')),
 CHECK (GROUP_ID IS NULL AND PICT_ID IS NOT NULL OR
   GROUP_ID IS NOT NULL AND PICT_ID IS NULL)
);
```

Podsumowanie

- W trakcie zajęć poznaliście Państwo mechanizmy db4o pozwalające na pracę ze złożonymi strukturami danych.
- Dowiedzieliście się w jaki sposób można zapisywać, odszukiwać i odczytywać, modyfikować oraz usuwać złożone struktury danych.
- Zainteresowani dalszą nauką o OSZBD db4o powinni zajrzeć do dokumentacji technicznej i tutoriala, dostarczanych razem z db4o.