



# JPS Ćw 2

## Skład danych

### ZAŁOŻENIA SBA

---

Obiektowa baza w oparciu o SBA może przechowywać następujące dane:

- Proste
  - <OID, nazwa, wartość>
    - np. <i0, imie, "Jan">, <i1, pensja, 3354>
- Złożone
  - <OID, nazwa, { OID1, OID2, OID3, ... }>
    - np. <i2, pracownik, { i0, i1 }>
- Referencyjne (tych ostatnich nie będziemy potrzebować, ale można przygotować skład danych na ich obsłużenie).
  - <OID, nazwa, OID1>
    - np. <i3, pracuje\_w, i4>

Nasza baza danych powinna być przygotowana na przechowanie danych prostych o typach:

- String
- Integer
- Double/Real
- Boolean

Wszystkie obiekty w bazie danych trzymane są w jednym obiekcie-korzenie (zwanym dalej „entry”) dostępnym globalnie z każdego miejsca programu.

## OID

---

W założeniu naszego projektu wszystkie obiekty przechowywane są w pamięci. W celu ich składowania wykorzystana zostanie mapa (np. **Dictionary** w C# lub **HashMap** w Javie) pozwalająca na szybkie tworzenie i dostęp do obiektu na podstawie jego unikalnego OID.



Skład danych powinien dbać o nadawanie OID-ów nowym obiektom oraz dbać o ich unikalność. OID może być liczbą całkowitą (np. inkrementowaną o jeden z każdym nowym obiektem) lub łańcuchem znaków. Ważne jest jedynie zachowanie stu procentowej unikalności.

Należy również dopilnować, by każdy obiekt utworzony w systemie został zapisany do Hashmapy (najłatwiej w konstruktorze nadtypu).

## CRUD

---

Skład danych powinien jest również odpowiedzialny za podstawowe operacje na danych zwanych CRUD – Create, Retrieve, Update, Delete.

### Create

1. Pusta baza danych

```
CREATE(COMPLEX, "ENTRY", {}, -)
```

```
<i0, "entry", {}>
```

2. Tworzymy pracownika

```
CREATE(COMPLEX, "EMP", {}, 10)
```

```
<i0, "entry", {i1}>
```

```
<i1, "emp", {}>
```

3. Tworzymy nazwisko

```
CREATE(STRING, "ENAME", "KOWALSKI", 11)
```

```
<i0, "entry", {i1}>
```

```
<i1, "emp", {i2}>
```

```
<i2, "ename", "Kowalski">
```

4. Tworzymy pensję

```
CREATE(INTEGER, "SALARY", 1000, 11)
```

```
<i0, "entry", {i1}>
```

```
<i1, "emp", {i2, i3}>
```

```
<i2, "ename", "Kowalski">
```

```
<i3, "salary", 1000>
```



## Retrieve

*Dla bazy danych:*

$\langle i0, \text{"entry"}, \{i1, i4\} \rangle$

$\langle i1, \text{"emp"}, \{i2, i3\} \rangle$

$\langle i2, \text{"ename"}, \text{"Kowalski"} \rangle$

$\langle i3, \text{"salary"}, 1000 \rangle$

$\langle i4, \text{"emp"}, \{i5\} \rangle$

$\langle i5, \text{"ename"}, \text{"Nowak"} \rangle$

*Operacje:*

$RETRIEVE(i0) = \{i1, i4\}$

$RETRIEVE(i1) = \{i2, i3\}$

$RETRIEVE(i2) = \text{"KOWALSKI"}$

$RETRIEVE(i3) = 1000$

## Update

*Dla bazy danych:*

$\langle i0, \text{"entry"}, \{i1, i4\} \rangle$

$\langle i1, \text{"emp"}, \{i2, i3\} \rangle$

$\langle i2, \text{"ename"}, \text{"Kowalski"} \rangle$

$\langle i3, \text{"salary"}, 1000 \rangle$

$\langle i4, \text{"emp"}, \{i5\} \rangle$

$\langle i5, \text{"ename"}, \text{"Nowak"} \rangle$

*Aktualizacja nazwiska i pensji:*

$UPDATE(i2, \text{"WALEWSKI"})$

$UPDATE(i3, 2000)$

*Baza po modyfikacjach:*

$\langle i0, \text{"entry"}, \{i1, i4\} \rangle$

$\langle i1, \text{"emp"}, \{i2, i3\} \rangle$

$\langle i2, \text{"ename"}, \text{"Walewski"} \rangle$

$\langle i3, \text{"salary"}, 2000 \rangle$

$\langle i4, \text{"emp"}, \{i5\} \rangle$

$\langle i5, \text{"ename"}, \text{"Nowak"} \rangle$



## Delete

*Dla bazy danych:*

*<i0, "entry", {i1, i4, i7}>*

*<i1, "emp", {i2, i3, i6}>*

*<i2, "ename", "Kowalski">*

*<i3, "salary", 1000>*

*<i4, "emp", {i5}>*

*<i5, "ename", "Nowak">*

*<i6, "works\_in", i7>*

*<i7, "dept", {i8, i9}>*

*<i8, "dname", "Sales">*

*<i9, "location", "Warsaw">*

*Usunięcie pierwszego pracownika, nazwiska drugiego pracownika i departamentu:*

*Delete(i5) - usuwa obiekt i5*

*Delete(i7) - usuwa obiekt i7, wszystkie jego podobiekty i obiekt i6*

*Delete(i1) - usuwa obiekt i1 i wszystkie jego podobiekty*

*Baza po zmianach:*

*<i0, "entry", {i4}>*

*<i4, "emp", {}>*



## WCZYTANIE XML – PRZYKŁAD

Oto przykładowy XML:

```
<baza>
  <test>
    10
  </test>
  <osoba>
    <numer>s0000</numer>
    <imie>Anna</imie>
    <nazwisko>Kowalska</nazwisko>
    <adres>
      <ulica>Jerozolimskie</ulica>
      <kod>00-000</kod>
      <miasto>Warszawa</miasto>
    </adres>
    <ksiazka>
      <autor>Adam Mickiewicz</autor>
      <tytul>Pan Tadeusz</tytul>
    </ksiazka>
  </osoba>
</baza>
```

W wyniku jego przetworzenia powinniśmy uzyskać następujące obiekty:

*<i0, „entry”, {i1, i2}>*

*<i1, „test”, 10>*

*<i2, „osoba”, {i3,i4,i5,i9}>*

*<i3, „imie”, Anna>*

*<i4, „nazwisko”, Kowalska>*

*<i5, „adres”, {i6, i7, i8}>*

*<i6, „ulica”, Jerozolimskie>*

*<i7, „kod”, 00-000>*

*<i8, „miasto”, Warszawa>*

*<i9, „ksiazka”, {i10, i11}>*

*<i10, „autor”, Adam Mickiewicz>*

*<i11, „tytul”, Pan Tadeusz>*

## IMPLEMENTACJA

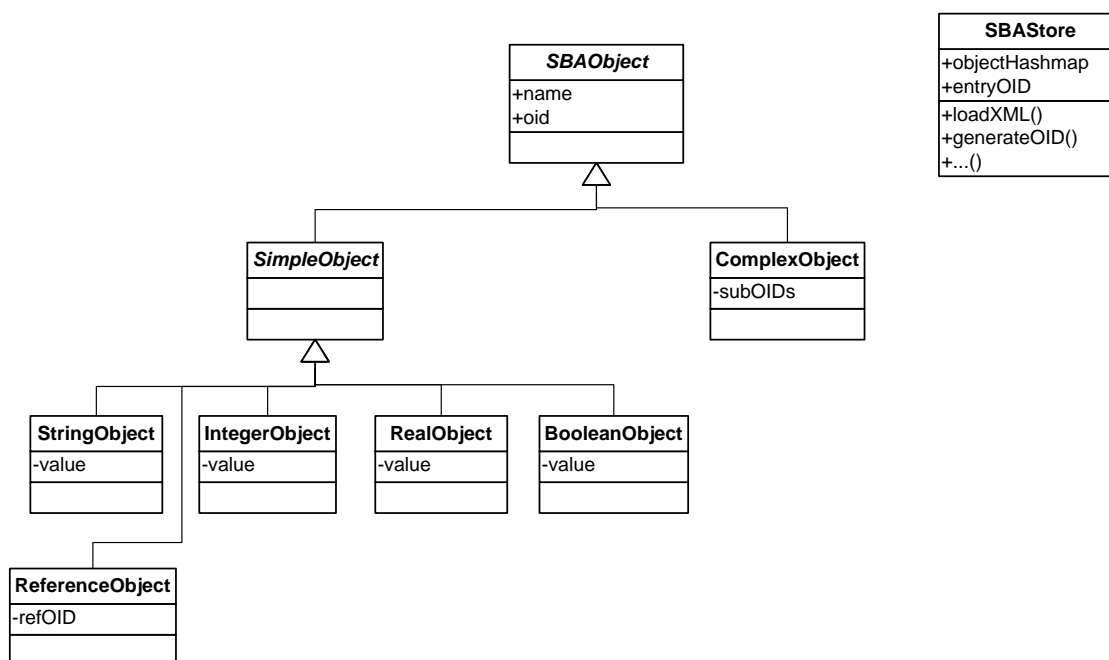


Diagram UML powyżej przedstawia docelowy kształt składu danych. Dodatkowo, klasę **SBAStore** należy wyposażyć w zestaw metod potrzebny do realizacji zadań CRUD.

**Wczytanie XML** powinno przebiegać na zasadzie:

1. Wczytaj plik.
2. Utwórz element-korzeń „entry”. Wczytaj wszystkie tagi korzenia.
3. Dla każdego elementu:
  - a. Jeśli jest elementem prostym utwórz obiekt **SimpleObject** (spróbuj odczytać jego typ na podstawie zawartości)
  - b. Jeśli jest elementem złożonym:
    - i. Utwórz obiekt **ComplexObject**.
    - ii. Powtórz czynność 3. i wszystkie odczytane obiekty zapisz do tablicy `oid`.

**ZALICZENIE MINI-PROJEKTU OPIERAĆ SIĘ BĘDZIE NA WCZYTANIU PRZYGOTOWANEGO PLIKU XML I WYPISANIU NA EKRAŃ WSZYSTKICH WCZYTANYCH OBIEKTÓW NA ZASADZIE ZAPREZENTOWANEJ NA STRONIE 5 TEGO DOKUMENTU.**