



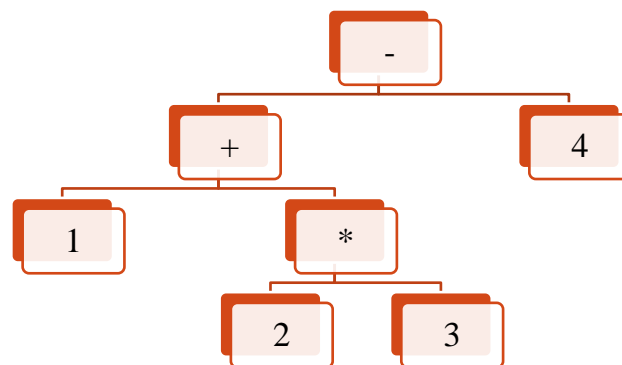
# JPS Ćw 6

## AST – drzewo składniowe

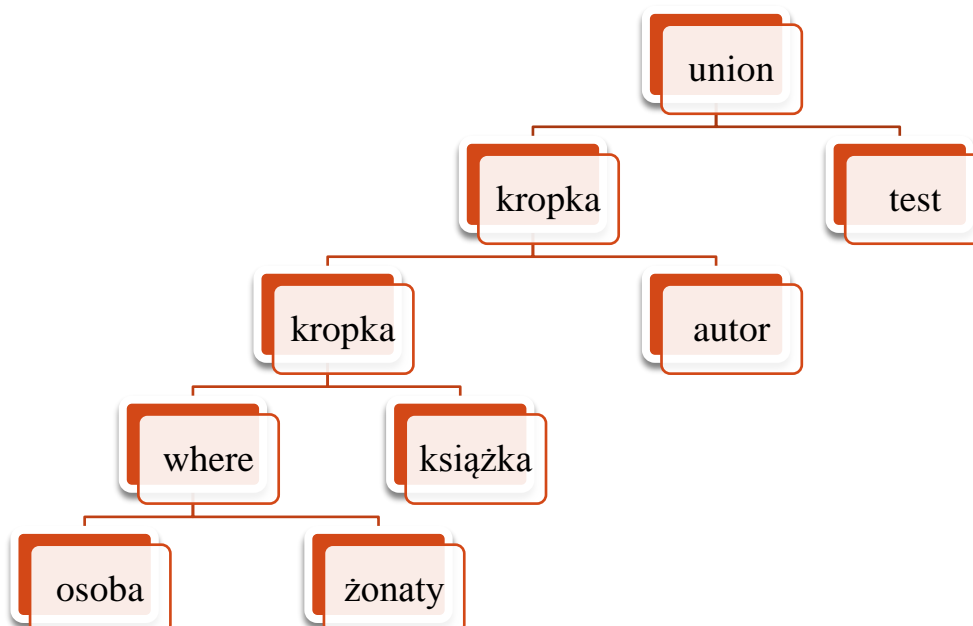
AST (Abstract Syntax Tree) jest strukturą odpowiedzialną za przechowywanie zapytania (np. w SBQL) w sposób pozwalający na jego łatwe wykonanie i odwzorowujący złożoność tego zapytania. Poprzez analizę AST możemy jednoznacznie określić **kolejność wykonywania działań**, co jest najważniejszym zadaniem AST.

Kilka drzew składniowych zostało już przedstawionych w poprzednich zadaniach. Dla przypomnienia:

**1+2\*3-4**

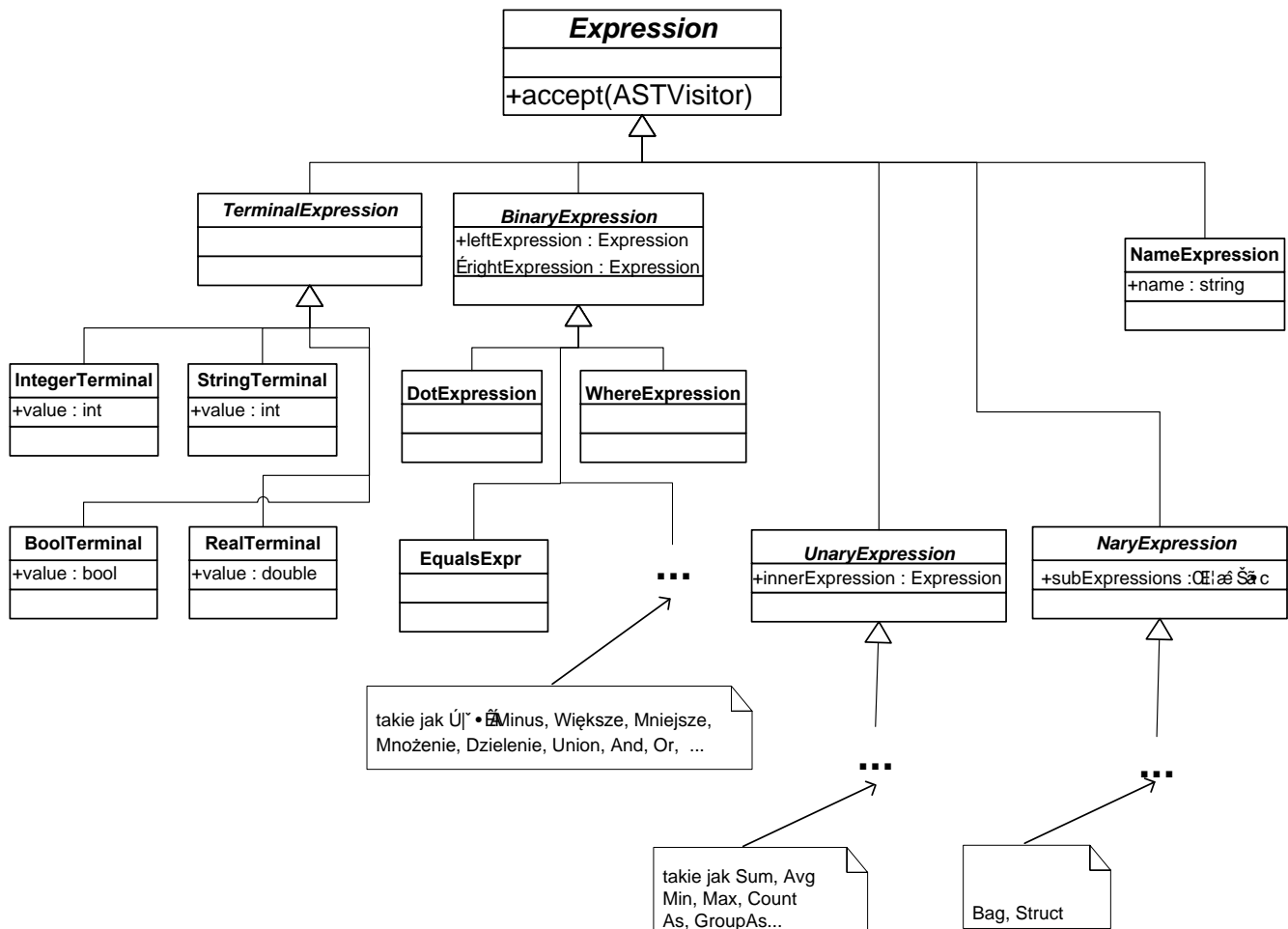


**((osoba where zonaty).ksiazka.autor) union (test);**





## IMPLEMENTACJA



Implementacja AST powinna opierać się na bazie wzorca Visitor przedstawionego na poprzednich ćwiczeniach. Klasa **ASTVisitor** została pominięta na diagramie lecz należy ją zaimplementować zgodnie z kanonem wzorca.

## ZALICZENIE

Zaliczenie projektu polega na przerobieniu kodu z mini-projektu 3 tak aby ewaluacja zapytań nie odbywała się poprzez ręczne pisanie pop i push (itp.) na obu stosach, lecz tak by fragmenty odpowiadające przetwarzaniu kolejnych operatorów znalazły się w odpowiednich metodach odwiedzającego.



Kod w main powinien przybrać podobny kształt:

```
Expression ex = new DotExpression(  
    new DotExpression(  
        new WhereExpression(  
            new NameExpression(„osoba”),  
            new NameExpression(„żonaty”)  
        ),  
        new NameExpression(„książka”)  
    ),  
    new NameExpression(„autor”)  
);  
ex.accept(new ASTVisitor());
```

W ramach zaliczenia, należy zaimplementować następujące zapytania:

Jeśli pierwsza litera twojego imienia jest od A do K:

1. ((osoba where zonaty).książka.autor)

Jeśli pierwsza litera twojego imienia jest od L do Z

2. ((osoba where imie=„Maciej”).adres.ulica)

Jeśli druga litera twojego imienia jest od A do K:

3. ((osoba.adres) where numer=50).(ulica);

Jeśli druga litera twojego imienia jest od L do Z

4. ((osoba.adres) where kod=„00-222”).(miasto);

Przykładowo osoba o imieniu Marcin wykonuje zapytania drugie i trzecie.