# 📘 Raport z Implementacji Interpretera Języka ArchiCode

## 1. Cel projektu

Celem projektu było stworzenie interaktywnego interpretera języka ArchiCode, który pozwala na:

• definiowanie i wykonywanie funkcji (blueprintów),

• operacje arytmetyczne, logiczne, warunkowe i pętle,

• dynamiczne zarządzanie zmiennymi w różnych zakresach,

• obsługę błędów wykonania i składniowych ze śledzeniem stosu wywołań,

• prosty system typów: int, float, bool, char, string z inferencją i rzutowaniem.

Interpreter został zaimplementowany w języku Java przy użyciu ANTLR4 jako generatora parsera.

## 2. Filozofia i Decyzje Projektowe

Język ArchiCode został zaprojektowany z myślą o czytelności, czerpiąc z terminologii architektonicznej, gdzie kod ma przypominać techniczny plan.

• **Metafora architektoniczna:** Słowa kluczowe jak blueprint, Core czy define mają za zadanie budować mentalny model programu jako projektu konstrukcyjnego.

• **Model wykonania:** Kluczową decyzją było przyjęcie modelu, w którym **każde wywołanie funkcji (blueprint) tworzy własny, odizolowany kontekst wykonania** (nowy obiekt ArchiCodeVisitorImpl i Memory). Ten kontekst współdzieli jedynie globalny zakres zmiennych i definicje funkcji, co upraszcza zarządzanie stanem i zapobiega niepożądanym efektom ubocznym między wywołaniami.

• **Przeciążanie funkcji:** Język wspiera przeciążanie blueprintów. Dwie funkcje mogą mieć tę samą nazwę, o ile różnią się sygnaturą (liczbą i/lub typami parametrów).

Mechanizm ten jest realizowany dynamicznie w czasie wykonania.

## 3. Gramatyka języka (ANTLR4)

Parser języka ArchiCode oparty jest na gramatyce zdefiniowanej w pliku ArchiCode.g4. Definiuje ona konstrukcje takie jak pętle, warunki, definicje zmiennych i funkcji.

```
// Fragment gramatyki dla kluczowych instrukcji
grammar ArchiCode;

program: statement+ EOF;

statement
    : ...
    | defineStatement
    | assignStatement
    | repeatStatement
    | checkStatement
    | blueprintStatement
    | blueprintCallStatement
    | coreStatement
    ;

coreStatement
    : 'Core' '(' paramList? ')' 'delivers' type VarName block
    ;

blueprintStatement
    : 'blueprint' CapitalVarName '(' paramList? ')'
      ('delivers' type VarName ('=' expr)?)? block
    ;

blueprintCallStatement
    : CapitalVarName expr* (';')?
    ;

// ...reszta gramatyki...
```

## 4. Struktura i Kluczowe Komponenty

• **ArchiCodeVisitorImpl.java** – Serce interpretera. Implementuje logikę dla każdego węzła drzewa AST.

• **Memory.java** – System zarządzania pamięcią, zakresami i funkcjami.

• **Value.java (i podklasy)** – Polimorficzna reprezentacja wartości.

• **Blueprint.java** – Obiekt reprezentujący zdefiniowaną funkcję.

## 5. Główne Mechanizmy Działania

• **Zarządzanie zakresem:** Każdy blok kodu {...} tworzy nowy zakres przez wywołanie memory.newScope().

• **System typów, rzutowanie i inferencja:** Typ zmiennej może być jawny lub wywnioskowany. Przy przypisaniach i przekazywaniu parametrów typy są sprawdzane i konwertowane.

• **Wywołanie funkcji (blueprint):** Proces ten obejmuje generowanie sygnatury, wyszukanie funkcji, tworzenie izolowanego kontekstu wykonania, wiązanie parametrów i zwrot wartości.

• **Obsługa pętli:** Pętle repeat automatycznie tworzą i zarządzają zmienną o nazwie step.

## 6. Obsługa Błędów

• **Błędy składniowe:** Obsługiwane przez ArchiCodeSyntaxErrorListener.

• **Błędy semantyczne i wykonania:** Wzbogacane o numer linii, kolumny, a także stos wywołań funkcji (FunctionCallStack). Wprowadzono limit rekurencji, aby zapobiec zawieszeniu interpretera.

## 7. Przykładowy kod źródłowy w języku ArchiCode

```
blueprint Silnia(int a) delivers float result = 1 {
    repeat 2 a + 1 {
        result = result * step
    }
    show "factorial(" a ") = " result
}
Core() delivers int coreResult {
```

```
    define float x = 0
    show "Podaj liczbę:"
    request x
    Silnia( (int)x ) // Wywołanie z jawnym rzutowaniem
}
```

## 8. Możliwe rozszerzenia

• Dodanie instrukcji break i continue.

• Wprowadzenie złożonych struktur danych (tablice, listy).

• System modułów i importów (import).

• Debugger krokowy.

## 9. Wnioski

Implementacja interpretera ArchiCode jest kompletnym, działającym projektem, który z powodzeniem realizuje postawione cele. Zastosowane rozwiązania, jak izolacja kontekstu wykonania funkcji, czynią architekturę interpretera solidną i rozszerzalną.