Specyfikacja projektu: Symulacja Filtra Cyfrowego

Piotr Pokornowski Alicja Misterka

15 maja 2024

Opis ogólny

Projekt zakłada stworzenie aplikacji do symulacji filtrów cyfrowych używanych do przetwarzania sygnałów audio. Aplikacja będzie napisana w języku Java i umożliwi użytkownikowi składanie układu filtru z poszczególnych bloków, takich jak filtry dolnoprzepustowe (LP), górnoprzepustowe (HP) itp.

Funkcjonalności

1. **Składanie układu filtru:**
   - Użytkownik będzie mógł składać układ filtru z różnych bloków, takich jak filtry LP, HP, itd.
   - Układ filtru będzie można konfigurować, dodając, usuwając lub modyfikując poszczególne bloki.
2. **Łączenie bloków:**
   - Umożliwienie łączenia bloków ze sobą w celu uzyskania bardziej złożonych filtrów.
   - Bloki będą można łączyć w sposób modularny, tak jak klocki.
3. **Wyświetlanie charakterystyki amplitudowej i odpowiedzi impulsowej filtru:**
   - Aplikacja będzie umożliwiała wyświetlanie charakterystyki amplitudowej i odpowiedzi impulsowej zdefiniowanego przez użytkownika filtru.
   - Charakterystyka amplitudowa będzie pokazywała, jak filtr wpływa na amplitudę sygnału w zależności od częstotliwości.
   - Odpowiedź impulsowa będzie pokazywała, jak filtr reaguje na impulsowe zmiany w sygnale wejściowym.
4. **Wyświetlanie widma sygnału wejściowego i wyjściowego:**
   - Aplikacja umożliwi wyświetlanie widma częstotliwościowego sygnału wejściowego i wyjściowego po przefiltrowaniu.
   - Użytkownik będzie mógł porównać widmo sygnału przed i po filtracji.
5. **Wczytywanie plików dźwiękowych w formacie WAV:**
   - Aplikacja będzie umożliwiała wczytywanie plików dźwiękowych w formacie WAV jako sygnał wejściowy.
6. **Eksport sygnału po przefiltrowaniu w formacie WAV:**
   - Po przefiltrowaniu, użytkownik będzie mógł wyeksportować sygnał do pliku w formacie WAV.

Interfejs użytkownika

Interfejs użytkownika będzie składał się z następujących elementów:

- Panel z listą dostępnych bloków filtrów.
- Obszar roboczy, gdzie użytkownik będzie mógł tworzyć i konfigurować układ filtru.
- Przyciski do wyświetlania charakterystyki amplitudowej i odpowiedzi impulsowej filtru.
- Przyciski do wczytywania i eksportu plików dźwiękowych.
- Widok widma sygnału wejściowego i wyjściowego.

Struktura projektu

Klasa `FilterBlock`

```
public abstract class FilterBlock {
    public abstract double[] process(double[] inputSignal);
    public abstract double[] getMagnitudeResponse();
    public abstract double[] getImpulseResponse();
}
```

Klasa `LowPassFilter`

```
public class LowPassFilter extends FilterBlock {
    // Konstruktor, metody przetwarzania, charakterystyki filtru
}
```

Klasa `HighPassFilter`

```
public class HighPassFilter extends FilterBlock {
    // Konstruktor, metody przetwarzania, charakterystyki filtru
}
```

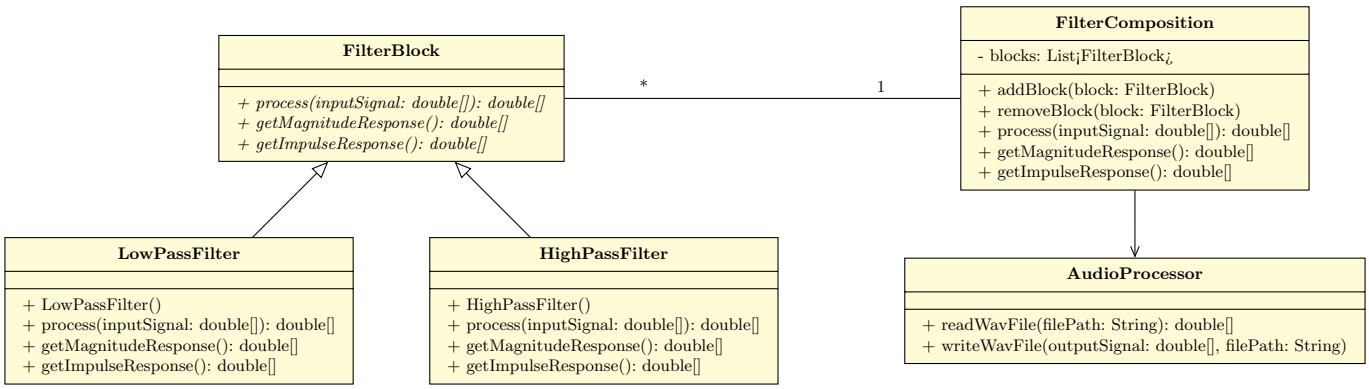
Klasa `FilterComposition`

```
public class FilterComposition {
    private List<FilterBlock> blocks;

    public void addBlock(FilterBlock block);
    public void removeBlock(FilterBlock block);
    public double[] process(double[] inputSignal);
    public double[] getMagnitudeResponse();
    public double[] getImpulseResponse();
}
```

Klasa `AudioProcessor`

```
import javax.sound.sampled.AudioInputStream;

public class AudioProcessor {
    public static double[] readWavFile(String filePath);
    public static void writeWavFile(double[] outputSignal, String filePath);
}
```

Diagram UML


FilterBlock
+ process(inputSignal: double[]): double[]
+ getMagnitudeResponse(): double[]
+ getImpulseResponse(): double[]
*
1
FilterComposition
- blocks: List¡FilterBlock¿
+ addBlock(block: FilterBlock)
+ removeBlock(block: FilterBlock)
+ process(inputSignal: double[]): double[]
+ getMagnitudeResponse(): double[]
+ getImpulseResponse(): double[]
LowPassFilter
+ LowPassFilter()
+ process(inputSignal: double[]): double[]
+ getMagnitudeResponse(): double[]
+ getImpulseResponse(): double[]
HighPassFilter
+ HighPassFilter()
+ process(inputSignal: double[]): double[]
+ getMagnitudeResponse(): double[]
+ getImpulseResponse(): double[]
AudioProcessor
+ readWavFile(filePath: String): double[]
+ writeWavFile(outputSignal: double[], filePath: String)