

Warszawa, 20.01.2021

Politechnika Warszawska
Wydział Mechatroniki

Programowanie w Java (PJAVA)
PROJEKT

GENERATOR SYGNAŁÓW PODSTAWOWYCH Z MOŻLIWOŚCIĄ
DODANIA SZUMÓW (PROSTKĄTNY, SINUS, TRÓJKĄT, DELTA) WRAZ
Z ANALIZĄ KORELACJI

Prowadzący: mgr inż. Iryna Gorbenko
Wykonawcy: Aleksandra Krakowiak, Joanna Rancew

Wprowadzenie:

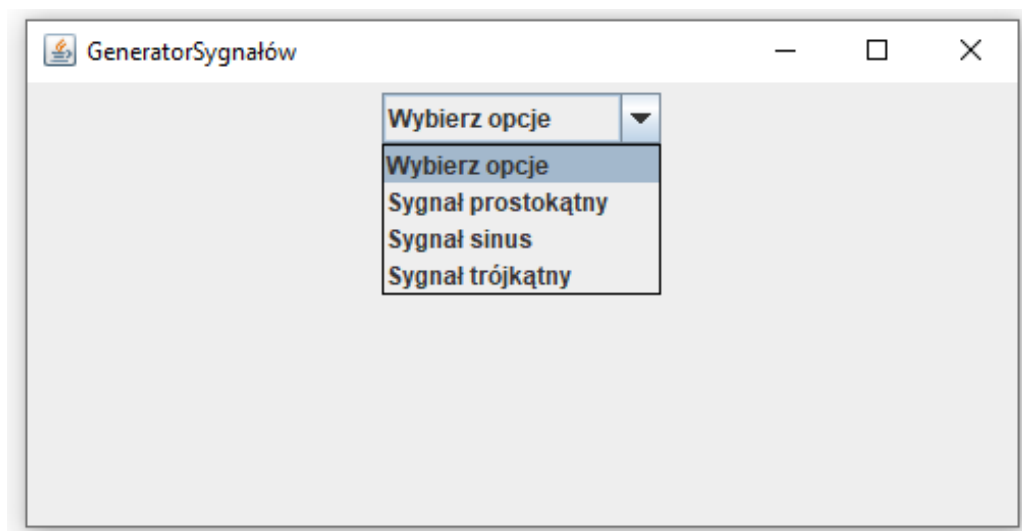
Generator sygnałów to aplikacja do tworzenia i wyświetlania sygnałów podstawowych oraz sygnałów zaszumionych. Użytkownik ma wpływ na parametry tworzącego się sygnału oraz ma możliwość porównywania tych samych sygnałów, ale o innych parametrach.

Opis programu:

Program pobiera od użytkownika powyższe parametry:

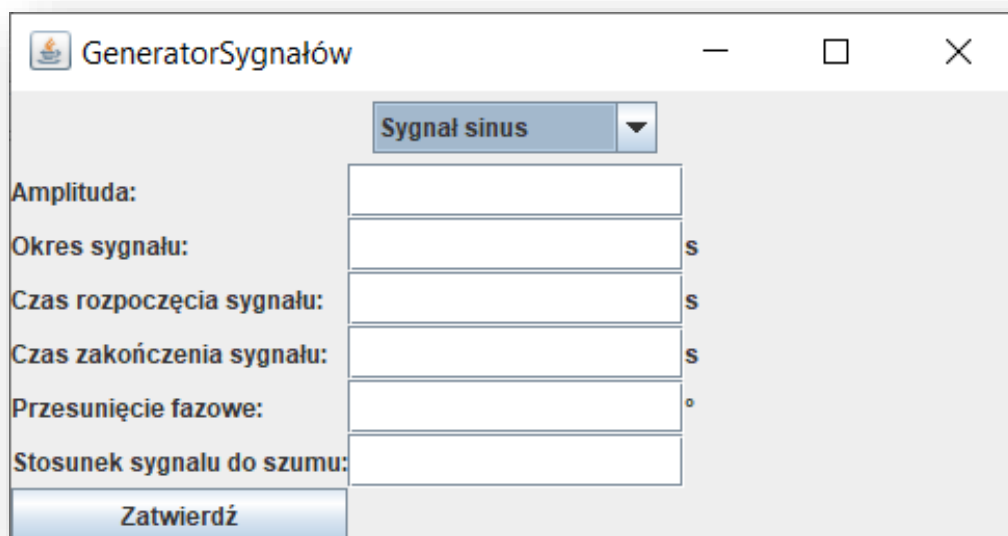
- typ sygnału (prostokątny, sinus, trójkątny - programie odpowiadający im parametr typu int),
- wartość amplitudy,
- okres sygnału (w sekundach),
- czas rozpoczęcia sygnału (w sekundach),
- czas zakończenia sygnału (w sekundach),
- wartość stosunku sygnału do szumu,
- wartość przesunięcia fazowego (w stopniach, jedynie dla sinusoidy).

Podczas uruchomienia programu tworzy się panel główny (Rys. 1.), który odpowiada za wyświetlenie opcji do wyboru sygnału przez użytkownika.



Rys. 1 Widok główny - wybór typu sygnału

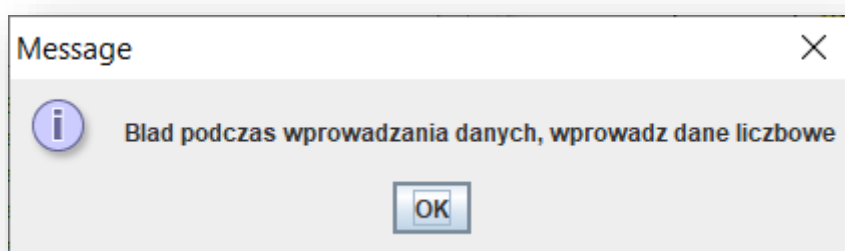
W wyniku wybrania jednego z trzech opcji, wygląd panelu głównego zmienia się w zależności od wybranej opcji. Przykładowy wygląd panelu dla wyboru *Sygnal sinus* został przedstawiony na Rys. 2.



The screenshot shows a window titled "GeneratorSygnałów" with a standard Windows title bar (minimize, maximize, close buttons). Inside the window, there is a dropdown menu currently set to "Sygnal sinus". Below this, there are six input fields with labels to their left: "Amplituda:", "Okres sygnału:", "Czas rozpoczęcia sygnału:", "Czas zakończenia sygnału:", "Przesunięcie fazowe:", and "Stosunek sygnału do szumu:". Each input field has a unit indicator to its right: "s" for the first four and "°" for the fifth. At the bottom of the window is a button labeled "Zatwierdź".

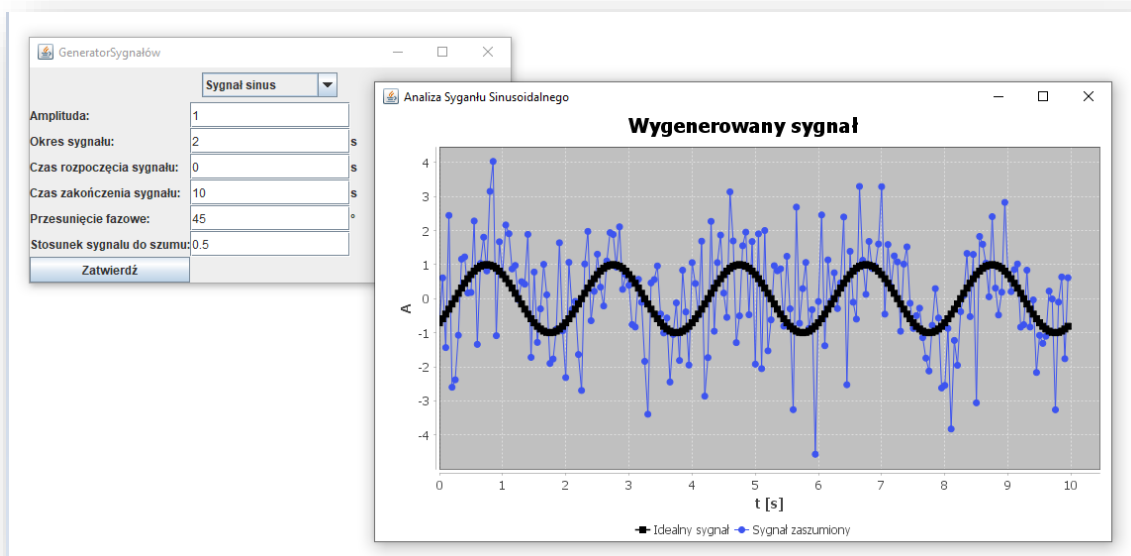
Rys. 2 Widok panelu wprowadzania parametrów sygnału

Po wyświetleniu się powyższego panelu użytkownik wpisuje wartości do pól dla wymienionych parametrów. Po wpisaniu wartości parametrów przez użytkownika, użytkownik powinien zatwierdzić swoje dane klikając przycisk „Zatwierdź”. Jeśli dane nie spełniają warunków, które zostały określone wewnątrz kodu (np. zostanie wprowadzony błędny format danych lub jakiś parametr pozostanie pusty) to pojawia się komunikat (Rys. 3). Oznacza to, że program jest odporny na błędy podczas tworzenia sygnału.



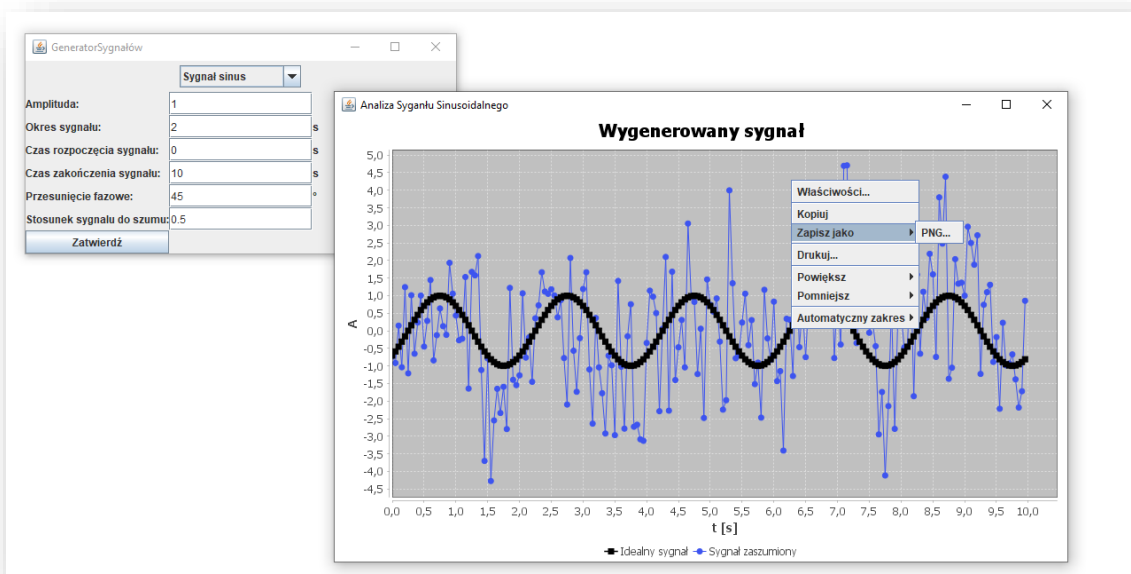
Rys. 3 Komunikat o błędzie, wyświetlany przy błędnym formacie danych lub ich braku

Jeśli wszystkie dane są poprawne, tzn. nie pojawił się komunikat błędu to tworzy się wykres przedstawiający sygnał o danych parametrach wraz z zaszumionym sygnałem (Rys. 4.)



Rys. 4. Wygenerowany sygnał w wyniku podania przez użytkownika poprawnych danych

Program posiada również opcje zapisu wykresu w formacie PNG. Najeżdżając na wykres myszką i klikając prawym przyciskiem myszy pojawia się MENU, gdzie mamy opcje *Zapisz jako* (Rys. 5). Wyświetlone MENU, posiada opcje *Powiększ*, *Pomniejsz*, które obsługują zwiększanie lub pomniejszanie skali osi.



Rys. 5. Prezentacja zapisu wykresu

Jeśli chcemy zakończyć prace aplikacji naciskamy przycisk „X” na okienku, gdzie zostały prowadzone dane. W efekcie cały program się zamyka.

Wymagane testy:

Testy poprawności wprowadzanych wartości:

- test typu wprowadzania danych (dane liczbowe), używając struktury try-catch
- test poprawności wprowadzonych danych (dane dodatnie), używając struktury if-else
- test poprawności wprowadzonych danych (porównanie dwóch wartości, używając struktury if-else)

Każdorazowo użytkownik otrzymuje komunikat z komentarzem o błędzie.

Klasy:

KLASA	OPIS	KOMENTARZ
Sygnal	Klasa odpowiadająca za sygnał	
Szum	Klasa odpowiadająca za szum	Sygnał ten jest generowany losowo z uwzględnieniem SNR, czyli stosunku sygnału do szumu
WykresSygnal	odpowiada za wyświetlania przetworzonych informacji	Tworzony przy użyciu biblioteki <i>JFreeChart</i>
GeneratorGUI	odpowiada za interfejs aplikacji	Tworzony przy użyciu biblioteki Swing
GeneratorSygnałow	odpowiada za uruchomienie programu	

Algorytmy i działanie programu:

Tworzenie Sygnału:

Sygnał tworzymy przy pomocy informacji pobranych od użytkownika (Rys. 2.). Na początku program pobiera od użytkownika informacje jaki sygnał będzie tworzony. Każdy sygnał ma przyporządkowaną swój typ przedstawiony jako zmienną *int*, która jest przekazywana do programu. Do stworzenia sygnału potrzeba jest wartość amplitudy (*double*), wartość okresu (*double*), wartość rozpoczęcia sygnału (*double*), wartość zakończenia sygnału (*double*), wartość parametru SNR (*double*) oraz dla sygnału sinusoidalnego potrzebna jest jeszcze wartość przesunięcia fazowego (*double*). Na podstawie pobranych wartości parametrów oraz wykorzystania funkcji matematycznych przy użyciu klasy *Math* wyliczane są kolejne wartości sygnału. Poniżej zaprezentowano tworzenie kolejnych punktów sygnału dla sinusa, sygnału trójkątnego i prostokątnego.

```

for(int i= 0; i<liczbaElementow; i++){
    tablicaWartosci[i]= this.getAmplituda()*Math.sin(2*Math.PI/this.getOkres()*tablicaArgumentow[i]-Math.toRadians(this.getPrzesFaz()));
    wykresSzum.add(tablicaArgumentow[i],tablicaWartosci[i]+s.getWartosc(i));
    wykresBrakSzumu.add(tablicaArgumentow[i],tablicaWartosci[i]);
}

```

```

for(int i= 0; i<liczbaElementow; i++){
    tablicaWartosci[i] = (2*this.getAmplituda()/Math.PI)*Math.asin(Math.sin(2*Math.PI*tablicaArgumentow[i]/this.getOkres()));

    wykresSzum.add(tablicaArgumentow[i],tablicaWartosci[i]+s.getWartosc(i));
    wykresBrakSzumu.add(tablicaArgumentow[i],tablicaWartosci[i]);
}

```

```

for(int i= 0; i<liczbaElementow; i++){
    tablicaWartosci[i] = Math.signum(Math.sin(2*Math.PI/this.getOkres()*tablicaArgumentow[i]))*this.getAmplituda();

    wykresSzum.add(tablicaArgumentow[i],tablicaWartosci[i]+s.getWartosc(i));
    wykresBrakSzumu.add(tablicaArgumentow[i],tablicaWartosci[i]);
}

```

Tworzenie Szumu:

Szum tworzymy przy pomocy informacji pobranej od użytkownika. Szum zależny jest od parametru SNR (*double*), który odpowiada za stosunek sygnału do szumu. W zależności, jaką wartość poda użytkownik, wyliczana jest amplituda Szum. Do wyliczenia wartości kolejnych punktów sygnału Szum, skorzystano z funkcji *nextGaussian()*, która jest metodą zdefiniowaną w klasie *Random*. Odpowiada ona za wyliczenie losowych wartości w zakresie (0,1). Następnie wyliczone wartości pomnożono, przez wyliczoną wartość amplitudy.

```

public Szum (double A, int n, double SNR){
    double amp=(A*A)/SNR;
    this.AmpSzum=Math.sqrt(amp);
    this.n = n;
    this.wartosci = new double[n];

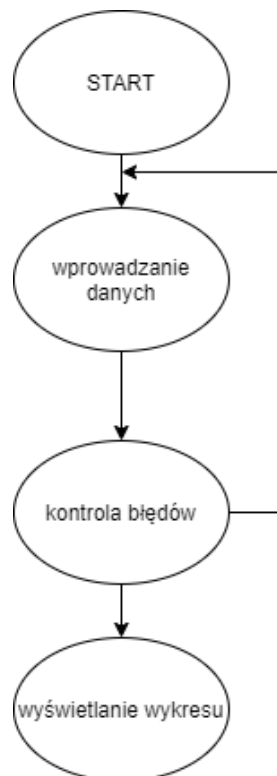
    for(int i = 0; i < n; i++){
        Random r = new Random();
        this.wartosci[i] = r.nextGaussian()*this.AmpSzum;
    }
}

```

Tworzenie interfejsu (GeneratorGui):

GeneratorGui tworzy się przy pomocy biblioteki *Swing*. Dziedziczy po klasie *JFrame* oraz odpowiada za wykonywane akcje. Po wyborze typu sygnału, generowany jest odpowiedni widok GUI do wprowadzenia danych. Po wprowadzeniu danych i kontroli błędów, w nowym oknie wyświetla się wykres.

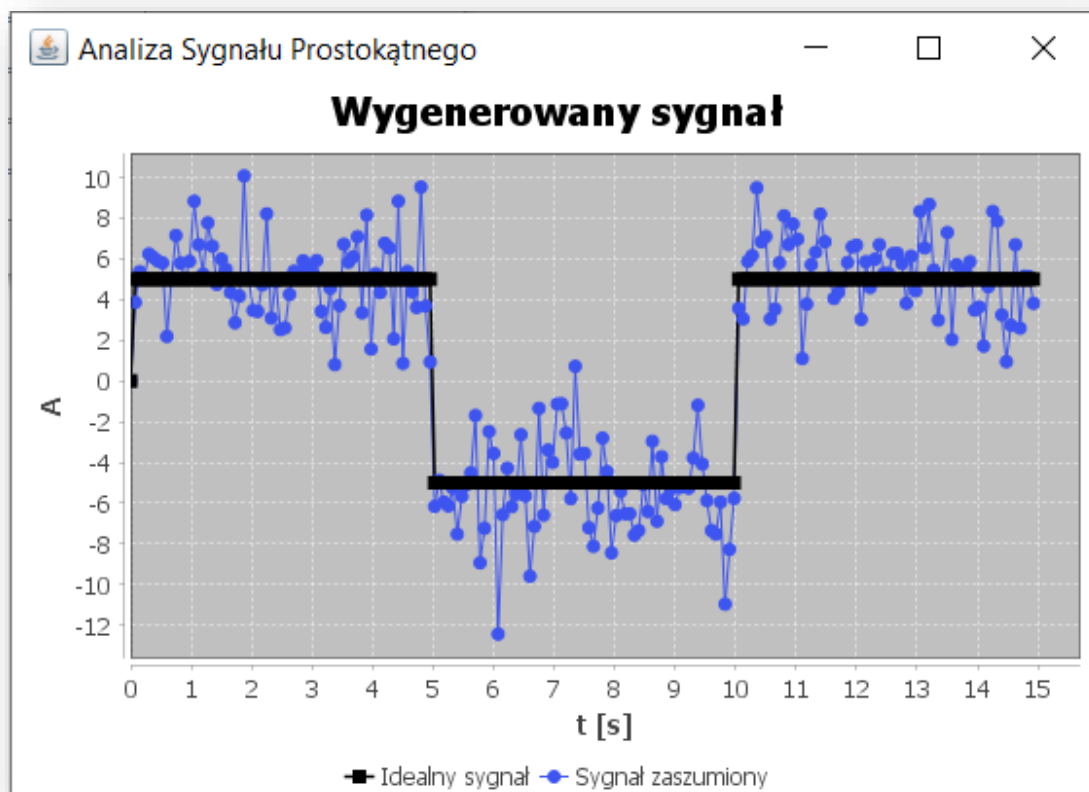
Algorytm wprowadzania:



Rys. 6. Algorytm wprowadzania

Wyświetlanie wykresu:

Wyświetlanie wykresu obejmuje wyświetlenie 200 punktów, których częstotliwość próbkowania zależy od wyświetlanego zakresu czasu. Na tej podstawie, korzystając ze wzorów dla sygnałów próbkowanych, wyświetlimy odpowiednie punkty, które dają obraz tego obszaru sygnału.



Rys. 7. Prezentowanie wyświetlanego sygnału prostokątnego o parametrach:
amplituda - 5, okres - 10 s, czas rozpoczęcia - 0 s, czas zakończenia - 15 s, SNR - 6

GUI:

Założenia:

Interfejs złożony jest z wyboru, który sygnał będziemy tworzyć oraz z pól, gdzie użytkownik będzie wpisywał odpowiednie parametry. Uwzględniony będzie również wybór dodania szumu do sygnału oraz analizy korelacji. Interfejs będzie pozwalał także na reset oraz zakończenie pracy programu.

Realizacja:

Interfejs złożony jest z wyboru, który sygnał będziemy tworzyć oraz z pól, gdzie użytkownik będzie wpisywał odpowiednie parametry. Szum zawsze będzie generowany i wartość jego amplitudy zależy od parametru SNR, o którym decydować będzie użytkownik. Za zakończenie pracy programu wywołuje się poprzez zamknięcie okna interfejsu („X”).

Interfejs oparty jest na klasie *CardLayout*. Klasa ta zarządza składnikami – 2 lub więcej (JPanel), które dzielą tę samą przestrzeń wyświetlacza. Dzięki czemu pozwala użytkownikowi wybierać między komponentami za pomocą „suwaka” - *JComboBox*. W poniższym kodzie pokazano deklarowanie opcji wyboru, tworzenie „kart” oraz ich dodanie do panelu głównego.

```
JPanel wybor;  
  
// opcje wyboru generowanego sygnału  
final static String OPCJE = "Wybierz opcje";  
final static String SYGNAPROSTO = "Sygnał prostokątny";  
final static String SYGNASIN = "Sygnał sinus";  
final static String SYGNATROJ = "Sygnał trójkątny";  
final static String RYSUNEK = "Generowany sygnał";  
  
(...)  
  
JPanel wyborOpcji = new JPanel();  
  
(...)  
  
//Panel dla sygnału Prostokątnego  
JPanel sygProsto = new JPanel(new GridLayout(0,3));  
  
(...)  
  
//Panel dla sygnału Sinusa  
JPanel sygSin = new JPanel(new GridLayout(0,3));  
  
(...)  
  
//Panel dla sygnału Trójkątnego  
JPanel sygTroj = new JPanel(new GridLayout(0,3));  
  
(...)  
  
wybor = new JPanel(new CardLayout());  
wybor.add(wyborOpcji, OPCJE);  
wybor.add(sygProsto, SYGNAPROSTO);  
wybor.add(sygSin, SYGNASIN);  
wybor.add(sygTroj, SYGNATROJ);
```

Aby wybierać między komponentami należy je zdefiniować w poniższy sposób:

```
//tworzenie listy wyboru  
JPanel PanelListaWyboru = new JPanel();  
String listaWyboru[] = {OPCJE, SYGNAPROSTO, SYGNASIN, SYGNATROJ};  
JComboBox lista = new JComboBox(listaWyboru);  
lista.setEditable(false);  
lista.addItemListener(this);  
PanelListaWyboru.add(lista);  
  
(...)  
  
w.add(PanelListaWyboru, BorderLayout.PAGE_START);  
w.add(wybor, BorderLayout.CENTER);
```

gdzie w to kontener, który przechowuje komponenty z listy.

Poniższa metoda jest odpowiedzialna za przetwarzanie elementu listy, który został wybrany:

```
@Override
public void itemStateChanged(ItemEvent ie) {
    CardLayout cl = (CardLayout) wybor.getLayout();
    cl.show(wybor, (String)ie.getItem());
}
```

Do stworzenia i wyświetlania utworzonego interfejsu odpowiada poniższa metoda:

```
/**
 * Metoda odpowiedzialna za tworzenie interfejsu
 */
public static void tworzenieInterfejsu(){
    JFrame frame = new JFrame("GeneratorSygnałów");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    GeneratorGui gs = new GeneratorGui();
    gs.dodOpcjiDoPanelu(frame.getContentPane());

    frame.pack();
    frame.setVisible(true);
}
```

Podczas tworzenia „kart” użyto klasy *GridLayout*, która odpowiada za rozmieszczenie elementów. Poniży kod opisuje stworzenie obiektów oraz ich rozmieszczenie.

```
//Panel dla sygnału Prostokątnego
JPanel sygProsto = new JPanel(new GridLayout(0,3));

JLabel ampP = new JLabel("Amplituda:");
this.ampProsto = new JTextField();
JLabel jedlP = new JLabel("");
sygProsto.add(ampP);
sygProsto.add(this.ampProsto);
sygProsto.add(jedlP);

JLabel okP = new JLabel("Okres sygnału:");
this.okrProsto = new JTextField();
JLabel jed2P = new JLabel("s");
sygProsto.add(okP);
sygProsto.add(this.okrProsto);
sygProsto.add(jed2P);
```

Wywołany konstruktor *GridLayout* ma dwa parametry (0,3). Pierwszy parametr opisuje ilość wierszy (w naszym przypadku jest to 0, ponieważ oznacza to dowolna ilość wierszy). Drugi parametr odpowiada, ile będzie kolumn (wszystkie kolumny mają takie same rozmiary). W naszym kodzie są trzy kolumny. Pierwsza kolumna odpowiada za nazewnictwo atrybutu sygnału (korzystamy z *JLabel*). Druga kolumna to pole tekstowe (*JTextField*), gdzie użytkownik będzie wprowadzał dane. Ostatnia kolumna odpowiada za jednostkę

wprowadzanego atrybutu (*JLabel*). Pokazany kod opisuje pierwsze dwa wiersze: wartość amplitudy i wartość okresu. W ostatnim wierszu znajduje się przycisk zatwierdzający wprowadzone dane.

```
this.zatProsto = new JButton("Zatwierdź");
sygProsto.add(this.zatProsto);
this.zatProsto.addActionListener(this);
```

Nasz interfejs odpowiada za wszystkie zdarzenia zachodzą w programie. Stworzone przyciski wykonują zdarzenia. Do wywoływania zdarzeń korzystamy z metody *actionPerformed(ActionEvent ae)*, która jest zawarta w klasie *ActionListener*.

```
@Override
public void actionPerformed(ActionEvent ae) {
    if (ae.getSource() == this.zatProsto) {
        try{
            System.out.println("Sygnał Prostokątny");
            String AString = this.ampProsto.getText();
            ampSyg = Double.parseDouble(AString);
            String OString = this.okrProsto.getText();
            okresSyg = Double.parseDouble(OString);
            String CRString = this.czRozProsto.getText();
            czasRozSyg = Double.parseDouble(CRString);
            String CZString = this.czZakProsto.getText();
            czasZakSyg = Double.parseDouble(CZString);
            rodzaj = 1;
            String snrString = this.snrProsto.getText();
            snrSyg = Double.parseDouble(snrString);
```

W tym przykładzie przycisk „Zatwierdź” odpowiada za pobranie informacji z pól tekstowych, które zostały wprowadzone przez użytkownika.

Biblioteki

W projekcie wykorzystano biblioteki:

- *Swing* – odpowiedzialna za tworzenie interfejsu graficznego,
- *JFreeChart* – odpowiedzialna za tworzenie i wyświetlanie wykresów.

Swing jest to typowa biblioteka używana do tworzenia interfejsu graficznego. W oparciu o layouty rozmieszczamy różne komponenty, które spełniają odpowiednie funkcje w interfejsie.

JFreeChart to biblioteka opensource opracowana w Javie. Korzysta się z niej, aby tworzyć wykresy w aplikacjach napisanych w języku JAVA. Biblioteka JFreeChart umożliwia tworzenie wykresów 2D i 3D, takie jak wykres kołowy, wykres słupkowy, wykres liniowy, wykres XY.

Biblioteka cechuje się dużym wyborem generowanych wykresów, umożliwia zapis tych wykresów w formacie PNG, JPEG. Dodatkowo jest bardzo prosta w użyciu. Dużym plusem używania tej biblioteki jest fakt, że umożliwia tworzenie wykresów na podstawie plików .txt oraz połączeń z bazą danych.

Użycie Biblioteki JFreeChart w aplikacji:

```
/** Konstruktor klasy WykresSygnal
 *
 * @param title   tytuł okienka wyświetlającego wykres
 * @param database wyświetlane dane
 */
public WykresSygnal(String title, XYDataset database) {
    super(title);
    JFreeChart wykSygnal = ChartFactory.createXYLineChart(
        "Wygenerowany sygnał",
        "t [s]",
        "A",
        database,
        PlotOrientation.VERTICAL,
        true, true, false);
    ChartPanel chartPanel = new ChartPanel(wykSygnal);
    chartPanel.setPreferredSize(new java.awt.Dimension(560, 367));
    final XYPlot plot = wykSygnal.getXYPlot();

    XYLineAndShapeRenderer renderer = new XYLineAndShapeRenderer();
    renderer.setSeriesPaint( 0 , Color.BLACK );
    renderer.setSeriesPaint( 1 , new Color(61, 84, 240) );
    renderer.setSeriesStroke( 0 , new BasicStroke(1.5f) );
    renderer.setSeriesStroke( 1 , new BasicStroke( 1.0f ) );
    plot.setRenderer( renderer );
    setContentPane( chartPanel );

    try{
        ChartUtils.saveChartAsPNG(new File("wykresAnalizy.png"), wykSygnal, 800, 600);
    }catch (IOException e){
        JOptionPane.showMessageDialog(null,"Błąd z zapisem");
    }
}
```

WykresSygnal to konstruktor klasy WykresSygnal, który ma parametry *title* (odpowiada za nazwę okna, w którym utworzy się wykres) oraz *XYDataset* (odpowiada za wprowadzone dane, z których utworzy się wykres). Tworzymy wykres przy pomocy klasy *JFreeChart*. Określamy, że jego typ to *XYLineChart* (typ wykres, gdzie mamy parę (X,Y) oraz korzystamy z faktu, że mamy dwie serie danych – Sygnal Czysty, Sygnal Zaszumiony), nadajemy mu podstawowe atrybuty: opis osi x, y oraz tytuł wykresu, określamy zbiór danych, orientację wykresu, wybieramy opcje wystąpienia legendy (jeśli mamy dwie serie danych jest to bardzo przydatne), ostatnie dwa atrybuty odpowiadają za tooltips oraz urls. Użyliśmy również klasy, która pomogła nam określić kolory wykresy, grubość linii. Utworzony wykres będzie kolejnym panelem. Wyświetlany wykres można zapisać naciskając prawym przyciskiem myszy w jego obrębie. Opcje zapisu umożliwiła nam metoda *saveChartAsPNG()*.

Analiza postawionych założeń oraz ich realizacji

Postawione założenie	Realizacja
Funkcjonalności GUI	GUI jest w pełni funkcjonalne oraz wygodne dla użytkownika.
Kontrola wprowadzanych danych	Kontrola w pełni zrealizowana, wraz z komunikatami do użytkownika.
Dziedziczenie klas sygnału – dziedziczenie klas po klasie bazowej Sygnal	Ze względu na złożoność klasy Sygnal, informacje o typie sygnału są przechowywane jako pola klasy Sygnal i w zależności od wartości stosowane są metody w inny sposób.
Możliwość dodania szumu – użytkownik będzie miał możliwość decydowania o tym, czy szum zostanie dodany.	Ze względu na lepsze prezentowanie różnic i działania programu, szum jest tworzony zawsze, z uwzględnieniem parametru SNR wybranego przez użytkownika.
Analiza korelacji sygnałów	Nie zrealizowaliśmy tej funkcji programu ze względu na złożoność matematyczną tej funkcji.
Wyświetlanie sygnałów – wyświetlanie próbkowanych 4 typów sygnałów w oparciu o wyliczone pojedyncze punkty wykresu.	Wyświetlanie próbkowanych 3 typów sygnałów (prostokątny, sinus, trójkątny) w oparciu o parametry wprowadzana przez użytkownika. Wyświetlany jest zaszumiony i czysty sygnał na jednym wykresie.
Dodatkowa funkcja	Możliwość zapisu generowanego sygnału do pliku .png lub delikatnej edycji wykresu (rozciągnięcie osi).

Literatura:

1. K. Snopek, J. Wojciechowski: Sygnały i systemy ZBIÓR ZADAŃ, OWPW 2010
2. docs.oracle.com/javase/tutorial/
3. <https://www.jfree.org/jfreechart/index.html>
4. <https://www.tutorialspoint.com/jfreechart/index.htm>