**Niezawodność i diagnostyka układów cyfrowych**

**System tolerujący uszkodzenia czujników z głosowaniem**

Autors:

Michał Nowicki(280944)

Wojciech Biskup(280987)

Anton Homan(282992)

Prowadzący:

dr hab. inż. Stanisław Piestrak

**Spis treści**

1. **Opis i cel symulacji………………………………………………3**
   1. Wstęp............................................................................................................................3
   2. Opis programu symulacji..............................................................................................3
2. **Wybór technologii..........................................................................4**
3. **Opis klass oraz ich działanie i przeznaczenie..............................5**

3.1 Class SimulationController…………………………….……………………………5

3.2 Class Parametr………………………………………………………………………7

3.3 Class MainViewController………………………………………………………….8

3.4 Class Main…………………………………………………………………………10

3.5 Class InputSignal……………………………….………………………………….12

3.6 Class ConsensusVoting…………………………………………………………….13

3.7 Class FormalizedMajorityVoting…………………………………………………..14

3.8 Class GeneralizedMedianVoting…………………………………….……………..16

3.9 Class VotingAlgorithm……………………………………………………………..18

3.10 Class WeightedAveragingAlgorithm………………………………………………19

3.11 Class Sensor…………………………………………………………….………….21

3.12 Class SensorTest………………………………………………….………………..23

3.13 Class BiasError…………………………………………………………………….24

3.14 Class ConstantValueError……………………………………………………….....26

3.15 Class DriftError…………………………………………………………………….27

3.16 Class ErrorModel…………………………………………………………………..29

3.17 Class IntermittentDropoutError……………………………...…………………….31

3.18 Class OscillatingError………………………………………………………...……33

3.19 Class RandomNoiseError………………………………………………………….35

**Opis i cel symulacji**

**Wstęp**

Program symulacji to system przeznaczony do testowania i analizowania czujników z różnymi modelami błędów. System umożliwia badanie zachowania czujników i analizowanie wpływu różnych czynników, takich jak szum, przesunięcie, awarie oraz inne rodzaje błędów. Głównym celem programu jest zapewnienie narzędzia do symulowania pracy czujników w warunkach rzeczywistych, uwzględniając różne zakłócenia, które mogą wystąpić podczas pracy sprzętu.

**Opis programu symulacji**

**1)Czujniki:** Czujniki to klasy, które implementują funkcjonalność pozyskiwania danych o środowisku. Każdy czujnik może być skonfigurowany z określonymi parametrami, takimi jak wysokość, temperatura i inne wskaźniki, które mogą być zmieniane za pomocą parametrów przekazywanych do modelu.

2) **Modele błędów:** Program korzysta z różnych modeli błędów, które nakładają zakłócenia na dane z czujników, symulując rzeczywiste sytuacje. Każdy model błędu może być skonfigurowany za pomocą parametrów i wprowadza odpowiednie zmiany do wartości czujnika. Oto główne typy błędów:

* **BiasError** (Błąd przesunięcia): Symuluje stałe przesunięcie w odczytach czujnika.
* **ConstantValueError** (Błąd stałej wartości): Zwraca stałą wartość dla czujnika, ignorując jego rzeczywiste odczyty.
* **DriftError** (Błąd dryfu): Symuluje stopniową zmianę odczytów czujnika w czasie.
* **IntermittentDropoutError** (Intermitentny błąd odczytu): Okresowo zwraca wartość zero, symulując awarie czujnika.
* **OscillatingError** (Błąd oscylacji): Tworzy oscylacje w odczytach czujnika z określoną amplitudą i częstotliwością.
* **RandomNoiseError** (Błąd szumu losowego): Dodaje szum losowy do odczytów czujnika w zależności od określonego zakresu wartości.

**3)Kontroler symulacji:** Program korzysta z kontrolera symulacji (SimulationController), który zarządza czasem i stanem czujnika, a także synchronizuje go z różnymi modelami błędów. Kontroler symulacji odpowiada za obliczanie błędów i zarządzanie parametrami w każdym kroku symulacji. Udostępnia metody do pobierania bieżącego czasu i wartości wysokości czujnika, a także do aktualizacji wartości parametrów.

**Wybór technologii**

W naszym projekcie do tworzenia interfejsu użytkownika wykorzystujemy technologię FXML. FXML to język znaczników w formacie XML, opracowany do opisu interfejsów graficznych na platformie JavaFX. Jak wykorzystujemy FXML w naszym projekcie:

- Definiowanie interfejsu: Wszystkie główne elementy interfejsu (np. przyciski, pola tekstowe, menu) są opisane w plikach FXML. To pozwala na efektywniejszą współpracę między projektantem interfejsu a programistą.

- Integracja z Java: Pliki FXML są powiązane z odpowiednimi kontrolerami napisanymi w języku Java, które obsługują zdarzenia i współpracują z logiką aplikacji. Jest to realizowane za pomocą adnotacji i mechanizmów wiązania danych JavaFX

**Opis klass oraz ich działanie i przeznaczenie**

**Class SimulationController**

**1)Opis classa**

SimulationController to kontroler symulacji, który zarządza i koordynuje procesem modelowania. Ta klasa służy do zarządzania czasem symulacji, stosowaniem różnych błędów na czujnikach oraz interakcją z innymi komponentami systemu, takimi jak czujniki i modele błędów. Zapewnia prawidłowe działanie całej symulacji, monitorując czas oraz inne parametry, stosując błędy i generując wyniki.

**2)Zmienne**

private static float time = 0f : Zmienna przechowująca bieżący czas symulacji. Czas ten jest wykorzystywany do śledzenia postępu symulacji i może być używany w obliczeniach

private static Sensor currentSensor: Zmienna przechowująca bieżący czujnik używany w symulacji. Ten czujnik może zmieniać się w zależności od warunków i parametrów symulacji.

private static float inputSignal = 0f: Zmienna przechowująca wartość sygnału wejściowego, który może zostać przekazany do czujnika do przetworzenia i dalszego modelowania błędów. Może to być początkowy sygnał, z którym pracuje symulator.

private static List<ErrorModel> appliedErrors = new ArrayList<>(): Lista błędów, które będą stosowane do czujników w trakcie symulacji. Błędy te mogą być różnego rodzaju, takie jak szum, dryf i inne zakłócenia, które modelują rzeczywiste zachowanie czujników.

private static VotingAlgorithm votingAlgorithm: Zmienna przechowująca algorytm głosowania, który jest stosowany do agregowania danych z wielu czujników. Może to być algorytm oparty na głosowaniu ważonym lub innym metodzie.

**3)Metody klasowe**

public static void setTime(float newTime): Metoda służąca do ustawienia nowej wartości czasu symulacji. Może być użyteczna do ręcznego zarządzania czasem symulacji, np. w przypadku pauz lub zmiany interwałów czasowych.

public static float getTime(): Metoda do pobrania bieżącego czasu symulacji. Umożliwia to innym częściom programu dostęp do aktualnego czasu symulacji do obliczeń i logiki.

public static void setInputSignal(float inputSignal): Metoda służąca do ustawienia nowego sygnału wejściowego, który będzie przekazywany do czujników. Może to być użyteczne do aktualizacji danych dostarczanych do systemu modelowania.

public static float getInputSignal(): Metoda do pobrania bieżącej wartości sygnału wejściowego. Może być używana do uzyskiwania aktualnych danych wejściowych w systemie.

public static void addErrorModel(ErrorModel errorModel): Metoda do dodania modelu błędu do listy błędów, które będą stosowane do czujników. Pozwala to na konfigurację różnych modeli błędów w celu symulacji rzeczywistych warunków pracy czujników.

public static void setVotingAlgorithm(VotingAlgorithm algorithm): Metoda do ustawienia algorytmu głosowania, który będzie używany do przetwarzania danych z czujników. Może to być algorytm oparty na głosowaniu ważonym lub innym typie agregacji danych.

public static float simulateStep(): Metoda do wykonania jednego kroku symulacji. Ta metoda aktualizuje stan systemu, w tym stosowanie błędów, agregację danych i zmianę czasu.

public static void resetSimulation(): Metoda do zresetowania symulacji. Może przywrócić wszystkie zmienne do początkowego stanu, w tym zresetować czas i sygnały wejściowe, co pozwala na ponowne uruchomienie symulacji od początku.

**Class Parametr**

**1)Opis classa**

**Parameter** to klasa, która reprezentuje pojedynczy parametr, który może być używany w różnych częściach programu, w tym w modelach błędów i czujnikach. Klasa ta przechowuje nazwę parametru, jego typ oraz opcjonalny opis. Dzięki tej klasie, parametry mogą być łatwo zarządzane, przechowywane i wykorzystywane w aplikacjach symulacyjnych lub innych systemach, które wymagają elastycznego zarządzania danymi.

**2)Zmienne**

private String name : Nazwa parametru. Służy do jednoznacznej identyfikacji parametru w ramach aplikacji. Jest to tekstowy identyfikator, który można wykorzystać do przypisania i odwoływania się do tego parametru w różnych częściach programu.

private Class<?> type: Typ parametru. Określa typ danych, jaki ma być przypisany do tego parametru.

private String description: Opis parametru. Jest to opcjonalna wartość, która może zawierać dodatkowe informacje na temat tego, co parametr reprezentuje, jak należy go używać, lub jak wpływa na działanie systemu.

**3)Metody klasowe**

public Parameter(String name, Class<?> type, String description): Konstruktor klasy Parameter. Umożliwia tworzenie nowych obiektów typu Parameter z określoną nazwą, typem danych oraz opisem. Konstruktor ten jest kluczowy dla inicjowania obiektów reprezentujących parametry w różnych częściach programu.

public String getName(): Metoda zwracająca nazwę parametru. Używana do uzyskania nazwy, która identyfikuje dany parametr w systemie.

public Class<?> getType(): Metoda zwracająca typ danych parametru. Używana do sprawdzenia, jakiego typu danych należy się spodziewać przy pracy z tym parametrem.

public String getDescription(): Metoda zwracająca opis parametru. Może być używana w celu uzyskania dodatkowych informacji o tym, jak parametr powinien być używany lub jak wpływa na działanie systemu.

**Class MainViewController**

**1)Opis classa**

MainViewController to klasa, która zarządza głównym interfejsem użytkownika (UI) aplikacji. Zawiera metody do inicjalizacji i aktualizacji interfejsu oraz do obsługi interakcji użytkownika z elementami graficznymi, takimi jak przyciski, pola tekstowe i inne komponenty UI.

**2)Zmienne**

private Button startButton: Przycisk "Start", który uruchamia wykonywanie programu lub symulacji po naciśnięciu. Element ten jest sterowany przez użytkownika i inicjuje działania związane z rozpoczęciem procesu.

private Button stopButton: Przycisk "Stop", który zatrzymuje bieżącą symulację lub wykonywanie programu. Również sterowany przez użytkownika, kończy proces.

private TextField inputField: Pole tekstowe, w które użytkownik może wprowadzić dane do symulacji lub programu. Element ten umożliwia interakcję z użytkownikiem, zbierając potrzebne informacje do dalszego przetwarzania.

private Label statusLabel: Etykieta wyświetlająca bieżący status pracy programu (np. "Działa", "Zatrzymano", błędy itp.). Etykieta ta jest aktualizowana w zależności od stanu aplikacji oraz interakcji użytkownika.

private SimulationController simulationController: Kontroler symulacji, który zarządza procesami związanymi z uruchamianiem, zatrzymywaniem i konfigurowaniem symulacji.

private boolean isSimulationRunning: Zmienna logiczna, która śledzi stan symulacji (czy działa, czy nie). Jest to flaga używana do określenia bieżącego stanu programu, na przykład do aktywacji lub dezaktywacji przycisków.

**3)Metody klasowe**

public void initializeUI(): Metoda do inicjalizacji wszystkich komponentów interfejsu (przycisków, pól tekstowych, etykiet itp.). Metoda ta jest wywoływana przy starcie aplikacji i ustawia początkowy stan wszystkich elementów UI.

public void onStartButtonClicked(): Metoda wywoływana po kliknięciu przycisku "Start". Uruchamia proces symulacji lub programu. Może również zmieniać stan interfejsu, na przykład blokować przyciski lub wyświetlać komunikaty.

void onStopButtonClicked(): Metoda wywoływana po kliknięciu przycisku "Stop". Zatrzymuje symulację lub proces programu i zmienia stan interfejsu (np. przywraca przyciski do stanu aktywnego).

public void updateStatus(String status): Metoda aktualizująca status programu, który jest wyświetlany w etykiecie statusLabel. Służy do informowania użytkownika o bieżącym stanie programu (np. zakończeniu symulacji lub błędach).

public void onInputChanged(): Metoda, która śledzi zmiany w polu tekstowym. Może być wywoływana po zmianie tekstu w polu inputField i aktualizować stan programu zgodnie z nowymi danymi.

public void setSimulationController(SimulationController controller): Metoda do ustawienia kontrolera symulacji. Pozwala połączyć instancję klasy SimulationController z interfejsem, co daje możliwość zarządzania symulacją przez metody MainViewController.

private void updateUIComponents(): Prywatna metoda do aktualizacji stanu różnych komponentów UI, takich jak przyciski i etykiety, w zależności od bieżącego stanu programu lub symulacji.

public void handleError(String errorMessage): Metoda do obsługi błędów, które mogą wystąpić podczas wykonywania programu. Wyświetla komunikaty o błędach w interfejsie, informując użytkownika o występujących problemach.

**Class Main**

**1)Opis classa**

Main to główna klasa aplikacji, która uruchamia program. Zawiera metodę main, która jest punktem wejścia do aplikacji. Klasa ta odpowiada za inicjalizację i uruchomienie aplikacji, tworzenie instancji głównych komponentów, takich jak kontroler symulacji czy widok, oraz uruchomienie wątku GUI (interfejsu użytkownika). Jest to klasa startowa, która inicjuje wszystkie potrzebne zasoby i umożliwia rozpoczęcie działania programu.

**2)Zmienne**

private static SimulationController simulationController: Przechowuje instancję kontrolera symulacji, który zarządza procesami symulacji. Ta zmienna pozwala na komunikację pomiędzy interfejsem użytkownika a logiką aplikacji.

private static MainViewController mainViewController: Przechowuje instancję kontrolera widoku, który zarządza interfejsem użytkownika aplikacji. Zmienna ta umożliwia połączenie logiki (kontroler symulacji) z częścią wizualną (widok).

**3)Metody klasowe**

public static void main(String[] args): Główna metoda aplikacji. Jest to punkt wejścia do programu, który jest wywoływany podczas uruchamiania aplikacji. Inicjalizuje wszystkie komponenty, takie jak kontroler symulacji, kontroler widoku, i uruchamia interfejs użytkownika. Działa jako "centrum dowodzenia", łącząc wszystkie części aplikacji w jeden działający system.

private static void initializeSimulationController(): Prywatna metoda służąca do inicjalizacji kontrolera symulacji. Tworzy instancję klasy SimulationController i ustawia odpowiednie parametry lub dane wejściowe do symulacji.

private static void initializeMainViewController(): Prywatna metoda służąca do inicjalizacji kontrolera widoku. Tworzy instancję klasy MainViewController i przekazuje jej kontroler symulacji, umożliwiając interakcję z interfejsem użytkownika.

private static void startApplication(): Prywatna metoda, która uruchamia aplikację po dokonaniu inicjalizacji wszystkich komponentów. Może obejmować uruchomienie GUI i rozpoczęcie działania głównych procesów aplikacji.

**Class InputSignal**

**1)Opis classa**

InputSignal to klasa reprezentująca sygnał wejściowy w systemie symulacji. Służy do przechowywania wartości sygnału, który jest przekazywany do różnych elementów systemu, takich jak czujniki. Klasa ta może posiadać dodatkowe funkcje związane z przetwarzaniem i modyfikowaniem sygnału wejściowego w zależności od zastosowanych modeli błędów.

**2)Zmienne**

private float height: Zmienna przechowująca wartość wysokości, która jest sygnałem wejściowym. Może reprezentować jakąś mierzoną wartość, na przykład wysokość w przestrzeni.

private float frequency: Zmienna przechowująca częstotliwość sygnału wejściowego. Może być używana w kontekście symulacji, gdzie sygnał może zmieniać się z różną częstotliwością.  
private float amplitude: Zmienna przechowująca amplitudę sygnału wejściowego. Określa zakres wartości sygnału, które mogą być używane w różnych obliczeniach lub symulacjach.

**3)Metody klasowe**

public float getHeight(): Metoda zwracająca wartość wysokości sygnału wejściowego. Umożliwia dostęp do zmiennej height.

public void setHeight(float height): Metoda ustawiająca wartość wysokości sygnału wejściowego. Pozwala na modyfikację zmiennej height.

public float getFrequency(): Metoda zwracająca częstotliwość sygnału wejściowego. Umożliwia dostęp do zmiennej frequency.

public void setFrequency(float frequency): Metoda ustawiająca wartość częstotliwości sygnału wejściowego. Pozwala na modyfikację zmiennej frequency.

public float getAmplitude(): Metoda zwracająca amplitudę sygnału wejściowego. Umożliwia dostęp do zmiennej amplitude.

public void setAmplitude(float amplitude): Metoda ustawiająca wartość amplitudy sygnału wejściowego. Pozwala na modyfikację zmiennej amplitude.

public String toString(): Metoda nadpisana, która zwraca reprezentację tekstową obiektu InputSignal, zapewniającą łatwiejsze debugowanie i wyświetlanie danych.

**Class ConsensusVoting**

**1)Opis classa**

ConsensusVoting to klasa odpowiedzialna za implementację mechanizmu głosowania konsensusu w systemie symulacji. Używana jest do podejmowania decyzji na podstawie wyników głosowania od różnych uczestników (np. czujników), które mogą reprezentować różne wyniki pomiarów lub interpretacji sygnałów. Klasa ta może być używana w przypadku, gdy system wymaga konsensusu między wieloma komponentami, aby podjąć ostateczną decyzję.

**2)Zmienne**

private List<Float> sensorValues: Zmienna przechowująca listę wartości odczytanych z różnych czujników. Jest to zbiór danych, które będą analizowane w procesie głosowania konsensusu.

private int threshold: Zmienna określająca próg, który musi zostać osiągnięty lub przekroczony przez liczbę głosów, aby decyzja została uznana za konsensus. Jest to liczba określająca minimalną liczbę głosów, które muszą zgadzać się w ramach głosowania.

**3)Metody klasowe**

public ConsensusVoting(List<Float> sensorValues, int threshold): Konstruktor klasy. Inicjalizuje obiekt **ConsensusVoting** z listą wartości czujników oraz określa próg głosów, który musi zostać osiągnięty, aby podjąć decyzję.

public boolean evaluateConsensus(): Metoda oceniająca, czy konsensus został osiągnięty na podstawie wartości czujników i zadanego progu. Zlicza, ile czujników zgadza się na tę samą wartość, i porównuje to z progiem. Zwraca true, jeśli konsensus został osiągnięty, lub false, jeśli nie.

public float getConsensusValue(): Metoda zwracająca wartość, która została uznana za konsensus w wyniku głosowania. Zwykle jest to wartość, która najczęściej pojawia się w wynikach czujników, gdy osiągnięty jest konsensus.

public void setThreshold(int threshold): Metoda ustawiająca próg głosowania, który określa, ile czujników musi zgodzić się na tę samą wartość, aby konsensus został osiągnięty. Może być używana do dynamicznej zmiany progu podczas działania programu.

public List<Float> getSensorValues(): Metoda zwracająca listę wartości czujników, które są używane w procesie głosowania.

**Class FormalizedMajorityVoting**

**1)Opis classa**

FormalizedMajorityVoting to klasa implementująca algorytm głosowania oparty na formalnym podejściu do większościowego głosowania w systemach rozproszonych, takich jak sieci czujników. Klasa ta jest odpowiedzialna za podejmowanie decyzji na podstawie danych wejściowych, w których opcja uzyskująca większość głosów zostaje uznana za ostateczną. W odróżnieniu od prostych algorytmów głosowania, ten model może uwzględniać różne wagi głosów, a także zdefiniowane zasady dla uzyskania większości.

**2)Zmienne**

private List<Float> sensorValues: Lista wartości odczytanych z różnych czujników, które będą poddane analizie w procesie głosowania. Może to być zestaw wyników pomiarów, które uczestniczą w podejmowaniu decyzji.

private int majorityThreshold: Próg określający, jaka liczba głosów jest uznawana za większość. Zwykle wyrażany jako procent lub liczba głosów, które muszą zostać oddane na daną opcję, aby została ona uznana za zwycięską.

private boolean weightedVotingEnabled: Zmienna określająca, czy głosowanie powinno uwzględniać wagi przypisane poszczególnym głosom. Jeśli ustawiona na true, głosy będą traktowane według przypisanej wagi, co może zmieniać wynik głosowania.

private Map<Integer, Integer> voteCounts: Mapa, która przechowuje liczbę głosów oddanych na poszczególne opcje. Klucze mapy reprezentują różne opcje głosowania (np. wartości lub kategorie), a wartości to liczba głosów oddanych na każdą z opcji.

**3)Metody klasowe**

public FormalizedMajorityVoting(List<Float> sensorValues, int majorityThreshold, boolean weightedVotingEnabled): Konstruktor klasy, który inicjalizuje obiekt FormalizedMajorityVoting z listą wartości czujników, progiem większościowym oraz informacją o tym, czy głosowanie ma uwzględniać wagi.

public boolean evaluateMajority(): Metoda, która ocenia, czy któraś z opcji uzyskała większość głosów. Określa, czy liczba głosów oddanych na daną opcję przekracza próg większości (np. 51% wszystkich głosów).

public float getMajorityDecision(): Metoda zwracająca wartość, która została uznana za decyzję większościową. Zwykle jest to wartość lub opcja, która uzyskała największą liczbę głosów, spełniając wymagania progu większościowego.

public void setMajorityThreshold(int majorityThreshold): Metoda do ustawiania progu majority, który decyduje o tym, jaką liczbę głosów musi zdobyć opcja, by zostać uznana za większość.

public Map<Integer, Integer> getVoteCounts(): Metoda zwracająca mapę, która zawiera liczbę głosów oddanych na poszczególne opcje. Umożliwia to podgląd, które opcje były najczęściej wybierane przez czujniki.

public void addVotes(int option, int votes): Metoda umożliwiająca dodanie głosów na określoną opcję. Zwiększa liczbę głosów dla danej opcji w mapie voteCounts.

public void enableWeightedVoting(boolean enabled): Metoda do włączania lub wyłączania głosowania z wagami. Jeśli głosowanie z wagami jest włączone, to różne opcje głosowania mogą mieć różną wagę, wpływając na wynik głosowania.

**Class GeneralizedMedianVoting**

**1)Opis classa**

GeneralizedMedianVoting to klasa implementująca algorytm głosowania oparty na wyznaczaniu mediany wartości. Klasa ta jest odpowiedzialna za podejmowanie decyzji na podstawie danych wejściowych, w których mediana wartości zebranych od różnych czujników jest uznawana za wynik końcowy. Głosowanie to jest bardziej odpornie na skrajne wartości (outliers) niż głosowanie większościowe, ponieważ mediana nie jest tak wrażliwa na wartości ekstremalne.

**2)Zmienne**

private List<Float> sensorValues: Lista przechowująca wartości odczytów z czujników, które będą używane do obliczenia mediany. Dane te są podstawą do określenia końcowego wyniku głosowania.

private boolean sorted: mienna kontrolująca, czy lista wartości czujników została już posortowana. Wartość true oznacza, że lista jest posortowana, a false — że jeszcze nie została posortowana.

private float medianValue: Zmienna przechowująca obliczoną medianę z wartości czujników. To będzie ostateczny wynik głosowania.

**3)Metody klasowe**

public GeneralizedMedianVoting(List<Float> sensorValues): Konstruktor klasy, który inicjalizuje obiekt GeneralizedMedianVoting z listą wartości odczytów czujników. Następnie oblicza medianę z tych wartości.

public float getMedianValue(): Metoda zwracająca medianę z wartości czujników. Jeśli wartości czujników zostały już posortowane, zwraca medianę; w przeciwnym razie sortuje listę przed obliczeniem mediany.

private void calculateMedian(): Metoda pomocnicza, która sortuje listę wartości czujników, a następnie oblicza medianę na podstawie posortowanej listy. Jeśli lista ma liczbę nieparzystą, mediana to wartość środkowa, jeśli parzystą — średnia dwóch środkowych wartości.

private void sortSensorValues(): Metoda pomocnicza, która sortuje wartości czujników w rosnącej kolejności. Sortowanie jest wymagane do obliczenia mediany.

public void updateSensorValues(List<Float> newValues): Metoda umożliwiająca aktualizację listy wartości czujników. Po zaktualizowaniu wartości, mediana zostaje ponownie obliczona.

public String toString(): Metoda zwracająca reprezentację tekstową klasy, w tym przypadku przedstawiająca obliczoną medianę jako string.

**Class VotingAlgorithm**

**1)Opis classa**

VotingAlgorithm — jest to klasa abstrakcyjna, która pełni rolę bazową dla wszystkich algorytmów głosowania. Klasa ta definiuje ogólną strukturę i interfejs dla różnych metod głosowania, które mogą być implementowane w systemie. Celem tej klasy jest umożliwienie aplikacji pracy z różnymi strategiami głosowania (np. głosowanie większościowe, głosowanie na medianie), które dzielą wspólną funkcjonalność, ale różnią się konkretną metodą obliczania wyniku na podstawie odczytów z czujników lub głosów.

**2)Metody klasowe**

public abstract String getDisplayName(): Metoda abstrakcyjna, która musi być zaimplementowana w klasach dziedziczących. Powinna zwracać nazwę algorytmu głosowania w postaci łańcucha znaków.

public abstract String getDescription(): Metoda abstrakcyjna, która musi być zaimplementowana w klasach dziedziczących. Powinna zwracać opis algorytmu głosowania, wyjaśniając jego działanie i cel.

public abstract float calculateVotingResult(List<Float> sensorValues): Metoda abstrakcyjna, która musi być zaimplementowana w klasach dziedziczących. Służy do obliczenia wyniku głosowania na podstawie listy wartości z czujników. W zależności od konkretnego algorytmu, ta metoda będzie różnić się implementacją, ale zawsze będzie przyjmować listę wartości (np. wyników od czujników) i zwracać wynik głosowania w postaci liczby zmiennoprzecinkowej.

public abstract List<Parameter> getParameters(): Metoda abstrakcyjna, która powinna być zaimplementowana w klasach dziedziczących. Powinna zwracać listę parametrów, które są dostępne do ustawienia w kontekście algorytmu głosowania. Parametry mogą obejmować różne ustawienia, które kontrolują sposób działania algorytmu głosowania.

public abstract Map<String, Object> getParameterValues(): Metoda abstrakcyjna, która powinna zwracać mapę wartości parametrów przypisanych do instancji algorytmu głosowania. Mapa ta ma klucze będące nazwami parametrów i wartości przypisane do tych parametrów.

public abstract void setParameterValues(Map<String, Object> parameters): Metoda abstrakcyjna, która powinna pozwalać na ustawienie wartości parametrów algorytmu głosowania na podstawie mapy, która zawiera nowe wartości dla parametrów.

public abstract VotingAlgorithm getNewInstance(): Metoda abstrakcyjna, która powinna zwracać nową instancję klasy algorytmu głosowania. Jest to przydatne, gdy chcemy stworzyć nowy obiekt tego samego algorytmu.

public String toString(): Metoda zwracająca reprezentację tekstową algorytmu głosowania. Zwykle ta metoda wykorzystuje getDisplayName() do zwrócenia nazwy algorytmu w formie tekstowej.

**Class WeightedAveragingAlgorithm**

**1)Opis classa**

WeightedAveragingAlgorithm to klasa implementująca algorytm głosowania oparty na ważonym średnim (Weighted Averaging). Algorytm ten oblicza wynik głosowania na podstawie odczytów z różnych czujników, przypisując im różne wagi, które odzwierciedlają ich względną ważność. Wynik głosowania jest średnią ważoną wszystkich wartości z czujników, gdzie każda wartość jest pomnożona przez przypisaną jej wagę. Algorytm ten jest przydatny w sytuacjach, gdy różne źródła danych mają różną wiarygodność, a bardziej wiarygodne źródła powinny mieć większy wpływ na końcowy wynik.

**2)Zmienne**

private List<Float> weights: Lista przechowująca wagi przypisane do odczytów z czujników. Każdy element listy odpowiada wadze dla konkretnego czujnika. Wagi są używane do obliczania średniej ważonej wyników czujników.

private static final List<Parameter> parameters: Statyczna lista parametrów algorytmu. W tej liście znajdują się parametry, które mogą być używane w algorytmie głosowania, takie jak wagi przypisane do czujników.

**3)Metody klasowe**

public String getDisplayName(): Zwraca nazwę algorytmu głosowania — w tym przypadku "Weighted Averaging". Jest to tekstowa reprezentacja algorytmu, który implementuje średnią ważoną.

public String getDescription(): Zwraca krótki opis algorytmu głosowania. W tym przypadku opisuje on algorytm jako metodę głosowania, która wykorzystuje średnią ważoną wyników czujników.

public List<Parameter> getParameters(): Zwraca listę parametrów dostępnych dla algorytmu głosowania. Parametry te mogą być ustawiane przez użytkownika, aby kontrolować sposób działania algorytmu. W przypadku tej klasy, parametrami są wagi przypisane do czujników.

public Map<String, Object> getParameterValues(): Zwraca mapę, która zawiera wartości aktualnych parametrów algorytmu, w tym przypadku wagi przypisane do różnych czujników.

public void setParameterValues(Map<String, Object> parameters): Pozwala ustawić wartości parametrów algorytmu głosowania. Parametry przekazywane są w postaci mapy, której klucze to nazwy parametrów, a wartości to przypisane im wartości.

public float calculateVotingResult(List<Float> sensorValues): Główna metoda algorytmu, która oblicza wynik głosowania na podstawie wartości czujników. Działa to na zasadzie średniej ważonej, gdzie każda wartość czujnika jest mnożona przez przypisaną jej wagę, a następnie wyniki są sumowane i dzielone przez sumę wag.

public VotingAlgorithm getNewInstance(): Zwraca nową instancję algorytmu głosowania, pozwalając na utworzenie świeżego obiektu tej samej klasy.

public String toString(): Zwraca tekstową reprezentację algorytmu głosowania. Zwykle wykorzystuje metodę getDisplayName() do uzyskania nazwy algorytmu.

**Class Sensor**

**1)Opis classa**

Sensor — jest to abstrakcyjna klasa reprezentująca czujnik, który może zbierać dane z otoczenia i generować wartości. Klasa ta definiuje podstawowe operacje związane z parametrami czujnika, takie jak dostęp do parametrów, ustawianie wartości, a także metodę do pobierania danych z czujnika. Jest to klasa bazowa, z której mogą dziedziczyć różne konkretne implementacje czujników, takie jak czujniki do symulacji błędów czy testowe czujniki.

**2)Zmienne**

private static final List<Parameter> parameters: Jest to lista parametrów czujnika, która przechowuje dostępne parametry (np. jednostki miary, zakres, czy inne cechy czujnika). Ta zmienna jest statyczna, co oznacza, że jest współdzielona przez wszystkie instancje klasy.

private float height: Przechowuje wartość wysokości, która może być używana do przechowywania lub obliczania informacji specyficznych dla czujnika, jak np. pomiar odległości.

**3)Metody klasowe**

public String getDisplayName(): Jest to metoda abstrakcyjna, która powinna zostać zaimplementowana przez klasy dziedziczące. Zwraca nazwę czujnika, która jest wyświetlana w systemie (np. "Test Sensor").

public String getDescription(): Jest to metoda abstrakcyjna, która powinna zostać zaimplementowana przez klasy dziedziczące. Zwraca krótki opis działania czujnika (np. "Sensor for testing purposes").

public List<Parameter> getParameters(): Jest to metoda abstrakcyjna, która zwraca listę dostępnych parametrów dla czujnika. Parametry mogą obejmować różne ustawienia i właściwości czujnika, takie jak np. zakres, dokładność, jednostki pomiaru itp.

public Map<String, Object> getParameterValues(): Zwraca mapę, w której klucze to identyfikatory parametrów (np. nazwa parametru), a wartości to przypisane wartości tych parametrów (np. wartości liczbowe, takie jak zakres, czy dokładność).

public void setParameterValues(Map<String, Object> parameters): Umożliwia ustawienie wartości parametrów czujnika na podstawie mapy. Mapowanie parametrów pozwala na dynamiczną modyfikację właściwości czujnika w trakcie działania programu.

public float getHeight(): Zwraca wysokość czujnika, która jest przechowywana w zmiennej height. Może być używana do obliczeń, np. przy symulacji pomiaru odległości.

public void setHeight(float height): Umożliwia ustawienie wysokości czujnika. Możliwe jest, że ta wartość jest używana w obliczeniach związanych z wynikami pomiarów.

public Sensor getNewInstance(): Zwraca nową instancję czujnika. Metoda ta jest zazwyczaj implementowana przez klasy dziedziczące, które tworzą konkretne instancje czujników (np. czujniki testowe, czujniki z błędami).

**Class SensorTest**

**1)Opis classa**

SensorTest to klasa implementująca czujnik do celów testowych. Rozszerza ona abstrakcyjną klasę Sensor i zapewnia konkretne implementacje dla jej metod. Jest wykorzystywana głównie do testowania innych komponentów systemu, symulując czujnik z prostymi parametrami, takimi jak wysokość i wartość parametru testowego. Klasa ta jest również przykładem implementacji czujnika, który przechowuje parametry i umożliwia ich manipulację.

**2)Zmienne**

private static final ArrayList<Parameter> parameters: Lista parametrów czujnika testowego, która zawiera jeden parametr — testParam (parametr typu float). Jest to statyczna lista, która przechowuje definicję parametru, który można modyfikować i ustawiać w ramach testów.

private float test\_\_height: Zmienna przechowująca wysokość czujnika. Jest używana w metodzie getHeight(), gdzie może być wykorzystywana do obliczeń lub symulacji.

private float test\_\_param: Zmienna przechowująca wartość parametru testowego testParam, który może być ustawiany i modyfikowany przez użytkownika lub system w ramach testów.

**3)Metody klasowe**

public String getDisplayName(): Zwraca nazwę czujnika — w tym przypadku "Test Sensor". Jest to metoda, która nadpisuje metodę z klasy bazowej i dostarcza nazwę wyświetlaną w systemie.

public String getDescription(): Zwraca opis czujnika — w tym przypadku "Sensor for testing purposes". Jest to metoda nadpisująca metodę z klasy bazowej, dostarczająca krótki opis czujnika.

public ArrayList<Parameter> getParameters(): Zwraca listę parametrów dla czujnika testowego. W tym przypadku lista zawiera jeden parametr: testParam (typ Float), który jest używany do testowania wartości.

public Map<String, Object> getParameterValues(): Zwraca mapę, która zawiera aktualne wartości parametrów czujnika. W tym przypadku mapa zawiera jedną parę: "testParam" z przypisaną wartością test\_\_param.

public void setParameterValues(Map<String, Object> parameters): Ustawia wartości parametrów czujnika na podstawie mapy. W przypadku, gdy w mapie znajduje się klucz "testParam", wartość tego parametru jest przypisywana do zmiennej test\_\_param.

public Sensor getNewInstance(): Tworzy i zwraca nową instancję czujnika testowego. Jest to metoda nadpisująca metodę z klasy bazowej, która pozwala na dynamiczne tworzenie nowych obiektów tego samego typu.

public float getHeight(): Zwraca wartość wysokości czujnika, która jest obliczana przy pomocy metody calculateAppliedErrors z klasy SimulationController. Metoda ta symuluje wpływ błędów na wysokość odczytaną przez czujnik.

public void test\_\_setHeight(float height): Ustawia wysokość czujnika, przypisując wartość do zmiennej test\_\_height. Metoda ta jest używana w testach, aby ustawić wartość wysokości w celach testowych (do symulacji).

**Class BiasError**

**1)Opis classa**

BiasError to klasa implementująca model błędu, który symuluje stały offset (przesunięcie) w odczycie czujnika. Oznacza to, że każde odczytane przez czujnik dane są modyfikowane o stałą wartość (bias). Jest to typowy model błędu stosowany w symulacjach, gdzie czujnik ma systematyczny błąd w postaci przesunięcia w górę lub w dół, który wpływa na odczytywane wartości.

**2)Zmienne**

private float biasValue = 10f: Zmienna przechowująca wartość biasu (offsetu), która jest dodawana do każdego odczytu czujnika. Domyślnie wartość ta wynosi 10.0f. Jest to główny parametr, który definiuje "stały błąd" w systemie.

private static final ArrayList<Parameter> parameters:

Lista parametrów modelu błędu, która zawiera jeden parametr — biasValue (typ Float). Jest to lista statyczna, przechowująca definicję parametru, który użytkownik może ustawić w ramach konfiguracji modelu błędu.

**3)Metody klasowe**

public String getDisplayName(): Zwraca nazwę modelu błędu — w tym przypadku "Bias Error". Jest to metoda, która nadpisuje metodę z klasy bazowej ErrorModel i dostarcza nazwę wyświetlaną w systemie.

public String getDescription(): Zwraca opis modelu błędu — w tym przypadku "Simulates a constant offset in the sensor reading". Opis ten wyjaśnia, że model błędu symuluje stały offset (przesunięcie) w odczycie czujnika.

public ArrayList<Parameter> getParameters(): Zwraca listę parametrów dla modelu błędu. W tym przypadku lista zawiera jeden parametr: biasValue (typ Float), który definiuje wartość przesunięcia w odczytach czujnika.

public Map<String, Object> getParameterValues(): Zwraca mapę, która zawiera aktualne wartości parametrów modelu błędu. W tym przypadku mapa zawiera parę: "biasValue" z przypisaną wartością biasValue.

public void setParameterValues(Map<String, Object> parameters): Ustawia wartości parametrów modelu błędu na podstawie mapy. W przypadku, gdy w mapie znajduje się klucz "biasValue", wartość tego parametru jest przypisywana do zmiennej biasValue.

public float getErrorValue(float inputValue): Zwraca wartość z błędem, który jest obliczany na podstawie wartości parametru biasValue. Wartość inputValue (odczyt z czujnika) jest zwiększana o wartość biasValue, co symuluje stały błąd w odczycie.

public ErrorModel getNewInstance(): Tworzy i zwraca nową instancję modelu błędu BiasError. Jest to metoda nadpisująca metodę z klasy bazowej, która pozwala na dynamiczne tworzenie nowych obiektów tego samego typu.

**Class ConstantValueError**

**1)Opis classa**

ConstantValueError — jest to klasa implementująca model błędu, który przypisuje stałą wartość odczytu czujnika na podstawie parametru "constantValue". Oznacza to, że niezależnie od rzeczywistego odczytu czujnika, zwróci on zawsze określoną wartość. Jest to przydatne w sytuacjach, gdzie chcemy symulować sytuację, w której czujnik zawsze daje błędne, ale stałe odczyty.

**2)Zmienne**

private float constantValue = 10f: Zmienna przechowująca stałą wartość, która jest przypisywana do każdego odczytu czujnika. Domyślnie wynosi 10.0f. Każdy odczyt będzie modyfikowany do tej stałej wartości, niezależnie od prawdziwego odczytu.

private static final ArrayList<Parameter> parameters: Lista parametrów dla tego modelu błędu, która zawiera jeden parametr — constantValue (typ Float). Lista jest statyczna, a jej celem jest udostępnienie użytkownikowi możliwości dostosowania wartości tej stałej.

**3)Metody klasowe**

public String getDisplayName(): Zwraca nazwę modelu błędu, która jest wyświetlana w systemie. W tym przypadku jest to "Constant Value Error".

public String getDescription(): Zwraca opis modelu błędu. Opis informuje, że model błędu przypisuje stałą wartość do odczytu czujnika. Dla tego modelu czujnik zawsze zwraca ustaloną wartość, niezależnie od rzeczywistego pomiaru.

public ArrayList<Parameter> getParameters(): Zwraca listę parametrów, które mogą być przypisane do modelu błędu. W tym przypadku lista zawiera jeden parametr — constantValue, który ustala stałą wartość przypisywaną do odczytu czujnika.

public Map<String, Object> getParameterValues(): Zwraca mapę, która zawiera aktualne wartości parametrów. W tym przypadku mapa zawiera parę "constantValue" i przypisaną wartość zmiennej constantValue.

public void setParameterValues(Map<String, Object> parameters): Ustawia wartości parametrów modelu błędu na podstawie mapy. Jeśli mapa zawiera klucz "constantValue", to przypisuje wartość tego parametru do zmiennej constantValue. Metoda ta pozwala na dynamiczną zmianę wartości stałego błędu.

public float getErrorValue(float inputValue): Zwraca stałą wartość (zdefiniowaną przez parametr constantValue), niezależnie od wartości inputValue (odczytu z czujnika). Celem tej metody jest symulowanie sytuacji, w której czujnik zwraca zawsze tą samą wartość, ignorując rzeczywisty odczyt.

public ErrorModel getNewInstance(): Tworzy i zwraca nową instancję modelu błędu ConstantValueError. Jest to metoda, która pozwala na łatwe tworzenie nowych obiektów tego samego typu, co jest przydatne w przypadku dynamicznych symulacji.

**Class DriftError**

**1)Opis classa**

DriftError — jest to klasa implementująca model błędu, który symuluje stopniową dryfowanie odczytów czujnika w czasie. Błąd ten powoduje, że odczyty czujnika będą się zmieniały w sposób stopniowy (np. o określoną wartość na sekundę), aż osiągną maksymalny dozwolony próg. Tego rodzaju model jest przydatny w symulacjach czujników, które z biegiem czasu stają się coraz mniej dokładne, co może odzwierciedlać naturalne zużycie sprzętu lub wpływ warunków zewnętrznych.

**2)Zmienne**

private float driftRate = 0.1f: Zmienna przechowująca tempo, w jakim następuje dryf odczytu czujnika, mierzona w jednostkach na sekundę (np. metry na sekundę). Domyślnie wynosi 0.1f, co oznacza, że odczyt czujnika będzie się zmieniał o 0.1 jednostki na sekundę.

private float maxDrift = 10f: Zmienna określająca maksymalny dozwolony dryf odczytu czujnika. Gdy ta wartość zostanie osiągnięta, dalszy dryf zostanie zatrzymany, a odczyt pozostanie stały. Domyślnie wynosi 10f.

private float accumulatedDrift = 0f: Zmienna przechowująca zgromadzoną wartość dryfu. Jest to suma wszystkich zmiennych dryfów, które narosły w czasie. Ta zmienna jest aktualizowana za każdym razem, gdy metoda getErrorValue() jest wywoływana.

private static final ArrayList<Parameter> parameters: Lista parametrów, które są używane w tym modelu błędu. Zawiera dwa parametry: driftRate (szybkość dryfu) i maxDrift (maksymalny dozwolony dryf). Lista ta jest statyczna i pozwala na modyfikację tych parametrów w czasie działania programu.

**3)Metody klasowe**

public String getDisplayName(): Zwraca nazwę modelu błędu, wyświetlaną w systemie. W tym przypadku jest to "Sensor Drift Error".

public String getDescription(): Zwraca opis modelu błędu. Opis informuje, że model ten symuluje stopniową zmianę odczytu czujnika z upływem czasu, co jest typowe dla efektu dryfu w czujnikach.  
public ArrayList<Parameter> getParameters(): Zwraca listę parametrów, które mogą być przypisane do modelu błędu. W tym przypadku lista zawiera dwa parametry: driftRate (tempo dryfu) i maxDrift (maksymalny dozwolony dryf).

public Map<String, Object> getParameterValues(): Zwraca mapę, która zawiera aktualne wartości parametrów dla tego modelu błędu. Zawiera dwie pary: "driftRate" i "maxDrift", które odpowiadają wartościom zmiennych w tej klasie.

public void setParameterValues(Map<String, Object> parameters): Ustawia wartości parametrów modelu błędu na podstawie mapy. Jeśli mapa zawiera klucze "driftRate" i "maxDrift", przypisuje nowe wartości do zmiennych driftRate i maxDrift. Dzięki temu użytkownik może dynamicznie zmieniać te parametry.

public float getErrorValue(float inputValue): Zwraca wartość błędu (dryfu) na podstawie odczytu z czujnika. Metoda ta zwiększa zgromadzony dryf o wartość driftRate, aż osiągnie wartość maxDrift. Po osiągnięciu tego maksimum, dryf zostaje zablokowany i nie będzie dalej wzrastał.  
public ErrorModel getNewInstance(): Tworzy nową instancję modelu błędu DriftError. Jest to metoda, która pozwala na łatwe tworzenie nowych obiektów tego samego typu w symulacjach.

**Class ErrorModel**

**1)Opis classa**

ErrorModel to klasa abstrakcyjna, która definiuje ogólną strukturę dla modeli błędów, które mogą być stosowane w różnych symulacjach odczytów czujników. Jest to klasa bazowa, z której dziedziczą inne klasy reprezentujące konkretne typy błędów (takie jak dryf, offset, hałas, etc.). Każdy model błędu implementuje metody umożliwiające konfigurację parametrów błędu, obliczanie wartości błędu na podstawie danych wejściowych oraz tworzenie nowych instancji tego modelu.

**2)Metody klasowe**

public abstract String getDisplayName(): Jest to abstrakcyjna metoda, którą muszą zaimplementować klasy dziedziczące. Powinna zwracać nazwę wyświetlaną modelu błędu (np. "Bias Error", "Drift Error", itp.).

public abstract String getDescription(): Abstrakcyjna metoda, która musi być zaimplementowana przez klasy dziedziczące. Powinna zwracać opis modelu błędu, który wyjaśnia, czym ten błąd jest i jak działa.

public abstract ArrayList<Parameter> getParameters(): Abstrakcyjna metoda, która zwraca listę parametrów, które mogą być ustawione w danym modelu błędu. Jest to lista obiektów klasy Parameter, która przechowuje informacje o typie parametru, jego nazwie i opisie.

public abstract Map<String, Object> getParameterValues(): Abstrakcyjna metoda, która zwraca mapę z aktualnymi wartościami parametrów przypisanych do modelu błędu. Klasa dziedzicząca musi zdefiniować, jakie dokładnie parametry są używane i jak je zwrócić.

public abstract void setParameterValues(Map<String, Object> parameters): Abstrakcyjna metoda, która pozwala ustawić wartości parametrów dla modelu błędu. Parametry są przekazywane w formie mapy, gdzie klucze to identyfikatory parametrów, a wartości to nowe wartości, które mają zostać przypisane.

public void setParameterValue(String parameterID, Object value): Jest to metoda pomocnicza, która pozwala na ustawienie wartości pojedynczego parametru w modelu błędu. Jest to wygodne, gdy chcemy zmienić tylko jeden parametr, zamiast przekazywać całą mapę.

public abstract float getErrorValue(float inputValue): Abstrakcyjna metoda, która oblicza wartość błędu na podstawie wartości wejściowej (np. odczytu z czujnika). Każdy model błędu musi określić, jak przekształcić dane wejściowe na wartość z błędem.

public abstract ErrorModel getNewInstance(): Abstrakcyjna metoda, która tworzy nową instancję tego samego modelu błędu. Jest to użyteczne, gdy chcemy stworzyć kopię obiektu modelu błędu lub utworzyć nowy obiekt do symulacji.

public String toString(): Metoda, która zwraca nazwę modelu błędu jako łańcuch znaków. Zwykle wykorzystywana do reprezentacji obiektu w formie tekstowej.

**Class IntermittentDropoutError**

**1)Opis classa**

IntermittentDropoutError to klasa, która dziedziczy po klasie ErrorModel i symuluje okresowe zaniki odczytów czujnika. W tym modelu czujnik czasami "gubi" dane, a jego odczyt jest równy zeru. Zaniki występują losowo, z określonym prawdopodobieństwem, które można ustawić jako parametr. Model ten jest użyteczny do symulacji błędów związanych z niestabilnością lub utratą sygnału w czujnikach.

**2)Zmienne**

dropoutProbability: Określa prawdopodobieństwo wystąpienia zaniku sygnału. Jest to wartość w zakresie [0.0, 1.0], gdzie 0 oznacza brak zaniku, a 1 oznacza, że zawsze będzie występować. Domyślnie ustawiona na 0.1 (10% szans na zanik).

Parameters: ista parametrów przypisanych do modelu błędu. Zawiera jeden parametr: dropoutProbability. Jest to lista obiektów klasy Parameter, która zawiera informacje o nazwie parametru, jego typie oraz opisie.

random: Obiekt klasy Random, który jest używany do generowania losowych liczb, aby symulować okresowe zaniki danych. Służy do generowania wartości w zakresie [0.0, 1.0], które porównuje się z wartością dropoutProbability.

**3)Metody klasowe**

public String getDisplayName(): Zwraca nazwę modelu błędu, czyli "Intermittent Dropout Error". Jest to metoda implementowana z klasy bazowej ErrorModel.

public String getDescription(): Zwraca opis modelu błędu, czyli "Simulates intermittent loss of sensor readings, outputting zero occasionally". Opisuje, że błąd ten symuluje okresowe zaniki odczytów czujnika, zwracając zero w przypadku zaniku.

public ArrayList<Parameter> getParameters(): Zwraca listę parametrów przypisanych do tego modelu błędu. W tym przypadku jest to lista zawierająca parametr dropoutProbability, który definiuje prawdopodobieństwo wystąpienia zaniku odczytu.

public Map<String, Object> getParameterValues(): Zwraca mapę zawierającą aktualne wartości parametrów tego modelu błędu. Zawiera jedną parę klucz-wartość: "dropoutProbability" oraz jego aktualną wartość.

public void setParameterValues(Map<String, Object> parameters): Ustawia wartości parametrów na podstawie przekazanej mapy. Jeżeli mapa zawiera klucz "dropoutProbability", wartość tego parametru jest ustawiana na podstawie przekazanej wartości.

public float getErrorValue(float inputValue): Oblicza wartość błędu na podstawie losowego zaniku. Jeśli losowo wygenerowana liczba jest mniejsza niż wartość dropoutProbability, to zwraca 0.0f (zanik sygnału), w przeciwnym razie zwraca oryginalną wartość wejściową (brak błędu).

public ErrorModel getNewInstance(): Tworzy i zwraca nową instancję tego samego modelu błędu (w tym przypadku IntermittentDropoutError). Jest to metoda pomocnicza do tworzenia nowych obiektów tego samego typu.

**Class OscillatingError**

**1)Opis classa**

OscillatingError to klasa, która dziedziczy po klasie ErrorModel i symuluje okresowe fluktuacje w odczytach czujnika. Błąd w tym modelu ma postać oscylacji o określonej amplitudzie i częstotliwości. Wartość błędu zmienia się cyklicznie w czasie, tworząc sinusoidalne wahania w odczytach czujnika. Tego typu błąd może być użyteczny w symulacjach czujników, które mogą doświadczać regularnych, okresowych zakłóceń, takich jak te spowodowane drganiami lub zakłóceniami w systemie.

**2)Zmienne**

amplitude: Amplituda oscylacji (w metrach), która określa zakres wahań odczytu czujnika. Im wyższa wartość, tym większe wahania. Domyślnie ustawiona na 5f.

frequency: Częstotliwość oscylacji (w cyklach na sekundę), która określa, jak często odczyt czujnika zmienia się w cyklu. Im wyższa częstotliwość, tym szybciej następują zmiany wartości odczytu. Domyślnie ustawiona na 1f.

parameters: Lista parametrów przypisanych do modelu błędu. Zawiera dwa parametry: amplitude i frequency, które kontrolują amplitudę i częstotliwość oscylacji.

**3)Metody klasowe**

public String getDisplayName(): Zwraca nazwę modelu błędu, czyli "Oscillating Error". Jest to metoda implementowana z klasy bazowej ErrorModel.

public String getDescription(): Zwraca opis modelu błędu, czyli "Simulates periodic fluctuations in the sensor reading". Opisuje, że błąd ten symuluje okresowe fluktuacje w odczytach czujnika.

public ArrayList<Parameter> getParameters(): Zwraca listę parametrów przypisanych do tego modelu błędu. W tym przypadku jest to lista zawierająca dwa parametry: amplitude i frequency, które kontrolują oscylacje.

public Map<String, Object> getParameterValues(): Zwraca mapę zawierającą aktualne wartości parametrów tego modelu błędu. Zawiera pary klucz-wartość dla parametrów amplitude i frequency.

public void setParameterValues(Map<String, Object> parameters): Ustawia wartości parametrów na podstawie przekazanej mapy. Jeżeli mapa zawiera klucze "amplitude" lub "frequency", odpowiednie wartości tych parametrów są ustawiane na podstawie przekazanych wartości.

public float getErrorValue(float inputValue): Oblicza wartość błędu na podstawie oscylacji. Z wykorzystaniem funkcji sinusoidalnej (sinus) oblicza oscylację na podstawie czasu, a następnie dodaje ją do oryginalnej wartości wejściowej. Czas jest pobierany z kontrolera symulacji. Im wyższa częstotliwość, tym szybciej zachodzą zmiany. Wartość oscylacji jest ograniczona przez amplitudę.

public ErrorModel getNewInstance(): Tworzy i zwraca nową instancję tego samego modelu błędu (w tym przypadku OscillatingError). Jest to metoda pomocnicza do tworzenia nowych obiektów tego samego typu.

**Class RandomNoiseError**

**1)Opis classa**

RandomNoiseError — jest to klasa, która dziedziczy po klasie ErrorModel i symuluje dodanie losowego szumu do odczytu czujnika. Szum jest generowany w losowy sposób w zakresie określonym przez parametr noiseRange. Tego typu błąd jest przydatny w symulacjach czujników, które mogą doświadczać zakłóceń losowych, jak np. w wyniku interferencji elektromagnetycznej, zakłóceń sieciowych lub innych czynników zewnętrznych.

**2)Zmienne**

noiseRange: Maksymalna amplituda losowego szumu. Określa zakres, w którym generowane będą wartości szumu, tj. od -noiseRange do +noiseRange. Domyślnie ustawiona na 5f.

parameters: Lista parametrów przypisanych do tego modelu błędu. Zawiera jeden parametr: noiseRange, który kontroluje zakres szumu losowego.

random: Obiekt klasy Random, używany do generowania losowych wartości w celu symulacji szumu. Służy do generowania liczb losowych w zadanym zakresie.

**3)Metody klasowe**

public String getDisplayName(): Zwraca nazwę modelu błędu, czyli "Random Noise Error". Jest to metoda implementowana z klasy bazowej ErrorModel.

public String getDescription(): Zwraca opis modelu błędu, czyli "Adds random noise to the sensor reading based on 'Noise range' parameter". Opisuje, że błąd ten dodaje losowy szum do odczytu czujnika w zależności od wartości parametru noiseRange.

public ArrayList<Parameter> getParameters(): Zwraca listę parametrów przypisanych do tego modelu błędu. W tym przypadku jest to lista zawierająca jeden parametr: noiseRange, który kontroluje zakres losowego szumu.

public Map<String, Object> getParameterValues(): Zwraca mapę zawierającą aktualną wartość parametru noiseRange tego modelu błędu.

public void setParameterValues(Map<String, Object> parameters): Ustawia wartości parametrów na podstawie przekazanej mapy. Jeżeli mapa zawiera klucz "noiseRange", wartość tego parametru jest ustawiana na podstawie przekazanej wartości.

public float getErrorValue(float inputValue): Oblicza wartość błędu, dodając losowy szum do wartości wejściowej. Szum jest generowany w zakresie od -noiseRange do +noiseRange i dodawany do wartości inputValue. Wartość szumu jest obliczana przy użyciu obiektu random.

public ErrorModel getNewInstance(): Tworzy i zwraca nową instancję tego samego modelu błędu (w tym przypadku RandomNoiseError). Jest to metoda pomocnicza do tworzenia nowych obiektów tego samego typu.