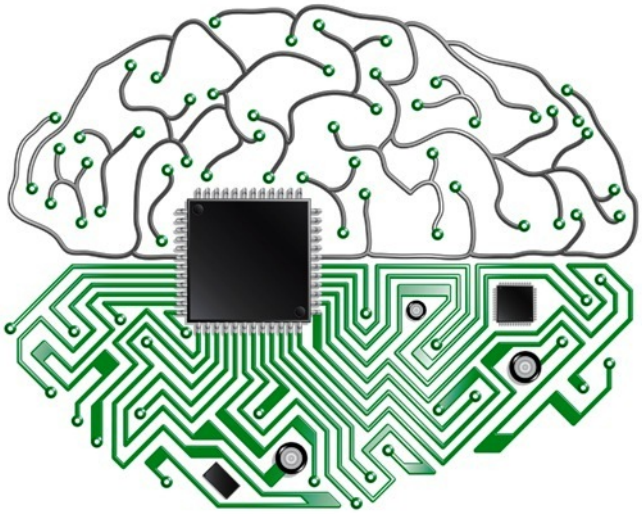
Specyfikacja implementacyjna programu "Noobot".

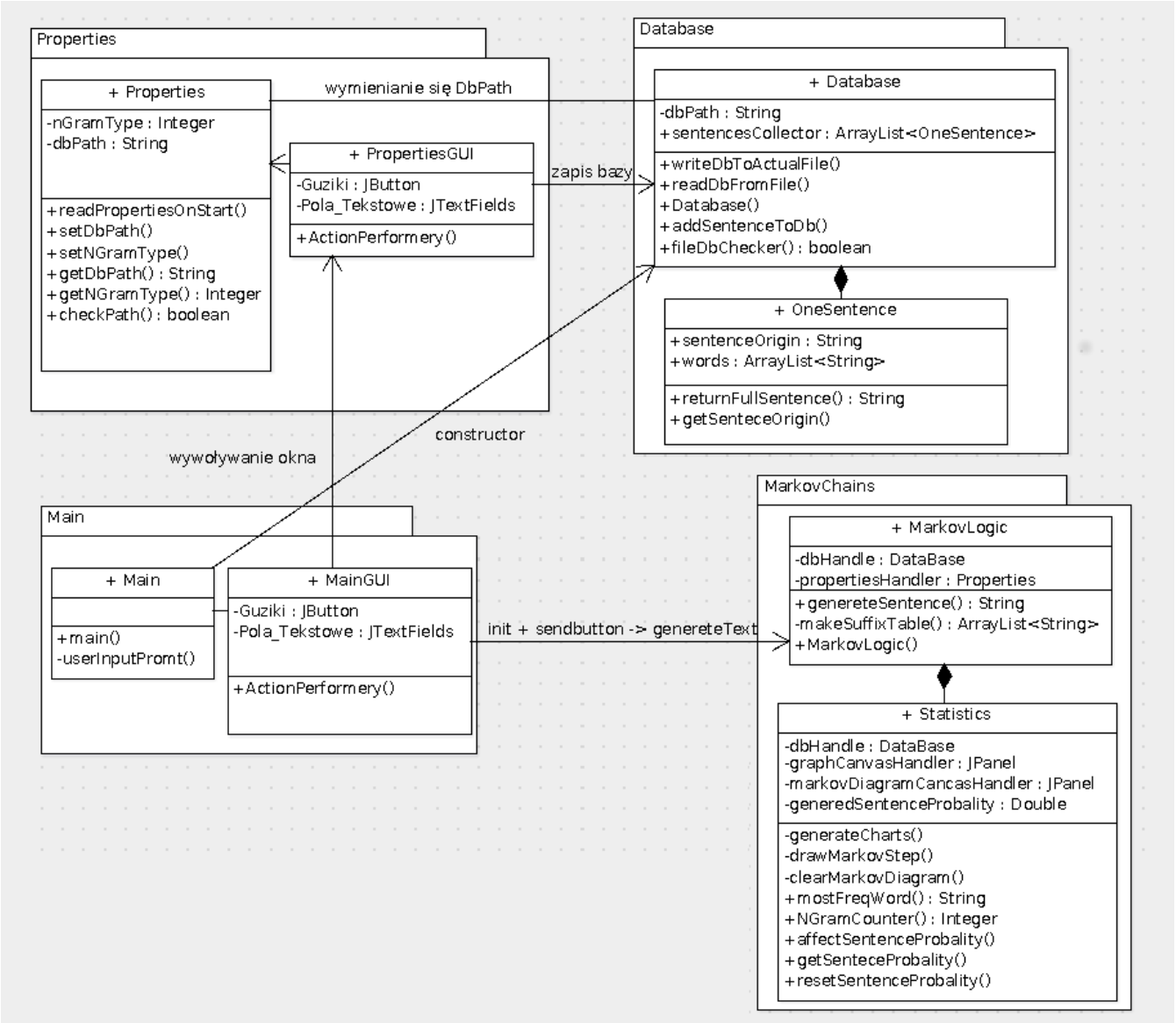


Projekt na zajęcia: *"Języki i Metodyka Programowania"*
wydz. Elektryczny Politechniki Warszawskiej, rok akademicki 2014/2015
wykonał: Paweł Drapiewski
pod kierownictwem: mgr inż. Zuzanny Krawczyk

1. Ogólny opis programu

Jest to program do czatu ze sztuczną inteligencją, jego ogólny opis został zamieszczony w poprzednim dokumencie nazwanym "specyfikacja_funkcjonalna_JIMP_proj_JAVA_ver1.pdf". Z technicznych ogólny aspektów należy zaznaczyć iż będzie on korzystał ze standardowej biblioteki graficznej javy - swing w wersji JSE 7.

2. Poglądowy diagram klas



3. Szczegółowy elementów modelu

3.1 pakiet Main

Pakiet ten jest odpowiedzialny przede wszystkim za rozruch programu, a także za interfejs do komunikacji z użytkownikiem wraz z elementami ułatwiającymi korzystanie, jak np. podpowiadanie zdań podczas wpisywania tekstu przez użytkownika.

3.1.1 klasa Main

Zarządza logiczną stroną głównego okna programu

Metody tej klasy:

- **public static void main()** - główna metoda programu, to za jej pomocą inicjalizowane są wszystkie klasy z pakietów Database i MarkovChains.
- **private void userInputPrompt()** - analizuje wprowadzane znaki do okna czatu, i na ich podstawie wyświetla proponowane słowo, kliknięcie TAB będzie akceptowało to słowo.

3.1.2 klasa MainGUI

Odpowiada za layout głównego okna, oraz za wywoływanie akcji np. "Ustawienia" spowoduje otworzenie okna z ustawieniami.

Metody tej klasy:

- **różnego rodzaju ActionListenery**

3.2 pakiet Properties

Zawiaduje on ustawieniami programu czyli m.in. wczytywaniem ustawień domyślnych podczas uruchamiania programu.

3.2.1 klasa Properties

Dostarcza szereg funkcjonalności służących do obsługi zmiany ustawień.

Metody tej klasy:

- **public void readPropertiesOnStart()** - czyta ona plik konfiguracyjny zapisany w notacji XML i na jej podstawie ustawia swoje pola. Istnieje możliwość zastąpienia nazwy tej poprzez nazwę będącą konstruktorem tej klasy.
- **public boolean setDbPath(String dbPath)** - jeżeli pod daną ścieżką istnieje plik będący odpowiednią bazą danych to ustawi on bazę danych na ten plik.
- **public boolean setNGramType(Integer ngram)** - sprawdza czy jest to liczba z przedziału [1, 50] i ustawia pole nGramType na ten typ.
- **public Integer get...()** - metody służące do pobrania wartości pól tej klasy, gdzie zamiast "..." należy wstawić nazwę pola.

3.2.2 klasa PropertiesGUI

Layout okna ustawienia, oraz zarządzanie zdarzeniami na kliknięcia.

Metody tej klasy:

- **różnego rodzaju ActionListenery**

3.3 pakiet Database

Służy do tworzenia bazy danych i jej obsługi wliczając w to zapis i odczyt z pliku. Sama baza danych to kolekcja zbiorów słów, gdzie pojedynczy zbiór oznacza logiczny ciąg słów, które razem tworzą logiczne zdania.

3.3.1 klasa Database

Główna klasa pakietu Database. Dostarcza przede wszystkim interfejs Database.

Metody tej klasy:

- **public void writeDbToActualFile()** - zapisuje aktualną bazę danych do pliku znajdującego się pod scieżką określoną polem dbPath.
- **public void readFromFile()** - sprawdza plik metodą fileDbChecker(dbPath), a następnie wczytuje plik. Scieżka pliku jest zdefiniowana przez pole dbPath.
- **Database()** - konstruktor, który musi zainicjować swoją bazę danych, czyli ArrayList klass OneSentence.
- **public void addSentenceToDb(String sentence, String sentenceOrigin)** - rozdziela przekazane zdanie na słowa i chowa je do swojej bazy danych, SenteceOrigin oznacza nazwę źródła z którego pochodzi np. user_input_14.03.14:22:23:02
- **public boolean fileDbChecker(String filePath)** - sprawdza czy plik pod daną ścieżką istnieje, a kolejno czy jest on zapisany w notacji zgodnej z tą programową.

3.3.2 klasa OneSentence

Klasa pomocnicza, jest ona ArrayList'em słów, które pochodzą z pojedynczego źródła, czyli powinny tworzyć logiczny tekst.

Metody tej klasy:

- **public String returnFullSentence()** - zwraca pełne zdanie, czyli to co kiedyś dostał na wejściu.
- **public String getSenteceOrigin()** - zwraca nazwę źródła z którego ten ciąg słów pochodzi.

3.4 pakiet MarkovChains

Pakiet dostarczający główną funkcjonalność programu czyli generacje tekstu, a także pobocznie, szereg statystyk dotyczących generowanego tekstu jak np. diagram przejść.

3.4.1 klasa MarkovLogic

Zadaniem tej klasy jest generacja tekstu.

Metody tej klasy:

- **MarkovLogic(Database dbHandl, Properties propHandl)** - konstruktor, który na wejściu wymaga referencje do bazy danych na której będzie operować, a także referencji klasy Properties, czyli do ustawień na których będzie polegać.

- **public String generateSentece(Integer userNmbOfWords, ArrayList<String>startNGram)** - generuje tekst, gdzie wylosuje ilość słów do generowania (gdy większe od ngramType + 1) na podstawie userNmbOfWords (czyli ilość słów wpisana przez użytkownika), i liczba wygenerowana będzie z zakresu ±20% od userNmbOfWords. A gdy userNmbOfWords =<ngramType + 1 to userNmbOfWords + 1. A argument startNGram to lista wejściowego ngramu, czyli tego od którego rozpocznie generacje. I jest to ngram wylosowany spośród tekstu wejściowego użytkownika. Sama genreacja odbywać się będzie na podstawie łańcuchów Markova, czyli do aktualnego ngramu będzie tworzona tablica sufixów, a następnie losowany bedzie z niej jeden sufix, po czym będzie on dołączany do koniec ngramu (gdzie początek będzie ucięty by zachować rodzaj ngramu) i wypisany. I taka operacja będzie powtórzona póki nie osiągnie się zadanej długości bądź nie zabraknie możliwości przejścia (przypadek gdy tablica sufixów będzie pusta).

- **private ArrayList<String>makeSuffixTAble(ArrayList<Stirng>NGram)** - tworzy tablicę sufixów opisaną w metodzie wyżej.

3.4.2 klasa Statistics

Obejmuje szereg metod obsługujących statystyki, zarówno tych które zwyczajnie zwracają dane np. jako liczbę, jak i tych które obsługują rysowanie wykresów itp.

Metody tej klasy:

- **Statistics(JPanel chartsHndl, JPanel diagramHndl)** - konstruktor pobierający referencje do JPnel'ów w których będzie rysował swoje dane statystyczne. I tak chartHndl to JPanel od rysowanie słupków z liczebnością słów w całej bazie danych dla kolejnych słów wygenerowanych. A digaramHndl to JPanel od rysowania diagramów przejść algorytmu Markova.

- **private void generateChart()** - Na podstawie wygenerowanego zdania tworzy wykres słupkowy liczebności w bazie danych dla każdego z wyrazów z wygenerowanego zdania.

- **private void drawMarkovStep(ArrayList<String>suffix, String choosen)** - Rysuje diagram przejść algorytmu, metoda ta jest wywoływana przy generacji każdego pojedynczego ngramu. Polega on na wypisywaniu w kolumnie wszystkich słów, a te wylosowane zaznaczane jest kolorem czerwonym. Każda generacja ngramu nie nadpisuje poprzedniego wykresu tylko wypisuje się obok, oraz tworzone są linie pomiędzy słowami wylosowanymi.

- **private void clearMarkovDiagram()** - czyści JPanel w którym jest rysowany diagram przejść. Metoda ta jest wywoływana na początku generacji tekstu.

- **public String mostFreqWord()** - najczęściej występujące słowo w całej bazie danych

- **public Integer nGramCounter(ArrayList<String>ngram)** - liczy ile dany ngram wystąpił w bazie danych.

- **public void affectSentenceProbality(ArrayList<String>ngram)** - liczy prawdpodobieństwo wystąpienia danego ngramu, a otrzymaną liczbę zapisuje w polu generedSentenceProbality jako iloczyn tej liczby i generedSentenceProbality.

- **public void getSentenceProbality()** - pobiera prawdopodbieństwo zapisane pod polem genredSenteceProbality.

- **public void resetSentenceProbality()** - ustawia wartość pola generedSentenceProbality na 1.

4. Tstowanie

4.1 Ogólna procedura testowania

Testowanie będzie odbywać się przede wszystkim metodą "code, run and check" czyli uruchamianiem i sprawdzaniem jak to działa, wliczając to wspomaganie się debuggerem dostarczonym do IDE NetBeans 8.0.2. A dodatkowo klasy Database, MarkovLogic, Statistics zostaną przetestowane za pomocą metodyki testowania zwanej "testy jednostkowe" dostarczanej przez bibliotekę JUnit.

4.2 Test jednostkowy

Poniżej zamieszczony jest kod źródłowy testu jednostkowego klasy Database.

```
public class DatabaseTest
@Test
public void TestDatabse() {
    String tekst1 = "Tekst przykladowy numer 1";
        String tekst2 = "Brzeczyszyczkiewicz zawsze do uslug!";
        String tekst3 = "Trololololo, lalallala";
        String tekst4 = "Rome ohhh, Romeo";

        Database tstDb = new Database();

        tstDb.addSentenceToDb(tekst1);
        tstDb.addSentenceToDb(tekst2);
        tstDb.addSentenceToDb(tekst3);
        tstDb.addSentenceToDb(tekst4);

        assertEquals(tstDb.sentencesCollector.get(0).returnFullSentence(),
        "Tekst przykladowy numer 1");
        assertEquals(tstDb.sentencesCollector.get(1).returnFullSentence(),
        "Brzeczyszyczkiewicz zawsze do uslug!");
        assertEquals(tstDb.sentencesCollector.get(2).returnFullSentence(),
        "Trololololo, lalallala");
        assertEquals(tstDb.sentencesCollector.get(3).returnFullSentence(),
        "Rome ohhh, Romeo");
}
```