Operacje wejścia / wyjścia

Ćwiczenie 1

Napisz klasę `Sentence`, która będzie zawierała metodę:

```
+readSentence(filename:String):String
```

odczytującą kolejne linie z pliku tekstowego i zwracającą je jako zdanie (pierwsza litera duża, na końcu kropka). Wykorzystaj plik `test.txt` z pakietu `zad1`. Obsłuż wyjątek `FileNotFoundException` w metodzie `readSentence()`

Ćwiczenie 2

Napisz klasę `Sentencer` będącą rozszerzeniem klasy `Sentence` i dostarczającą metodę:

```
+writeSentence(filename:String, sentence:String):String
```

która zapisze do pliku tekstowego przekazany ciąg znaków. Plik zapisz w pakiecie `zad2` pod nazwą `mySentence.txt`

Ćwiczenie 3

Napis klasę `MoneyConverter`, która dostarczy metody:

```
-readCourse(currency:String):double
+convert(money:double, to:String):double
+convert(money:double, to:String, from:String):double
```

Prywatna metoda `readCourse()` powinna zwracać kurs dla przekazanej waluty jako wartość `double`.

Jeżeli zostanie wywołana metoda `convert()` bez trzeciego parametru przyjmujemy, że przeliczanie kosztów nastąpi z waluty PLN. Wykorzystaj plik currency.txt do dokonania testowych obliczeń.

Ćwiczenie 4

Napisz klasę `TempConverter`, która będzie dostarczała metody:

```
+toKelvin(temp:double):double
+toFahrenheit(temp:double):temp
+readTemp(filename:String):double[]
+writeTemp(temp:double[]):void
```

W pliku tekstowym `tempC.txt` w pakiecie `zad4` znajdują się poszczególne temperatury dla danych miesięcy zapisane jako temperatura w Celciusach. Do plików `tempK.txt` oraz `tempF.txt` zapisz (w pakiecie `zad3`) odpowiednio wyniki przekonwertowanej temperatury w Kelvinach i Fahrenheitach.

Ćwiczenie 5

Napisz klasę `LengthChecker`, która dostarczy metody:

```
-isProperLength(String arg, int len):boolean
-readFile(String filename):String[]
-writeFile(String[] fileContent):void
+make(String fileInput, int len):void
```

Działanie klasy powinno wyglądać następująco; w metodzie `make()` wywołujemy pozostałe prywatne metody w taki sposób, aby w pliku wynikowym znalazły się słowa dłuższe niż przekazany drugi argument. Do odczytu słów użyj pliku `words.txt`, wynik pracy zapisz `words_X.txt`, gdzie X to właśnie drugi przekazany parametr metody `make()`.

Ćwiczenie 6

Napisz klasę `Student`, która będzie dostarczała pola:

```
-index:int
-name:String
-secondName:String
-course:String
-averageMark:double
```

oraz `konstruktory` (bez i sparametryzowany), `gettery` i `settery`.

Następnie utwórz klasę `University`, która będzie zawierała następujące metody:

```
-isStudentExists(index:int):boolean
+getStudent(index:int):Student
+putStudent(student:Student):void
```

Klasa `Univeristy` powinna działać na pliku `students.txt` znajdującym się w pakiecie zad6. Format pliku powinien być następujący (bez używania serializacji obiektu):

```
123   Paweł Testowy    Algorytmy  4.12
```

Wartości powinny być rozdzielone znakiem tabulacji `"\t"`. Metoda `getStudent()` powinna zwracać nową instancję klasy `Student` z uzupełnionymi danymi studenta według indeksu, a metoda `putStudent()` dodawać studenta na końcu listy, w przypadku, gdy taki student nie istnieje. Do sprawdzenia występowania studenta o zadanym indeksie wykorzystaj prywatną metodę `isStudentExists()`.

Ćwiczenie 7

Utwórz klasę `AvgChecker`, która dostarczy pola i metody:

```
-filename:String

+AvgChecker(String filename)
+process():void
```

Konstruktor powinien ustawiać pole prywatne na wartość przekazaną w parametrze. Metoda `process()` powinna odczytać wszystkie wartości z pliku przekazanego jako parametr konstruktora, a następnie nadpisać plik w taki sposób, aby znalazły się w nim wartości powyżej średniej. Średnią należy liczyć poziomo. według kolumn, które są rozdzielone znakiem tabulacji. Przykład:

```
Paweł Testowy   5     6     2
```

Daje średnią `4,33(3)`. Jeżeli średnia wszystkich średnich z pliku będzie mniejsza niż `4,33(3)` taki wpis powinien pozostać w pliku.

Ćwiczenie 8

Napisz klasę `Columner` zawierającą metody i pola:

```
-filename:String
-currentColumn:double[]

+Columner
+Columner(filename:String)
+sumColumn(column:int):double
+avgColumn(column:int):double
+countColumn(column:int):int
+maxColumn(column:int):double
+minColumn(column:int):double
-readFile():void
+writeColumn(filename:String):void
+readColumn(column:int)
```

Metoda `readColumn()` powinna odczytywać wartości aktualnej kolumny do pola `currentColumn`. Metody `*Column` powinny zwracać sumę, średnią elementów oraz ich ilość, maksymalny i minimalny element.

Metoda `writeColumn()` powinna zapisywać wartości wybranej kolumny do pliku. W przypadku braku danej kolumny (ilość kolumn w pliku mniejsza niż przekazany argument) wyrzuć wyjątek `IllegalArgumentException`.

Ćwiczenie 9

Napisz klasę `PresentChecker`, która dostarczy metody i pola:

```
-filename:String

+PresentChecker
+PresentChecker(filename:String)
-checkIfExists(sentence:String):boolean
+readWords():void
```

Metoda `readWords()` powinna wczytywać od użytkownika słowa, sprawdzać czy jest ono zapisane w pliku i zwracać informację do konsoli czy podane słowo występuje w słowniku czy nie. Słowa zapisane są w pliku words.txt w pakiecie `zad9`.

Ćwiczenie 10

Napisz klasę `Word` przetrzymującą słowo:

```
-polishWord:String
-englishWord:String
```

oraz `konstruktory`, `gettery` i `settery`. Następnie utwórz klasę `Dict`, która będzie zawierała:

```
-dict:Word[]

+readDict(filename:String):void
+addWord(polishWord:String, englishWord:String):void
+removeWord(polishWord:String):void
+translateToEn(polishContent:String):String
+translateToPl(englishContent:String):String
```

Klasa `Dict` powinna realizować tłumaczenie tekstu na podstawie słownika. Ponadto metoda `readDict()` powinna być wywoływana w konstruktorze.