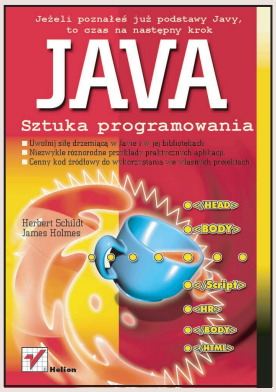
Java. Sztuka programowania



Autorzy: Herbert Schildt, James Holmes
Tłumaczenie: Rafał Jońca
ISBN: 83-7361-422-2
Tytuł oryginału: The Art of Java
Format: B5, stron: 324

Przejdź na wyższy poziom programowania dzięki dwóm „guru" Javy: Herbowi Schildtowi i Jamesowi Holmesowi. Połączenie ich wiedzy pozwoli Ci poznać wiele sekretów i sztuczek wykorzystywanych przez profesjonalistów. W książce autorzy przedstawiają przykłady użytecznych aplikacji oraz towarzyszące im opisy użytych technologii. Prezentowane przykłady możesz wykorzystać w swojej codziennej pracy. Począwszy od interpreterów języka, agentów internetowych i podsystemów e-mail, skończywszy na analizatorach wyrażeń, narzędziach statystycznych i apletach finansowych – wszystkie aplikacje są gotowe do użycia. Można je też dowolnie modyfikować i rozszerzać.

W książce znajdziesz:

- Omówienie zalet języka Java,
- Tworzenie analizatora wyrażeń numerycznych,
- Tworzenie agenta przeszukującego internet,
- Projektowanie i implementacja interpretera języka programowania,
- Wykonanie funkcjonalnego systemu pocztowego,
- Konstruowanie programu do pobierania danych z internetu z możliwością pobierania fragmentów stron,
- Wykonanie narzędzi statystycznych obliczających średnia, medianę, modalną, odchylenie standardowe, itp.
- Wykonanie apletów i serwletów finansowych obliczających równe raty pożyczki, przyszłą wartość inwestycji, kwotę emerytury, itp.
- Prześledzenie różnych technik wyszukiwania bazujących na sztucznej inteligencji,
- Zapoznanie się z możliwością przeglądania stron HTML-a w Javie.

O autorach:

Herb Schildt jest jednym z najpopularniejszych autorów książek o programowaniu. Jest autorytetem w sprawach języków C, C++, Java i C#, a także doskonałym programistą systemu Windows. Jego książki sprzedały się na całym świecie w ponad 3 milionach egzemplarzy i zostały przetłumaczone na większość języków.

James Holmes jest konsultantem do spraw tworzenia programowania oraz aplikacji serwerowych w środowiskach biznesowych. Zdobył wiele nagród, między innymi Summer Olympic Games oraz nagrodę Java Developer w roku 2002, przyznawaną przez Oracle Magazine.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 8.
Statystyka i wykresy

Autor: Herb Schildt

Javy używa się przede wszystkim do tworzenia małych programów, na przykład apletów i serwletów, służących do przetwarzania i wyświetlania danych. Dane często są liczbami, na przykład reprezentują ceny akcji, temperatury dzienne, ruch klientów itp. Bardzo często trzeba przetworzyć te dane lub też narysować wykres na ich podstawie. Na przykład aplet może wyświetlać średnią cenę akcji w przeciągu ostatnich kilku miesięcy i rysować wykres prezentujący zmiany ceny. Statystyka i wykresy bardzo często pojawiają się w trakcie pisania programów w Javie, zatem zajmiemy się nimi w tym rozdziale.

W tym rozdziale zostanie pokazane wykonanie metody służącej do obliczania następujących statystyk:

- średniej;
- mediany;
- dominanty (mody, wartości modalnej);
- odchylenia standardowego;
- równania regresji (linia najlepszego dopasowania);
- współczynnika korelacji.

W tym rozdziale pokażemy także sposób rysowania wykresów. Przykłady tutaj przedstawione każdy może dostosować do własnych potrzeb.

W niniejszym rozdziale szczególny nacisk zostanie położony na dwa aspekty: obliczenia matematyczne i wyświetlanie uzyskanych wyników w sposób graficzny. Java nie jest zoptymalizowana pod kątem obliczeń matematycznych ale wspiera różne metody obliczeń. Choć obliczenia pokazane w tym rozdziale nie wymagają dużej mocy procesora ani nie są wysoce złożone, stanowią dobrą ilustrację możliwych sposobów radzenia sobie z danymi.

Java od samego początku była zaprojektowana jako język zapewniający interfejs graficzny. Z tego powodu istnieje wiele klas zajmujących się interfejsami graficznymi. Java posiada aktualnie dwa systemy graficznych interfejsów użytkownika: AWT i Swing. System Swing był dosyć intensywnie opisywany w poprzednich rozdziałach, zatem teraz warto zapoznać się z systemem AWT. Czytelnik dowie się, w jaki sposób można tworzyć okna AWT, jak zapewnia się obsługę zmiany ich rozmiaru, ponowne rysowanie okna i inne elementy. Java posiada unikalne cechy, takie jak klasy wewnętrzne i adaptory, zatem kod interfejsu graficznego w tym języku jest bardziej elegancki i krótszy od tego samego kodu uzyskiwanego w innych językach.

Próbki, zbiory, rozkład i zmienne

Przed rozpoczęciem omawiania właściwego programu należy zdefiniować kilka podstawowych terminów i koncepcji związanych ze statystyką. Ogólnie informacje statystyczne otrzymuje się w postaci *próbek*. Następnie dokonuje się uogólnienia wyników. Każda z próbek pochodzi z określonego zbioru wartości przewidzianych dla danej sytuacji. Stosuje tu się po prostu nazwę *zbiór*. Na przykład można szacować wielkość produkcji fabryki zapałek w przeciągu całego roku, uogólniając dane zebrane tylko w jednym dniu. W ten sposób można dokonać ekstrapolacji dotyczącej całego roku na podstawie znacznie skromniejszej informacji.

Jeżeli próbka jest wyczerpująca, jest równa całemu zbiorowi. W przypadku omawianej, przykładowej fabryki, jeśli próbka zawiera wielkość produkcji fabryki dla całego roku, wtedy jest ona równa całemu zbiorowi. Jeśli próbka jest mniejsza od całego zbioru, jest możliwe wystąpienie błędu. Wtedy należy znać współczynnik określający wielkość błędu. Na potrzeby niniejszego rozdziału zakłada się, iż próbka jest równa zbiorowi, więc nie występują błędy statystyczne.

Informacje statystyczne zależą od rozkładu zmiennych losowych w zbiorze. Możliwych jest kilka różnych rodzajów rozkładów prawdopodobieństwa przyjmowania wartości przez zmienne losowe. Najbardziej znanym jest *rozkład normalny*, czyli *rozkład Gaussa*, którego reprezentacją jest tzw. krzywa dzwonowa. Rozkład analizowanych wartości przyjmuje kształt symetrycznego dzwonu. Oznacza to, że zmienne losowe o wartości zbliżonej do przeciętnej występują najczęściej.

W trakcie badania statystycznego określa się *zmienne zależne* (badane) i *zmienne niezależne* (ich wartość pozwala wyliczyć zmienną zależną). W tym rozdziale zmienną niezależną jest czas. Zmienna ta jest zwiększana w każdym roku o wartość jednostkową. Zastosowanie czasu jako zmiennej niezależnej jest powszechnym zjawiskiem. Na przykład przy badaniu kursów stosuje się zmienną niezależną z odstępem jeden dzień.

Podstawy statystyki

U podstawy większości analiz statystycznych występują trzy parametry badanych prób: *średnia*, *mediana* i *moda* (*dominanta*). Każdy z wymienionych współczynników jest użyteczny ale dopiero ich połączenie pozwala na dokładne przeanalizowanie charakterystyki próbki.

Metody statystyczne opisywane w niniejszym rozdziale opierają się na założeniu, iż zmienne losowe, składające się na próbkę znajdują się w tablicy zmiennych typu `double`. Wszystkie metody statystyczne są metodami typu `static` przechowywanymi w klasie `Stats`. Klasę tą w całości przedstawimy w dalszej części rozdziału. Prezentowane metody są typu `static`, zatem można je wywołać bez potrzeby tworzenia obiektu `Stats`.

Średnia

Średnia jest najczęściej wykorzystywaną funkcją w statystyce. Zapewnia obliczanie wartości średniej arytmetycznej podanego zbioru zmiennych losowych. W ten sposób znajduje się niejako „środek ciężkości" zestawu danych. Aby obliczyć średnią arytmetyczną danego zbioru liczb, należy zsumować wszystkie jego elementy i otrzymaną sumę podzielić przez liczbę elementów. Na przykład sumą dla wartości

1 2 3 4 5 6 7 8 9 10

jest 55. Podzielenie tej wartości przez liczbę elementów w próbce (10), prowadzi do uzyskania wyniku 5.5. Wzór na średnią arytmetyczną jest następujący.

$$M = \frac{1}{N} \sum_{i=1}^{N} D_i$$

W tym wzorze D_i reprezentuje element danych, natomiast N jest liczbą elementów (zmiennych losowych) w próbce.

Poniższa metoda o nazwie `mean()` oblicza średnią dla wartości przekazanych w tablicy jako parametr. Metoda zwraca średnią arytmetyczną.

```
// Zwraca średnią podanych wartości.
public static double mean(double[] vals) {
  double avg = 0.0;

  for(int i=0; i < vals.length; i++)
    avg += vals[i];

  avg /= vals.length;

  return avg;
}
```

Aby użyć metody `mean()`, wystarczy tylko przekazać referencję do tablicy zawierającej zbiór wartości. Jako wynik wykonania metody otrzymuje się wartość średniej arytmetycznej elementów przekazanej tablicy.

Mediana

Mediana próbki jest określana jako wartość środkowa dla uporządkowanego rosnąco ciągu wartości. Na przykład dla zbioru

1 2 3 4 5 6 7 8 9

wartość mediany wynosi 5. W przypadku parzystej liczby elementów medianą jest średnia arytmetyczna dwóch środkowych wartości. Na przykład dla zbioru

1 2 3 4 5 6 7 8 9 10

wartość mediany wynosi 5.5. W przypadku próbek o rozkładzie normalnym wartości mediany i średniej są podobne. Im jednak rozkład próbki będzie bardziej odbiegał od rozkładu normalnego, tym różnica między medianą a średnią będzie większa.

Najprostszym sposobem otrzymania mediany z próbki jest posortowanie danych a następnie pobranie środkowej wartości. Oto sposób działania metody `median()`.

```
// Zwraca medianę podanych wartości.
public static double median(double[] vals) {
  double temp[] = new double[vals.length];

  System.arraycopy(vals, 0, temp, 0, vals.length);

  Arrays.sort(temp); // sortowanie danych

  // Zwrócenie wartości środkowej.
  if((vals.length)%2==0) {
    // Jeśli nieparzysta liczba wartości, znajdź średnią
    return (temp[temp.length/2] +
            temp[(temp.length/2)-1]) /2;
  } else return
    temp[temp.length/2];
}
```

Aby użyć metody `median()`, wystarczy tylko przekazać referencję do tablicy zawierającej zbiór wartości. Jako wynik wykonania metody otrzymuje się wartość mediany.

Warto zauważyć, iż prezentowana metoda wykonuje kopię przekazanej tablicy za pomocą polecenia `System.arraycopy()`. Sortowaniu podlega właśnie kopia. W ten sposób przekazana tablica nie ulega modyfikacji. Zachowanie oryginalnej kolejności jest bardzo ważne, na przykład w przypadku tworzenia wykresu.

Moda (dominanta)

Moda próbki jest wartością najczęściej występującą w zbiorze. Na przykład dla próbki

1 2 3 3 4 5 6 7 7 7 8 9

modą jest wartość 7, ponieważ występuje najczęściej. Moda nie musi być unikalna. Na przykład w sytuacji

10 20 30 30 40 50 60 60 70

zarówno wartość 30, jak i 60 występuje po dwa razy. Każda z tych wartości jest modą. Taki zbiór nazywany jest *bimodalnym*. Zbiór zawierający tylko jedną modą jest zbiorem *unimodalnym*. W prezentowanych przykładach zostanie zastosowane podejście, w którym w przypadku wielu wartości mody zostanie zwrócona tylko pierwsza z nich. Jeżeli żadna z wartości nie występuje częściej od innych, próbka nie posiada mody.

Poniższa metoda `mode()` znajduje modę zbioru.

```
/* Zwraca modę (dominantę) dla podanych wartości.
   Wyjątek NoModeException jest zgłaszany wtedy, gdy
   żadna z liczb nie występuje częściej od pozostałych.
   Gdy dwie lub więcej wartości pojawia się równie często,
   zwracana jest tylko pierwsza z nich. */
public static double mode(double[] vals)
    throws NoModeException
{
  double m, modeVal = 0.0;
  int count, oldcount = 0;

  for(int i=0; i < vals.length; i++) {
    m = vals[i];
    count = 0;

    // Zliczanie występowania poszczególnych wartości.
    for(int j=i+1; j < vals.length; j++)
      if(m == vals[j]) count++;

    /* Jeśli ta wartość występuje częściej niż poprzednie
       zapamiętaj ją. */
    if(count > oldcount) {
      modeVal = m;
      oldcount = count;
    }
  }

  if(oldcount == 0)
    throw new NoModeException();
  else
    return modeVal;
}
```

Metoda `mode()` zlicza najpierw liczbę wystąpień poszczególnych wartości w tablicy `vals`. Jeżeli znajdzie wartość występującą więcej razy niż poprzednia, zapamiętuje nową wartość w `modeVal`. Po zakończeniu procesu najczęściej występująca wartość znajduje się w zmiennej `modeVal` i jest ona zwracana. W przypadku wstępowania wielu wartości mody, metoda zwraca tylko pierwszą. Jeżeli próbka nie zawiera dominanty, następuje zgłoszenie wyjątku `NoModeException`. Oto postać klasy `NoModeException`.

```
// Wyjątek zgłaszany przez Mode()
class NoModeException extends Exception {
  public String toString() {
    return "Zbiór nie zawiera mody.";
  }
}
```

Wariancje i odchylenie standardowe

Choć podsumowanie zbioru zmiennych losowych jedną wartością, taką jak średnia lub mediana wydaje się być bardzo przekonujące, takie podejście nie zapewnia możliwości wykonania wyczerpującej analizy danych. Czasem nawet takie rozwiązanie może być mylące. Jeżeli próbka zawiera na przykład wartości skrajne, wtedy średnia i mediana nie reprezentuje jej w wystarczający sposób. Oto przykład

10 11 9 1 0 2 3 12 11 10

Średnia wynosi 6,9, ale wartość ta raczej jest niedostateczną reprezentacją próbki, ponieważ żadna ze zmiennych losowych nie jest nawet w przybliżeniu równa średniej. Problem polega na tym, iż średnia nie przekazuje informacji na temat wariacji lub rozkładu danych. Dobrze jest znać odstępy pomiędzy wartościami poszczególnych zmiennych losowych. W ten sposób lepiej można zinterpretować średnią, medianę i modę.

Aby znaleźć stopień zmienności próbki, trzeba obliczyć odchylenie standardowe. Odchylenie standardowe uzyskiwane jest po obliczeniu wariancji. Obie wartości określają rozkład danych w zbiorze. Z tych dwóch współczynników odchylenie standardowe jest ważniejsze, gdyż wyznacza średnią odległość pomiędzy wartościami zmiennych losowych a średnią.

Wariancję oblicza się z następującego wzoru.

$$M = \frac{1}{N}\sum_{i=1}^{N}(D_i - M)^2$$

W powyższym wzorze N jest liczbą elementów, M jest średnią a D_i — to wartości poszczególnych zmiennych losowych w próbie. Konieczne jest podnoszenie wyniku do kwadratu, aby uzyskiwać tylko wartości dodatnie. Jeżeli wzór nie uwzględniałby podnoszenia do kwadratu, wynik często wynosiłby zero.

Odchylenie standardowe znajduje się obliczając pierwiastek kwadratowy wariancji. Z tego powodu wzór na odchylenie standardowe ma następującą postać.

$$std = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(D_i - M)^2}$$

Jak już wspomniano wcześniej, odchylenie standardowe jest bardziej przydatne od wariancji. Warto rozważyć następujący zbiór:

11 20 40 30 99 30 50

Wariancję określa się jako średnią arytmetyczną kwadratów odchyleń poszczególnych wartości zmiennych od ich wartości średniej (zobacz tabelę na następnej stronie).

Z powyższej tabeli wynika, iż średnia kwadratów różnic wartości zmiennych losowych od ich wartości średniej wynosi 717,43. Aby znaleźć teraz odchylenie standardowe, wystarczy znaleźć pierwiastek kwadratowy wariancji. Wynosi on około 26,78. Odchylenie standardowe określa średnią odległość poszczególnych punktów od średniej wartości wszystkich elementów.

D_i	D_i–M	$(D_i–M)^2$
11	–29	841
20	–20	400
40	0	0
30	–10	100
99	59	3481
30	–10	100
50	10	100
	Suma:	5022
	Średnia sumy:	717.43

Odchylenie standardowe informuje o tym, czy wartość średniej dla elementów dobrze reprezentuje zbiór. Jeśli na przykład kupiono fabrykę batoników a kierownik informuje o tym, iż w poprzednim miesiącu wyprodukowano średnio 2 500 batoników dziennie ale odchylenie standardowe wyniosło 2 000, należy się zastanowić nad poprawą linii produkcyjnej.

Oto bardzo ważna zasada. Przy założeniu, iż dane stosują się do rozkładu normalnego, około 68% zmiennych losowych znajdzie się w pojedynczym odchyleniu standardowym od średniej a koło 95% znajdzie się w podwójnym odchyleniu standardowym.

Przedstawiona poniżej metoda `stdDev()` oblicza odchylenie standardowe dla tablicy wartości.

```
// Zwraca odchylenie standardowe zbioru wartości.
public static double stdDev(double[] vals) {
  double std = 0.0;
  double avg = mean(vals);

  for(int i=0; i < vals.length; i++)
    std += (vals[i]-avg) * (vals[i]-avg);

  std /= vals.length;
  std = Math.sqrt(std);

  return std;
}
```

Równanie regresji

Jednym z najczęstszych zastosowań statystyki jest prognozowanie przyszłości. Choć dane pochodzące z przeszłości nie zawsze pozwalają na przewidywanie przyszłości, często korzysta się z tak zwanej *analizy trendu*. Prawdopodobnie najbardziej rozpowszechnionym narzędziem do tego rodzaju analizy jest *równanie regresji*. Równanie to opisuje związek dwóch wartości, które wchodzą w skład dwuwymiarowej zmiennej losowej. Często linię tę nazywa się *linią najmniejszych kwadratów*.

Przed zaprezentowaniem odpowiedniego wzoru warto przypomnieć, iż linia prosta w dwóch wymiarach jest opisywana następującym równaniem

$$Y = a + bX$$

W tym przypadku X jest zmienną niezależną, Y jest zmienną zależną, *a* opisuje wartość przesunięcia na osi Y a *b* — stopień nachylenia linii. Aby w pełni określić położenie linii w układzie współrzędnych, trzeba odnaleźć wartości współczynników *a* i *b*.

Aby znaleźć równanie regresji, można skorzystać z metody najmniejszych kwadratów. Ogólnie pomysł polega na tym, by odnaleźć linię, która zminimalizuje sumę kwadratów odchyleń między poszczególnymi danymi a linią. Znalezienie tego równania wymaga dwóch kroków. Najpierw oblicza się wartość *b*, używając wzoru:

$$b = \frac{\sum_{i=1}^{N} (X_i - M_x)(Y_i - M_y)}{\sum_{i=1}^{N} (X_i - M_x)^2}$$

W powyższym wzorze M_x jest wartością średnią współrzędnej X, M_y — średnią współrzędnej Y. Po obliczeniu *b* wartość *a* oblicza się z następującego wzoru.

$$a = M_y - bM_x$$

Po określeniu równania regresji liniowej można użyć dowolnej wartości X i wskazać dla niej przewidywaną wartość Y.

Aby zrozumieć znaczenie linii regresji, można rozważyć następujący przykład. Załóżmy, że mamy dostęp do średnich wartości akcji firmy XYZ w przeciągu ostatnich 10 lat. Oto zebrane dane:

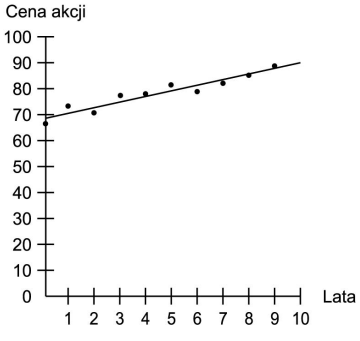
Rok	Cena
0	68
1	75
2	74
3	80
4	81
5	85
6	82
7	87
8	91
9	94

Równanie regresji dla tych danych jest następujące.

$$Y = 70{,}22 + 2{,}55\ X$$

Dane i linię regresji przedstawiono na rysunku 8.1. Jak wynika z rysunku, linia regresji jest nachylona do góry. Oznacza to tendencję wzrostową akcji firmy XYZ. Warto także zauważyć, iż punkty reprezentujące zmienne losowe (dane) są położone bardzo blisko linii regresji. Analizując przebieg linii można przewidzieć, iż w roku 11 cena akcji wzrośnie do 98,27. Aby uzyskać tę wartość, wystarczy w równaniu regresji zamiast X wstawić wartość 11. Oczywiście rozwiązanie to ma jedną wadę — to tylko przewidywanie. Nie ma gwarancji, iż rzeczywiście cena akcji za dwa lata będzie dokładnie taka.

Rysunek 8.1.
Wykres średnich cen i linia regresji



Współczynnik korelacji

Choć linia regresji z rysunku 8.1 wydaje się wskazywać tendencję wzrostową, nie wiadomo, z jakim stopniem dokładności odwzorowuje ona zebrane dane. Jeżeli dane i linia regresji są ze sobą słabo powiązane, wykres regresji liniowej nie jest dobrym wskaźnikiem. Jeżeli jednak punkty reprezentujące dane leżą dokładnie na linii wykresu, ma on dużą wartość.

Aby określić znaczenie otrzymanej linii regresji, najczęściej oblicza się *współczynnik korelacji*, który może przyjmować wartości od –1 do 1. Współczynnik ten opisuje siłę liniowego związku między dwiema zmiennymi. Brzmi to być może dziwnie, ale jest to bardzo proste zagadnienie. Graficznie współczynnik korelacji jest związany z odległością poszczególnych punktów reprezentujących dane (zmienne losowe) od linii, stanowiącej wykres regresji. Jeżeli współczynnik wynosi 1, dane są doskonale dopasowane do linii (doskonała korelacja dodatnia). Wartość 0 oznacza brak związku między linią a danymi (w takim przypadku każda dowolna linia prosta byłaby równie dobra jak rozważana). Znak korelacji jest taki sam jak współczynnik nachylenia linii (wartość *b*). Jeżeli współczynnik ten jest dodatni, oznacza to bezpośredni związek zmiennej zależnej od zmiennej niezależnej. Dla ujemnego współczynnika występuje związek odwrotny.

Wzór służący do obliczania współczynnika korelacji jest następujący.

$$b=\frac{\frac{1}{N}\sum_{i=1}^{N}(X_i-M_x)(Y_i-M_y)}{\sqrt{\sum_{i=1}^{N}(X_i-M_x)^2}\sqrt{\sum_{i=1}^{N}(Y_i-M_y)^2}}$$

We wzorze M_x oznacza średnią współrzędnej X, M_y — średnią współrzędnej Y. Znak współczynnika zależy od nachylenia linii regresji. Ogólnie wartość wynosząca 0,81 lub więcej jest traktowana jako silny związek. Oznacza to, iż około 66% danych jest dopasowanych do linii regresji. Aby zamienić współczynnik na wartość procentową, wystarczy podnieść go do kwadratu i pomnożyć przez 100. Uzyskana w ten sposób wartość jest nazywana *współczynnikiem determinacji*.

Metoda `regress()` przedstawiona poniżej oblicza równanie regresji oraz współczynnik korelacji.

```
/* Obliczanie regresji i współczynnik korelacji
   dla podanego zbioru wartości. Wartości reprezentują
   współrzędną Y. Współrzędną X jest czas
   (inkrementacja o jeden w każdym kroku). */
public static RegData regress(double[] vals) {
  double a, b, yAvg, xAvg, temp, temp2, cor;
  double vals2[] = new double[vals.length];

  // Utworzenie formatu liczbowego z dwoma cyframi po przecinku.
  NumberFormat nf = NumberFormat.getInstance();
  nf.setMaximumFractionDigits(2);

  // Znalezienie średniej wartości Y.
  yAvg = mean(vals);

  // Znalezienie średniej komponentu X.
  xAvg = 0.0;
  for(int i=0; i < vals.length; i++) xAvg += i;
  xAvg /= vals.length;

  // Znalezienie b.
  temp = temp2 = 0.0;
  for(int i=0; i < vals.length; i++) {
    temp += (vals[i]-yAvg) * (i-xAvg);
    temp2 += (i-xAvg) * (i-xAvg);
  }

  b = temp/temp2;

  // Znalezienie a.
  a = yAvg - (b*xAvg);

  // Obliczenie współczynników korelacji.
  for(int i=0; i < vals.length; i++) vals2[i] = i+1;
  cor = temp/vals.length;
  cor /= stdDev(vals) * stdDev(vals2);

  return new RegData(a, b, cor, "Y = " +
                     nf.format(a) +  " + " +
                     nf.format(b) + " * X");
}
```

Należy podkreślić, iż metoda opiera się na założeniu, że zmienną niezależną (X) jest czas. Z tego powodu w każdym kroku zwiększa wartość tej zmiennej o 1. Średnia wartości X jest obliczana następującymi poleceniami.

```
// Znalezienie średniej komponentu X.
xAvg = 0.0;
for(int i=0; i < vals.length; i++) xAvg += i;
xAvg /= vals.length;
```

Wartości od 0 do liczby elementów są ze sobą sumowane a następnie wynik tej operacji jest dzielony przez liczbę elementów. W ten sposób powstaje średnia X.

Oś X oznacza czas, zatem często taką analizę nazywa się analizą czasową. Wartością odniesienia podczas próbkowania jest czas, zatem do metody przekazuje się tylko jedną tablicę. Możliwe jest stosowanie dwóch tablic: jednej dla wartości X i jednej dla wartości Y, ale taki sposób działania nie jest rozpatrywany w niniejszym rozdziale.

Metoda `regress()` zwraca wartości *a* i *b*, tekstową reprezentację równania regresji i współczynnik korelacji, wszystkie dane zostają umieszczone w obiekcie `RegData`. Klasę tego obiektu przedstawiono poniżej.

```
// Ta klasa przechowuje dane analizy regresyjnej.
class RegData {
  public double a, b;
  public double cor;
  public String equation;

  public RegData(double i, double j, double k, String str) {
    a = i;
    b = j;
    cor = k;
    equation = str;
  }
}
```

Cała klasa Stats

Wszystkie metody statystyczne znajdują się w jednej klasie `Stats`, której kod przedstawiono poniżej. W tym samym pliku źródłowym umieszczono także klasy `RegData` i `NoModeException`.

```
import java.util.*;
import java.text.*;

// Ta klasa przechowuje dane analizy regresyjnej.
class RegData {
  public double a, b;
  public double cor;
  public String equation;

  public RegData(double i, double j, double k, String str) {
    a = i;
    b = j;
    cor = k;
    equation = str;
  }
}
```

```
// Wyjątek zgłaszany przez Mode()
class NoModeException extends Exception {
  public String toString() {
    return "Zbiór nie zawiera mody.";
  }
}

// Ogólna klasa statystyk.
public class Stats {
  // Zwraca średnią podanych wartości.
  public static double mean(double[] vals) {
    double avg = 0.0;

    for(int i=0; i < vals.length; i++)
      avg += vals[i];

    avg /= vals.length;

    return avg;
  }

  // Zwraca medianę podanych wartości.
  public static double median(double[] vals) {
    double temp[] = new double[vals.length];

    System.arraycopy(vals, 0, temp, 0, vals.length);

    Arrays.sort(temp); // sortowanie danych

    // Zwrócenie wartości środkowej.
    if((vals.length)%2==0) {
      // Jeśli nieparzysta liczba wartości, znajdź średnią
      return (temp[temp.length/2] +
              temp[(temp.length/2)-1]) /2;
    } else return
      temp[temp.length/2];
  }

  /* Zwraca modę (dominantę) dla podanych wartości.
     Wyjątek NoModeException jest zgłaszany wtedy, gdy
     żadna z liczb nie występuje częściej od pozostałych.
     Gdy dwie lub więcej wartości pojawia się równie często,
     zwracana jest tylko pierwsza z nich. */
  public static double mode(double[] vals)
      throws NoModeException
  {
    double m, modeVal = 0.0;
    int count, oldcount = 0;

    for(int i=0; i < vals.length; i++) {
      m = vals[i];
      count = 0;

      // Zliczanie występowań poszczególnych wartości.
      for(int j=i+1; j < vals.length; j++)
        if(m == vals[j]) count++;
```

```
    /* Jeśli ta wartość występuje częściej niż poprzednie
       zapamiętaj ją. */
    if(count > oldcount) {
      modeVal = m;
      oldcount = count;
    }
  }

  if(oldcount == 0)
    throw new NoModeException();
  else
    return modeVal;
}

// Zwraca odchylenie standardowe zbioru wartości.
public static double stdDev(double[] vals) {
  double std = 0.0;
  double avg = mean(vals);

  for(int i=0; i < vals.length; i++)
    std += (vals[i]-avg) * (vals[i]-avg);

  std /= vals.length;
  std = Math.sqrt(std);

  return std;
}

/* Obliczanie regresji i współczynnika korelacji
   dla podanego zbioru wartości. Wartości reprezentują
   współrzędną Y. Współrzędną X jest czas
   (inkrementacja o jeden w każdym kroku). */
public static RegData regress(double[] vals) {
  double a, b, yAvg, xAvg, temp, temp2, cor;
  double vals2[] = new double[vals.length];

  // Utworzenie formatu liczbowego z dwoma cyframi po przecinku.
  NumberFormat nf = NumberFormat.getInstance();
  nf.setMaximumFractionDigits(2);

  // Znalezienie średniej wartości Y.
  yAvg = mean(vals);

  // Znalezienie średniej komponentu X.
  xAvg = 0.0;
  for(int i=0; i < vals.length; i++) xAvg += i;
  xAvg /= vals.length;

  // Znalezienie b.
  temp = temp2 = 0.0;
  for(int i=0; i < vals.length; i++) {
    temp += (vals[i]-yAvg) * (i-xAvg);
    temp2 += (i-xAvg) * (i-xAvg);
  }

  b = temp/temp2;
```

```
        // Znalezienie a.
        a = yAvg - (b*xAvg);

        // Obliczenie współczynników korelacji.
        for(int i=0; i < vals.length; i++) vals2[i] = i+1;
        cor = temp/vals.length;
        cor /= stdDev(vals) * stdDev(vals2);

        return new RegData(a, b, cor, "Y = " +
                           nf.format(a) +  " + " +
                           nf.format(b) + " * X");
    }
}
```

Tworzenie wykresów

Choć statystyka jest użyteczna sama w sobie, nie zawsze daje pełny obraz sytuacji. W wielu przypadkach pożądane jest przedstawienie danych w postaci graficznej. Pokazanie danych na wykresie pozwala od razu zauważyć związki i anomalie, które nie zawsze są oczywiste po przeanalizowaniu statystyk. Poza tym wykres dokładnie obrazuje rozkład i zmienność danych. Z powodu dużego znaczenia wykresów w statystyce, w tym podrozdziale zostanie zaprezentowany kod rysujący trzy rodzaje wykresów.

Poza samym wyświetlaniem wykresów, przedstawiony w tym podrozdziale kod obrazuje także zasady korzystania z systemu AWT i obsługi zdarzeń. AWT stanowi część podstawowej biblioteki klas Javy. Zapewnia okienkowe, graficzne środowisko interfejsu użytkownika. Aplikacje z takim interfejsem komunikują się z użytkownikiem za pomocą zdarzeń. Zdarzeniem może być naciśnięcie przycisku na klawiaturze, wybranie polecenia z menu a także zmiana rozmiaru okna. W trakcie omawiania różnych metod rysowania wskażemy także kilka aspektów związanych ze środowiskiem interfejsu graficznego. Na przykład trzeba umożliwić dynamiczne skalowanie wykresu, ponieważ użytkownik może dowolnie zmienić rozmiar okna.

W tym rozdziale zostanie zaprezentowane wykonanie trzech rodzajów wykresów. Pierwszym z nich jest wykres słupkowy, drugi to wykres punktowy a trzeci — to wykres punktowy z linią regresji. Jak będzie się można przekonać, większość kodu, na przykład ten zapewniający skalowanie, jest taka sama dla wszystkich wykresów.

Skalowanie danych

Aby metoda rysowania mogła obsługiwać dowolne jednostki wielkości, potrzebne jest odpowiednie skalowanie danych. Skalowanie musi być powiązane z aktualnym rozmiarem okna. Co więcej, skalowanie musi się odbywać dynamicznie. Każda zmiana rozmiaru okna powinna powodować dostosowanie skali.

Proces skalowania wymaga znalezienia stosunku między zakresem danych a fizycznymi wymiarami okna. Gdy odnajdzie się ten współczynnik, dane można rysować, mnożąc

przez niego poszczególne wartości. W ten sposób uzyskuje się wartości współrzędnych wewnątrz okna. Oto przykład wzoru na skalowanie dla osi Y.

Y' = Y * (szerokość_okna / (max – min))

gdzie Y' jest skalowaną wartością, opisująca położenie wewnątrz okna.

Choć przedstawiony wzór jest bardzo prosty, pojawiają się komplikacje związane ze środowiskiem graficznym. Na przykład szerokość okna trzeba pobierać za każdym razem, gdy następuje ponowne rysowanie wykresu, gdyż szerokość ta mogła ulec zmianie. Co więcej, od łącznej szerokości okna trzeba odjąć szerokość obramowania okna. Trzeba także uwzględnić opis wykresu. W ten sposób skalowanie wymaga wykonania kilku kroków, ale nie są one szczególnie złożone.

Klasa Graphs

Metody rysowania wykresów znajdują się w klasie `Graphs`. Klasa ta rozszerza klasę `Frame`. Z tego powodu wykresy znajdują się wewnątrz głównych okien. Czyni to wykresy niezależnymi i łatwo skalowalnymi. Można na przykład wyświetlić wykres a następnie go zminimalizować bez chowania całej aplikacji.

Poniżej znajduje się pełny kod klasy `Graphs`. Klasę tę dokładniej opisano w dalszej części rozdziału.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.*;

// Ogólna klasa rysowania wykresów.
public class Graphs extends Frame {
  // Stałe rodzajów wykresów.
  public final static int BAR = 0;
  public final static int SCATTER = 1;
  public final static int REGPLOT = 2;

  private int graphStyle;

  /* Określa ilość miejsca do pozostawienia
     między obszarem danych a krawędzią. */
  private final int leftGap = 2;
  private final int topGap = 2;
  private final int bottomGap = 2;
  private int rightGap; // obliczanie tej wartości

  // Przechowuje wartość minimalną i maksymalną danych.
  private double min, max;

  // Odniesienie się do danych.
  private double[] data;

  // Kolory wykresu.
  Color gridColor = new Color(0, 150, 150);
  Color dataColor = new Color(0, 0, 0);
```

```
// Różne wartości używane do skalowania i wyświetlania danych.
private int hGap;   // odstęp między punktami danych
private int spread; // odstęp między wartościami min i max
private double scale; // współczynnik skalowania
private int baseline; // pionowy współczynnik podstawy

// Lokalizacja obszaru danych w oknie.
private int top, bottom, left, right;

public Graphs(double[] vals, int style) {
  // Obsługa zdarzeń zamknięcia okna.
  addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent we) {
      setVisible(false);
      dispose();
    }
  });

  // Obsługa zmiany rozmiaru.
  addComponentListener(new ComponentAdapter() {
    public void componentResized(ComponentEvent ce) {
      repaint();
    }
  });

  graphStyle = style;

  data = vals;

  // Sortowanie danych w celu znalezienia wartości min i max.
  double t[] = new double[vals.length];
  System.arraycopy(vals, 0, t, 0, vals.length);
  Arrays.sort(t);
  min = t[0];
  max = t[t.length-1];

  setSize(new Dimension(200, 120));

  switch(graphStyle) {
    case BAR:
      setTitle("Wykres słupkowy");
      setLocation(25, 250);
      break;
    case SCATTER:
      setTitle("Wykres punktowy");
      setLocation(250, 250);
      break;
    case REGPLOT:
      setTitle("Wykres regresji");
      setLocation(475, 250);
      break;
  }

  setVisible(true);
}
```

```
public void paint(Graphics g) {

  Dimension winSize = getSize(); // rozmiar okna
  Insets ins = getInsets(); // rozmiar granic

  // Rozmiar aktualnie stosowanej czcionki.
  FontMetrics fm = g.getFontMetrics();

  // Obliczenie odpowiedniego odstępu.
  rightGap = fm.stringWidth("" + data.length);

  // Obliczenie wszystkich przerw dla danego regionu.
  left = ins.left + leftGap + fm.charWidth('0');
  top = ins.top + topGap + fm.getAscent();
  bottom = ins.bottom + bottomGap + fm.getAscent();
  right = ins.right + rightGap;

  /* Jeśli wartość minimalna jest dodatnia, użyj 0
     jako miejsca początkowego.
     jeśli wartość maksymalna jest ujemna, użyj 0. */
  if(min > 0) min = 0;
  if(max < 0) max = 0;

  /* Oblicz odległość między minimum i maksimum. */
  spread = (int) (max - min);

  // Oblicz współczynnik skalowania.
  scale = (double) (winSize.height - bottom - top) / spread;

  // Poznaj położenie podstawy.
  baseline = (int) (winSize.height - bottom + min * scale);

  // Oblicz odległość między danymi.
  hGap = (winSize.width - left - right) / (data.length-1);

  // Ustaw kolor siatki.
  g.setColor(gridColor);

  // Narysuj współrzędne.
  g.drawLine(left, baseline,
             left + (data.length-1) * hGap, baseline);

  // Narysuj oś Y.
  if(graphStyle != BAR)
    g.drawLine(left, winSize.height-bottom, left, top);

  // Wyświetl  wartości min, max i 0.
  g.drawString("0", ins.left, baseline+fm.getAscent()/2);

  if(max != 0)
    g.drawString("" + max, ins.left, baseline -
                 (int) (max*scale) - 4);
  if(min != 0)
    g.drawString("" + min, ins.left, baseline -
                 (int) (min*scale)+fm.getAscent());
```

```
    // Wyświetl wartości.
    g.drawString("" + data.length,
                 (data.length-1) * (hGap) + left,
                 baseline + fm.getAscent());

    // Ustaw kolor danych.
    g.setColor(dataColor);

    // Wyświetl dane.
    switch(graphStyle) {
      case BAR:
        bargraph(g);
        break;
      case SCATTER:
        scatter(g);
        break;
      case REGPLOT:
        regplot(g);
        break;
    }
  }

  // Wyświetlanie wykresu słupkowego.
  private void bargraph(Graphics g) {
    int v;

    for(int i=0; i < data.length; i++) {
      v = (int) (data[i] * scale);

      g.drawLine(i*hGap+left, baseline,
                 i*hGap+left, baseline - v);
    }
  }

  // Wyświetlanie wykresu punktowego.
  private void scatter(Graphics g) {
    int v;

    for(int i=0; i < data.length; i++) {
      v = (int) (data[i] * scale);
      g.drawRect(i*hGap+left, baseline - v, 1, 1);
    }
  }

  // Wykres punktowy z linią regresji.
  private void regplot(Graphics g) {
    int v;

    RegData rd = Stats.regress(data);

    for(int i=0; i < data.length; i++) {
      v = (int) (data[i] * scale);
      g.drawRect(i*hGap+left, baseline - v, 1, 1);
    }
```

```
    // Rysowanie linii regresji.
    g.drawLine(left, baseline - (int) ((rd.a)*scale),
               hGap*(data.length-1)+left+1,
               baseline - (int) ((rd.a+(rd.b*(data.length-1)))*scale));
  }
}
```

Zmienne klasy Graphs

Klasa `Graphs` zaczyna się od zdefiniowania następujących zmiennych.

```
// Stałe rodzajów wykresów.
public final static int BAR = 0;
public final static int SCATTER = 1;
public final static int REGPLOT = 2;

private int graphStyle;
```

Pierwsze trzy zmienne typu `final static` noszą nazwy `BAR`, `SCATTER` i `REGPLOT`. Wartości te oznaczają różne typy wykresów. Typ wykresu jest przechowywany w zmiennej `graphStyle`.

Następnie definiuje się zmienne przechowujące wielkość odstępu między granicą okna a początkiem obszaru wyświetlania danych na wykresie.

```
/* Określa ilość miejsca do pozostawienia
   między obszarem danych a krawędzią. */
private final int leftGap = 2;
private final int topGap = 2;
private final int bottomGap = 2;
private int rightGap; // obliczanie tej wartości
```

Wszystkie zmienne poza `rightGap` są typu `final`. Zmienna `rightGap` jest obliczana później, gdyż zależy ona od aktualnej szerokości znaków oraz liczby wyświetlanych danych.

Następnie deklarowane są następujące zmienne.

```
// Przechowuje wartość minimalną i maksymalną danych.
private double min, max;

// Odniesienie się do danych.
private double[] data;
```

Zmienne `min` i `max` przechowują minimalną i maksymalną wartość danych. Tablica przechowująca dane jest dostępna przez referencję `data`.

Kolory używane przez wykres znajdują się w zmiennych `gridColor` i `dataColor`.

```
// Kolory wykresu.
Color gridColor = new Color(0, 150, 150);
Color dataColor = new Color(0, 0, 0);
```

Linie siatki są jasnozielone, natomiast dane są zaznaczane na czarno. Oczywiście można zmienić te kolory na dowolne inne.

Następnie deklaruje się różne zmienne związane ze skalowaniem.

```
// Różne wartości używane do skalowania i wyświetlania danych.
private int hGap;   // odstęp między punktami danych
private int spread; // odstęp między wartościami min i max
private double scale; // współczynnik skalowania
private int baseline; // pionowy współczynnik podstawy
```

Odległość między punktami danych na osi X zawiera zmienna `hGap`. Liczbę jednostek między wartością minimalną a maksymalną przechowuje zmienna `spread`. Współczynnik skali znajduje się w zmiennej `scale`. Pionowe położenie linii bazy (czyli osi X) znajduje się w zmiennej `baseline`.

Na końcu zadeklarowano obszary `top`, `bottom`, `left` i `right`.

```
// Lokalizacja obszaru danych w oknie.
private int top, bottom, left, right;
```

Zmienne te definiują obszar danych wewnątrz okna.

Konstruktor klasy Graphs

Konstruktor przyjmuje dwa argumenty. Pierwszy to referencja do danych, które mają zostać wyświetlone. Drugim jest rodzaj wykresu. Przekazaną wartością musi być `Graph.BAR`, `Graph.SCATTER` lub `Graph.REGPLOT`.

W konstruktorze najpierw należy przypisać nasłuchiwanie zdarzenia zamykania okna.

```
// Obsługa zdarzeń zamknięcia okna.
addWindowListener(new WindowAdapter() {
  public void windowClosing(WindowEvent we) {
    setVisible(false);
    dispose();
  }
});
```

Nasłuchiwanie dodaje się przez wywołanie metody `addWindowListener()` i przekazanie obiektu `WindowAdapter`, w którym przysłonięto metodę `windowClosing()`. Warto przypomnieć, iż klasa adapterowa zawiera puste implementacje wszystkich metod wymaganych przez dany interfejs. W tym przypadku adapter `WindowAdapter` implementuje interfejs `WindowListener`. Adapter zapewnia puste implementacje wszystkich wymaganych metod, zatem trzeba tylko przysłonić odpowiednią metodę, w tym przypadku `windowClosing()`.

W przypadku zamykania okna należy je ukryć wywołując polecenie `setVisible(false)`. Następnie usuwa się okno z systemu poleceniem `dispose()`. Metody te są zdefiniowane dla okien `Frame` z AWT.

Następnie należy dodać nasłuchiwanie zdarzeń zmiany rozmiaru okna. Wywołuje się metodę `addComponentListener()` i przekazuje obiekt `ComponentAdapter`, który przysłania metodę `componentResized()`.

```
// Obsługa zmiany rozmiaru.
addComponentListener(new ComponentAdapter() {
  public void componentResized(ComponentEvent ce) {
    repaint();
  }
});
```

W przypadku zmiany rozmiaru okna metoda `componentResized()` wywołuje metodę `repaint()`, co natomiast powoduje wywołanie metody `paint()`. Jak za chwilę Czytelnik zobaczy, metoda `paint()` dynamicznie zmienia skalę na podstawie aktualnego rozmiaru okna. Po zmianie rozmiaru cały wykres jest rysowany od nowa z wykorzystaniem nowych wymiarów.

Następnie do zmiennych `graphStyle` i `data` należy przypisać otrzymane dane i utworzyć tymczasową kopię danych. Dane są sortowane w kopii tablicy. Z posortowanej tablicy danych pobiera się wartość minimalną i maksymalną. Kod wykonujący to zadanie przedstawiono poniżej.

```
graphStyle = style;

data = vals;

// Sortowanie danych w celu znalezienia wartości min i max.
double t[] = new double[vals.length];
System.arraycopy(vals, 0, t, 0, vals.length);
Arrays.sort(t);
min = t[0];
max = t[t.length-1];
```

Konstruktor klasy kończy się następującym kodem.

```
setSize(new Dimension(200, 120));

switch(graphStyle) {
  case BAR:
    setTitle("Wykres słupkowy");
    setLocation(25, 250);
    break;
  case SCATTER:
    setTitle("Wykres punktowy");
    setLocation(250, 250);
    break;
  case REGPLOT:
    setTitle("Wykres regresji");
    setLocation(475, 250);
    break;
}

setVisible(true);
```

Najpierw ustala się początkowy rozmiar okna wykresu na 200 pikseli w poziomie i 120 w pionie, wywołując metodę `setSize()`. Potem na podstawie wartości zmiennej `graphStyle` przypisuje się odpowiedni tytuł, używając metody `setTitle()`. Dodatkowo należy ustalić także położenie okna poleceniem `setLocation()`. Na samym końcu włącza się wyświetlanie okna poleceniem `setVisible(true)`. Powoduje to odrysowanie okna metodą `paint()`.

Metoda paint()

Większość zadań związanych z wyświetlaniem wykresu jest wykonywana przez metodę `paint()`. Wykonuje ona następujące działania.

- Odczytuje rozmiar okna oraz rozmiar granicy.
- Pobiera rozmiar aktualnie wybranej czcionki.
- Oblicza rozmiar obszaru danych wewnątrz okna. Jest to rozmiar okna minus granice i ewentualne odstępy.
- Oblicza współczynnik skali.
- Oblicza współrzędną Y podstawy wykresu, czyli oś X.
- Rysuje podstawę oraz oś Y.
- Wyświetla minimalne i maksymalne wartości X i Y.
- Wywołuje odpowiednią metodę rysowania w celu wykonania wykresu.

Komentarze w metodzie `paint()` wyjaśniają kolejne działania, ale sama metoda jest bardzo prosta. Jest to jednak najważniejsza metoda klas, zatem prześledzimy ją wiersz po wierszu.

Metoda zaczyna się następującymi deklaracjami.

```
Dimension winSize = getSize(); // rozmiar okna
Insets ins = getInsets(); // rozmiar granic
```

Okno `Frame` składa się z dwóch głównych części: granicy, w której skład chodzi obramowanie, pasek tytułowy i menu (jeśli istnieje) oraz głównego obszaru wyświetlania danych. Rozmiar okna pobiera się poleceniem `getSize()`. Metoda ta zwraca ogólne wymiary okna w postaci obiektu `Dimension`. Referencja do zwróconego obiektu znajdzie się w zmiennej `winSize`. Obiekt `Dimension` zawiera dwa pola: `width` i `height`. Z tego powodu ogólne wymiary okna odczytuje się jako `winSize.width` i `winSize.height`.

Aby znaleźć obszar okna, w którym można wyświetlać dane, trzeba od ogólnego rozmiaru okna odjąć granice. Do tego celu służy metoda `getInsets()`. Zwraca ona wymiary granic w postaci obiektu `Insets` zawierającego pola `left`, `right`, `top` i `bottom`. Referencja do tego obiektu jest przechowywana w `ins`. Należy pamiętać, iż współrzędnymi lewego górnego narożnika okna są 0, 0. Z tego powodu lewy górny narożnik obszaru danych to `ins.left` i `ins.top`, natomiast prawy dolny narożnik to `winSize.width-ins.right` i `winSize.height-ins.bottom`.

Następnie metoda `paint()` pobiera metrykę aktualnie wybranej czcionki.

```
// Rozmiar aktualnie stosowanej czcionki.
FontMetrics fm = g.getFontMetrics();
```

Informacje znajdujące się w zmiennej `fm` posłużą później do obliczenia wysokości i szerokości znaków używanych do wyświetlenia zakresu.

Choć granica określa maksymalny użyteczny obszar okna, nie zawsze jest to obszar najlepszy ze względów estetycznych. Najczęściej warto pozostawić niewielki odstęp miedzy danymi a granicą. Z tego powodu dostępny obszar dodatkowo redukujemy wartościami `leftGap`, `topGap`, `bottomGap` i `rightGap`. Trzy pierwsze zmienne zawierają wartość 2, ale ostatnią zmienną oblicza się na podstawie szerokości tekstu zawierającego liczbę elementów w zbiorze.

```
// Obliczenie odpowiedniego odstępu.
rightGap = fm.stringWidth("" + data.length);
```

Tekst wyświetlający liczbę elementów znajduje się po prawej stronie, zatem należy przygotować dla niego odpowiednią ilość miejsca. Właśnie z tego powodu należało pobrać metrykę czcionki poleceniem `getFontMetrics()`. Stosując metodę `stringWidth()` uzyskanego obiektu można ustalić odpowiedni odstęp i przypisać go do `rightGap`.

Teraz oblicza się łączne odstępy po każdej stronie, używając wszystkich zebranych wartości. Wyniki zapisujemy w zmiennych `left`, `right`, `top` i `bottom`.

```
// Obliczenie wszystkich przerw dla danego regionu.
left = ins.left + leftGap + fm.charWidth('0');
top = ins.top + topGap + fm.getAscent();
bottom = ins.bottom + bottomGap + fm.getAscent();
right = ins.right + rightGap;
```

Warto zauważyć, iż pozostawiono miejsce na wyświetlenie zakresu danych.

Następnych kilka wierszy oblicza współczynnik skali.

```
/* Jeśli wartość minimalna jest dodatnia, użyj 0
   jako miejsca początkowego.
   jeśli wartość maksymalna jest ujemna, użyj 0. */
if(min > 0) min = 0;
if(max < 0) max = 0;

/* Oblicz odległość między minimum i maksimum. */
spread = (int) (max - min);

// Oblicz współczynnik skalowania.
scale = (double) (winSize.height - bottom - top) / spread;
```

Proces zaczyna się od normalizacji wartości w `min` i `max`. Wszystkie wykresy mają swój początek w punkcie 0, 0. Jeżeli więc wartość minimalna jest większa od 0, należy ustawić `min` na 0. Jeżeli wartość maksymalna jest mniejsza od 0, należy ustawić `max` na 0. Następnie oblicza się odstęp między `min` i `max`. Uzyskaną wartość używa się do obliczenia współczynnika skalowania przechowywanego w zmiennej `scale`.

Po obliczeniu współczynnika skalowania położenie linii bazy znajduje się skalując wartość `min` w sposób przedstawiony poniżej.

```
// Poznaj położenie bazy.
baseline = (int) (winSize.height - bottom + min * scale);
```

Jeżeli `min` wynosi zero, linia bazy znajdzie się na dole okna. W innym przypadku znajdzie się gdzieś w jego środkowej części. Jeżeli wszystkie wartości są ujemne, znajdzie się na górze.

Odstęp między danymi jest określany za pomocą dzielenia szerokości obszaru danych przez liczbę elementów.

```
// Oblicz odległość między danymi.
hGap = (winSize.width - left - right) / (data.length-1);
```

W następnym kroku ustawia się aktualny kolor na `gridColor`. Rysuje się osie i zakresy. Oto kod wykonujący to zadanie.

```
// Ustaw kolor siatki.
g.setColor(gridColor);

// Narysuj współrzędne.
g.drawLine(left, baseline, left + (data.length-1) * hGap, baseline);

// Narysuj oś Y.
if(graphStyle != BAR)
  g.drawLine(left, winSize.height-bottom, left, top);

// Wyświetl  wartości min, max i 0.
g.drawString("0", ins.left, baseline+fm.getAscent()/2);

if(max != 0)
  g.drawString("" + max, ins.left, baseline - (int) (max*scale) - 4);
if(min != 0)
  g.drawString("" + min, ins.left, baseline - (int) (min*scale)+fm.getAscent());

// Wyświetl wartości.
g.drawString("" + data.length,
            (data.length-1) * (hGap) + left,
             baseline + fm.getAscent());
```

Ważnym jest, iż dla wykresu słupkowego nie wyświetla się osi Y. Poza tym maksymalny zakres wyświetla się tylko wtedy, jeżeli nie jest on zerowy. Podobnie ma się sprawa z zakresem minimum. Wysokość znaków określa ich położenie w oknie.

W kolejnym kroku należy ustawić kolor na `dataColor` i wywołać odpowiednią metodę rysowania wykresu.

```
// Ustaw kolor danych.
g.setColor(dataColor);

// Wyświetl dane.
switch(graphStyle) {
  case BAR:
    bargraph(g);
    break;
  case SCATTER:
    scatter(g);
    break;
  case REGPLOT:
    regplot(g);
    break;
}
```

Metoda bargraph()

Metoda `bargraph()` skaluje poszczególne elementy danych a następnie wyświetla linię, której długość jest proporcjonalna do tych danych. Linia jest rysowana od podstawy. Oto kod metody.

```
// Wyświetlanie wykresu słupkowego.
private void bargraph(Graphics g) {
  int v;

  for(int i=0; i < data.length; i++) {
    v = (int) (data[i] * scale);

    g.drawLine(i*hGap+left, baseline,
               i*hGap+left, baseline - v);
  }
}
```

Skoro tak wiele zadań zostało już wykonanych w metodzie `paint()`, metoda `bargraph()` tylko skaluje poszczególne elementy za pomocą współczynnika skalowania a następnie rysuje linie. Linie zaczynają się od podstawy, czyli od osi X. Punkt końca oblicza się przez odejmowanie skalowanej wartości od `baseline`. Należy pamiętać, iż współrzędne lewego górnego narożnika okna mają wartość 0, 0. Z tego powodu mniejsze wartości Y znajdują się wyżej w oknie niż większe wartości Y. Z powyższego wynika, że od `baseline` należy odjąć wartość `v`. Odstęp między słupkami wynosi `hGap`. Położenie X każdego słupka jest obliczane przez pomnożenie indeksu elementu przez wielkość odstępu i dodanie wartości przesunięcia względem lewej krawędzi (zmienna `left`).

Metoda scatter()

Metoda `scatter()` działa bardzo podobnie jak metoda `bargraph()` ale rysuje punkty zamiast linii. Oto kod metody.

```
// Wyświetlanie wykresu punktowego.
private void scatter(Graphics g) {
  int v;

  for(int i=0; i < data.length; i++) {
    v = (int) (data[i] * scale);
    g.drawRect(i*hGap+left, baseline - v, 1, 1);
  }
}
```

Metoda `scatter()` skaluje poszczególne elementy danych a następnie wyświetla punkt na osi Y, którego odległość od początku osi X jest proporcjonalna do tych danych.

Metoda regplot()

Podobnie jak metoda `scatter()`, metoda `regplot()` rysuje punkty na wykresie. Różnica polega na tym, iż dodatkowo rysuje także linię regresji, używając funkcji `regress()`.

```
// Wykres punktowy z linią regresji.
private void regplot(Graphics g) {
  int v;

  RegData rd = Stats.regress(data);

  for(int i=0; i < data.length; i++) {
    v = (int) (data[i] * scale);
    g.drawRect(i*hGap+left, baseline - v, 1, 1);
  }

  // Rysowanie linii regresji.
  g.drawLine(left, baseline - (int) ((rd.a)*scale),
             hGap*(data.length-1)+left+1,
             baseline - (int) ((rd.a+(rd.b*(data.length-1)))*scale));
}
```

Warto zwrócić uwagę na sposób rysowania linii regresji. W wywołaniu `drawLine()` punkt końcowy linii regresji jest obliczany na podstawie wartości `rd.a` i `rd.b`, czyli miejsca przecięcia osi Y i linii regresji.

Aplikacja tworzenia statystyk

Dzięki klasom `Stats` i `Graphs` można wykonać prostą, ale użyteczną aplikację. Główne okno aplikacji powstaje dzięki klasie `StatsWin`, której kod przedstawiono poniżej.

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.text.*;

// Przetworzenie i wyświetlenie danych statystycznych.
public class StatsWin extends Frame
    implements ItemListener, ActionListener  {

  NumberFormat nf = NumberFormat.getInstance();

  TextArea statsTA;
  Checkbox bar = new Checkbox("Wykres słupkowy");
  Checkbox scatter = new Checkbox("Wykres punktowy");
  Checkbox regplot = new Checkbox("Linia regresji");
  Checkbox datawin = new Checkbox("Pokaż dane");

  double[] data;

  Graphs bg;
  Graphs sg;
  Graphs rp;
  DataWin da;

  RegData rd;
```

```
public StatsWin(double vals[]) {
  data = vals; // zapamiętaj referencję do danych

  addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent we) {
      shutdown();
    }
  });

  // Utworzenie menu.
  createMenu();

  // Zmiana układu okna, centrowanie komponentów.
  setLayout(new FlowLayout(FlowLayout.CENTER));

  setSize(new Dimension(300, 240));
  setTitle("Statistical Data");

  rd = Stats.regress(data);

  // Ustawienie formatu z 2 miejscami po przecinku.
  nf.setMaximumFractionDigits(2);


  // Konstrukcja wyjścia.
  String mstr;
  try {
    // Pobranie mody, jeśli istnieje.
    mstr = nf.format(Stats.mode(data));
  } catch(NoModeException exc) {
    mstr = exc.toString();
  }

  String str = "Średnia: " +
               nf.format(Stats.mean(data)) + "\n" +
              "Mediana: " +
               nf.format(Stats.median(data)) + "\n" +
              "Moda: " + mstr + "\n" +
              "Odchyleniestandardowe: " +
               nf.format(Stats.stdDev(data)) + "\n\n" +
              "Równanie regresji: " + rd.equation +
              "\nWspółczynnik korelacji: " +
               nf.format(rd.cor);

  // Umieszczenie wyjścia w polu tekstowym.
  statsTA = new TextArea(str, 6, 38, TextArea.SCROLLBARS_NONE);
  statsTA.setEditable(false);

  // Dodanie komponentów do okna.
  add(statsTA);
  add(bar);
  add(scatter);
  add(regplot);
  add(datawin);

  // Dodanie nasłuchiwania.
  bar.addItemListener(this);
```

```
    scatter.addItemListener(this);
    regplot.addItemListener(this);
    datawin.addItemListener(this);

    setVisible(true);
  }

  // Obsługa zamykania okna.
  public void actionPerformed(ActionEvent ae) {
    String arg = (String)ae.getActionCommand();

    if(arg == "Close") {
      shutdown();
    }
  }

  // Użytkownik zmienił opcję.
  public void itemStateChanged(ItemEvent ie) {
    if(bar.getState()) {
      if(bg == null) {
        bg = new Graphs(data, Graphs.BAR);
        bg.addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent we) {
            bar.setState(false);
            bg = null;
          }
        });
      }
    }
    else {
      if(bg != null) {
        bg.dispose();
        bg = null;
      }
    }

    if(scatter.getState()) {
      if(sg == null) {
        sg = new Graphs(data, Graphs.SCATTER);
        sg.addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent we) {
            scatter.setState(false);
            sg = null;
          }
        });
      }
    }
    else {
      if(sg != null) {
        sg.dispose();
        sg = null;
      }
    }

    if(regplot.getState()) {
      if(rp == null) {
        rp = new Graphs(data, Graphs.REGPLOT);
```

```
        rp.addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent we) {
            regplot.setState(false);
            rp = null;
          }
        });
      }
    }
    else {
      if(rp != null) {
        rp.dispose();
        rp = null;
      }
    }

    if(datawin.getState()) {
      if(da == null) {
        da = new DataWin(data);
        da.addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent we) {
            datawin.setState(false);
            da = null;
          }
        });
      }
    }
    else {
      if(da != null) {
        da.dispose();
        da = null;
      }
    }
  }

  // Utworzenie menu.
  private void createMenu()
  {
    MenuBar mbar = new MenuBar();
    setMenuBar(mbar);

    Menu file = new Menu("Plik");
    MenuItem close = new MenuItem("Zamknij");
    file.add(close);
    mbar.add(file);
    close.addActionListener(this);
  }

  // Zamknięcie okien.
  private void shutdown() {
    if(bg != null) bg.dispose();
    if(sg != null) sg.dispose();
    if(rp != null) rp.dispose();
    if(da != null) da.dispose();
    setVisible(false);
    dispose();
  }
}
```

Klasa StatsWin rozszerza klasę Frame w celu utworzenia okna głównego, w którym będą wyświetlane informacje statystyczne. Zawiera także opcje umożliwiające użytkownikowi wybór sposobu prezentacji danych. Klasa implementuje interfejsy ItemListener i ActionListener.

Klasa zaczyna się od pobrania obiektu NumberFormat. Klasa NumberFormat jest pomocna w formatowaniu danych numerycznych. Klasa StatsWin używa tej klasy do określenia liczby miejsc po przecinku, jaka ma być wyświetlana.

W dalszej części klasa deklaruje kilka zmiennych przechowujących referencje do elementów graficznych interfejsu. Dotyczy to pola tekstowego, czterech opcji i trzech obiektów Graphs. Referencja do obiektu DataWin jest przechowywana w zmiennej da. Klasa DataWin jest oknem wyświetlającym analizowane dane numeryczne. Referencja do danych znajduje się w zmiennej data a referencja do danych regresji w zmiennej rd.

Konstruktor klasy StatsWin

Do konstruktora przekazuje się referencję do analizowanych danych. Konstruktor przeprowadza analizę statystyczną dla wprowadzonych danych. Większość kodu konstruktora jest prosta, więc zostanie omówiony tylko w ogólny sposób.

Na początku zapamiętuje się referencję do danych. Następnie dodaje się nasłuch dla zdarzenia zamknięcia okna. W przypadku wystąpienia takiego zdarzenia, następuje wywołanie metody shutdown(), która zamyka wszystkie okna otwarte przez StatsWin.

Następnie konstruktor tworzy menu, używając do tego metody createMenu(). Menu zawiera tylko jedno polecenie: *Zamknij*. Wybranie go powoduje wyłączenie aplikacji.

W kolejnym kroku ustalamy w menedżerze układu okna układ z centrowaniem. Jest to konieczne, gdyż klasa Frame domyślnie stosuje układ graniczny.

W następnym kroku ustawia się tytuł okna, jego rozmiary oraz pobiera się dane regresji. Należy także ustawić format liczbowy na dwa miejsca po przecinku, używając poniższego kodu.

```
nf.setMaximumFractionDigits(2);
```

Jak wspomniano wcześniej, nf odnosi się do obiektu NumberFormat. Jest to obiekt wykorzystywany między innymi do opisu wyświetlania wartości numerycznych. Metoda setMaximumFractionDigits() pozwala na ustalenie maksymalnej liczby cyfr wyświetlanej po przecinku. Klasa StatsWin wykorzystuje ten obiekt do ustawienia całego wyświetlania danych. Aby zobaczyć więcej miejsc po przecinku, należy przekazać większą wartość do metody setMaximumFractionDigits().

Kilka następnych wierszy tworzy wynikowy tekst z wynikami różnych analiz statystycznych.

```
// Konstrukcja wyjścia.
String mstr;
```

```
try {
  // Pobranie mody, jeśli istnieje.
  mstr = nf.format(Stats.mode(data));
} catch(NoModeException exc) {
  mstr = exc.toString();
}

String str = "Średnia: " +
             nf.format(Stats.mean(data)) + "\n" +
             "Mediana: " +
             nf.format(Stats.median(data)) + "\n" +
             "Moda: " + mstr + "\n" +
             "Odchylenie standardowe: " +
             nf.format(Stats.stdDev(data)) + "\n\n" +
             "Równanie regresji: " + rd.equation +
             "\nWspółczynnik korelacji: " +
             nf.format(rd.cor);
```

Interesujący jest sposób pobierania informacji o modzie. Należy pamiętać, iż metoda `mode()` zwraca wyjątek, gdy próbka nie zawiera wartości mody. Zmienna `mstr` zawiera albo informację o modzie, albo wyjaśnienie, iż moda dla danej próbki nie istnieje.

Po wykonaniu całego tekstu `str` jest on umieszczany w obiekcie `TextArea` dostępnym dzięki zmiennej `statsTA`. Obiekt ten następnie ustawia się na tryb tylko do odczytu za pomocą polecenia `setEditable(false)`. W ten sposób możliwy jest tylko odczyt informacji.

Następnie należy dodać do okna poszczególne elementy interfejsu oraz elementy nasłuchujące. Obszar tekstowy jest obszarem tylko do odczytu, zatem nie wymaga ustawienia nasłuchu. Na końcu należy wyświetlić okno.

Procedura obsługi itemStateChanged()

Wiele działań klasy `StatsWin` zachodzi w metodzie `itemStateChanged()`. Metoda ta obsługuje zmiany zaznaczenia czterech opcji. Gdy użytkownik włączy daną opcję, wyświetlane jest związane z nią okno. Gdy wyłączy opcję, dane okno jest zamykane. Aby zrozumieć sposób działania tego procesu, można prześledzić kod służący do obsługi zmian opcji wykresu słupkowego. Oto kod.

```
if(bar.getState()) {
  if(bg == null) {
    bg = new Graphs(data, Graphs.BAR);
    bg.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent we) {
        bar.setState(false);
        bg = null;
      }
    });
  }
}
else {
  if(bg != null) {
    bg.dispose();
    bg = null;
  }
}
```

Najpierw pobiera się stan obiektu za pomocą zmiennej `bar` i metody `getState()`. Jeżeli zostanie zwrócona wartość `true`, opcja jest włączona. W przeciwnym razie jest wyłączona. Jeżeli opcja jest włączona a zmienna `bg` zawiera wartość `null`, okno wykresu słupkowego jest włączane po raz pierwszy. W tym przypadku do `bg` przypisuje się nowy obiekt `Graphs` wyświetlający wykres słupkowy. Gdy `bg` jest różne od `null`, po prostu nic nie trzeba robić, gdyż okno istnieje.

W trakcie tworzenia nowego okna należy dodać nasłuch monitorujący okno. Interesująca jest informacja o zdarzeniu zamykania okna. W ten sposób obiekt `StatsWin` otrzymuje informację o zamykaniu okna wykresu. W momencie otrzymania powiadomienia o zamykaniu okna należy wyłączyć opcję i ustawić `bg` na `null`.

Jeżeli użytkownik wyłączył opcję a okno było otwarte, należy zamknąć okno metodą `dispose()` i ustawić `bg` na `null`. Ten sam mechanizm stosuje się dla wszystkich czterech opcji.

Metoda actionPerformed()

Metoda `actionPerformed()` obsługuje zamykanie programu za pomocą polecenia *Zamknij* z menu. Metoda po prostu wywołuje inną metodę o nazwie `shutdown()`.

Metoda shutdown()

W momencie zamykania okna `StatsWin` jest wywoływana metoda `shutdown()`. Zamyka ona wszystkie okna otwarte przez obiekt `StatsWin`, dotyczy to zarówno głównego okna, jak i okien wykresów i danych. Z tego powodu z ekranu znikają wszystkie wykresy, choć są one wyświetlane w niezależnych oknach.

Metoda createMenu()

Metoda `createMenu()` tworzy menu aplikacji. Najpierw tworzy się obiekt `MenuBar` i umieszcza się go w zmiennej `mbar`. Następnie powstaje obiekt `Menu` o nazwie `file`, w którym umieszcza się obiekt `MenuItem` o nazwie `close`. Następnie obiekt `StatsWin` staje się odbiorcą akcji z menu. W ten sposób zdarzenia akcji wygenerowane przez menu będą trafiały do opisanej wcześniej metody `actionPerformed()`.

Klasa DataWin

Klasa `StatsWin` wykorzystuje obiekt `DataWin` do wyświetlenia wejściowych danych liczbowych poddawanych analizie. Oto postać klasy `DataWin`.

```
import java.awt.event.*;
import java.awt.*;

// Wyświetla tablicę danych numerycznych.
class DataWin extends Frame {
  TextArea dataTA;
```

```
  DataWin(double[] data) {
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent we) {
        setVisible(false);
        dispose();
      }
    });

    dataTA = new TextArea(10, 10);
    dataTA.setEditable(false);

    for(int i=0; i < data.length; i++)
      dataTA.append(data[i]+"\n");

    setSize(new Dimension(100, 140));
    setLocation(320, 100);

    setTitle("Dane");
    setResizable(false);

    add(dataTA);

    setVisible(true);
  }
}
```

Klasa `DataWin` rozszerza klasę `Frame` i tworzy główne okno. Konstruktor klasy przyjmuje jako parametr referencję do tablicy danych do wyświetlenia. Następnie tworzy obiekt `TextArea` wyświetlający dane. Obszar tekstowy jest ustawiany tylko do odczytu bez możliwości zmiany skali. Okno można zminimalizować.

Łączymy wszystko razem

Poniższy program przedstawia sposób wykorzystania klas `Stats` i `Graphs`.

```
// Demonstruje działanie klas Stats i Graphs.
import java.io.*;
import java.awt.*;

class DemoStat {
  public static void main(String args[])
    throws IOException
  {
    double nums[] = { 10, 10, 11, 9, 8, 8, 9,
                      10, 10, 13, 11, 11, 11,
                      11, 12, 13, 14, 16, 17,
                      15, 15, 16, 14, 16 };

    new StatsWin(nums);
  }
}
```

Aby skompilować program, należy wydać następujące polecenie:

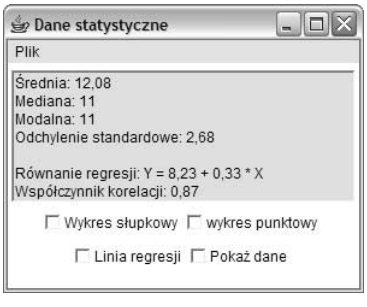
```
javac DemoStat.java DataWin.java StatsWin.java Stats.java Graphs.java
```

Aby uruchomić program, należy wpisać następujące polecenie.

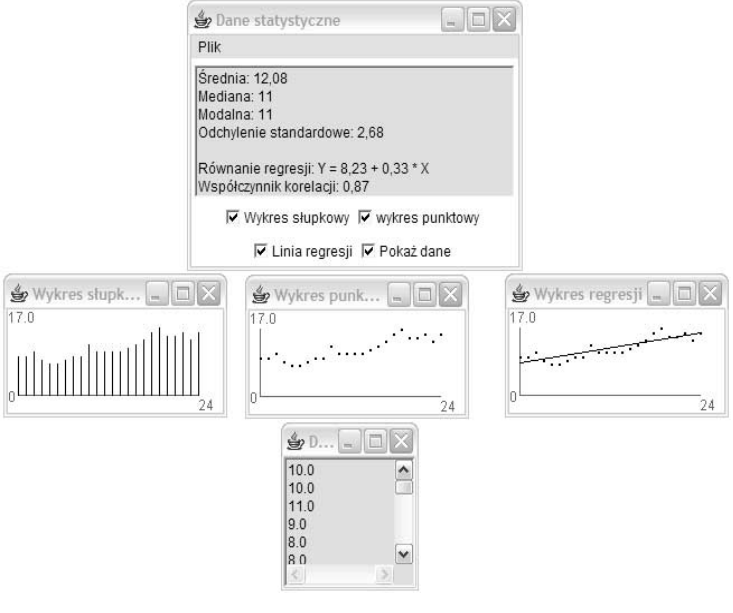
```
javaw DemoStat
```

Warto zauważyć, iż używamy programu `javaw` (zamiast `java`), aby uruchomić aplikację bez okna konsoli. Zastosowanie `javaw` zapewnia odpowiednie wyłączenie programu po zamknięciu głównego okna. W Java 1.4 można zastosować polecenie `java`, ale we wcześniejszych wersjach konieczne jest użycie `javaw`. Rysunki od 8.2 do 8.4 przedstawiają działanie klas statystyk.

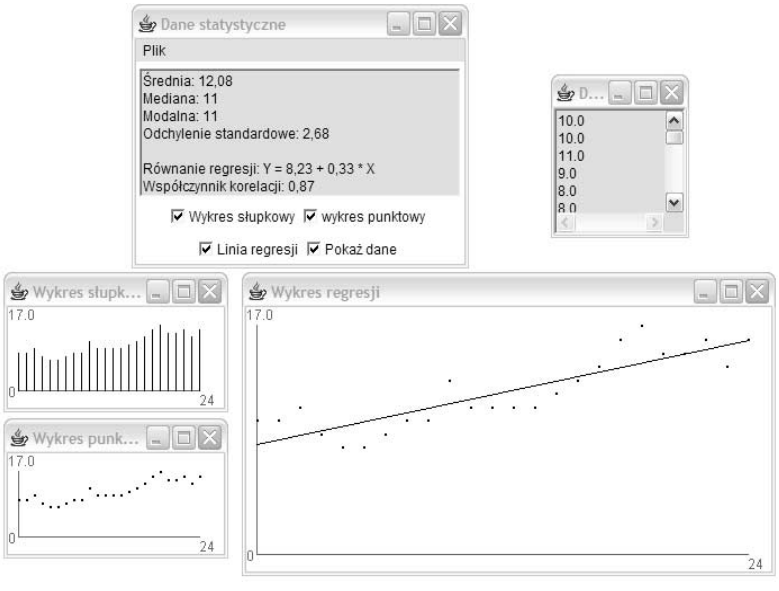
Rysunek 8.2. *Główne okno StatsWin*



Rysunek 8.3. *Okna wykresów*



Jedną z bardziej interesujących funkcji programu jest możliwość dowolnego skalowania wykresów, gdyż znajdują się one w osobnych oknach. Gdy użytkownik zmieni rozmiar okna, wykres automatycznie zmieni skalę. Można także zminimalizować okno wykresu. Zapewnia to usunięcie okna z ekranu, ale nie z systemu.



Rysunek 8.4. *Efekt zmiany rozmiaru jednego z okien*

Prosty aplet ze statystykami

W poprzednim podrozdziale pokazano sposób wykonania samodzielnej aplikacji korzystającej z klas `Stats` i `Graphs`. Nie jest to jednak jedyne zastosowanie tych klas. Równie dobrze klasy te mogą wspomagać aplety lub serwlety. Poniżej przedstawiono przykład prostego apletu. Powoduje on wyświetlanie informacji statystycznych dla dowolnych przekazanych do niej danych.

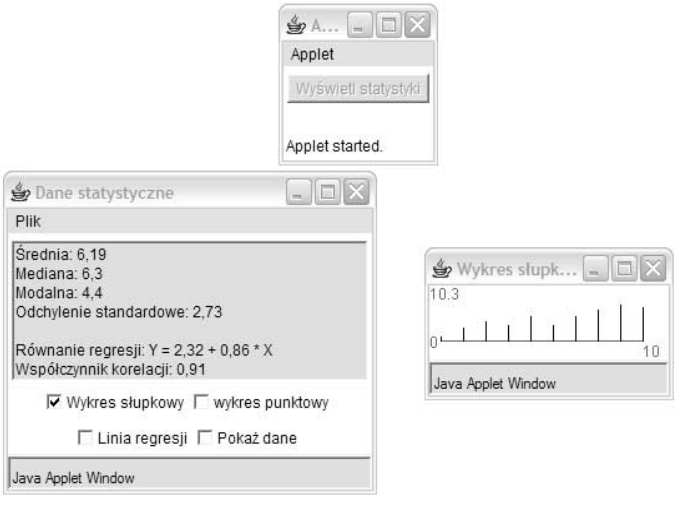
```
// Przykład apletu wykorzystującego Stats i Graphs.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.*;
/*
  <applet code="StatApplet" width=120 height=50>
  <param name=data value="1.2, 3.6, 5.7, 4.4, 7.1, 4.4,
                          6.89, 8.9, 10.3, 9.45">
  </applet>
*/

public class StatApplet extends Applet implements ActionListener {
  StatsWin sw;
  Button show;

  ArrayList al = new ArrayList();

  public void init() {
    StringTokenizer st = new
      StringTokenizer(getParameter("data"), ", \n\r");
```

```
    String v;

    // Odczytanie wartości z HTML.
    while(st.hasMoreTokens()) {
      v = st.nextToken();
      al.add(v);
    }

    show = new Button("Wyświetl statystyki");
    add(show);

    show.addActionListener(this);
  }

  public void actionPerformed(ActionEvent ae) {

    if(sw == null) {
      double nums[] = new double[al.size()];
      try {
        for(int i=0; i<al.size(); i++)
          nums[i] = Double.parseDouble((String)al.get(i));
      } catch(NumberFormatException exc) {
        System.out.println("Błąd odczytu danych.");
        return;
      }

      sw = new StatsWin(nums);
      show.setEnabled(false);

      sw.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent we) {
          sw = null;
          show.setEnabled(true);
        }
      });
    }
  }
}
```

Ważnym jest, iż przekazywanie danych odbywa się za pomocą parametru HTML o nazwie `data`. Tekst zawiera wartości oddzielone przecinkami. Klasa `StatApplet` wykorzystuje obiekt `StringTokenizer` do pobierania poszczególnych wartości jako obiektów `String`. Następnie wartości są zapamiętywane w obiekcie `ArrayList` o nazwie `al`. Klasa `ArrayList` umożliwia przechowywanie tablic o dynamicznie zmieniającej się długości.

Gdy użytkownik kliknie przycisk *Wyświetl statystyki*, wykonywana jest metoda `actionPerformed()`. Obiekt `StatsWin` wymaga tablicy obiektów `double` jako parametr, więc należy dokonać konwersji danych tekstowych `al` do tablicy wartości `double`. W ten sposób powstaje obiekt `StatsWin` ze statystykami.

Przykładowy wynik został uzyskany za pomocą programu *Applet Viewer*. Zaprezentowano go na rysunku 8.5. Tego rodzaju aplet jest doskonałym dodatkiem do wielu stron WWW.



Rysunek 8.5.
Przykład działania apletu StatApplet

Możliwe udoskonalenia

Oto kilka pomysłów na udoskonalenie projektu. Jak już wspomniano, metody rysowania wykresów i metoda `regress()` mogą działać tylko na danych dla wartości Y. Wartościami osi X są punkty w czasie. Można jednak zmienić działanie programu na takie, w którym jako argumenty podaje się dwie tablice — druga z tablic zawiera wartości X.

Interesujące może okazać się umożliwienie zamiany osi, na przykład w czasie rzeczywistym. Użytkownik powinien mieć możliwość ustalenia szerokości słupków lub też kształtu punktów.

Na końcu można poeksperymentować z osadzaniem okna wykresu wewnątrz okna `StatsWin` zamiast tworzenia osobnych okien. Można na przykład określać za pomocą listy opcji, jaki typ wykresu ma być wyświetlany w oknie wykresu.