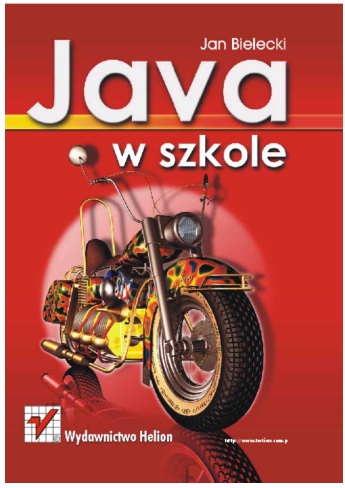
Java w szkole

Autor: Jan Bielecki
Format B5, 400 stron, ISBN: 83-7197-188-5
Zawiera dyskietkę
Data wydania: 10/1999
Cena książki: 39.00 zł
Przesyłka gratis! Odbiorca pokrywa jedynie koszty pobrania (2,70 zł) w przypadku przesyłki za zaliczeniem pocztowym



Wydawnictwo Helion
ul. Chopina 6, 44-100 Gliwice, POLAND
telefon: (32) 230-98-63, 231-22-19
fax: (32) 230-98-63 w.10
mail: helion@helion.com.pl

Niniejsza książka to zarówno podręcznik do nauki programowania aplikacji bazodanowych, jak i kompendium wiedzy o bibliotece VCL w części, która dotyczy baz danych. Studiując tę książkę można samodzielnie zbudować własne komponenty bazodanowe do edycji danych. Czytelnik znajdzie w niej wiadomości między innymi na temat: narzędzi tworzenia baz danych (Database Desktop, SQL Explorer), możliwości klasy TDataSet, użycia klasy TTable, używania komponentu klasy TDatabase, serwera SQL Interbase, który jest dostarczany wraz z pakietem Delphi, komponentów TServerSocket oraz TClientSocket, zaawansowanych technik obsługi zdalnych serwerów SQL, w szczególności bazy Interbase.

Zobacz fragment książki

Spis treści

Jeżeli znasz tę książkę oceń ją

Aktualny cennik książek e-mailem

Książki i "3D" Online

Informacje o nowościach e-mailem

Zamów najnowszy katalog

Zobacz opis dołączonego pliku

Od Autora

Coraz częściej ***Java*** staje się **ulubionym** językiem programowania. Można oczekiwać, że już wkrótce stanie się językiem **powszechnym**, który jako jedyny będzie nauczany w klasach informatycznych Szkół Średnich oraz na pierwszym roku Uczelni Wyższych.

Już obecnie nie istnieje w Polsce ***Uczelnia Akademicka***, w której nie nauczano by *Javy*. Od lat dzieje się tak w ***Polsko-Japońskiej Wyższej Szkole Technik Komputerowych*** w Warszawie oraz w tych uczelniach wyższych, których dydaktycy wzorem kolegów z **USA**, zadali sobie trud przestawienia z ***C***, ***C++***, ***Delphi*** i ***Pascala*** na ***Javę***.

Mimo istnienia już ponad **1500** książek na temat *Javy*, kilku przetłumaczonych na język polski oraz **5** napisanych przeze mnie, wciąż spotykam się z utyskiwaniami, że brakuje łatwego tekstu na temat *Javy*.

Słuchacze moich wykładów, a jest ich każdego roku ponad **300**, nie mają tego problemu, bo wszystko co jest w moich książkach trudne czynię prostym na wykładzie. Biorąc jednak pod uwagę pozostałych, a zwłaszcza tych uczniów **szkół średnich**, którzy pod kierunkiem swoich nauczycieli, chcieliby poznać *Javę* jeszcze **przed** podjęciem studiów wyższych, napisałem książkę, u podstaw której leży następujące wnioskowanie:

> **Jeśli w dotychczasowym nauczaniu programowania najpierw wykłada się C i C++, a dopiero po nich Javę, to dlaczego nie ograniczyć się do Javy, ale najpierw podać ją w postaci podobnej do C, a następnie w postaci podobnej do C++. W ten sposób można całkowicie zrezygnować z C i C++, a Javę wyłożyć jako jedyny język programowania.**

Aby zadanie to zrealizować, opracowałem ***Bibliotekę***, która osobom praktycznie **bez przygotowania**, umożliwia natychmiastowe pisanie programów w *Javie*. Ten liczący ok. **4000** wierszy produkt, który w postaci skompresowanego pliku ****.jar*** zajmuje ok. **40** KB, może być użyty w dowolnym środowisku uruchomieniowym **3**-generacji.

Na podstawie moich doświadczeń **polecam** jednak tylko dwa takie środowiska: bezpłatny tandem ***Kawa 3.22 – Java 2 Paltform*** oraz kosztujący w wersji edukacyjnej ok. **$100** kompilator ***JBuilder 3.0***.

Życząc Czytelnikom pożytecznej i łatwej lektury, z przyjemnością informuję, że wszystkie omawiane w książce programy źródłowe, wraz z biblioteką ***View.jar***, można znaleźć na dołączonej dyskietce oraz w serwerze ***Wydawnictwa Helion***.

prof. Jan Bielecki

Programy

Programowanie jest zapisywaniem ***czynności*** przewidzianych do wykonania przez komputer. Zapisem czynności jest ***program źródłowy***, na przykład napisany w *Javie*. Program źródłowy poddaje się ***kompilacji***, a następnie ***łączy*** z podprogramami dostarczanymi wraz z kompilatorem. Powstaje wówczas ***program wykonalny***. Jego wykonanie powierza się *Maszynie Wirtualnej*. Jest to specjalny program, który interpretując instrukcje zawarte w programie wykonalnym, realizuje czynności, jakie wyrażono w programie źródłowym.

W procesie tworzenia i interpretowania programu specjalną rolę odgrywają **2** pliki: ***Master.java*** zawierający ***klasę apletową*** oraz ***Project.html*** zawierający ***opis apletu***. Przeglądarka zapoznaje się z opisem apletu, takim jak podany w tabeli ***Plik Project.html*** i znajduje w nim:

1. Nazwę klasy apletowej (**code**).
2. Szerokość (**width**) i wysokość (**height**) prostokątnej ramki, udostępnionej programowi do komunikowania się z użytkownikiem.

Tabela
Plik Project.html

```
<applet code=Master.class
        width=400 height=420>
</applet>
```

albo

```
<applet code=janb.java3.Master.class
        width=400 height=420>
</applet>
```

Nazwy **Master** i **Project** można zastąpić innymi. Należy jedynie pamiętać, że jeśli nazwą **publicznej** klasy apletowej jest ***Name***, to definicja klasy musi się znajdować w pliku ***Name*.java**.

Struktura programu

Program zapisany jako aplet **Master**, znajduje się w pliku **Master.java**. Opis apletu znajduje się w pliku **Project.html**. Parametrami opisu są: **Master.class**, **400** i **420**. W dalszych przykładach będzie podawana tylko zawartość pliku **Master.java**.

Aplet ma postać przedstawioną w tabeli ***Struktura apletu***. Składa się on z ***poleceń importu*** oraz z definicji klasy **Master**. Klasa ta stanowi nadbudowę nad ***klasą widoku*** (**View**). Napisy od pary znaków **//** (*ukośnik, ukośnik*) do końca wiersza włącznie, są ***komentarzami*** i jako takie nie mają żadnego wpływu na przebieg wykonania programu.

Tabela
Struktura apletu

```
package janb.java3;       // zalecane, ale nieobowiązkowe
import janb.view.*;

// tu jest miejsce na dodatkowe polecenia importu

public
class Master
    extends View {

//*******************************************************//
      // tu należy wstawić własne instrukcje programu
      // pod żadnym pozorem nie wstawiając funkcji
      //          public void init()
//*******************************************************//

}
```

dla dociekliwych

Nazwy klas można uprościć do ***identyfikatorów*** (np. **Applet**, **Graphics** albo **Color**). Aby to umożliwić, należy użyć ***poleceń importu***, na przykład

```
import java.awt.Graphics;
```

albo

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

Dzięki poleceniu

```
import java.applet.Graphics;
```

nazwę klasy **java.awt.Graphics** można uprościć do **Graphics**, a dzięki poleceniu

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

odwołania do klas pakietu **java.awt**, których nazwy zaczynają się od **java.awt**, można uprościć do identyfikatora kończącego taką nazwę (np. **java.awt.Graphics** do **Graphics**).

A zatem drugie z rozpatrzonych poleceń importu jest **silniejsze**, gdyż pierwsze z niego **wynika**.

Widok i klatka

W chwili ***odpalenia*** apletu, na ekranie pojawia się prostokątna ***ramka*** podzielona na ***widok*** i ***klatkę***. Widok jest obszarem, w którym aplet przedstawia swoje ***wyniki***, a klatka jest obszarem do wprowadzania ***danych***. W każdej chwili na jeden z tych obszarów jest nastawiony ***celownik***.

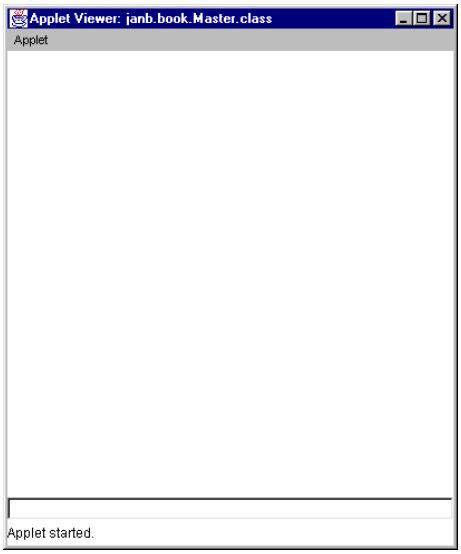
Tuż po rozpoczęciu wykonywania apletu celownik jest nastawiony na **klatkę**. Objawia się to wyświetleniem w niej ***karetki***. Naciśnięcie klawisza *Enter* gdy celownik jest nastawiony na **pustą** klatkę albo kliknięcie w obszarze widoku przenosi celownik na **widok**. Innym sposobem przeniesienia celownika na widok jest wywołanie funkcji **requestFocus**.

```
void requestFocus()
```

Przenosi celownik na widok.

Na ekranie ***Widok i klatka*** pokazano początkowy wygląd apletu, z celownikiem nastawionym na klatkę.

Ekran
Widok i klatka



Klasa widoku jest dostarczona w części ***Biblioteka View***. Zapoznanie się z jej kodem jest pożytecznym ćwiczeniem, którego wykonanie zaleca się jednak dopiero po przestudiowaniu **całej** książki.

Najprostszy aplet

Między komentarzami `/* ... */` należy wstawić program apletu. W najprostszym przypadku jest to dowolny zestaw ***deklaracji pól*** i ***definicji funkcji***, takich jak **initView**, **runView** i **drawView**, wywoływanych z klasy **View** albo wnętrza apletu.

W **initView** umieszcza się czynności ***inicjujące***, a w **drawView** czynności ***regenerujące*** widok. Dodatkowo można używać funkcji **startView**, **stopView** i **destroyView**.

```
void initView()
```

Wywoływana **jednokrotnie**, bezpośrednio po wyświetleniu apletu.

```
void drawView()
```

Wywoływana **wielokrotnie**, po każdym **zewnętrznym** zniszczeniu widoku, spowodowanym na przykład przejściowym ***zasłonięciem*** widoku przez **obce okno** albo ***ikonizacją*** okna przeglądarki.

```
void stopView()
```

Wywoływana wówczas, gdy przeglądarka przestaje pokazywać aplet.

```
void startView()
```

Wywoływana przed każdym pokazaniem apletu, w szczególności tuż po wywołaniu funkcji **initView**.

```
void destroyView()
```

Wywoływana **jednokrotnie**, tuż przed zniszczeniem apletu.

```
void runView()
```

Wywoływana **jednokrotnie**, bezpośrednio po zakończeniu wykonywania funkcji **initView**. Nie zaleca się jej stosować jednocześnie z **drawView**, gdyż jest z nią wykonywana ***współbieżnie***, a to wymaga umiejętnego posługiwania się ***synchronizowaniem wątków.***

Pomocna w wyprowadzaniu rezultatu jest funkcja **showResult**. Jej argumentem jest dowolne ***wyrażenie łańcuchowe***.

```
void showResult(String result)
void showResult(int value)
```

Wyprowadza wartość ***result***, pochyloną i pogrubioną czcionką **30**-punktową, w połowie wysokości widoku.

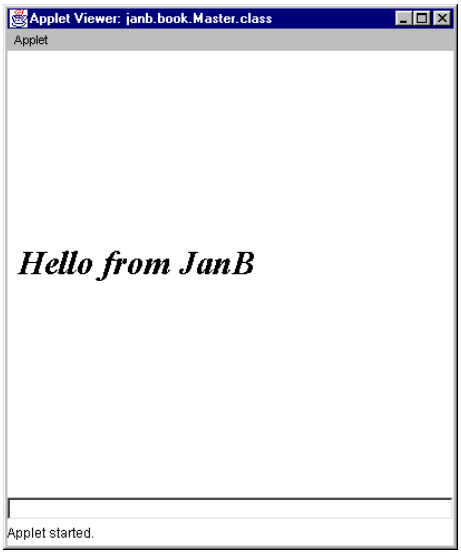
```
void showResult(String result, int size)
```

Wyprowadza wartość ***result***, pogrubioną czcionką o rozmiarze ***size***, w połowie wysokości widoku.

Czcionki poniżej **10** pt są słabo widoczne.

Na ekranie ***Aby tradycji stało się zadość*** pokazano aplet wyświetlający napis ***Hello from JanB***.

Ekran
Aby tradycji
stało się zadość




```
import janb.view.*;

public
class Master
    extends View {

// ************************************************** //

    public void drawView()
    {
        showResult("Hello from JanB");
    }

// ************************************************** //
}
```

Dane wejściowe

Aplet zazwyczaj oczekuje ***danych***, od których uzależnia wykonanie swoich czynności. Dane są wpisywane do ***klatki*** i wysyłane do apletu po naciśnięciu klawisza *Enter*. Powoduje to zainicjowanie parametru funkcji **dataEntered** daną wpisaną do klatki.

Następujący aplet, pokazany na ekranie ***Pozdrowienie z apletu***, napisano w taki sposób, że jeśli do klatki wprowadzi się imię użytkownika (np. **Jan**), a następnie naciśnie klawisz *Enter*, to na pulpicie wyświetli się pozdrowienie (tu: **Hello Jan**).

Ekran
Pozdrowienie z apletu


```
import janb.view.*;

public
class Master
    extends View {

// ************************************************** //

    public void dataEntered(String data)
    {
            // wyprowadzenie pozdrowienia
        showResult("Hello " + data);
    }

// ************************************************** //
}
```

Komunikaty

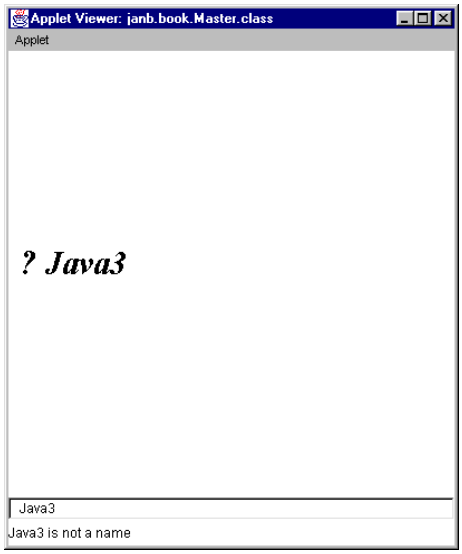
Przeglądarka dodatkowo udostępia pole, w którym można umieszczać krótkie ***komunikaty***. W przeglądarce środowiska *Kawa*, pole to występuje bezpośrednio pod klatką na dane.

```
void showMsg(String string)
```

Wyprowadza do pola komunikatów napis ***string***.

Następujący aplet, pokazany na ekranie ***Pole komunikatów***, napisano w taki sposób, że wyświetla pozdrowienie tylko wówczas, gdy w klatce podano napis **literowy**. W przeciwnym razie informuje o niepoprawnej danej.

Ekran
Pole komunikatów




```
import janb.view.*;

public
class Master
    extends View {

// ************************************************** //

    public void dataEntered(String data)
    {
        if(isAlpha(data))     // czy dana składa się z liter
            showResult("Hello " + data);
        else {
            showResult("? " + data);
            showMsg(data + " is not a name");
        }
    }

// ************************************************** //
}
```

Dostarczanie danych

Do rozpoznawania danych dostarczonych w klatce służą funkcje **dataEntered**. Jeśli aplet zawiera więcej niż jedną taką funkcję, to wybór funkcji odbywa się na podstawie rodzaju argumentu: ***znaku*** (np. **J**), ***łańcucha*** (np. **JB**) albo ***zestawu łańcuchów*** (np. **Jan Bielecki**).

Jeśli po naciśnięcie klawisza *Enter* aplet nie podejmie przetworzenia dostarczonych danych, co może wynikać z użycia funkcji **dataEntered** przystosowanej do rozpoznawania pojedynczych łańcuchów i dostar-

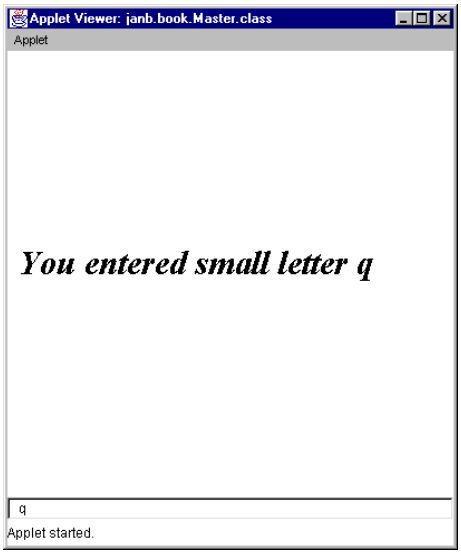
czenia więcej niż jednego łańcucha (np. **Johnny Walker**), to w celu poinformowania, że dane zostały **zignorowane**, rozlegnie się **sygnał dźwiękowy**.

```
void dataEntered(char data)
```

Wywoływana po każdym dostarczeniu w klatce pojedynczego **znaku**. Udostępnia w **data** kod znaku.

Następujący aplet, pokazany na ekranie ***Dane znakowe***, rozpoznaje czy wprowadzona dana jest małą literą alfabetu angielskiego.

Ekran
Dane znakowe




```
import janb.view.*;

public
class Master
    extends View {

// ************************************************** //

    public void dataEntered(char data)
    {
        if(data >= 'a' && data <= 'z')
            showResult("You entered small letter " + data);
        else
            showResult("You entered " + data);
    }

// ************************************************** //
}
```

```
void dataEntered(String data)
```

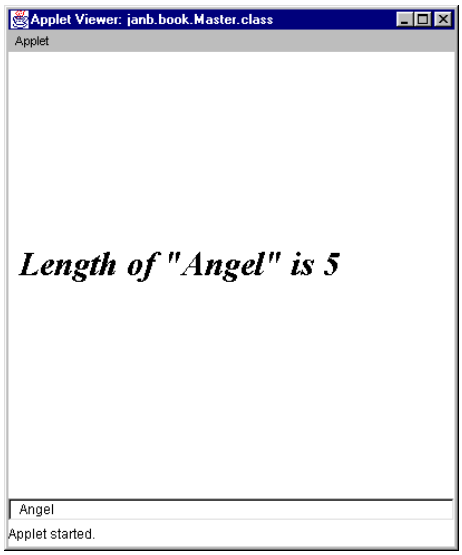
Wywoływana po każdym dostarczeniu w klatce wejściowej pojedynczego **łańcucha** znaków. Udostępnia w ***data*** łańcuch znaków.

```
int length(String string)
```

Dostarcza liczbę znaków łańcucha ***string***.

Następujący aplet, pokazany na ekranie ***Dane łańcuchowe***, wprowadza łańcuch i podaje liczbę jego znaków.

Ekran
Dane łańcuchowe




```
import janb.view.*;

public
class Master
    extends View {

// ************************************************** //

    public void dataEntered(String data)
    {
        showResult(
            "Length of \"" + data + "\" is " + length(data)
        );
    }

// ************************************************** //
}
```

`void` **`dataEntered`**`(String[] data)`

Wywoływana po każdym dostarczeniu w klatce wejściowej więcej niż jednego łańcucha znaków. Udostępnia w ***data*** odnośnik do tablicy łańcuchów znaków.

```
int length(String[] vector)
```

Dostarcza liczbę łańcuchów zawartych w tablicy ***vector***.

Następujący aplet, pokazany na ekranie ***Wiele łańcuchów***, wprowadza ciąg łańcuchów i podaje **ostatni**.

Ekran
Wiele łańcuchów

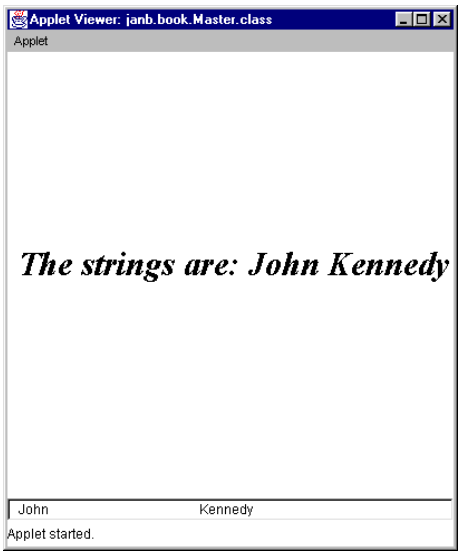

```
import janb.view.*;

public
class Master
    extends View {

// ************************************************** //

    public void dataEntered(String[] data)
    {
        int count = length(data);
        showResult("The last string is " + data[count-1]);
    }

// ************************************************** //
}
```

```
String getString(String[] strings)
```

Dostarcza łańcuch utworzony z połączenia łańcuchów zawartych w tablicy ***strings***, po uzupełnieniu każdego pojedynczą **spacją**. Jest przydatna wówczas, gdy łańcuchy wprowadzone do klatki chce się traktować jak **jeden** łańcuch.

Następujący aplet, pokazany na ekranie ***Łączenie łańcuchów***, podaje jakie wprowadzono łańcuchy.

Ekran
Łączenie łańcuchów




```
import janb.view.*;

public
class Master
    extends View {

// ************************************************** //

    public void dataEntered(String[] data)
    {
        showResult("The strings are: " + getString(data));
    }

// ************************************************** //
}
```

dla dociekliwych

Jeśli aplet zawiera więcej niż jedną funkcję **dataEntered**, to jest wywoływana tylko ta, która jest przystosowana do obsługi wprowadzonej danej. Jeśli takiej nie ma, to jest wywoływana funkcja **ogólniejsza**. W szczególności, jeśli aplet zawiera tylko funkcję z parametrem typu **char** i funkcję z parametrem typu **String**, to

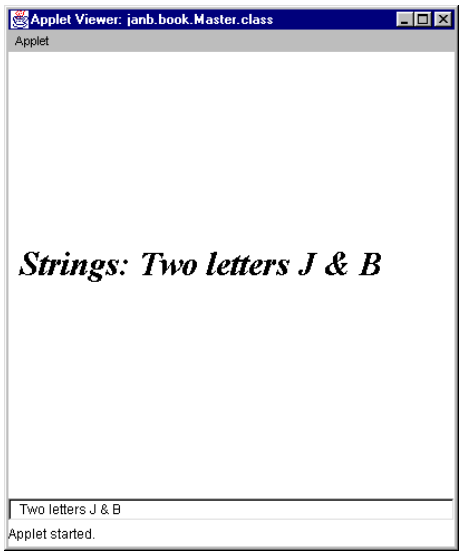
1. Po dostarczeniu jednego znaku zostanie wywołana funkcja z parametrem typu **char**.
2. Po dostarczeniu jednego łańcucha, który składa się z więcej niż z jednego znaku, zostanie wywołana funkcja z parametrem typu **String**.
3. Po dostarczeniu więcej niż jednego łańcucha nie zostanie wywołana **żadna** z funkcji **dataEntered**, a to spowoduje wygenerowanie **sygnału dźwiękowego**.

Funkcja przystosowana do przetwarzania danych ogólniejszych nadaje się do obsługiwania danych mniej ogólnych. W szczególności funkcja przystosowana do przetwarzania łańcuchów, może być użyta do prze-

tworzenia pojedynczych znaków, a funkcja do przetwarzania zestawów łańcuchów może być użyta do przetworzenia znaków, łańcuchów i zestawów łańcuchów.

Następujący aplet, pokazany na ekranie ***Rozpoznawanie danych***, informuje o rodzaju dostarczonej danej.

Ekran
Rozpoznawanie danych




```
import janb.view.*;

public
class Master
    extends View {

// ************************************************** //

    public void dataEntered(String[] data)
    {
        if(length(data) == 1)
            dataEntered(data[0]);
        showResult("Strings: " + getString(data));
    }

    public void dataEntered(String data)
    {
        if(length(data) == 1)
            showResult("Char: " + data);
        else
            showResult("String: " + data);
    }

// ************************************************** //
}
```

Operacje na klatce

Dane można ***umieszczać*** w klatce albo ***pobierać*** wprost z klatki. Służą do tego funkcje **setData** i **getData**. Funkcji **setData** należy używać z **ostrożnością**, ponieważ kolidują z edycją klatki.

```
void setData(String data)
```

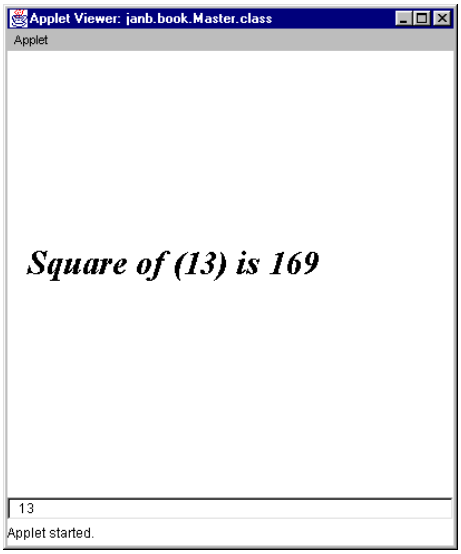
Umieszcza w klatce napis określony przez ***data***.

```
String getData()
```

Dostarcza napis znajdujący się w klatce.

Następujący aplet, pokazany na ekranie ***Operacje na klatce***, rozpoznaje czy w klatce znajduje się liczba całkowita, a jeśli tak, to wyświetla jej **kwadrat**.

Ekran
Operacje na klatce




```
import janb.view.*;

public
class Master
    extends View {

    public void runView()
    {
        boolean viewCleared = true;
        int oldData = MinInt;
        while(true) {
            String data = getData();
            if(isIntegral(data)) {
                int val = getInt(data);
```

```
                if(val != oldData) {
                    oldData = val;
                    clearView();
                    showResult(
                        " Square of (" + val + ") is "
                         + val * val
                    );
                    viewCleared = false;
                }
            } else if(!viewCleared)
                clearView();
        }
    }
}
```

Odtwarzanie widoku

Aplet powinien być napisany w taki sposób, aby po każdym wywołaniu funkcji **drawView**, następowało pełne ***odtworzenie widoku***.

Funkcja **drawView** jest zazwyczaj wywoływana przez *System*. Istnieje jednak możliwość wywołania jej z wnętrza programu. Służy do tego funkcja **redraw**.

```
void redraw()
```

Wyczyszcza widok kolorem tła i wywołuje funkcję **drawView**.

```
void setBackground(Color color)
```

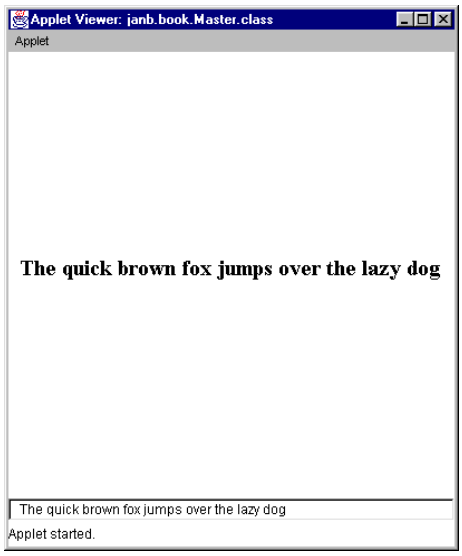
Ustawia kolor tła apletu na ***color*** (np. **setBackground(Red)**).

Gdyby zmienna **result** nie została zainicjowana, to podczas pierwszego wywołania funkcji **drawView** zostałby wyprowadzony komunikat ***String pointer is NULL***. Komunikat ten jest swoistym kuriozum, ponieważ w *Javie* nie ma wskaźników (powinien brzmieć ***String reference is NULL***).

Następujący aplet, pokazany na ekranie ***Odtwarzanie widoku***, wyświetla napis wprowadzony do klatki, ale dzięki użyciu funkcji **redraw** i **drawView** zapewnia, że napis ten pozostaje na widoku nawet jeśli nastąpi przejściowe zasłonięcie widoku przez obce okno.

```
import janb.view.*;

public
class Master
    extends View {

    private String result = "John von Neuman";
```

Ekran
Odtwarzanie widoku




```
public void dataEntered(String[] data)
    {
        result = getString(data);
        redraw();
    }

    public void drawView()
    {
        showResult(result, 20);
    }
}
```

Środowisko

Jeśli program ***od razu działa***, to albo jest ***banalnie prosty*** albo zawiera ***błędy logiczne***. Błędy te należy **rozpoznać** i **usunąć**. Czynności te wykonuje ***programista***.

Wbrew powszechnemu mniemaniu, programistą nie jest ten, kto umie napisać program, ale ten, kto umie go ***uruchomić***, ***przetestować*** i ***usunąć*** zawarte w nim **błędy**. Dlatego chociaż wiele osób umie programo-

wać, tylko **nieliczni** są programistami.

Programista ***nowoczesny***, a zwłaszcza programista ***zawodowy*** jest przyzwyczajony do komfortu, który oferują zintegrowane ***środowiska uruchomieniowe***. Pod wspólnym „dachem" umożliwiają one wykonywanie ***edycji*** programów źródłowych oraz ***kompilowanie***, ***łączenie*** i ***budowanie*** programów.

Wśród wielu środowisk uruchomieniowych ułatwiających programowanie w *Javie* znajdują się takie, które kosztują po kilka tysięcy dolarów, jak i takie, które można **za darmo** ściągnąć z *Internetu*. Wśród tych ostatnich znajduje się środowisko ***Kawa***, którego przez **3** miesiące można używać bez potrzeby zakupu.

Zainstalowanie biblioteki

Biblioteka implementująca klasę **View** wraz z jej dodatkami zajmuje ok. **40** KB i znajduje się w pliku **View.jar**, na dyskietce dołączonej do książki. Plik ten można umieścić w katalogu ***rozszerzeń*** pakietu *JDK* albo po ewentualnym rozpakowaniu do katalogu **janb\view**, skopiować

4. Dla środowiska ***Kawa*** do katalogu ***projektowego***.
5. Dla środowiska ***JBuilder*** do katalogu ***wyjściowego***.

Katalogiem rozszerzeń dla *JDK* zainstalowanego w **d:\JDK12Run** jest **d:\JDK12Run\jre\lib\ext**, a dla kompilatora ***JBuilder 3.0*** zainstalowanego w **d:\JBuilder3** jest **d:\JBuilder3\java\jre\lib\ext**.

Grafika

Na widoku można wykreślać ***napisy***, ***odcinki***, ***figury*** i ***obszary***. W celu wykreślenia obiektu graficznego należy podać ***współrzędne*** widoku i wybrać ***narzędzia*** do wykreślania. Wykreślanie może się odbywać w ***kolorach***, a napisy mogą być wykreślane ***czcionkami*** o dowolnie obranym ***kroju***, ***stylu*** i ***rozmiarze***.

Linie są wykreślane ***piórami***, a napisy obszary są zamalowywane ***pędzlami***. Pędzle mogą malować obszary ***farbami***, ***gradientami*** i ***teksturami***.

Współrzędne

Współrzędne są liczone względem lewego-górnego narożnika widoku.

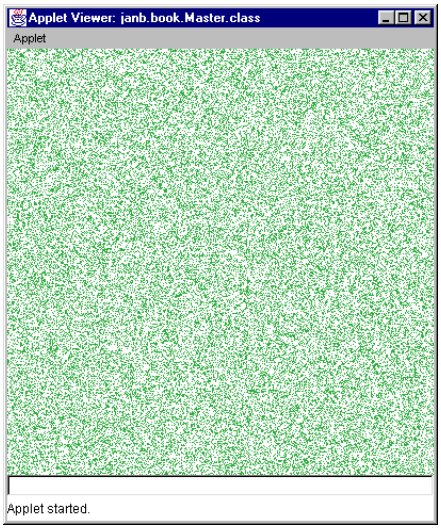
W celu wykreślenia

1. **Napisu** podaje się współrzędne lewego końca domyślnego odcinka bazowego, na którym napis ***spoczywa*** albo współrzędne lewego-górnego wierzchołka domyślnego prostokąta, w który ten napis ***wpisano***.
2. **Odcinka**, podaje się współrzędne jego końców.
3. **Figury**, **Obszaru** i **Obiektu** podaje się współrzędne lewego-górnego wierzchołka domyślnego prostokąta, w który wpisano tę figurę, obszar albo obiekt.

Dla okręgów podaje się **średnicę**, a dla kwadratów **bok**.

Następujący aplet, pokazany na ekranie ***Wykreślanie pikseli***, wykreśla piksele w przypadkowych punktach widoku.

Ekran
Wykreślanie pikseli




```
import janb.view.*;

public
class Master
    extends View {

    public void drawView()
    {
```

```
            int w = getWidth(),
                h = getHeight();
            while(true) {
                int x = getRandom(0, w),
                    y = getRandom(0, h);
                if(x == h/2 && y == h/2){
                    clearView();
                    setColor(getColor());
                }
                drawPixel(x, y);
            }
        }
    }
```

Wykresy

Najprostszymi figurami graficznymi są: ***pixel***, ***napis***, ***okrąg***, ***kwadrat*** i ***odcinek***. Okręgi i kwadraty można wykreślać jako

Cienkie	**(Thin)**
Liniowe	**(Line)**
Pierścieniowe	**(Ring)**
Wypełnione	**(Full)**

```
void drawPixel(int x, int y)
```

Wykreśla piksel w punkcie o współrzędnych (***x***, ***y***).

```
void drawString(String string, int x, int y)
```

Wykreśla napis ***string***, spoczywający na odcinku bazowym o **lewym** końcu w (***x***, ***y***).

```
void drawString(int x, int y, String string)
void drawString(int x, int y, String string, Font font)
```

Wykreśla napis ***string***, czcionką **bieżącą** albo czcionką ***font***, w domyślnym prostokącie o narożniku w (***x***, ***y***).

```
void drawLine(int xA, int yA, int xZ, int yZ)
```

Wykreśla odcinek łączący punkty (***xA***, ***yA***) i (***xZ***, ***yZ***). Punkt (***xZ***, ***yZ***) definiuje jako ***końcowy***.

```
void moveTo(int x, int y)
```

Punkt (***x***, ***y***) definiuje jako **końcowy**.

```
void lineTo(int x, int y)
```

Wykreśla odcinek łączący punkt końcowy z punktem (***x***, ***y***). Punkt (***x***, ***y***) definiuje jako **końcowy**.

```
void drawCircle(int x, int y, int d)
```

Wykreśla okrąg o średnicy ***d***, wpisany w domyślny prostokąt o wierzchołku w (***x***, ***y***).

```
void drawSquare(int x, int y, int s)
```

Wykreśla kwadrat o boku ***s*** i wierzchołku w (***x***, ***y***).

```
void drawEllipse(int x, int y, int w, int h)
```

Wykreśla elipsę o średnicach ***w*** *x* ***h***, wpisaną w domyślny prostokąt o wierzchołku w (***x***, ***y***).

```
void drawRectangle(int x, int y, int w, int h
```

Wykreśla prostokąt o rozmiarach ***w*** *x* ***h*** i wierzchołku w (***x***, ***y***).

```
void drawDiamond(int x, int y, int w, int h
```

Wykreśla romb o rozmiarach ***w*** *x* ***h***, wpisany w domyślny prostokąt o wierzchołku w (***x***, ***y***).

```
void setFill(int fill)
```

Ustawia na ***fill*** (**Thin, Line, Ring, Full**) sposób wykreślania **okręgów**, **kwadratów**, **elips** i **prostokątów**.

```
void setFont(Font font)
```

Ustawia na ***font***, czcionkę do wykreślania napisów.

```
void setStroke(int width)
void setStroke(int width, double ratio)
void setStroke(int width, double ratio, int cap, int join)
```

Ustawia na ***width*** szerokość pióra używanego do wykreślania linii. Pędzlem wypełnia tylko części podlinii określone przez ***ratio***. Linie kończy zgodnie z ***cap*** (**CapMiter**, **CapRound**, **CapBevel**), a łączy je zgodnie z ***join*** (**JoinButt**, **JoinRound**, **JoinSquare**).

```
void setPaint(Paint paint)
```

Ustawia na ***paint*** rodzaj pędzla używanego do wypełniania obszarów.

```
void setColor(Color color)
```

Ustawia na ***color*** rodzaj farby używanej do wykreślania linii i wypełniania obszarów.

Ponieważ malowanie **farbą** wyklucza się z malowaniem **teksturą** i **gradientem**, więc w każdej chwili obowiązuje tylko ostatnie z wywołań funkcji **setColor** i **setPaint**.

Następujący aplet, pokazany na ekranie ***Dobieranie piór i pędzli***, ilustruje zastosowanie funkcji do wybierania piór i pędzli.

Ekran
Dobieranie piór i pędzli


```
import janb.view.*;
import java.awt.*;

public
class Master
    extends View {

    private int r= 100,
                w, h;
    private Paint paint;
    private Font font;

    public void initView()
    {
        w = getWidth();
        h = getHeight();

        setFill(Ring);
        setStroke(50, 0.75);
        paint = getGradient(Blue, Green);
        setPaint(paint);
        font = getFont("Serif", Bold, 150);
        setFont(font);
    }

    public void dataEntered(String data)
    {
        showResult("Hello");
    }

    public void drawView()
    {
         drawCircle(w/2-r+50, h/2-r+50, 2*r);
         drawString(10, 10, "Kaja");
         setStroke(30, 0.5, JoinRound, CapRound);
         drawLine(50, 200, 50, 250);
         drawLine(50, 250, 100, 300);
    }
}
```

Czcionki

Czcionkę opisuje ***krój***, ***styl*** i ***rozmiar***. W każdym *Systemie* można się posługiwać się czcionkami o nazwach: **Serif**, **Sansserif**, **Monospaced** i **Dialog**, o stylach: **Bold**, **Italic**, **BoldItalic**, **Plain** oraz o rozmiarach wyrażonych w ***punktach*** (**1 pt = 1/72 cala**).

Do utworzenia czcionki używa się funkcji **getFont**, a do jej wybrania funkcji **setFont**.

```
Font getFont(String face, int style, int size)
```

Dostarcza czcionkę o kroju ***face***, stylu ***style*** i rozmiarze ***size***.

```
void setFont(Font font)
```

Ustawia czcionkę używaną do wykreślania napisów na ***font***.

```
Font getCurrentFont()
```

Dostarcza aktualną czcionkę

Metryka

Z napisem który ma być wykreślony wybraną czcionką jest związana jego ***metryka***. Do utworzenia metryki używa się funkcji **getMetrics**, a do wykonania operacji na metryce, funkcji **getWidth**, **getHeight**, **getAscent** i **getDescent**.

```
Metrics getMetrics(String string)
```

Dostarcza metrykę napisu, który w danej chwili zostałby wykreślony za pomocą funkcji **drawString**.

```
int getWidth(Metrics metrics)
```

Dostarcza **szerokość** domyślnego prostokąta, w którym będzie wykreślony napis o metryce ***metrics***.

```
int getHeight(Metrics metrics)
```

Dostarcza **wysokość** domyślnego prostokąta, w którym będzie wykreślony napis o metryce ***metrics***.

```
int getAscent(Metrics metrics)
```

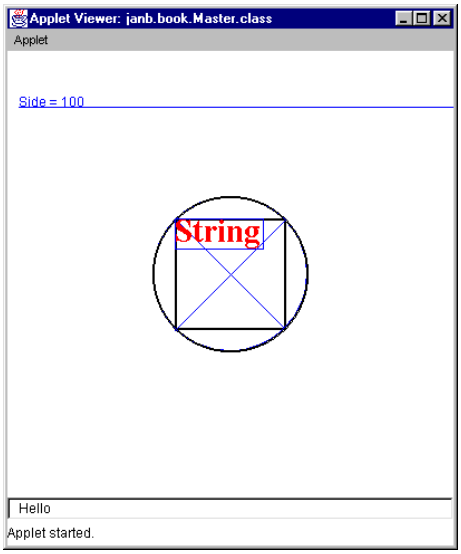
Dostarcza **uniesienie** względem **odcinka bazowego**, górnej podstawy domyślnego prostokąta, w którym będzie wykreślony napis o metryce ***metrics***

```
int getDescent(Metrics metrics)
```

Dostarcza **obniżenie** względem **odcinka bazowego**, dolnej podstawy domyślnego prostokąta, w którym będzie wykreślony napis o metryce ***metrics***.

Następujący aplet, pokazany na ekranie ***Wykreślanie figur***, wykreśla na środku widoku **kwadrat** o podanym boku, opisuje na nim **okrąg** i wykreśla jego **przekątne**. W górnej części widoku wykreśla **napis** spoczywający na odcinku bazowym oraz napis ***String*** otoczony jego **obwiednią**.

Ekran
Wykreślanie figur




```
import janb.view.*;
import java.awt.*;

public
class Master
    extends View {
private String string = "String";
    private int side = 100,
                w, h;
    private double sqrt2 = sqrt(2);

    public void dataEntered(String data)
    {
        if(!isIntegral(data)) {
            beep();
            return;
        }

        side = getInt(data);
        redraw();
    }

    public void initView()
    {
        w = getWidth();
        h = getHeight();
    }

    public void drawView()
    {
        int x = (w - side) >> 1,
            y = (h - side) >> 1;

        drawSquare(x, y, side);

        drawLine(x, y, x+side, y+side);
```

```
        drawLine(x+side, y, x, y+side);

        int dd = (int)(side * (sqrt2-1)/2);
        drawCircle(x-dd, y-dd, side + 2*dd);

        drawString("Side = " + side, 10, 50);
        drawLine(10, 50, w, 50);

        setStroke(30);
        setColor(Red);
        Font font = getFont("Serif", Bold, 30);
        drawString(x, y, string, font);

        Metrics metrics = getMetrics(string, font);
        int a = getAscent(metrics),
            d = getDescent(metrics),
            w = getWidth(metrics),
            h = getHeight(metrics);
        setStroke(1);
        setColor(Blue);
        drawRectangle(x, y, w, h);
    }
}
```

Obrazy

Jeśli zamierza się wykreślić obraz, uprzednio przygotowany w pliku z rozszerzeniem *GIF*, to należy

1. W celu załadowania obrazu do pamięci użyć funkcji **getImage**.
2. W celu wykreślenia obrazu użyć funkcji **drawImage**.

Aby określić rozmiary załadowanego obrazu można użyć funkcji **getWidth** i **getHeight**.

```
Image getImage(String file)
```

Ładuje obraz z pliku ***file*** i dostarcza odnośnik do reprezentującego go obiektu.

```
void drawImage(Image image, int x, int y, int w, int h)
void drawImage(Image image, int x, int y)
```

Obraz identyfikowany przez **image** wykreśla w taki sposób, że współrzędne jego lewego-górnego narożnika znajdą się w punkcie (**x**, **y**), a jeśli podano parametry **w** i **h**, to **przeskalowuje** obraz i wyświetla go w prostokącie o rozmiarach **w** x **h**.

```
int getWidth(Image image)
```

Dostarcza **szerokość** obrazu ***image***.

```
int getHeight(Image image)
```

Dostarcza **wysokość** obrazu ***image***.

Następujący aplet, pokazany na ekranie ***Wykreślenie obrazu***, wykreśla na środku widoku, obraz o nazwie podanej w klatce. W polu komunikatów podaje informację o rozmiarach obrazu.

Ekran
Wykreślenie obrazu


```
import janb.view.*;
import java.awt.*;

public
class Master
    extends View {

 private String defData = "Roger.gif";

    public void initView()
    {
        setData(defData);
    }

    public void dataEntered(String data)
    {
            // pobranie obrazu
        Image img = getImage(data);

            // sprawdzenie, czy obraz istnieje
        if(data == null) {
            showMsg("Image does not exist");
            return;
        }

        int imgW  = getWidth(img),   // szerokość obrazu
            imgH  = getHeight(img),  // wysokość obrazu
            viewW = getWidth(),      // szerokość widoku
            viewH = getHeight();     // wysokość widoku

            // wyprowadzenie komunikatu
        showMsg("Width = " + imgW + ", Height = " + imgH);

        imgW = 200;
        imgH = 200;
            // wyznaczenie współrzędnych
            // lewego-górnego
```

```
            // narożnika obrazu
        int x = (viewW - imgW) / 2,
            y = (viewH - imgH) / 2;

            // wykreślenie obrazu
        drawImage(img, x, y, imgW, imgH);
    }

    public void dataEntered(String[] data)
    {
        clearView();
            // wyprowadzenie informacji,
            // że podano więcej niż 1 łańcuch
        showMsg("Error: filename please!");
    }
}
```

Obiekty

Figurze graficznej można nadać cechy ***obiektu***. Obiekt figurowy przechowuje informacje o właściwościach figury, umożliwia ***kolekcjonowanie*** figur oraz ***odtwarzanie*** ich bezpośrednio z kolekcji.

W chwili utworzenia obiektu figurowego, wstawia się do niego bieżąca **czcionkę**, **pióro**, **pędzel** i sposób **wypełniania**. Te wstępne ustawienia można zmienić za pomocą funkcji **set*XXX***.

```
Figure getPixel(int x, int y)
```

Dostarcza odnośnik do obiektu **piksela** o współrzędnych (***x***, ***y***).

```
Figure getString(int x, int y, String string)
```

Dostarcza odnośnik do **napisu**, zdefiniowanego przez ***string***, wpisanego w domyślny prostokąt o wierzchołku w (***x***, ***y***).

```
Figure getImage(int x, int y, Image image)
Figure getImage(int x, int y, int w, int h, Image image)
```

Dostarcza odnośnik do **obrazu** zdefiniowanego przez ***image***, o współrzędnych (***x***, ***y***) i rozmiarach ***w*** *x* ***h*** (do przeskalowania).

```
Figure getLine(int xA, int yA, int xZ, int yZ)
```

Dostarcza odnośnik do **odcinka** łączącego punkty (***xA***, ***yA***) i (***xZ***, ***yZ***).

```
Figure getCircle(int x, int y, int d)
Figure getEllipse(int x, int y, int w, int h)
```

Dostarcza odnośnik do **koła** albo **elipsy** o współrzędnych (***x***, ***y***) i rozmiarach ***d*** *x* ***d*** albo ***w*** *x* ***h***.

```
Figure getSquare(int x, int y, int s)
Figure getRectangle(int x, int y, int w, int h)
```

Dostarcza odnośnik do **kwadratu** albo **prostokąta** o współrzędnych (***x***, ***y***) i rozmiarach ***s*** *x* ***s*** albo ***w*** *x* ***h***.

```
Figure getDiamond(int x, int y, int s)
Figure getDiamond(int x, int y, int w, int h)
```

Dostarcza odnośnik do **rombu** albo **równoległoboku** o współrzędnych (***x***, ***y***) i rozmiarach ***s*** *x* ***s*** albo ***w*** *x* ***h***.

```
Metrics getMetrics(Figure figure)
```

Jeśli figura ***figure*** jest napisem, dostarcza jego metrykę. W przeciwnym razie dostarcza **null**.

```
void drawFigure(Figure figure)
```

Wykreśla obiekt graficzny ***figure***.

```
void setXY(Figure figure, int x, int y)
```

Ustawia na (***x***, ***y***) współrzędne narożnika prostokąta, w który wpisano figurę ***figure***.

```
void setSize(Figure figure, int w, int h)
```

Ustawia na ***w*** *x* ***h*** rozmiary prostokąta, w który wpisano ***figure***.

```
void setStroke(Figure figure, Stroke stroke)
```

Ustawia na ***stroke*** pióro którym będzie wykreślana figura ***figure***.

```
void setPaint(Figure figure, Paint paint)
```

Ustawia na ***paint*** pędzel, którym będzie wykreślana figura ***figure***.

```
void setColor(Figure figure, Color color)
```

Ustawia na **(Paint)*color*** pędzel, którym będzie wykreślana figura ***figure***.

```
void setFont(Figure figure, Font font)
```

Ustawia na ***font*** czcionkę, jaką będzie wykreślany napis figury ***figure***.

```
void setFill(Figure figure, int fill)
```

Ustawia na ***fill*** (**Thin**, **Line**, **Ring**, **Full**) sposób wypełnienia figury ***figure***.

```
int getX(Figure figure)
```

Dostarcza ***odciętą*** narożnika prostokąta, w który wpisano figurę ***figure***.

```
int getY(Figure figure)
```

Dostarcza ***rzędną*** narożnika prostokąta, w który wpisano figurę ***figure***.

```
int getWidth(Figure figure)
```

Dostarcza ***szerokość*** prostokąta, w który wpisano figurę ***figure***.

```
int getHeight(Figure figure)
```

Dostarcza ***wysokość*** prostokąta, w który wpisano figurę ***figure***.

```
Stroke getStroke(Figure figure)
```

Dostarcza odnośnik do ***pióra***, którym będzie wykreślana figura ***figure***.

```
Paint getPaint(Figure figure)
```

Dostarcza odnośnik do ***pędzla***, którym będzie wykreślana figura ***figure***.

```
Paint getColor(Figure figure)
```

Dostarcza odnośnik do ***pędzla***, którym będzie wykreślana figura ***figure***. Jeśli pędzel figury nie wykreśla farbą, dostarcza odnośnik **pusty**.

```
Font getFont(Figure figure)
```

Dostarcza odnośnik do ***czcionki***, jaką będzie wykreślany napis figury ***figure***.

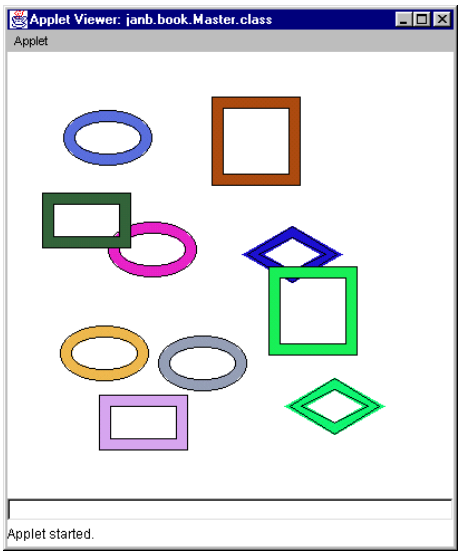
```
int getFill(Figure figure)
```

Dostarcza informacji o sposobie wypełnienia (**Thin**, **Line**, **Ring**, **Full**) figury ***figure***.

> Jeśli do figury nie stosuje się funkcja **getPaint** albo **getFont**, to każda z nich dostarcza **null**

Następujący aplet, pokazany na ekranie ***Wykreślanie obiektów***, wykreśla przykładowe obiekty figurowe.

Ekran
Wykreślanie obiektów




```
import janb.view.*;

public
class Master
    extends View {

    private Figure figure;
    private int r = 40, d = 2*r, s = d, s2 = s/2,
            w = 80, w2 = w/2,
            h = 50, h2 = h/2;
```

```
    public void mouseReleased(int x, int y, int flags)
    {
        if(isMeta(flags)) {
           int what = getRandom(2);
           if(what == 0)
              figure = getCircle(x-r, y-r, d);
           else
              figure = getSquare(x-s2, y-s2, s);
        } else {
            int what = getRandom(3);
            switch(what) {
                case 0:   // elipsa
                    figure = getEllipse(x-w2, y-h2, w, h);
                    break;
                case 1:   // prostokąt
                    figure = getRectangle(x-w2, y-h2, w, h);
                    break;
                case 2:   // romb
                    figure = getDiamond(x-w2, y-h2, w, h);
            }
        }

            // określenie wyglądu figury
        setColor(figure, getColor());
        setFill(figure, Ring);
        setStroke(figure, getStroke(10));

            // wykreślenie figury
        drawFigure(figure);
    }
}
```

Widoki

Po wykreśleniu figury, pozostaje ona na widoku, ale nie dłużej niż do chwili, gdy widok zostanie uszkodzony na skutek **przesłonięcia** przez obce okno albo na skutek ***ikonizacji*** okna przeglądarki. Dlatego każdy wykreślony obiekt graficzny powinien być zapamiętany w ***bazie danych*** apletu i **odtworzony** z niej po każdym wywołaniu funkcji **drawView**.

Do utworzenia bazy danych służy funkcja **getFigures**, a do jej przetwarzania funkcje **addFigure**, **setFigure**, **getFigure**, **drawFigures** i **length**.

```
Figures getFigures()
```

Dostarcza odnośnik do pustej kolekcji.

```
void addFigure(Figures figures, Figure figure)
```

Do kolekcji ***figures*** dokłada figurę ***figure***.

```
void setFigure(Figures figures, int pos, Figure figure)
```

Umieszcza figurę ***figure*** w kolekcji ***figures***, na pozycji ***pos***.

```
Figure getFigure(Figures figures, int pos)
```

Dostarcza odnośnik do figury, która w kolekcji ***figures*** znajduje się na pozycji ***pos***.

```
void drawFigures(Figures figures)
```

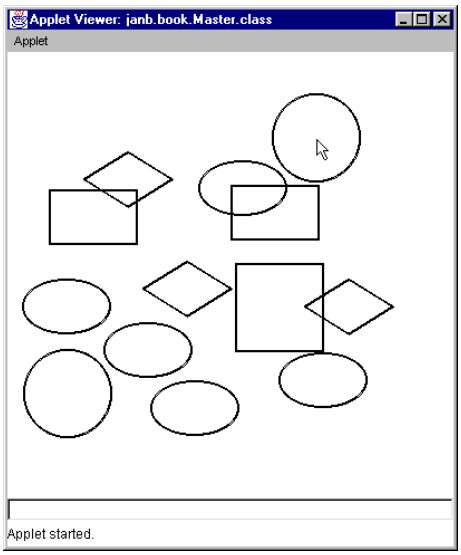
Wykreśla wszystkie figury kolekcji ***figures***.

```
int length(Figures figures)
```

Dostarcza liczbę elementów kolekcji ***figures***.

Następujący aplet, pokazany na ekranie ***Odtwarzanie widoku***, napisano w taki sposób, że wszystkie wykreślone na nim obiekty zostaną automatycznie odtworzone w wypadku uszkodzenia widoku.

Ekran
Odtwarzanie widoku




```
import janb.view.*;

public
class Master
    extends View {

    private Figures dataBase = getFigures();
    private Figure figure;
    private int r = 40, d = 2*r, s = d, s2 = s/2,
            w = 80, w2 = w/2,
            h = 50, h2 = h/2;

    public void drawView()
    {
        drawFigures(dataBase);
    }
```

```
    public void mouseReleased(int x, int y, int flags)
    {
        if(isMeta(flags)) {
           int what = getRandom(2);
           if(what == 0)
              figure = getCircle(x-r, y-r, d);
           else
              figure = getSquare(x-s2, y-s2, s);
        } else {
            int what = getRandom(3);
            switch(what) {
                case 0:   // elipsa
                    figure = getEllipse(x-w2, y-h2, w, h);
                    break;
                case 1:   // prostokąt
                    figure = getRectangle(x-w2, y-h2, w, h);
                    break;
                case 2:   // romb
                    figure = getDiamond(x-w2, y-h2, w, h);
            }
        }

            // zapamiętanie w bazie
        addFigure(dataBase, figure);

            // wykreślenie figury
        drawFigure(figure);
    }
}
```

Narzędzia

Narzędziami do wykreślania są ***pióro*** i ***pędzel***. Pióro służy do **wykreślania** linii, a pędzel do **wypełniania** obszarów. Obszary można wypełniać ***farbą***, ***gradientem*** i ***teksturą***. Do definiowania piór i pędzli służą funkcje **setStroke** i **setPaint**.

```
Color getColor(int red, int green, int blue)
```

Dostarcza kolor farby zdefiniowany przez składniki *RGB*: ***red***, ***green***, ***blue***, każdy o wartości z przedziału **0.. 255**. Np. **getColor(255,255,0)** dostarcza kolor **żółty**.

```
Color getColor()
```

Dostarcza kolor **przypadkowy**, ale nie czarny, nie szary i nie biały.

```
void setColor(Color color)
```

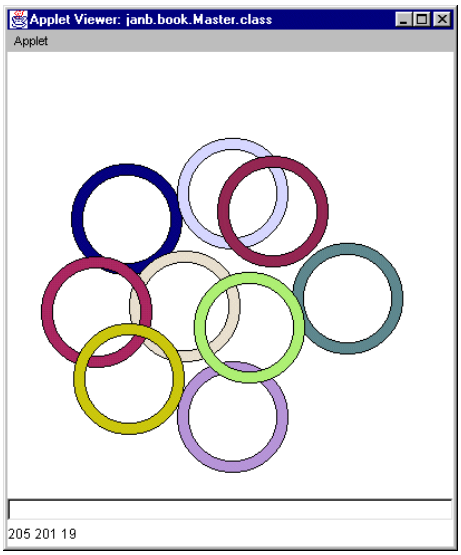
Ustawia kolor bieżący na ***color***.

Można posługiwać się następującymi symbolicznymi oznaczeniami kolorów

Red Green Blue Cyan Magenta Yellow Black White

Następujący aplet, pokazany na ekranie ***Dobieranie kolorów***, wykreśla wokół punktu kliknięcia pierścień w kolorze przypadkowym oraz podaje jego składowe *RGB*.

Ekran
Dobieranie kolorów




```
import janb.view.*;
import java.awt.*;

public
class Master
    extends View {

    private int rad = 50;

    public void mouseReleased(int x, int y)
    {
        x -= rad;
        y -= rad;
        Color color = getColor();
        setColor(color);
        setStroke(10);
        drawCircle(x, y, 2 * rad);
        int r = getRed(color),
            g = getGreen(color),
            b = getBlue(color);
        showMsg("" + r + ' ' + g + ' ' + b);
    }
}
```

Pióro

Pióro może ***rysować*** cienką linią ***ciągłą*** albo dowolnej grubości linią ciągłą albo ***przerywaną***.

Linia przerywana składa się z ***podlinii***, w których występują ***kreski*** i ***przerwy***. Jako ***udział*** kreski określa się **stosunek** długości **kreski** do długości **podlinii**.

```
void setStroke(int width, double ratio)
void setStroke(int width)
```

Ustawia szerokość linii na ***width*** i udział kreski na ***ratio***. Udział domyślny wynosi **1.0**.

```
Stroke getStroke(int width)
```

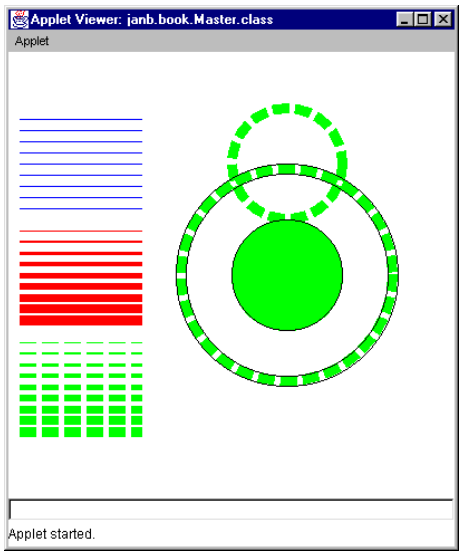
Dostarcza pióro o szerokości ***width***.

```
Stroke getCurrentStroke()
```

Dostarcza aktualne pióro.

Następujący aplet, pokazany na ekranie ***Wybieranie piór***, wykreśla zestaw linii poziomych narysowanych za pomocą kilku piór.

Ekran
Wybieranie piór




```
import janb.view.*;

public
class Master
    extends View {

    public void drawView()
    {
        for(int i = 1; i < 10; i++) {
            int j = 10 * i;

            setColor(Blue);
            setStroke(1);
            drawLine(10, 50+j, 120, 50+j);

            setColor(Red);
```

```
            setStroke(i);
            drawLine(10, 150+j, 120, 150+j);

            setColor(Green);
            setStroke(i, 0.75);
            drawLine(10, 250+j, 120, 250+j);
        }

        drawCircle(200,  50, 100, Line);
        drawCircle(150, 100, 200, Ring);
        drawCircle(200, 150, 100, Full);
    }
}
```

Pędzel

Pędzel może ***malować farbą***, ***gradientem*** i ***teksturą***. Do tworzenia farb, gradientów i tekstur, używa się funkcji **getColor**, **getGradient** i **getTexture**, a do wybrania pędzla funkcji **setColor** i **setPaint**.

Funkcje **setColor** i **setPaint** wzajemnie się wykluczają, gdyż pędzel może malować tylko farbą, tylko gradientem albo tylko teksturą.

```
Paint getGradient(Color from, Color to)
Paint getGradient(Color from, Color to,
                  int x, int y, int w, int h)
```

Dostarcza gradient od koloru ***from*** do koloru ***to***, z rozwinięciem gradientu w całym widoku albo w prostokącie opisanym przez (***x***, ***y***) i ***w*** x ***h***.

```
Paint getTexture(Image image)
```

Dostarcza teksturę zdefiniowaną przez obszar **8** *x* **8** narożnikowych pikseli obrazu ***image***.

```
void setPaint(Paint paint)
```

Ustawia pędzel na malowanie określone przez ***paint***.

```
void setPaint(Color from, Color to)
```

Ustawia pędzel na malowanie gradientem od koloru ***from*** do koloru ***to***.

```
void setPaint(Image image)
```

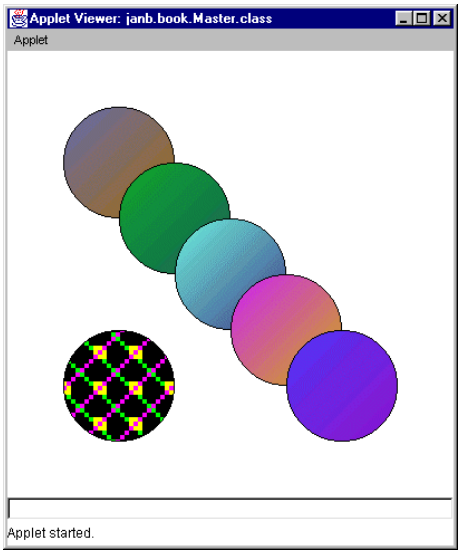
Ustawia pędzel na malowanie teksturą zdefiniowaną przez obraz ***image***.

```
Paint getCurrentPaint()
```

Dostarcza aktualny pędzel.

Następujący aplet, pokazany na ekranie ***Wybieranie pędzli***, wykreśla zestaw kół namalowanych za pomocą pędzla gradientowego i teksturowego.

Ekran
Wybieranie pędzli




```
import janb.view.*;
import java.awt.*;

public
class Master
    extends View {

    private Image image;

    public void initView()
    {
        image = getImage("Carpet.gif");
    }

    public void drawView()
    {
        for(int i = 0; i < 5; i++) {
            int j = 50 * (i+1);

            Color a = getColor(),
                  z = getColor();

            Paint grad = getGradient(a, z);
            setPaint(grad);
            drawCircle(j, j, 100, Full);
        }

        Paint paint = getTexture(image);
        setPaint(paint);
        drawCircle(50, 250, 100, Full);
    }
}
```

Czcionka

Napisy można wykreślać czcionkami o wybranym ***kroju***, ***stylu*** i ***rozmiarze***, a ponadto można je ***malować***: ***farbą***, ***gradientem*** oraz ***teksturą***.

Następujący aplet, pokazany na ekranie ***Dobieranie czcionek***, wykreśla zestaw napisów, posługując się czcionkami o różnych krojach, rozmiarach i stylach.

Ekran
Dobieranie czcionek


```
import janb.view.*;
import java.awt.*;

public
class Master
    extends View {

    private Font serif = getFont("Serif", Bold, 30),
                 mono =  getFont(
                             "Monospaced", Italic, 20
                         ),
                 sans;
    private Image image;

    public void initView()
    {
        image = getImage("Carpet.gif");
    }

    public void drawView()
    {
        setFont(serif);
        drawString("Serif = Times Roman", 20, 80);

        setFont(mono);
        setColor(Red);
        drawString("Monospaced = Courier", 100, 130);
```

```
            sans = getFont(
                        "Sansserif", Font.BOLD | Font.ITALIC, 20
                   );

            setFont(sans);
            setColor(Color.blue);
            drawString("Sansserif = Helvetica", 100, 180);

                // napis gradientowy
            Paint paint = getGradient(Red, Green);
            setPaint(paint);

            Font font = getFont("Serif", BoldItalic, 80);
            setFont(font);
            drawString("Serif", 5, 350);

                // napis teksturowy
            paint = getTexture(image);
            setPaint(paint);
            drawString("Serif", 205, 350);

        }
    }
```

Buforowanie

Każda z funkcji wykreślających może wykonywać swoje czynności nie tylko na **widoku**, ale również w ***buforze*** utworzonym w pamięci operacyjnej. Do przełączania wykreślania między widokiem i buforem służą funkcje **toBuffer** i **toScreen**. Po skompletowaniu obrazu w buforze, można za pomocą funkcji **drawBuffer**, wykreślić go na widoku. Czynność ta może być wykonana bez konieczności przełączenia wykreślania.

```
void toBuffer()
void toBuffer(int w, int h)
void toBuffer(Image image)
```

Przełącza wykreślanie na wyczyszczony bufor o rozmiarach identycznych z rozmiarami widoku, na bufor o rozmiarach ***w*** *x* ***h*** albo na już istniejący bufor ***image***, bez zmiany jego zawartości.

```
void toScreen()
```

Przełącza wykreślanie na widok, po uprzednim zniszczeniu bufora.

```
void toScreen(boolean keep)
```

Przełącza wykreślanie na widok, uzależniając od ***keep*** zachowanie bufora.

```
void drawBuffer()
void drawBuffer(int x, int y)
```

Wykreśla bufor na widoku, w prostokącie o narożniku w punkcie **(*x*,*y*)**; domyślnie **(0,0)**.

```
Image getBuffer()
```

Dostarcza odnośnik do obrazu w buforze.

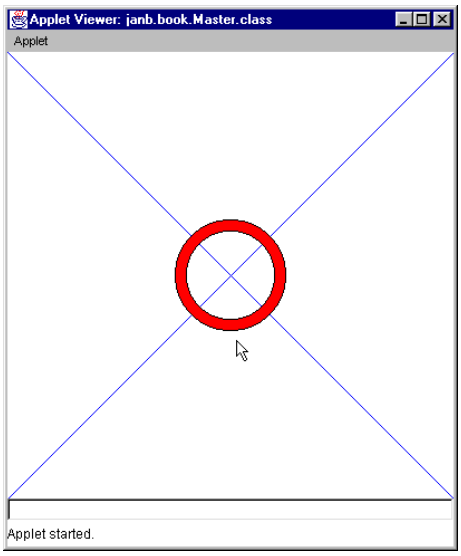
W każdej chwili jest ***aktywny*** tylko **1** bufor, ale umiejętne posługiwanie się funkcjami **getBuffer** i **toBuffer(Image)** umożliwia jednoczesne używanie więcej niż jednego bufora.

Następujący aplet, pokazany na ekranie ***Buforowanie wykreślania***, wykreśla na pulpicie pierścień i umożliwia jego **przeciąganie** za pomocą klawiszy kierunkowych. Gdyby z programu usunięto wywołanie funkcji **toBuffer**, to podczas przeciągania pierścienia wystąpiłoby ***migotanie***. Możliwość usunięcia tego niemiłego dla oka efektu jest jedną z **zalet** buforowania.

Bardziej spostrzegawczy zauważą, że wygląd pierścienia wykreślonego poprzez bufor jest **nieco inny** niż wykreślonego bezpośrednio. Wynika to stąd, że podczas buforowania okrąg jest wykreślany jako **wielokąt** o znacznej (ale nie dostatecznej!) liczbie boków.

```
import janb.view.*;

public
class Master
    extends View {

    private int x, y,
                w, h;

    public void initView()
    {
        w = getWidth();
        h = getHeight();

        x = (w-100) >> 1;
        y = (h-100) >> 1;

        drawView();
        toBuffer();
    }
```

Ekran
Buforowanie wykreślania




```
    public void drawView()
    {
        setColor(Blue);
        setStroke(1);
        drawLine(0, 0, w, h);
        drawLine(w, 0, 0, h);

        setStroke(10);
        setColor(Red);
        drawCircle(x, y, 100, Ring);
    }

    public void keyPressed(int key)
    {
        switch(key) {
            case UpKey:
                y--;  break;
            case DnKey:
                y++; break;
            case LtKey:
                x--; break;
            case RtKey:
                x++; break;
            default:
                beep();
        }
        redraw();
        drawBuffer();
    }
}
```

Kursory

Z panelem, w szczególności z **widokiem**, można związać ***kursor***, zdefiniowany za pomocą ***przezroczystego*** obrazu *GIF*. Do zarządzania kursorami służą funkcje **getCursor** i **setCursor**.

```
Cursor getCursor(String file, int x, int y)
```

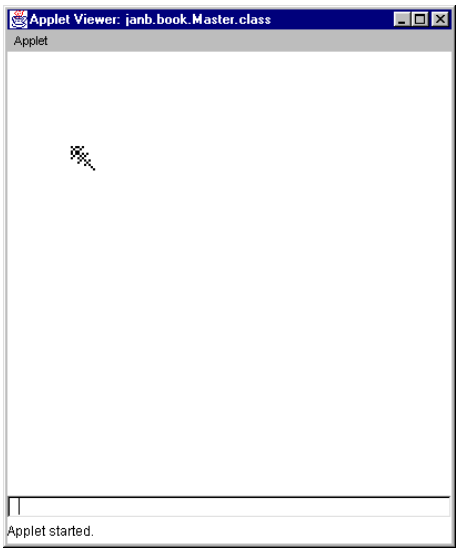
Dostarcza kursor zaprojektowany w pliku ***file*** z punktem wyróżnionym w (***x***, ***y***). Plik powinien mieć rozszerzenie **.gif** i zawierać przezroczysty obraz w formacie *GIF*, o rozmiarach **16** *x* **16** pikseli.

```
void setCursor(Cursor cursor)
```

Związuje z widokiem kursor ***cursor***.

Następujący aplet, pokazany na ekranie ***Własny kursor***, zastępuje standardowy kursor, kursorem zdefiniowanym w pliku ***Cursor.gif***.

Ekran
Własny kursor




```
import janb.view.*;
import java.awt.*;

public
class Master
    extends View {

        // gorący punkt kursora
    private int xHot = 0,
                yHot = 0;
```

```
    public void initView()
    {
            // utworzenie kursora
        Cursor cursor = getCursor("Cursor.gif", xHot, yHot);

            // związanie kursora z widokiem
        setCursor(cursor);
    }
}
```