

Stefan Sokołowski

PROGRAMOWANIE OBIEKTOWE II

wykład 8

**Inst. Informatyki Stosowanej, ANS
Elbląg, 2024/2025**

Strumienie

```
Scanner czyt = new Scanner(System.in);
```

Strumienie

```
Scanner czyt = new Scanner(System.in);
```

↑
klasa

Strumienie

```
Scanner czyt = new Scanner(System.in);
```

↑
klasa

↑
konstruktor

Strumienie

```
Scanner czyt = new Scanner(System.in);
```

↑
klasa

↑
konstruktor

↑
argument
konstruktora

Strumienie

```
Scanner czyt = new Scanner(System.in);
```

↑
klasa

↑
konstruktor

↑
argument
konstruktora

Jakiego typu jest argument `System.in` ?

Strumienie

```
Scanner czyt = new Scanner(System.in);
```

↑
klasa

↑
konstruktor

↑
argument
konstruktora

Jakiego typu jest argument `System.in` ?

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/System.html>

Class `System`

Fields:

`static InputStream in`

— *The "standard" input stream.*

Strumienie

```
Scanner czyt = new Scanner(System.in);
```

↑
klasa

↑
konstruktor

↑
argument
konstruktora

Jakiego typu jest argument `System.in` ?

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/System.html>

Class `System`

Fields:

`static InputStream in`
— *The "standard" input stream.*

Co to są strumienie?



Strumienie

Strumień — dostarcza danych tego samego typu, jedna za drugą, jak `next` w kolekcji, ale:

Strumienie

Strumień — dostarcza danych tego samego typu, jedna za drugą, jak `next` w kolekcji, ale:

- Strumień nie przechowuje elementów

Strumienie

Strumień — dostarcza danych tego samego typu, jedna za drugą, jak `next` w kolekcji, ale:

- Strumień nie przechowuje elementów,
 - albo je generuje na bieżąco

Strumienie

Strumień — dostarcza danych tego samego typu, jedna za drugą, jak `next` w kolekcji, ale:

- Strumień nie przechowuje elementów,
 - albo je generuje na bieżąco,
 - albo czerpie z jakiegoś *źródła*.

Strumienie

Strumień — dostarcza danych tego samego typu, jedna za drugą, jak `next` w kolekcji, ale:

- Strumień nie przechowuje elementów,
 - albo je generuje na bieżąco,
 - albo czerpie z jakiegoś *źródła*.
- Żadne operacje działające na już utworzonym strumieniu niczego w nim nie zmieniają, ale mogą tworzyć nowy strumień.

Strumienie

Strumień — dostarcza danych tego samego typu, jedna za drugą, jak `next` w kolekcji, ale:

- Strumień nie przechowuje elementów,
 - albo je generuje na bieżąco,
 - albo czerpie z jakiegoś *źródła*.
- Żadne operacje działające na już utworzonym strumieniu niczego w nim nie zmieniają, ale mogą tworzyć nowy strumień.
- Operacje działają na strumieniu *leniwie*

Strumienie

Strumień — dostarcza danych tego samego typu, jedna za drugą, jak `next` w kolekcji, ale:

- Strumień nie przechowuje elementów,
 - albo je generuje na bieżąco,
 - albo czerpie z jakiegoś *źródła*.
- Żadne operacje działające na już utworzonym strumieniu niczego w nim nie zmieniają, ale mogą tworzyć nowy strumień.
- Operacje działają na strumieniu *leniwie*, to znaczy elementy nie są pobierane (generowane) zanim coś w programie ich nie zażąda.

Strumienie

Strumień — dostarcza danych tego samego typu, jedna za drugą, jak `next` w kolekcji, ale:

- Strumień nie przechowuje elementów,
 - albo je generuje na bieżąco,
 - albo czerpie z jakiegoś *źródła*.
- Żadne operacje działające na już utworzonym strumieniu niczego w nim nie zmieniają, ale mogą tworzyć nowy strumień.
- Operacje działają na strumieniu *leniwie*, to znaczy elementy nie są pobierane (generowane) zanim coś w programie ich nie zażąda.
- Do generowania i obsługi strumieni najczęściej używa się *lambd*

Strumienie

Strumień — dostarcza danych tego samego typu, jedna za drugą, jak `next` w kolekcji, ale:

- Strumień nie przechowuje elementów,
 - albo je generuje na bieżąco,
 - albo czerpie z jakiegoś *źródła*.
- Żadne operacje działające na już utworzonym strumieniu niczego w nim nie zmieniają, ale mogą tworzyć nowy strumień.
- Operacje działają na strumieniu *leniwie*, to znaczy elementy nie są pobierane (generowane) zanim coś w programie ich nie zażąda.
- Do generowania i obsługi strumieni najczęściej używa się *lambd* — wyrażen postaci `x -> ...`.

Strumienie

Przykład: (nieskończony trójkąt z gwiazdek...)

Strumienie

Przykład: (nieskończony trójkąt z gwiazdek...)

Generowanie strumienia nieskończonego:

```
Stream<String> gw = Stream.iterate( "*" + "\n", s -> "*" + s );
```

Strumienie

Przykład: (nieskończony trójkąt z gwiazdek...)

Generowanie strumienia nieskończonego:

```
Stream<String> gw = Stream.iterate( "*\n", s -> "*" + s );
```

↑
początek

Strumienie

Przykład: (nieskończony trójkąt z gwiazdek...)

Generowanie strumienia nieskończonego:

```
Stream<String> gw = Stream.iterate( "*"\\n", s -> "*" + s );
```

początek

LAMBDA:
przejsie do
następnego

Strumienie

Przykład: (nieskończony trójkąt z gwiazdek...)

Generowanie strumienia nieskończonego:

```
Stream<String> gw = Stream.iterate( "*\n", s -> "*" + s );
```

↑
początek

LAMBDA:
przejdźcie do
następnego

Drukowanie danych ze strumienia:

```
gw.forEach(  
    s -> {  
        System.out.print(s);  
        try { Thread.sleep(100); } catch (InterruptedException ex) { }  
    }  
);
```

Strumienie

Przykład: (nieskończony trójkąt z gwiazdek...)

Generowanie strumienia nieskończonego:

```
Stream<String> gw = Stream.iterate( "*\n", s -> "*" + s );
```

↑
początek

LAMBDA:
przejsie do
następnego

Drukowanie danych ze strumienia:

```
gw.forEach(  
    s -> {  
        System.out.print(s);  
        try { Thread.sleep(100); } catch (InterruptedException ex) { }  
    }  
);
```

— lambda określająca, co robić na elementach

Strumienie

Przykład: (nieskończony trójkąt z gwiazdek...)

Generowanie strumienia nieskończonego:

```
Stream<String> gw = Stream.iterate( "*\n", s -> "*" + s );
```

↑
początek

LAMBDA:
przejdzie do
następnego

Drukowanie danych ze strumienia:

```
gw.forEach(  
    s -> {  
        System.out.print(s);  
        try { Thread.sleep(100); } catch (InterruptedException ex) { }  
    }  
);
```

lambda określająca, co robić na elementach

Strumienie

Przykład: (liczby pierwsze i inne...)

Generowanie strumienia nieskończonego:

```
Stream<Integer> nat = Stream.iterate( 0, n -> n+1 );
```

Strumienie

Przykład: (liczby pierwsze i inne...)

Generowanie strumienia nieskończonego:

```
Stream<Integer> nat = Stream.iterate( 0, n -> n+1 );
```

↑
początek

Strumienie

Przykład: (liczby pierwsze i inne...)

Generowanie strumienia nieskończonego:

```
Stream<Integer> nat = Stream.iterate( 0, n -> n+1 );
```

↑
początek

↖
LAMBDA:
przejdźcie do
następnego

Strumienie

Przykład: (liczby pierwsze i inne...)

Generowanie strumienia nieskończonego:

```
Stream<Integer> nat = Stream.iterate( 0, n -> n+1 );
```

↑
początek

LAMBDA:
przejsie do
następnego

Drukowanie ze strumienia:

```
nat.forEach(  
    n -> System.out.print(" " + n)  
);
```

↑
lambda określająca, co robić na elementach

Strumienie

Przykład: (liczby pierwsze i inne...)

Generowanie strumienia nieskończonego:

```
Stream<Integer> nat = Stream.iterate( 0, n -> n+1 );
```

Drukowanie ze strumienia:

```
nat.forEach(  
    n -> System.out.print(" " + n)  
);
```

lambda określająca, co robić na elementach

początek

LAMBDA:
*przejście do
następnego*

Filtrowanie strumienia:

```
nat  
    .filter(n -> n%2 != 0)  
    .forEach(  
        n -> System.out.print(" " + n)  
    );
```

*przepuszcza
tylko
nieparzyste*

Strumienie

Przykład: (liczby pierwsze i inne...)

Generowanie strumienia nieskończonego:

```
Stream<Integer> nat = Stream.iterate( 0, n -> n+1 );
```

Drukowanie ze strumienia:

```
nat.forEach(
    n -> System.out.print(" " + n)
);
```

lambda określająca, co robić na elementach

początek

LAMBDA:
*przejście do
następnego*

Filtrowanie strumienia:

```
nat
    .filter(n -> n%2 != 0)
    .forEach(
        n -> System.out.print(" " + n)
    );
```

*przepuszcza
tylko
nieparzyste*

Strumienie

Przykład:

Generowanie strumienia skończonego z kolekcji

Strumienie

Przykład:

Generowanie strumienia skończonego z kolekcji:

```
ArrayList<String> tk = new Słowa(args[0]).tekscik();  
tk.stream() ...
```

Strumienie

Przykład:

Generowanie strumienia skończonego z kolekcji:

```
ArrayList<String> tk = new Słowa(args[0]).tekscik();  
tk.stream() ...
```

Drukowanie ze strumienia:

```
tk.stream()  
    .forEach( sl -> System.out.print(sl) );
```

Strumienie

Przykład:

Generowanie strumienia skończonego z kolekcji:

```
ArrayList<String> tk = new Słowa(args[0]).tekscik();  
tk.stream() ...
```

Drukowanie ze strumienia:

```
tk.stream()  
    .forEach( sl -> System.out.print(sl) );
```

Filtrowanie strumienia:

```
... .filter( sl -> sl.trim().length()>10 )
```

— *tylko słowa dłuższe niż 10 znaków.*

Strumienie

Przykład:

Generowanie strumienia skończonego z kolekcji:

```
ArrayList<String> tk = new Słowa(args[0]).tekscik();  
tk.stream() ...
```

Drukowanie ze strumienia:

```
tk.stream()  
    .forEach( sl -> System.out.print(sl) );
```

Filtrowanie strumienia:

```
... .filter( sl -> sl.trim().length()>10 )
```

— *tylko słowa dłuższe niż 10 znaków.*

```
iis.ans-elblag.pl/~stefan/Dydaktyka/Prog0biII/Prog07/P3a_Słowa.java  
iis.ans-elblag.pl/~stefan/Dydaktyka/Prog0biII/Prog07/P3_Kopia.java
```

Operacje na strumieniach

Generowanie:

- nieskończony przez iterowanie:
`Stream.iterate(..., ...)`

Operacje na strumieniach

Generowanie:

- nieskończony przez iterowanie:

`Stream.iterate(..., ...)`

↑
pierwszy
element

↖
jak z elementu
zrobić następny

Operacje na strumieniach

Generowanie:

- nieskończony przez iterowanie:

`Stream.iterate(..., ...)`

↑
pierwszy
element

↖
jak z elementu
zrobić następny

- skończony z listy:

`lista.stream()`

Operacje na strumieniach

Filtrowanie:

`.filter( ... )` — „odcedza” z tego strumienia elementy nie spełniające podanego predykatu (zostawia tylko spełniające)

Operacje na strumieniach

Filtrowanie:

`.filter( ... )` — „odcedza” z tego strumienia elementy nie spełniające podanego predykatu (zostawia tylko spełniające)

Mapowanie:

`.map( ... )` — tworzy nowy strumień z wyników działania podanej funkcji na elementach tego strumienia

Operacje na strumieniach

Filtrowanie:

`.filter( ... )` — „odcedza” z tego strumienia elementy nie spełniające podanego predykatu (zostawia tylko spełniające)

Mapowanie:

`.map( ... )` — tworzy nowy strumień z wyników działania podanej funkcji na elementach tego strumienia

Działania sterowane strumieniem:

`.forEach( ... )` — na każdym elemencie tego strumienia wykonuje podaną funkcję

Operacje na strumieniach

Filtrowanie:

`.filter( ... )` — „odcedza” z tego strumienia elementy nie spełniające podanego predykatu (zostawia tylko spełniające)

Mapowanie:

`.map( ... )` — tworzy nowy strumień z wyników działania podanej funkcji na elementach tego strumienia

Działania sterowane strumieniem:

`.forEach( ... )` — na każdym elemencie tego strumienia wykonuje podaną funkcję

... są jeszcze inne...

— patrz: dokumentacja Javy

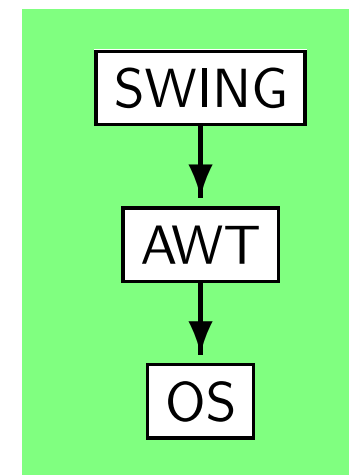
Okna, ramki, grafika, . . .

Okna, ramki, grafika, . . .

- *AWT*
- *SWING*

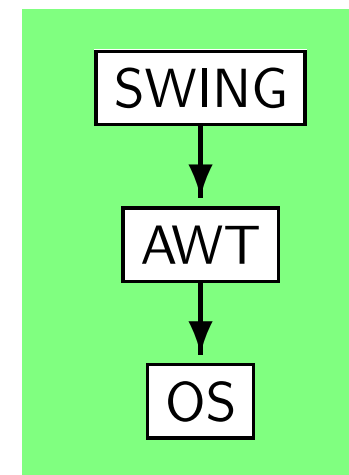
Okna, ramki, grafika, ...

- *AWT*
- *SWING*



Okna, ramki, grafika, ...

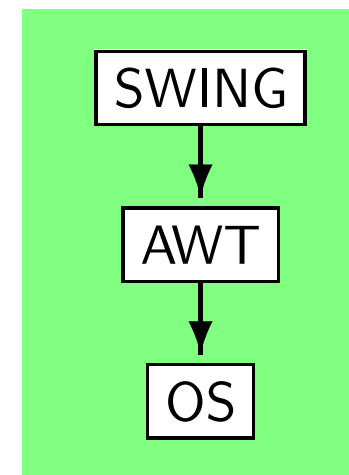
- **AWT** — *A*bstract *W*indow *T*oolkit
- **SWING** — ?



Okna, ramki, grafika, ...

- **AWT** — *A*bstract *W*indow *T*oolkit
- **SWING** — ?

Nikt dobrze nie wie, dlaczego **SWING**...
... może **SWING** dancing...
... może *Some* **W**indowing **G**raphics...



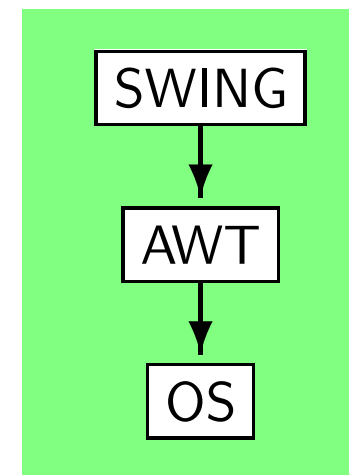
Okna, ramki, grafika, ...

- **AWT** — *A*bstract *W*indow *T*oolkit
- **SWING** — ?

Nikt dobrze nie wie, dlaczego **SWING**...

... może **SWING** dancing...

... może *Some* **W**indowing **G**raphics — to prawdopodobnie jest późna „ludowa etymologia”...



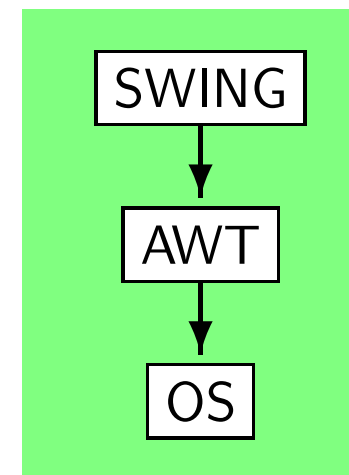
Okna, ramki, grafika, ...

- **AWT** — *A*bstrakt *W*indow *T*oolkit
- **SWING** — ?

Nikt dobrze nie wie, dlaczego **SWING**...

... może **SWING** dancing...

... może *Some* **W**indowing **G**raphics — to prawdopodobnie jest późna „ludowa etymologia”...



Pakiet `javax.swing` — okna i ramki, w których można umieszczać

- grafikę
 - własną: linie, obszary, kolory, itp.

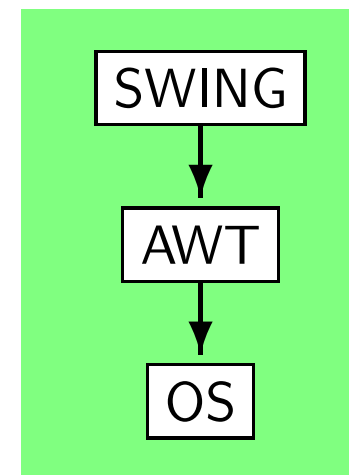
Okna, ramki, grafika, ...

- **AWT** — *A*bstrakt *W*indow *T*oolkit
- **SWING** — ?

Nikt dobrze nie wie, dlaczego **SWING**...

...może **SWING** dancing...

...może *Some* **W**indowing **G**raphics — to prawdopodobnie jest późna „ludowa etymologia”...



Pakiet `javax.swing` — okna i ramki, w których można umieszczać

- grafikę
 - własną: linie, obszary, kolory, itp.,
 - importowane obrazki

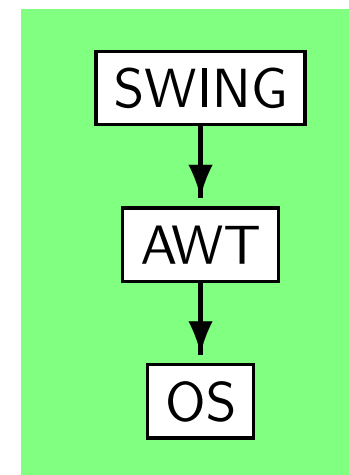
Okna, ramki, grafika, ...

- **AWT** — *A*bstr**W**indow *T*oolkit
- **SWING** — ?

Nikt dobrze nie wie, dlaczego **SWING**...

...może **SWING** dancing...

...może *Some Win*dowing *G*raphics — to prawdopodobnie jest późna „ludowa etymologia”...



Pakiet `javax.swing` — okna i ramki, w których można umieszczać

- grafikę
 - własną: linie, obszary, kolory, itp.,
 - importowane obrazki;
- **GUI** — *G*raphical *U*ser *I*nterface
 - przyciski, ramki dialogowe, itp.

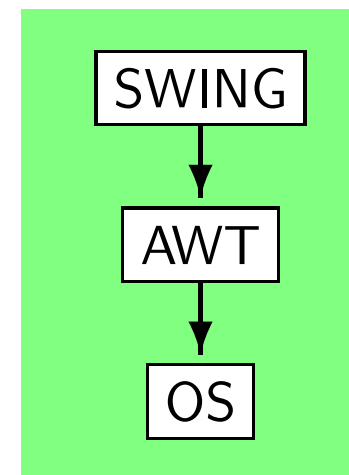
Okna, ramki, grafika, ...

- **AWT** — *A*bstract *W*indow *T*oolkit
- **SWING** — ?

Nikt dobrze nie wie, dlaczego **SWING**...

...może **SWING** dancing...

...może *Some* **W**indowing **G**raphics — to prawdopodobnie jest późna „ludowa etymologia”...



Pakiet `javax.swing` — okna i ramki, w których można umieszczać

- grafikę
 - własną: linie, obszary, kolory, itp.,
 - importowane obrazki;
- **GUI** — *G*raphical *U*ser *I*nterface
 - przyciski, ramki dialogowe, itp.,
 - reakcje na zdarzenia

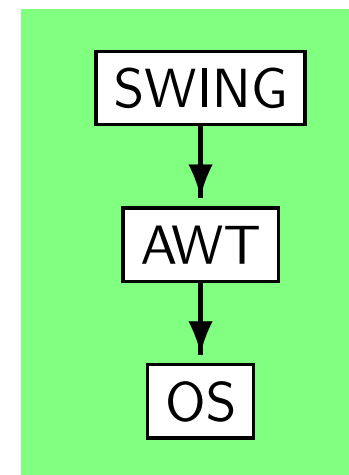
Okna, ramki, grafika, ...

- **AWT** — *A*bstract *W*indow *T*oolkit
- **SWING** — ?

Nikt dobrze nie wie, dlaczego **SWING**...

...może **SWING** dancing...

...może *Some* **W**indowing **G**raphics — to prawdopodobnie jest późna „ludowa etymologia”...



Pakiet `javax.swing` — okna i ramki, w których można umieszczać

- grafikę
 - własną: linie, obszary, kolory, itp.,
 - importowane obrazki;
- **GUI** — *G*raphical *U*ser *I*nterface
 - przyciski, ramki dialogowe, itp.,
 - reakcje na zdarzenia;
- inne.

Malarstwo w Jawie

Malarstwo w Jawie

Przygotowanie sztalugi:

```
import javax.swing.JFrame;

class SamaRama {
    public static void main(String[] args) {
        JFrame rama = new JFrame();
        rama.setVisible(true);
    }
}
```



Malarstwo w Jawie

Przygotowanie sztalugi:

```
import javax.swing.JFrame;

class SamaRama {
    public static void main(String[] args) {
        JFrame rama = new JFrame();
        rama.setVisible(true);
    }
}
```

konstruktor



Malarstwo w Jawie

Przygotowanie sztalugi:

```
import javax.swing.JFrame;

class SamaRama {
    public static void main(String[] args) {
        JFrame rama = new JFrame();
        rama.setVisible(true);
    }
}
```

konstruktor

uwidacznianie



Malarstwo w Jawie

Przygotowanie sztalugi:

```
import javax.swing.JFrame;

class SamaRama {
    public static void main(String[] args) {
        JFrame rama = new JFrame();
        rama.setVisible(true);
    }
}
```

konstruktor

uwidacznianie



Sam konstruktor tworzy okno „bez własności”.

Malarstwo w Jawie

Przygotowanie sztalugi:

```
import javax.swing.JFrame;

class SamaRama {
    public static void main(String[] args) {
        JFrame rama = new JFrame();
        rama.setVisible(true);
    }
}
```

konstruktor

uwidacznianie



Sam konstruktor tworzy okno „bez własności”.

iis.ans-elblag.pl/~stefan/Dydaktyka/ProgObiII/Prog08/P0_SamaRama.java

Malarstwo w Jawie

Podstawowe własności okna

Malarstwo w Jawie

Podstawowe właściwości okna:

```
import javax.swing.*;

class RamaWym {
    public static void main(String[] args) {
        JFrame rama = new JFrame();
        rama.setBounds(0,0,300,200);
        rama.setTitle("Moja ramka");
        rama.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        rama.setVisible(true);
    }
}
```

Malarstwo w Jawie

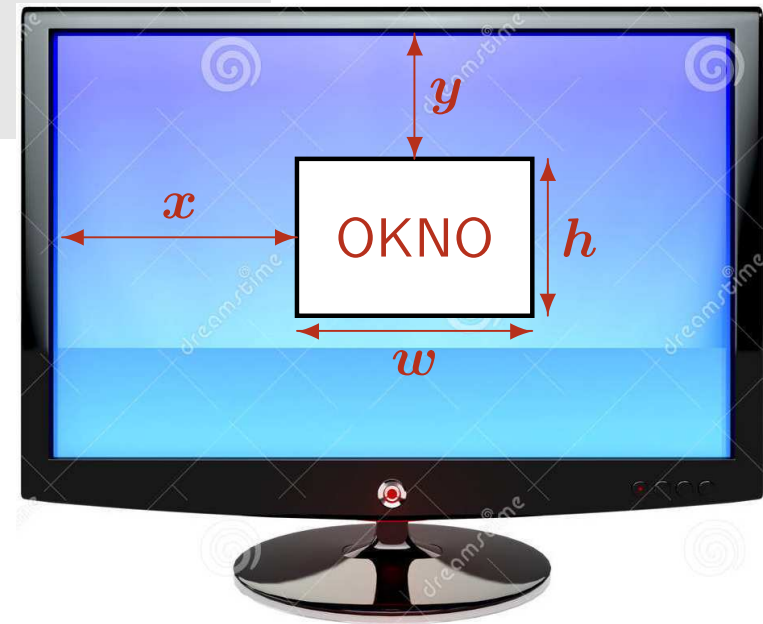
Podstawowe właściwości okna:

```
import javax.swing.*;

class RamaWym {
    public static void main(String[] args) {
        JFrame rama = new JFrame();
        rama.setBounds(0,0,300,200);
        rama.setTitle("Moja ramka");
        rama.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        rama.setVisible(true);
    }
}
```

← geometria okna

`rama.setBounds( $x$ ,  $y$ ,  $w$ ,  $h$ )`
albo równoważnie:
`rama.setLocation( $x$ ,  $y$ )`
`rama.setSize( $w$ ,  $h$ )`
(wszystko liczone w pikselach)



Malarstwo w Jawie

Podstawowe właściwości okna:

```
import javax.swing.*;
```

```
class RamaWym {  
    public static void main(String[] args) {  
        JFrame rama = new JFrame();  
        rama.setBounds(0,0,300,200);  
        rama.setTitle("Moja ramka");  
        rama.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        rama.setVisible(true);  
    }  
}
```

← geometria okna

← tytuł ramki

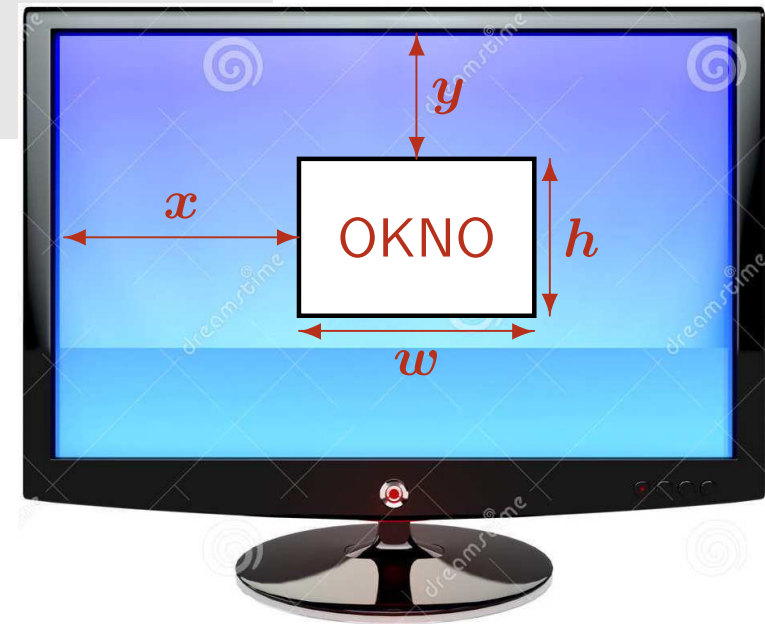
`rama.setBounds( $x$ ,  $y$ ,  $w$ ,  $h$ )`

albo równoważnie:

`rama.setLocation( $x$ ,  $y$ )`

`rama.setSize( $w$ ,  $h$ )`

(wszystko liczone w pikselach)



Malarstwo w Jawie

Podstawowe właściwości okna:

```
import javax.swing.*;
```

```
class RamaWym {
    public static void main(String[] args) {
        JFrame rama = new JFrame();
        rama.setBounds(0,0,300,200);
        rama.setTitle("Moja ramka");
        rama.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        rama.setVisible(true);
    }
}
```

← geometria okna

← tytuł ramki

← po zabicu okna
zakończyć program

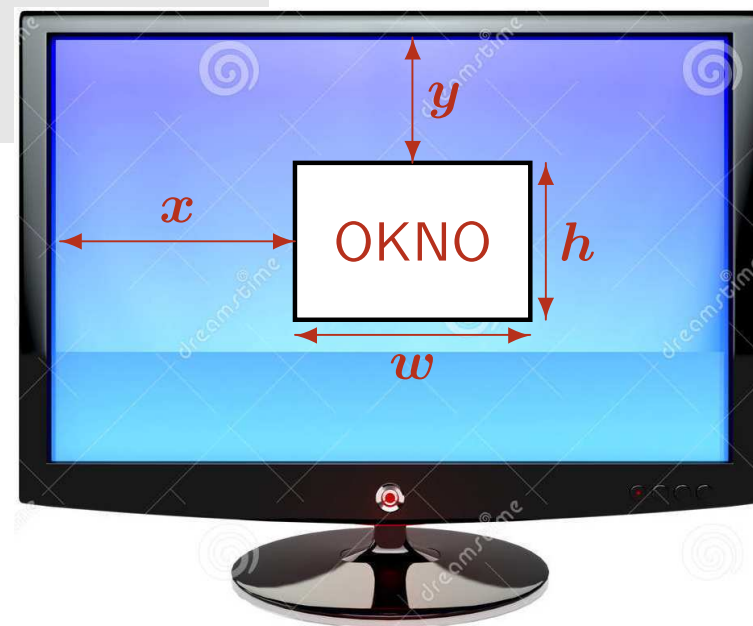
`rama.setBounds( $x$ ,  $y$ ,  $w$ ,  $h$ )`

albo równoważnie:

`rama.setLocation( $x$ ,  $y$ )`

`rama.setSize( $w$ ,  $h$ )`

(wszystko liczone w pikselach)



Malarstwo w Jawie

Podstawowe właściwości okna:

```
import javax.swing.*;
```

```
class RamaWym {
    public static void main(String[] args) {
        JFrame rama = new JFrame();
        rama.setBounds(0,0,300,200);
        rama.setTitle("Moja ramka");
        rama.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        rama.setVisible(true);
    }
}
```

← geometria okna

← tytuł ramki

← po zabicu okna
zakończyć program

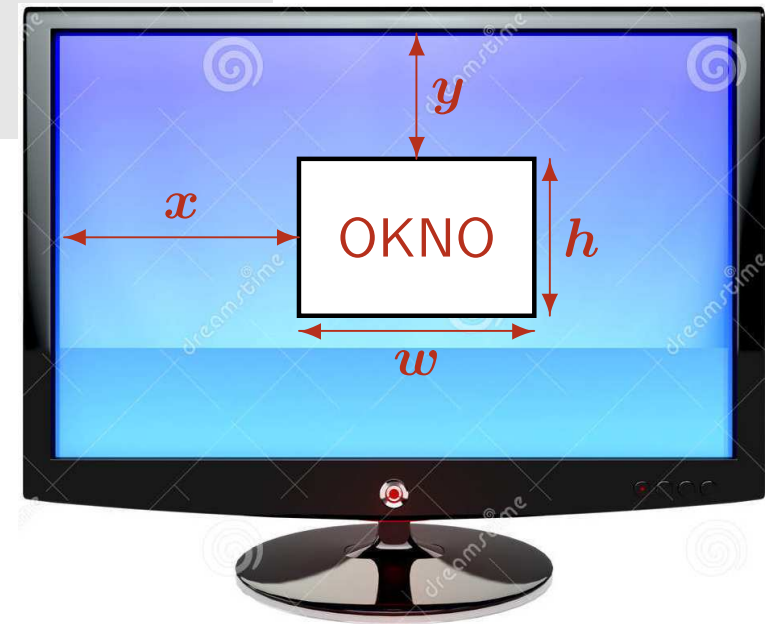
`rama.setBounds( $x$ ,  $y$ ,  $w$ ,  $h$ )`

albo równoważnie:

`rama.setLocation( $x$ ,  $y$ )`

`rama.setSize( $w$ ,  $h$ )`

(wszystko liczone w pikselach)



Malarstwo w Jawie

Uwagać na podawanie geometrii okna w pikselach!

Malarstwo w Jawie

Uważać na podawanie geometrii okna w pikselach!

Jeśli nie zależy ona od rozmiarów i rozdzielczości ekranu, to na różnych komputerach może wyjść tak

Malarstwo w Jawie

Uważać na podawanie geometrii okna w pikselach!

Jeśli nie zależy ona od rozmiarów i rozdzielczości ekranu, to na różnych komputerach może wyjść tak

mały ekran
800x600



Malarstwo w Jawie

Uważać na podawanie geometrii okna w pikselach!

Jeśli nie zależy ona od rozmiarów i rozdzielczości ekranu, to na różnych komputerach może wyjść tak

mały ekran
800x600



duży ekran
1600x1200



Malarstwo w Jawie

Uważać na podawanie geometrii okna w pikselach!

Jeśli nie zależy ona od rozmiarów i rozdzielczości ekranu, to na różnych komputerach może wyjść tak

mały ekran
800x600



duży ekran
1600x1200



mały ekran
1600x1200



Malarstwo w Jawie

Uważać na podawanie geometrii okna w pikselach!

Jeśli nie zależy ona od rozmiarów i rozdzielczości ekranu, to na różnych komputerach może wyjść tak

mały ekran
800x600



duży ekran
1600x1200



mały ekran
1600x1200



Okno, dobrze wyglądające na jednym ekranie, może być nieczytelne na innym.

Malarstwo w Jawie

Przykład: okno o rozmiarach

pół szerokości ekranu × pół wysokości ekranu

umieszczone w centrum ekranu

Malarstwo w Jawie

Przykład: okno o rozmiarach

pół szerokości ekranu × pół wysokości ekranu

umieszczone w centrum ekranu:

```
int szerEkranu = Toolkit.getDefaultToolkit().getScreenSize().width;  
int wysEkranu = Toolkit.getDefaultToolkit().getScreenSize().height;  
rama.setBounds(szerEkranu/4, wysEkranu/4,  
               szerEkranu/2, wysEkranu/2);
```

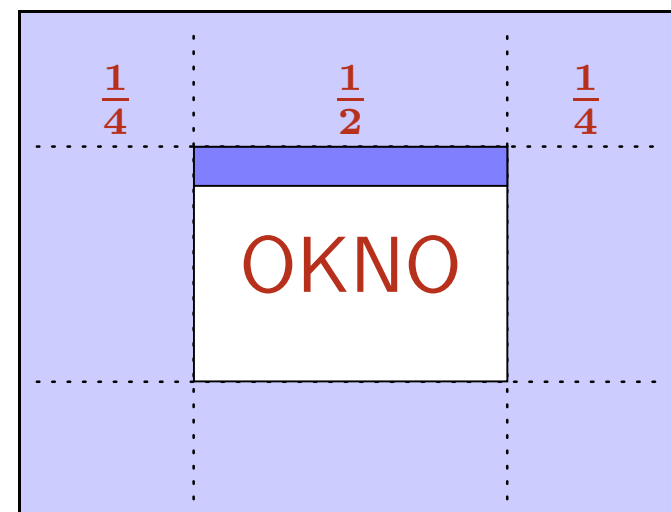
Malarstwo w Jawie

Przykład: okno o rozmiarach

pół szerokości ekranu $\times$ pół wysokości ekranu

umieszczone w centrum ekranu:

```
int szerEkranu = Toolkit.getDefaultToolkit().getScreenSize().width;  
int wysEkranu = Toolkit.getDefaultToolkit().getScreenSize().height;  
rama.setBounds(szerEkranu/4, wysEkranu/4,  
               szerEkranu/2, wysEkranu/2);
```



Malarstwo w Jawie

Przykład: okno o rozmiarach

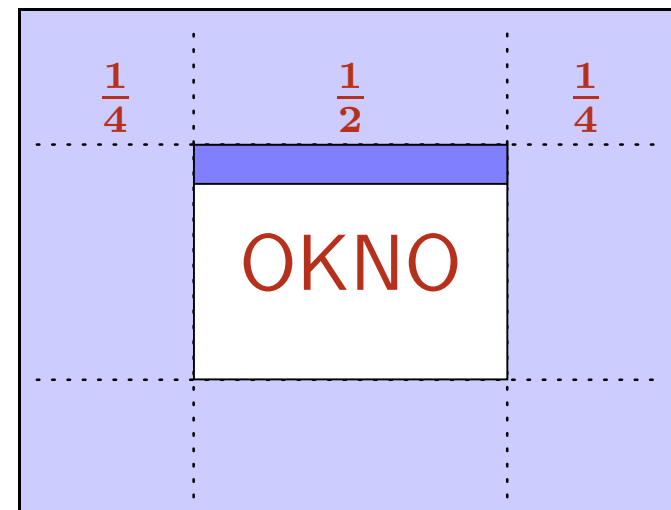
pół szerokości ekranu $\times$ pół wysokości ekranu

umieszczone w centrum ekranu:

```
int szerEkranu = Toolkit.getDefaultToolkit().getScreenSize().width;  
int wysEkranu = Toolkit.getDefaultToolkit().getScreenSize().height;  
rama.setBounds(szerEkranu/4, wysEkranu/4,  
               szerEkranu/2, wysEkranu/2);
```

Uwaga:

górny pasek okna **wlicza się** w jego wysokość;
więc na rysowanie wewnątrz okna
mamy trochę mniej miejsca.



Malarstwo w Jawie

Przykład: okno o rozmiarach

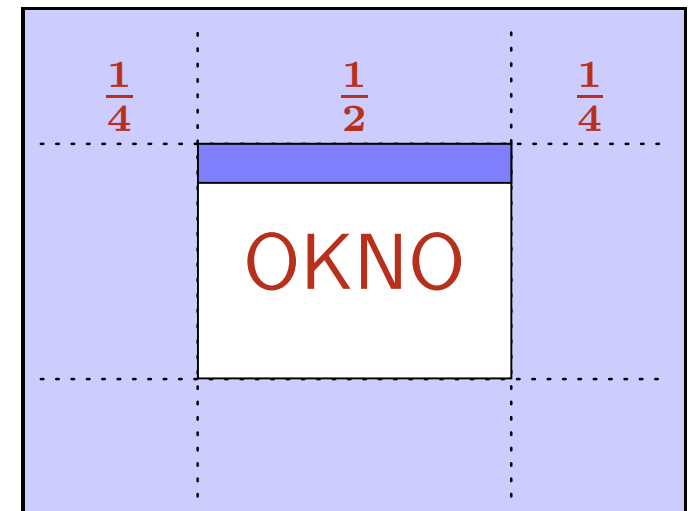
pół szerokości ekranu $\times$ pół wysokości ekranu

umieszczone w **centrum** ekranu:

```
int szerEkranu = Toolkit.getDefaultToolkit().getScreenSize().width;  
int wysEkranu = Toolkit.getDefaultToolkit().getScreenSize().height;  
rama.setBounds(szerEkranu/4, wysEkranu/4,  
               szerEkranu/2, wysEkranu/2);
```

Uwaga:

górny pasek okna **wlicza się** w jego wysokość;
więc na rysowanie wewnątrz okna
mamy trochę mniej miejsca.



Malarstwo w Jawie

Do wypełniania okna treścią służy *callback* czyli wywołanie zwrotne.

Malarstwo w Jawie

Do **wypełniania okna treścią** służy *callback* czyli wywołanie zwrotne.

Użytkownik dostarcza funkcję rysującą —

— odpowiednie klasy Jawy ją wywołują.

Malarstwo w Jawie

Do **wypełniania okna treścią** służy *callback* czyli wywołanie zwrotne.

Użytkownik dostarcza funkcję rysującą —

— odpowiednie klasy Jawy ją wywołują.

Przykład:

Abstrakcyjna klasa *JComponent*

z abstrakcyjną metodą *paintComponent(Graphics g)*

Malarstwo w Jawie

Do **wypełniania okna treścią** służy *callback* czyli wywołanie zwrotne.

Użytkownik dostarcza funkcję rysującą —

— odpowiednie klasy Jawy ją wywołują.

Przykład:

Abstrakcyjna klasa *JComponent*

z abstrakcyjną metodą *paintComponent(Graphics g)*,
którą trzeba wypełnić czymś konkretnym

Malarstwo w Jawie

Do **wypełniania okna treścią** służy *callback* czyli wywołanie zwrotne.

Użytkownik dostarcza funkcję rysującą —
— odpowiednie klasy Jawy ją wywołują.

Przykład:

Abstrakcyjna klasa `JComponent`
z abstrakcyjną metodą `paintComponent(Graphics g)`,
którą trzeba wypełnić czymś konkretnym:

```
rama.add(new JComponent() {  
    public void paintComponent(Graphics g) {  
        g.setColor(Color.BLUE); g.fillOval(50,50,300,200);  
        g.setColor(Color.RED); g.drawOval(100,0,200,300);  
    }  
});
```

Malarstwo w Jawie

Do **wypełniania okna treścią** służy *callback* czyli wywołanie zwrotne.

Użytkownik dostarcza funkcję rysującą —
— odpowiednie klasy Jawy ją wywołują.

Przykład:

Abstrakcyjna klasa `JComponent`
z abstrakcyjną metodą `paintComponent(Graphics g)`,
którą trzeba wypełnić czymś konkretnym:

```
rama.add(new JComponent() {  
    public void paintComponent(Graphics g) {  
        g.setColor(Color.BLUE); g.fillOval(50,50,300,200);  
        g.setColor(Color.RED); g.drawOval(100,0,200,300);  
    }  
});
```

iis.ans-elblag.pl/~stefan/Dydaktyka/ProgObiII/Prog08/P3_Elipsy.java

Malarstwo w Jawie

Zawartość okna:

Napisy, rysunki itp. umieszcza się w „komponencie”, który rozszerza abstrakcyjną klasę `JComponent`

Malarstwo w Jawie

Zawartość okna:

Napisy, rysunki itp. umieszcza się w „komponencie”, który rozszerza abstrakcyjną klasę `JComponent`; to rozszerzenie

- **musi** zdefiniować metodę

```
public void paintComponent(Graphics g)
```

Malarstwo w Jawie

Zawartość okna:

Napisy, rysunki itp. umieszcza się w „komponencie”, który rozszerza abstrakcyjną klasę `JComponent`; to rozszerzenie

- **musi** zdefiniować metodę
`public void paintComponent(Graphics g)`
- w tej metodzie **może** zmienić ustawienia domyślne klasy `Graphics`

Malarstwo w Jawie

Zawartość okna:

Napisy, rysunki itp. umieszcza się w „komponencie”, który rozszerza abstrakcyjną klasę `JComponent`; to rozszerzenie

- **musi** zdefiniować metodę
`public void paintComponent(Graphics g)`
- w tej metodzie **może** zmienić ustawienia domyślne klasy `Graphics`

```
class Rysunek extends JComponent {  
    public void paintComponent(Graphics g) {  
        . . . . . ← komendy obiektu graficznego g  
    }  
}
```

Malarstwo w Jawie

Zawartość okna:

Napisy, rysunki itp. umieszcza się w „komponencie”, który rozszerza abstrakcyjną klasę `JComponent`; to rozszerzenie

- **musi** zdefiniować metodę
`public void paintComponent(Graphics g)`
- w tej metodzie **może** zmienić ustawienia domyślne klasy `Graphics`

```
class Rysunek extends JComponent {
    public void paintComponent(Graphics g) {
        . . . . . ← komendy obiektu graficznego g
    }
}
```

```
. . . . .
rama.add(new Rysunek()); ← dodanie komponentu do ramy
. . . . .
```

Malarstwo w Jawie

Niektóre narzędzia malarskie



Malarstwo w Jawie

Niektóre narzędzia malarskie:

- `drawLine(int x1, int y1, int x2, int y2)`
rysuje odcinek od `(x1,y1)` do `(x2,y2)`



Malarstwo w Jawie

Niektóre narzędzia malarskie:

- `drawLine(int x1, int y1, int x2, int y2)`
rysuje odcinek od `(x1,y1)` do `(x2,y2)`
- `drawRect(int x, int y, int w, int h)`
rysuje prostokąt **bez wnętrza** o szerokości `w`, wysokości `h` i lewym górnym rogu w `(x,y)`



Malarstwo w Jawie

Niektóre narzędzia malarskie:

- `drawLine(int x1, int y1, int x2, int y2)`
rysuje odcinek od `(x1,y1)` do `(x2,y2)`
- `drawRect(int x, int y, int w, int h)`
rysuje prostokąt **bez wnętrza** o szerokości `w`, wysokości `h` i lewym górnym rogu w `(x,y)`
- `fillRect(int x, int y, int w, int h)`
rysuje **wypełniony** prostokąt o szerokości `w`, wysokości `h` i lewym górnym rogu w `(x,y)`



Malarstwo w Jawie

Niektóre narzędzia malarskie:

- `drawLine(int x1, int y1, int x2, int y2)`
rysuje odcinek od $(x1, y1)$ do $(x2, y2)$
- `drawRect(int x, int y, int w, int h)`
rysuje prostokąt **bez wnętrza** o szerokości w , wysokości h i lewym górnym rogu w (x, y)
- `fillRect(int x, int y, int w, int h)`
rysuje **wypełniony** prostokąt o szerokości w , wysokości h i lewym górnym rogu w (x, y)
- `drawOval(int x, int y, int w, int h)`
rysuje elipsę **bez wnętrza** o szerokości w , wysokości h i środka w $(x+w/2, y+h/2)$



Malarstwo w Jawie

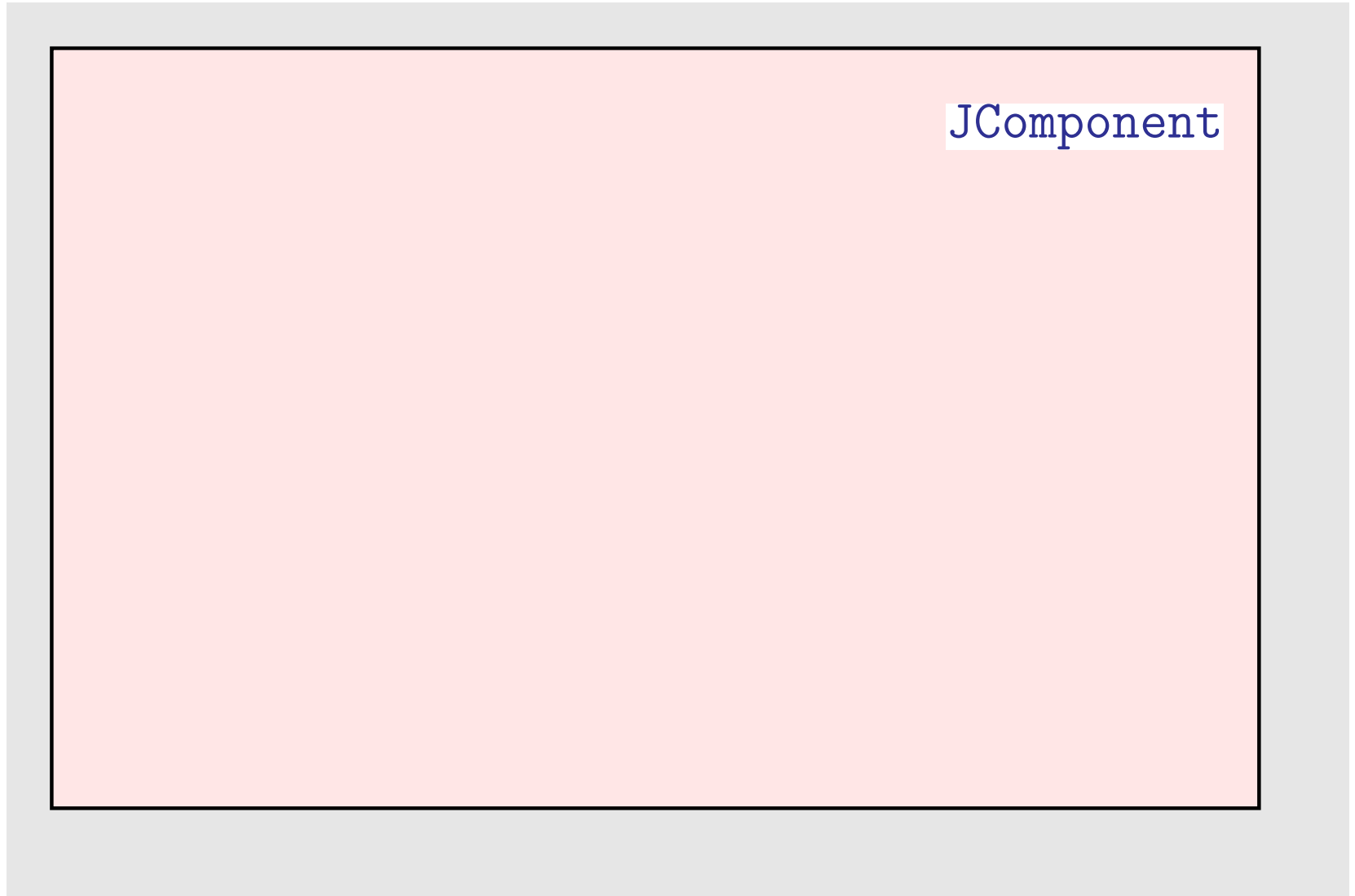
Niektóre narzędzia malarskie:

- `drawLine(int x1, int y1, int x2, int y2)`
rysuje odcinek od $(x1, y1)$ do $(x2, y2)$
- `drawRect(int x, int y, int w, int h)`
rysuje prostokąt **bez wnętrza** o szerokości w , wysokości h i lewym górnym rogu w (x, y)
- `fillRect(int x, int y, int w, int h)`
rysuje **wypełniony** prostokąt o szerokości w , wysokości h i lewym górnym rogu w (x, y)
- `drawOval(int x, int y, int w, int h)`
rysuje elipsę **bez wnętrza** o szerokości w , wysokości h i środku w $(x+w/2, y+h/2)$
- `fillOval(int x, int y, int w, int h)`
rysuje **wypełnioną** elipsę o szerokości w , wysokości h i środku w $(x+w/2, y+h/2)$



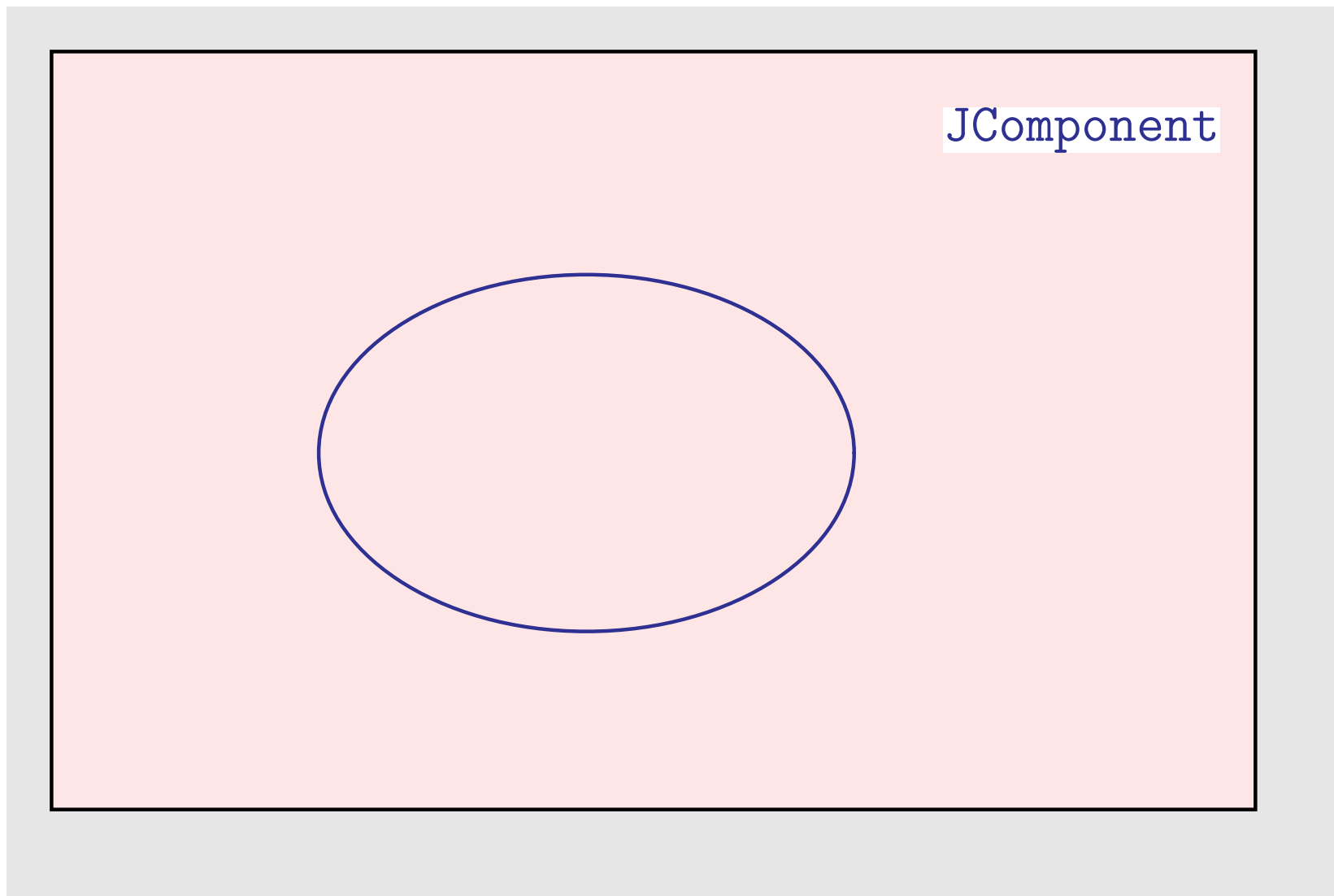
Malarstwo w Jawie

Elipsy:



Malarstwo w Jawie

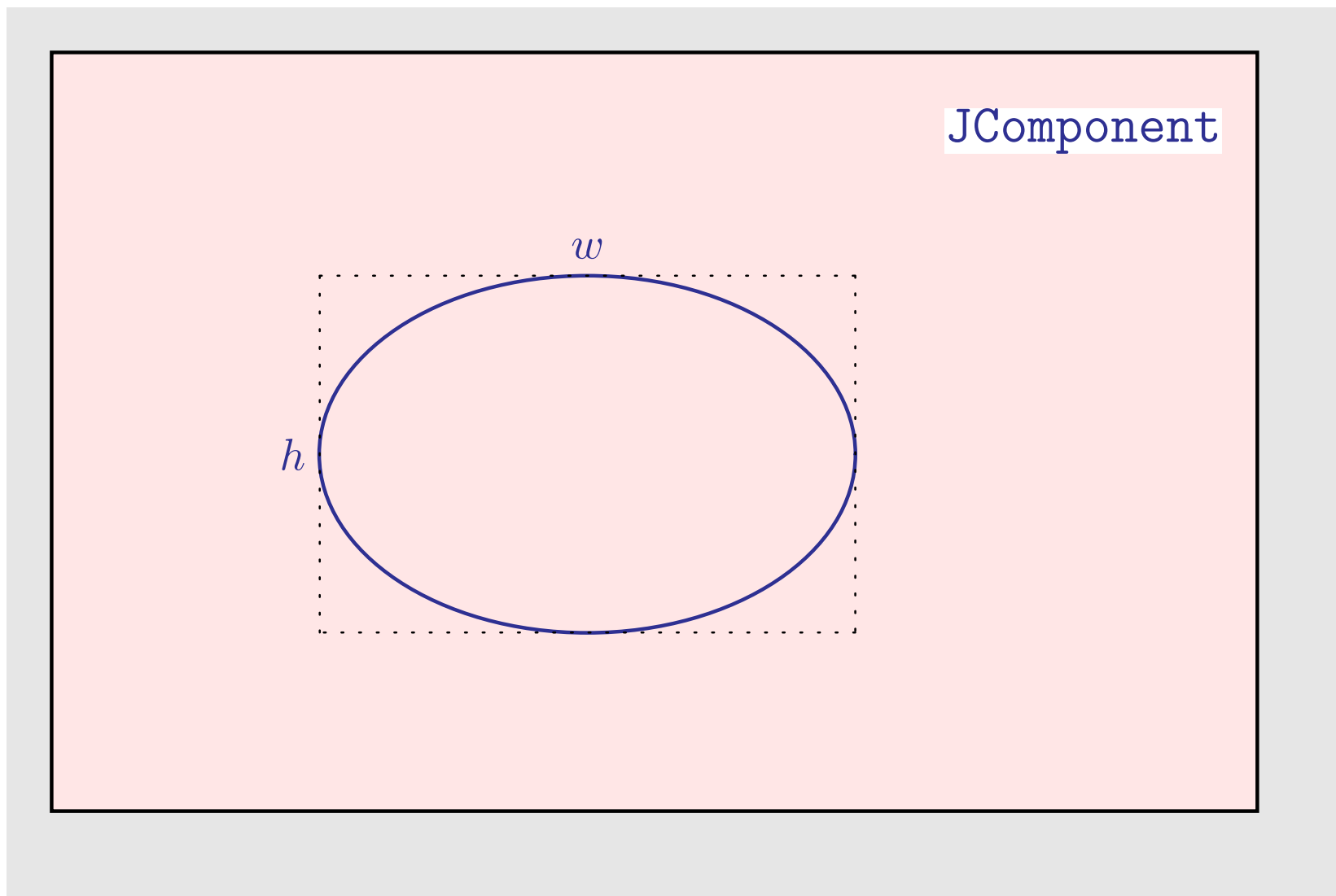
Elipsy:



```
g.drawOval(x,y,w,h);
```

Malarstwo w Jawie

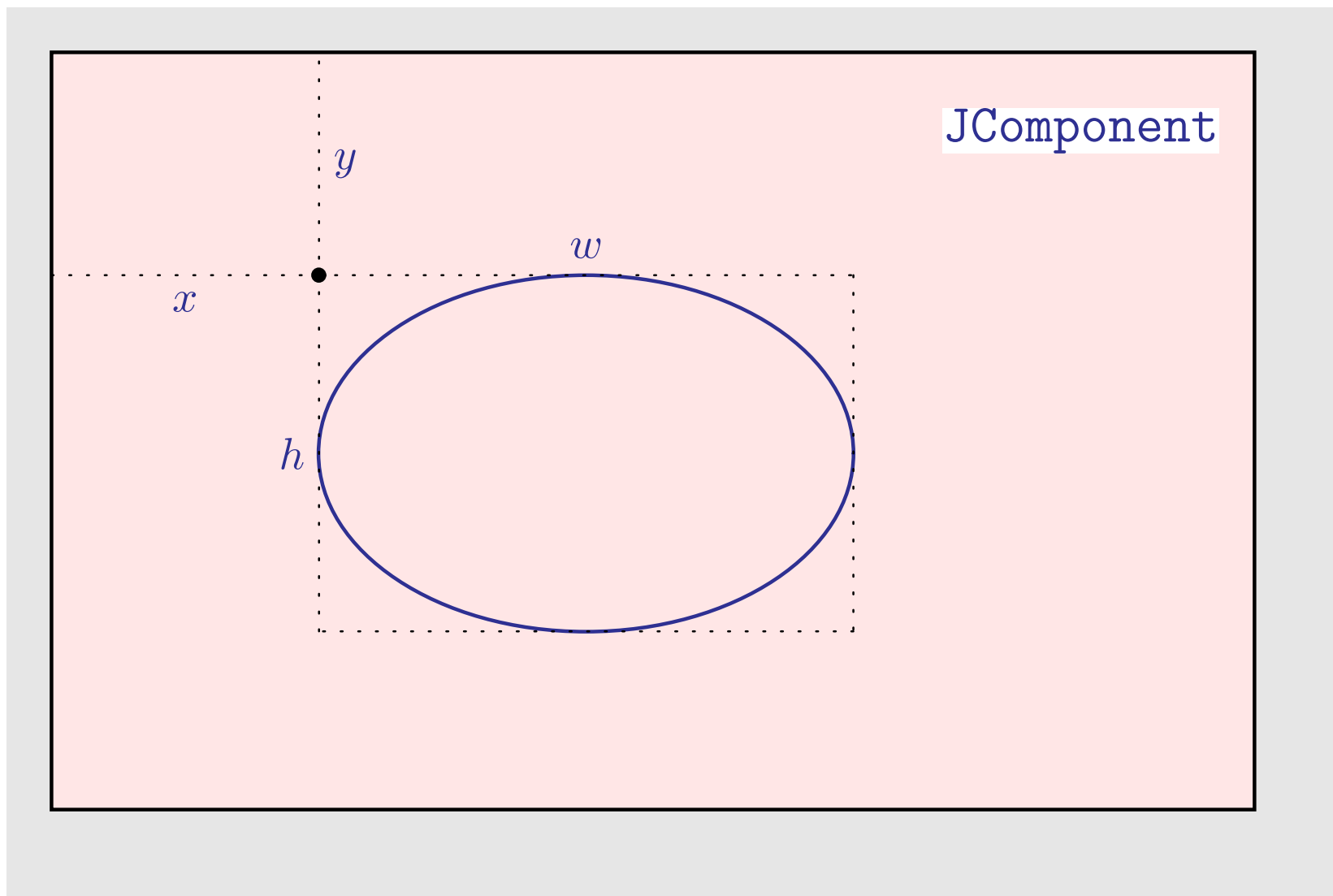
Elipsy:



```
g.drawOval( $x$ ,  $y$ ,  $w$ ,  $h$ );
```

Malarstwo w Jawie

Elipsy:



```
g.drawOval( $x$ ,  $y$ ,  $w$ ,  $h$ );
```

Malarstwo w Jawie

Niektóre narzędzia malarskie:

- `setColor(Color c)`
ustawia aktualny kolor



Malarstwo w Jawie

Niektóre narzędzia malarskie:

- `setColor(Color c)`
ustawia aktualny kolor
- `drawString(String napis, int x, int y)`
umieszcza `napis` w `(x,y)`



Malarstwo w Jawie

Niektóre narzędzia malarskie:

- `setColor(Color c)`
ustawia aktualny kolor
- `drawString(String napis, int x, int y)`
umieszcza `napis` w `(x,y)`
- `setFont(Font f)`
ustawia aktualny font



Malarstwo w Jawie

Niektóre narzędzia malarskie:

- `setColor(Color c)`
ustawia aktualny kolor
- `drawString(String napis, int x, int y)`
umieszcza `napis` w `(x,y)`
- `setFont(Font f)`
ustawia aktualny font



Szczegóły i wiele innych narzędzi — w dokumentacji; np. w

docs.oracle.com/en/java/javase/11/docs/api/java.desktop/java/awt/Graphics.html

Malarstwo w Jawie

Włączanie gotowców

Malarstwo w Javie

Włączanie gotowców:

`ImageIcon` — klasa konkretna, jej konstruktor pobiera adres pliku i tworzy obrazek

Malarstwo w Jawie

Włączanie gotowców:

`ImageIcon` — klasa konkretna, jej konstruktor pobiera adres pliku i tworzy obrazek

`g.drawImage` — umieszcza obiekt abstrakcyjnej klasy `Image`

Malarstwo w Jawie

Włączanie gotowców:

`ImageIcon` — klasa konkretna, jej konstruktor pobiera adres pliku i tworzy obrazek

`g.drawImage` — umieszcza obiekt abstrakcyjnej klasy `Image`

```
rama.add(new JComponent() {  
    public void paintComponent(Graphics g) {  
        Image fotka =  
            new ImageIcon("fotka.jpg").getImage();  
        g.drawImage(fotka, 0, 0, null);  
    }  
});
```

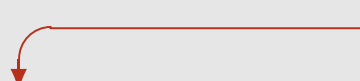
Malarstwo w Jawie

Włączanie gotowców:

`ImageIcon` — klasa konkretna, jej konstruktor pobiera adres pliku i tworzy obrazek

`g.drawImage` — umieszcza obiekt abstrakcyjnej klasy `Image`

```
rama.add(new JComponent() {  
    public void paintComponent(Graphics g) {  
        Image fotka =  
            new ImageIcon("fotka.jpg").getImage();  
        g.drawImage(fotka, 0, 0, null);  
    }  
});
```



ścieżka dyskowa do fotki

Malarstwo w Jawie

Włączanie gotowców:

`ImageIcon` — klasa konkretna, jej konstruktor pobiera adres pliku i tworzy obrazek

`g.drawImage` — umieszcza obiekt abstrakcyjnej klasy `Image`

```
rama.add(new JComponent() {  
    public void paintComponent(Graphics g) {  
        Image fotka =  
            new ImageIcon("fotka.jpg").getImage();  
        g.drawImage(fotka, 0, 0, null);  
    }  
});
```

ścieżka dyskowa do fotki

pozycja lewego górnego rogu

Malarstwo w Jawie

Włączanie gotowców:

`ImageIcon` — klasa konkretna, jej konstruktor pobiera adres pliku i tworzy obrazek

`g.drawImage` — umieszcza obiekt abstrakcyjnej klasy `Image`

```
rama.add(new JComponent() {  
    public void paintComponent(Graphics g) {  
        Image fotka =  
            new ImageIcon("fotka.jpg").getImage();  
        g.drawImage(fotka, 0, 0, null);  
    }  
});
```

ścieżka dyskowa do fotki

pozycja lewego górnego rogu

na razie nieważne...

Malarstwo w Jawie

Włączanie gotowców:

`ImageIcon` — klasa konkretna, jej konstruktor pobiera adres pliku i tworzy obrazek

`g.drawImage` — umieszcza obiekt abstrakcyjnej klasy `Image`

```
rama.add(new JComponent() {  
    public void paintComponent(Graphics g) {  
        Image fotka =  
            new ImageIcon("fotka.jpg").getImage();  
        g.drawImage(fotka, 0, 0, null);  
    }  
});
```

ścieżka dyskowa do fotki

pozycja lewego górnego rogu

na razie nieważne...

Malarstwo w Jawie

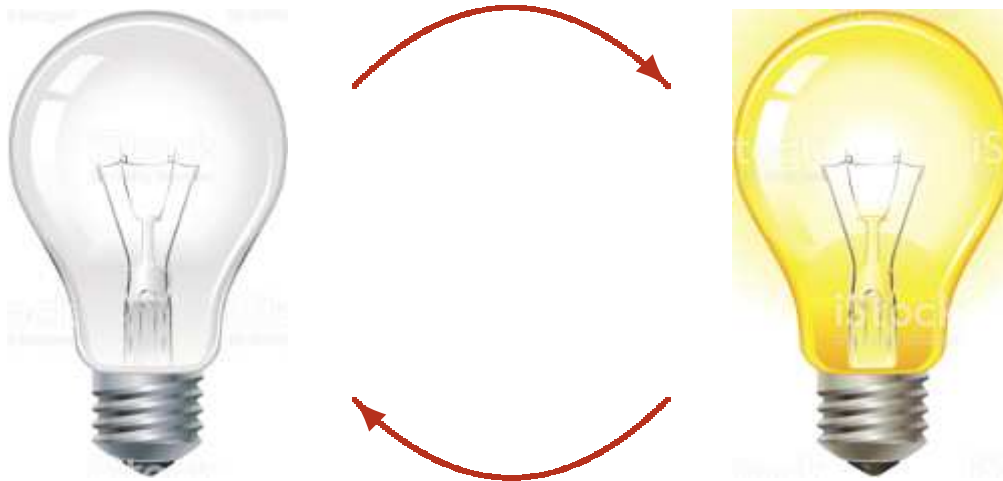
Filmy rysunkowe:

iis.ans-elblag.pl/~stefan/Dydaktyka/Prog0biII/Prog08/P5_Zarowka.java

Malarstwo w Jawie

Filmy rysunkowe:

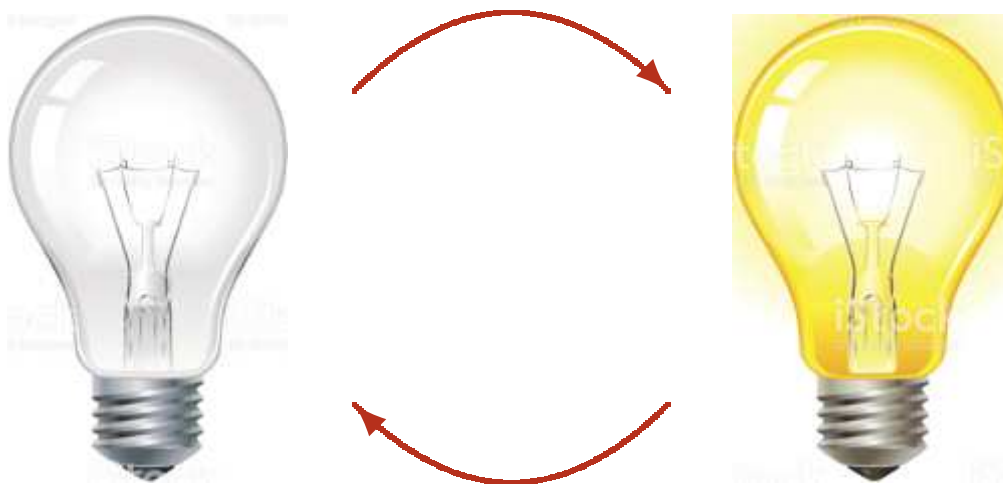
iis.ans-elblag.pl/~stefan/Dydaktyka/Prog0biII/Prog08/P5_Zarowka.java



Malarstwo w Jawie

Filmy rysunkowe:

iis.ans-elblag.pl/~stefan/Dydaktyka/Prog0biII/Prog08/P5_Zarowka.java



Jak to zrobić?

Malarstwo w Java

Filmy rysunkowe:

Do już widocznej ramki (po wykonaniu `setVisible(true)`) ciągle jeszcze można dodawać komponenty

Malarstwo w Jawie

Filmy rysunkowe:

Do już widocznej ramki (po wykonaniu `setVisible(true)`) ciągle jeszcze można dodawać komponenty; ale trzeba tą ramkę **uaktualnić**:

`validate()`

Malarstwo w Jawie

Filmy rysunkowe:

Do już widocznej ramki (po wykonaniu `setVisible(true)`) ciągle jeszcze można dodawać komponenty; ale trzeba tą ramkę **uaktualnić**:

`validate()`

Z już widocznej ramki można usunąć wszystkie komponenty

Malarstwo w Jawie

Filmy rysunkowe:

Do już widocznej ramki (po wykonaniu `setVisible(true)`) ciągle jeszcze można dodawać komponenty; ale trzeba tą ramkę **uaktualnić**:

```
validate()
```

Z już widocznej ramki można usunąć wszystkie komponenty:

```
getContentPane().removeAll()
```

Malarstwo w Jawie

Filmy rysunkowe:

Do już widocznej ramki (po wykonaniu `setVisible(true)`) ciągle jeszcze można dodawać komponenty; ale trzeba tą ramkę **uaktualnić**:

```
validate()
```

Z już widocznej ramki można usunąć wszystkie komponenty:

```
getContentPane().removeAll()
```

To daje możliwość realizowania filmików rysunkowych.

Malarstwo w Jawie

Filmy rysunkowe:

```
class Zar extends JComponent {  
    private static Boolean czyCiemna = true;  
    public void paintComponent(Graphics g) {  
        Image fotka =  
            new ImageIcon(  
                czyCiemna ? "zar_ciemna.jpg" : "zar_jasna.jpg")  
            ).getImage();  
        g.drawImage(fotka, 0, 0, null);  czyCiemna = !czyCiemna;  
    }  
}
```

Malarstwo w Jawie

Filmy rysunkowe:

```
class Zar extends JComponent {  
    private static Boolean czyCiemna = true;  
    public void paintComponent(Graphics g) {  
        Image fotka =  
            new ImageIcon(  
                "zar_"+(czyCiemna ? "ciemna" : "jasna")+ ".jpg"  
            ).getImage();  
        g.drawImage(fotka, 0, 0, null);  czyCiemna = !czyCiemna;  
    }  
}
```

← własność statyczna!

Malarstwo w Jawie

Filmy rysunkowe:

```
class Zar extends JComponent {  
    private static Boolean czyCiemna = true;  
    public void paintComponent(Graphics g) {  
        Image fotka =  
            new ImageIcon(  
                "zar_"+(czyCiemna ? "ciemna" : "jasna")+ ".jpg"  
            ).getImage();  
        g.drawImage(fotka, 0, 0, null);  czyCiemna = !czyCiemna;  
    }  
}
```

← **własność statyczna!**

wybór rysunku:
zar_ciemna.jpg
lub
zar_jasna.jpg

Malarstwo w Jawie

Filmy rysunkowe:

```
class Zar extends JComponent {  
    private static Boolean czyCiemna = true;  
    public void paintComponent(Graphics g) {  
        Image fotka =  
            new ImageIcon(  
                "zar_"+(czyCiemna ? "ciemna" : "jasna")+ ".jpg"  
            ).getImage();  
        g.drawImage(fotka, 0, 0, null); czyCiemna = !czyCiemna;  
    }  
}
```

← własność statyczna!

wybór rysunku:
zar_ciemna.jpg
lub
zar_jasna.jpg

zmiana

Malarstwo w Jawie

Filmy rysunkowe:

```
class Zar extends JComponent {
    private static Boolean czyCiemna = true;
    public void paintComponent(Graphics g) {
        Image fotka =
            new ImageIcon(
                "zar_"+(czyCiemna ? "ciemna" : "jasna")+ ".jpg"
            ).getImage();
        g.drawImage(fotka, 0, 0, null);
        czyCiemna = !czyCiemna;
    }
}
```

własność statyczna!

wybór rysunku:
zar_ciemna.jpg
lub
zar_jasna.jpg

zmiana

```
. . . . .
while (true) {
    rama.add(new Zar());
    rama.validate(); Thread.sleep(500);
    rama.getContentPane().removeAll();
}
. . . . .
```

nic nie robić
przez 500 milisekund