Faculty of Physics

WARSAW UNIVERSITY OF TECHNOLOGY

Programowanie Obiektowe Java

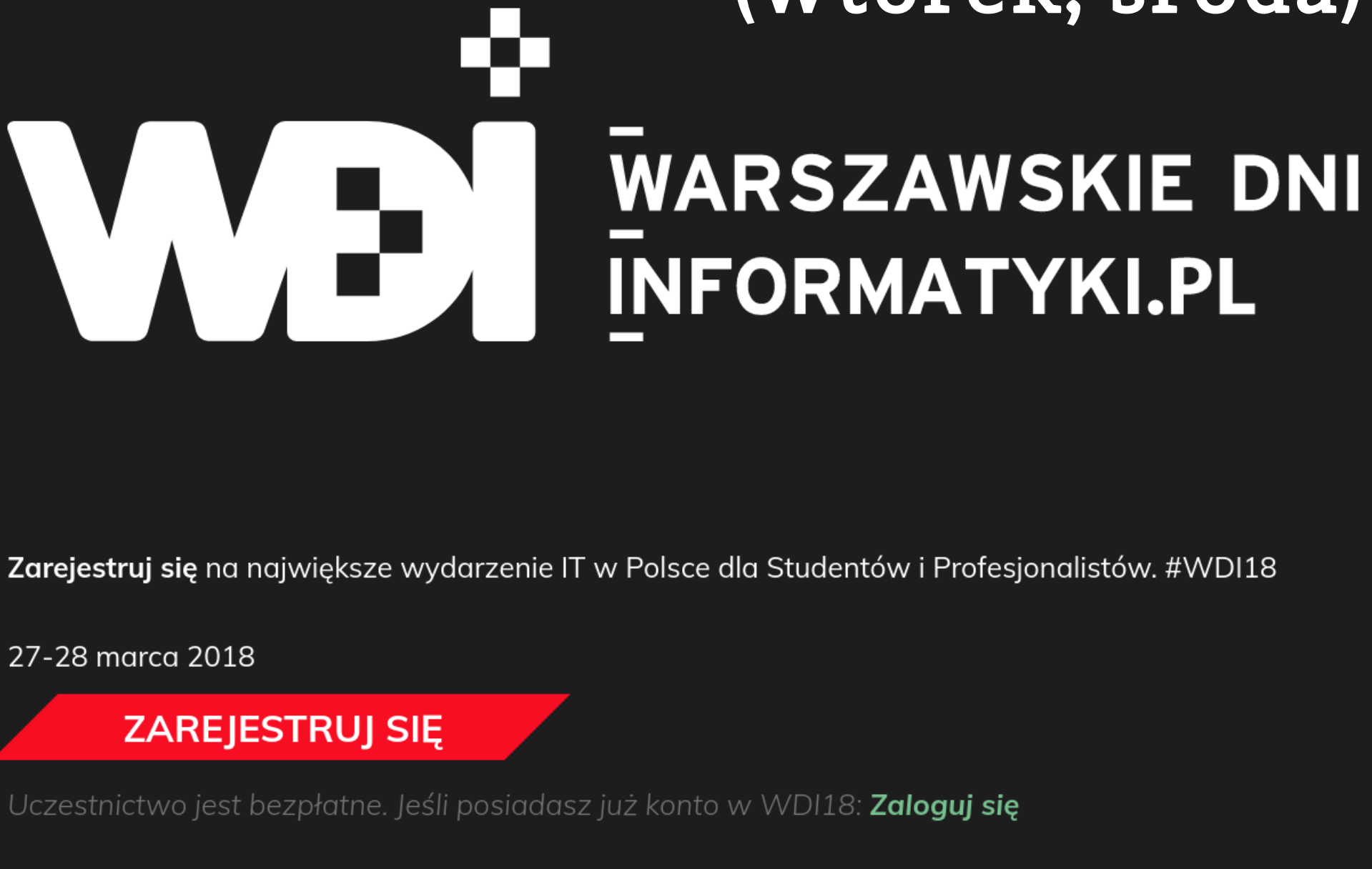
Małgorzata Janik

Zakład Fizyki Jądrowej
malgorzata.janik@pw.edu.pl
http://java.fizyka.pw.edu.pl/

WDI – 27-28 marca

https://www.warszawskiedniinformatyki.pl/

(wtorek, środa)



Wyjątki
czyli jak sobie radzić z nagłymi problemami

Obsługa błędów za pomocą wyjątków

- Najlepiej oczywiście wyłapać wszystkie możliwe błędy jeszcze przed uruchomieniem programu
  - Nie zawsze jednak jest to takie proste
- Pozostałe błędy trzeba obsłużyć podczas wykonywania programu, przekazując informację o wystąpieniu problemu do odbiorcy, który będzie wiedział co zrobić z zainstniałym problemem
  - W starszych językach np. wyrzucano na ekran odpowiedni status błędu.
    - To niestety jest bardzo trudne w implementacji – trzeba przewidzieć KAŻDY MOŻLIWY PROBLEM.
  - Nowoczesnym sposobem jest *obsługa sytuacji wyjątkowych* (ang. *exception handling*)

Krótka powtórka

Obsługa wyjątków

Sytuacje wyjątkowe – sytuacje, jakie mogą mieć miejsce, choć nie muszą (np. kiedy funkcja nie może się z jakiś powodów wykonać się poprawnie). Poprawnie napisany program "przewidzi", co ma wydarzyć się w przypadku ich wystąpienia. Sytuacją wyjątkową może być wszystko, co za takową zostanie uznane.

Jak napisać poprawnie program z obsługą sytuacji wyjątkowych?

1) Określić, kiedy zaczyna się obszar spodziewanego wystąpienia sytuacji wyjątkowej (blok *try*)
2) W przypadku wystąpienia sytuacji wyjątkowej należy zasygnalizować jej wystąpienie (instrukcja *throw*)
3) Reakcja na wystąpienie sytuacji wyjątkowej (blok *catch*)

Po co są wyjątki?

… wykorzystywane w wielu sytuacjach...

… kiedy dwa odmienne fragmenty kodu mają ze sobą współpracować (jeden wykrywa sytuację wyjątkową – funkcja biblioteczna, drugi ją obsługuje..)

3

H. Zbroszczyk, *Języki Programowania, Wykład 13*
http://www.if.pw.edu.pl/~gos/students/jp/PO/wyklad13-2017.pdf

Sytuacje wyjątkowe

- Sytuacje, które mogą mieć miejsce, chociaż nie muszą
  - np. kiedy funkcja nie może z jakiegoś powodu wykonać się poprawnie
- Poprawnie napisany program powinien **przewidzieć ich wystąpienie** i zaradzić na taką sytuację
- Sytuacją wyjątkową może być wszystko co za takową zostanie uznane
  - zwykle jest to problem, który wstrzymuje wykonywanie metody lub bloku

Jak napisać poprawnie program z obsługą sytuacji wyjątkowych?

1) Określić, gdzie zaczyna się obszar spodziewanego wystąpienia sytuacji wyjątkowej
   - Blok **try**

2) W przypadku wystąpienia sytuacji wyjątkowej należy zasygnalizować jej wystąpienie
   - Instrukcja **throw**

3) Reakcja na wystąpienie sytuaciji wyjątkowej
   - Blok **catch**

Co może pójść nie tak?

```
public class DzieleniePrzezZero {

public static void main(String[] args) {

int licznik;
int mianownik;

System.out.println("Podaj dwie liczby:");
Scanner sc = new Scanner(System.in);


licznik = sc.nextInt();
mianownik = sc.nextInt();

int wynik = licznik/mianownik;

System.out.println("Wynikiem dzielenia jest: "+wynik);


sc.close();
}

}
```

Wczytywanie liczb z klawiatury od użytkownika

Co może pójść nie tak?

```
public class DzieleniePrzezZero {

public static void main(String[] args) {

int licznik;
int mianownik;

System.out.println("Podaj dwie liczby:");
Scanner sc = new Scanner(System.in);


licznik = sc.nextInt();
mianownik = sc.nextInt();

int wynik = licznik/mianownik;

System.out.println("Wynikiem dzielenia jest: "+wynik);


sc.close();
}

}
```

Wczytywanie liczb z klawiatury od użytkownika

A JEŚLI PODZIELI PRZEZ 0???!!!

Spróbujmy się zabezpieczyć

Co może pójść nie tak?

```
public class DzieleniePrzezZero {

public static void main(String[] args) {

int licznik;
int mianownik;

System.out.println("Podaj dwie liczby:");
Scanner sc = new Scanner(System.in);

licznik = sc.nextInt();
mianownik = sc.nextInt();

int wynik = licznik/mianownik;

System.out.println("Wynikiem dzielenia jest: "+wynik);

sc.close();
}

}
```

Określamy obraszar spodziewanej sytuacji wyjątkowej i wkładamy go w **blok try**

Co może pójść nie tak?

```
public class DzieleniePrzezZero {

public static void main(String[] args) {

int licznik;
int mianownik;

System.out.println("Podaj dwie liczby:");
Scanner sc = new Scanner(System.in);

try{
    licznik = sc.nextInt();
    mianownik = sc.nextInt();

    int wynik = licznik/mianownik;

    System.out.println("Wynikiem dzielenia jest: "+wynik);
}

sc.close();
}

}
```

Określamy obraszar spodziewanej sytuacji wyjątkowej i wkładamy go w **blok try**

Co może pójść nie tak?

```
public class DzieleniePrzezZero {

public static void main(String[] args) {

int licznik;
int mianownik;

System.out.println("Podaj dwie liczby:");
Scanner sc = new Scanner(System.in);

try{
    licznik = sc.nextInt();
    mianownik = sc.nextInt();

    if(mianownik==0) throw new ArithmeticException();
    int wynik = licznik/mianownik;
    System.out.println("Wynikiem dzielenia jest: "+wynik);
}
catch(ArithmeticException e){
    System.out.println("Nie mogę podzielić przez zero! Podaj inną liczbę.");
}

sc.close();
}
```

Sygnalizujemy problem: wyrzucenie wyjątku (**instrukcja throw**)

Co może pójść nie tak?

```
public class DzieleniePrzezZero {

public static void main(String[] args) {

int licznik;
int mianownik;

System.out.println("Podaj dwie liczby:");
Scanner sc = new Scanner(System.in);

try{
    licznik = sc.nextInt();
    mianownik = sc.nextInt();

    if(mianownik==0) throw new ArithmeticException();
    int wynik = licznik/mianownik;
    System.out.println("Wynikiem dzielenia jest: "+wynik);
}
catch(ArithmeticException e){
    System.out.println("Nie mogę podzielić przez zero! Podaj inną liczbę.");
}

sc.close();
}
```

Złapanie wyjątku i reakcja na wystąpienie sytuacji wyjątkowej (**blok catch**)

Jak napisać poprawnie program z obsługą sytuacji wyjątkowych?

1) Określić, gdzie zaczyna się obszar spodziewanego wystąpienia sytuacji wyjątkowej

- Blok **try**

```
try{
    …
}
```

2) W przypadku wystąpienia sytuacji wyjątkowej należy zasygnalizować jej wystąpienie

- Instrukcja **throw**

```
throw new NazwaWyjatkuException();
```

3) Reakcja na wystąpienie sytuaciji wyjątkowej (wyłapanie wyjątku)

- Blok **catch**

```
catch(NazwaWyjatkuException e)
{
    …
}
```

Chwila, chwila

- Chyba już gdzieś widzieliśmy coś, co wygląda jak wyjątki, chociaż żadnych try-catch w kodzie nie było....

```
Exception in thread "main" java.util.InputMismatchException
at java.util.Scanner.throwFor(Scanner.java:864)
at java.util.Scanner.next(Scanner.java:1485)
at java.util.Scanner.nextInt(Scanner.java:2117)
at java.util.Scanner.nextInt(Scanner.java:2076)
at wyk4.DzieleniePrzezZero.main(DzieleniePrzezZero.java:17)
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
at wyk4.DzieleniePrzezZero.main(DzieleniePrzezZero.java:20)
```

```
Exception in thread "main" java.lang.NullPointerException
at wyk4.DzieleniePrzezZero.jakasFunkcja(DzieleniePrzezZero.java:9)
at wyk4.DzieleniePrzezZero.main(DzieleniePrzezZero.java:33)
```

Wyjątki zgłaszane przez Javę

- Wyjątki mogą być zgłaszane zarówno przez programistę (throw), jak i przez wirtualną maszynę Javy.
- Java w wielu przypadkach sama wyrzuca wyjątek gdy napotka problem, więc często możemy pominąć instrukcję throw.
- Jednakże, jeśli takiego wyjątku nie obsłużymy, to cały progam się wykrzaczy z błędami w konsoli:

```
Exception in thread "main" java.lang.NullPointerException
at wyk4.DzieleniePrzezZero.jakasFunkcja(DzieleniePrzezZero.java:9)
at wyk4.DzieleniePrzezZero.main(DzieleniePrzezZero.java:33)
```

- Żeby temu zapobiec, należy w kodzie umieścić instrukcje wyłapujące takie wyjątki (czyli **blok try-catch**).
  - Bądź zapobiec sytuacji która powoduje generowanie wyjątku.

Wyjątki zgłaszane przez Javę

```
public class DzieleniePrzezZero {

public static void main(String[] args) {

int licznik;
int mianownik;

System.out.println("Podaj dwie liczby:");
Scanner sc = new Scanner(System.in);

try{
    licznik = sc.nextInt();
    mianownik = sc.nextInt();

    if(mianownik==0) throw new ArithmeticException();
    int wynik = licznik/mianownik;
    System.out.println("Wynikiem dzielenia jest: "+wynik);
}
catch(ArithmeticException e){
    System.out.println("Nie mogę podzielić przez zero! Podaj inną liczbę.");
}

sc.close();
}
```

Java sama sobie wyrzuci wyjątek w momencie dzielenia przez zero

Obsługa kilku wyjątków jednocześnie

```
try {
  //kod który może zgłosić wyjątki
}
catch (TypWyjatku1 w) {
  //obsługa wyjątków Typ1
}
catch (TypWyjatku2 w) {
  //obsługa wyjątków Typ2
}
catch (TypWyjatku3 w) {
  //obsługa wyjątków Typ3
}
finally{
  //instrukcje – wykonane niezależnie
  od tego czy wyjątek wystąpił, czy nie...
}
```

Obsługa kilku wyjątków jednocześnie

```
try {
  //kod który może zgłosić wyjątki
}
catch (TypWyjatku1 w) {
  //obsługa wyjątków Typ1
}
catch (TypWyjatku2 w) {
  //obsługa wyjątków Typ2
}
catch (TypWyjatku3 w) {
  //obsługa wyjątków Typ3
}
finally{
  //instrukcje – wykonane niezależnie
  od tego czy wyjątek wystąpił, czy nie...
}
```

finally wykona się zawsze:
tutaj zwykle będziemy sprzatątać
(zamykać otwarte połączenia, itp.).

Obsługa kilku wyjątków jednocześnie

```
public class KilkaWyjatkow {
   public static void main(String[] args) {
     int[] licznik = {1, 2, 3 , 0 };
     int mianownik[] = {2, 0, 1, 0, 5};
     double ulamek = 0.0;

     for (int i=0; i<6; i++){
         try{
            ulamek = (double) ( licznik[i]/mianownik[i] );
         }
         catch (IndexOutOfBoundsException e) {
             System.out.println("Indeks tablicy poza zakresem");
         }
         catch (ArithmeticException e) {
             System.out.println("Proba dzielenia przez zero");
         }
         finally {
             System.out.println("Blok finally wykonany zawsze");
         System.out.println("wartosc zmiennej ulamek: " + ulamek);
         }
      } //koniec petli for
    } // koniec metody main()
}
```

Jednoczesne przechwytywanie kilku wyjątków jednym blokiem catch

```
try {
  //kod który może zgłosić wyjątki
}
catch (TypWyjatku1 | TypWyjatku2 w) {
  //obsługa wyjątków Typ1 lub Typ2
}
```

Jednoczesne przechwytywanie kilku wyjątków jednym blokiem catch

```
public class KilkaWyjatkow2 {
   public static void main(String[] args) {
     int[] licznik = {1, 2, 3 , 0 };
     int mianownik[] = {2, 0, 1, 0, 5};
     double ulamek = 0.0;

     for (int i=0; i<6; i++){
         try{
            ulamek = (double)(licznik[i]/mianownik[i]);

         }
         catch (IndexOutOfBoundsException | ArithmeticException e) {
             System.out.println("Przechwycony wyjatek: " +
e.getClass().getName());
         }

      } //koniec petli for
    } // koniec metody main()
}
```

Zagnieżdżanie bloków try-catch

- Instrukcja try może występować w bloku instrukcji innej instrukcji try.
- Konstrukcja taka powoduje, że wyjątki zgłaszane przez wewnętrzy blok try będą posiadały swój kontekst wywołania, inny niż wyjątki bloku zewnętrznego.
- Jeżeli wewnętrzny blok try zgłosi wyjątek, dla którego nie posiada odp. sekcji catch, będzie przeszukiwany kontekst bloku zewnętrznego w poszukiwaniu odp. sekcji catch.

```
public class ZagniezdzanieWyjatkow1{
    public static void main(String[] args) {

        int[] licznik = {1, 2, 3 , 0 };
        int mianownik[] = {2, 0, 1, 0, 5};
        double ulamek = 0.0;

        try{
        for (int i=0; i<6; i++){
            int l=0, m =0;
            try{
               l = licznik[i];
               m = mianownik[i];
            }
            catch (IndexOutOfBoundsException e) {
            System.out.println("Indeks tablicy poza zakresem");
                 System.out.println("Wyjątek z bloku wewnetrznego - przejscie do
                                                    kolejnej iteracji petli");
            }
            ulamek = (double) (l/m); // zglosi wyjatek zewnetrzny
            //ulamek = (double) l / m;  // nie zglosi wyjatku zewn
            System.out.println(ulamek);
             } //koniec petli for
        }
        catch(ArithmeticException e){
        System.out.println("Proba dzielenia przez zero");
        System.out.println("Wyjątek z bloku zewnetrznego - koniec petli");
        }
    }
}
```

Łapanie wyjątków

- obsługa wyjątku **nie musi** nastąpić w tej samej funkcji, w której wyjatek został wyrzucony
  - cała idea wyjątków opiera się na pomyśle, że **nie potrafimy rozwiązać problemów w danym kontekście więc delegujemy go** dalej (do kodu, który go złapie – procedury obsługi wyjątku)
  - tak jak w przykładzie z zagnieżdżaniem, możemy go np. wysłać do zewnętrznego bloku try
- Jak przechwycić wyjątki w innej funkcji?

throws

Jeśli metoda zgłasza wyjątek, którego sama nie obsługuje, to deklaracja metody musi zawierać informację o tym.

Służy do tego słowo kluczowe **throws**, **umieszczane po** deklaracji metody, a po nim wymieniane są typy wszystkich wyjątków zgłaszanych przez metodę (za wyjątkiem **Error i RuntimeException i ich podklas**).

Ogólna postać definicji metody zgłaszającej nieobsługiwane wyjątki :

typ nazwa_metody(lista-parametrów) **throws** ***lista-wyjątków***

{

// ciało metody

}

throws

```
public static void Funkcja() throws ArithmeticException
{
int licznik;
int mianownik;

System.out.println("Podaj dwie liczby:");
Scanner sc = new Scanner(System.in);
licznik = sc.nextInt();
mianownik = sc.nextInt();

if(mianownik==0) throw new ArithmeticException();
int wynik = licznik / mianownik;

System.out.println("Wynikiem dzielenia jest: " + wynik);

sc.close();

}
```

Te wyjątki zostaną wyrzucone z funkcji i trzeba je złapać gdzieś indziej

throws

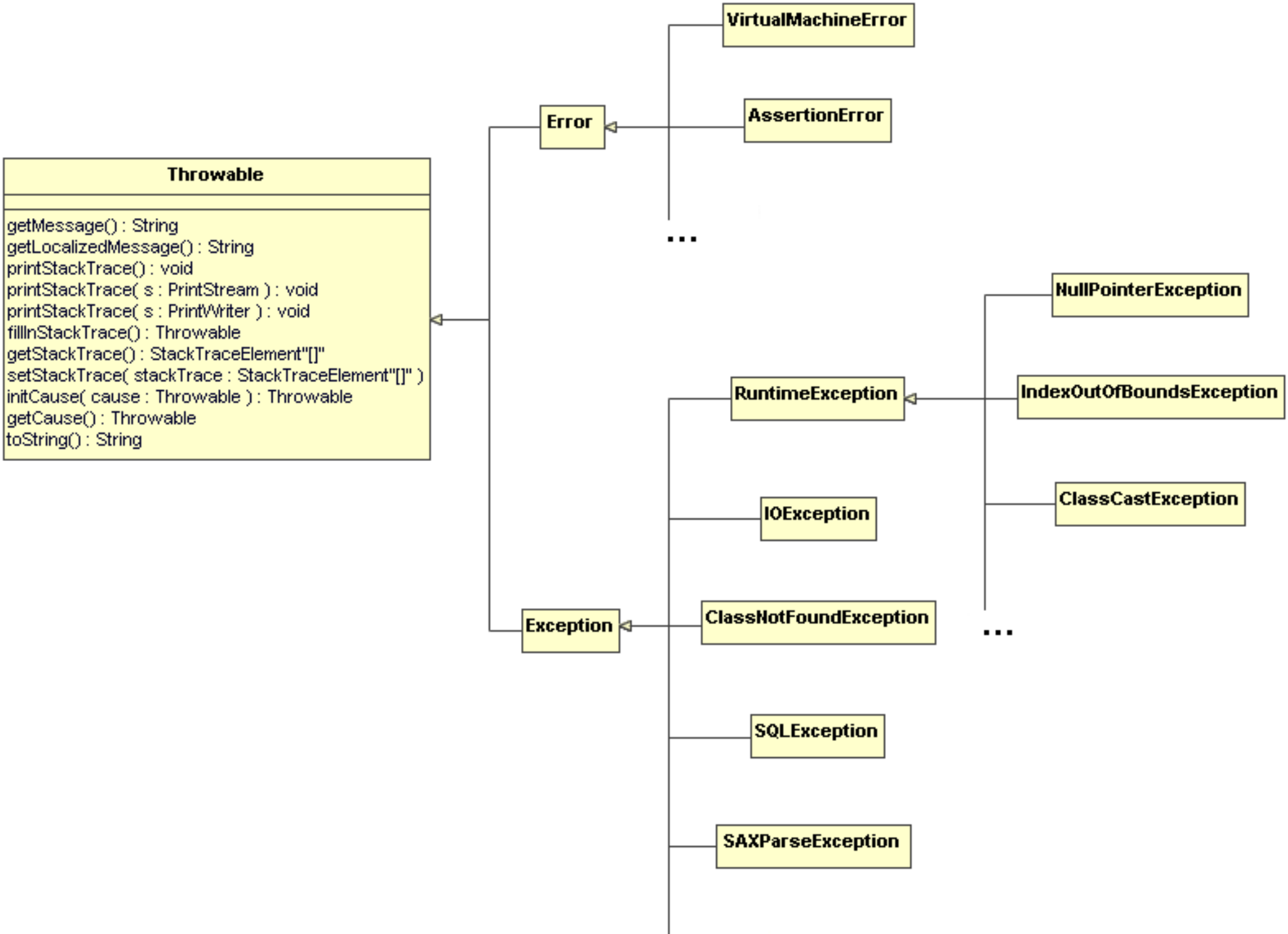
```
public static void Funkcja() throws ArithmeticException
{
int licznik;
int mianownik;

System.out.println("Podaj dwie liczby:");
Scanner sc = new Scanner(System.in);
licznik = sc.nextInt();
mianownik = sc.nextInt();

if(mianownik==0) throw new ArithmeticException();
int wynik = licznik / mianownik;

System.out.println("Wynikiem dzielenia jest: " + wynik);

sc.c

}
```

Trafiają do miejsca, gdzie funkcja została wywołana....

```
public static void main(String[] args) {
    try{
        Funkcja();
    }
    catch(ArithmeticException e){
        System.out.println("Nie mogę podzielić przez zero! Podaj inną
            liczbę.");
    }
}
```

Czym właściwie są wyjątki?

- Wyjątek w Java jest **obiektem**, który opisuje sytuację błędną powstałą w kodzie.
- Wszystkie wyjątki i błędy są podklasami klasy **Throwable**.
- Podklasami klasy Throwable są:
  - Klasa **Exception** i potomne; służą do opisywania sytuacji błędnych, które mogą być spowodowane przez kod użytkownika lub mogą być przez kod użytkownika wykryte i obsłużone.
  - Klasa **Error** i potomne; są używane przez maszynę wirtualną do zgłaszania *błędów fatalnych, takich jak: OutOfMemoryError, VirtualMachineError, LinkageError...*

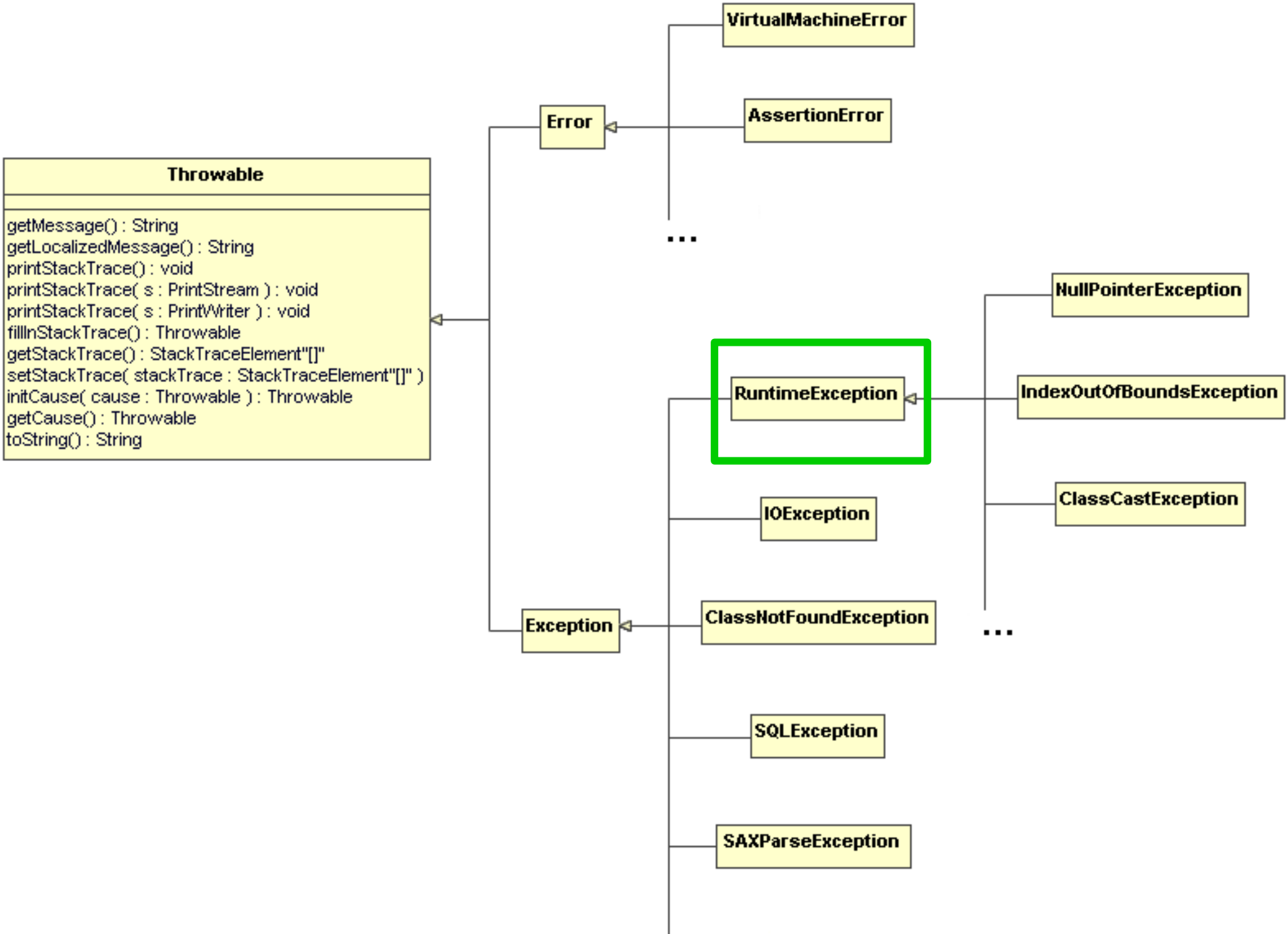

Throwable
getMessage() : String
getLocalizedMessage() : String
printStackTrace() : void
printStackTrace(s : PrintStream) : void
printStackTrace(s : PrintWriter) : void
fillInStackTrace() : Throwable
getStackTrace() : StackTraceElement"[]"
setStackTrace(stackTrace : StackTraceElement"[]")
initCause(cause : Throwable) : Throwable
getCause() : Throwable
toString() : String
Error
VirtualMachineError
AssertionError
...
Exception
RuntimeException
NullPointerException
IndexOutOfBoundsException
ClassCastException
...
IOException
ClassNotFoundException
SQLException
SAXParseException

Błędy - Error

- Wyjątki dziedziczące po **Error** reprezentują **poważne problemy,** których aplikacja **nie będzie w stanie rozwiązać.**
- Przykładową podklasą jest *VirtualMachineError*.
  - Wystąpienie takiego wyjątku oznacza, że maszyna wirtualna nie może dalej kontynuować pracy, np. z powodu wyczerpania się zasobów systemowych.
- **Wyjątków rozszerzających *Error* nie należy przechwytywać**, gdyż nie ma możliwości zaradzenia sytuacjom wyjątkowym, które je spowodowały. Z założenia te wyjątki mogą wystąpić praktycznie w każdej instrukcji kodu i nie muszą być wymieniane w klauzulach throws.

Wyjątki - Exceptions

- Wyjątki dziedziczące po Exception reprezentują sytuacje, na które dobrze napisana aplikacja powinna być przygotowana.
- To właśnie tę klasę rozszerza się tworząc własne rodzaje wyjątków.
- Jej przykładowe podklasy to:
  - **IOException** - sytuacje wyjątkowe związane z wejściem/wyjściem,
  - **ClassNotFoundException** - maszyna wirtualna nie odnalazła klasy o nazwie podanej w opisie wyjątku,
  - **SQLException** - wyjątki związane z dostępem do bazy danych
  - **SAXParseException**, która wskazuje, że podczas parsowania dokumentu XML wystąpił błąd.


Throwable
getMessage() : String
getLocalizedMessage() : String
printStackTrace() : void
printStackTrace(s : PrintStream) : void
printStackTrace(s : PrintWriter) : void
fillInStackTrace() : Throwable
getStackTrace() : StackTraceElement"[]"
setStackTrace(stackTrace : StackTraceElement"[]")
initCause(cause : Throwable) : Throwable
getCause() : Throwable
toString() : String
Error
VirtualMachineError
AssertionError
...
Exception
RuntimeException
NullPointerException
IndexOutOfBoundsException
ClassCastException
...
IOException
ClassNotFoundException
SQLException
SAXParseException

Klasa RuntimeException

- Bardzo ciekawą podklasą Exception jest RuntimeException, która sama posiada wiele podklas.
- **Takie wyjątki są zgłaszane automatycznie przez Javę i nie trzeba włączać ich w specyfikacji wyjątków.**
- Wyjątki rozszerzające RuntimeException mogą wystąpić podczas typowych operacji, jak rzutowanie zmiennej, odwołanie się do elementu tablicy lub odwołanie się do składowej obiektu.
- Ich wystąpienie zazwyczaj oznacza, że programista popełnił błąd w swoim kodzie lub nieumiejętnie korzystał z kodu napisanego przez innych.

Klasa RuntimeException

Przykładowymi podklasami RuntimeException są:

- **ClassCastException** - oznacza próbę rzutowania zmiennej na niepasujący typ,
- **IndexOutOfBoundsException** - oznacza odwołanie się do indeksu z poza zakresu
- **NullPointerException** - oznacza że zamiast referencji wskazującej na obiekt pojawiła się wartość null (np. obiekt nie utworzony)
- **IllegalArgumentException** - oznacza, że do metody przekazany został niewłaściwy argument

Klasa RuntimeException

Wyjątki RuntimeException oznaczają zwykle błąd programisty, dlatego praktycznie nigdy nie przechwytuje się takich wyjątków.

- **Co się wtedy dzieje z takim wyjątkiem?**
  - Przedostaje się przez wszystkie wywołania na zewnątrz (jak każdy inny wyjątek) aż trafi do metody main(). Jeśli tam też nie zostanie złapany, to przed wyjściem z programu wywoływana jest dla niego metoda printStackTrace();
- **Jeśli zobaczymy taki wyjątek, to zwykle znaczy, że musimy coś poprawić w naszym kodzie.**

Klasa RuntimeException

Wyjątki Run
programisty
przechwytuj

```
Exception in thread "main"
java.util.InputMismatchException
at java.util.Scanner.throwFor(Scanner.java:864)
at java.util.Scanner.next(Scanner.java:1485)
at java.util.Scanner.nextInt(Scanner.java:2117)
at java.util.Scanner.nextInt(Scanner.java:2076)
at
wyk4.DzieleniePrzezZero.main(DzieleniePrzezZero.java:17)
```

- **Co się wtedy dzieje z takim wyjątkiem?**
  - Przedostaje się przez wszystkie wywołania na zewnątrz (jak każdy inny wyjątek) aż trafi do metody main(). Jeśli tam też nie zostanie złapany, to przed wyjściem z programu wywoływana jest dla niego metoda printStackTrace();
- **Jeśli zobaczymy taki wyjątek, to zwykle znaczy, że musimy coś poprawić w naszym kodzie.**

Podsumujmy...

- Error
  - Błąd maszyny wirtualnej Javy, pojawia się automatycznie
  - Zwykle: nic nie możemy poradzić, więc nic nie robimy
- RuntimeException
  - Wyjątek obsługiwany przez Javę (automatycznie rzucany i przekazywany)
  - Zwykle: wynika z naszego błędu, więc musimy odpowiednio poprawić kod
- Pozostałe Exception
  - Sami musimy zająć się zgłaszaniem i łapaniem odpowiednich wyjątków (wczytywanie plików, bazy danych...)

Co ciekawego siedzi w obiekcie wyjątku?

Wiemy już, że jest tam m.in..

- **printStackTrace()** - czyli mamy dostęp do informacji skąd dokładnie pochodzi nasz wyjątek.

Warto wiedzieć, że możemy przenosić w wyjątku także wiadomość / tekst:

- Jeśli w konstruktorze wyjątku ustawimy wiadomość:

```
if(mianownik==0) throw new ArithmeticException("Mianownik = 0");
```

- To taką wiadomość możemy potem odczytać w miejscu obsługującym wyjątek:

```
catch( ArithmeticException | IndexOutOfBoundsException e)
{
    System.out.println(e.getMessage()+". Podaj inną liczbę.");
}
```

Na ekranie pojawi się:
Mianownik =0. Podaj inną liczbę.

Tworzenie własnych wyjątków

Java posiada wbudowane wyjątki obsługujące najczęściej spotykane błędy. Jednak często zachodzi potrzeba zdefiniowania nowych wyjątków specyficznych dla naszego programu.

Aby utworzyć nową klasę wyjątku należy zdefiniować klasę dziedziczącą po klasie **Exception.** Nowo zdefiniowana klasa nie musi nawet niczego implementować.

Najczęściej implementowane zmiany w klasach nowych wyjątków, to:
- dodatkowe zmienne instancyjne przechowujące stan sytuacji błędnej
- pokrywanie standardowych metod klasy **Throwable takich, jak: getLocalizedMessage(), getMessage(), printStackTrace() i toString()**

Przykład: WlasnyWyjatek.java

```
//deklaracja klasy wlasnego wyjatku
class LiczbaNieparzystaException extends Exception{
    int n;
    LiczbaNieparzystaException(int liczba){
        n = liczba;
    }
    public String toString(){
        return "Wyjątek! Liczba " + n + " jest nieparzysta";
    }
}
```

Przykład: WlasnyWyjatek.java

```
//deklaracja klasy wlasnego wyjatku
class LiczbaNieparzystaException extends Exception{
    int n;
    LiczbaNieparzystaException(int liczba){
        n = liczba;
    }
    public String toString(){
        return "Wyjątek! Liczba " + n + " jest nieparzysta";
    }
}

public class WlasnyWyjatek {

    //deklaracja metody zglaszajacej wyjatek
    static void sprawdzParzystosc(int liczba) throws LiczbaNieparzystaException{
        if (liczba %2 != 0 )
            throw new LiczbaNieparzystaException(liczba);
    }

    public static void main(String[] args) {

        for (int i = 1; i<10; i++){
            try {
                sprawdzParzystosc(i);
            } catch (LiczbaNieparzystaException e) {
                System.out.println(e);
            }
        }
    }
}
```

Tworzenie własnych wyjątków

```
public class WlasnyWyjatek {

    //deklaracja metody zglaszajacej wyjatek
     static void sprawdzParzystosc(int liczba) throws LiczbaNieparzystaExceptio{
        if (liczba %2 != 0 )
            throw new LiczbaNieparzystaException(liczba);
    }

    public static void main(String[] args) throws LiczbaNieparzystaException {

        for (int i = 1; i<10; i++){

            sprawdzParzystosc(i);

        }
    }
}
```

W tym przykładzie, w przypadku wystąpienia wyjątku w metodzie sprawdzParzystosc() zostanie on przekazany do metody main(), a co za tym idzie obsługa będzie przez JVM (koniec programu)

Podsumowanie

- Wyjątek w Java jest obiektem, który opisuje sytuację błędną powstałą w kodzie.
- Zaistnienie sytuacji błędnej w metodzie powoduje utworzenie obiektu reprezentującego wyjątek i zgłoszenie go przez metodę, w której błąd wystąpił. Następnie metoda może sama obsłużyć wyjątek lub przesłać go do obsługi przez inne metody/obiekty.
- Wyjątki mogą być zgłaszane przez maszynę wirtualną Javy lub przez kod użytkownika. Wyjątki zgłaszane przez maszynę wirtualną są związane z tzw. *błędami fatalnymi*, natomiast zgłaszane przez użytkownika z błędami związanymi z logiką programu.
- Składnia programu obsługującego wyjątki bazuje na pięciu słowach kluczowych: **try, catch, throw, throws i finally.**

Instrukcja throw

Służy do zgłaszania wyjątków przez nasz program:

throw WYJĄTEK;

Wykonanie komendy **throw powoduje natychmiastowe** przerwanie sekwencyjnego wykonania programu.

Wykonanie programu przenosi się do najbliższej sekcji obsługi zgłoszonego wyjątku.

Jeżeli takiej sekcji nie ma, to program zostanie zatrzymany, a domyślny program obsługi wypisze ścieżkę wywołań metod aż do zgłoszonego wyjątku.

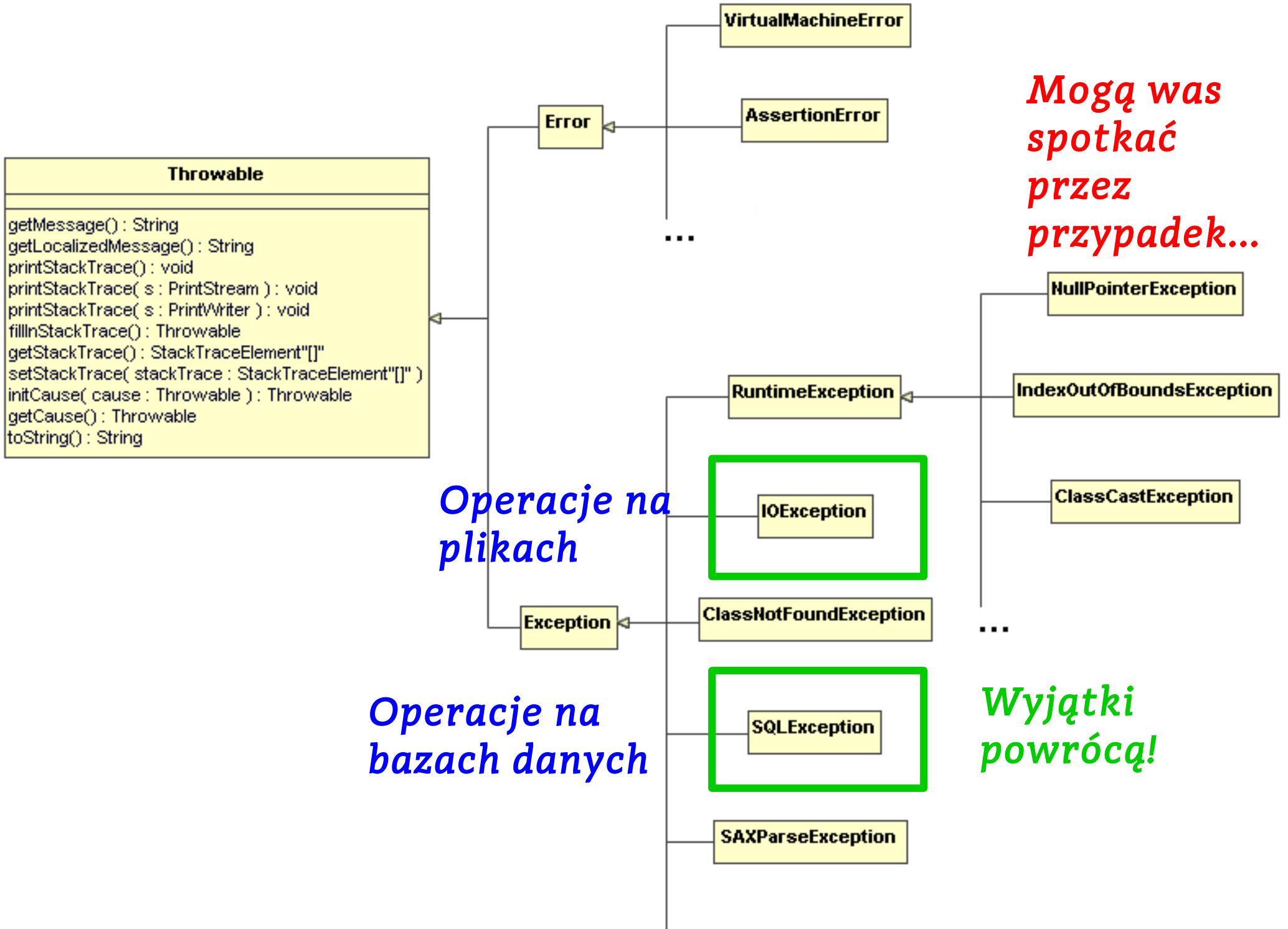
```
public void setY(double y) {
      if (y < YMIN || x > YMAX)
            throw new IllegalArgumentException();
      this.y = y;
}
```

Obsługa wyjątków: try - catch

W przypadku wystąpienia nieobsługiwanego wyjątku program kończy pracę.

Aby samemu obsłużyć błąd powodujący przerwanie programu należy umieścić go w bloku **try {}, a następnie w bloku catch{} umieścić typy** wyjątków, na które chcemy reagować oraz związać z nimi kod obsługujący zgłoszony wyjątek.

Należy również pamiętać, że po obsłudze wyjątku przez blok **try/catch** program nie wraca do komendy następnej w bloku **try lecz przechodzi** do wykonania pierwszej instrukcji za blokiem **try/catch.**


VirtualMachineError
Error
AssertionError
Mogą was spotkać przez przypadek...
Throwable
getMessage() : String
getLocalizedMessage() : String
printStackTrace() : void
printStackTrace(s : PrintStream) : void
printStackTrace(s : PrintWriter) : void
fillInStackTrace() : Throwable
getStackTrace() : StackTraceElement"[]"
setStackTrace(stackTrace : StackTraceElement"[]")
initCause(cause : Throwable) : Throwable
getCause() : Throwable
toString() : String
...
NullPointerException
RuntimeException
IndexOutOfBoundsException
Operacje na plikach
IOException
ClassCastException
Exception
ClassNotFoundException
...
Operacje na bazach danych
SQLException
Wyjątki powrócą!
SAXParseException

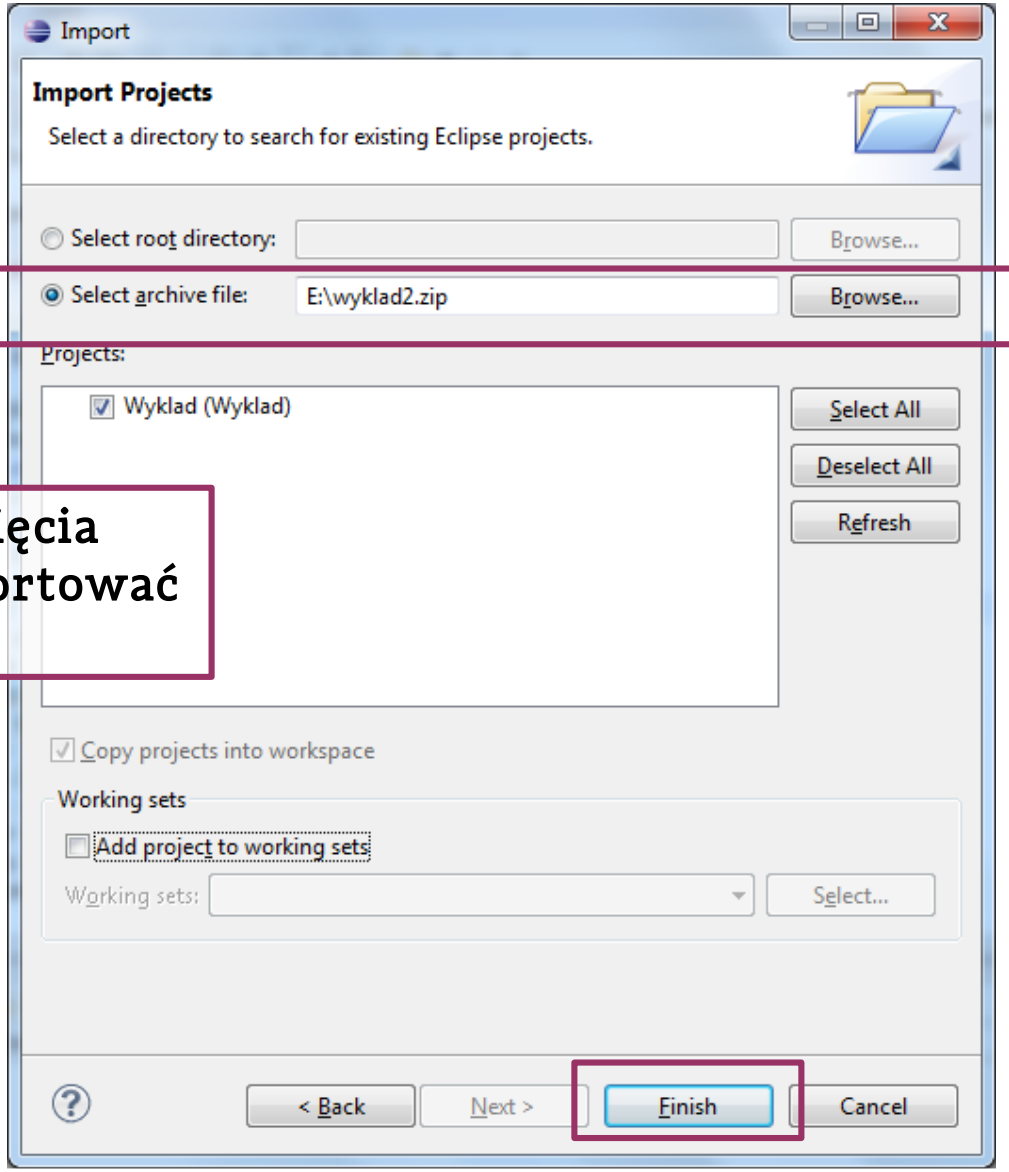
Swing

JComponent
(setToolTopText, setBorder)

LookAndFeel

Importowanie projektu do Eclipse:

File → Import → General – Existing Projects into Workspace → archive file:



Przykłady do ściągnięcia ze strony należy zaimportować w ten sposób

http://realsearchgroup.org/SEMaterials/tutorials/import_export/index.html#section2_0

JComponent

- Wszystkie komponenty Swing których nazwy zaczynają się od „J” są pochodnymi klasy JComponent.
- Wybrane metody:

– paintComponent() - rysowanie na komponencie,

– setToolTipText() - ustawienie dymku podpowiedzi,

– setBorder() - ustawienie obramowania,

- metody używane przez „zarządców układu” (layout managers) –

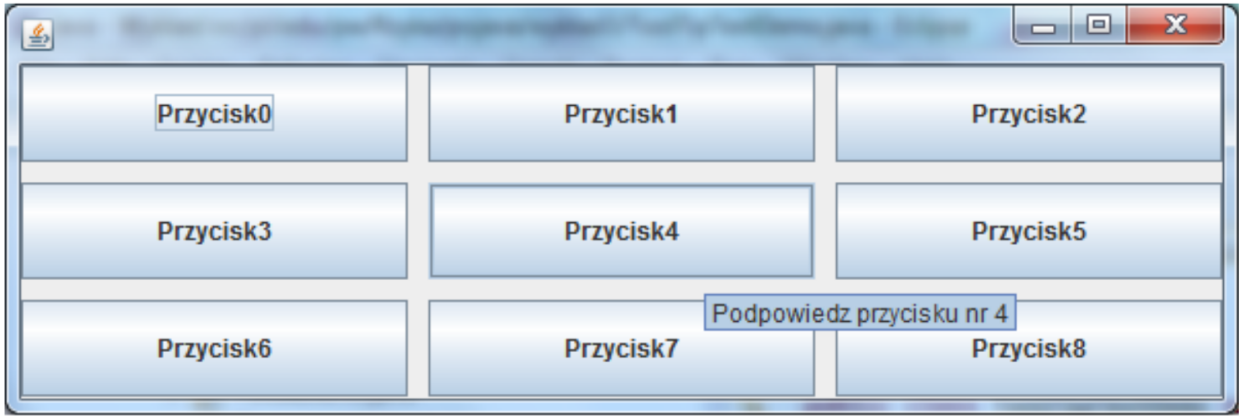
metody getPreferredSize, getAligmentX, getMinimumSize,

getMaximumSize

Każdy komponent ma odpowiadający obiekt ComponentUI, który przeprowadza rysowanie, przechwytywanie zdarzeń, ustalanie rozmiaru, itd. jest on zależny od bieżącego wyglądu interfejsu, który ustawiamy poleceniem UIManager.setLookAndFeel(...)

setToolTipText() - przykład

ToolTipTextDemo.java




```
setLayout(new GridLayout(3,3,10,10));
        JButton guziki[] = new JButton[9];
        for (int i = 0; i<9; i++){
                guziki[i] = new JButton ("Przycisk" + i);
                guziki[i].setToolTipText("Podpowiedz przycisku nr " + i);
                add(guziki[i]);
                }
```

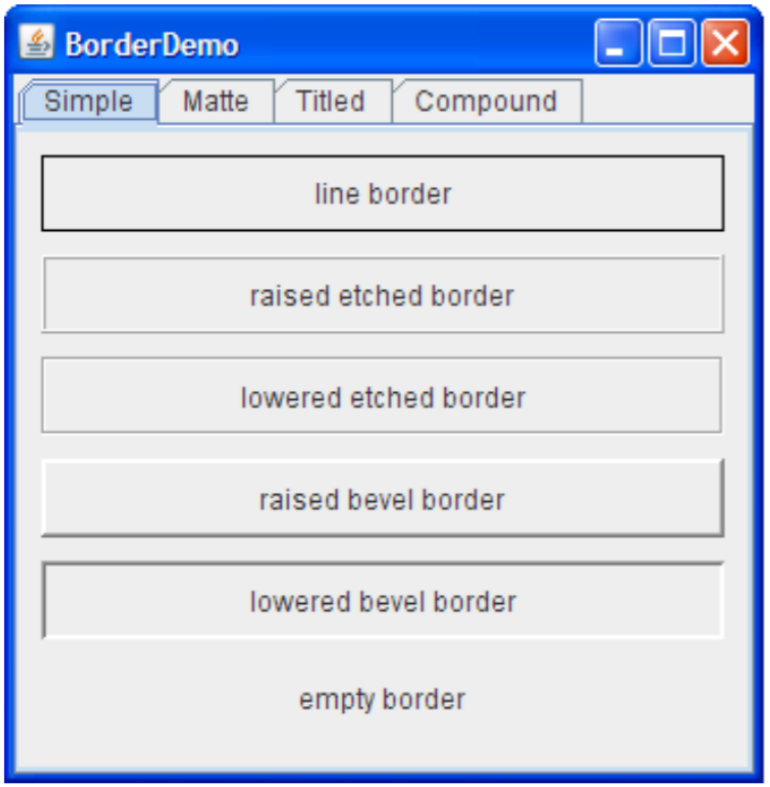
setBorder() - przykład

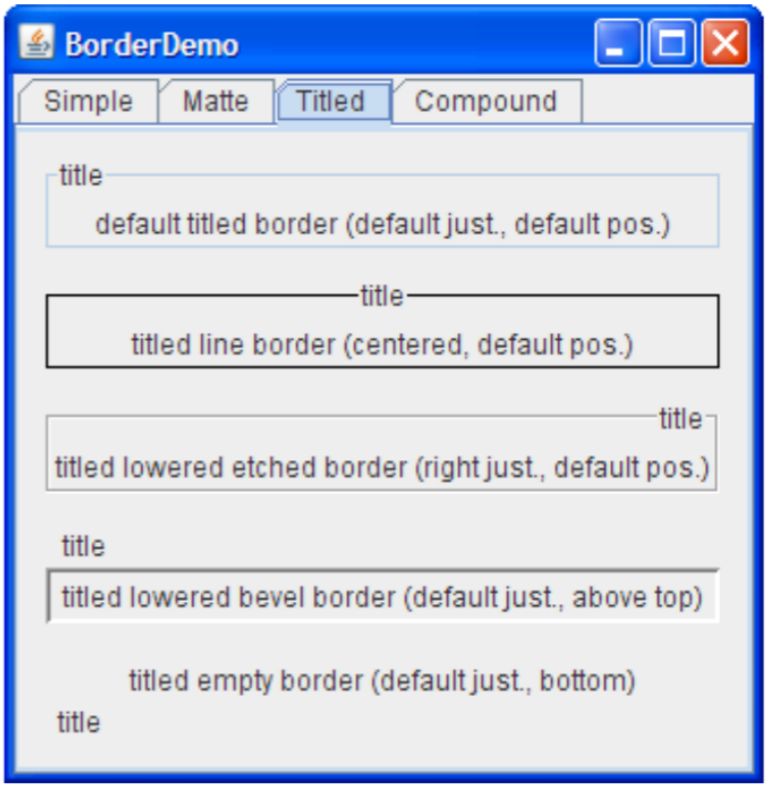
BorderDemo.java

```
JPanel panele[] = new JPanel[9];
for (int i = 0; i<9; i++){
  panele[i] = new JPanel ();
  panele[i].setBorder(BorderFactory.createLineBorder(new
                                                Color(i*1,i*20,i*10)));
  add(panele[i]);
}
```

setBorder() - przykład

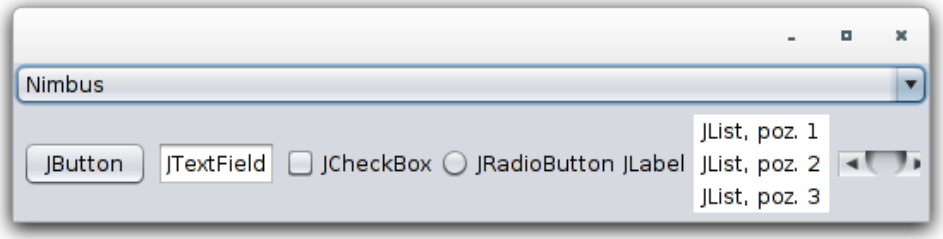
Więcej typów ramek w dokumentacji:
http://docs.oracle.com/javase/tutorial/uiswing/components/border.html

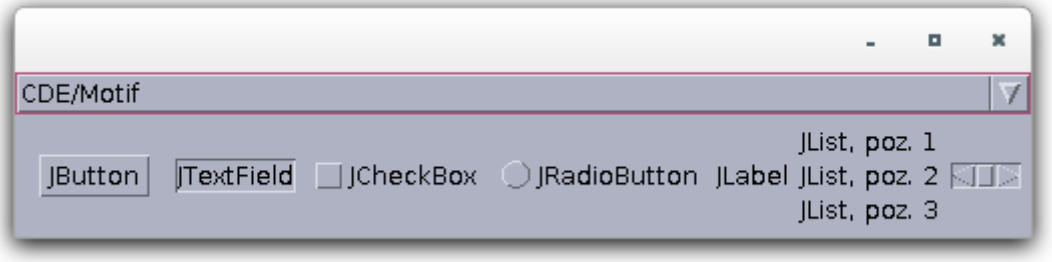


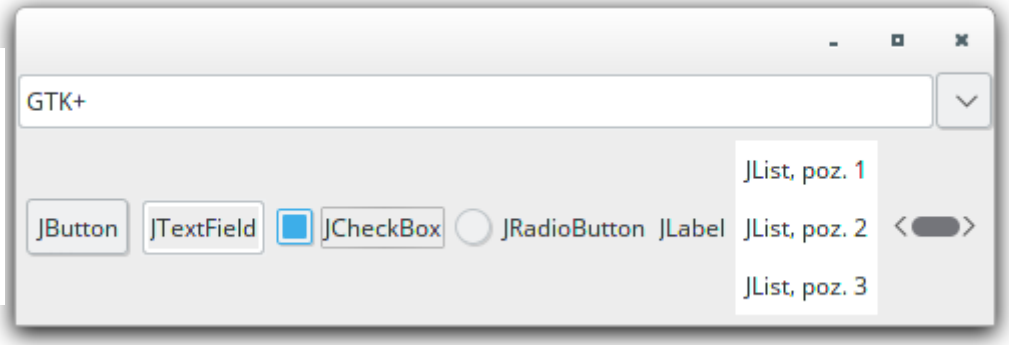


Look And Feel

- "Look and Feel" przy projektowaniu GUI to jest zespół cech związanych z wyglądem (*look*) kolorami, kształtami, jak również zachowaniem (*feel*) komponentów
- Zwykle każdy system operacyjny ma własny LookAndFeel (np. wszystkie okienka aplikacji na windowsie wyglądają podobnie), istnieją też LookAndFeel dostępne na wszystkich platformach.







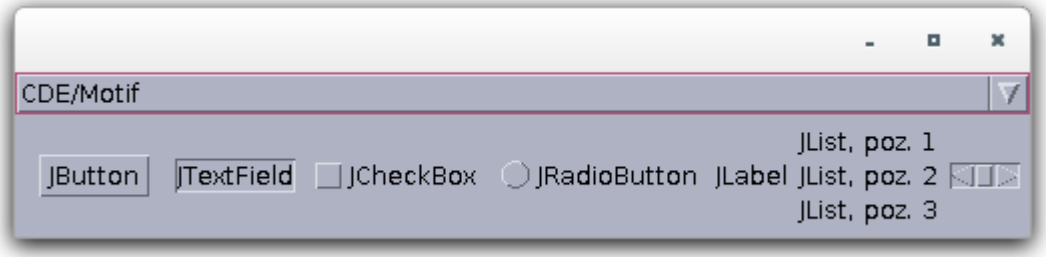
Look And Feel

Jak ustawić konkretny wygląd?

- UIManager.setLookAndFeel (nazwa look-and-feel)

Jak sprawdzić jakie są dostępne na naszym systemie?

- final UIManager.LookAndFeelInfo[] installedLF = UIManager.getInstalledLookAndFeels();



LookAndFeelChooser.java

Tworzenie GUI

Layout

Wybrane kontrolki:

JMenu
Okna dialogowe
JColorChooser
inne

Layout Manager

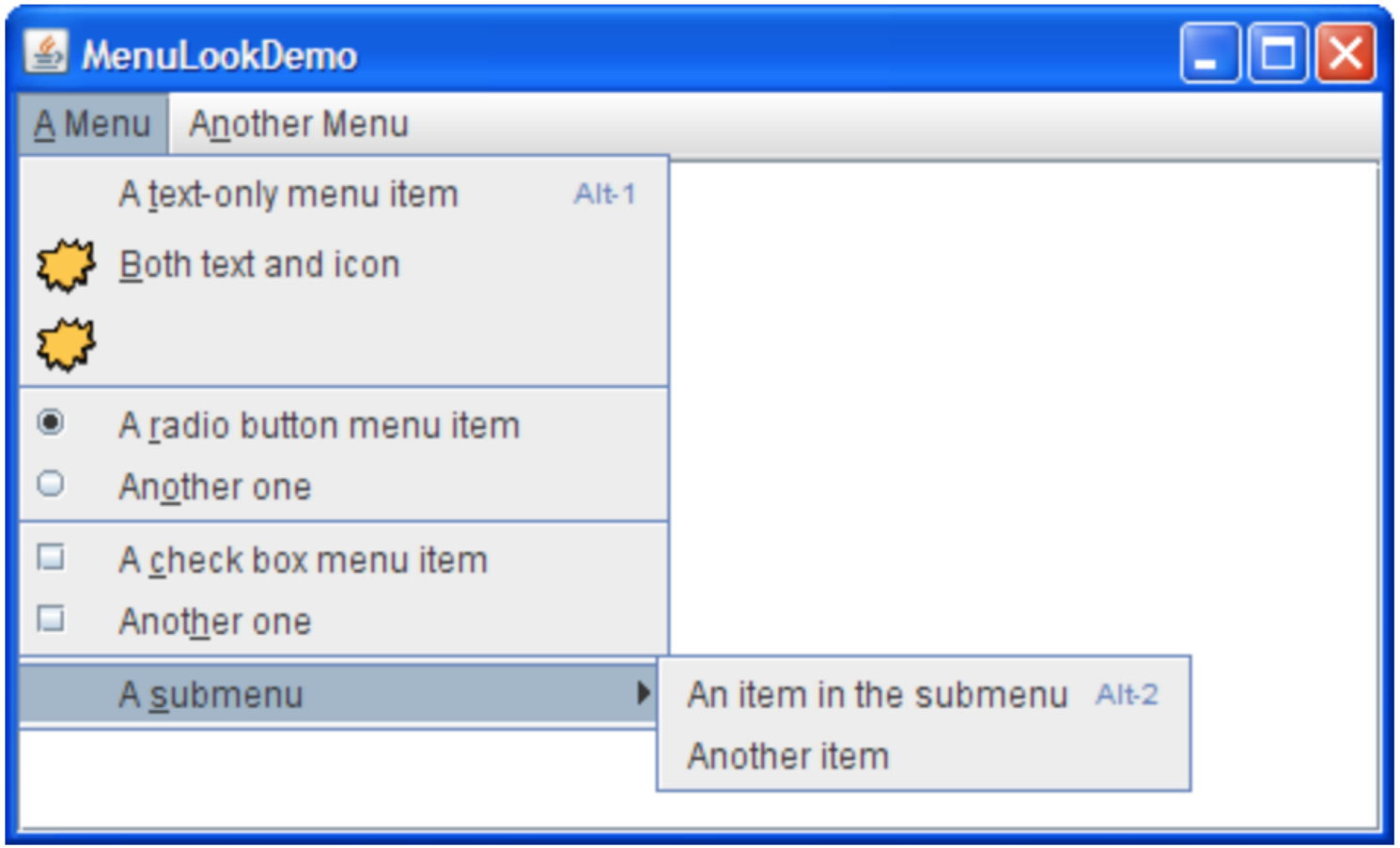
- Pierwszy krok tworzenia GUI to zwykle opracowanie układu i rozmieszczenia komponentów

LayoutManagerDemo.java

- Warto pamiętać:
  - Nie wszystkie składniki BorderLayout muszą być dodane
  - W ramach różnych paneli można stosować różne typy zarządzania rozmieszczaniem komponentów
  - Domyślny zarządca układu dla JFrame to BorderLayout, a dla JPanel: FlowLayout


```
JPanel p1 = new JPanel();
JPanel p2 = new JPanel();
JPanel p3 = new JPanel();

add(BorderLayout.WEST, p1);
add(BorderLayout.NORTH, p2);
add(BorderLayout.CENTER, p3);
p1.setLayout(new GridLayout(4,1));
p1.add(new JLabel("Panel 1"));
p1.add(new JButton("Przycisk 1"));
p1.add(new JButton("Przycisk 2"));
p1.add(new JButton("Przycisk 3"));

p2.setLayout(new FlowLayout());
p2.add(new JLabel("Panel 2"));
p2.add(new JCheckBox("Kontrolka 1"));
p2.add(new JCheckBox("Kontrolka 1"));
p2.add(new JCheckBox("Kontrolka 1"));
```

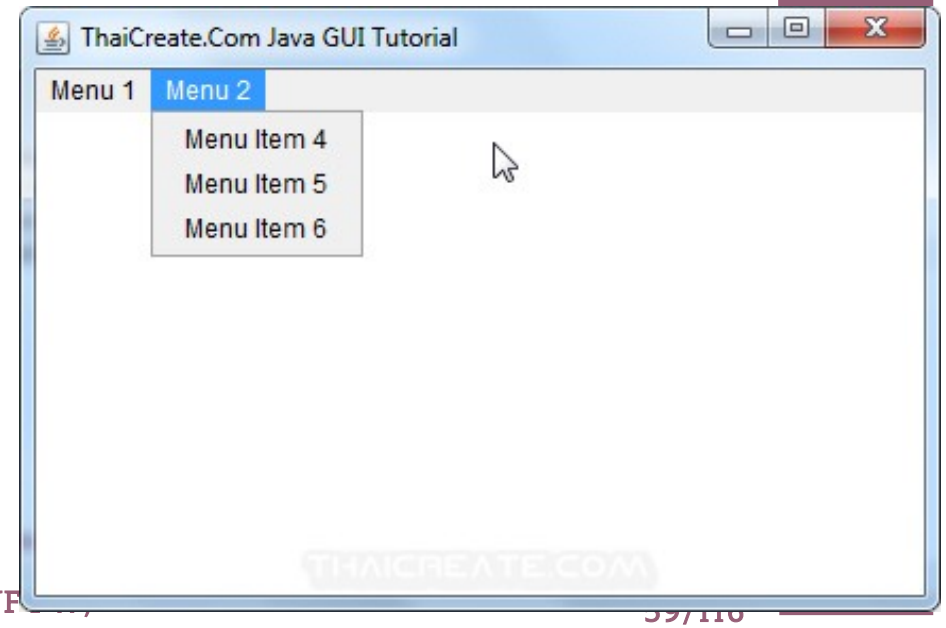
JMenu



https://docs.oracle.com/javase/tutorial/uiswing/components/menu.html

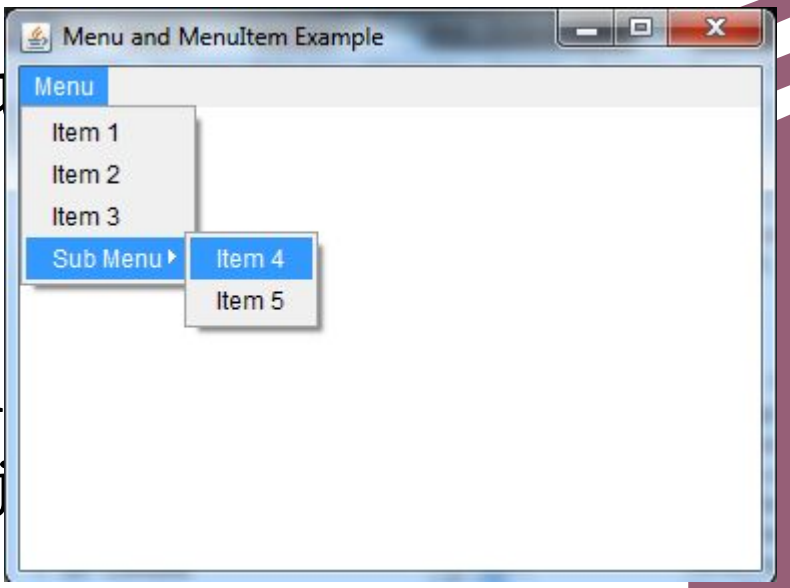
JMenu – jak działa?

- Niezbędne elementy:
  - **JMenuBar** – pasek menu – ustawiany dla JFrame
    - **JMenu** – pojedynczy wybór z paska menu (np. “Plik”, “Edycja”) - dodawany do JMenuBar
      - **JMenuItem** – element na rozwijanej liście menu – dodawany do JMenu

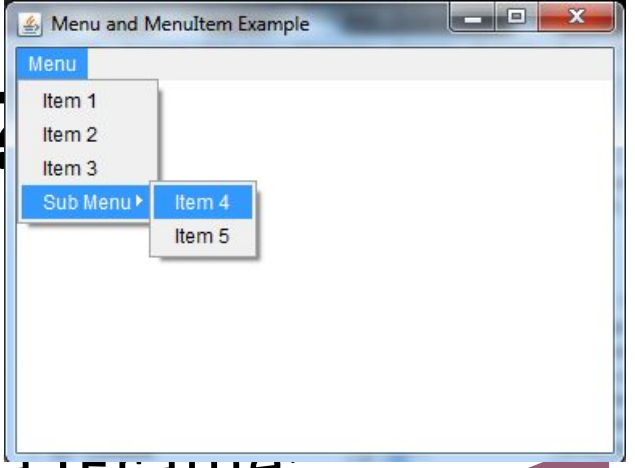


JMenu – jak działa?

- Niezbędne elementy:
  - **JMenuBar** – pasek menu – u
    JFrame
    - **JMenu** – pojedynczy wybór z
      "Plik", "Edycja") - dodawany d
      - **JMenuItem** – element na rozwij
        dodawany do Jmenu
      - Żeby zrobić kolejne pod-menu wystarczy dodać nowy **JMenu** do już istniejącego.



JMenu – jak działa?



- Niezbędne elementy:
  - **JMenuBar** – pasek menu – ustawiany dla JFrame:

```
JMenuBar menuBar = new JMenuBar();
frame.setJMenuBar(menuBar);
```

    - **JMenu** – pojedynczy wybór z paska menu (np. “Plik”, “Edycja”) - dodawany do JMenuBar

```
JMenu menu = new JMenu("Menu");
menuBar.add(menu);
```

      - **JMenuItem** – element na rozwijanej liście menu – dodawany do JMenu

```
JMenu menuItem1 = new JMenu(“Item 1");
menu.add(menuItem1);
```

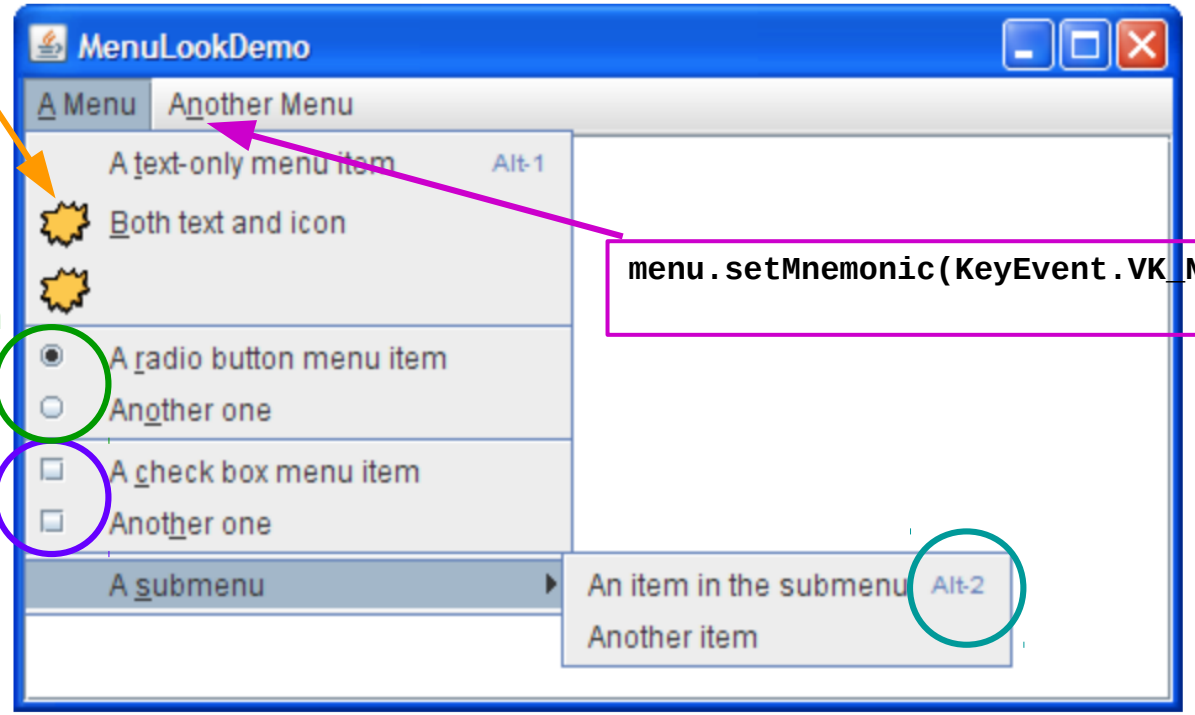
      - Żeby zrobić kolejne pod-menu wystarczy dodać nowy **JMenu** do już istniejącego.

```
JMenu submenu = new Jmenu(“Sub Menu");
JMenuItem menuItem4 = new JmenuItem("Item 4");
submenu.add(menuItem4);
menu.add(submenu);
```

Tworzenie GUI
JMenu
[przykład]

JMenu

```
menuItem = new JMenuItem("Both text and icon",
                         new ImageIcon("images/middle.gif"));
```




```
menu.setMnemonic(KeyEvent.VK_N);
```

JRadioButtonMenuItem

JCheckBoxMenuItem

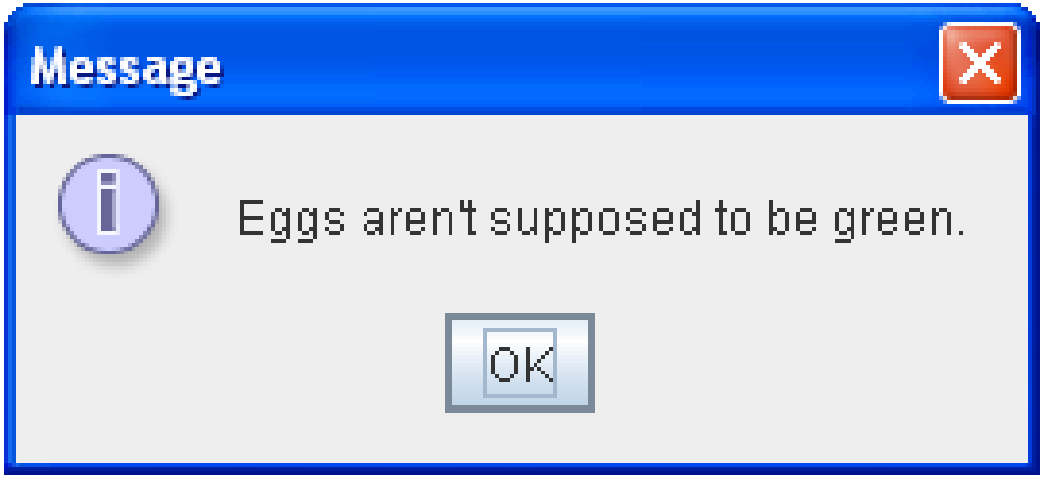
```
menuItem.setAccelerator(
 KeyStroke.getKeyStroke(KeyEvent.VK_2,
                        ActionEvent.ALT_MASK));
```

https://docs.oracle.com/javase/tutorial/uiswing/components/menu.html

```
public class MenuWindow extends JFrame {

MenuWindow()
{
     super("Okno z menu");
     setDefaultCloseOperation(EXIT_ON_CLOSE);
     setSize(800,800);

     JMenuBar menuBar = new JMenuBar();

     JMenu plik = new JMenu("File");

     JMenu submenu = new JMenu("A submenu");
     JMenuItem menuItem = new JMenuItem("An item in the submenu");
     menuItem.setAccelerator(KeyStroke.getKeyStroke(
         KeyEvent.VK_2, ActionEvent.ALT_MASK));
     submenu.add(menuItem);
     plik.add(submenu);

     plik.addSeparator();
     JMenuItem exit = new JMenuItem("Exit");
     exit.addActionListener(new ActionListener(){
          @Override
          public void actionPerformed(ActionEvent arg0) {
               System.exit(0);
          }});
     plik.add(exit);

     menuBar.add(plik);
     this.setJMenuBar(menuBar);
}
public static void main(String[] args) {
     MenuWindow frame = new MenuWindow();
     frame.setVisible(true);
}
}
```

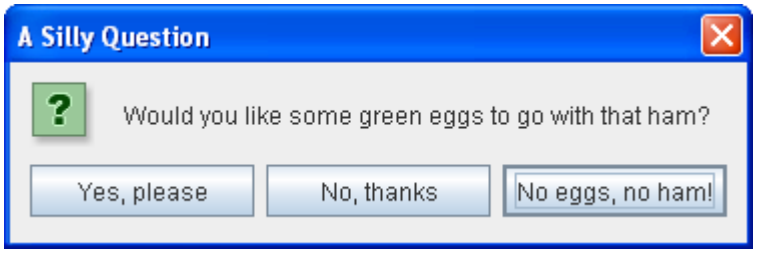
Więcej:
JMenuDemo.java
JMenuDemo2.java

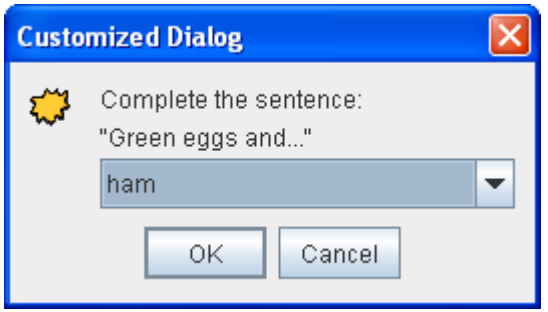
Okno dialogowe

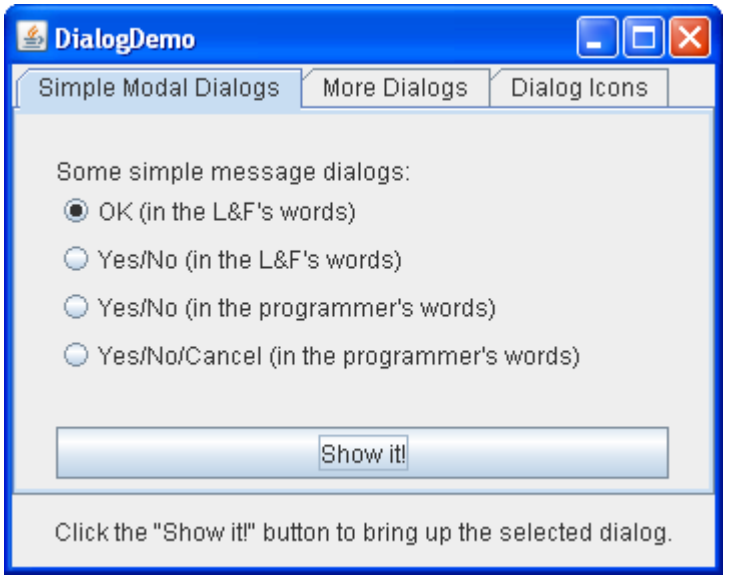


https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html

Okno dialogowe

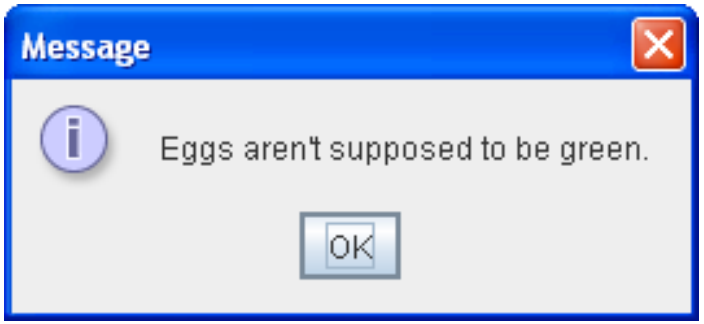






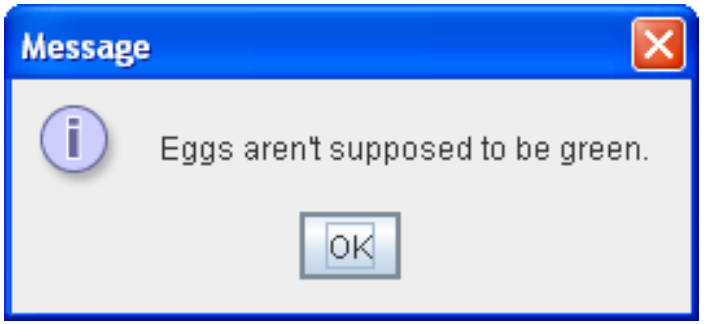
https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html

Okno dialogowe

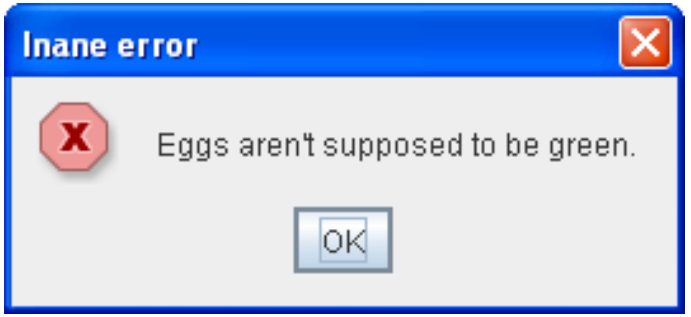



```
JOptionPane.showMessageDialog(null,
    "Eggs aren't supposed to be green.");
```

Okno dialogowe




```
JOptionPane.showMessageDialog(null,
    "Eggs aren't supposed to be green.");
```

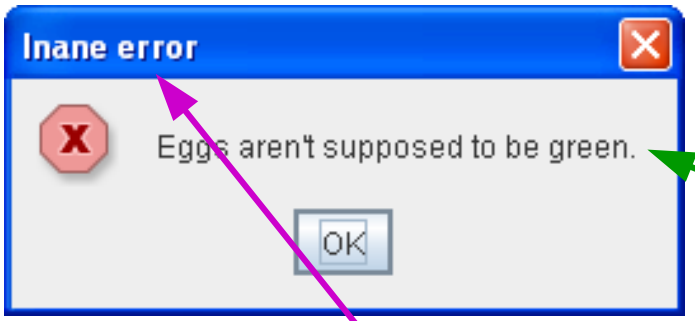



```
JOptionPane.showMessageDialog(frame, "Eggs aren't supposed to be
green.","Inane error",JOptionPane.ERROR_MESSAGE);
```

Okno dialogowe

showMessageDialog(Component parentComponent, Object message, String title, int messageType, Icon icon)

Tytuł okienka



Informacja w okienku

```
JOptionPane.showMessageDialog(frame, "Eggs aren't supposed to be
green.","Inane error",JOptionPane.ERROR_MESSAGE);
```

- **null – jeśli okienko ma się pojawić po środku ekranu**
- **Referencja do JFrame – jeśli okienko ma się pojawić na środku okna programu**

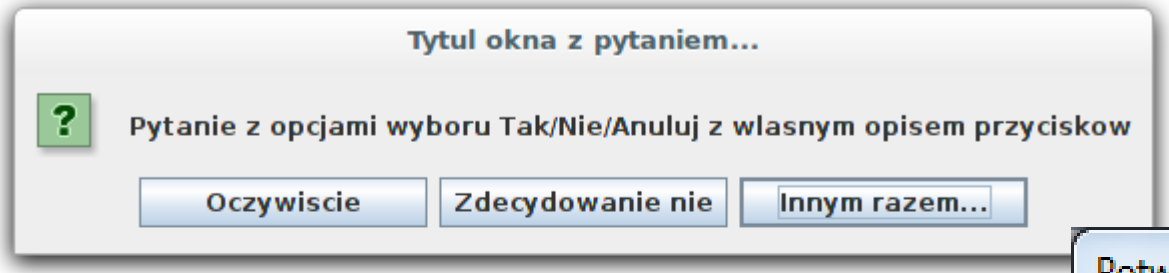
Ikonka

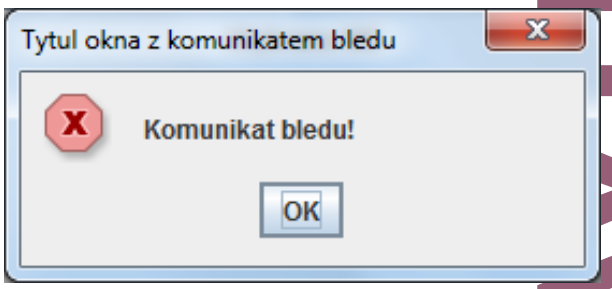
Icons used by JOptionPane

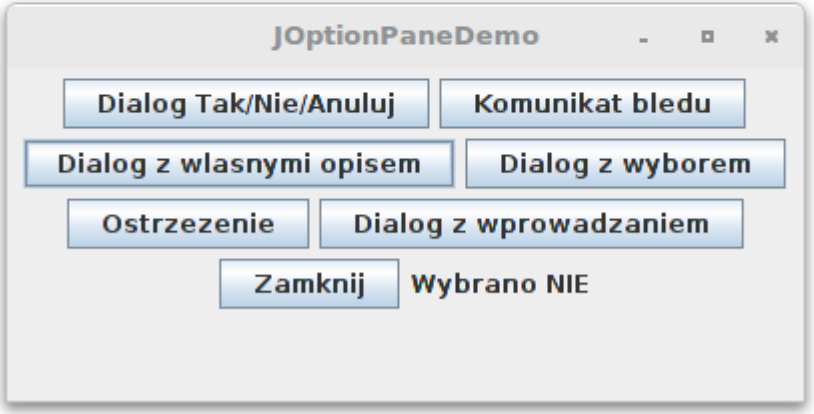
	Icon description	Java look and feel	Windows look and feel
QUESTION_MESSAGE	question		
INFORMATION_MESSAGE	information		
WARNING_MESSAGE	warning		
ERROR_MESSAGE	error		

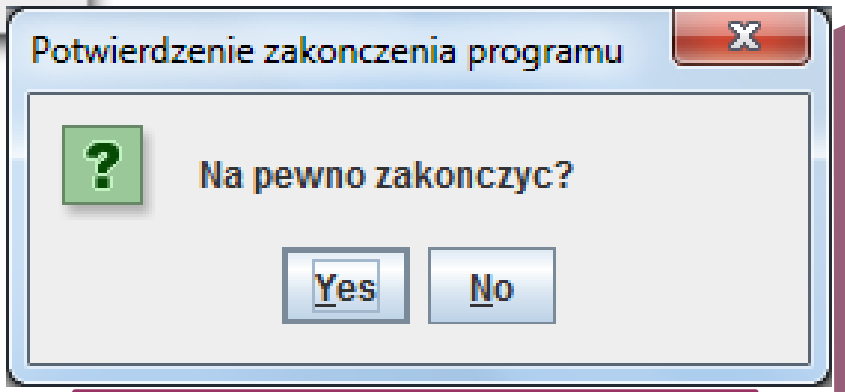
Tworzenie GUI
JOptionPane [przykład]

Okno dialogowe

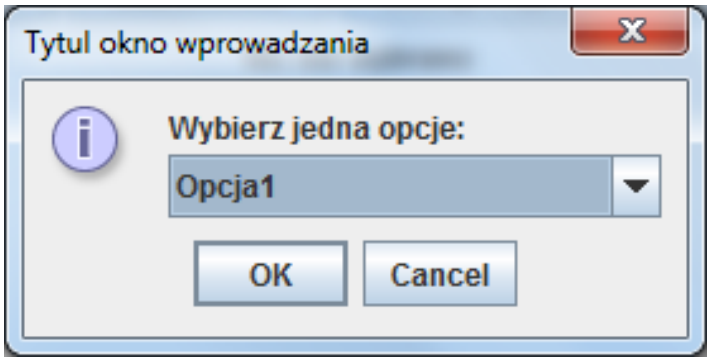


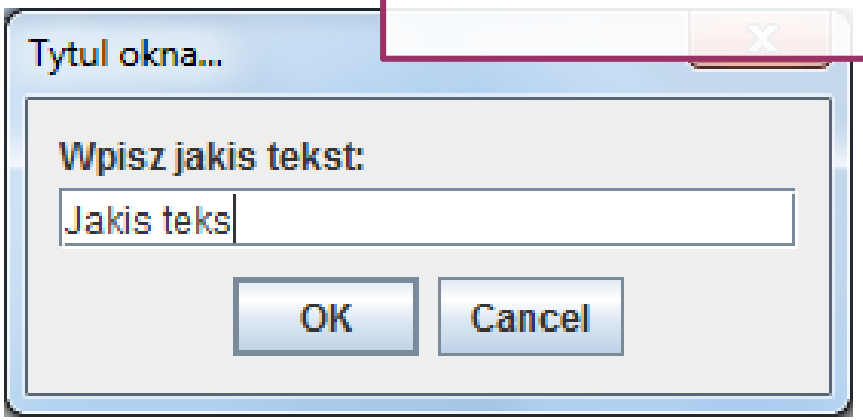






Więcej:
JOptionPaneDemo.java

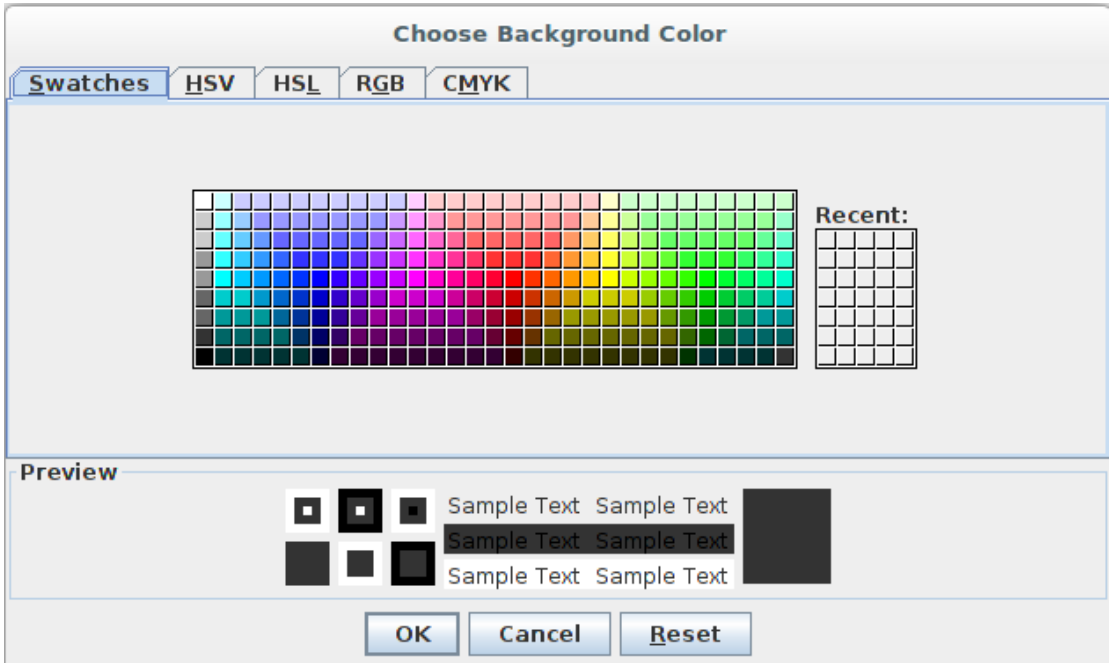




Tworzenie GUI
JEditorPane
[przykład]

JColorChooser

JColorChooser pozwala wybrać kolor z palety kolorów.



Wyskakujące okienko dialogowe z paletą kolorów:

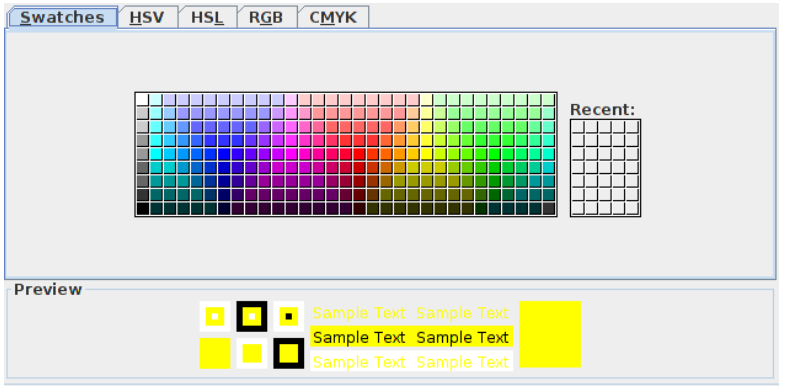
```
Color newColor = JColorChooser.showDialog(
                     null,
                     "Choose Color",
                     textEditor.getForeground());
```

https://docs.oracle.com/javase/tutorial/uiswing/components/colorchooser.html

Tworzenie GUI
JColorChooser [przykład]

JColorChooser

Można też wstawić sobie paletę bezpośrednio do okienka lub panelu:



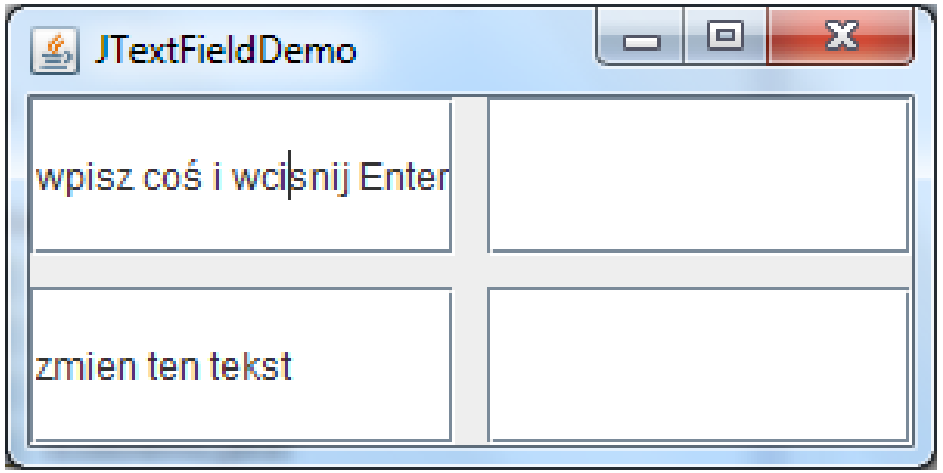

```
protected JColorChooser tcc;

    public ColorChooserDemo() {
        super(new BorderLayout());

        tcc = new JColorChooser(kolorStartowy);
        tcc.getSelectionModel().addChangeListener(this);
        tcc.setBorder(BorderFactory.createTitledBorder(
                                              "Choose Text Color"));

        add(tcc, BorderLayout.PAGE_END);
    }

    public void stateChanged(ChangeEvent e) {
        Color newColor = tcc.getColor();
    }
```

Więcej: ColorChooserDemo.java

https://docs.oracle.com/javase/tutorial/uiswing/components/colorchooser.html

Tworzenie GUI
Przykłady wybranych implementacji

JTextFieldDemo.java




```
JTextField pole1 = new JTextField(„wpisz cos i wcisnij Enter");
JTextField pole2 = new JTextField(20);
JTextField pole3 = new JTextField("zmien ten tekst");
JTextField pole4 = new JTextField(20);

pole1.addActionListener(pole1Listener);
pole3.addKeyListener(pole3Listener);
add(pole1);
add(pole2);
add(pole3);
add(pole4);
```

JTextFieldDemo.java II

```
ActionListener pole1Listener = new ActionListener() {
        public void actionPerformed(ActionEvent e)
        {
              pole2.setText( pole1.getText()) ;
        }
  };
```

Action:
naciśnięcie enter po wpisaniu tekstu

```
  KeyListener pole3Listener = new KeyListener() {
        @Override
        public void keyTyped(KeyEvent e) {
        }

        @Override
        public void keyReleased(KeyEvent e) {
              pole4.setText( pole3.getText()) ;
        }

        @Override
        public void keyPressed(KeyEvent e) {
        }
  };
```

Key listener:
Metoda keyReleased odpali się
po puszczeniu danego klawisza

keyPressed - when the key goes down
keyReleased - when the key comes up
keyTyped - when the unicode character represented by this key is sent by the keyboard to system input.

JTextFieldDemo.java III

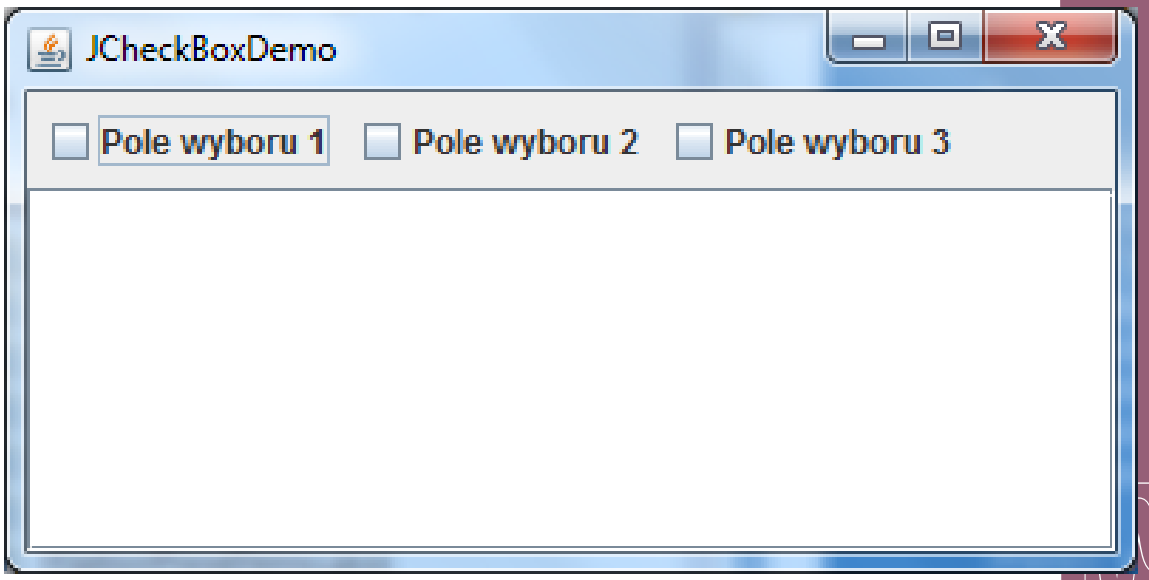
```
// Jesli nie wszystkie metody z KeyListener sa
                                        wykorzystane
// bardziej przejrzystej jest korzystanie z KeyAdaptera
// implementujacego wybrane metody:
KeyListener pole3Adapter = new KeyAdapter() {
      public void keyReleased(KeyEvent e) {
            pole4.setText( pole3.getText()) ;
      }
};
```

Ważne!
Jeśli nie potrzebujemy wszystkich metod danego listenera, to wystarczy użyć Adapter

Inne przykłady adapterów:
MouseAdapter (zamiast MouseListener)
MouseMotionAdapter (zamiast MouseMotionListener)
WindowAdapter (zamiast WindowListener)

JCheckBoxDemo.java

```
JCheckBox cb1 = new JCheckBox("Pole wyboru 1");
JCheckBox cb2 = new JCheckBox("Pole wyboru 2");
JCheckBox cb3 = new JCheckBox("Pole wyboru 3");

cb1.addActionListener(cbListener);
cb2.addActionListener(cbListener);
cb3.addActionListener(cbListener);

Panel panelCheckBox = new JPanel();
panelCheckBox.setLayout(new FlowLayout(FlowLayout.LEFT));

panelCheckBox.add(cb1);
panelCheckBox.add(cb2);
panelCheckBox.add(cb3);
```



JCheckBoxDemo.java

```
ActionListener cbListener = new ActionListener() {

      public void actionPerformed(ActionEvent e)
      {
            Object obj = e.getSource();

            if (obj == cb1)
              if (cb1.isSelected()) t.append("Pole 1 ustawione\n");
                else t.append("Pole 1 wyczyszczone\n");

            if (obj == cb2)
              if (cb2.isSelected()) t.append("Pole 2 ustawione\n");
                else t.append("Pole 2 wyczyszczone\n");

            if (obj == cb3)
              if (cb3.isSelected()) t.append("Pole 3 ustawione\n");
                else t.append("Pole 3 wyczyszczone\n");

      }
 };
```

Ważna metoda: isSelected()

Przykład obsługi kilku CheckBox'ów w jednym interfejsie…

JCheckBoxDemo.java

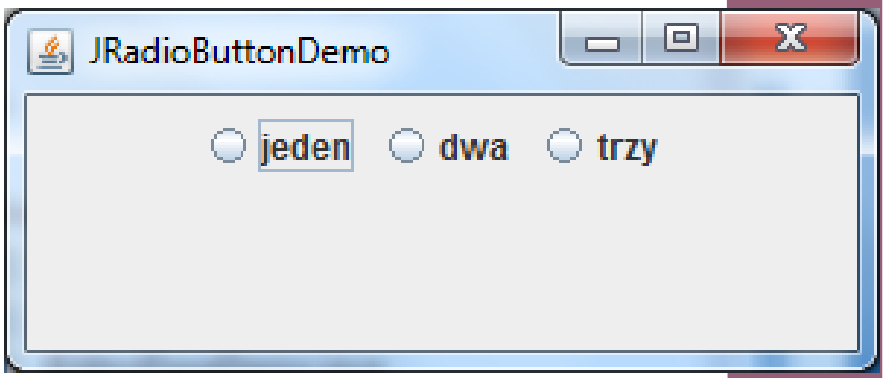
```
ActionListener cbListener = new ActionListener() {

      public void actionPerformed(ActionEvent e)
      {
            Object obj = e.getSource();

            if (obj == cb1)
              if (cb1.isSelected()) t.append("Pole 1 ustawione\n");
                else t.append("Pole 1 wyczyszczone\n");

            if (obj == cb2)
              if (cb2.isSelected()) t.append("Pole 2 ustawione\n");
                else t.append("Pole 2 wyczyszczone\n");

            if (obj == cb3)
              if (cb3.isSelected()) t.append("Pole 3 ustawione\n");
                else t.append("Pole 3 wyczyszczone\n");

      }
 };
```

Ważne!
Możemy wyciągnąć obiekt który został naciśnięty z obiektu danego zdarzenia

Przykład obsługi kilku CheckBox'ów w jednym interfejsie…

JRadioButtonDemo.java

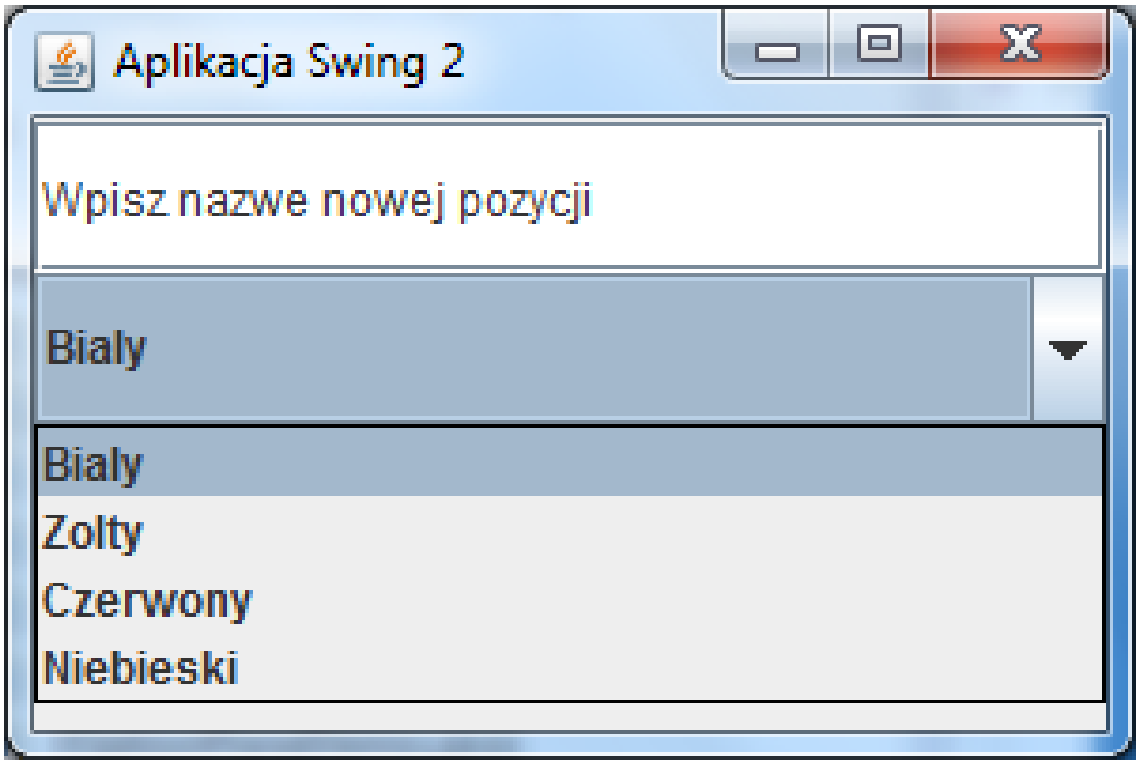
```
  ButtonGroup grupa = new ButtonGroup();
  JRadioButton
    rb1 = new JRadioButton("jeden", false),
    rb2 = new JRadioButton("dwa", false),
    rb3 = new JRadioButton("trzy", false);

// Grupowanie obektow JRadioButton do ButtonGroup
//- tylko jeden moze byc zaznaczony
  grupa.add(rb1); grupa.add(rb2); grupa.add(rb3);
  rb1.addActionListener(listener);
  rb2.addActionListener(listener);
  rb3.addActionListener(listener);

  add(rb1); add(rb2); add(rb3);
```




```
  ActionListener listener = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      etykieta.setText("Przycisk wyboru " +
        ((JRadioButton)e.getSource()).getText());
    }
  };
```

JComboBoxDemo.java

```
String[] description = { "Bialy", "Zolty",
            "Czerwony", "Niebieski", };
JTextField poleTekstowe = new JTextField("Wpisz nazwe nowej pozycji");
JComboBox comboBox = new JComboBox(description);
JButton przycisk = new JButton("Dodaj pozycje");
JLabel etykieta = new JLabel();
```



JComboBoxDemo.java

```
String[] description = { "Bialy", "Zolty",
            "Czerwony", "Niebieski", };
JTextField poleTekstowe = new JTextField("Wpisz nazwe nowej pozycji");
JComboBox comboBox = new JComboBox(description);
JButton przycisk = new JButton("Dodaj pozycje");
JLabel etykieta = new JLabel();

 comboBox.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e){
          etykieta.setText("indeks: "+ comboBox.getSelectedIndex()
            + "   " + comboBox.getSelectedItem());
          if (comboBox.getSelectedItem().equals("Bialy"))
                etykieta.setText("Wybrano kolor Bialy");

        }

add(comboBox);
```

JComboBoxDemo.java

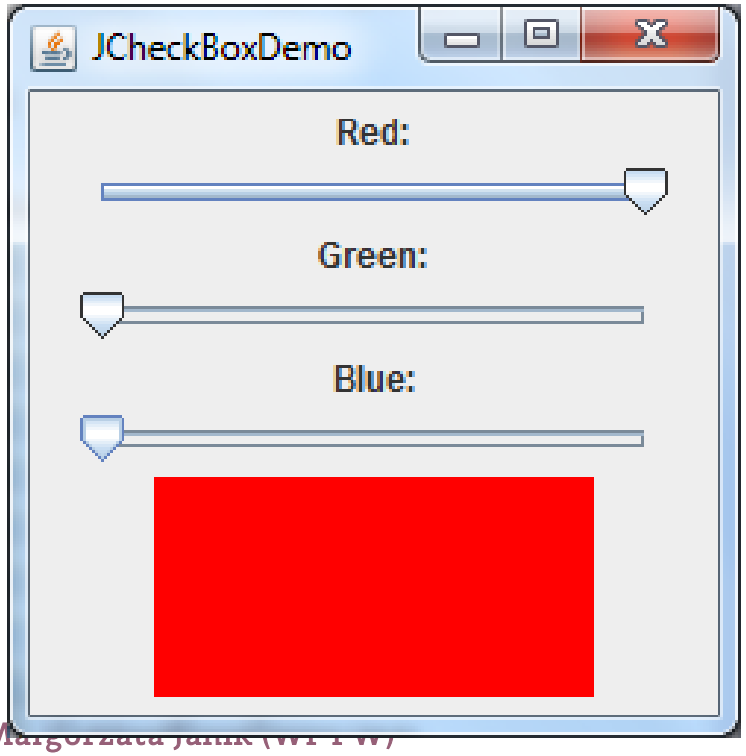
```
String[] description = { "Bialy", "Zolty",
            "Czerwony", "Niebieski", };
JTextField poleTekstowe = new JTextField("Wpisz nazwe nowej pozycji");
JComboBox comboBox = new JComboBox(description);
JButton przycisk = new JButton("Dodaj pozycje");
JLabel etykieta = new JLabel();

  comboBox.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e){
          etykieta.setText("indeks: "+ comboBox.getSelectedIndex()
            + "    " + comboBox.getSelectedItem());
          if (comboBox.getSelectedItem().equals("Bialy"))
           etykieta.setText("Wybrano kolor Bialy");
        }

add(comboBox);

przycisk.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e){
          if (poleTekstowe.getText() != "")
                comboBox.addItem(poleTekstowe.getText());
            etykieta.setText("Dodano: " + poleTekstowe.getText());
           }
      });
```

JSliderDemo.java

```
JSlider redSlider,greenSlider,blueSlider;

redSlider = new JSlider();
redSlider.setMinimum(0);
redSlider.setMaximum(255);
redSlider.setValue(127);

greenSlider = new JSlider(0,255,127);

blueSlider = new JSlider(JSlider.HORIZONTAL, 0, 255, 127);

redSlider.addChangeListener(slidersListener);
greenSlider.addChangeListener(slidersListener);
blueSlider.addChangeListener(slidersListener);

add(new JLabel("Red:"));
add(redSlider);
add(new JLabel("Green:"));
add(greenSlider);
add(new JLabel("Blue:"));
add(blueSlider);
```

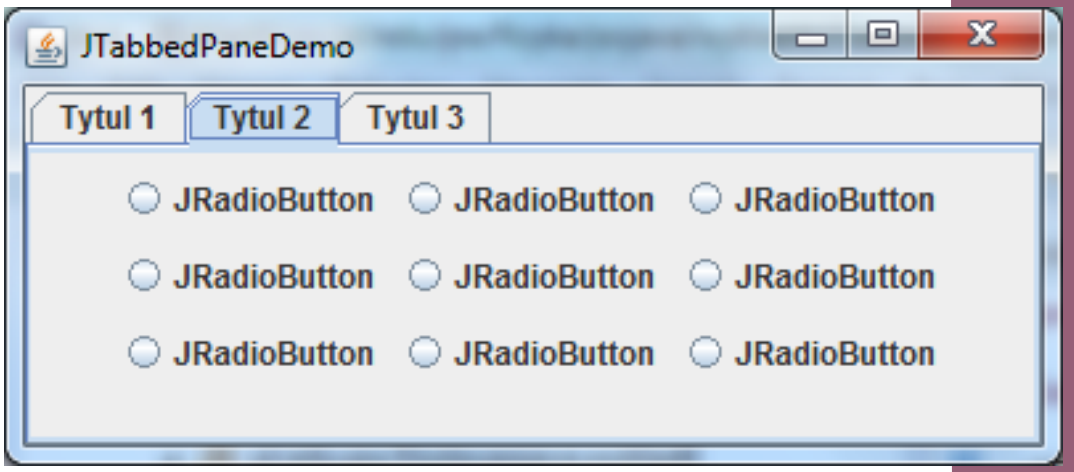
JSliderDemo.java

```
JPanel panel = new JPanel();

ChangeListener slidersListener = new ChangeListener() {

        @Override
   public void stateChanged(ChangeEvent e) {
        int red = redSlider.getValue();
        int green = greenSlider.getValue();
        int blue = blueSlider.getValue();
        panel.setBackground(new Color(red, green, blue));
   }
};
```



JTabbedPaneDemo.java

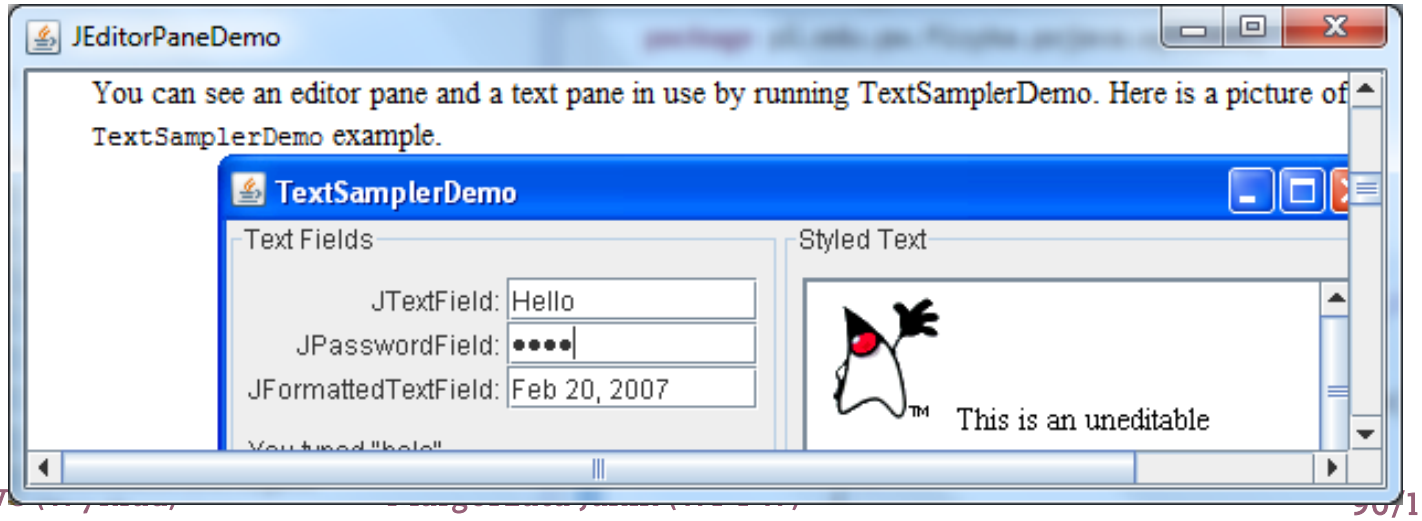
```
JTabbedPane tabbedPane = new JTabbedPane();

JPanel panel1 = new JPanel(new FlowLayout());
JPanel panel2 = new JPanel(new FlowLayout());
JPanel panel3 = new JPanel(new FlowLayout());

tabbedPane.addTab("Tytul 1", panel1);
tabbedPane.addTab("Tytul 2", panel2);
tabbedPane.addTab("Tytul 3", panel3);
add(tabbedPane);
// tabbedPane.remove(panel2);
```



JEditorPaneDemo.java

```
JEditorPane edytor = new JEditorPane();
edytor.setEditable(false);
    try
    {   URL link = new   URL("http://jakasstrona.html");
        edytor.setPage(link);
    }
    catch(IOException e)
    {
        edytor.setText("Wyjatek:"+e);
    }
add(new JScrollPane(edytor));
```

Poprawnie wyświetli tylko najprostrze strony....

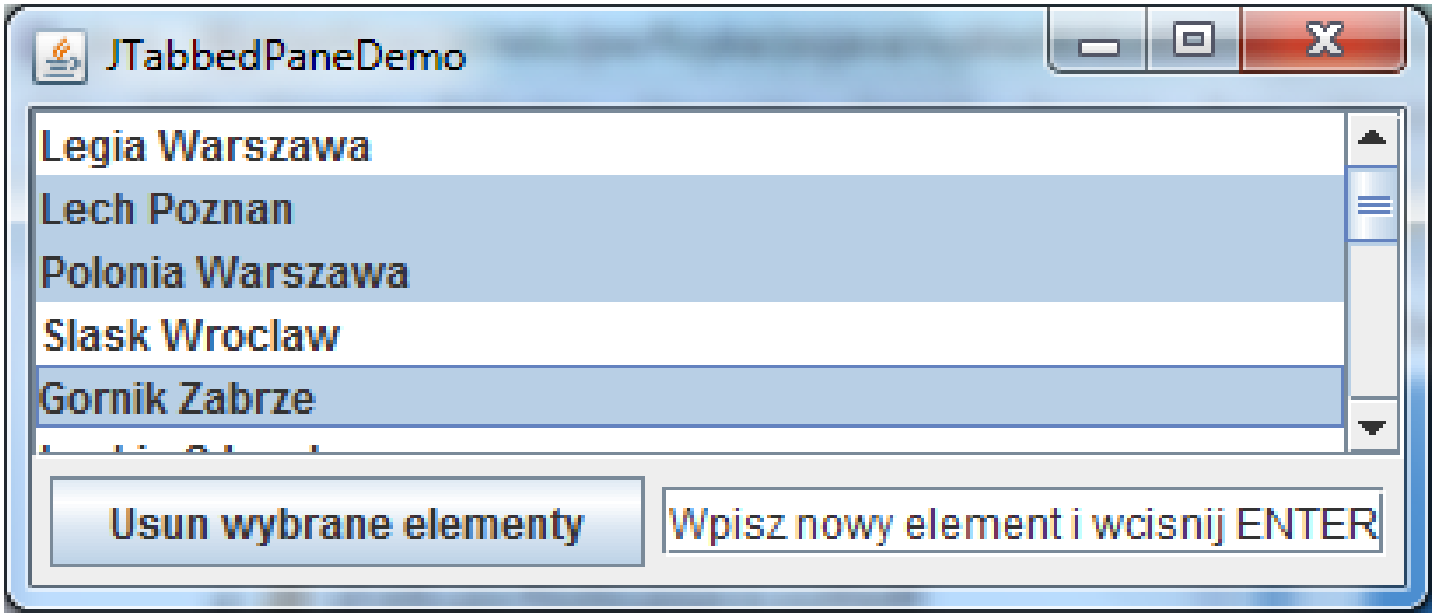


JListDemo.java

```
String[] nazwyDruzyn = { "Legia Warszawa",  "Lech Poznan",  "Polonia Warszawa",
                         "Slask Wroclaw",  "Gornik Zabrze",  "Lechia Gdansk",
                        "Zaglebie Lubin",  "Piast Gliwice",  "Wisla Krakow",
                        "Jagiellonia Bialystok", "Korona Kielce",
                         "Widzew Lodz", "Pogon Szczecin", "Ruch Chorzow",
                         "Podbeskidzie Bielsko-Biala", "GKS Belchatow"};

      DefaultListModel listaElementy = new DefaultListModel();
      JList lista = new JList(listaElementy);


      for (int i=0; i < nazwyDruzyn.length; i++)
             listaElementy.addElement(nazwyDruzyn[i]);

      lista.addListSelectionListener(listaListener);
      lista.setVisibleRowCount(5);

       //Dodawanie JScrollPane:
       JScrollPane listScrollPane = new JScrollPane(lista);
       listScrollPane.setPreferredSize( new Dimension(300,100));

      //add(lista, BorderLayout.CENTER);
       add(listScrollPane, BorderLayout.CENTER);
```

JListDemo.java

```
ListSelectionListener listaListener = new ListSelectionListener() {

        public void valueChanged(ListSelectionEvent e) {
                if (e.getValueIsAdjusting()) return;
                System.out.println("Zaznaczone elementy listy:");
                for (Object wybrane : lista.getSelectedValuesList())
                        System.out.println(wybrane);
                }
};
```



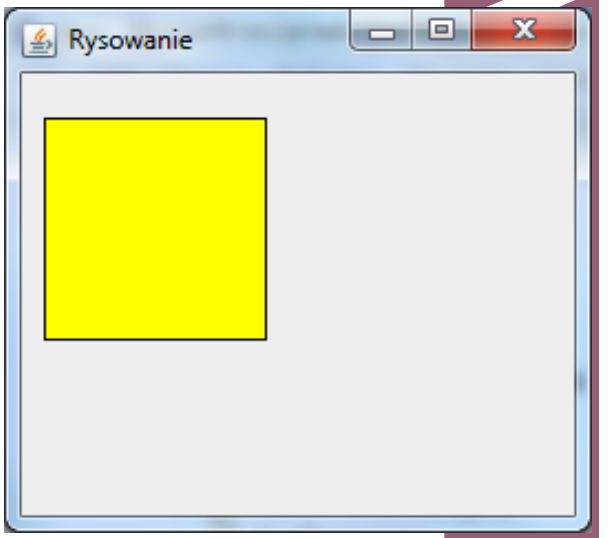
JListDemo.java

```
JButton usunElementy = new JButton("Usun wybrane elementy");
JTextField poleDodawania = new JTextField("Wpisz nowy element i wcisnij ENTER");

usunElementy.addActionListener(usuwanieElementow);
poleDodawania.addActionListener(dodawanieElementow);

ActionListener usuwanieElementow = new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
                for (Object wybrane : lista.getSelectedValuesList())
                        listaElementy.removeElement(wybrane);
        }
};

ActionListener dodawanieElementow = new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
                listaElementy.addElement(poleDodawania.getText());
        }
};
```

Rysowanie
PaintComponent i MouseMotionListener

Rysowanie.java

Każdy komponent posiada metodę paintComponent(Graphics g), którą można zmodyfikować w klasach pochodnych, np.:




```
class MyPanel extends JPanel {
    private int squareX = 10;
    private int squareY = 20;
    private int squareW = 100;
    private int squareH = 100;
     public Dimension getPreferredSize() {
             return new Dimension(250,200);
     }


    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.YELLOW);
        g.fillRect(squareX,squareY,squareW,squareH);
        g.setColor(Color.BLACK);
        g.drawRect(squareX,squareY,squareW,squareH);
    }
}
```

Rysowanie.java

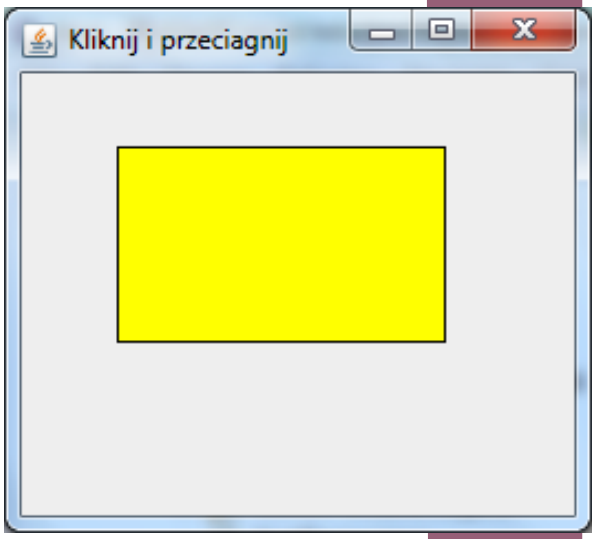
```
public Dimension getPreferredSize() {
        return new Dimension(250,200);
}
```

- Metoda getPreferredSize() została przedefioniowana, żeby ustalać preferowany rozmiar tworzonego panelu rysowania (wykorzystywane przez niektórych zarządców rozmieszczenia komponentów – *layout managers*)

RysowanieMysz.java

- Dodając do tak stworzonego panelu interfejs obsługi zdarzeń myszy można rysować prostokąty o rozmiarach definiowanych myszą

```
public MyPanel() {
    addMouseListener(new MouseAdapter() {
        public void mousePressed(MouseEvent e) {
            squareX = e.getX();
            squareY = e.getY();
           repaint();
        }
    }
}
```



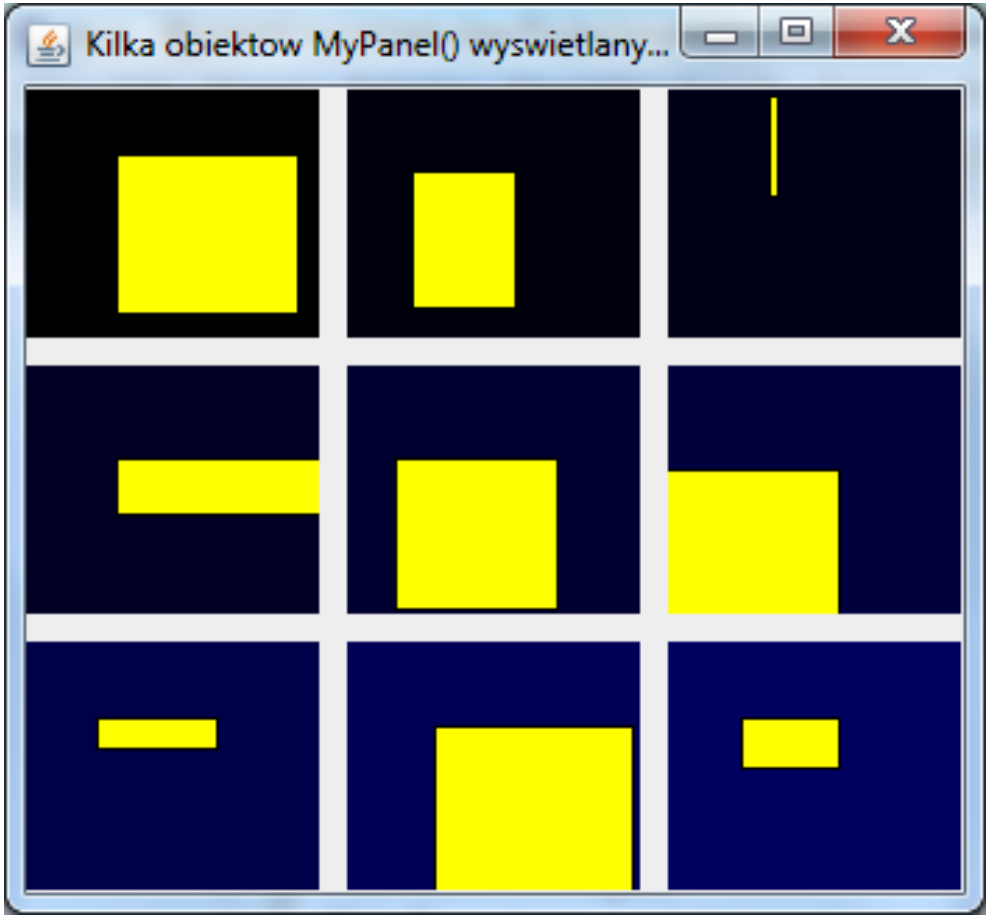
RysowanieMysz.java

- Dodając obsługę „puszczenia” klawisza myszy można definiować rozmiar komponentu:

```
addMouseListener(new MouseAdapter() {
    public void mousePressed(MouseEvent e) {
        squareX = e.getX();
        squareY = e.getY();
    }

    public void mouseReleased(MouseEvent e) {
       if (e.getX() > squareX) squareW = e.getX() - squareX;
               else {
                       squareW = squareX - e.getX();
                       squareX = e.getX();
                   }
       if (e.getY() > squareY) squareH = e.getY() - squareY;
                       else {
                       squareH = squareY - e.getY();
                       squareY = e.getY();
                       }
                   repaint();
               }
});
```

RysowanieMysz2.java

- Tak utworzony panel z opcją rysowania przy pomocy myszy jest traktowany jak każdy inny komponent Swing, może być wielokrotnie wykorzystany:



RysowanieMysz2.java

```
JFrame f = new JFrame("Kilka obiektow MyPanel() wyswietlanych
                                                    jednoczesnie");
 f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 f.setLayout(new GridLayout(3,3,10,10));

 MyPanel[] panele = new MyPanel[9];

 for (int i=0; i<9; i++){
        panele[i] = new MyPanel();
        panele[i].setBackground(new Color(i*12));
        f.add(panele[i]);
 }

 f.pack();
```

RysowanieMysz3.java

Poprzedni przykład można zmodyfikować tak, aby rysowane było kilka elementów, których współrzędne są przechowywane w tablicach:

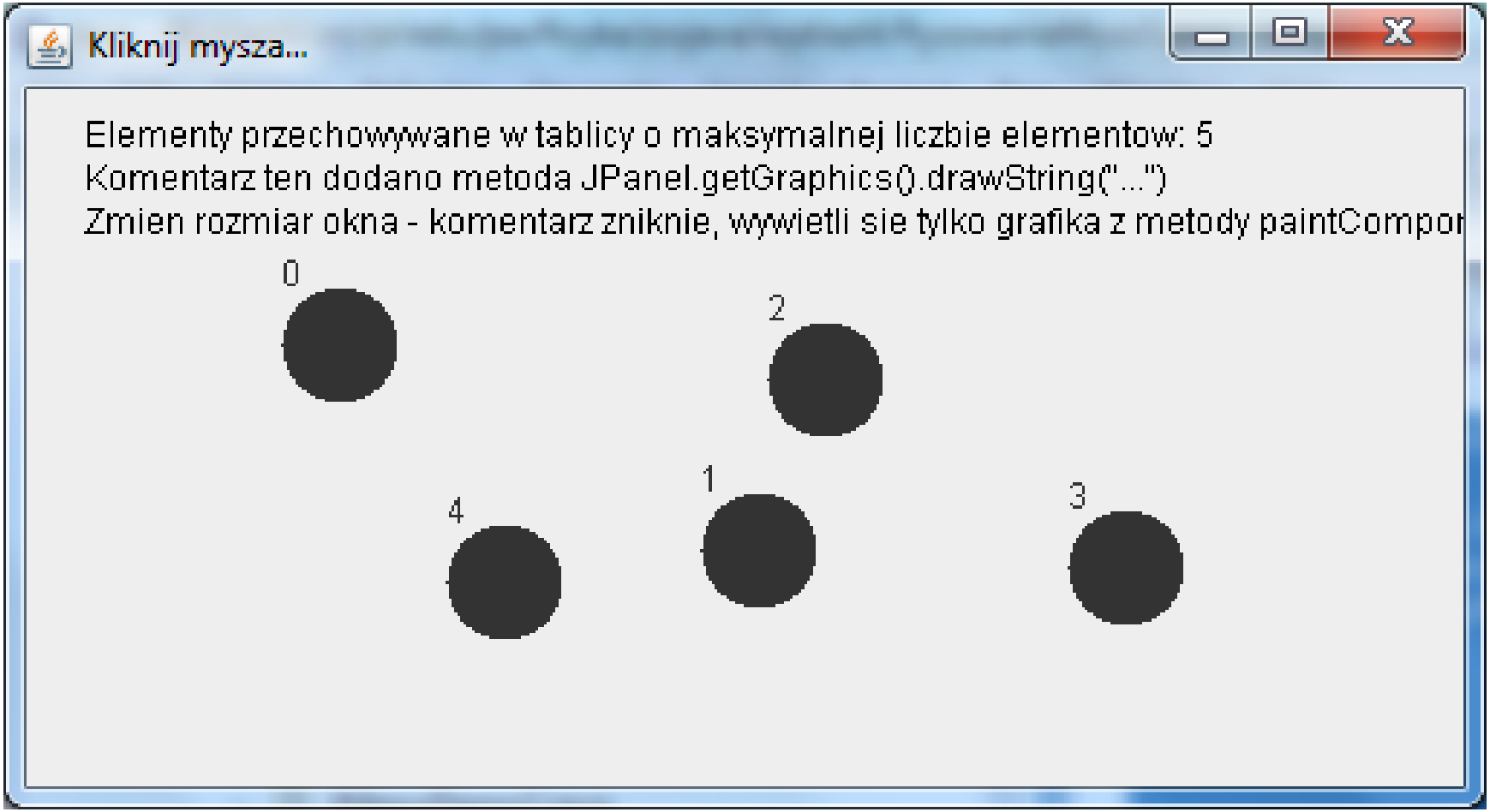
```
class MyPanel3 extends JPanel {
    private int MAKSYMALNA_LICZBA_ELEMENTOW = 5;
        private int[] x = new int [MAKSYMALNA_LICZBA_ELEMENTOW];
        private int[] y = new int [MAKSYMALNA_LICZBA_ELEMENTOW];
        private int licznikKlikniec = 0;

   // (... konstrukor i inne metody...)

    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
       for (int i = 0; i<licznikKlikniec; i++){
            g.fillOval(x[i], y[i], 40, 40);
            g.drawString(""+i, x[i], y[i]);
        }
    }

}
```

RysowanieMysz3.java

Poprzedni przykład można zmodyfikować tak, aby rysowane było kilka elementów, których współrzędne są przechowywane w tablicach:

```
public MyPanel3() {

        addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
               if (licznikKlikniec < MAKSYMALNA_LICZBA_ELEMENTOW){
                       x[licznikKlikniec] = e.getX();
                       y[licznikKlikniec] = e.getY();
                       licznikKlikniec++;
                       repaint();
               }
               else{
                       //Przekroczona dopuszczalna ilosc klikniec..
                    }

               if (e.getButton() == MouseEvent.BUTTON3) wyczyscElementy();
               //obsluga prawego klawisza myszy
               }
         });
 }
```

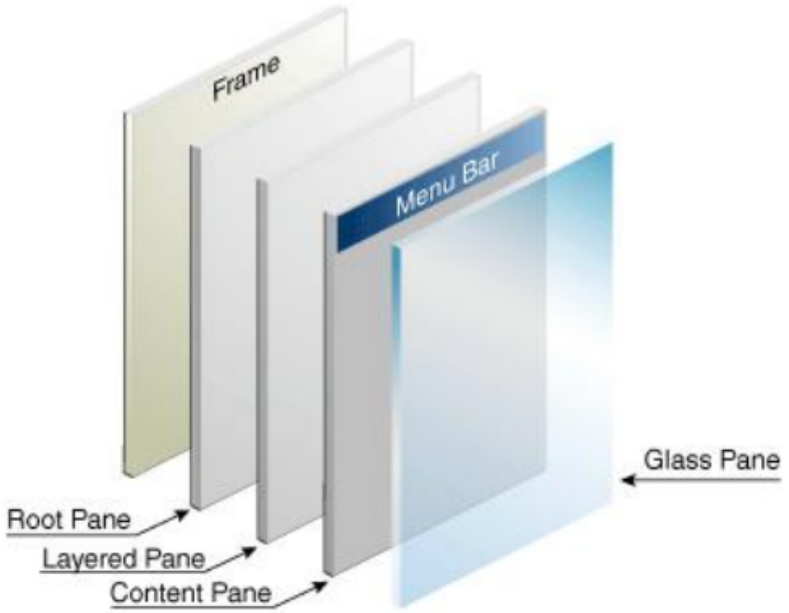
RysowanieMysz3.java

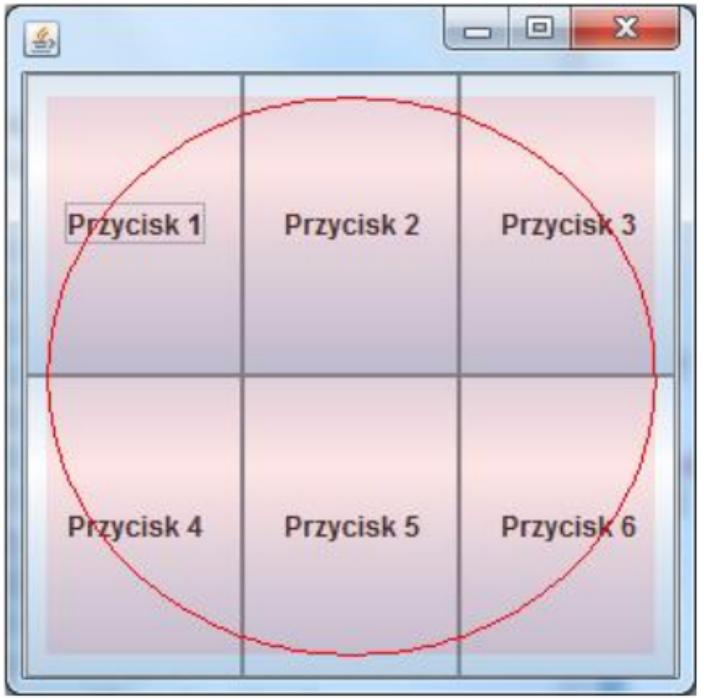


Ciekawostki

GlassPane

GlassPane można wykorzystać do uzyskiwania ciekawych efektów





- Przykład: ProsteRysowanieGlass.java

Dziękuję za Uwagę!

Do zobaczenia za tydzień.
Będziemy mówić o plikach i strumieniach
+
Ładowanie zewnętrznych bibliotek
oraz rysowanie wykresów