Języki i metody programowania – Java
INF302W
Wykład 3 (część 1)

Autor

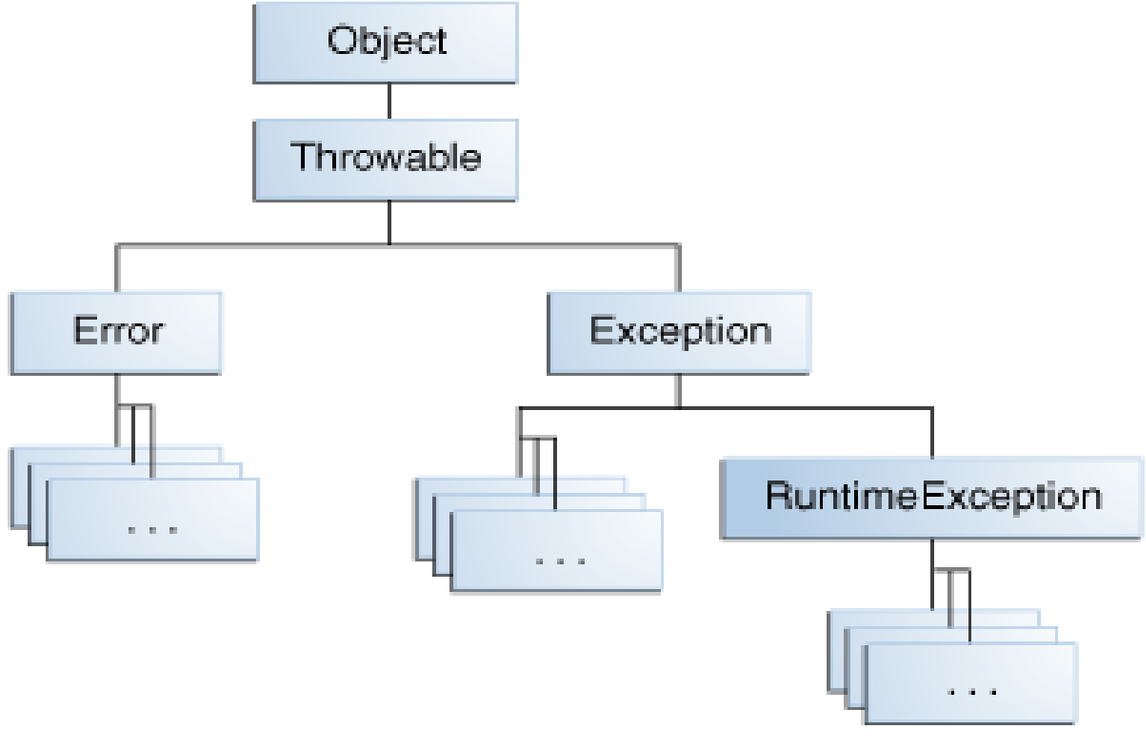
Dr inż. Zofia Kruczkiewicz

STRUKTURA WYKŁADU

1. Wyjątki, obsługa wyjątków (*try—catch, throw, throws finally*), w programach użytkowników (R-1)

WYJĄTKI

Definicja: **wyjątek jest zdarzeniem (błędem)**, które występuje podczas wykonywania programu, który zakłóca normalny przepływ instrukcji programu.



Klasy dziedziczące po klasie *Exception* są typami wyjątków, które automatycznie obsługuje Java lub program przy jawnym przechwytywaniu i obsłudze wyjątków, wywołanymi odpowiadającymi błędami, w blokach **try...catch**

- Klasa ***Error*** określa wyjątki, które **nie powinny być przechwytywane przez program**
- Do obsługi wyjątków, typów pochodnych klasy **Exception**, używa się następujących słów kluczowych: **try...catch**, **throw**, **throws**, **finally**

Rodzaje wyjątków

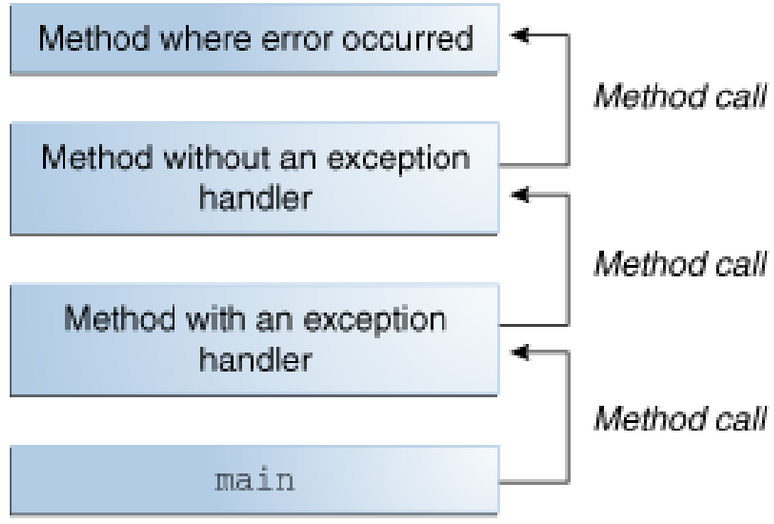
1. **checked exception – obowiązkowo sprawdzane błędy** za pomocą obsługi wyjątków, ponieważ nie ma innego sposobu, aby im zapobiegać np **java.io.FileNotFoundException**
2. **error -** błędy powodowane niepoprawnym działaniem np sprzętu; **java.io.IOError**
3. **runtime exception –** błędy wewnętrzne programu np **NullPointerException. Należy poprawić kod programu w celu wyeliminowania tych błędów** np stosując instrukcje warunkowe. **Są to błędy, które nie muszą być obowiązkowo sprawdzane** za pomocą obsługi wyjątków**.**

Uwagi:

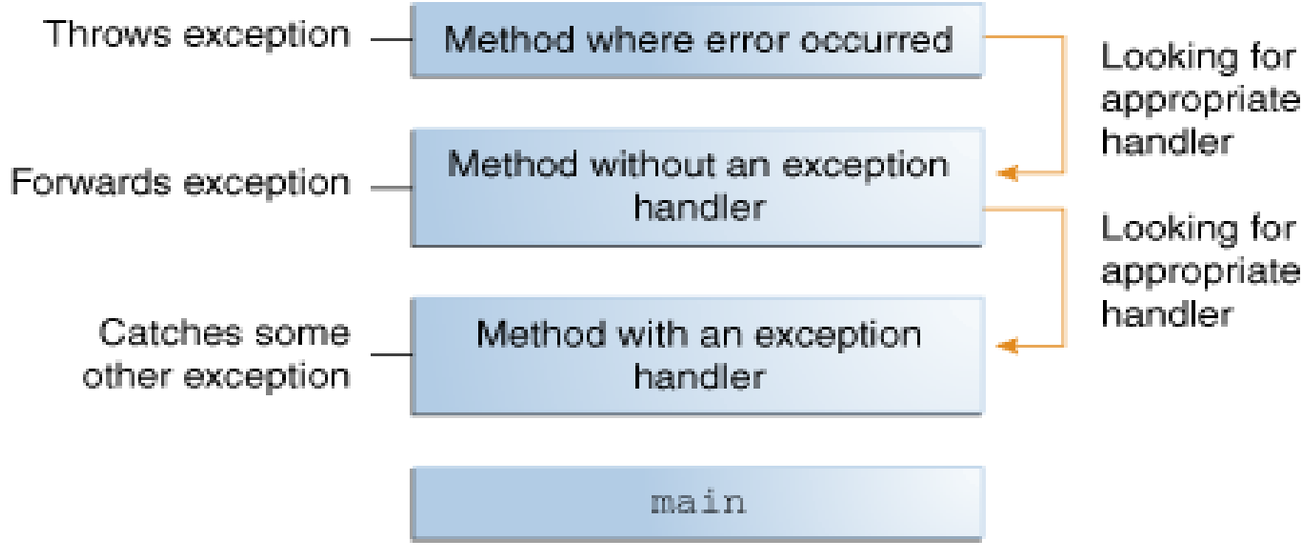
1. **Obsługa wyjątków jest czasochłonna i prowadzi do dużego zapotrzebowania na zasoby systemu.**
2. **Wyjątki rodzaju 1 są identycznie obsługiwane jak wyjątki rodzaju 3. Do celów edukacyjnych zastosowano głównie wyjątki rodzaju 3 (oprócz przykładu z p. 12)**

Obsługa wyjątków

1) Łańcuch wywołań metody, w której wystąpił błąd



2) Poszukiwanie obsługi błędów



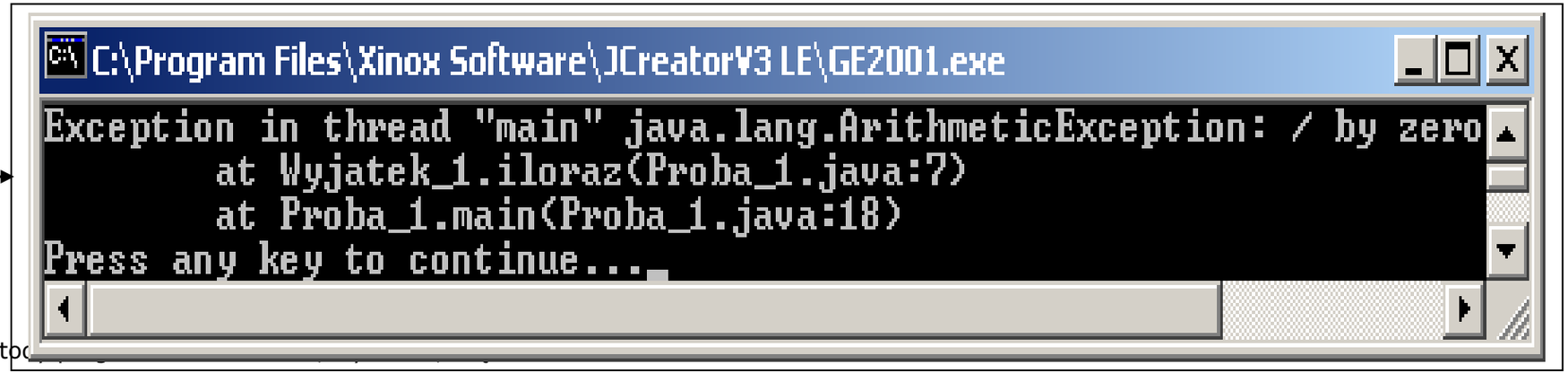
1) Obsługa wyjątków przez platformę SE Javy – przerwanie programu

```
class Wyjatek_1                                    //plik Proba_1.java
{     int x;
       Wyjatek_1(int x_)
            {     x = x_;    }
      int iloraz()
            {     int p = 45/x;          // możliwość generowania wyjątku od dzielenia przez 0  i
                  return p;}             // przerwanie programu
      int podaj_x()
            {     return x; }
}
```


```
public class Proba_1
{
  public static void main(String ags[])
 {
   Wyjatek_1 w1= new Wyjatek_1(0);          //3 –poprawna wartość
   System.out.println("45/"+w1.podaj_x()+" ="+w1.iloraz());
 }
}
```

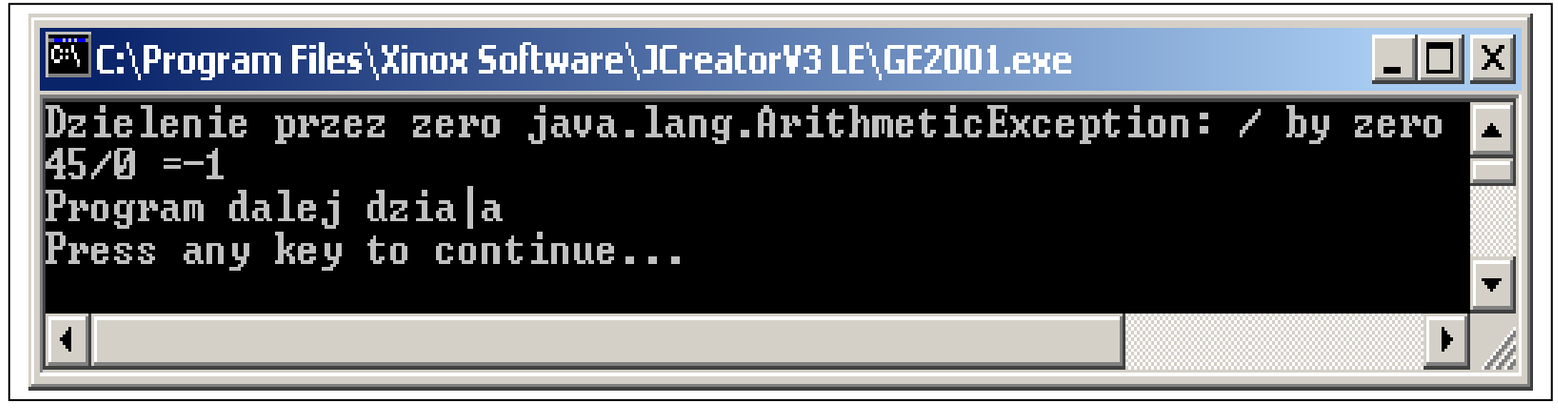
Normalne zakończenie programu

Zakończenie programu po wystąpieniu błędu dzielenia przez 0



2) Przechwytywanie wyjątków przez program – kontynuowanie programu po obsłudze wyjątku – blok try... catch

```
class Wyjatek_2                                    //plik Proba_2.java
{  int x;
   Wyjatek_2(int x_)
     {    x = x_;    }
   int iloraz()
     {    int p = -1;
          try
           {   p=45/x; }                  //możliwość generowania wyjątku od dzielenia przez 0
          catch( ArithmeticException e)   //przechwycenie wyjątku
           {   System.out.println("Dzielenie przez zero "+e); }
          return p;                       //kontynuacja programu
      }
int podaj_x()
     {    return x; }
}

public class Proba_2
{
 public static void main(String ags[])
 { Wyjatek_2 w1=new Wyjatek_2(0);
   System.out.println("45/"+w1.podaj_x()+" ="+w1.iloraz());  // wystąpienie i obsługa wyjątku
   System.out.println("Program dalej działa");
 }}
```



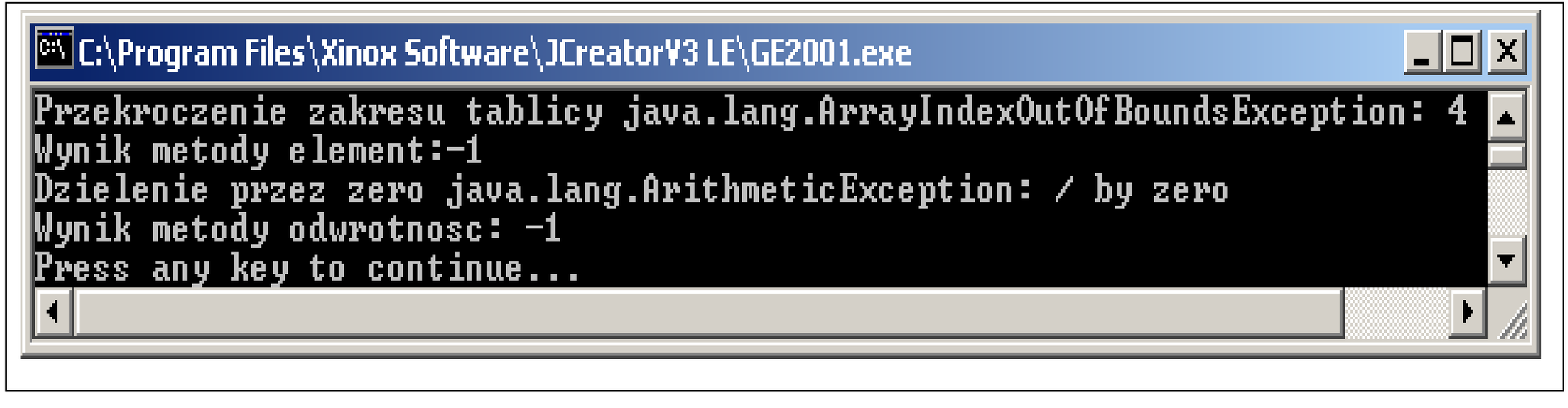
Budowa bloku try i catch

```
try
  {
        //blok try śledzenia wystąpienia błędów (wyjątków)
        //w zastosowanych instrukcjach kodu bloku.
        //Po wystąpieniu błędu przerwanie tego
        // fragmentu kodu i przejście do bloku catch
  }
catch( Typ_wyjatku e) //blok catch przechwycenia wyjątku
  {
        //blok catch kodu do obsługi wyjątku
        //który działa jedynie po wystąpieniu wyjatku
        //Typ_wyjątku lub jego typu pochodnego
  }

        // dalszy kod programu, który może rówież działać
        //po wystąpieniu wyjątku
```

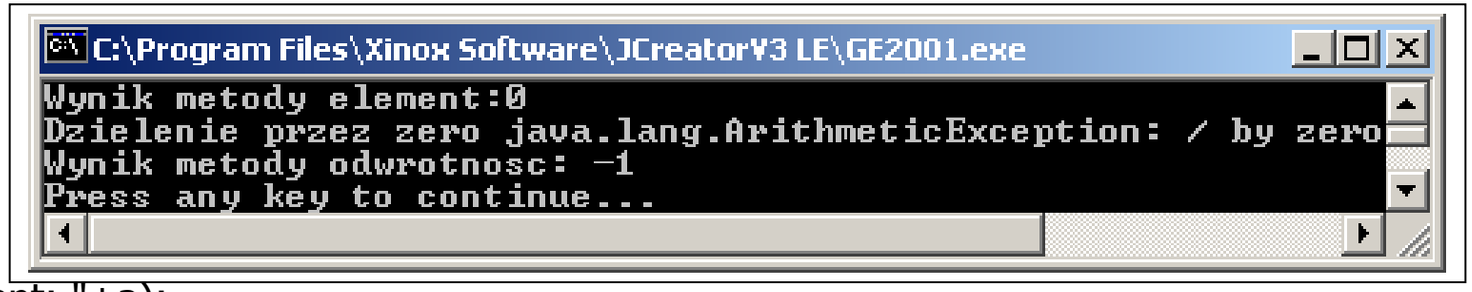
3) Przechwytywanie wyjątków przez program w różnych blokach try...catch – kontynuowanie programu po obsłudze wyjątku

```
class Wyjatek_3                                    //plik Proba_3.java
  { int x[];
    Wyjatek_3(int x_, int y_)
      { x = new int [x_];
        x[0]=y_;
      }
    int element(int p)
       {
            try
             {    int el = x[p];       //możliwość generowanie wyjątku od przekroczenia indeksu tablicy
                  return el; }
            catch( ArrayIndexOutOfBoundsException e)       //przechwycenie wyjątku
              {  System.out.println("Przekroczenie zakresu tablicy "+e); }
          return -1;
       }

   int odwrotnosc()
       {    int a=-1;
            try
              { a=1/x[0]; }                        //możliwość generowania wyjątku od dzielenia przez 0
            catch(ArithmeticException e)           //przechwycenie wyjątku
                 {    System.out.println("Dzielenie przez zero "+e); }
            return a;                              //kontynuacja programu
       }}
```

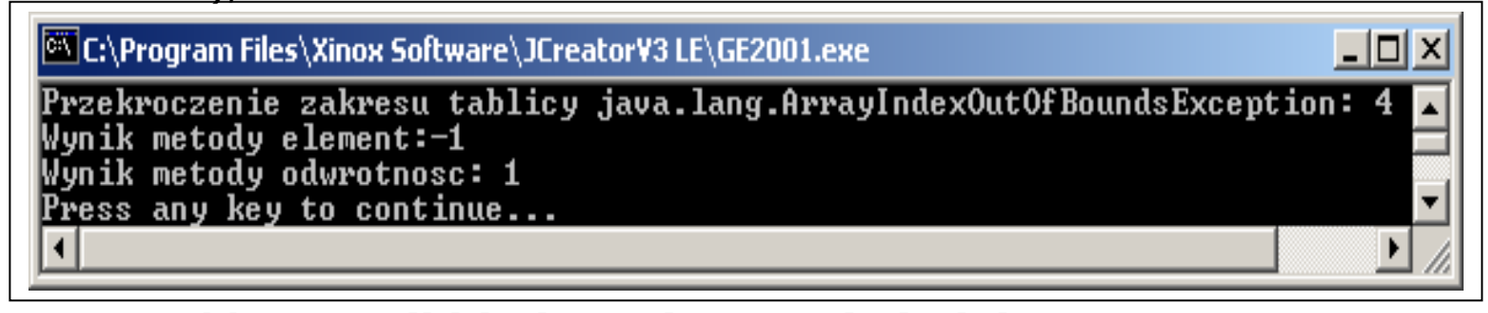
```
public class Proba_3        //1-y przypadek przebiegu programu: 2 błędy
{
 public static void main(String ags[])
  {
    Wyjatek_3 w1=new Wyjatek_3(2, 0);
                                // wystąpienie i obsługa wyjątku, gdy nastąpi próba dostępu poza tablicę
    int a=w1.element(4);
    System.out.println("Wynik metody element: "+a);
                                // wystąpienie i obsługa wyjątku, gdy nastąpi próba  dzielenia przez 0
    int b= w1.odwrotnosc();
    System.out.println("Wynik metody odwrotnosc: "+b);
  }
}
//wystąpiły dwa wyjątki: przekroczenie indeksu i dzielenie przez 0
```



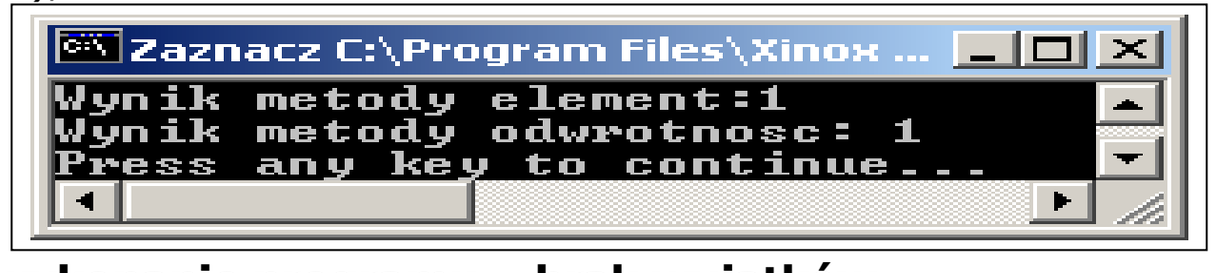
```
public class Proba_3    //2-i przypadek
{
  public static void main(String ags[])
  { Wyjatek_3 w1=new Wyjatek_3(2, 0);
    int a=w1.element(0);
       System.out.println("Wynik metody element: "+a);
    int b= w1.odwrotnosc();                          //wystąpił błąd dzielenia przez 0
       System.out.println("Wynik metody odwrotnosc: "+b);
   }
}
```




```
public class Proba_3  //3-i przypadek
{ public static void main(String ags[])
  { Wyjatek_3 w1=new Wyjatek_3(2, 1);
    int a=w1.element(4);
       System.out.println("Wynik metody element: "+a); //wystąpił błąd przekroczenia indeksu
    int b= w1.odwrotnosc();
       System.out.println("Wynik metody odwrotnosc: "+b);
   }
}
```

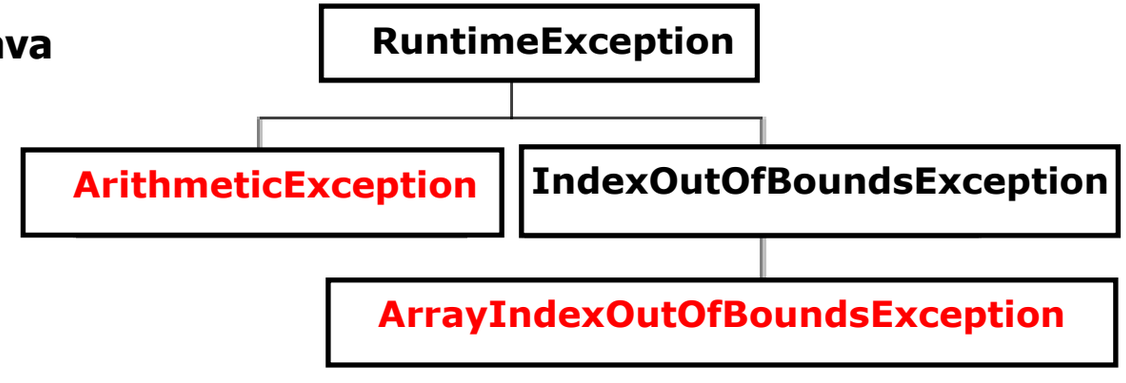



```
public class Proba_3        //4-y przypadek
{ public static void main(String ags[])
  { Wyjatek_3 w1=new Wyjatek_3(2, 1);
    int a=w1.element(0);              //normalne wykonanie programu – brak wyjątków
       System.out.println("Wynik metody element: "+a);
    int b= w1.odwrotnosc();
       System.out.println("Wynik metody odwrotnosc: "+b);   }}
```



4) Przechwytywanie 1 z wielu wyjątków w jednym bloku try przez jeden z wielu bloków catch – kontynuowanie programu po obsłudze wyjątku

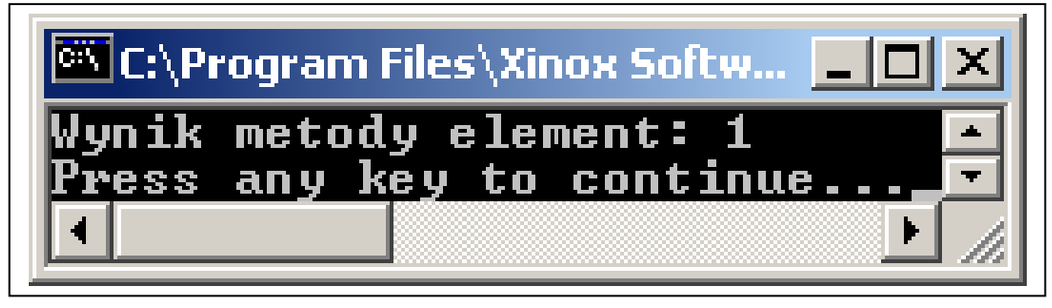
- Blok **try** zawiera wiele instrukcji, które generują więcej, niż jeden wyjątek
- Konstrukcja obsługi wyjątków zawiera więcej niż jeden blok **catch**
- Wybierany jest pierwszy z bloków **catch**, dla którego typ wyjątku jest zgodny
- Klasy wyjątków w blokach **catch** nie mogą być powiązane dziedziczeniem w kolejności ich umieszczenia, ponieważ każdy wyjątek będzie obsługiwany przez blok catch dla wyjątku położonego najwyżej w drzewie dziedziczenia




```
class Wyjatek_4            //plik Proba_4.java
{  int tab[];
    Wyjatek_4(int x_, int y_)
     {     tab = new int [x_];
           tab[0]=y_; }

   int element(int p)
    {   try
         { int el=1/tab[0];          //1- możliwość generowania wyjątku od dzielenia przez 0
           tab[p]=el;                //2- możliwość generowania wyjątku od przekroczonego indeksu
           return el; }
        catch(ArithmeticException e)                          //przechwycenie wyjątku 1
              {    System.out.println("Dzielenie przez zero "+e); }
         catch(ArrayIndexOutOfBoundsException e)              //przechwycenie wyjątku 2
              {    System.out.println("Przekroczenie zakresu tablicy "+e); }
      return -1;  }
 }
```

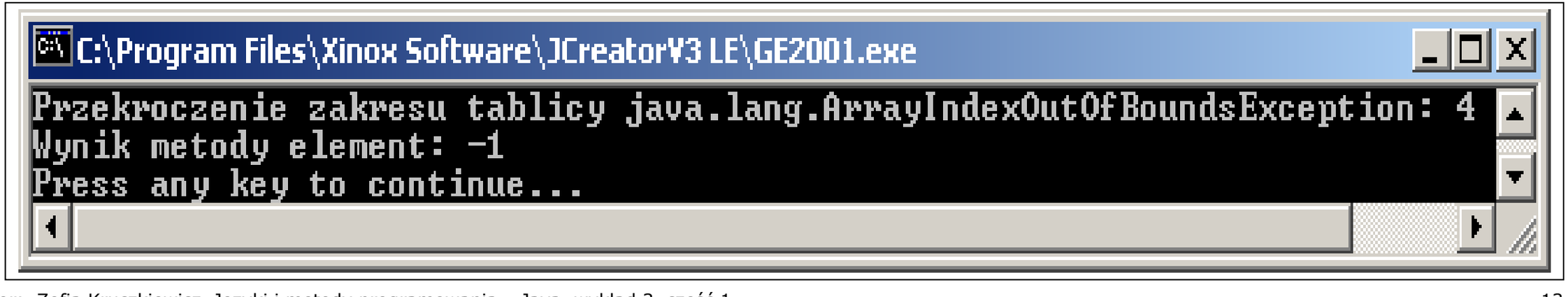
//Przypadek poprawnego wykonania programu

```
public class Proba_4
{
 public static void main(String ags[])
  {   Wyjatek_4 w1=new Wyjatek_4(2, 1);
      int a=w1.element(0);
      System.out.println("Wynik metody element: "+a);
  }
}
```

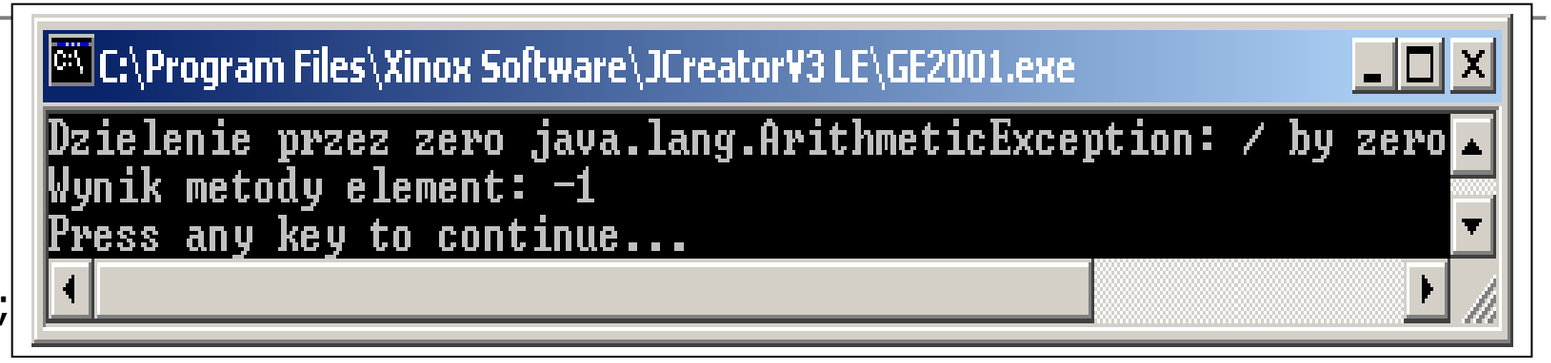


//Przypadek przekroczenia indeksu

```
 public class Proba_4
{
 public static void main(String ags[])
  {
      Wyjatek_4 w1=new Wyjatek_4(2, 1);
      int a=w1.element(4);                    // wystąpienie i obsługa wyjątku
      System.out.println("Wynik metody element: "+a);
  }
}
```



```
//Przypadek dzielenia przez 0
public class Proba_4
{
 public static void main(String ags[])
  {
     Wyjatek_4 w1=new Wyjatek_4(2, 0);
      int a=w1.element(1);
      System.out.println("Wynik metody element: "+a);      // wystąpienie i obsługa wyjątku
  }
}
```




```
//Przypadek obu błędów jednocześnie
public class Proba_4
{
 public static void main(String ags[])
  {
     Wyjatek_4 w1=new Wyjatek_4(2, 0);
       int a=w1.element(4);                          //wystąpienie 2 błędów i obsługa 1-go wyjątku
      System.out.println("Wynik metody element: "+a);      //dalszy kod bloku try jest przerwany
  }
}
```

5) Przechwytywanie 1 z wielu wyjątków w jednym bloku try przez jeden blok catch – kontynuowanie programu po obsłudze wyjątku

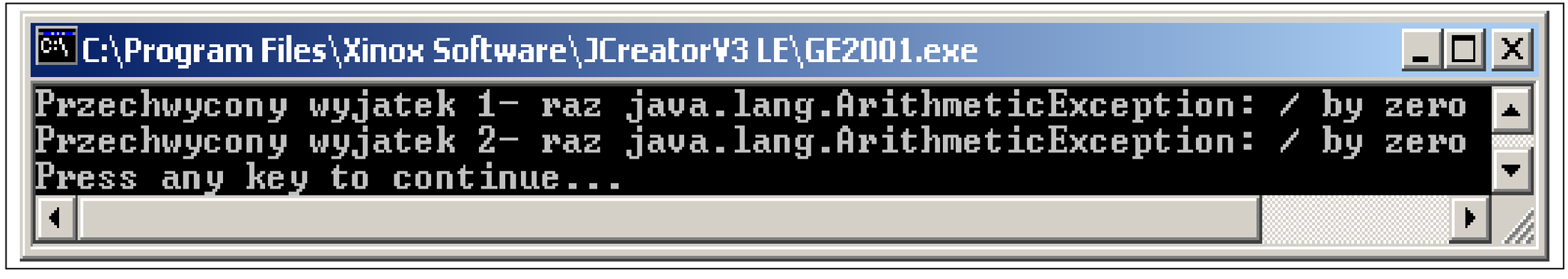
```
class Wyjatek_4_1                                          //plik Wyjatki1.java
{
    int tab[];
    Wyjatek_4(int x_, int y_) {
        tab = new int[x_];
        tab[0] = y_;
    }
   int element(int p) {
        try {
            int el = 1 / tab[0];          //1- możliwość generowania wyjątku od dzielenia przez 0
            tab[p] = el;                  //2- możliwość generowania wyjątku od przekroczonego indeksu
            return el;
     } catch (ArithmeticException|ArrayIndexOutOfBoundsException e)     //przechwycenie wyjątków
        {  System.out.println("Błąd " + e); }
     return -1;
    }
}
public class Wyjatki1 {
   public static void main(String[] args) {
        Wyjatek_4_1 w1 = new Wyjatek_4_1(2, 1);          //new Wyjatek_4_1(2, 0)
        int a = w1.element(4);                           //wystąpienie i obsługa wyjątków
        System.out.println("Wynik metody element: " + a);
    }
}
```

```
Błąd java.lang.ArrayIndexOutOfBoundsException: 4
Wynik metody element: -1
```

```
Błąd java.lang.ArithmeticException: / by zero
Wynik metody element: -1
```

6) Ponowne generowanie wyjątku („ręczne”) - kontynuowanie programu po obsłudze wyjątku

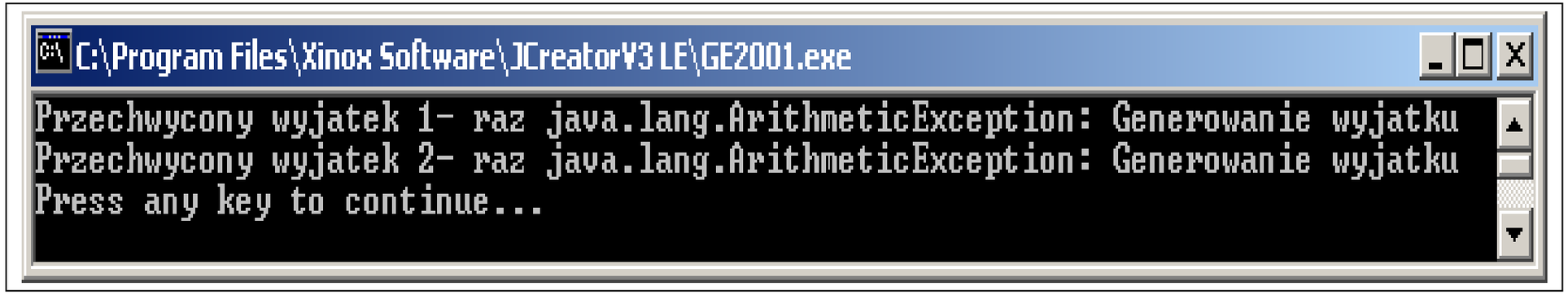
```
class Wyjatek_5                                          //plik Proba_5.java
{ static void odwrotnosc (int a)
  {   try
       { int b=1/a;  }                                   //automatyczne wywołanie wyjątku, gdy a=0
      catch (ArithmeticException e)                      //przechwycenie wyjątku od dzielenia przez 0
       {   System.out.println("Przechwycony wyjatek 1- raz "+e);
           throw e;                                      //ręczne generowanie powtórzenia wyjątku
       }
  }
}
```




```
public class Proba_5
{
   public static void main(String ags[])
       {   try
             { Wyjatek_5.odwrotnosc(0); }                 //zagnieżdżona obsługa wyjątku -  wymuszona obsługa wyjątku - throw
           catch (ArithmeticException e)
             {  System.out.println("Przechwycony wyjatek 2- raz "+e);}
       }
}
```

7) Generowanie wyjątku („ręczne”) - kontynuowanie programu po obsłudze wyjątku klauzula throw *Wystapienie_klasy_pochodnej_Throwable*

```
class Wyjatek_6                                          //plik Proba_6.java
{  static void odwrotnosc (int a)
    { try
      { if (a>1)
       throw new ArithmeticException("Generowanie wyjatku"); }      // ręczne generowanie wyjątku
     catch (ArithmeticException e)
          {     System.out.println("Przechwycony wyjatek 1- raz "+e);
                throw e;                                           //ręczne generowanie powtórzenia wyjątku
          }
    }
 }
```

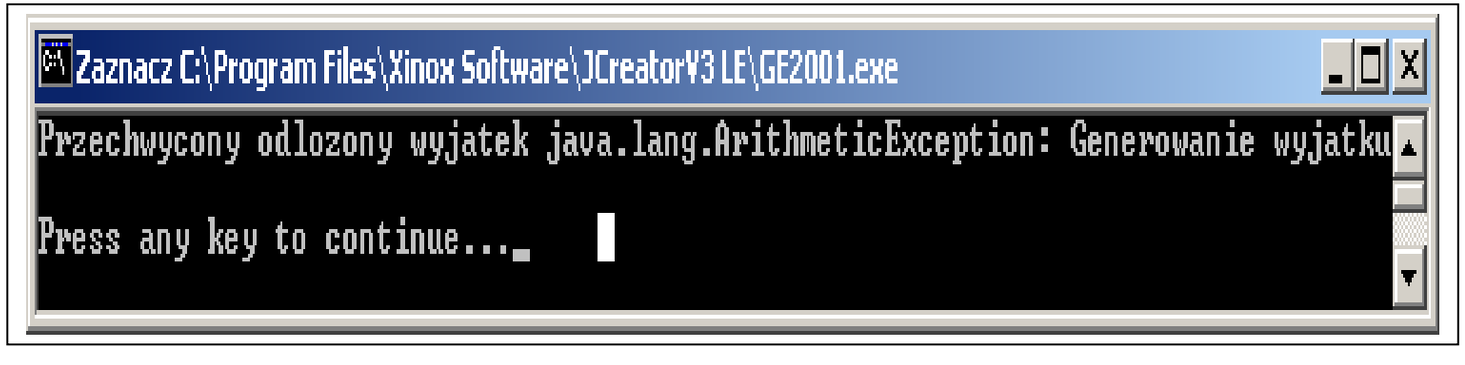



```
public class Proba_6
{
  public static void main(String ags[])
    {  try
        {  Wyjatek_6.odwrotnosc(2);   }          //zagnieżdżona 1 –a obsługa wyjątku i  2-a obsługa tego samego
                                                 // wyjątku wygenerowanego za pomocą throw
       catch (ArithmeticException e)
          {     System.out.println("Przechwycony wyjatek 2- raz "+e);}
    }
 }
```

8) Przekazanie obsługi wyjątku do innej części programu – klauzula throws

typ nazwa metody (lista_parametrów) **throws** *lista_wyjątków*

```
class Wyjatek_7                                          //plik Proba_7.java
 { static void odwrotnosc (int a )  throws Exception
    { if (a>1)
       throw new ArithmeticException ("Generowanie wyjatku"); }
 }

public class Proba_7
  { public static void main(String ags[])
    { try
       {  Wyjatek_7.odwrotnosc(2); }
      catch (Exception e)
      { System.out.println("Przechwycony odlozony wyjatek "+e); }
    }
  }
```



Zasada:
Obowiązkowa obsługa wyjątków w miejscu wywołania metody *odwrotnosc()* dotyczy **grupy wyjątków rodzaju checked exception** użytych w klauzuli **throws**. Typ tego wyjątku w bloku **catch** musi być albo identycznej klasy użytej w **throws** lub klasy, od której dziedziczy klasa wyjątku użyta w **throws**.

Zasada ta nie dotyczy pozostałych rodzajów wyjątków, czyli:

- **Error** (np. OutOfMemoryError)
- **RuntimeException** (np. ArithmeticException)

oraz dziedziczących od tych klas.

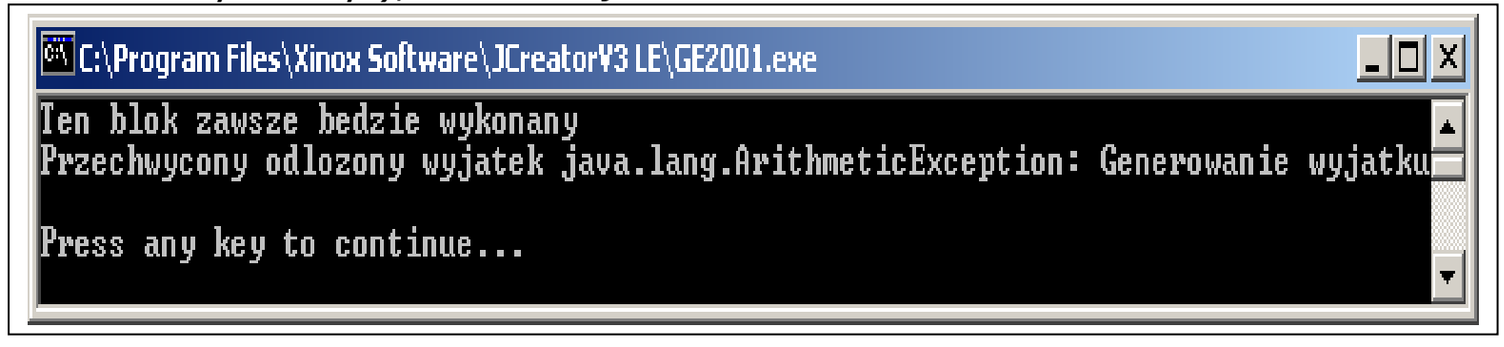
Wniosek:
Z tej zasady wynika, że klasy bazowe dla obu tyów wyjątków np. *Exception*, są wyjątkami rodzaju checked exception, obowiązkowo sprawdzanych.

9) Wyjątki generowane w bloku zagnieżdżonym w bloku try, mogą być obsłużone w jego bloku catch (str. 5, 8)

```
class Wyjatek_7_1                                    //plik Proba_7_1.java
 {
    static void odwrotnosc (int a)
      { if (a>1)
          throw new ArithmeticException("Generowanie wyjatku");
      }

    static void oblicz(int b)
      {  odwrotnosc(b); }

 }
public class Proba_7_1
 {
public static void main(String ags[])
 {  try
    {
      Wyjatek_7_1.oblicz(2);
    }
    catch (Exception e)
     {   System.out.println("Przechwycony odlozony wyjatek "+e);}
 }
                                                        }
```

10) Wykonanie wskazanej części metody po bloku try po wystąpieniu w niej wyjątku lub przy braku jego wystąpienia - klauzula finally

```
class Wyjatek_8                                          //plik Proba_8.java
  {   static void odwrotnosc (int a) throws Exception
        { try
           {  if (a>1)
              throw new ArithmeticException("Generowanie wyjatku");   }
          finally                  // (zamiast catch) wykonanie w bloku finally instrukcji po wystąpieniu wyjątku lub bez wystąpienia wyjątku
            {  System.out.println("Ten blok zawsze bedzie wykonany");          }
        }
  }
public class Proba_8  // blad a>1
{ public static void main(String args[])
   { try
        { Wyjatek_8.odwrotnosc(2);  }
      catch (Exception e)
        {   System.out.println("Przechwycony odlozony wyjatek "+e); }
     }
}
public class Proba_8  // brak bledu a=1
{ public static void main(String args[])
   { try
        { Wyjatek_8.odwrotnosc(1);  }
      catch (Exception e)
        {   System.out.println("Przechwycony odlozony wyjatek "+e); }
     }}
```



11) Wykonanie wskazanej części metody po bloku try po wystąpieniu w niej wyjątku lub przy braku jego wystąpienia - klauzule finally i catch

```
class Wyjatek_9                                          //plik Proba_9.java
 {  static void odwrotnosc (int a) throws Exception
      {
        try
           {  if (a>1) throw new ArithmeticException("Generowanie wyjatku");  }
        catch (ArithmeticException e)
          {  System.out.println("Przechwycony wyjatek 1- raz " + e);
             throw e;            //ręczne generowanie powtórzenia wyjątku
           }
        finally                  //wykonanie instrukcji z bloku finally po wystąpieniu wyjątku lub bez wystąpienia wyjątku
          {  System.out.println("Ten blok zawsze bedzie wykonany");          }
      }
}
```

```
Przechwycony wyjatek 1- raz java.lang.ArithmeticException: Generowanie wyjatku
Ten blok zawsze bedzie wykonany
Przechwycony wyjatek 2- raz java.lang.ArithmeticException: Generowanie wyjatku
```

```
public class Proba_9
{ public static void main(String args[])
   { try
        { Wyjatek_9.odwrotnosc(2);  }        // poprawny dla 1; blok finally również wykonany
     catch (Exception e)
       {   System.out.println("Przechwycony odlozony wyjatek "+e); }
   } }
```

```
Ten blok zawsze bedzie wykonany
```

12) Blok try z definicją źródeł np z deklaracją obiektów implementujących interfejsy java.lang.AutoCloseable oraz java.io.Closeable (BufferedReader, FileReader itd)

1. Gdy wystąpi **błąd w bloku try**, nastąpi **zamknięcie automatyczne źródeł**, jednak w **odwrotnej kolejności** niż przy tworzeniu w bloku try
2. Jeśli zostana dodane bloki **catch i finally,** najpierw zostaną **zamknięte źródła, a potem wykonane bloki catch i finally**

```
package wyjatki3;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Wyjatki3 {


static String readFirstLineFromFile(String path) throws IOException
   {
      try (BufferedReader br = new BufferedReader(new FileReader(path)))
        {
          return br.readLine();
        }
   }
   public static void main(String[] args) throws IOException
     {
         String s=readFirstLineFromFile("src/wyjatki3/Wyjatki3.java");
         System.out.println(s);
     }
}
```

Obowiązkowa klauzula przy braku bloków **catch** i/lub **finally** - **przekazanie obslugi wyjątku do metody wywołującej metodę** readFirstLineFromFile

Otwarcie zródel BufferedReader, FileReader w bloku try

Przekazanie obslugi wyjatku do JVM

```
run:
package wyjatki3;
BUILD SUCCESSFUL (total time: 0 seconds)
```