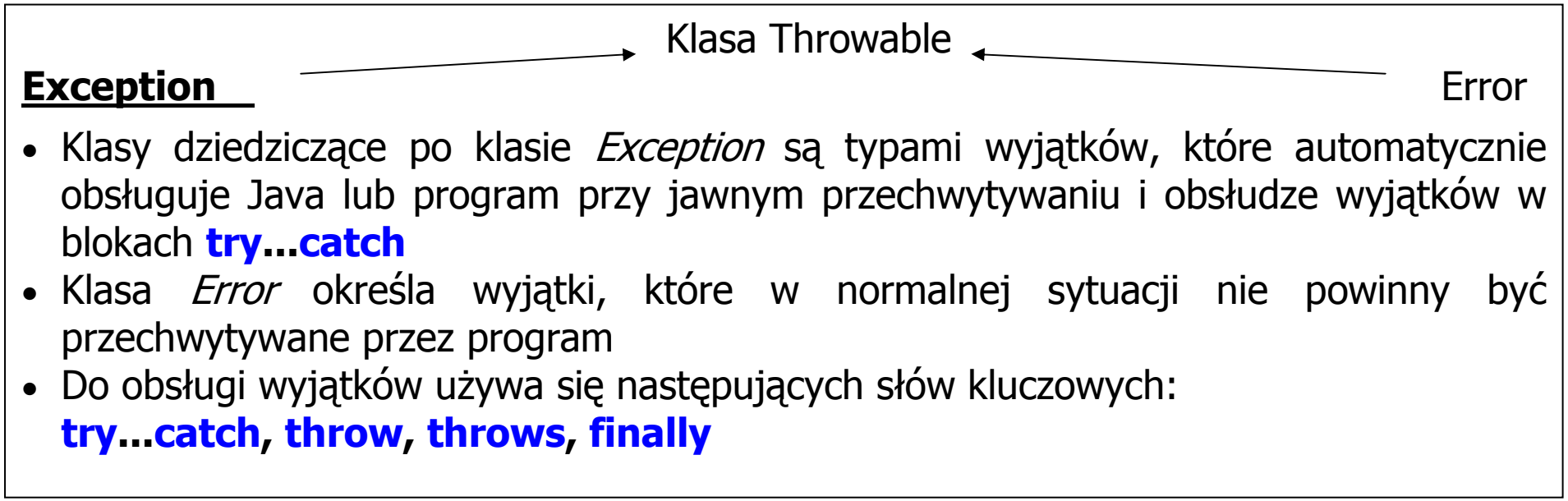
WYJĄTKI



Jeśli nie można zapobiec wystąpieniu wyjątków za pomocą instrukcji warunkowych lub wyboru, np. przy wyjątkach typu IOException (błędy we/wy), wtedy należy stosować obsługę wyjątków.

Jest ona jednak czasochłonna i prowadzi do dużego zapotrzebowania na zasoby systemu.

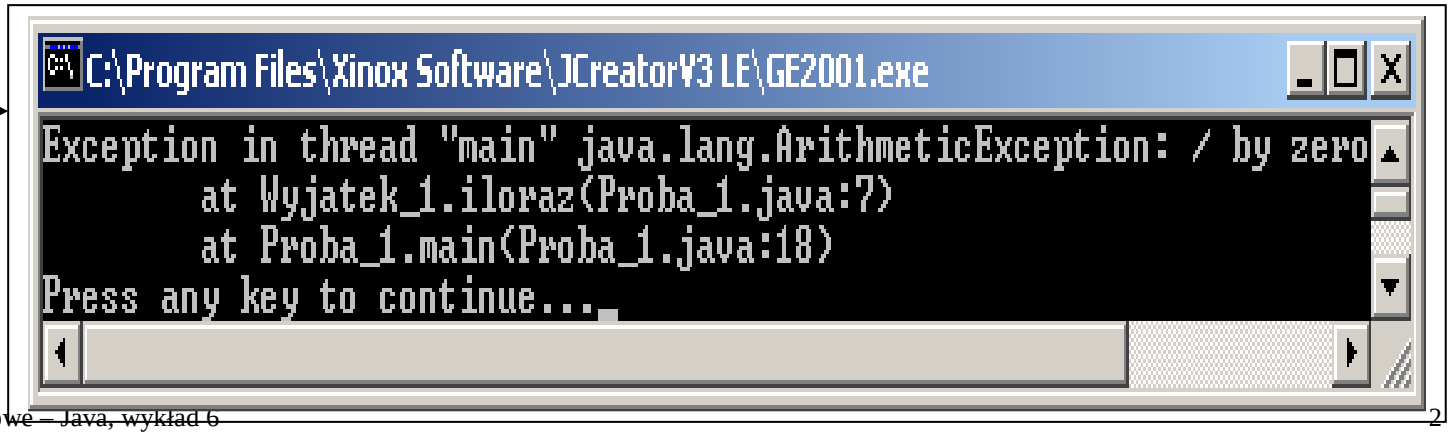
1) Obsługa wyjątków system czasu przebiegu Javy – przerwanie programu

```
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; }
}
```

```
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());
 }
}
```

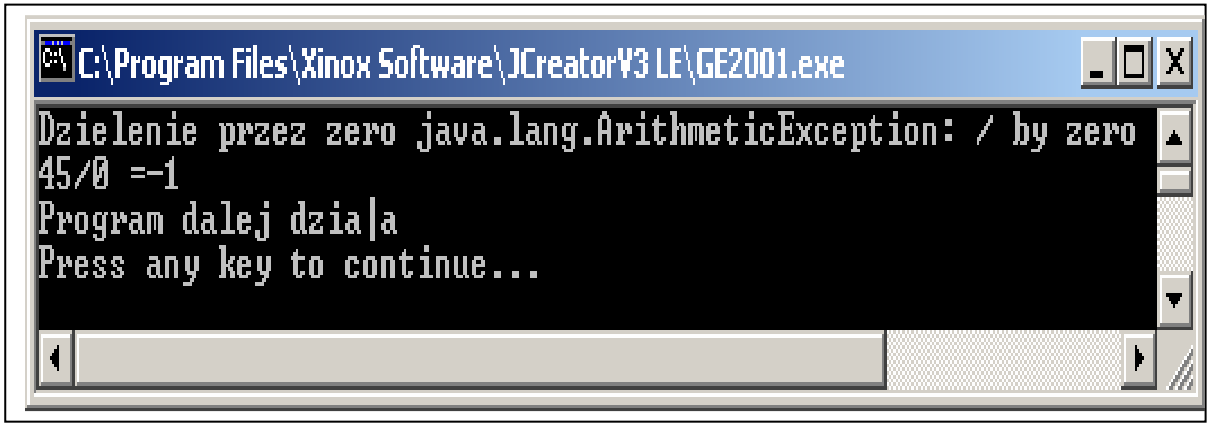
//Normalne zakończenie programu

//Zakończenie programu po
//wystąpieniu błędu dzielenia przez 0



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

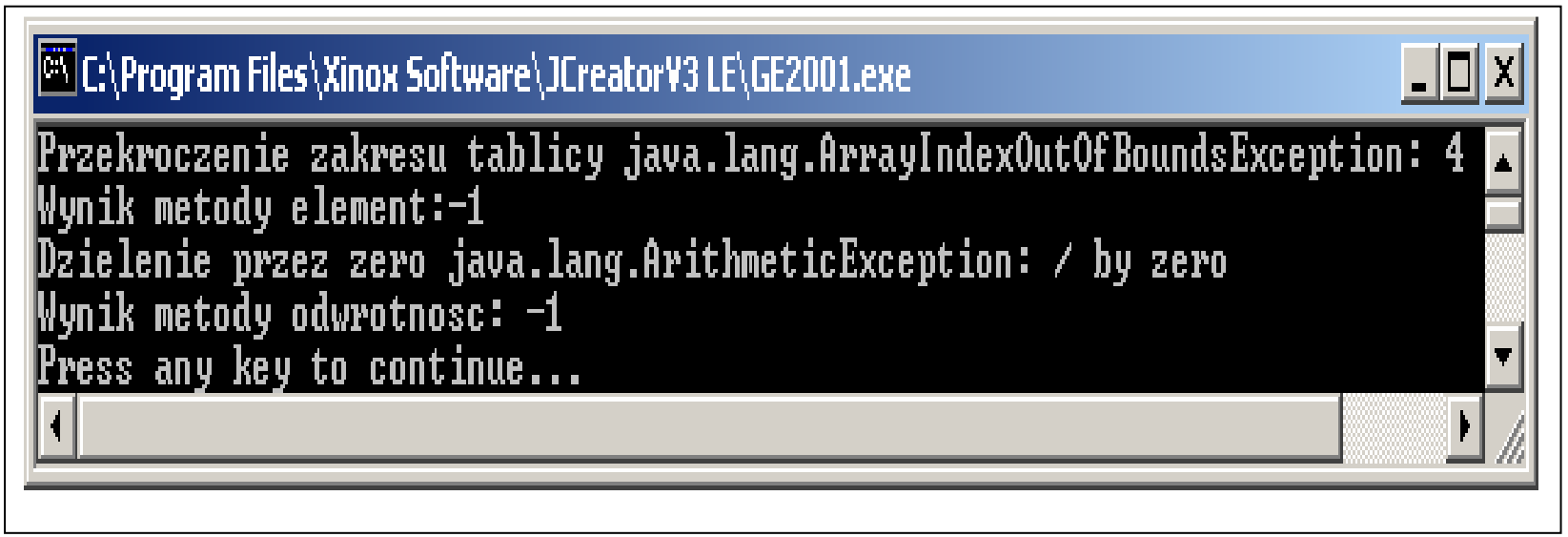
```
class Wyjatek_2                 //plik Proba_2.java
{  int x;
   Wyjatek_2(int x_)
     {    x = x_;    }
   int iloraz()
     {    int p = -1;
          try
            {   p=45/x;   }                    //możliwość generowania wyjątku od dzielenia przez 0
          catch( ArithmeticException e)        //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");
 }}
```



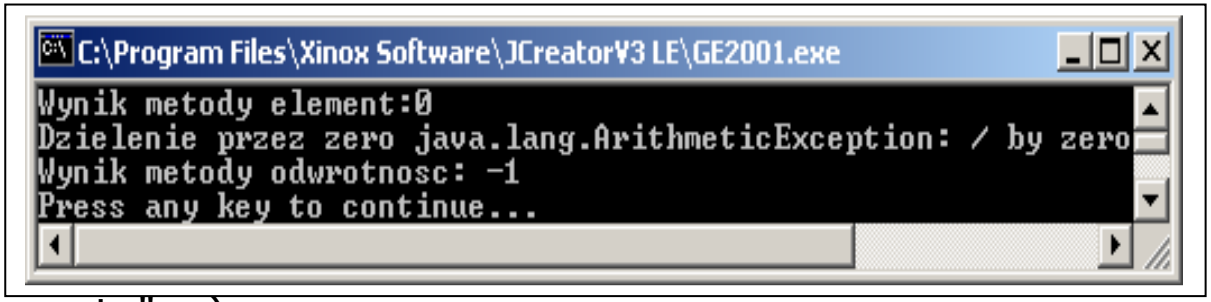
3) Przechwytywanie wyjątków przez program w różnych blokach try...catch – kontynuowanie programu po obsłudze wyjątku

```
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
      }}
```

```
class Proba_3
{
 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
```



```
class Proba_3
{
  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();
       System.out.println("Wynik metody odwrotnosc: "+b);
   }
}
```

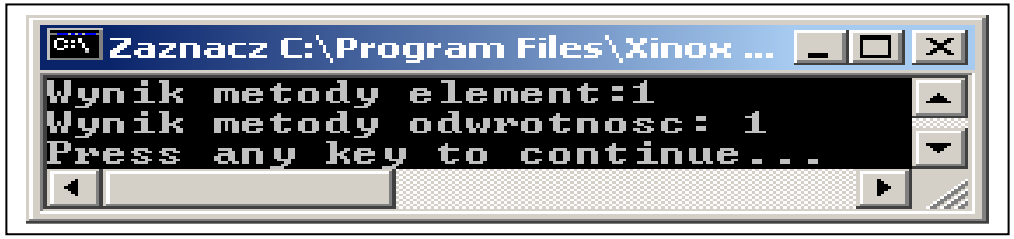


//wystąpił błąd dzielenia przez 0

```
class Proba_3
{ 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);
    int b= w1.odwrotnosc();
       System.out.println("Wynik metody odwrotnosc: "+b);
   }
}
```

//wystąpił wyjątek przekroczenia indeksu

```
class Proba_3
{ public static void main(String ags[])
  { Wyjatek_3 w1=new Wyjatek_3(2, 1);
    int a=w1.element(0);
       System.out.println("Wynik metody element: "+a);
    int b= w1.odwrotnosc();
       System.out.println("Wynik metody odwrotnosc: "+b);   }}
```



//normalne wykonanie programu – brak wyjątków

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

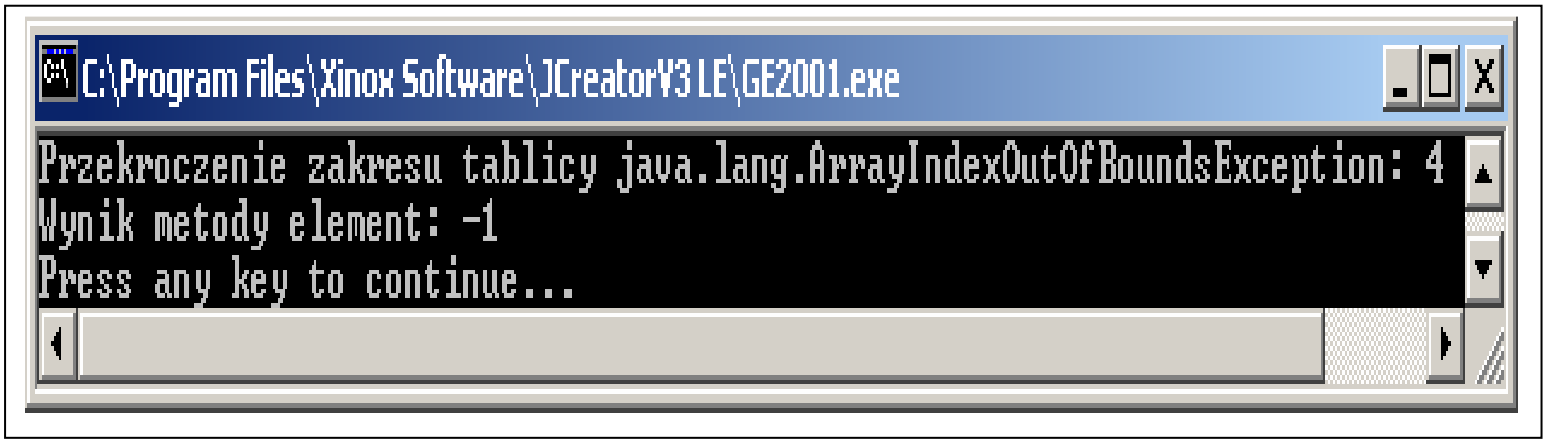
- Blok **try** zawiera wiele instrukcji, które generują więcej, niż jeden wyjątek
- Konstrukcja obsługi wyjątków zawiera więcej niż jeden blok **catch**
- Wybierany jest pierwszy z bloków **catch**, dla którego typ wyjątku jest zgodny
- Klasy wyjątków w blokach **catch** nie mogą być powiązane dziedziczeniem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

Przykład

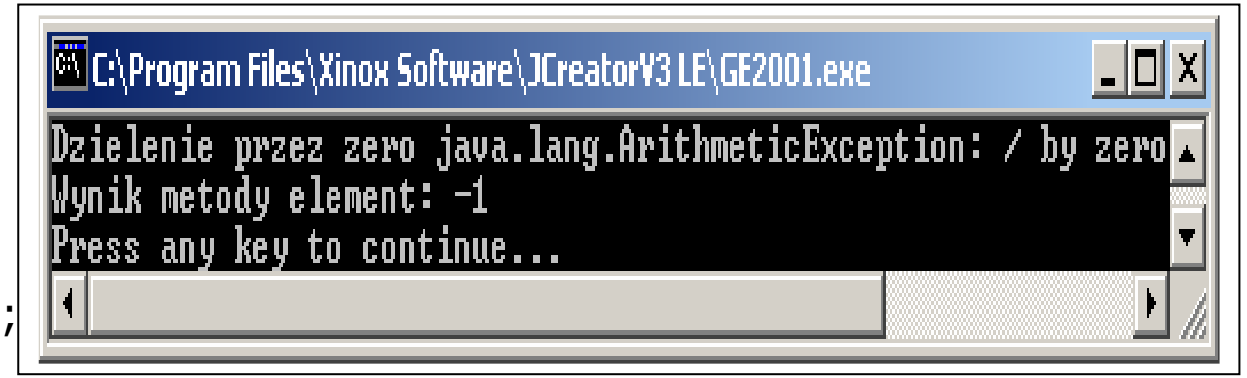
```
class Wyjatek_4                 //plik Proba_4.java
{  int x[];
    Wyjatek_4(int x_, int y_)
     {      x = new int [x_];
            x[0]=y_; }
    int element(int p)
     {   try
          { int el=1/x[0];      //1- możliwość generowania wyjątku od dzielenia przez 0
            x[p]=el;            //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);                          // wystąpienie i obsługa wyjątku
      System.out.println("Wynik metody element: "+a);
  }
}
```

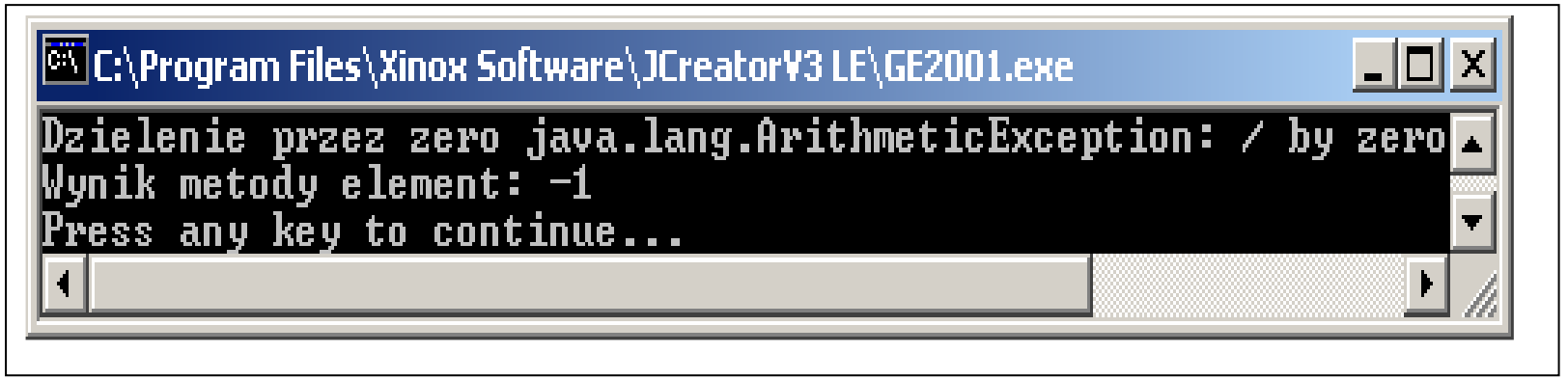

```
//Przypadek przekroczenia indeksu
 public class Proba_4
{
 public static void main(String ags[])
  {
      Wyjatek_4 w1=new Wyjatek_4(2, 1);
      int a=w1.element(4);                          // wystąpienie i obsługa wyjątku
      System.out.println("Wynik metody element: "+a);
  }
}
```



```
//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
  }
}
```

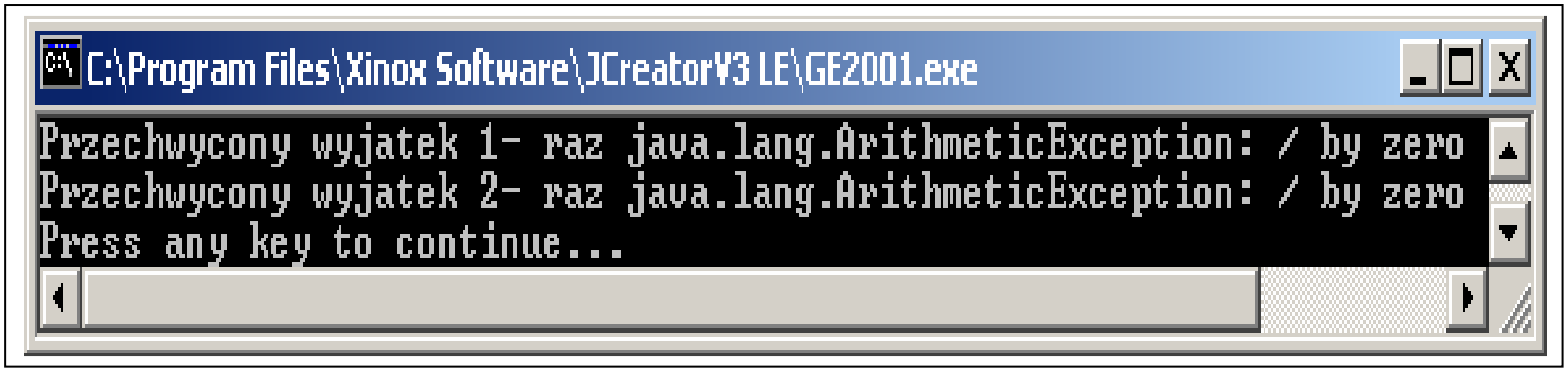


```
//lub obu wyjątków jednocześnie
public class Proba_4
{
 public static void main(String ags[])
  {
     Wyjatek_4 w1=new Wyjatek_4(2, 0);
       int a=w1.element(4);                  //wystąpienie i obsługa wyjątku
      System.out.println("Wynik metody element: "+a);
  }
}
```



5) 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
       }
  }
}
```

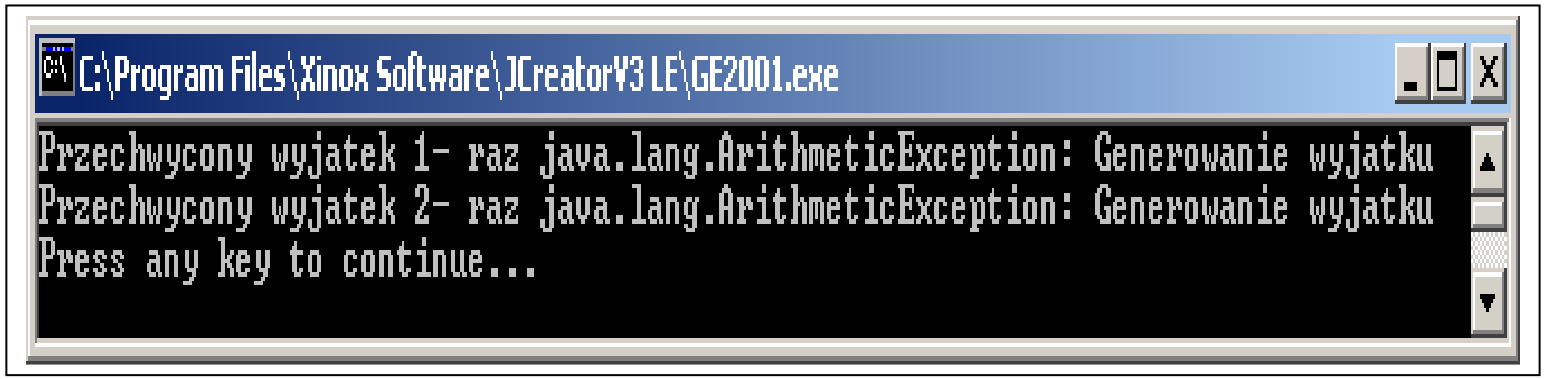



```
public class Proba_5
{
   public static void main(String ags[])
       {   try
             { Wyjatek_5.odwrotnosc(0); }//zagnieżdżona obsługa wyjątku -  wymuszona obsługa wyjątku - throw
                catch (ArithmeticException e)
             {   System.out.println("Przechwycony wyjatek 2- raz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

6) 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

  }
}
```

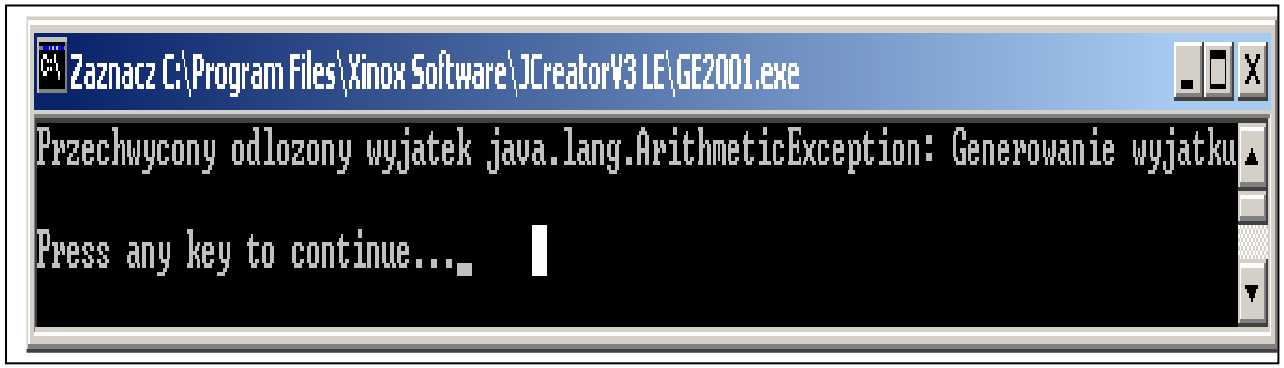



```
public class Proba_6
{
   public static void main(String ags[])
    {  try
        {  Wyjatek_6.odwrotnosc(2);   }      //zagnieżdżona obsługa wyjątku - wymuszona obsługa wyjątku - throw
       catch (ArithmeticException e)
           {     System.out.println("Przechwycony wyjatek 2- raz "+e);}
    }
 }
```

7) 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); }
      }
    }
```



Obowiązkowa obsługa wyjątków w miejscu wywołania metody *odwrotnosc()* dotyczy grupy wyjątków bezpośrednich (*explicite exception* użytych w klauzuli **throws**. Typ 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 wyjątków typu pośredniego (*implicit exception*) , czyli:

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

oraz dziedziczących od tych klas.

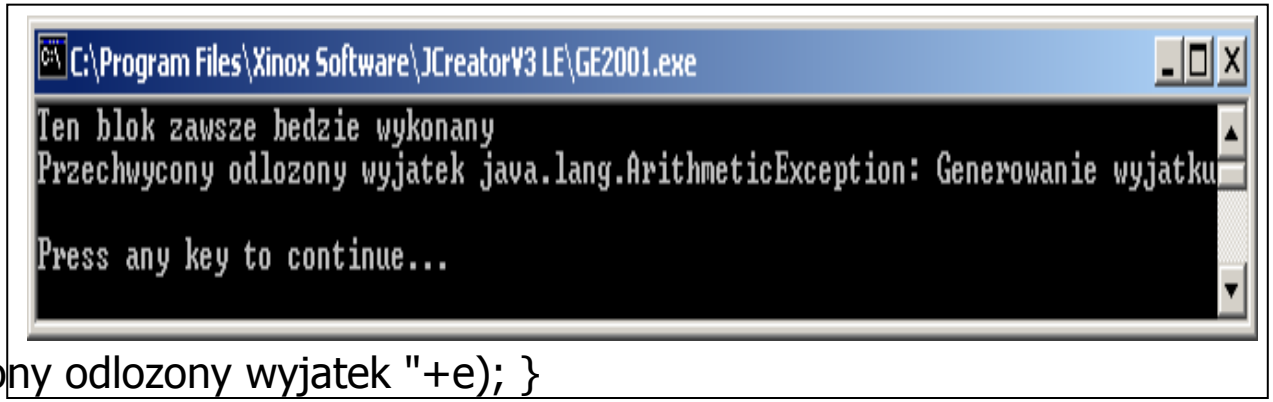
Z tej zasady wynika, że klasy bazowe dla obu typów wyjątków np. *Exception*, są wyjątkami bezpośrednimi

8) Wyjątki generowane w bloku zagnieżdżonym w bloku try, mogą być obsłużone w jego bloku catch

```
class Wyjatek_7_1              //plik Proba_7.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);}
 }
}
```

9) 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 instrukcji po wystąpieniu wyjątku lub bez wystąpienia wyjątku
              {  System.out.println("Ten blok zawsze bedzie wykonany");          }
        }
  }
public class Proba_8
{ 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
{ public static void main(String args[])
   {  try
         { Wyjatek_8.odwrotnosc(1);  }
      catch (Exception e)
        {   System.out.println("Przechwycony odlozony wyjatek "+e); }
     }}
```