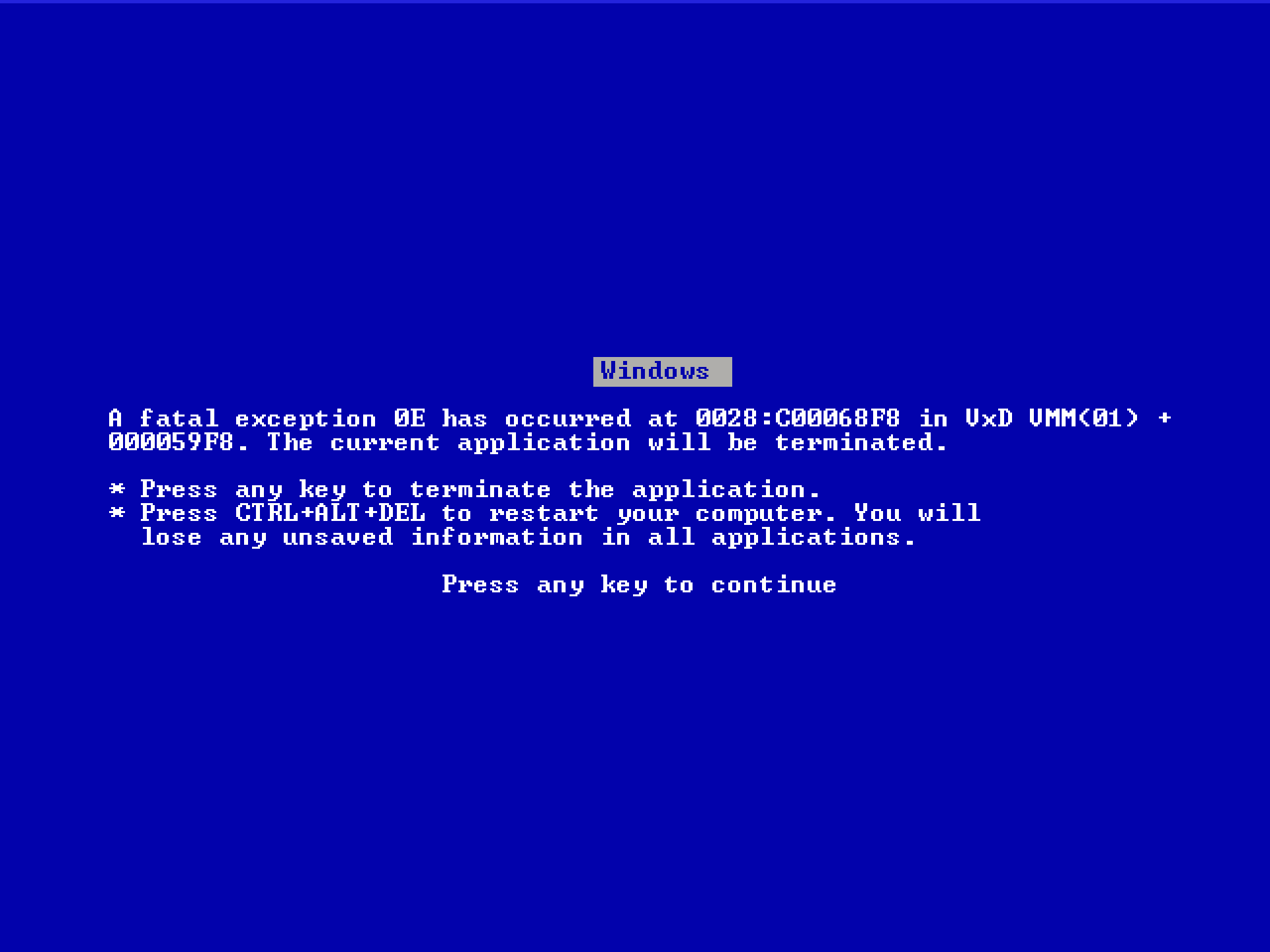
Politechnika Wrocławska

Obsługa błędów za pomocą wyjątków

Paweł Motofa (140746)


Windows
A fatal exception 0E has occurred at 0028:C00068F8 in VxD VMM(01) +
000059F8. The current application will be terminated.
* Press any key to terminate the application.
* Press CTRL+ALT+DEL to restart your computer. You will
lose any unsaved information in all applications.
Press any key to continue

Co zrobić, gdy wystąpi błąd

Przykład

```
1   class prog1
2   {
3   .       private static int silnia(int a)
4   .       {
5   .       .       int s = 1;
6   .       .       for(int i = 2; i <= a; i++)
7   .       .       .       s *= i;
8   .       .       return s;
9   .       }
10  
11  .       public static void main(String[] args)
12  .       {
13  .       .       for(int i = 4; i >= -2; i--)
14  .       .       .       System.out.println(i + "! = " + silnia(i));
15  .       }
16  .       /* Wynik:
17  .       4! = 24
18  .       3! = 6
19  .       2! = 2
20  .       1! = 1
21  .       0! = 1
22  .       -1! = 1
23  .       -2! = 1
24  .       */
25  }
```

Co zrobić, gdy wystąpi błąd

```
1  class prog2
2  {
3          private static int silnia(int a)
4          {
5                  if(a >= 0)
6                  {
7                          int s = 1;
8                          for(int i = 2; i <= a; i++)
9                                  s *= i;
10                         return s;
11                 }
12                 else
13                 {
14                         System.out.println("Błąd");
15                         return 0; // metoda musi zwrócić liczbę
16                 }
17         }
18
19         public static void main(String[] args)
20         {
21                 for(int i = 4; i >= -2; i--)
22                         System.out.println(i + "! = " + silnia(i));
23         }
24         /* Wynik:
25         4! = 24
26         3! = 6
27         2! = 2
28         1! = 1
29         0! = 1
30         Błąd
31         -1! = 0
32         Błąd
33         -2! = 0
34         */
35 }
```

Co zrobić, gdy wystąpi błąd

```
1  class prog3
2  {
3          private static int silnia(int a)
4          {
5                  if(a >= 0)
6                  {
7                          int s = 1;
8                          for(int i = 2; i <= a; i++)
9                                  s *= i;
10                         return s;
11                 }
12                 else
13                         return 0; // niech 0 oznacza błąd
14         }
15
16         public static void main(String[] args)
17         {
18                 int x;
19                 for(int i = 4; i >= -2; i--)
20                 {
21                         x = silnia(i);
22                         if(x != 0)
23                                 System.out.println(i + "! = " + silnia(i));
24                         else
25                                 System.out.println(i + "! - błąd");
26                 }
27         }
28         /* Wynik:
29         4! = 24
30         3! = 6
31         2! = 2
32         1! = 1
33         0! = 1
34         -1! - błąd
35         -2! - błąd
36         */
37 }
```

Co zrobić, gdy wystąpi błąd

Jak wyglądałaby funkcja obliczająca np. wartość symbolu Newtona?

```
private static int newton(int n, int k)
{
.       int n_silnia = silnia(n);
.       if(n_silnia == 0)
.       .       return 0; // przekazujemy błąd dalej
.       int k_silnia = silnia(k);
.       if(k_silnia == 0)
.       .       return 0; // przekazujemy błąd dalej
.       int n_minus_k_silnia = silnia(n - k);
.       if(n_minus_k_silnia == 0)
.       .       return 0; // przekazujemy błąd dalej
.       return n_silnia / (k_silnia * n_minus_k_silnia);
}
```

Sytuacje wyjątkowe w programie

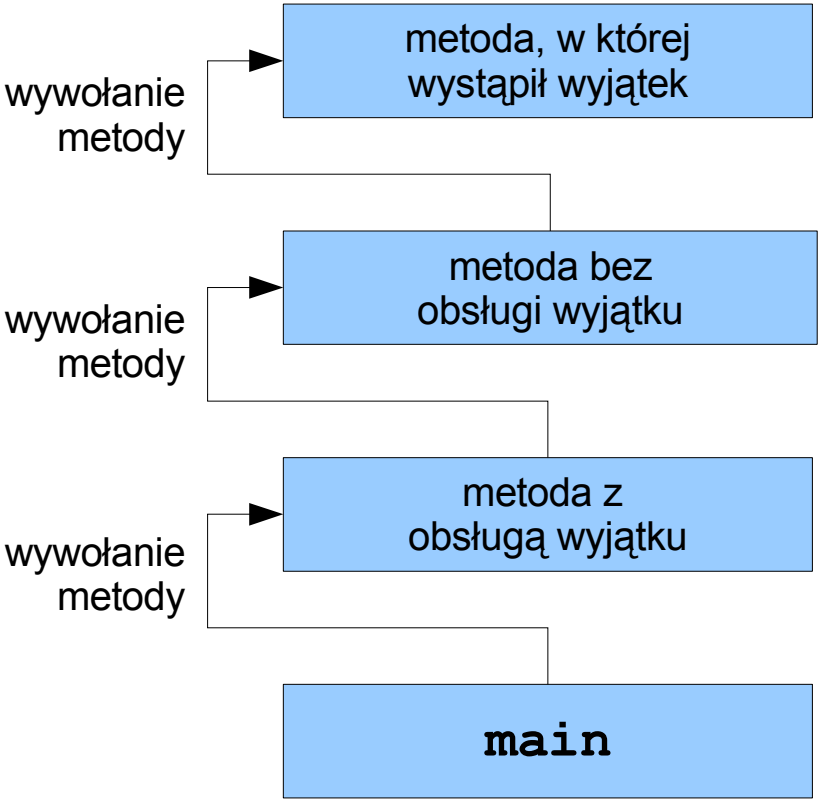
- *„Jeżeli coś może się nie udać — nie uda się na pewno.”*
- *„W każdym programie błędy wykazują skłonność do występowania w tym miejscu, które sprawdzasz jako ostatnie.”*
- Programując, należy brać pod uwagę, każdą możliwość wystąpienia błędu.

Wyjątek (exception) to zdarzenie, występujące podczas działania programu, które zmienia jego normalny przebieg. W Javie wyjątek reprezentowany jest jako obiekt.

Procedura obsługi wyjątku (exception handler) to blok programu przeznaczony do wykonania w razie wystąpienia wyjątku.

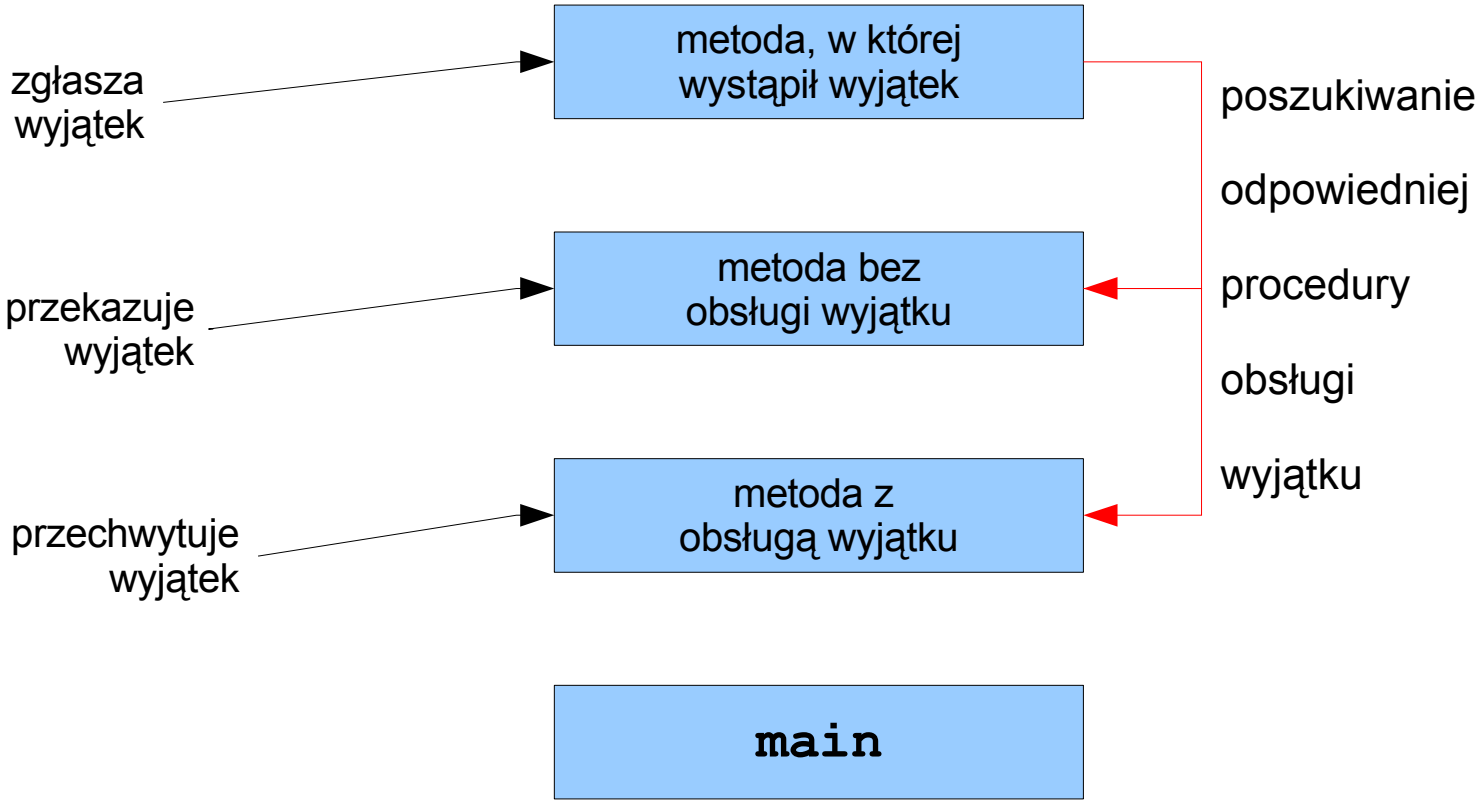
Co się dzieje, gdy wystąpi wyjątek

Stos wywołań (call stack)



Co się dzieje, gdy wystąpi wyjątek

Poszukiwanie procedury obsługi wyjątku



Zgłaszanie wyjątków

- Aby zgłosić wystąpienie wyjątku należy najpierw stworzyć obiekt go reprezentujący. Np.:

```
new MyException()
```

- Klasa takiego obiektu musi dziedziczyć po klasie *Throwable*.
- Utworzony obiekt zgłaszamy przy użyciu instrukcji *throw*. Np.:

```
throw new MyException();
```

- Po zgłoszeniu wyjątku, wykonywanie metody zostaje przerwane, a obiekt reprezentujący wyjątek przekazany „dalej” do procedury jego obsługi.

Specyfykacja wyjątków

- W Javie wymagane jest deklarowanie, jakie wyjątki może zgłosić dana metoda.

```
void f() throws TooBig, TooSmall, DivZero
{
        //...
}
```

- Jeśli dana metoda może „wyrzucić” wyjątek, którego specyfikacji zabraknie po insrukcji *throws*, zostanie zgłoszony błąd w czasie kompilacji. Nie dotyczy to wyjątków typu *RuntimeException*.

```
void f() throws TooSmall, DivZero
{       //...
        throw new TooBig();
}
```

Błąd kompilacji
„unreported exception TooBig;
must be caught or declared to be thrown”

Ograniczenia wyjątków

W metodzie przeciążonej można zgłaszać jedynie te wyjątki, które zostały podane w specyfikacji jej wersji z klasy bazowej.

Przechwytywanie wyjątków

- Blok *try* oraz *catch*

```
try {

     // kod który może zgłosić wyjątek

} catch(Typ1 wyjatek1) {

     // kod wykonywany, w przypadku
     // wystąpienia wyjątku typu Typ1 w bloku try

} catch(Typ2 wyjatek2) {

     // kod wykonywany, w przypadku
     // wystąpienia wyjątku typu Typ2 w bloku try
}
```

Przechwytywanie wyjątków

- Każdy człon *catch* działa jak metoda, która jako parametr pobiera obiekt reprezentujący wyjątek.
- Jeśli podczas wykonywania kodu w bloku *try* wystąpi wyjątek, wykonywana jest pierwsza procedura obsługi, której typ parametru odpowiada zgłoszonemu wyjątkowi (również przez rzutowanie w górę).
- Jedna procedura może obsłużyć wiele różnych wyjątków.

Przykład

```
1  class prog5
2  {
3          private static int silnia(int a) throws Exception
4          {
5                  if(a >= 0)
6                  {
7                          int s = 1;
8                          for(int i = 2; i <= a; i++)
9                                  s *= i;
10                         return s;
11                 }
12                 else
13                         throw new Exception("Silnia z liczby ujemnej (" + a + ")");
14         }
15
16         private static int newton(int n, int k) throws Exception
17         {
18                 return silnia(n) / (silnia(k) * silnia(n-k));
19         }
20
```

Nieprzechwycony wyjątek
jest automatycznie przekazywany dalej.

```
21         public static void main(String[] args)
22         {
23                 try {
24                         for(int i = 2; i >= -2; i--)
25                                 System.out.println(i + "! = " + silnia(i));
26                 }
27                 catch(Exception e)
28                 {
29                         System.out.println(e.getMessage());
30                 }
31
32                 try {
33                         System.out.println("5 nad 3 = " + newton(5, 3));
34                         System.out.println("5 nad 7 = " + newton(5, 7));
35                 }
36                 catch(Exception e)
37                 {
38                         System.out.println(e.getMessage());
39                 }
40         }
41         /* Wynik:
42         2! = 2
43         1! = 1
44         0! = 1
45         Silnia z liczby ujemnej (-1)
46         5 nad 3 = 10
47         Silnia z liczby ujemnej (-2)
48         */
49 }
50
```

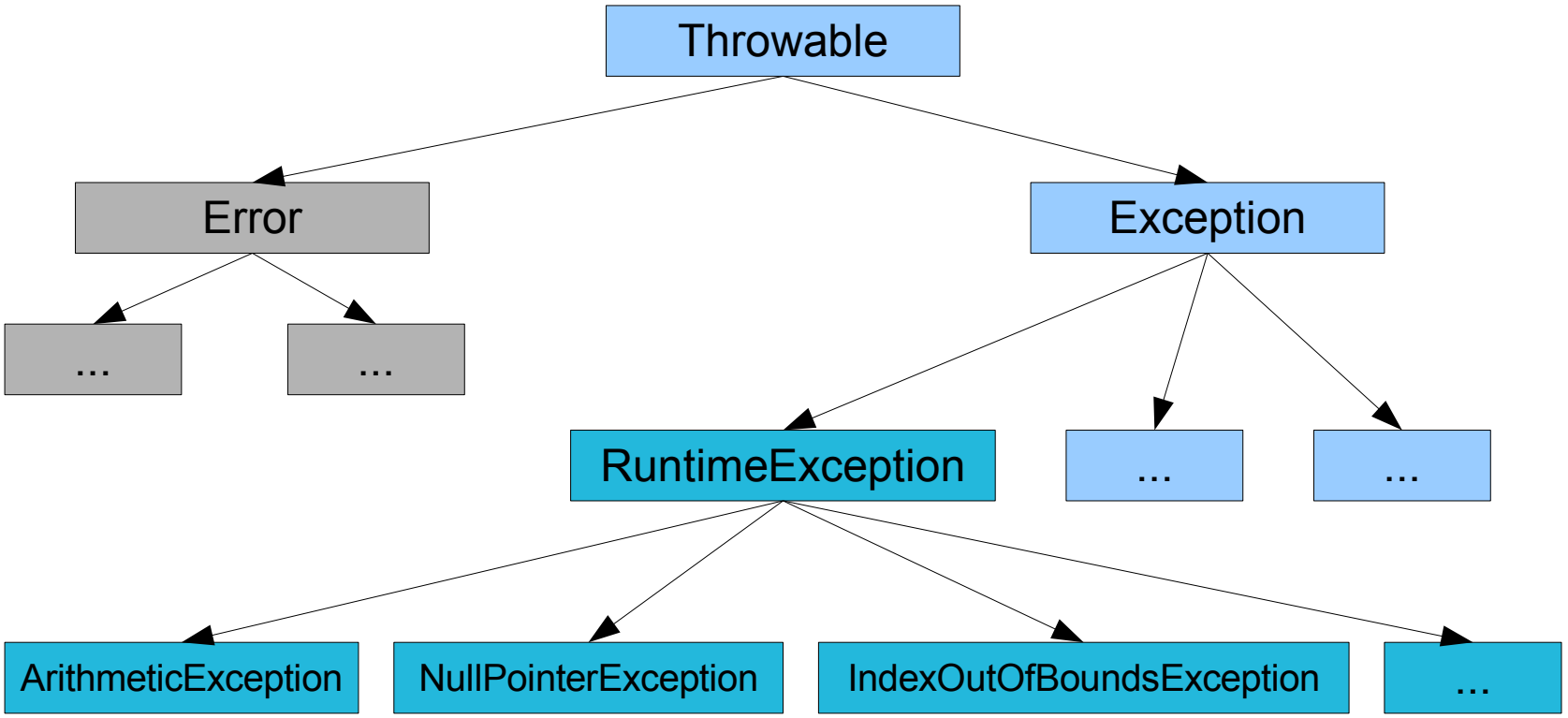
Sekcja *finally*

Kod w sekcji *finally* wykonywany jest niezależnie od tego, czy w bloku *try* wystąpił wyjątek.

```
try {
    // działania, które mogą zgłosić wyjątek
} catch(...) {
    // obsługa sytuacji wyjątkowej
} finally {
    // czynności wykonywane za każdym razem
    // np. porządkowanie
}
```

Standardowe wyjątki Javy

Hierarchia klas



Klasa Throwable

- Konstruktory:

```
Throwable()
Throwable(String message)
Throwable(String message, Throwable cause)
Throwable(Throwable cause)
```

- Ważniejsze metody:

```
Throwable getCause()
String getMessage()
String getLocalizedMessage()
StackTraceElement[] getStackTrace()
void printStackTrace()
```

Tworzenie własnych wyjątków

- Własne wyjątki tworzymy tworząc klasę dziedziczącą po istniejącym typie wyjątku.
- Często, aby obsłużyć sytuację wyjątkową, wystarczy znać jedynie typ wyjątku.
  Np.:

```
class MojWyjatek extends Exception
{
      // Tu nic nie ma
}
```

Dziękuję za uwagę