Programowanie w języku Java - Wyjątki, obsługa wyjątków, generowanie wyjątków

mgr inż. Maciej Lasota <m.lasota@tu.kielce.pl>

Version 1.0, 13-05-2017

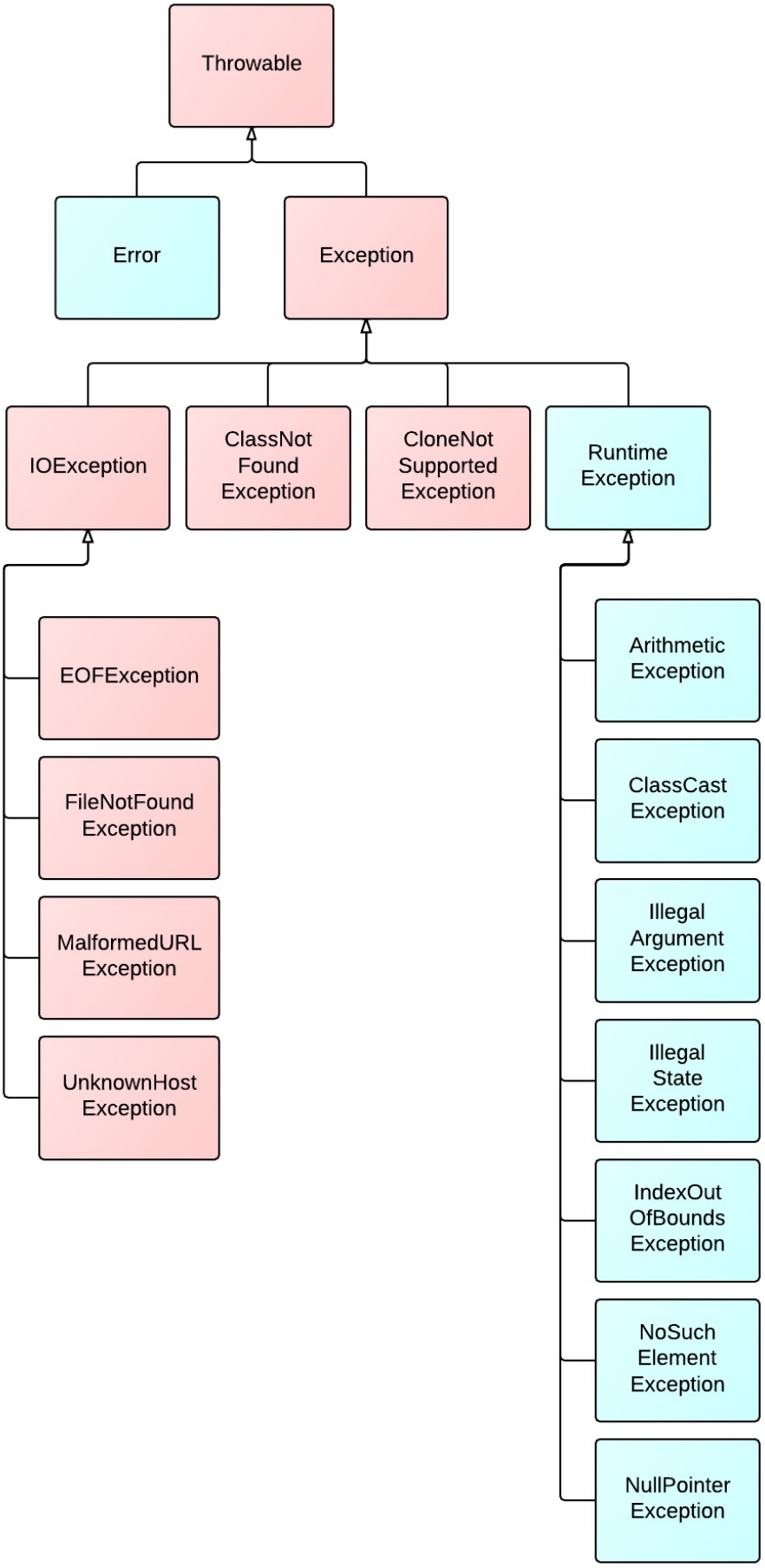
Spis treści

Zajęcia **laboratoryjne nr 8** z programowania w języku Java - jedne zajęcia **1,5h**.

Wyjątki

Przez pojęcie **wyjątków** w językach obiektowych rozumiemy mechanizm kontroli przepływu służący do obsługi **zdarzeń wyjątkowych** *(w szczególności błędów)*. Wyjątek w języku Java to obiekt, który opisuje pewną sytuację błędną lub nieprawidłową – wyjątkową. Jest to obiekt odpowiedniego typu, tj. obiekt klasy `Throwable` lub jej dowolnej podklasy *(np. Exception, Error itd.)*.

Ważną częścią każdej aplikacji jest **sygnalizowanie** oraz obsługa pojawiających się błędów. Wystąpienie błędu w aplikacji zaimplementowanej w języku Java sygnalizujemy poprzez **rzucenie (wyrzucenie) wyjątku**. Obsługa błędów to tzw. **łapanie (przechwytywanie) wyjątków**.



Rysunek 1: Hierarchia klas wyjątków.

Wyjątki dzielą się na dwa rodzaje:

- **kontrolowane** - to takie które musimy deklarować i obsługiwać,
- **niekontrolowane** - to takie które możemy obsługiwać, ale nie musimy.

Wyjątki **niekontrolowane** to instancje klas `Error` i `RuntimeException` oraz ich dowolnych podklas. Wyjątki **kontrolowane** to instancje klas `Throwable` i `Exception` oraz ich podklas, z wyłączeniem podklas klas `Error` i `RuntimeException`. Wyjątki mogą być rzucane (zgłaszane) przez **Wirtualną Maszynę Javy (JVM)** oraz przez programistę. Najpopularniejszym wyjątkiem rzucanym przez JVM jest `NullPointerException`. Wyjątek ten rzucany jest przy próbie odwołania się do obiektu (np. próba uruchomienia metody) z użyciem referencji, która ma wartość `null`. `NullPointerException` jest podklasą klasy `RuntimeException` tak więc jest wyjątkiem niekontrolowanym – nie musimy go w żaden sposób deklarować czy obsługiwać.

Obsługa wyjątków

Wyjątki są **wyrzucane** i **propagowane** tak długo jak długo pozostają nie przechwycone w kodzie programu. Wyjątki nie przechwycone propagują się aż do metody `main(⋯)` i jeśli także tam nie są przechwycone powodują przerwanie wykonania programu.

W celu przechwycenia i obsłużenia wyjątków w języku Java korzysta się z bloku **try-catch-finally**.

Kod którego wyjątki chcemy przechwytywać umieszczamy w klauzuli `try {⋯}`. Wewnątrz klauzuli `catch {⋯}` umieszczamy **kod obsługi błędu** - kod ten będzie wykonany tylko i wyłącznie wówczas gdy kod programu wyrzuci wyjątek *typu* zgodnego z typem określonym w deklaracji klauzuli catch. Kod umieszczony wewnątrz klauzuli `finally {⋯}` będzie wykonany **zawsze**, niezależnie od tego czy kod programu wyrzucił wyjątek czy nie.

Przykład 1. Przykład definicji bloku try-catch-finally

```
try {
  //kod programu generujacy wyjatek
} catch (Exception e) {
  //obsluga wyjatkow klasy Exception
} catch (Throwable t) {
  //obsluga wyjatkow klasy Throwable
} finally {
  //kod zawsze wykonywany
}
```

Gdy w wyniku wykonania instrukcji w bloku try powstanie wyjątek typu `Exception` lub `Throwable` to sterowanie zostanie przekazane do kodu umieszczonego w w/w klauzulach catch.

Klauzula **finally** jest opcjonalna, za to klauzul **catch** może być dowolnie wiele.

Przykład 2. Przykład użycia bloku try-catch-finally

```
class TestExceptions {

   public static void main(String  args[]) {
      int d, a;
      try {
        d = 0;
        a = 42 / d;
        System.out.println("This will not be printed.");
      } catch (ArithmeticException e) {
        System.out.println("Division by zero.");
      } finally {
        System. out. println( "Finally") ;
      }
      System.out.println("Out of try/catch/finally - statement");
   }
}
```

Generowanie (propagowanie) wyjątków

Wyjątki mogą być generowane **jawnie** w kodzie programu tj. przez programistę - z użyciem słowa kluczowego `throw`. Tak wygenerowany wyjątek może zostać obsłużony bezpośrednio przez klauzulę `catch` lub może zostać propagowany bezpośrednio wyżej do metody wywołującej nasz kod (naszą metodę) w której został wyrzucony wyjątek.

Przykład 3. Przykład użycia instrukcji throw do generowania wyjątku

```
if (o == null)
  throw new NullPointerException();
```

W istocie, instrukcja `throw` jest niczym innym jak sposobem specyficznego przekazywania sterowania do jakichś punktów programu (do miejsc obsługi wyjątku). Należy jednak korzystać z niej wyłącznie w celu **sygnalizowania błędów**.

Przykład 4. Przykład ponownego wyrzucenia wyjątku

```
try {
  ...
} catch (Exception e) {
  ...
  throw e; //ponowne wyrzucenie wyjątku
}
```

Przykład 5. Przykład propagowania wyjątku

```
// propagowanie wyjątku którego obsługą zajmie się kod wywołujący metodę
void method() throws IOException {

}
```

Przykład 6. Przykład propagowania wyjątku

```
class TestExceptions {

   static void myMethod(int testnum) throws Exception {
      System.out.println ("start - myMethod");
      if (testnum == 12)
         throw new Exception();
      System.out.println("end - myMethod");
      return;
   }

   public static void main(String  args[]) {
      int testnum = 12;
      try {
         System.out.println("try - first statement");
         myMethod(testnum);
         System.out.println("try - last statement");
      }
      catch ( Exception ex) {
         System.out.println("An Exception");
      }
      finally {
         System. out. println( "finally") ;
      }
      System.out.println("Out of try/catch/finally - statement");
   }
}
```

Tworzenie własnych klas wyjątków

Oprócz wykorzystywania dostępnych klas wyjątków do generowania/obsługi sytuacji wyjątkowych programista ma możliwość tworzenia **własnych klas wyjątków**. Żeby stworzyć własny wyjątek należy zdefiniować odpowiednią klasę która dziedziczy funkcjonalność z klasy `Exception`.

Przykład 7. Definiowanie własnej klasy wyjątku

```
class SuperWyjatek extends Exception {
  ...
}
```

Zwykle w naszej klasie wystarczy umieścić dwa konstruktory: **bezparametrowy** oraz z **jednym argumentem** typu `String` (komunikat o przyczynie powstania wyjątku). W konstruktorach tych należy wywołać **konstruktor** nadklasy (za pomocą odwołania `super(…)`, w drugim przypadku z argumentem `String`).

Przykład 8. Przykład użycia własnej klasy wyjątku

```
class MyException extends Exception{
  public MyException() {
    System.out.println("Error");
  }
}

class TestExceptions {

   static void myMethod(int testnum) throws MyException {
      System.out.println ("start - myMethod");
      if (testnum == 12)
         throw new MyException();
      System.out.println("end - myMethod");
      return;
   }
   public static void main(String  args[]) {
      int testnum = 12;
      try {
         System.out.println("try - first statement");
         myMethod(testnum);
         System.out.println("try - last statement");
      }
      catch ( MyException ex) {
         System.out.println("An Exception");
      }
      finally {
         System. out. println( "finally") ;
      }
      System.out.println("Out of try/catch/finally - statement");
   }
}
```

Bibliografia

- **Bruce Eckels**, *"Thinking in Java. Edycja polska. Wydanie IV"*, wydawnictwo Helion.
- **Cay S. Horstmann, Gary Cornell**, *"Java. Podstawy. Wydanie IX"*, wydawnictwo Helion.
- **Cay S. Horstmann, Gary Cornell**, *"Java. Techniki zaawansowane. Wydanie IX"*, wydawnictwo Helion.
- **Krzysztof Barteczko**, *"Podstawy programowania w języku Java, PJWSTK"*, http://edu.pjwstk.edu.pl/wyklady/ppj/scb/

- **Konrad Kurczyna**, *"Laboratorium Java"*, Politechnika Świętokrzyska w Kielcach.
- **Mariusz Lipiński**, *"Nauka Javy"*, http://www.naukajavy.pl/