Metody programowania

Wprowadzenie do platformy Java

Dr inż. Andrzej Grosser

Częstochowa, 2013

Spis treści

1. Wyjątki

Rozdział opisuje wyjątki - ogólną ich filozofię, sposoby wyrzucania i wyłapywania. Pokazano także, w jaki sposób można utworzyć własną klasę wyjątku. Na koniec wspomniano także o nowej konstrukcji try z zasobami wprowadzonej w Javie 7. Do napisanie tego rozdziału użyto książek "Core Java"[2] i "Thinking in Java"[1]

1.1. Wprowadzenie

Obsługa błędów w starszych językach programowania, takich jak Pascal, polegała na zwracaniu specjalnej wartości lub ustawianiu odpowiedniej flagi, gdy wykonanie podprogramu zakończyło się błędem. Ten sposób obsługi błędów jest jednak dość skomplikowany w użyciu, gdyż wymaga kontroli flag i wartości - w tej sytuacji, gdy próbowano obsłużyć za każdym razem wystąpienie błędu kod szybko stawał się nieczytelny. Z drugiej strony, całkowite pominięcie obsługi błędu, też nie było dobrym pomysłem. Z pomocą w tym problemie przyszedł mechanizm rzucania i obsługi wyjątków.

W kodzie zostaje zgłoszony wyjątek (inaczej wyrzucany jest wyjątek), kontrolę przejmuje mechanizm obsługi wyjątków - bieżąca ścieżka sterowania jest przerywana, następuje wyszukanie możliwości kontynuacji programu z obsługą błędu. Przy czym sposób obsługi błędu nie musi być koniecznie w funkcji, w której został wywołany wyjątek, wyjątek może być obsłużony w wcześniej wywołanych funkcjach, korzystających z tej, w której został zgłoszony wyjątek. Nie ma tutaj potrzeby jawnego przekazywania wyjątku pomiędzy kolejnymi wywołaniami funkcji, wszystko dzieje się automatycznie.

Wyjątki narzuciły konwencje obsługi błędów. Spowodowały, że nie ma możliwości pominięcia obsługi błędów - w danym miejscu wyrzucony wyjątek można zignorować, ale musi być, w którymś miejscu obsłużony.

Ponieważ nie trzeba od razu wyjątku obsługiwać, można poczekać, aż uzyska się wystarczającą informację do podjęcia odpowiedniej procedury obsługi go, kod staje się dużo

bardziej przejrzysty. Kod staje się również czytelniejszy, ze względu na jasny podział normalnego sterowania od procedury obsługi błędów (opisane dalej bloki try i catch).

Wyjątki mimo swojej użyteczności i wygody stosowania mają też wady, przede wszystkim spowalniają wykonanie programu. Dlatego też powinny być używane w sytuacjach, w których nie ma określonej ścieżki postępowania, gdy dalsze zwyczajne postępowanie jest niemożliwe lub niepożądane.

1.2. Klasyfikacja wyjątków

Wszystkie wyjątki Javy są wyprowadzone z interfejsu `Throwable`. Hierarchia wyjątków dzieli się później na błędy `Error` i wyjątki właściwe `Exception`.

Błędy (klasy wyprowadzone z Error) opisują sytuacje wyjątkowe związane z błędami wewnętrznymi i brakiem zasobów wewnątrz systemu uruchomieniowego Javy. Programiści nie powinny rzucać tego typu wyjątków, są one zarezerwowane do mechanizmów wewnętrznych Javy.

Wyjątki z gałęzi `Exception` dzielą się na wyjątki uruchomieniowe (wyprowadzone z gałęzi klasy RuntimeException) i inne wyjątki związane na przykład z obsługą wejścia wyjścia, niepoprawnymi operacjami na danych itp. Pierwsze z nich są związane z błędami programistycznymi (próba wykonania operacji na odniesieniu pustym - null, błędne rzutowanie itd.), drugie zaś z problemami zewnętrznymi, które wpływają na aplikację.

Wyjątki dziedziczące po Error i RunntimeException są nazywane wyjątkami niekontrolowanymi, wszystkie inne są nazywane kontrolowalnymi. Nazwa wynika ze sposobu działania kompilatora - kompilator sprawdza czy wszystkie wyjątki kontrolowane mają obsługę (jeżeli nie mają wymagana jest opisana dalej etykieta przy nagłówku metody).

1.2.1. Powiadamianie o wyjątkach rzucanych przez metody

Kompilator Javy sprawdza w czasie kompilacji czy wszystkie metody, które mogą wyrzucić wyjątek kontrolowany są oznaczone w odpowiedni sposób. W tym celu dodaje się do deklaracji metody informację o typach wyjątków, jakie może wyrzucić metoda:

```
1 class Przyklad {
2    public void m() throws IOException {/*---*/}
3 }
```

W sytuacji, gdy metoda może zwrócić więcej niż jeden wyjątek kontrolowany, wszystkie wyjątki muszą być wyspecyfikowanie w jej nagłówku:

```
1 class ParserError extends Exception {
2 //...
3 }
4
5 class Parser {
6     public void parsuj() throws IOException, MojWyjatek {
7     //...
8     }
9 }
```

1.2.2. Rzucanie wyjątków

Do rzucania wyjątków w Javie służy, podobnie jak w C++ - `throw`. Żeby wyrzucić wyjątek należy stworzyć jego obiekt a następnie użyć go:

```
1 EOFException exc = new EOFException();
2 throw exc;
3 //lub po prostu
4 throw new EOFException();
```

Należy jedynie pamiętać, że obiektem wyjątku może być obiekt klasy pochodzącej od `Throwable` (w przeciwieństwie do C++, gdzie da się zwracać wyjątki dowolnego typu).

1.2.3. Tworzenie klas wyjątków

W sytuacji, gdy żadna z istniejących klas wyjątków nie pasuje do specyfiki rozpatrywanego problemu, należy utworzyć własną klasę, musi ona dziedziczyć od klasy `Exception` lub jednej z podklas tej klasy.

Zazwyczaj obiekty klas wyjątków posiadają oprócz konstruktora domyślnego posiadają konstruktor przyjmujący obiekt łańcucha znaków umożliwiający tworzenie szczegółowego komunikatu o błędzie (przekazuje się go do klasy nadrzędnej, a używa w metodzie `toString()` klasy Throwable, pozwala to na wyświetlanie tego komunikatu, co ułatwia znajdowanie błędów):

```
1  class MojWyjatek extends Exception {
2
3  public MojWyjatek() {}
4
```

```
5  public MojWyjatek(String problem) {
6      super(problem);
7  }
8 }
```

1.2.4. Wyłapywanie wyjątków

Jeżeli wyjątek nie zostanie wyłapany, program konsolowy zakończy się i wypisze stosowny komunikat w konsoli - typ wyjątku, ślad stosu. Programy z graficznym interfejsem wypiszą jedynie komunikat w konsoli.

Uniknięcie opisywanego wcześniej, domyślnego sposobu obsługi wyjątku wymaga zapisu kodu odpowiedzialnego za wychwycenia go. Do wyłapywania wyjątków służą bloki `try-catch`. Na przykład:

```
1 try {
2 /* Kod, w którym może dojść do wyrzucenia wyjątku */
3 }
4 catch(IOException w) {
5 }
```

Gdy w trakcie wykonywania bloku try nie zostanie wyrzucony żaden wyjątek, to pomijane jest wykonywanie kodu znajdującego się w blokach catch.

Jeżeli w trakcie wykonywania instrukcji wewnątrz bloku try zostanie wyrzucony wyjątek, zostanie pominięte wykonywanie pozostałych instrukcji w tym bloku i sprawdza się czy typ wyrzuconego wyjątku pasuje do klauzuli catch - jeżeli pasuje to wykonywany jest jej kod. W sytuacji, gdy wyjątek nie zostanie wychwycony w bieżącej metodzie jest przekazywany do obsłużenia w metodzie, która ją wywołała.

Można również bloku try-catch wyłapywać wiele różnych wyjątków, w tym celu zapisuje się bloki catch odpowiedzialne za wychwycenie stosownego wyjątku jeden po drugim:

```
1 try {
2 //...
3 }
4 catch(IOException w) {
5 //...
6 }
7 catch(MojWyjatek w) {
8 //...
9 }
```

1.2.5. Powtórne zwrócenie wyjątku

W niektórych sytuacjach konieczne jest wyrzucenie tego samego wyjątku, po wychwyceniu go w bloku catch (w bieżącym bloku catch nie była możliwa pełna obsługa wyjątku).

```
try {
/* Kod, w którym jest
wyrzucany wyjątek */
}
catch(MojWyjatek w) {
    throw w;
}
```

1.2.6. Klauzula finally

Klauzula `finally` umożliwia wykonanie pewnego kodu, niezależnie od tego czy wyjątek został wychwycony, czy nie. Przydaje się to, gdy jakiś kod musi być zawsze wykonany - na przykład do zamykania otwartych plików:

```
InputStream in = null;
try {
    in = new FileInputStream(nazwaPliku);
    //...
}
catch(IOException e) {
//...
}
finally {
//Co ciekawe zamykanie pliku też może wyrzucić wyjątek
// IOException.
    in.close();
}
```

Klauzula finally nie musi być poprzedzana uchwytami wyjątków (catch). Można zapisać blok w następujący sposób (kod w klauzuli finally wykona się zawsze):

```
try {
  //Kod try
}
finally {
    //Kod finally
}
```

Klauzula finally może dawać nieoczekiwane wyniki, w sytuacji, gdy wykonywanie bloku try zakończy się instrukcją powrotu (return) - w tej sytuacji zostanie jeszcze, przed zwróceniem wartości z funkcji, zostanie wykonany kod z klauzuli finally - jeżeli i ona zakończy się instrukcją powrotu, to wartość zwracana z finally nadpisze wartość oryginalnie zwracaną z bloku try. Na przykład:

```
public static int method2() {
    try {
        //...
        return 1;
    } catch (Exception e) {
        //...
    }
    finally {
        //...
        return 2;
    }
}
```

Metoda method2() z kodu powyżej będzie zwracać wartość 2, nawet jeżeli w bloku try zostanie wykonana instrukcja return 1 (return w finally jest ignorowane jedynie w sytuacji, gdy metoda nie obsłuży wyrzuconego wyjątku).

1.2.7. Try z zasobami

Java 1.7 wprowadza skrót dla kodu otwierającego, używającego i zamykającego jakiś zasób. Na przykład zamiast pisać:

```
Scanner scanner = new Scanner(new FileInputStream(path);
try {
   while (scanner.hasNext())
       System.out.println(scanner.next());
}
finally {
   scanner.close();
}
```

można użyć następującego skrótu:

```
try (Scanner scanner = new Scanner(new FileInputStream(path)){
   while (scanner.hasNext())
       System.out.println(scanner.next());
}
```

Zasób (zasoby) otwierany w tego rodzaju instrukcji musi implementować interfejs `AutoCloseable`, który wymusza implementację metody `close()`[1].

[1]Dokładna sygnatura to void close() throws Exception

Literatura

[1] B. Eckel. *Thinking in Java*. Helion, 2006.

[2] C. S. Horstmann and G. Cornell. *Core Java Volume I–Fundamentals*. Prentice Hall, 2011.