Programowanie obiektowe

Wykład 8: Obsługa wyjątków

dr inż. Marcin Luckner
mluckner@mini.pw.edu.pl

Wydział Matematyki i Nauk Informacyjnych

Wersja 1.4
4 marca 2021

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Politechnika Warszawska
Unia Europejska Europejski Fundusz Społeczny

Projekt „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca” współfinansowany jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Zadanie 10 pn. „Modyfikacja programów studiów na kierunkach prowadzonych przez Wydział Matematyki i Nauk Informacyjnych”, realizowane w ramach projektu „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca”, współfinansowanego jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Politechnika Warszawska
Unia Europejska Europejski Fundusz Społeczny

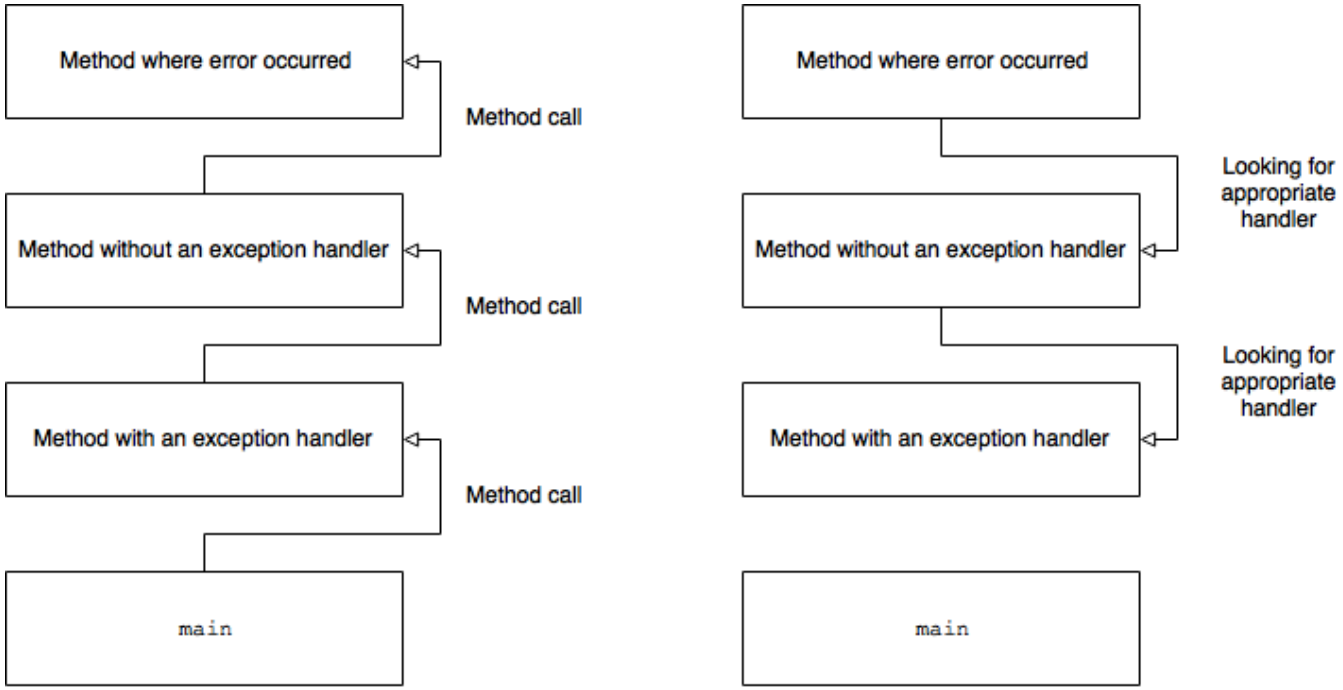
Przerwanie działania aplikacji

- Nawet najlepiej zaprojektowany algorytm realizuje tylko rutynowe działania.
- W przypadku nieprzewidzianych zdarzeń algorytm może nie mieć możliwości ich poprawnej realizacji.
- Przyczynami problemów mogą być:
  - Niepoprawne dane wejściowe
  - Przerwane połączenie
  - Błędna inicjalizacja metody
- W przypadku wystąpienia problemu aplikacja powinna zachować się w sposób przyjazny dla użytkownika.
  - Powiadomić wyczerpująco o błędzie.
  - Zapisać dotychczasową pracę.
  - Zakończyć poprawnie działanie elementów programu, których przerwanie mogłoby prowadzić do dalszych problemów.
- Java pozwala radzić sobie z takimi sytuacjami dzięki *obsłudze wyjątków*.

Informowanie o błędnym działaniu metody

- Jeżeli działanie funkcji będzie błędne i błąd da się wykryć to mamy kilka możliwości postępowania.
  - Funkcja może zwrócić określoną wartość ze swojej przeciwdziedziny.
    - Metoda wywołująca może nie rozpoznać komunikatu błędu
    - Możemy nie mieć dostępnej wartości do przypisania błędu
  - Funkcja może zwrócić wartość `null`.
    - Metoda wywołująca może próbować potraktować wynik jako pełnoprawny obiekt.
    - Metoda wywołująca musi kontrolować poprawność zwracanego argumentu.
  - Funkcja może wygenerować wyjątek.
    - Musi istnieć mechanizm obsługi wyjątków.

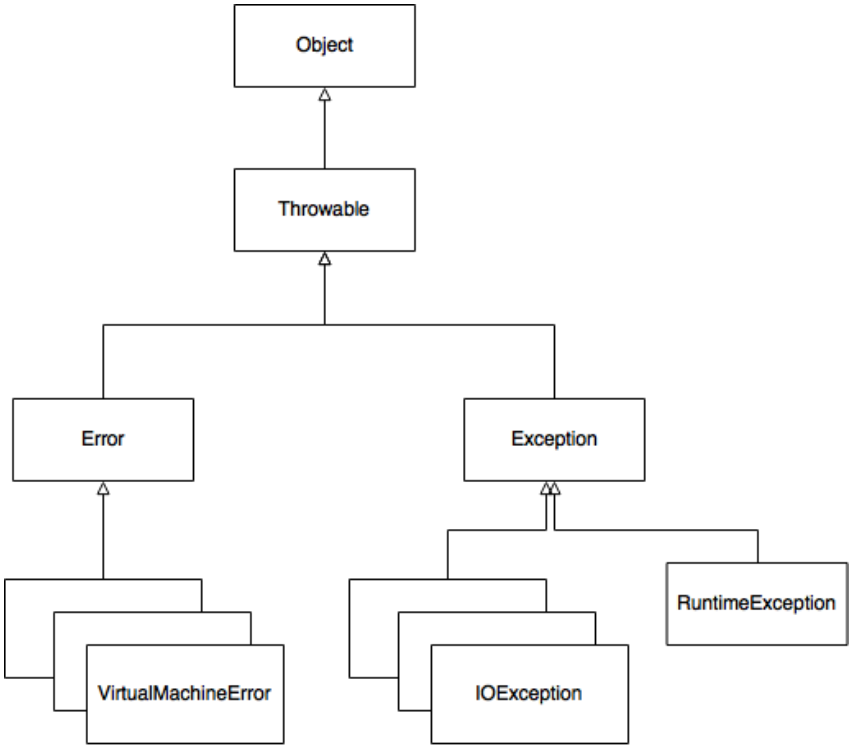
Obsługa wyjątków



Rysunek 1: Stos wywołań metod i wsteczne przekazywanie wyjątku.

- Po wystąpieniu wyjątku przerywanie jest działanie metod, które nie zapewniają obsługi wyjątku.
- Jeżeli przerwanie dojdzie do metody `main()` aplikacja zostanie zatrzymana.

Typy wyjątków



Rysunek 2: Hierarchia wyjątków

- Wszystkie wyjątki należą do klasy `Throwable`.
- Wyjątki z klasy `Error` odpowiadają poważnym błędom wewnętrznym i nie można nic na nie poradzić.
- Wyjątki z klasy `RuntimeException` są wynikiem błędnego programowania i da się je wyeliminować podczas pisania kodu.
- Pozostałe wyjątki powinny być obsłużone.

Wyjątki niekontrolowane

- Wyjątki `Error` i `RuntimeException` są wyjątkami niekontrolowanymi.
- Oznacza to, że nie możemy przewidzieć ich wystąpienia lub nie powinniśmy zakładać, że mogą wystąpić.
- Nie możemy też ostrzec użytkowników, w sposób formalny, że mogą wystąpić.

Złota zasada

Jeśli wystąpił wyjątek RuntimeException, znaczy, że popełniłeś błąd jako programista!

Deklarowanie wyjątków kontrolowanych

- Możliwość wystąpienia wyjątków kontrolowanych jest sygnalizowana w deklaracji metody.
- Deklaracja następuje poprzez słowo kluczowe `throws` i wymienienie wyjątków, które mogą wystąpić

```
1  public Image loadImage(String s) throws
       FileNotFoundException, EOFException
```

- Jeżeli funkcja deklaruje wyjątek kontrolowany to funkcja wykonująca musi przekazać go dalej lub obsłużyć.

Przekazywanie wyjątków

- Jeżeli sami nie potrafimy obsłużyć wyjątku to powinniśmy przekazać jego obsługę funkcji wywołującej poprzez dodanie klauzuli `throws`.
- Tworzy to jednak pewne problemy.
  - Załóżmy, że modyfikujemy istniejącą metodę i wymaga to dodania klauzuli `throws`.
  - Teraz wszystkie funkcje wywołujące naszą metodę muszą albo przechwycić wyjątek, albo zadeklarować kontrolowany wyjątek.
  - Może to skutkować masową refaktoryzacją kodu.

Przechwytywanie wyjątków

- Przechwytywanie wyjątku opiera się na zastosowaniu trzech bloków.

Przechwycenie wyjątku

```
try {
    readData(fileName);
}
catch (EOFException e) {
    e.printStackTrace();
}
finally {
    closeFile(fileName);
}
```

try deklaracja bloku kodu obserwowanego pod kątem generowania wyjątków.

catch definicja sposobu postępowania z wyjątkami.

finally definicja operacji. które muszą być wykonane niezależnie od wystąpienia wyjątku.

Blok `try`

- Blok `try` określa fragment kodu, który może wygenerować wyjątek.
- Każda metoda z klauzulą `throws` musi być umieszczona w bloku `try` (lub wyjątek zostanie przekazany)

Blok catch

- Blok catch opisuje zachowanie po wykryciu konkretnego wyjątku.
- Możemy obsłużyć wiele wyjątków z jednego bloku try.

Zestaw bloków catch

```
1  try {
2      readData(fileName);
3  }
4  catch (EOFException e) {
5      e.printStackTrace();
6  }
7  catch (FileNotFoundException e |
       UnknownHostException e) {
8      e.printStackTrace();
9  }
```

- Bloki mogą być definiowane oddzielnie dla różnych wyjątków lub grupować wyjątki.

Blok finally

- Blok finally zawiera fragment kodu, który zostanie wykonany niezależnie od tego czy wystąpił wyjątek.
- Zazwyczaj służy do zamknięcia otwartych połączeń z źródłami danych.
- Może być źródłem nowych wyjątków.

Blok finally generujący wyjątek

```
1  try {
2      openFile(fileName);
3      readData(fileName);
4  }
5  catch (FileNotFoundException e) {
6      e.printStackTrace();
7  }
8  finally {
9      closeFile(fileName);
10 }
```

- Próba zamknięcia pliku przez metodę closeFile spowoduje wyjątek.

Zagnieżdżenie bloków

- Problem kodu generującego wyjątki w bloku `finally` można rozwiązać zagnieżdżając bloki

Blok `finally` generujący wyjątek

```
1   public static void main(String[] args) {
2       try {
3           if(args.length > 0) { //Nie zawsze potrzebujemy
                wyjatkow
4               String fileName = args[0];
5               openFile(fileName);
6               try {
7                   readData(fileName);
8               }
9               catch (EOFException e) {
10                  e.printStackTrace();
11              }
12              finally {
13                  closeFile(fileName);
14              }
15          }
16      }
17      catch (FileNotFoundException e) {
18              e.printStackTrace();
19      }
20  }
```

- Zagnieżdżanie dodatkowo porządkuje funkcjonalność kodu

Blok try z zasobami

- Innym rozwiązaniem jest stosowanie bloku `try` z zasobami.
- Rozwiązanie można stosować do obiektów implementujących interfejs `AutoCloseable` i nadpisujących jego metodę `close`.

Przepisanie pliku

```
1  try (Scanner in = new Scanner(new
       FileInputStream("/usr/share/dict/words") , "UTF-8"),
2  PrintWriter out = new PrintWriter("out.txt"))
3  {
4      while (in.hasNext())
5          out.println(in.next().toUpperCase());
6  }
```

- Niezależnie od sposobu zakończenia wykonywania bloku zasoby `in` i `out` zostaną zamknięte.

Wywoływanie wyjątków

- Wyjątek wywołujemy używając słowa kluczowego `throw`.
- W przypadku konieczności wywołania wyjątku zastanawiamy się jakiej klasy wyjątku użyć i rzucamy jej instancję.
- Przykładowo, jeżeli otrzymaliśmy informację o końcu pliku, a logika wskazuje, że nie powinien on nastąpić to rzucamy wyjątek końca pliku.
  - `throw new EOFException();`
- Do wyjątku możemy dodać informację opisującą okoliczności jego wystąpienia

```
1  String gripe = "W pliku " + nazwaPliku + ", nastapil
       niespodziewany koniec pliku w lini: " + numerLinii;
2  throw new EOFException(gripe);
```

- Możemy też utworzyć własną klasę wyjątku.

Tworzenie własnej klasy wyjątków

- Klasę wyjątku tworzymy nadpisując klasę `Exception` lub jedną z jej podklas.
- Klasa zawiera
  - konstruktor bezparametryczny,
  - konstruktor przekazujący tekst wiadomości
  - metodę zwracającą wiadomość `getMessage`

Klasa wyjatku

```
1  class FileFormatException extends IOException{
2      public FileFormatException() {}
3      public FileFormatException(String gripe)
4      {
5          super(gripe);
6      }
7  }
```

- Już sam fakt wyrzucenia wyspecjalizowanego wyjątku niesie dodatkową informację, której nie mamy przy wyrzuceniu standardowego wyjątku.

Łańcuchy wyjątków

- Może być wskazane wyrzucenie wyspecjalizowanego wyjątku po przechwyceniu ogólnego wyjątku

```
1  try{
2      readFile(fileName);
3  }
4  catch (IOException e){
5      throw new FileFormatException("Blad formatu pliku:
           " + e.getMessage());
6  }
```

- Aby nie tracić informacji o pierwotnym wyjątku można utworzyć łańcuch wyjątków

```
1  catch (IOException e){
2      Throwable se = new FileFormatException("Blad
           formatu pliku:");
3      se.initCause(e);
4      throw se;
5  }
```

- Pierwotny wyjątek da się odczytać przy pomocy metody `getCause` klasy `Throwable`.

Testowanie kodu

- Możliwość rzucania błędów może być wykorzystana do testowania kodu.
- Nie stosuje się do tego wyjątków, bo spowolniałyby pracę aplikacji w normalnych warunkach.
- Zamiast tego stosujemy mechanizm *asercji*.

Asercje

- W Javie dostępne jest słowo kluczowe `assert`, które sprawdza prawdziwość zadanego warunku.
  - `assert warunek;`
  - `assert warunek:zwracane_wyrazenie;`
- Jeżeli warunek jest spełniony wykonywanie kodu jest kontynuowane.
- Jeżeli warunek nie jest spełniony wyrzucany jest błąd `AssertionError` nadpisujący klasę `Error`.

Wyliczanie pierwiastka

- Spróbujmy wyliczyć pierwiastek kwadratowy z podanej liczby.

```
1  double x = externalObject.getPositiveX();
2  double y = Math.sqrt(x);
```

- Podczas kodowania, istnieje niebezpieczeństwo, że nie dopilnowano, aby metoda `getPositiveX` dostarczała nieujemny wynik.
- Zabezpieczamy się przed tym stosując asercję

```
1  assert x>=0:"pierwiastek z " + x;
2  double y = Math.sqrt(x);
```

Wynik

```
Exception in thread "main" java.lang.AssertionError: pierwiastek z -1.0 at
pl.edu.pw.mini.mluckner.op.lecture08.AssertionTest.main(AssertionTest.java:6)
```

Włączanie asercji

- Przy standardowych ustawieniach asercje są wyłączone.
- Uruchamiamy je poprzez użycie opcji *-enableassertions* lub *-ea* w parametrach maszyny wirtualnej.

Wywołanie

```
java -enableassertions MyApp
```

- Asercji używa się tylko podczas testowania oprogramowania, do sprawdzenia czy metody poprawnie przesyłają parametry.
- Nie trzeba ich usuwać z kodów w upublicznionej wersji oprogramowania, aplikacje są po prostu uruchamiane bez ich aktywacji.

Bibliografia