Kurs programowania

Wykład 3

Wojciech Macyna

Wyjątki

Dobrze napisany kod powinien zakładać możliwość wystąpienia sytuacji wyjątkowych. Zazwyczaj jednak nie da się podjąć żadnych działań naprawczych w miejscu ich wykrycia. Trzeba przerwać normalny tok działania programu i przekazać informacje o błędzie „na zewnątrz”, do szerszego kontekstu (do metody, która wywołała daną metodę lub jeszcze dalej), gdzie mogą być podjęte działania naprawcze.

Mechanizm obsługi wyjątków

Nowoczesne obiektowe języki programowania mają wbudowany specjalny mechanizm ułatwiający i upraszczający radzenie sobie z obsługą sytuacji wyjątkowych. W chwili wykrycia takiej sytuacji można stworzyć specjalny obiekt nazywany wyjątkiem (ang. exception), zawrzeć w nim pewne informacje i przy pomocy specjalnej instrukcji `throw` zgłosić ten wyjątek do obsłużenia. Zgłoszenie wyjątku wymusza przerwanie normalnego trybu wykonywania programu i zwinięcie stosu wywołań, aż do napotkania kontekstu zawierającego kod obsługi dla wyjątków tego rodzaju.

Deklarowanie wyjątków

Deklarowanie wyjątku (Java)

Wyjątki muszą być klasą pochodną klasy `Exception`.

```
class MojException extends Exception {};
```

Deklarowanie wyjątku (C++)

Wyjątki mogą być typu podstawowego (`int` lub `string`) lub dowolnie zdefiniowaną klasą.

Deklarowanie użycia wyjątku przez metodę (Java)

```
void mojaMetoda() throws MojException;
```

Deklarowanie użycia wyjątku przez metodę (C++)

```
void mojaMetoda() noexcept(false);
```

Metoda nie wyrzuci wyjątku. W razie problemów program będzie przerwany (C++)

```
void mojaMetoda() noexcept;
```

Wywoływanie wyjątku

Wywołanie wyjątku (Java)

```
throw new MojException();
```

Wywołanie wyjątku (C++)

```
throw (string)"MojException";
```

Przechwytywanie wyjątku

Instrukcje `try` i `catch`

```
try {
  <metoda mogąca zwrócić wyjątek>
}
catch(typ_wyjątku e) {
  <obsługa wyjątku>
}
```

Po jednym bloku `try` może być kilka bloków `catch` – zadziała pierwszy w którym dopasuje się typ wyjątku (działają tu zasady dziedziczenia i polimorfizmu).

Klasa `Exception` w Java-ie

Główne konstruktory

- `Exception(String message)` – konstruktor inicjowany komunikatem błędu;
- `Exception()` – wyjątek bez komunikatu błędu.

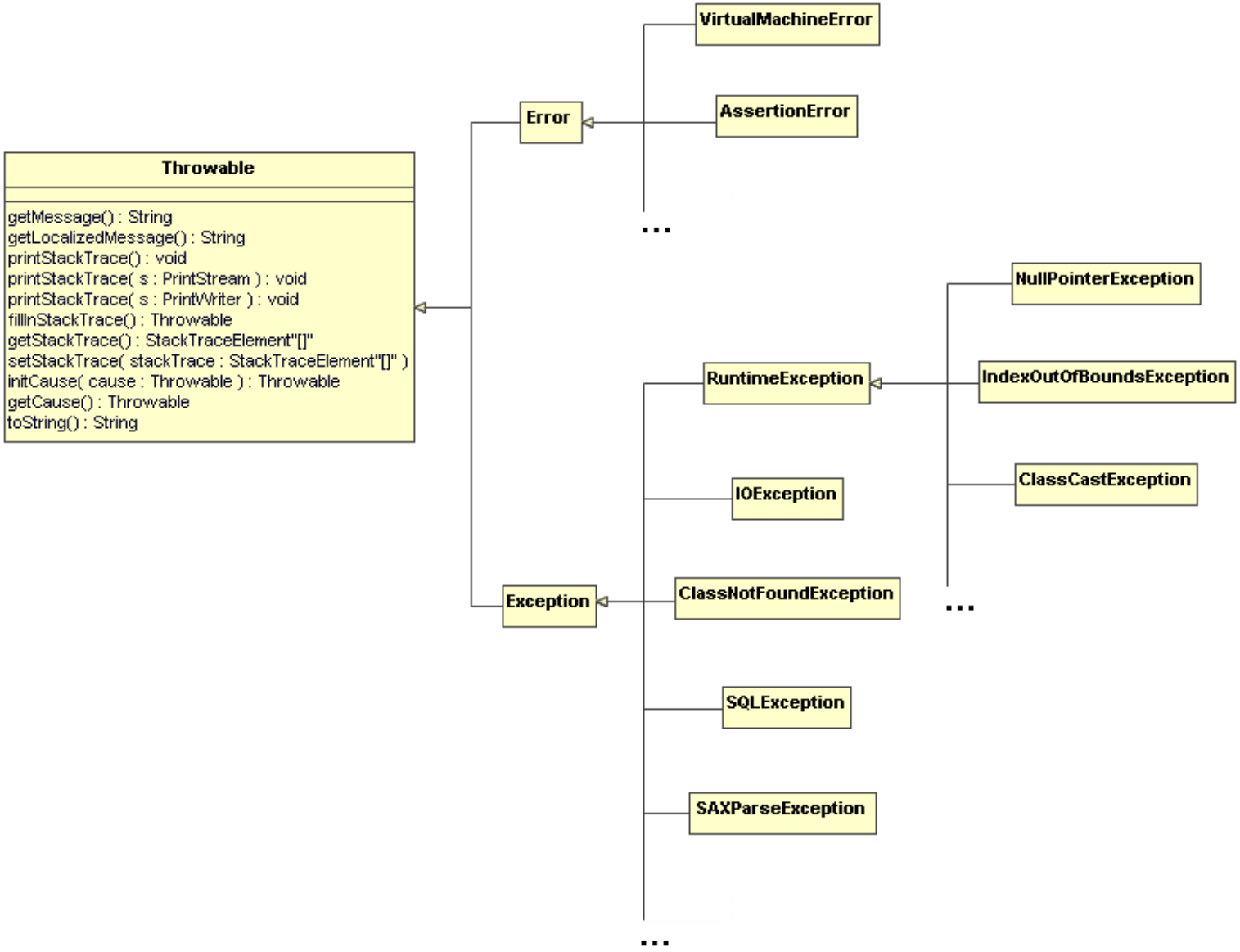
Metody

- `getMessage()` – zwraca komunikat błędu.

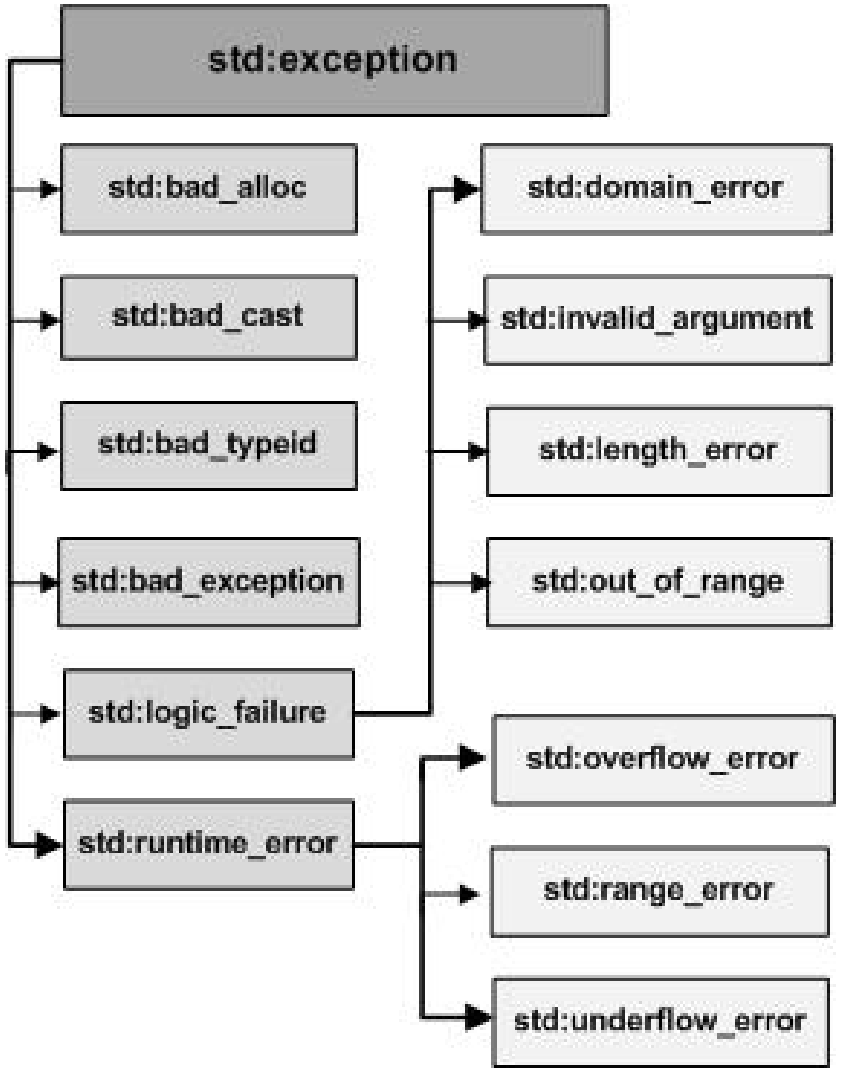
Przykłady

Wyj.cpp, *Wyj1.cpp*, *Wyj2.cpp*, *ExceptionTest.java*

Hierarchia klas wyjątków w Java-ie



Hierarchia klas wyjątków w bibliotece standardowej C++



Wyjątki - podsumowanie

Wyjątki zostały wprowadzone do języków programowania, ponieważ obsługa wszystkich sytuacji wyjątkowych mogących wystąpić w trakcie wywoływania każdej kolejnej metody jest zbyt uciążliwa. Dzięki obsłudze błędów z całego bloku instrukcji, ilość kodu koniecznego do obsługi sytuacji wyjątkowych ulega ograniczeniu.

Mechanizm obsługi wyjątków pozwala przenieść kod obsługi sytuacji wyjątkowej z dala od miejsca jej wykrycia. Zgłoszenie wyjątku przerywa normalny przepływ sterowania i rozpoczyna rozwijanie stosu wywołań, aż do napotkania kodu obsługującego ten typ wyjątku. Dzięki temu, jeżeli w danym miejscu nie wiadomo jak zareagować na dany wyjątek, po prostu się go ignoruje i obsługuje gdzie indziej. Zebranie i przeniesienie w dogodne miejsce kodu radzącego sobie z sytuacjami wyjątkowymi sprawia, że reszta programu staje się bardziej przejrzysta i spójna.

Klasy wewnętrzne

Klasa wewnętrzna

```
class A {
....
     class B {
     ....
     }
....
}
```

Klasa B jest klasą wewnętrzną w klasie A. Klasa A jest klasą otaczającą klasy B.

Klasy wewnętrzne

Właściwości

- Może być zadeklarowana ze specyfikatorem private (normalne klasy nie!), uniemożliwiając wszelki dostęp spoza klasy otaczającej
- Może odwoływać się do niestatycznych składowych klasy otaczającej (jeśli nie jest zadeklarowana ze specyfikatorem static)
- Może być zadeklarowana ze specyfikatorem static (normalne klasy nie!), co powoduje, że z poziomu tej klasy nie można odwoływać się do składowych niestatycznych klasy otaczającej (takie klasy nazywają się zagnieżdżonymi)
- Może mieć nazwę (klasa nazwana)
- Może nie mieć nazwy (wewnętrzna klasa anonimowa)
- Może być lokalna – zdefiniowana w bloku (metodzie lub innym bloku np. w bloku po instrukcji if)

Klasy wewnętrzne

Zastosowanie

- Klasy wewnętrzne mogą być ukryte przed innymi klasami pakietu (względy bezpieczeństwa).
- Klasy wewnętrzne pozwalają unikać kolizji nazw (np. klasa wewnętrzna nazwana Vector nie koliduje nazwą z klasą zewnętrzną o tej samej nazwie).
- Klasy wewnętrzne (w szczególności anonimowe) są intensywnie używane przy implementacji standardowych interfejsów Javy.
- Anonimowe klasy wewnętrzne pozwalają na traktowanie fragmentów kodu do wykonania (ściślej: metod przedefiniowywanych w tych klasach) jak obiektów.

Anonimowe klasy wewnętrzne

Cechy charakterystyczne

- Anonimowe klasy wewnętrzne nie mają nazwy.
- Najczęściej tworzymy klasy wewnętrzne po to, by przedefiniować jakieś metody klasy dziedziczonej przez klasę wewnętrzną bądź zdefiniować metody implementowanego przez nią interfejsu na użytek jednego obiektu.

Klasa wewnętrzna

Definicję anonimowej klasy dostarczamy w wyrażeniu new.

```
1     new NazwaTypu( parametry ) {
2          // pola i metody klasy wewnętrznej
3     }
```

NazwaTypu – nazwa nadklasy (klasy dziedziczonej w klasie wewnętrznej) lub implementowanego przez klasę wewnętrzną interfejsu
parametry – argumenty przekazywane konstruktorowi nadklasy; w przypadku gdy NazwaTypu jest nazwą interfejsu lista parametrów jest oczywiście pusta (bo chodzi o implementację interfejsu).

Anonimowe klasy wewnętrzne

Uwagi

- Anonimowe klasy wewnętrzne nie mogą mieć konstruktorów (bo nie mają nazwy)
- Za pomocą anonimowej klasy wewnętrznej można stworzyć tylko jeden obiekt, bo definicja klasy podana jest w wyrażeniu `new` czyli przy tworzeniu obiektu, a nie mając nazwy klasy nie możemy potem tworzyć innych obiektów
- Definiowanie klas wewnętrznych implementujących interfejsy stanowi jedyny dopuszczalny przypadek użycia nazwy interfejsu w wyrażeniu `new`
- Anonimowe klasy wewnętrzne są kompilowane do plików .class o nazwach automatycznie nadawanych przez kompilator

Typy wyliczeniowe

Typ wyliczeniowy

Definicja typu wyliczeniowego polega na umieszczeniu po słowie enum w nawiasach klamrowych elementów wyliczenia, rozdzielonych przecinkami:

```
[ public] enum NazwaTypu {
    elt1, elt2, ..., eltN
}
```

gdzie:
elt - elementy wyliczenia

Typy wyliczeniowe

Prosty typ wyliczeniowy

```
1 public enum Kolor {
2     CZERWONY, ZIELONY, NIEBIESKI;
3 }
```

Jako klasa wewnętrzna

```
1 public class EnumTest {
2     public enum Kolor {
3         CZERWONY, ZIELONY, NIEBIESKI;
4     }
5 }
```

Wywołanie

```
1 public static void main(String[] args) {
2     EnumTest.Kolor kolor = EnumTest.Kolor.CZERWONY;
3     if(kolor.equals(EnumTest.Kolor.CZERWONY)) {
4         System.out.println("Zgadza␣sie");
5     }
6  }
```