Л Е К Ц И Я №3

Обработка на изключения в Java

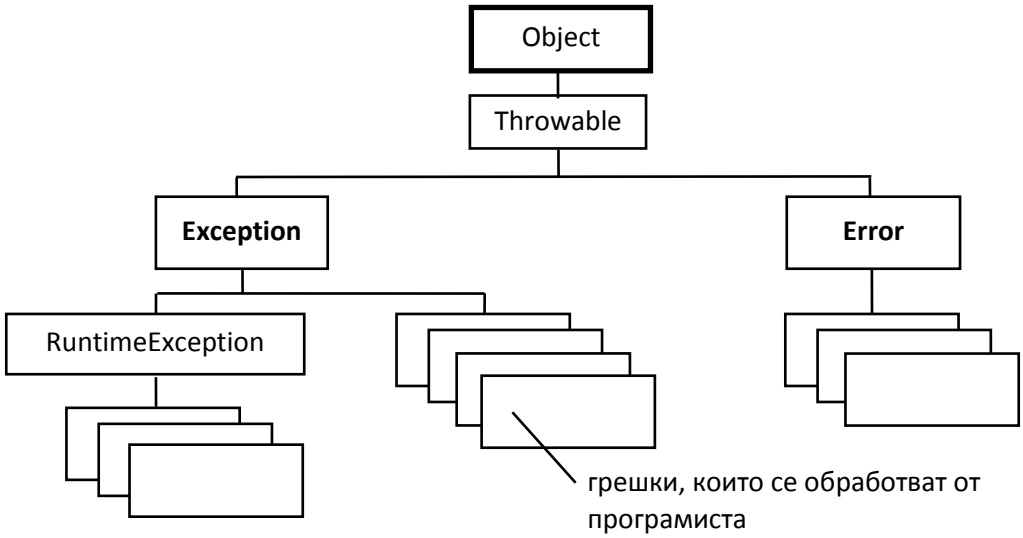
I. ВЪВЕДЕНИЕ

Всеки програмен език се справя по различен начин с възникващите по време на изпълнение грешки. Например, С функциите връщат код на грешката, който трябва да се обработи от програмиста чрез анализ на стойността на този код (например -3). Този начин на обработка на грешки е прекалено досаден - някои функции връщат десетки кодове на грешки. Теоретично би трябвало да се напише код, обработващ всяка възможна грешка. В този случай размерът на кода, обработващ възможните грешки, ще бъде прекалено голям и логиката на програмата ще се проследява по-трудно. На практика, в зависимост от своя опит, програмистът решава кои грешки да обработи и кои не. Това обаче влошава надеждността на приложенията.

Java използва коренно различна стратегия за обработка на грешки. Когато в тялото на даден метод възникне грешка се създава специален *обект*, който съдържа информация за типа и мястото на възникване на грешката. Грешките в Java са два основни типа:

- *Фатални грешки* - възникват основно поради неправилно функциониране на JVM. Обектите, които ги описват, имат един родителски клас - Error. Този тип грешки не се обработват от програмиста. При възникването им програмата се терминира от JVM.
- *Изключения* – тези грешки не са фатални за функционирането на JVM, но прехващането и обработката им е задължителна, за да може да се гарантира нормалната логика на програмния код. Обектите, които ги описват, имат един родителски клас - Exception.

Базовият клас за Error и Exception е Throwable, както е показано на Фиг. 3.1.



Фиг. 3.1. Йерархия на класовете в Java, които описват грешки

Един по-особен подклас на Exception е RuntimeException. Този тип изключения се обработват от JVM. Пример за подобно изключение е NullPointerException, което се получава при опит да се адресира неинициализиран обект (с нулева референция). Тази грешка не е фатална, тъй като след като обектът се инициализира, той ще може да се адресира. Идеята при Java, относно обработката на изключения, е да не се компилира програмен код, който може да генерира изключение. Следователно, Java компилаторът ще застави програмиста да обработи по подходящ начин кода от методите, който може да генерират грешка. Всички изключения, които задължително трябва да бъдат обработени, се наричат *контролирани изключения* (checked), а останалите - за които се грижи компилаторът или JVM – *неконтролирани изключения* (unchecked). Неконтролирани изключения са всички фатални грешки и тези от клас RuntimeException. Най-често възникващите грешки от този клас са:

- ArithmeticException - генерира се, ако аритметичен израз върне резултат, който е извън границите на допустимите стойности, например при деление на нула;
- ArrayIndexOutOfBoundsException - опит за достъп до елемент от масив с индекс по-малък от 0 или по-голям от общия брой на елементите в масива -1;
- NullPointerException - опит за работа с все още неинициализиран обект.

Този тип изключения могат да възникнат на всяко място в програмния код и са по вина на програмиста. Той, ако желае, може да прехване изключенията от клас RuntimeException, но може и да ги пропусне! *Контролираните изключения* изискват обслужване. То се реализира по два начина:

1. Прехващане и обслужване чрез клаузи try-catch-finally.
2. Деклариране, че изключенията, които даден метод може да генерира, ще бъдат обслужени от друг метод или от JVM.

Най-често възникващите контролирани изключения са:

- IOException - генерира се при неуспешна входно-изходна комуникация. Както ще видим по-късно Java използва еднотипен начин за комуникация чрез входно-изходни канали, независимо кой е предавателя и получателя (клавиатура, файл или сървър);
- FileNotFoundException - опит за достъп до несъществуващ файл;
- OutOfMemoryException - генерира се при недостиг на динамична памет.

II. ПРЕХВАЩАНЕ И ОБРАБОТКА НА ИЗКЛЮЧЕНИЯ

Java позволява лесно да се декларират областите от кода на приложението, които могат да генерират изключение (*охранявана зона*) и кодът, който ще реализира обработката на изключението (*манипулатор на изключението*). За целта се използва клауза try, за да се зададат рамките на охраняваната зона и клауза catch, за да се дефинира манипулатора на изключението. Синтаксисът е следния:

```
try {
    // охранявана зона
}
catch (КласНаИзключението  обектКойтоГоОписва) {
    // манипулатор на изключението
}
```

Можете да напишете множество манипулатори за една охранявана зона, като за целта използвате колкото са необходими catch клаузи. Единственото изискване е да се спазва

йерархията на изключенията - всеки следващ манипулатор трябва да обработва събитие от клас, който е на по-ниско ниво в йерархията от класове, описващи изключения. Следва пример за грешна обработка на изключения:

```
try {
    // охранявана зона
}
catch (Exception e) {
    // Манипулатор на изключението. Обслужва всички възможни грешки,
    // тъй като е указан базовия клас Exception
}
catch (ArithmeticException ae) {
    // манипулатор на изключенията от тип ArithmeticException
}
```

В случая, вторият манипулатор, обработваш аритметично изключение, никога няма да се активира, тъй като предходният манипулатор обработва всички изключения. Проблемът се решава като се размени последователността на манипулаторите.

Java, за разлика от C++, позволява използването и на клауза finally, която следва клаузи try и catch. Управлението се предава на тялото на клауза finally, независимо дали е възникнало изключение в охраняваната зона, или не е. Тази клауза не е задължителна, но има своето практическо приложение. Например, ако даден метод реализира комуникация в мрежова среда, изградените в try блока обекти за входно-изходни канали, е най-добре да бъдат унищожени в тялото на finally блока, тъй като това трябва да се реализира независимо дали е имало или не грешка при комуникация.

Позволените комбинации за клаузи try, catch и finally са следните:

- try-catch-catch-...-finally;
- try-catch-...-catch;
- try-finally.

III. ПРОГРАМНО ГЕНЕРИРАНЕ НА ИЗКЛЮЧЕНИЯ

Java позволява програмистът да генерира изключения. Това се реализира чрез ключовата дума throw. Например, ако не сме сигурни че обектът myObject е инициализиран, може да се използва следния програмен код:

```
if (myObject == null) {
    throw new NullPointerException();
}
```

След ключовата дума throw се създава обект в хийпа, който описва типа на изключението (вика се конструктора на съответния клас, в случая NullPointerException). Докато при try-catch клаузите обработката на изключението се реализира в тялото на catch блока, то при генериране на изключение не се знае къде точно е манипулаторът, който ще го обслужи. Задачата на JVM е да открие манипулатора, който ще обслужи изключението. Списъкът от методи, които евентуално би трябвало да обслужат изключението, се нарича *call-стек*. Java не задължава методът, извикал програмен код, който може да генерира изключение, да съдържа манипулатор за него. Важното е поне един от методите от call-стека да има такъв манипулатор.

Възможно е да съсредоточим обработката на едно изключение в конкретен метод от call стека. Така, програмният код ще стане по-кратък и разбираем. Пренасочването на обработката на едно или няколко изключение от един метод към друг метод е възможно чрез ключова дума throws. За целта, след името на метода при деклариране му чрез ключовата дума throws се описват изключенията, които методът може да генерира. Манипулаторите, които ги обработват, могат да се намират в кой да е от методите от call-стека. Например, ако знаем че в метод main може да възникне входно-изходна грешка, но не желаем да се ангажираме с обработката й, можем да използваме следния код:

```
public static void main(String[] args) throws IOException {
    // код, който може да генерира I/O грешка
}
```

В този случай, тъй като метод main се вика от JVM, то тя трябва да се погрижи за евентуални входно-изходни грешки, възникнали при изпълнение на кода от тялото на метод main.

IV. ПОТРЕБИТЕЛСКИ ТИП ИЗКЛЮЧЕНИЯ

Java позволява и създаването на *потребителски тип изключения*. Тъй като изключенията са обекти с родителски клас Exception, потребителските изключения се получават като се декларира клас, който наследява клас Exception или негов наследник. Когато генерирате изключението чрез ключова дума new ще се изпълни кода от конструктора на така декларирания клас. Следва пример за създаване на потребителски тип изключение чрез клас ShapeException, който наследява клас RuntimeException (дъщерен на Exception). Логиката се състои само в предаване на низ с информация за изключението, параметър message.

```
public class ShapeException extends RuntimeException {
    /**
     * Creates a new instance of <code>ShapeException</code> without detail
     * message.
     * @param message
     */
    public ShapeException(String message) {
        super(message);
    }
}
```

Следва пример за използване на изключението:

```
try {
    // код, който генерира ShapeException, например:
    throw new ShapeException("Информация за грешката");

} catch (ShapeException e) {
    // обработка на изключението (само печат на конзолата)
    String info = e.getMessage();
    System.out.println(info);
}
```

V. ВЪПРОСИ И ЗАДАЧИ ЗА ИЗПЪЛНЕНИЕ

1. Анализирайте клаузите, които се използват при програмни езици C++, C# и Java с цел обработка на изключения?

2. Какво е предназначение на call-стека в Java? Има ли аналог на call-стека в C#?

3. Кой тип изключения могат да бъдат пропуснати и кой тип изключения не могат да не се обработят от програмиста – компилаторът ще генерира грешка?

4. Да се напише Java конзолно приложение, което проверява дали една година е високосна или не е. Това да става от специално написан метод isLeap, който връща true, ако годината е високосна и false – ако не е високосна. В тялото на метод main декларирайте масив, който съдържа N на брой години, които трябва да бъдат проверени.

Една година е високосна е високосна, ако:
(дели се без остатък на 4) И (дели се без остатък на 400 ИЛИ се дели с остатък на 100).

На Фиг. 3.1 е показан метод isLeap(), който имплементира условието една година да е високосна. Методът получава един атрибут year – стойността на годината и връща булев резултат, както е зададено по условие на задачата. Сложно съставното условие реализираме с един if оператор. За да анализираме стойността на остатъка от деленето на две числа използваме оператор %.

```
private static boolean isLeap(int year) {
    boolean result = false;
    if ((year % 4 == 0) && (year % 100 != 0) || (year % 400 == 0)) {
        result = true;
    }
    return result;
}
```

Фиг. 3.1. Задача 4, метод isLeap()

На Фиг. 3.2 е показан програмния код от тялото на метод main чрез който се реализира необходимата логика. Масивът с годините е years. Броят на годините в масива N се извлича програмно. Следва for цикъл чрез който: 1) Извлича се година от масива year и 2) Проверява се дали годината е високосна или не е чрез викане на метод isLeap().

След изпълнение на програмата, на конзолата се получава следния резултат:

```
Годината 2019 не е високосна.
Годината 2020 е високосна.
Годината 2021 не е високосна.
```

Програмният код работи коректно, но има следния недостатък: Ако трябва да проверим други години, трябва да компилираме приложението наново. По добрият вариант е когато годините се задават от потребителя на приложението, например през клавиатурата или чрез файл.

```
public static void main(String[] args) {
    int years[] = {2019, 2020, 2021};
    String status;

    int N = years.length;
    for (int i = 0; i < N; i++) {
        int year = years[i];
        if (isLeap(year)) {
            status = "е високосна";
        }
        else {
            status = "не е високосна";
        }
        String info = String.format("Годината %d %s.", year, status);
        System.out.println(info);
    }
}
```

Фиг. 3.2. Задача 4, метод main()

5. Да се напише модификация на Задача 4, която да позволява въвеждане на годините от клавиатурата.

Клавиатурата предава към компютъра позиционните кодовете на всеки натисна или отпуснат бутон (1 байт). Тези кодове се преобразуват от OS до символи (ако бутона има символ – буква или число). За целта ще използваме възможностите на Java за работа с потоци от данни. Ще използваме форматирано четене от потока – потока от символи ще преобразуваме до поток от цели числа, тъй като годините са цели числа.

Програмния код, който създава комуникационен канал към клавиатурата и чете въведените данни, до натискане на бутон Enter, е следния:

```
Scanner input = new Scanner(System.in);
int n = input.nextInt();
```

Комуникационният канал се създава чрез конструктора на клас Scanner. Източникът на данните се задава като аргумент към конструктора. В конкретният случай System.in. По подразбиране поле in от клас System се асоциира със стандартния вход – клавиатурата. Полученият низ, който съдържа символните кодове на бутоните (без Enter) се преобразуват до цяло число чрез метод nextInt(). Този метод чете от потока и автоматично адресира следващите данни в него, ако има такива.

Ако трябва да изчистите данните в потока, изпълнете метод next(). Това може да се наложи, ако потребителя е въвел неверни данни. Тези данни трябва да бъдат изтрита от потока чрез next() преди отново да извикате метод nextInt()!

На Фиг. 3.3 е показано едно примерно решение на Задача 5 (част от кода в тялото на метод main). Започваме с подкана да се въведат от клавиатурата броя на проверяваните години. Следва създаване на обект input за потоков обмен с клавиатурата.

```
System.out.print("Моля, въведете броя на годините: ");
Scanner input = new Scanner(System.in);
int n = input.nextInt();
if (n <= 0) {
    System.out.println("Очаква се да въведете цяло положително число!");
    return;
}
// ------------------------------------------------------------------------
int i = 0;
int years[] = new int[n];
do {
    System.out.print("Година " + (i + 1) + ": ");
    int year = input.nextInt();
    if (year > 0) {
        years[i] = year;
        i++;
    } else {
        System.out.println("Невалидна година: " + year);
    }
} while (i != n);
```

Фиг. 3.3. Примерно решение на Задача 5

Броят на годините се записва в променлива n. Ако стойността на n е невалидна, потребителя се уведомява за това, а програмата се терминира. Това се реализира чрез връщане от метод main() - оператор return.

След получаване на стойността на n се организира do-while цикъл чрез който се четат n години от клавиатурата и за всяка от тях се вика метод isLeap(). След изход от цикъла всички въведени години са в масив years (вижте как точно е деклариран този масив, при условие, че броя на елементите в него е известен – n).

Показването на резултата се реализира чрез програмния код от Задача 4:

```
Моля, въведете броя на годините: 3
Година 1: 2019
Година 2: 2020
Година 3: 2021
Годината 2019 не е високосна.
Годината 2020 е високосна.
Годината 2021 не е високосна.
```

Този вариант на задачата може да валидира входните данни, но само частично. Ако n не е по-голямо от 0 и ако е въведено отрицателно число за стойност на година. Какво ще стане, ако за n зададем нещо, което не е число, например низа „пет”?

На Фиг. 3.4 е показано какво се печата на конзолата в този случай. Програмата прекъсва аварийно своето изпълнение, тъй като входните данни са невалидни. Грешката се получава от метод nextInt() – ред 23 от файл PIS_LEC3_TASK5_V1.java. Този метод очаква да му се подаде като аргумент цяло число от тип int, а получава низ. Проблемът може да се реши чрез прехващане и обработка на изключения.

```
Моля, въведете броя на годините: пет
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:864)
        at java.util.Scanner.next(Scanner.java:1485)
        at java.util.Scanner.nextInt(Scanner.java:2117)
        at java.util.Scanner.nextInt(Scanner.java:2076)
        at pis_lec3_task4_v2.PIS_LEC3_TASK5_V1.main(PIS_LEC3_TASK5_V1.java:23)
Java Result: 1
```

Фиг. 3.4. Генериране на изключение от тип InputMismatchException при невалидни входни данни

6. Да се напише модификация на Задача 5, която валидира входните данни чрез използване на изключения.

Задачата ще бъде решена на Семинарно упражнение №3.

7. Да се валидират чрез потребителски тип изключение данните, които се предават на конструктура на клас Student (Задача 1 от Семинарно упражнение №2).