Raport

Cel:

Porównanie kosztów obsługi wyjątków ze względu na rodzaj obsługiwanego wyjątku, a także ze względu na metodykę jego obsługi (wewnątrz funkcji, w której został wygenerowany, bądź pochwycenie w funkcji zewnętrznej)

Metodyka:

Utworzenie par prostych funkcji, gdzie jedna jest zakodowana w sposób błędny – generujący wyjątki, a druga wywołuję ową funkcję podaną ilość razy, mierząc czasy egzekucji i uśredniając je, a następnie zwraca uśredniony wynik w 10-krotnej pętli w funkcji main, gdzie owe wyniki są zapisywane na ekranie. Zaproponowałem wygenerowanie następujących wyjątków – ArrayIndexOutOfBoundsException, ArithmeticException (Unchecked Exceptions), oraz IOException i NotSerializableException (Checked Exceptions). Każdy został obsłużony w dwa sposoby, tzn poprzez lokalne przechwycenie wyjątku try-catch, a także przekazanie go do metody wyższej, odpowiedzialnej za iterowanie funkcji generującej wyjątek i łapiącej go przy każdej iteracji. Koszta czasu obsługi wyjątków zostały porównane używając różnicy wartości System.nanoTime() po i przed wywołaniem metody generującej wyjątek. Dla uzyskania dokładności pomiarów, każdą z 4 opcji iteracji (1, 100, 1000 i 10000-krotnie) powtórzyłem 10 razy.

1. IOException
   Kod dwóch metod, z czego jedna generuje i wyrzuca wyjątek IOException związany z brakiem możliwości wczytania pliku, druga metoda natomiast wywołuje tą pierwszą i łapie przekazany w niej wyjątek. Cały proces odbywa się n razy, a następnie zwracany jest uśredniony wynik. czasu obsługi omówionego procesu

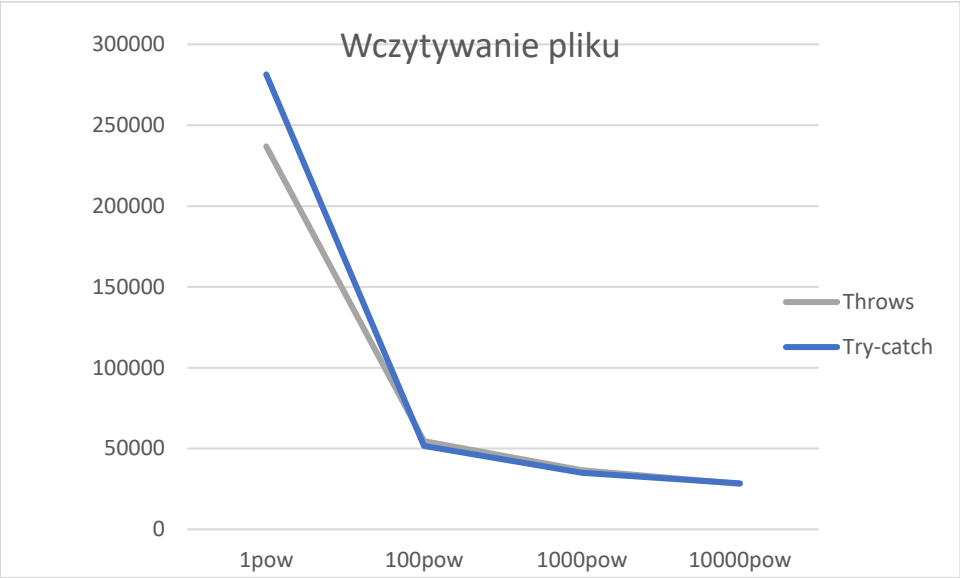
```
1 usage
public static void czytaniePliku() throws IOException
 {
            FileReader fr = new FileReader( fileName: "x.txt");
            BufferedReader br = new BufferedReader(fr);
}
```

```java
public static double iteracjaCzytaniaPliku(int iloscIteracji){
    double beforeTime;
    double afterTime;
    double srednia = 0;

    for(int i=0;i<=iloscIteracji;i++){
        beforeTime = System.nanoTime();
        try {
            czytaniePliku();
        }catch(IOException x){}
        afterTime = System.nanoTime();

        srednia+=(afterTime-beforeTime);
    }
    return srednia/iloscIteracji;

}
```

Poniżej natomiast są przedstawione wyniki pomiarów

Odczyt pliku (try-catch)	Kolumna1	Kolumna2	Kolumna3	Kolumna4
No	1 pow	100 pow	1000 pow	10000pow
1	703300	93320	55417,8	29044,82
2	269100	72846	35546	27778,94
3	175300	53516	32881,1	27414
4	156600	49762	33574,1	26696,43
5	185300	51427	32549,3	26652,28
6	190600	43074	35223	27828,85
7	162000	40050	31943,7	26220,43
8	168500	37745	37822,2	26206,41
9	181600	50812	35742,2	34822
10	176700	52952	35039,4	28329,95
Średnia	236900	54550,4	36573,88	28099,411

Odczyt pliku(throws)				
No	1pow	100pow	1000pow	10000pow
1	558900	93701	50112	37035,82
2	349000	57452	34380,5	28960,69
3	257500	51512	35886	29731,23
4	238100	61409	33028,6	28601,61
5	217600	55422	32092,4	27962,85
6	240300	42897	37597,4	26703,06
7	237900	36644	32773,9	25818,96
8	215700	41851	32270,4	26520,5
9	238900	35245	30759	26496,87
10	260100	39207	29661	26483,03
Średnia	281400	51534	34856,12	28431,462



Następnym przetestowanym w ten sam sposób wyjątkiem jest NotSerializableException, wygenerowany w metodzie próbującej zzerializować obiekt klasy, nie implementującej interfejsu Serializable. Pojawia się tutaj także IOException, jednak ten wyjątek został zadeklarowany tylko z powodu kompilatora, który go wymaga. IOException w tym przykładzie nie jest ani generowany ani obsługiwany, a więc nie ma wpływu na ogólny wynik.

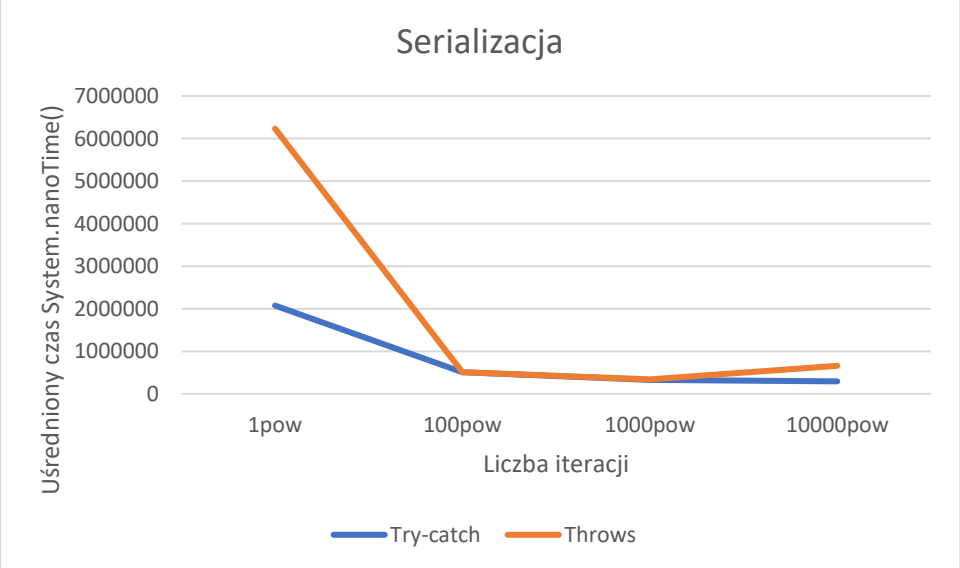
```java
public static double iteracjaSerializacji(int iloscIteracji){
    double beforeTime;
    double afterTime;
    double srednia = 0;

    for(int i=0;i<=iloscIteracji;i++){
        beforeTime = System.nanoTime();
//          try {
  //            try {
                  serializacja();
    //          } catch (NotSerializableException nse) {}
      //      }catch(IOException x){}
        afterTime = System.nanoTime();

        srednia+=(afterTime-beforeTime);
    }
    return srednia/iloscIteracji;

}
```

```java
1 usage
public static void serializacja() //throws NotSerializableException, IOException
 {
    try {
        try {
            KlasaBezMetody x = new KlasaDruga( x: 5);

            FileOutputStream fout = new FileOutputStream( name: "file.txt");
            ObjectOutputStream oout = new ObjectOutputStream(fout);

            oout.writeObject(x);

            oout.close();
        }catch(NotSerializableException nse){}
    } catch (IOException io) {
      //  System.out.println(io);
    }
}
```

Wyniki:

Serializacja obiektu (throws)				
No	1pow	100pow	1000pow	10000pow
1	4368220	961782	503604,3	308625,76
2	1870600	425975	453498,2	267776,37
3	2104500	513322	378140,2	265245,67
4	1853900	486882	315046,8	301419,18
5	1778700	489961	336848,8	420009,84
6	2056700	463179	234498,6	290188,52
7	1637600	350951	229836,1	260983,49
8	1587500	449486	233816,3	254614,71
9	1791400	466587	253070	316063,12
10	1693400	459308	341069,5	239467,99
Średnia	2074252	506743,3	327942,88	292439,465

Serializacja obiektu (try-catch)				
No	1pow	100pow	1000pow	10000pow
1	45463800	995837	521610,3	347288,99
2	2663100	452466	453509,9	298854,69
3	2342599	472237	298186,3	504009,18
4	2034200	361819	371208,5	923356,6
5	1812800	587633	329814,7	849339,46
6	1653700	451355	263649,2	750193,7
7	1668400	369748	255858,6	629126,32
8	1624900	483423	287514,9	674081,38
9	1553299	522663	278358,3	938848,46
10	1537300	343165	335809	667337,35
Średnia	6235409,8	504034,6	339551,97	658243,613



Kolejnym testowanym wyjątkiem jest ArithmeticException wywołane poprzez prosty podział dwóch zmiennych typu int, z czego dzielna ma przypisaną wartość 0.

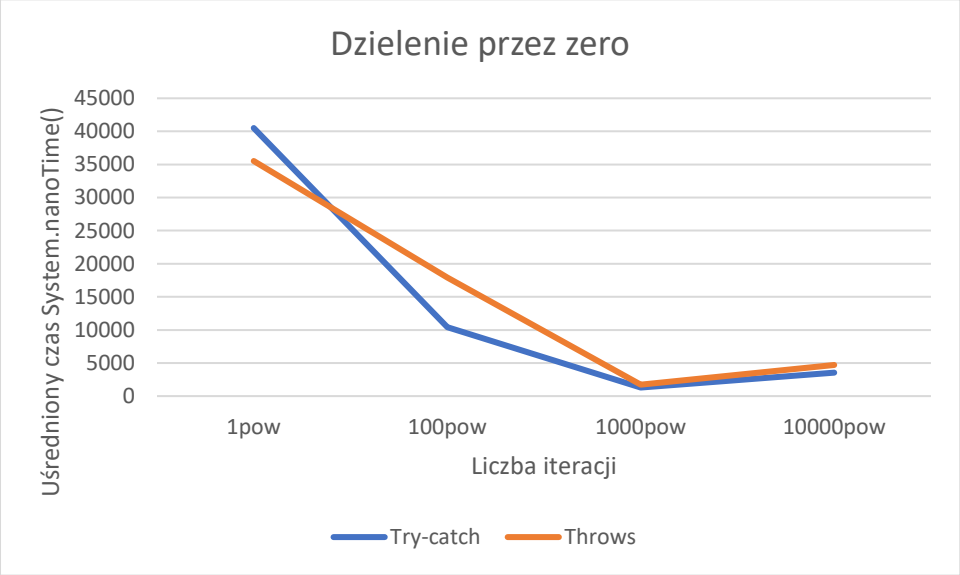
```java
public static int dzielenieZero() //throws ArithmeticException
 {
    try{
    int a = 10;
    int b = 0;
    return a/b;
    }catch(ArithmeticException x){return 0;}


}
```

```java
public static double iteracjaDzielenia(int iloscIteracji){
    double beforeTime;
    double afterTime;
    double srednia = 0;
    for(int i=0;i<=iloscIteracji;i++){
        beforeTime = System.nanoTime();
      //try {
          dzielenieZero();
      //}catch(ArithmeticException x){}
        afterTime = System.nanoTime();

        srednia+=(afterTime-beforeTime);
    }
    return srednia/iloscIteracji;

}
```

Wyniki:

Dzielenie przez zero (throws)				
No	1pow	100pow	1000pow	10000pow
1	29100	14695	14328,4	1685,26
2	47300	7293	1201,9	241,4
3	54700	10212	224,8	221,55
4	30000	12306	221,8	278,89
5	32400	8855	220,2	409,94
6	30500	62040	223,7	302,31
7	37300	12165	223,1	9184,65
8	32900	12496	233,4	13370,76
9	28500	19528	214,1	11851,24
10	32600	19497	213,3	9373,26
Średnia	35530	17908,7	1730,47	4691,926

Dielenie przez zero(try-catch)				
No	1pow	100pow	1000pow	10000pow
1	46700	7843	11093,5	1141,66
2	113800	8951	1367,5	52,29
3	35600	10657	68	53,15
4	33900	10513	64,6	52,81
5	28300	10704	63,1	51,35
6	30800	12091	70,1	52
7	25100	10559	72,9	7700,11
8	27100	10463	46,9	8928,71
9	29900	10470	46,2	9112,66
10	33900	11776	46,3	8243,6
Średnia	40510	10402,7	1293,91	3538,834



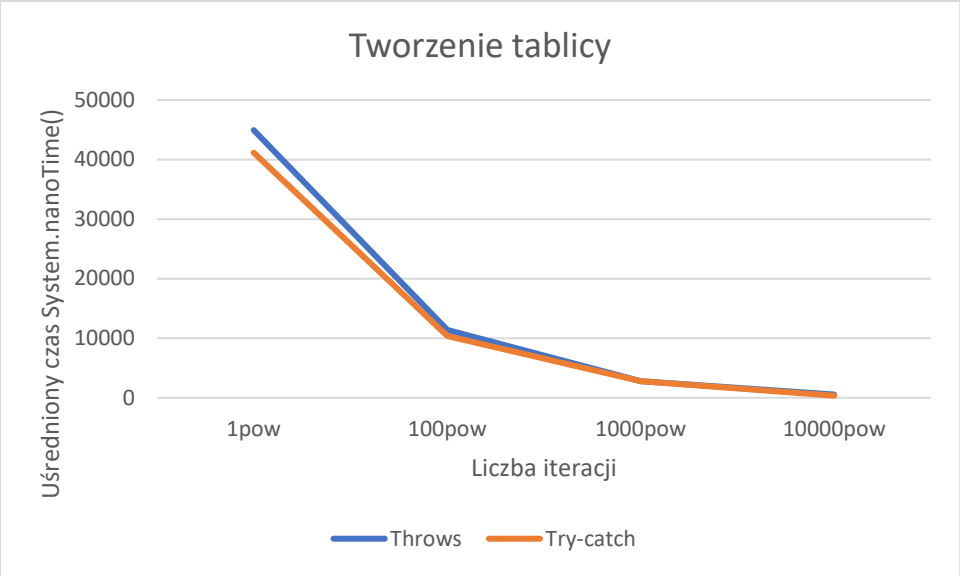
Ostatnią omawianą przeze mnie metodą jest taka, która generuje wyjątek ArrayIndexOutOfBondsException, dzięki prostej iteracji dodającej 6 elementów do 5 elementowej tablicy.

```
public static void tworzenieTablicy() throws ArrayIndexOutOfBoundsException{
    int[] tablica = new int[5];

    //try {
        for (int i = 0; i <= 5; i++)
            tablica[i] = 1;
    //}catch(ArrayIndexOutOfBoundsException aiobe){}
}
```

```java
public static double iteracjaTablicy(int iloscIteracji){
    double beforeTime;
    double afterTime;
    double srednia = 0;
    for(int i=0;i<=iloscIteracji;i++){
        beforeTime = System.nanoTime();
        try {
        tworzenieTablicy();
        }catch(ArrayIndexOutOfBoundsException x){}
        afterTime = System.nanoTime();

        srednia+=(afterTime-beforeTime);
    }
    return srednia/iloscIteracji;

}
```

Tworzenie tablicy(try-catch)				
No	1pow	100pow	1000pow	10000pow
1	69900	8601	10911,3	3081,06
2	53200	10401	7280,4	50,81
3	40400	15951	6838,6	50,28
4	40000	13086	2231,6	49,84
5	32600	11937	49,3	50,82
6	36500	11901	51,8	56,87
7	33600	7292	54	37,23
8	35700	9059	47,8	31,46
9	33900	8240	46,1	30,45
10	35700	6978	49	30,3
Średnia	41150	10344,6	2755,99	346,912

Tworzenie tablicy(throws)				
No	1pow	100pow	1000pow	10000pow
1	91200	9025	8905,8	3226,84
2	70000	15968	8547	206,03
3	39400	17505	7141,5	204,47
4	53100	8407	1958,1	330,49
5	38500	8169	207,6	371,74
6	49000	7178	208,8	247,02
7	38400	7033	229	197,65
8	27600	12988	206,1	151,42
9	19500	13610	209,6	178,45
10	22600	14063	213,2	252,84
Średnia	44930	11394,6	2782,67	536,695



Wnioski:
Porównując poszczególne wykresy, możemy łatwo zauważyć, iż wraz z ilością iteracji drastycznie spada czas egzekucji, wynika to z buforu, pierwsza iteracja jest zazwyczaj wielokrotnie bardziej kosztowna niż kolejne. Jeżeli zaś chodzi o różnice kosztów łapania wyjątków wewnątrz metody, a używania instrukcji throws do wyrzucania jej w wyższych instancjach – różnice owe są znikome i techniki te powinno się stosować w zależności od kontekstu, nie zaś martwiąc się o różnice wydajnościowe.