Dodatek

Author: Piotr Niemczuk
Nickname: Pyoneru

> ⓘ **Info**
>
> Dodatek zawiera informacje które nie chciałem zostawiać w głównym materiale ale też nie chciałem ich usuwać.

Operacje arytmetyczne, a typy danych

Jak już wiesz, w języku Java mamy typy danych. Skupmy się teraz na operacjach arytmetycznych patrząc na ich typ danych, a nie sam wynik - on nas teraz nie interesuje.

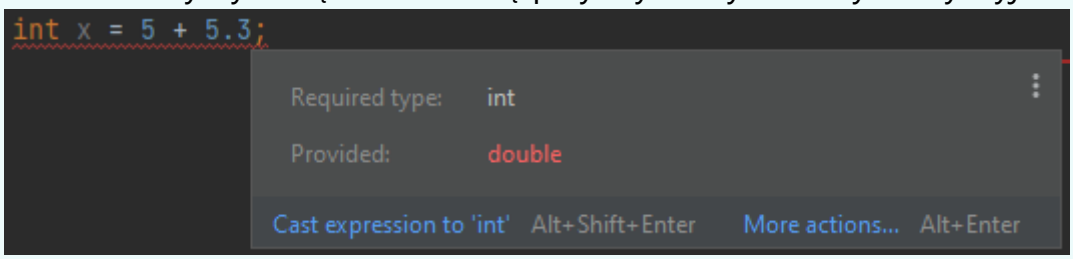
Czy jesteś w stanie się domyślić jaki typ danych zwrócą poszczególne działania?

```
5 + 3
5 + 5.3
0.13 + 1
0.1 + 0.1
1 + 3.14f
0.5f + 0.5f
0.7f + 3
0.8 + 1.3f
```

> ⓘ **Dlaczego to jest ważne?**
>
> Z poprzedniego materiału zapewne pamiętasz rzutowanie i to że nie możesz przypisać liczby zmienno przecinkowej do zmiennej przechowującej liczby całkowite. Wynik operacji arytmetycznych też ma swój typ i obowiozują te same zasady przy przypisywaniu wartości do zmiennej.

> 🔥 **Informacje o błędach**

Jeżeli linia jest podkreślana na czerwono to znaczy że w tym miejscu jest błąd który można wykryć przed kompilacją(uruchomieniem) programu. Jeżeli najedziemy kursorem myszy na tę linie i chwilę przytrzymamy to otrzymamy wyjaśnienie.



`Required type: int` informuje nas że zmienna oczekuje wartości typu int (Lewa strona operatora przypisania)
`Provided: double` informuje nas że przekazujemy wartość typu double (prawa strona operatora przypisania)

Wyjaśnijmy sobie zasady dotyczące typu wynikowego.

Domyślną wartością dla liczb całkowitych jest int, a dla zmienno przecinkowych double.

```
1 // int
3.14 // double
```

ALE

ⓘ Stała dosłowna

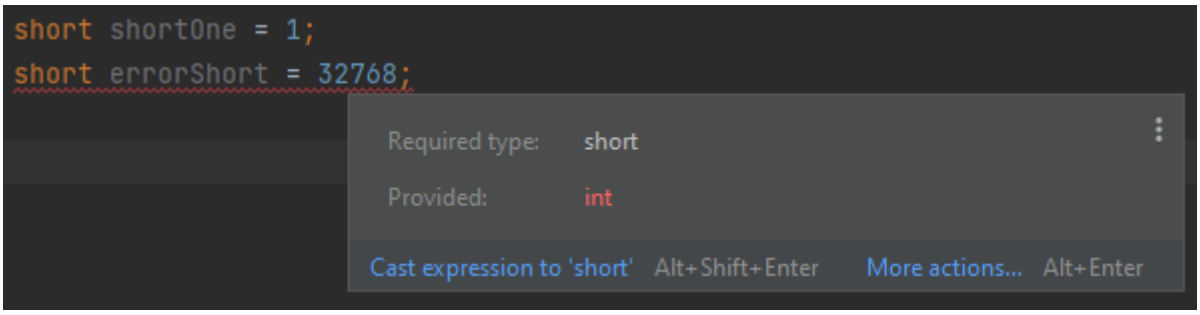
Stałe dosłowne to wszystkie wartości które piszemy jawnie.

```
1 // stała dosłowna
1_000_000 // stała dosłowna
"Ala ma kota" // Stała dosłowna
'X' // Stała dosłowna
3.14 // Stała dosłowna
int x = 5; // 5 jest tutaj stałą dosłowną
int y = x; // x już nie jest stałą dosłowną, to zmienna.
```

Jeżeli stałą dosłowną liczby całkowitej, przypisujemy do typu **byte** lub **short** to nie musimy dodawać sufixów ani innych dodatkowych rzeczy aby powiedzieć że ta liczba jest tego typu. O ile nie przekraczamy zasięgu danego typu.

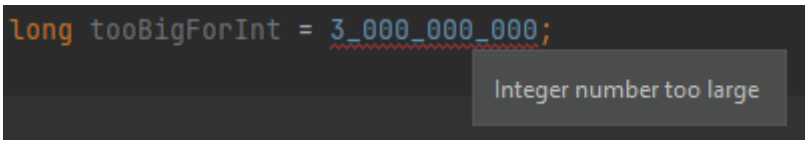
```java
byte bitOne = 1; // dobrze
byte errorByte = 129; // error, przekracza maksymalną wartość dla typu byte

short shortOne = 1;  // dobrze
short errorShort = 32768; // error, przekracza maksymalną wartość dla typu
short
```

Otrzymamy analogiczny komunikat błędu co przy próbuje przypisania **double** do **int**.



Dzieje się tak, ponieważ do póki nie przekraczamy zakresu **byte** lub **short**, to następuje rzutowanie **bez** stratne, czyli nie tracimy wartośći ponieważ mieści się ona w zakresie w oby dwóch typach. W takim przypadku nie musimy jawnie rzutować, tylko java sama się domyśli co ma zrobić.

Natomiast, gdy spróbujemy przypisać do typu **long** stałą dosłowną która mieści się w jej zakresie, ale nie mieści się w zakresie typu **int** to dostaniemy błąd.



Ponieważ domyślnym typem dla stałej dosłownej liczby całkowitej jest **int**.
W tym przypadku sami musimy jawnie powiedzieć że ta stała dosłowna jest typu **long**, poprzez dodanie suffixa **L**.

```java
long tooBigForInt = 3_000_000_000L;
```

> ⓘ **Systemy liczbowe | Ciekawostka**
>
> Jeżeli chcesz wiedzieć dlaczego bitowo `1111` to decymalnie 15 (po naszemu). To zapraszam Cię tutaj.

Niestety tych samych zasad nie możemy zastosować dla typów liczb zmienno przecinkowych.

```
double dPi = 3.14;
float fPi = 3.14;
```

Nawet jeżeli oba typy są w stanie pomieścić tą samą wartość to i tak musimy jawnie powiedzieć że stała dosłowna ma być typu **float** poprzed dodanie suffixa **F**.

Teraz jak wygląda sprawa gdy wykonujemy działania arytmetyczne?

```
5 + 3 // int + int = int
5 + 5.3 // int + double = double
0.13 + 1 // double + int = double
0.1 + 0.1 // double + double = double
1 + 3.14f // int + float = float
0.5f + 0.5f // float + float = float
0.7f + 3 // float + int = float
0.8 + 1.3f // double + float = double
```

Reguła jest w sumie prosta, zawsze zwracany jest typ który pomieści jak najwięcej informacji.
Dlatego

```
5 + 3L // int + long = long ponieważ long ma większy zakres
5 + 3.14 // int + double = double ponieważ double jest w stanie pomieścić wartość po przecinku
0.1 + 0.1f // double + float = double ponieważ double jest w stanie pomieścić więcej informacji
```

I tyle.

A co z dzieleniem?

A to akurat ciekawe i nawet mnie zdziwiło. Jeżeli chcielibysmy wykonać dzielenie `5 / 3` to zgodnie z regułą, typem wynikowym będzie int. Jednak wynikiem działania powinno być ~1.6.
Nawet jeżeli spróbujemy przypisać wynik do zmiennej double to typem wynikowym tego działania wciąż jest int, dlatego daje to 1.

⚠ Ale chwila, dlaczego 1? Nie powinno zaokrąglić do 2?

W programowaniu liczby nie są zaokrąglane, tylko wartość po przecinku jest ucinana. Więc jeżeli wynikiem działania `5 / 3` jest `~1.6` to wartość po przecinku jest ucinana. Tak po prostu, jakbyś ją gumką zmazał.
Jeżeli chcesz zaokrąglić liczbe, zgodnie z prawami matematyki to dodaj `0.5` do wyniku.
Jeżeli wartość po przecinku jest mniejsza od `0.5` to nigdy nie podniesie liczby całkowitej i po ucieciu zostanie to samo

```
1.4 + 0.5 = 1.9 // po ucięciu zostaje 1 więc nie zaokrąglono
```

. Jeżeli wartość po przecinku jest równa lub większa `0.5` to wartość całkowita się zwiększy i po ucięciu uzyskamy zaokrąglenie.

```
1.7 + 0.5 = 2.2 // po cięciu zostaje 2, więc zaokrąglono
```

... Pamiętaj że dodając `0.5` zmieniemamy typ wynikowy ;)