Bartek Kuczyński

Pieniądze w Javie

Język Java jest powszechnie wykorzystywany do tworzenia oprogramowania na potrzeby sektorów: finansowego, bankowego i ubezpieczeniowego. Codziennie powstaje bardzo dużo kodu, który ma za zadanie przeliczać, księgować i wydawać pieniądze. W tym artykule przyjrzymy się możliwościom, jakie daje nam Java do pracy z typami pieniężnymi, oraz pułapkom, jakie na nas czekają.

MATEMATYKA KOMPUTERÓW

Każdy, kto zaczyna pracę z typami reprezentującymi pieniądze (będę też używał pojęcia „typy pieniężne") na pewno spotkał się ze stwierdzeniem, że nie należy używać w tym celu typów takich jak `double` i `float`. Uzasadnienie jest bardzo proste: typy te nie pozwalają na precyzyjne obliczenia. Czasami można spotkać się ze stwierdzeniem, że wystarczy unikać problematycznych operacji takich jak dzielenie i odpowiednio używać zaokrągleń. To powinno wystarczyć, by uniknąć problemów z brakiem precyzji na 8 czy 10 miejscu po przecinku. Jako przykład może zostać tu podane działanie 1,03 - 0.42, którego wynik to 0,61, ale po wypisaniu na ekran to już 0,6100000000000001. Wystarczy zatem obciąć końcówkę zgodnie z zasadami zaokrąglania i będzie w porządku. Niestety sprawa robi się trochę bardziej skomplikowana, gdy dodatkowo w grę wchodzą naprzemienności działań.

Na Listingach 1 i 2 znajdują się dwa programy, które pokazują, na czym polega problem.

Listing 1. Kolejność działań wpływa na wyniki (Java)

```
public class RoundingTraps {

  public static void main(String[] args) {
    double[] numbers = new double[] { 0.1, 0.2, 0.3, 0.4, 0.5,
    0.6, 0.7, 0.8, 0.9 };
    for (double nb : numbers) {
      System.out.format("1 + nb - nb == nb - nb + 1 dla %s  to
      %s\n", nb, 1 + nb - nb == nb - nb + 1);
    }
  }
}
```

Listing 2. Kolejność działań wpływa na wyniki (Erlang)

```
Numbers = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9].
lists:foreach(fun(Nb)-> io:format("~w ~w ~n", [Nb, 1+Nb-Nb==Nb-
Nb+1]) end, Numbers).
```

Jak widać, niezależnie od języka kolejność działań wpływa na otrzymany wynik. Możemy zatem zapomnieć o typach zmiennoprzecinkowych do reprezentowania typów pieniężnych.

Trochę inaczej ma się sprawa z typami całkowitymi `Integer` i Long. Ten pierwszy nie może zostać wykorzystany tylko i wyłącznie ze względu na mały zakres (w Javie maksymalnie dwa miliardy). Jednak typ Long można już wykorzystać w pewnych sytuacjach. Jego maksymalna wartość to 2^{63}-1, co pozwala na bezpieczne zapisanie liczby jednego tryliona (jedynka i osiemnaście zer, w tzw. krótkiej skali używanej m.in. w USA liczba ta nazywa się quintillion, o problemach z nazewnictwem czytaj w ramce).

IMPLEMENTACJA W OPARCIU O TYP LONG

Jedynka i osiemnaście zer daje nam pewną elastyczność, jeżeli chcemy użyć typu Long do implementacji obsługi typu pieniężnego. Operacje takie jak dodawanie, odejmowanie i mnożenie (przez liczbę całkowitą) nie stanowią problemu. Oczywiście jeżeli tylko nie przekroczymy zakresu. Jednak możemy założyć, że taka sytuacja nie nastąpi. Stwórzmy zatem klasę, która będzie prostą reprezentacją typu pieniężnego.

Długa i krótka skala nazewnicza

Bardzo częstym problemem spotykanym w tłumaczeniach z języka angielskiego nazw dużych liczb jest konieczność zmiany skali.

Skala krótka, która jest stosowana m.in. w języku angielskim, arabskim czy rosyjskim, zakłada, że dla liczb powyżej miliona każdy nowy termin opisujący liczbę wprowadza się, gdy liczba jest tysiąc razy większa od poprzedniej.

Skala długa, używana w języku polskim, ale też hiszpańskim, portugalskim i francuskim wprowadza nowy termin, jeżeli liczba jest milion razy większa niż poprzednia. Dla liczb tysiąc razy większych używana jest odpowiednia końcówka albo przedrostek.

Przykładowo dla liczby 10^9 nazwa w krótkiej skali to bilion (nowy termin), a w długiej skali to miliard. W języku hiszpańskim można użyć konstrukcji mil millones. Kolejna tysiąc razy większa liczba, czyli 10^{12}, to trylion w krótkiej skali albo bilion w długiej.

Oczywiście istnieją też inne systemy nazewnicze. Języki takie jak tamilski czy tajski mają całkowicie własny system, a języki takie jak japoński czy chiński wprowadzają specyficzne nazwy dla pewnych liczb.

Listing 3. Prosta wersja klasy Money (Java)

```
public class Money {

  private final Long value;

  public Money(Long value) {
    Preconditions.checkArgument(value >= 0);
    this.value = value;
  }

  public Money add(Money money) {
    Preconditions.checkArgument(money != null, "Money can not be
    null");
    return new Money(this.value + money.value);
  }

  public Money multiply(Long multiplier) {
    Preconditions.checkArgument(multiplier != null, "Multiplier
    can not be null");
    Preconditions.checkArgument(multiplier >= 0);
    return new Money(this.value * multiplier);
  }

  public Money subtract(Money money){
    Preconditions.checkArgument(money != null, "Money can not be
    null");
    return new Money(this.value - money.value);
  }
}
```

Mamy tu dwa problemy. Pierwszym jest brak wsparcia dla wartości groszowych, a drugim brak wsparcia dla dzielenia. Rozwiązanie pierwszego problemu jest stosunkowo proste i stosowane od dawna. Wystarczy odpowiednio sformatować kwotę przy wyświetlaniu. Takie rozwiązanie jest stosowane na przykład w COBOLu, gdzie w trakcie definiowania zmiennej określamy, jak będzie ona wyglądać, ale jej zapis w pliku z danymi jest pozbawiony separatora dziesiętnego. Trochę bardziej kłopotliwe może być określenie, ile miejsc po przecinku chcemy obsługiwać. Najprostszym wydaje się zarezerwowanie dwóch ostatnich cyfr (dziesiątek i jedności), jednak jeżeli chcemy obsługiwać dzielenie, to lepiej zarezerwować więcej. Pozwoli to nam na lepszą obsługę zaokrągleń. O ile więcej? Tu z pomocą przychodzi nam NBP, które w swoich tabelach kursów walut zapisuje wartości z dokładnością do setnej części grosza. Zatem cztery ostatnie cyfry będą reprezentowały część groszową.