## 4.3 Tipi podatkov v Javi

Vsak, prav vsak podatek se v računalniku zapiše v binarni obliki, z "enkami in ničlami". Vendar pa se zapisi različnih vrst podatkov razlikujejo

- v dolžini porabljenega pomnilniškega prostora (številu bitov) in
- v načinu kodiranja.

Za velika števila potrebujemo na razpolago več bitov. Več bitov potrebujemo tudi za zapis števil z veliko mesti za decimalno vejico. Za dolgo besedilo potrebujemo več bitov kot za kratko.

Kodiranje podatkov pomeni zapis podatkov "z enkami in ničlami" *po (dogovorjenem) pravilu.* Cela števila so kodirana po drugačnih pravilih kot realna števila in spet drugače kot besedilo. Številko 5 lahko razumemo kot *število*, lahko kot besedilni *znak* ali kot zelo kratko *besedilo*. Ali ga bomo le izpisali? Ali pa morda korenili? Potrebujemo torej ustrezno kodirane podatke. Pri programiranju je način kodiranja podatkov zajet z izrazom *tip podatkov*[1].

Tip podatkov …

- določa prostor, ki je potreben v pomnilniku za hranjenje takšnega podatka
- določa način kodiranja,
- določa možne vrednosti takšnih podatkov,
- določa operacije, ki jih s takšnimi podatki lahko izvajamo.

Čemu moramo poznati tipe podatkov? Ker jih moramo podatkom v programu vnaprej določiti, napovedati. Če bomo za število avtomobilov v garažni hiši izbrali podatkovni tip za cela števila (v Javi se imenuje *int*), bo napoved izgledala takole:

```
int steviloAvtov = 0;
```

Z oznako *int* pred imenom podatka (spremenljivke) *steviloAvtov* smo določili, da bo ta podatek vedno lahko le celo število iz določenega razpona vrednosti.

V Javi so že pripravljeni osnovni tipi podatkov za:

- cela števila,
- realna števila,
- znaki,
- logične vrednosti.

Sami pa lahko tvorimo tudi bolj kompleksne strukture podatkov, pravzaprav lastne podatkovne tipe.

### 4.3.1 Cela števila

Cela števila so števila brez dela za vejico (oz. piko). To so lahko negativna ali pozitivna števila, pa še ničla spada zraven. Včasih potrebujemo samo relativno majhna cela števila, včasih ne želimo imeti negativnih števil, včasih pa potrebujemo zelo velika števila obeh predznakov. Z velikostjo je povezana tudi velikost zapisa in s tem poraba pomnilniškega prostora. To so razlogi, da imamo v Javi in tudi

[1] tudi: podatkovne tipe, angl. *data type*

drugih programskih jezikih več načinov za zapis celih števil – več celoštevilskih podatkovnih tipov. Tipi podatkov za cela števila v Javi so zbrani v spodnji tabeli.

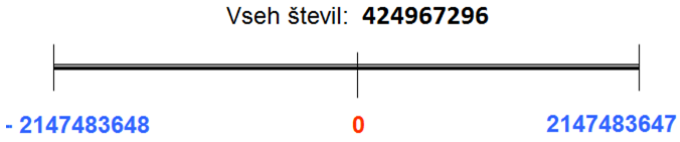
**Tabela 1:** Podatkovni tipi za cela števila

| tip | velikost zapisa | najmanjša vrednost | največja vrednost |
|---|---|---|---|
| `byte` | 1 zlog | -128 | 127 |
| `short` | 2 zloga | -32768 | 32767 |
| `int` | 4 zlogi | -2147483648 | 2147483647 |
| `long` | 8 zlogov | -9223372036854775808 | 9223372036854775807 |
| `char` | 2 zloga | 0 | 65535 |

Poskusimo razumeti razpon celoštevilskih vrednosti za podatkovni tip *int*. Vsak podatek tega tipa dobi v pomnilniku prostor velikosti 32 bitov (4 bajte). Z 32-bitnim zapisom lahko zapišemo N = $2^{32}$ = 4294967296 možnih vrednosti. To so npr. vrednosti od 0 do 4294967295, če želimo samo pozitivna števila in ničlo.


**Slika 20:** Obseg vrednosti in največje število

Če pa želimo zajeti tudi negativna števila, se razpoložljivi obseg vrednosti razpolovi na pozitivna in negativna števila in dobimo razpon vrednosti od – 2147483648 do 2147483647. Ničla spada med 'nenegativna' števila, kar je prav.




**Slika 21:** Razpon vrednosti za podatkovni tip *int*

### 4.3.2 Negativna cela števila in zapis z dvojiškim komplementom

Pri celih številih so pozitivna in negativna števila kodirana različno:

- pozitivna števila so zapisana v *naravnem dvojiškem zapisu,* brez posebnosti*,*
- negativna števila pa so kodirana s t. i. *zapisu z dvojiškim komplementom*[1]*.*

Kako dobimo zapis z dvojiškim komplementom? Pokažimo kar s postopkom pretvorbe, s primerom.

**Primer:** *Negativno število -37 pretvorimo v zapis z dvojiškim komplementom.*

Negativno celo (desetiško) število pretvorimo v (binarni) *zapis z dvojiškim komplementom* po naslednjem **postopku**:

[1] angl. *two's complement, 2-complement*