Rešitev 2. domače naloge (Štetje parov števil)

Ko preberemo število parov in ukaz, v zanki preberemo še posamezne pare in preštejemo, koliko med njimi jih izpolnjuje pogoj, ki ga določa ukaz. Štetje že dodobra obvladamo: pred začetkom zanke inicializiramo števec ustreznih parov (`stUstreznih`) na 0, v zanki pa za vsak par, ki izpolnjuje pogoj, števec povečamo za 1. Pogoj je seveda odvisen od ukaza. Po zaključku zanke izpišemo vrednost števca.

```
import java.util.Scanner;

public class StetjeParovStevil {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int stParov = sc.nextInt();
        int ukaz = sc.nextInt();

        int stUstreznih = 0;

        for (int i = 1;  i <= stParov;  i++) {
            int prvo = sc.nextInt();
            int drugo = sc.nextInt();
            boolean ustreza = false;

            switch (ukaz) {
                case 1:
                    ustreza = prvoManjseOdDrugega(prvo, drugo);
                    break;

                case 2:
                    ustreza = enakaZadnjaStevka(prvo, drugo);
                    break;

                case 3:
                    ustreza = enakaPrvaStevka(prvo, drugo);
                    break;

                case 4:
                    ustreza = permutacija(prvo, drugo);
                    break;
            }

            if (ustreza) {
                stUstreznih++;
            }
        }

        System.out.println(stUstreznih);
    }
    ...
}
```

Lotimo se preverjanja posameznih pogojev. Ker gre za dobro definirane in med seboj neodvisne podprobleme, je pogoje smiselno zapisati kot štiri ločene metode. Vsaka od

teh metod sprejme dve celi števili in vrne `true` natanko v primeru, če števili izpolnjujeta pripadajoči pogoj.

Metoda `prvoManjseOdDrugega` (prvi pogoj) enostavno preveri, ali je prvo število manjše od drugega. Če je, vrne `true`, sicer pa vrne `false`:

```
public class StetjeParovStevil {
    ...
    private static boolean prvoManjseOdDrugega(int a, int b) {
        if (a < b) {
            return true;
        } else {
            return false;
        }
    }
    ...
}
```

Metodo lahko napišemo tudi elegantneje. Rezultat izraza `a < b` je bodisi `true` (če je število `a` manjše od števila `b`) oziroma `false` (v nasprotnem primeru), torej točno tisto, kar moramo vrniti:

```
public class StetjeParovStevil {
    ...
    private static boolean prvoManjseOdDrugega(int a, int b) {
        return (a < b);
    }
    ...
}
```

Števili imata enako zadnjo števko natanko v primeru, če pri deljenju z 10 data enak ostanek:

```
public class StetjeParovStevil {
    ...
    private static boolean enakaZadnjaStevka(int a, int b) {
        return (a % 10 == b % 10);
    }
    ...
}
```

Če nas zanima prva števka, se bomo nekoliko bolj namučili. Število delimo z 10 tako dolgo, dokler ne ostane ena sama števka:

```
public class StetjeParovStevil {
    ...
    private static boolean enakaPrvaStevka(int a, int b) {
        while (a >= 10) {
            a /= 10;
        }
        while (b >= 10) {
            b /= 10;
        }
        return (a == b);
    }
    ...
}
```

Metoda spreminja vrednosti svojih parametrov (`a` in `b`), vendar pa to nima nikakršnih posledic za dejanska parametra `prvo` in `drugo`, s katerima kličemo metodo v metodi `main`. Kot vemo, se formalni parametri ustvarijo kot kopije dejanskih. Metoda tako spreminja zgolj kopiji, izvirnika pa pusti nedotaknjena.

Števili sta permutaciji druga druge natanko tedaj, ko imata enako mnogo enic, enako mnogo dvojk, ... in enako mnogo devetk. (Spomnimo se, da števila v pripadajočih primerih ne vsebujejo ničel.)

```
public class StetjeParovStevil {
    ...
    private static boolean permutacija(int a, int b) {
        for (int stevka = 1;  stevka <= 9;  stevka++) {
            if (stPojavitev(a, stevka) != stPojavitev(b, stevka)) {
                return false;
            }
        }
        return true;
    }

    //
    // Vrne število pojavitev števke stevka v številu stevilo.
    //
    private static int stPojavitev(int stevilo, int stevka) {
        int poj = 0;
        while (stevilo > 0) {
            int st = stevilo % 10;
            if (st == stevka) {
                poj++;
            }
            stevilo /= 10;
        }
        return poj;
    }
}
```

Naloge se lahko lotimo tudi drugače. Na primer tako, da prvo število z zaporednim deljenjem z 10 'lupimo' od desne proti levi, za vsako odstranjeno števko pa poiščemo in odstranimo enako števko v drugem številu. Če z opisanim postopkom obe števili pretvorimo do ničle, sta števili medsebojni permutaciji.

```
public class StetjeParovStevil {
    ...
    private static boolean permutacija(int a, int b) {
        while (a > 0) {
            int stevka = a % 10;
            a /= 10;

            // poiščemo števko stevka v številu b
            int polozaj = polozajStevke(b, stevka);
            if (polozaj < 0) {    // če je ni, števili ne moreta biti permutaciji
                return false;
            }
            // če obstaja, jo odstranimo
            int pot = potenca(10, polozaj);
            b = ((b / pot) / 10) * pot + b % pot;
        }
```

```
        return (b == 0);
    }

    //
    // Vrne položaj števke stevka v številu stevilo, gledano od desne proti levi
    // (zadnje mesto v številu ima indeks 0, predzadnje indeks 1 itd.).
    // Če število ne vsebuje števke, vrne -1.
    //
    private static int polozajStevke(int stevilo, int stevka) {
        int polozaj = 0;
        while (stevilo > 0) {
            if (stevilo % 10 == stevka) {
                return polozaj;
            }
            stevilo /= 10;
            polozaj++;
        }
        return -1;
    }

    //
    // Vrne vrednost potence osnova^eksponent.
    //
    private static int potenca(int osnova, int eksponent) {
        int rezultat = 1;
        for (int i = 1;  i <= eksponent;  i++) {
            rezultat *= osnova;
        }
        return rezultat;
    }
}
```