OOP - Bonusové zadanie

`public float fullMatrix[][]`

- public - obsah matice je možné zvonka ľubovoľne meniť
- float - porušuje zadanie: “matice pre daný číselný typ (celé čísla, čísla s desatinnou čiarkou, vlastný číselný typ - kladné celé čísla)”, keď maticu inicializujem celými číslami nemám zaručené že bude vždy obsahovať len celé čísla, príklad využitia celočíselných matíc: reprezentácia obrazu -intenzita pixelu (hodnota 0-255), maticový zápis Petriho siete

---

`public Matrix(String csvFile){...}`

- o parsovanie matice zo súboru by sa nemal starať konštruktor
- kvôli *separation of concerns* je potrebné vytvoriť zvlášť triedu na prácu so súborom, resp. s konkrétnym formátom súboru (CSV, XML, JSON, …)

`System.out.println("error message"); return null;`

- informácia vhodná len pre používateľa, nie pre programátora používajúceho knižnicu
- čo ak štandardný výstup nie je prístupný pre používateľa? napr. Java webová aplikácia, správu ani neuvidí
- lepšie použiť dostupnú výnimku alebo vytvoriť vlastnú

`try { … } catch (Exception e) { e.printStackTrace(); }`

- výnimka sa len vypíše, program ide ďalej aj keď sa nepodarilo vykonať celý try blok
- pokiaľ nie je možné ošetriť výnimku priamo v danej metóde je lepšie je necatchovať vôbec

---

`class Matrix <T extends Number> {}`
`class IntegerMatrix extends Matrix {}`

- trieda NaturalMatrix nešpecifikuje generický typ T triedy Matrix, lepšie by bolo `IntegerMatrix extends Matrix<Integer>`

`class Matrix<T extends Number> {`
`   private Number[][] arr;`

- pokiaľ nie je generický typ T využitý ako dátový typ premennej arr je zbytočný a vedie k chybným riešeniam ako napr.:

```
if(a instanceof Double || b instanceof Double) {
} else if(a instanceof Float || b instanceof Float) {
} else if(a instanceof Long || b instanceof Long) {
} else {
}
```

- takéto riešenie je neflexibilné, ak chcem využiť túto knižnicu a pridať si vlastný dátový typ tak musím prepisovať interný kód

---

```
class Multiplier {
     List<List<Integer>> matrixOfInteger(...) { ... }
     List<List<Float>> matrixOfFloat(...) { ... }
     List<List<Double>> matrixOfDouble(...) { ... }
}
```

- buď ponechať násobenie v oddedených triedach matíc alebo vytvoriť podobným štýlom samostatne Multiplier pre Integer, Float a Double

---

```
void Multiply(), T plus_operation()
```

- názov metódy má začínať malým písmenom -> `multiply()` (https://www.oracle.com/technetwork/java/codeconventions-135099.html)
- nepoužívať "_" v názvoch (okrem konštánt) -> `plusOperation()`

```
int r; // pocet riadkov
```

- nemá zmysel používať skratky a potom ich komentovať, radšej použiť plný názov

---

```
interface MatrixFunctionality {
    Matrix matrixAddition(Matrix matrix1);
    Matrix matrixMultiplication(Matrix matrix1);
    Matrix matrixSubtraction(Matrix matrix1);
    Matrix matrixImport(String path);
    void matrixExport(String path);
}
```

- ak už má interface v názve Matrix nemusí každá metóda začínať matrix, zbytočne to zhoršuje čitateľnosť kódu

---