SOFTWAROVÝ PROJEKT

CAESAROVA ŠIFRA

SASW (Specifikace architektury SW)

Jana Michnová

(michnjan@fel.cvut.cz)

Strategie a zpracování

V první řadě je nutné si zaopatřit prostředí Netbeans IDE 6.1 a vyšší, ve kterém budu svůj projekt vytvářet. Po té přecházím k návrhu toho, co má Caesarova šifra dělat a co vše chci do programu zavést.

Jelikož se jedná o lehkou úlohu, kde jsem jako testovací řetězec použila řetězce „abc“ a „def“, zavedla jsem i práci se soubory.

Máme tedy řetězec „abc“, který chceme zašifrovat do standardní velikost kroku 3 do řetězce „def“. Potřebujeme, aby se, nám daný string rozdělil na jednotlivé znaky, které následně uložíme do pole, a to potom pomocí cyklu převedeme na pole ASCII hodnot daných znaků. Toto pole je pak nutno využít v dalším cyklu, který každému prvku **přičte** velikost kroku zadanou uživatelem. Když máme pole ASCII hodnot navýšených o velikost kroku, musíme přistoupit opět k cyklu, který nám jej převede zpět na samotné znaky.

V praxi se později ukázalo, že proces „rozsekání“ stringu na znaky je snadněji proveditelné pomocí funkce **getChars** a cyklus přičítání a převádění číselných hodnot na znaky, stačí pouze jeden, během kterého se provedou obě tyto akce.

```
text.getChars(0, pomoc, znaky, 0);
for (int i = 0; i < pomoc; i++) {
    ascii = (int) znaky[i] + sifra;
    sifrovaneZnaky[i] = (char) ascii;
}
```

Obr. 1 Funkce getChars a cyklus zašifrování

Nastal však další problém s tím, že uživatel nám ve své nevědomosti může zadat velikost kroku jinou než číselnou nebo žádnou. Tuto mezeru jsem později řešila pomocí výjimek, které nám zachytí každý takový nepěkný pokus o spadnutí programu.

Nyní máme problém šifrace za sebou a nastal problém nový a to problém dešifrace. Díky předešlým zkušenostem bylo snadnější dát dohromady strategii výroby kódu.

V podstatě totiž jde o obrácený proces. Jediné co chceme je, ze zašifrovaného výrazu „abc“, který je nyní v podobě „def“, dostat opět původní řetězec „abc“.

Máme tedy testovací řetězec „def“, který potřebujeme rozdělit na jednotlivé znaky. Ty pak převést na jejich ASCII hodnotu a odečíst velikost kroku zadanou uživatelem. Po té znovu převést všechny ASCII hodnoty na původní znaky znakové sady a je vyhráno.

Všimněme si, že na rozdíl od šifrace se v dešifraci **odečítá**. Tedy jde opravdu o obrácený proces k šifrování. Nemělo by smysl přičítat, jelikož bychom se nikdy nedostali k požadovanému výsledku a jen bychom neustále šifrovali a šifrovali.

K dešifraci opět potřebujeme proces rozdělení stringu do pole znaků a dále cyklus převádění daných znaků na jejich ASCII hodnoty. Po převodu na ASCII hodnoty nezbývá nic jiného než odečíst velikost kroku a konvertovat je zpět na znaky.

V praxi se samo sebou ukázalo mnohem rozumnější použít funkci getChars na rozdělení řetězce do pole znaků a využití jednoho cyklu pro odečítání a konverze.

Ani teď se nevyhneme případným výjimkám, tudíž se znovu potýkáme s problémem filtrace těchto stavů. Se všemi těmito stavy nám pomáhá blok **Try{…}Catch(…){…}** jako řádný „odchytávač“ výjimek.

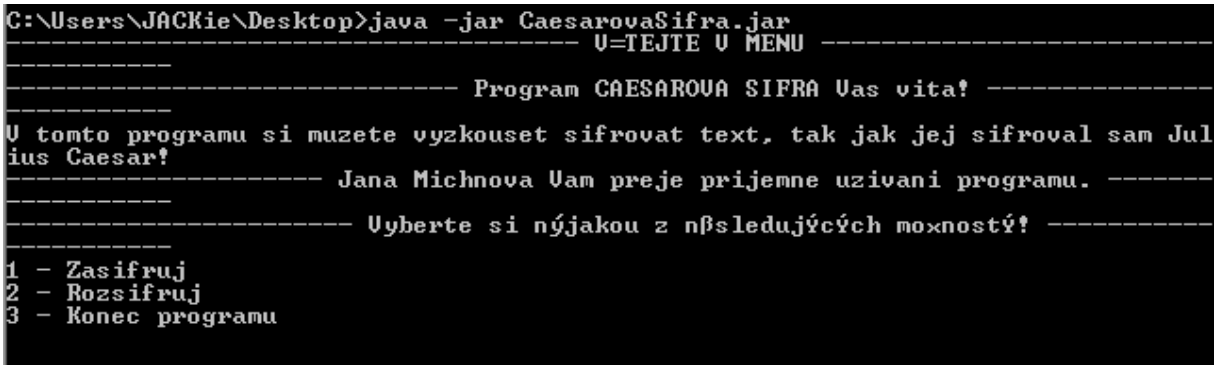
```
try {
    FileWriter fw = new FileWriter(cesta, true);
    BufferedWriter bw = new BufferedWriter(fw);
    bw.write(sifrovaneZnaky);
    bw.newLine();
    bw.close();
    fw.close();
} catch (FileNotFoundException e) {
    System.out.println("Soubor nebyl nalezen!");
} catch (IOException e1) {
    System.out.println("Chyba pri zapisu!!!");
}
```

Obr. 2 Blok Try{…}Catch(…){…}

V okamžiku, kdy je základní kostra a princip programu zachycen a sepsán tak jak má, zabředneme do práce se soubory. Jedná se

v podstatě o jednoduchou práci, ale přeci jenom nás může dosti pozlobit stavy, ve kterých nám může dojít k pádu aplikace. Tyto stavy jsou opět naše oblíbené výjimky, které se neřešily v tomto případě nijak jinak, než zase blokem Try{…}Catch(…){…}.

Po té, co máme zakomponovánu i **práci se soubory** v našem programu Caesarova šifra, se stáváme vítězi v této „bitvě“. Splnili jsme požadavky, které jsme si předem stanovili a už jen stačí si vytvořit z našeho projektu soubor *.jar. Nyní můžeme s programem pracovat v prostředí konzole Windows nebo terminálu v Linuxových systémech za použití příkazu „java –jar CaesarovaSifra.jar“ v lokaci, kde se jar nachází.



Obr. 3 Spuštění programu přes konzoli Windosws XP

Nutností je ovšem mít nainstalovaný balíček Java JVM, který je k dispozici na webových stránkách Sun.