UNICORN COLLEGE

Zaměstnanec (Moudrá Jana)
Vývoj v Java SE (verze 2010) Zima 2016 ▸ 04 Domácí úkoly / práce

UCL Exercise Assignment
UCL-BT:JSE10A16W.CZ/XA01_PT

JSE Domácí úkol 1 - Základy syntaxe (kombinovaná forma)

Kontext

Tento úkol slouží k procvičení si znalostí Javy a objektově orientovaného programování.

Zadání

Napište program, jenž bude provádět simulaci firem, které najímají programátory a pracují na softwarových projektech (přesněji dopište jeho chybějící části).

Vstupní data

Vstupní data programu se nacházejí přímo ve zdrojovém kódu, protože zatím neumíme pracovat např. se soubory či databází.

K firmám máme ve vstupních datech následující informace:

- `name` je název firmy.
- `capacity` udává, kolik může firma najmout programátorů.
- `dailyExpenses` jsou denní fixní náklady.
- `budget` je výchozí rozpočet firmy.

K programátorům máme ve vstupních datech následující informace:

- `name` je jméno programátora.
- `speed` udává, kolik člověkodní programátor odpracuje za den.
- `dailyWage` je denní mzda.

K projektům máme ve vstupních datech následující informace:

- `name` je název projektu.
- `manDays` je počet člověkodní kolik se musí na projektu odpracovat.
- `price` je částka, kterou firma obdrží při dokončení projektu.

Program

Úkolem firem je dokončit všech 10 projektů. K tomu, aby mohly na projektech pracovat, si mohou najmout programátory až do výše vlastní kapacity. Každý projekt má svoji náročnost udávanou v normovaných člověkodnech a také cenu, kterou firma obdrží po dokončení projektu. Programátoři pracují na projektech, s tím, že každý má trochu jinou výkonnost. Výkonnost (`speed`) je udávána v počtu normovaných člověkodnů, které je programátor schopen za den odpracovat. Každý programátor může pracovat maximálně na 1 projektu. Na 1 projektu může pracovat více programátorů.

Simulace firem probíhá postupně, každá firma postupně pracuje se stejnými projekty a stejnými programátory (liší se počet programátorů, které můžou najmout). Simulace běhu firmy může skončit v zásadě třemi způsoby.

1. Dodělá všechny projekty a stav se nastaví na *Finished*.
2. Rozpočet se dostane do mínusu a stav se nastaví na *Bankrupt*.
3. Když počet dnů běhu překročí 1000, simulace automaticky skončí a stav firmy se nastaví zpět na *Idle*. V našem testovacím běhu se žádná firma nedostala přes 200 dnů, takže toto slouží spíše jako bezpečnostní pojistka proti nekonečným cyklům.

Kostra programu v příloze tohoto artefaktu obsahuje již částečně hotovou implementaci, kterou máte za úkol doplnit. Místa k doplnění jsou jasně vyznačena pomocí komentářů, ve kterých se vyskytuje: "**IMPLEMENTUJTE ZDE**" nebo "**IMPLEMENTUJTE TUDO METODU**". Zároveň se v těchto komentářích nachází instrukce, jak má např. daná metoda fungovat.

Důležité upozornění: jako úspěšné řešení úkolu bude považováno pouze to, které vyjde z dodaného kódu a měnit bude pouze ta místa, která jsou pro to jasně označena. Jinými slovy: neměli byste mít důvod měnit jiná místa kódu, a tak jej ani nehledejte.

Pokud byste i přesto nabyli dojmu, že zadání nemůžete splnit, aniž byste zasáhli do okolního kódu, kontaktujte svého vyučujícího. Bude-li se skutečně jednat o chybu zadání, co nejrychleji je opravíme nebo doplníme.

Samozřejmě během práce na úkolu můžete chtít zasahovat i do jiných částí programu, abyste se v něm třeba lépe zorientovali nebo abyste odladili problematickou pasáž. To je pochopitelně v pořádku. Na závěr si ale ověřte, že jste tyto části vrátili do původního stavu.

List

Program používá datové struktury `List`, se kterou jste se již setkali. `List` patří mezi kolekce a jeho výhodou je, proti klasickému poli, že není omezen svou fixní délkou.

Prostudujte si zdrojový kód zadání a podívejte, jak se `List` (potažmo konkrétní implementace `ArrayList`) používá.

Vám se při implementaci tohoto domácího úkolu může hodit API dokumentace List a těchto pár věcí:

Vytvoření nového Listu

Prázdný `List` s konkrétní implementací `ArrayList` vytvoříme takto:

```
1 List<Project> someProjects = new ArrayList<Project>();
```

Přidání nové položky

Pro přidání jedné, čí více položek (celé kolekce najednou) do listu můžete využít metody `add()` a `addAll()`.

Smazání položky

Pro smazání jedné, či více položek (celé kolekce najednou) z listu můžete využít metody `remove()` a `removeAll()`.

Získání položky

Pokud potřebujete získat položku, můžete k tomu použít metodu `get()`.

Procházení skrz položky

Pro iteraci skrz seznam můžete použít iterátor, klasický for cyklus či for-each cyklus. Prohlédněte si zbytek aplikace, kde je vidět, jak se for-each s Listem používá.

Modifikace Listu v cyklu

Dejte si pozor na modifikaci seznamu při jeho iteraci. Pokud například při iteraci začneme mazat nějaké položky, vyhodí nám to `ConcurrentModificationException`. Řešením je pro iteraci buď používat iterátor (a mazat přes něj) a nebo si položky ke smazání (dočasně) uložit do jiného seznamu a tento celý seznam poté smazat z naší kolekce pomocí `removeAll()`.

Další metody

Další metody, které se vám mohou s Listy hodit:

- `size()` vrací aktuální velikost seznamu
- `isEmpty()` vrací, zda je seznam prázdný
- A další metody v API

Řešení

Odeslat řešení domácího úkolu