UNICORN COLLEGE

PES Domácí úkol 1 - Softwarové projekty (kombinovaná forma)

Zadat úkol skupině č. 1 | Zadat úkol skupině č. 2

Kontext

Tento úkol slouží k procvičení práce se soubory, třídami a objekty.

Zadání

Napište program, jenž bude provádět simulaci firem, které najímají programátory a pracují na softwarových projektech (přesněji dopište jeho chybějící část). Vstupní data programu se nacházejí ve třech CSV souborech, aby si je mohli uživatelé programu v budoucnu změnit i v tabulkovém procesoru (např. LibreOffice Calc). Než se pustíme do samotného zadání programu, tak si popíšeme strukturu jednotlivých souborů.

Seznam firem

Strukturu nejlépe ukáže konkrétní příklad tabulky:

Name	Capacity	DailyExpenses	Budget
Alpha	2	1000	90000
Beta	2	1500	100000
Gamma	3	3000	200000
Delta	5	6000	400000
Epsilon	6	8000	900000
Theta	7	10000	1200000
Omega	7	20000	20000000

Ukažme si rovněž, jak takováto tabulka může vypadat přímo ve formátu CSV:

```
Name,Capacity,DailyExpenses,Budget Alpha,2,1000,90000 Beta,2,1500,100000 Gamma,3,3000,200000
Delta,5,6000,400000 Epsilon,6,8000,900000 Theta,7,10000,1200000 Omega,7,20000,20000000
```

Jednotlivé sloupce mají následující význam:

- **Name** je název firmy
- **Capacity** udává kolik může firma najmout programátorů
- **DailyExpenses** jsou denní fixní náklady
- **Budget** je výchozí rozpočet firmy

Seznam programátorů

Strukturu nejlépe ukáže konkrétní příklad tabulky:

Name	Speed	DailyWage
Martin	1.9	2000
Jarda	1.0	1300
Lukas	0.6	900
Pepa	1.7	2200
Kamil	0.4	1800
Honza	1.3	1500
Filip	1.1	1000

Ukažme si rovněž, jak takováto tabulka může vypadat přímo ve formátu CSV:

```
Name,Speed,DailyWage Martin,1.9,2000 Jarda,1.0,1300 Lukas,0.6,900 Pepa,1.7,2200 Kamil,0.4,1800 Honza,1.3,1500 Filip,1.1,1000
```

Jednotlivé sloupce mají následující význam:

- **Name** je jméno programátora.
- **Speed** udává kolik člověkodní programátor odpracuje za den.
- **DailyWage** je denní mzda.

Seznam projektů

Strukturu nejlépe ukáže konkrétní příklad tabulky:

Name	ManDays	Price
Web	5.0	20000
Portal	15.0	60000
Email system	25.0	90000
Eshop	40.0	150000
CMS	60.0	250000
Forum	30.0	35000
B2B SYS	120.0	800000
Multimedia Web	7.0	50000
TODO List	3.0	10000
CRM	20.0	80000

Ukažme si rovněž, jak takováto tabulka může vypadat přímo ve formátu CSV:

```
Name,ManDays,Price Web,5.0,20000 Portal,15.0,60000 Email system,25.0,90000 Eshop,40.0,150000 CMS,60.0,250000 Forum,30.0,35000 B2B SYS,120.0,800000 Multimedia Web,7.0,50000 TODO List,3.0,10000 CRM,20.0,80000
```

Jednotlivé sloupce mají následující význam:

- **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 minusu 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 příloze tohoto artefaktu obsahuje již hotové třídy Programmer a Project a téměř hotovou třídu Company. Program umí načíst data z polí do objektů, spustit simulaci pro každou firmu a vypsat výsledky. Tento kód se stará o to, aby proběhla simulace pro všechny firmy.

```
# Tento cyklus se opakuje pro kazdou firmu
companies.each do |c|
  # Prijmout programatory
  c.allocate_programmers(programmers)
  # Nacist projekty
  c.allocate_projects(projects)
  # Spustit simulaci
  c.run
  # Vypsat vysledek
  c.output_result

  # Protoze pri simulaci pracujeme v kazde firme se stejnymi objekty programatoru a projektu
  # pak po kazde dokoncene simulaci pro jednu firmu resetujeme programatory a projekty do
  # vychoziho stavu, aby jim nezustal prirazen projekt a nebyla u nich evidovana odvedena prace.
  programmers.each do |prg|
    prg.clear_project
  end
  projects.each do |prj|
    prj.reset
  end
end
```

Vaším úkolem bude implementovat načtení dat ze souborů a implementovat samotné metody potřebné pro simulaci, které přiřazují programátory na projekty, kontrolují, jestli jsou projekty hotové, a starají se o přesun peněz mezi firmou, programátory a projekty.

```
FILE_NAME_COMPANIES= "companies.csv"
FILE_NAME_PROGRAMMERS = "programmers.csv"
FILE_NAME_PROJECTS = "projects.csv"
companies = []
programmers = []
projects = []
# IMPLEMENTUJTE zde nacteni dat ze vstupnich souboru do companies, programmers a projects
```

Na výše uvedené místo implementujte načítání ze souborů tak, že do polí v proměnných **companies**, **programmers** a **projects** povkládáte instance tříd Company, Programmer a Project.

Metoda **check_projects** je na ukázku implementovaná, zbytek už je na vás. Konkrétně se jedná o tyto metody: **allocate_programmers**, **check_programmers**, **assign_new_projects**, **programmers_work**, **check_company_state**. Popis jejich funkcionality je v komentářích přímo v kódu.

```
# Najmout tolik programatoru, kolik cini kapacita
def allocate_programmers(programmers_array)
  # IMPLEMENTUJTE TUTO METODU
```

```
  # Z pole programmers_array vyberte prvnich @capacity programatoru
  # v poradi podle nejvyhodnejsiho pomeru jejich rychlosti proti jejich cene.
  # Nasledne v tomto poradi vytvarejte prislusne instance tridy Programmer
  # a vytvorene objekty vkladejte do pole @programmers.
end
```

```
# Uvolnit programatory, co delaji na projektech,
# ktere uz jsou hotove.
def check_programmers
  # IMPLEMENTUJTE TUTO METODU
end
```

```
# Nastavit projekty programatorum, kteri jsou volni.
def assign_new_projects
  # IMPLEMENTUJTE TUTO METODU
  # Pro kazdeho volneho programatora hledejte projekt k prideleni nasledovne:
  # - Pokud existuje nejaky projekt v @projects_waiting, vyberte prvni takovy.
  #   (Nezapomente mu zmenit stav a presunout jej do @projects_current.)
  # - Pokud ne, vyberte takovy projekt z @projects_current, na kterem zbyva
  #   nejvice nedodelane prace.
end
```

```
# Programatori pracuji.
def programmers_work
  # IMPLEMENTUJTE TUTO METODU
  # Projdete vsechny programatory a predejte jejich denni vykon projektum,
  # ktere maji pridelene.
  # Zaroven snizte aktualni stav financi firmy o jejich denni mzdu a rovnez
  # o denni vydaje firmy.
end
```

```
# Zjistit stav spolecnosti.
def check_company_state
  # IMPLEMENTUJTE TUTO METODU
  # Pokud je aktualni stav financi firmy zaporny, nastavte
  # stav spolecnosti na :bankrupt.
  # Pokud ne a zaroven pokud jsou jiz vsechny projekty hotovy,
  # nastavte stav spolecnosti na :finished.
end
```

Tyto metody následně volá metoda run, která se stará o samotný běh simulace:

```
# Spusteni simulace. Cyklus se ukonci kdyz je stav firmy
# :bankrupt nebo :finished, nebo pokud simulace bezi vice nez 1000 dni
def run
  @state = :running
  while @state != :bankrupt and @state != :finished and @days <= 1000
    @days += 1
    check_projects
    check_programmers
    assign_new_projects
    programmers_work
    check_company_state
  end
  @state = :idle if @state == :running
end
```

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

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.

Pro usnadnění ladění programu jsme třídám Programmer, Project a Company přidali metodu print_debug_info. Tyto metody vypíší podrobné informace o stavu daného programátora, projektu nebo společnosti. Během vývoje svého řešení jich můžete využít, abyste se lépe zorientovali v tom, co se v kterém kroku v programu děje. Ještě přehledněji můžete pochopitelně stav simulace sledovat využitím debuggeru.

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.

Po provedení simulace program pro každou firmu vypíše informace o jejím koncovém stavu na konzoli, o čemž se přesvědčite po dokončení implementace.

Po vypsání informací o koncovém stavu na konzoli program uživateli umožní si nechat tyto informace zapsat do CSV souboru. **Implementujte tuto možnost** podle instrukcí ve zdrojovém kódu. Pokud soubor result.csv již existuje, pak je jeho obsan přepsán.

```
# IMPLEMENTUJTE zde:
# Program se uzivatele zepta zdali chce informace o koncovem stavu firem zapsat do souboru.
# Pokud uzivatel zvoli ze nechce, pak program konci. V opacnem pripade je koncovy stav firem
# zapsan do souboru result.csv. Tento soubor bude mit hlavickovy radek:
# CompanyName,DaysRunning,FinalBudget,FinalState,NumberOfProjectsDone

# Jeden radek v tomto souboru bude odpovidat informacim o koncovem stavu jedne firmy
```

Ošetřování chyb

V tomto úkolu nemusíte ošetřovat chybové stavy pomocí výjimek. Předpokládejte tedy korektnost vstupních dat, přítomnost souborů s danými názvy na očekávaném místě atd.

Řešení

Odeslat řešení domácího úkolu