Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de Computación
IIC2143 – Ingeniería de Software (II/2022)

Tarea Ruby

ENTREGA: 12 DE SEPTIEMBRE, 23:59.

1. Indicaciones generales

Las personas que realicen esta tarea obtendrán hasta 5 décimas extra en su primera evaluación escrita. Si bien la tarea es de carácter opcional, hacerla está muy recomendado, ya que es una excelente oportunidad para aprender Ruby, lenguaje sobre el cual se construye el framework Ruby on Rails utilizado durante todo el curso. IMPORTANTE: El carácter de esta tarea es 100 % INDIVIDUAL.

2. Introducción

En esta tarea se les pedirá modelar el funcionamiento de distintos supermercados, simulando la llegada y salida de clientes y las compras que hagan, manejando el stock de productos y el aforo máximo permitido dentro de cada local.

3. Modelación

Deberás modelar las siguientes clases:

`Supermarket, Product, Client`

Si bien la modelación presentada a continuación es sugerida, pero no obligatoria, **sí se debe tener todas las clases escritas en esta sección.**

`Clase Supermarket`:

Atributos:

- id: int
- name: str
- capacity: int
- earnings: int
- line: array
- products: array || hash

`Clase Product`:

Atributos:

- id: int
- name: str
- price: int

`Clase Client`:

Atributos:

- id: int
- name: str
- money: int
- cart: array

4. Eventos

Además de implementar cada una de las clases descritas anteriormente, deberán manejar distintos eventos que servirán para simular el funcionamiento de cada supermercado:

- `ARRIVAL {supermarket_id} {client_id}`: indica la llegada de el cliente con `id client_id` al supermercado con `id supermarket_id`. Si el supermercado se encuentra lleno en ese momento, deberá esperar en una fila hasta que se desocupe un cupo.
- `PICK {client_id} {product_id} {quantity}`: indica que el cliente con `id client_id` sacó `quantity` del producto con `id product_id` en el supermercado en el que se encontraba en ese momento. Si es que no quedaba *stock* suficiente, debe sacar lo que pueda. En el caso de que el cliente se encuentre en la fila, se debe ignorar este evento.
- `RESTOCK {supermarket_id} {product_id} {quantity}`: indica que se repuso `quantity` del producto con `id product_id` en el supermercado con `id supermarket_id`.
- `CHECKOUT {client_id}`: indica que el cliente con `id client_id` fue a pagar los productos que había recogido. En el caso de no tener dinero suficiente, debe empezar a devolver productos, partiendo por los últimos que recogió, hasta poder pagar. Una vez realizada la compra, el cliente automáticamente sale del supermercado, abriendo un cupo si es que hay gente en la fila.

5. Archivos CSV + TXT

Para cargar los datos se subirán al repositorio oficial del curso los siguientes archivos en formato csv: supermarkets.csv, products.csv, clients.csv, stock.csv. Adicionalmente, se te entregará un archivo adicional en formato txt que contendrá los eventos que tendrá que procesar el programa separados por líneas. Este archivo se entregará en la línea de comandos como un ARGV (más sobre este archivo en la sección 8. Ejecución). Los archivos tienen la siguiente estructura:

`supermarkets.csv`

Este archivo contiene la información de los supermercados que manejará la simulación. Cada fila contiene el id del supermercado, el nombre y el aforo, separados por una coma (","). Cabe destacar que el id es único y no necesariamente será igual a la posición en la que aparezca en el archivo.

```
id,name,capacity
1,Lider,10
2,Jumbo,25
3,Santa Isabel,5
```

`products.csv`

Este archivo contiene la información de los productos que manejará la simulación. Cada fila contiene el id del producto, el nombre y su precio, separados por una coma (","). Cabe destacar que el id es único y no necesariamente será igual a la posición en la que aparezca en el archivo.

```
id,name,price
1,apple,160
2,cereal,1500
3,matress,200000
4,coffee beans,8000
```

`clients.csv`

Este archivo contiene la información de los clientes que manejará la simulación. Cada fila contiene el id del cliente, el nombre y la cantidad de dinero que tiene, separados por una coma (","). Cabe destacar que el id es único y no necesariamente será igual a la posición en la que aparezca en el archivo.

```
id,name,money
1,Martin Orrego,35000
2,Moises Retamal,4000
```

`stock.csv`

Este archivo contiene el stock inicial que cada supermercado tiene de cada producto. Cada fila contiene el id del supermercado, el id del producto, y el stock de este último, separados por una coma (",").

```
supermarket,product,quantity
1,1,50
1,2,10
1,3,100
2,1,30
```

`events.txt`

Este archivo contiene el listado de eventos que tendrán que manejar, en el orden que aparecen.

```
ARRIVAL 1 1
ARRIVAL 1 2
PICK 1 2 10
CHECKOUT 1
```

6. Output

Para finalizar, una vez que hayan procesado todos los eventos, deberán generar un archivo de texto con la cantidad de dinero final de cada supermercado y cliente, y el stock global de cada producto, cada uno en orden creciente según su id. Cada recurso debe salir de la siguiente forma `{RECURSO} {NOMBRE} {DINERO}`, en el caso de los supermercados y clientes, y `PRODUCT {NOMBRE} {CANTIDAD}`, en el caso de los productos.

`output.txt`

```
SUPERMARKET Lider 53200
SUPERMARKET Jumbo 60000
CLIENT Moises Retamal 500
CLIENT Martin Orrego 10500
CLIENT Felipe Fuentes 10
PRODUCT apple 100
PRODUCT iphone 5
PRODUCT laptop 10
```

7. Aclaraciones y Consideraciones importantes

- El id de un recurso siempre será un número entero.
- Cada producto siempre tiene el mismo precio, sin importar en qué supermercado se venda.
- El supermercado no pierde dinero al recibir nuevo *stock*.
- El supermercado puede recibir *stock* de un producto que no tenía originalmente.
- Un cliente puede agregar productos a su carro, a pesar de que no le alcance el dinero para comprarlos. Una vez de que llegue a la caja, estos se devuelven y se habilitan para que otro cliente pueda comprarlos.
- Un cliente solo puede estar en un supermercado a la vez.
- Un cliente puede ir a más de un supermercado durante la simulación.
- Si un cliente se encontraba en la fila, y hace *checkout*, sale de esta sin haber nunca entrado al supermercado.
- Si la simulación ya terminó, y un cliente no ha termindo sus compras, los productos que tenía en el carro no se deben considerar en el stock del output final.

8. Ejecución

Para corregir la tarea, los ayudantes ejecutarán el siguiente comando en la terminal:

`ruby main.rb {datasets_folder} {events_file} {output_file}`

Los archivos .csv con los datos siempre se llamarán según lo especificado en la sección 6. Archivos CSV + TXT, pero irán dentro de la carpeta {datasets_folder}. El archivo instrucciones tendrá el nombre especificado en {instructions_file}, y el archivo al cual escribir el output tendrá el nombre especificado en {output_file}.

De manera visual, si nuestro repositorio tuviera la siguiente forma:

repo/
—| data/
———| supermarkets.csv
———| products.csv
———| clients.csv
———| stock.csv
—| events.txt
—| main.rb

El comando a correr sería:

```
ruby main.rb data events.txt output.txt
```

generando un archivo output.txt con los outputs esperados.

IMPORTANTE: La corrección se realizará de manera automatizada, por lo que se espera que el formato que utilicen para el archivo de output sea exactamente el que se pide en el enunciado. Para facilitar su desarrollo, se les entregará un archivo de prueba que podrán utilizar para verificar que, efectivamente, están cumpliendo con esto.

9. Cálculo del bonus

Como se dijo anteriormente, haciendo esta tarea podrán optar por hasta 5 décimas en su primer interrogación. Estas se calcularan de la siguiente manera:

$$\text{Décimas bonus} = \lfloor \frac{5 \cdot \text{P}\,\%}{100} \rfloor$$

siendo P el porcentaje de correctitud de su archivo de output, **truncado** hasta el segundo decimal.

10. Foro

Cualquier duda que surja respecto a la tarea pueden preguntarla en el foro oficial del curso, **escribiendo en el título de la issue "[Tarea] - Contenido pregunta"**.

11. Entrega

La entrega de la tarea se realiza mediante un assignment en Canvas, donde se deberá subir un archivo .zip **sin contener archivos .csv ni .txt**, dentro del cual deben incluir el o los archivos que utilicen en su programa, **incluyendo un archivo main.rb que servirá para ejecutar su tarea**. El plazo es hasta el jueves 12 de septiembre a las 23:59.

12. Política de integridad académica

Los alumnos de la Escuela de Ingeniería de la Pontificia Universidad Católica de Chile deben mantener un comportamiento acorde a la Declaración de Principios de la Universidad. En particular, se espera que mantengan altos estándares de honestidad académica. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un Procedimiento Sumario. Es responsabilidad de cada alumno

conocer y respetar el documento sobre Integridad Academica publicado por la Dirección de Docencia de la Escuela de Ingeniería en el SIDING.

Específicamente, para los cursos del Departamento de Ciencia de la Computación, rige obligatoriamente la siguiente política de integridad académica. Todo trabajo presentado por un alumno para los efectos de la evaluación de un curso debe ser hecho individualmente por el alumno, sin apoyo en material de terceros. Por "trabajo" se entiende en general las interrogaciones escritas, las tareas de programación u otras, los trabajos de laboratorio, los proyectos, el examen, entre otros. Si un alumno copia un trabajo, obtendrá nota final 1.1 en el curso y se solicitará a la Dirección de Pregrado de la Escuela de Ingeniería que no le permita retirar el curso de la carga académica semestral. Por "copia" se entiende incluir en el trabajo presentado como propio partes hechas por otra persona. Obviamente, está permitido usar material disponible públicamente, por ejemplo, libros o contenidos tomados de Internet, siempre y cuando se incluya la referencia correspondiente. Lo anterior se entiende como complemento al Reglamento del Alumno de la Pontificia Universidad Católica de Chile. Por ello, es posible pedir a la Universidad la aplicación de sanciones adicionales especificadas en dicho reglamento.