Departamento de Computación
FCEFQyN, Universidad Nacional de Río Cuarto
Asignatura: Estructuras de Datos y Algoritmos - Algoritmos y Estructuras de Datos II
Primer Cuatrimestre de 2025

**Práctica No. 3**

Para los siguientes ejercicios proveer test para todos los métodos que agregue para chequear que satisfacen las especificaciones.
Acceder al código de base de los ejercicios de esta práctica aceptando el siguiente assignment: `https://classroom.github.com/a/sPhZkyuS`.

1. Utilizando *Maps*, defina una clase *PhoneBook*, que se encargue de crear una agenda de contactos telefónicos y operaciones asociadas, tales como:

   - *addEntry* agrega un contacto conteniendo el nombre del contacto y el número de teléfono asociado.
   - *changePhone* dado un contacto, se encarga de modificar el número de teléfono.
   - *delContact* borra un contacto de la agenda.
   - *getAllContacts* Retorna un *Set* con todos los contactos almacenados en la agenda.
   - *lookupNumber* dado un contacto, busca y retorna el número de telefono asociado.

   1.1 ¿Cómo podrías verificar cuántas entradas tiene la agenda telefónica?.
   1.2 Crea una clase de test para la clase *PhoneBook* y analiza ¿qué sucede si agregas una entrada a un *Map* con una clave que ya existe?.
   1.3 ¿Qué sucede cuando agregas dos entradas a un *Map* con el mismo valor y dos claves diferentes?.
   1.4 ¿Cómo verificas si una clave dada está contenida en un *Map*?.
   1.5 ¿Qué sucede cuando intentas buscar un valor y la clave no existe en el *Map*?
   1.6 ¿Cómo se pueden mostrar todas las claves actualmente almacenadas en la agenda?

2. Ejercicios para resolver en la versión *tech-support-v3-teoria*

   2.1 Agrega más asignaciones de palabras/respuestas a tu aplicación. Puedes copiar algunas de las soluciones proporcionadas y agregar algunas por tu cuenta.
   2.2 Asegúrate de que la misma respuesta predeterminada no se repita dos veces seguidas.
   2.3 A veces, dos palabras (o variaciones de una palabra) se asignan a la misma respuesta. Soluciona esto mapeando sinónimos o expresiones relacionadas a la misma cadena, para que no necesites múltiples entradas en el *Map* de respuestas para la misma respuesta.
   2.4 Cuando no se reconozca ninguna palabra, utiliza otras palabras de la entrada del usuario para seleccionar una respuesta predeterminada adecuada: por ejemplo, palabras como *"why" "how"* y *"who"*.
   2.5 ¿Qué hace el método *putIfAbsent* de *HashMap*?
   2.6 Agrega una clase *WordCounter* al proyecto *tech-support-teoria-v3* y úsala para obtener la cantidad de apariciones de cada palabra después de que se haya mostrado el mensaje de *"goodbye"*.
   2.7 Imprime sólo la cantidad de las palabras que no sean ya claves en *responseMap* de la clase *Responder*. Necesitarás proporcionar una forma de acceder a las claves en *responseMap*.

3. Refactorice el código del proyecto *auction* de la práctica 2 para reemplazar la estructura de datos usada (listas) por *Maps*. Tenga en cuenta que no debería ser necesario hacer ningún cambio en las interfaces de las clases (especificaciones y perfiles de los constructores y de sus métodos públicos) para poder modificar la implementación. Verifique que la nueva implementación preserva el comportamiento de la versión anterior ejecutando los tests sobre la nueva versión (todos deberían pasar sin modificaciones).

4. Utilizando *Maps* complete los siguientes incisos en el proyecto *product*:

   4.1 Implementar la clase *StockManager* encargada de mantener una colección de objetos *Product*. Completar la implementación de los métodos:

   - *findProduct* Se encarga de buscar en la colección un producto con el *ID* proporcionado y si lo encuentra lo retorna, caso contrario retorna *null*.
   - *numberInStock* retorna la cantidad en stock del producto con el *ID* proporcionado.
   - *delivery*: se encarga de encontrar el producto con el *ID* proporcionado y luego llama al método *increaseQuantity*.
   - *printProductDetails* Itera sobre la colección de productos e imprime por consola el resultado de la llamada al método *toString* de cada producto.

   4.2 Agregue un método en la clase *StockManager* que retorne la lista de todos los productos con niveles de stock por debajo de un número dado (pasado como parámetro).

   4.3 Modifique el método *addProduct* de manera tal que un nuevo producto no pueda ser agregado a la colección de productos con el mismo *ID* que uno existente.

   4.4 Agregue un método a la clase *StockManager* con el siguiente perfil:

   ```
   public Product findProduct(String name).
   ```

   Este método retorna un producto encontrado por su nombre en lugar de su *ID*.
   Aclaración: dos objetos *String*, $s1$ y $s2$, pueden ser comparados por igualdad con la expresión booleana:

   ```
   s1.equals(s2)
   ```