Nombre y Apellido: ..... N° Legajo: .....

Segundo Parcial de Programación Orientada a Objetos (72.33)

13/11/2019

Ejercicio 1	*Ejercicio 2*	*Ejercicio 3*	*Nota*	*Firma Docente*
/2.5	**/3.5**	**/4**		

- **<u>Condición mínima de aprobación: SUMAR 5 PUNTOS</u>.**
- **Las soluciones que no se ajusten al paradigma OO, no serán aceptadas.**
- **Las soluciones que no se ajusten estrictamente al enunciado, no serán aceptadas.**
- **Puede entregarse en lápiz.**
- **No es necesario escribir las sentencias `import`.**
- **Además de las clases solicitadas se pueden agregar las que consideren necesarias.**
- **Escribir en cada hoja Nombre, Apellido, Legajo, Número de Hoja y Total Hojas entregadas.**

Ejercicio 1

Se desea crear una colección que ofrezca el método **`reduce`**. Este método recibe un valor inicial y una función de reducción que indica cómo cambia el acumulador en cada iteración en función de cada valor de la colección.

Implementar <u>todo</u> lo necesario para que, con el siguiente programa de prueba, se obtenga la salida indicada en los comentarios, donde **"`=> xxx`"** es la aclaración de cómo se consigue ese valor y no se debe imprimir en la salida**:**

```
public class SimpleListTester {

   public static void main(String[] args) {
       SimpleList<Integer> simpleList = new SimpleArrayList<>();

       simpleList.add(1);
       simpleList.add(3);
       simpleList.add(5);
       simpleList.add(7);

       System.out.println(simpleList.size()); // 4

       System.out.println(simpleList.contains(0)); // false

       Integer sum = simpleList.reduce(0, (accum, value) -> accum + value);
       System.out.println(sum); // 16 => 0 + 1 + 3 + 5 + 7

       SimpleList<Integer> emptyList = new SimpleArrayList<>();

       Integer emptySum = emptyList.reduce(0, (accum, value) -> accum + value);
       System.out.println(emptySum); // 0 => 0

       Integer prod = simpleList.reduce(1, (accum, value) -> accum * value);
       System.out.println(prod); // 105 => 1 * 1 * 3 * 5 * 7

       Integer emptyProd = emptyList.reduce(1, (accum, value) -> accum * value);
       System.out.println(emptyProd); // 1 => 1

       String s = simpleList.reduce("", (accum, value) -> accum + String.format("<%d>", value));
       System.out.println(s); // <1><3><5><7> => "" + "<1>" + "<3>" + "<5>" + "<7>"

       String t = emptyList.reduce(".", (accum, value) -> accum + String.format("<%d>", value));
       System.out.println(t); // . => "."
   }

}
```

Ejercicio 2

Se desea implementar un sistema para cargar los pasajeros de un vuelo para obtener luego el **orden de llamada para el embarque**. Un pasajero cuenta con su **nombre** (que no se repite en el vuelo), una **fila de asiento** y una **categoría** de membresía de la aerolínea.

Existen dos formas de embarque:

- **Por fila de asiento:** Donde embarcan primero los pasajeros de las filas menores (orden ascendente en función de la fila del asiento).
- **Por categoría del pasajero:** Donde embarcan primero los pasajeros de mejor categoría (orden ascendente en función de la categoría del pasajero donde la mejor categoría `EMERALD` tiene el valor más bajo y la peor categoría `ECONOMY` tiene el valor más alto).

Implementar todo lo necesario para que, con el siguiente programa de prueba

```
public class BoardingFlightTester {

   public static void main(String[] args) {
       PassengerCategory[] categories = PassengerCategory.values();
       System.out.println(Arrays.toString(categories));
       Arrays.sort(categories);
       System.out.println(Arrays.toString(categories));
       System.out.println("-----");
       BoardingFlight rowBoardingFlight = new RowBoardingFlight();
       rowBoardingFlight.addForBoarding("Passenger 3", 10, PassengerCategory.ECONOMY);
       rowBoardingFlight.addForBoarding("Passenger 2", 5, PassengerCategory.RUBY);
       rowBoardingFlight.addForBoarding("Passenger 1", 15, PassengerCategory.ECONOMY);
       rowBoardingFlight.addForBoarding("Passenger 4", 5, PassengerCategory.EMERALD);
       Iterator<String> rowIterator = rowBoardingFlight.boardingCallIterator();
       while(rowIterator.hasNext()) {
           System.out.println(rowIterator.next());
       }
       try {
           rowIterator.next();
       } catch (NoSuchElementException ex) {
           System.out.println("No more elements");
       }
       System.out.println("-----");
       BoardingFlight categoryBoardingFlight = new CategoryBoardingFlight();
       categoryBoardingFlight.addForBoarding("Passenger 3", 10, PassengerCategory.ECONOMY);
       categoryBoardingFlight.addForBoarding("Passenger 2", 5, PassengerCategory.RUBY);
       categoryBoardingFlight.addForBoarding("Passenger 1", 15, PassengerCategory.ECONOMY);
       categoryBoardingFlight.addForBoarding("Passenger 4", 5, PassengerCategory.EMERALD);
       Iterator<String> categoryIterator = categoryBoardingFlight.boardingCallIterator();
       while(categoryIterator.hasNext()) {
           System.out.println(categoryIterator.next());
       }
       try {
           categoryIterator.next();
       } catch (NoSuchElementException ex) {
           System.out.println("No more elements");
       }
   }

}
```

se obtenga la siguiente salida

```
[EMERALD, SAPPHIRE, RUBY, ECONOMY]
[EMERALD, SAPPHIRE, RUBY, ECONOMY]
-----
Passenger 2
Passenger 4
Passenger 3
Passenger 1
No more elements
-----
Passenger 4
Passenger 2
Passenger 1
Passenger 3
No more elements
```

Ejercicio 3

Se cuenta con la interfaz **`Cache`** que modela una forma de consultar y establecer valores asociados a claves de forma similar a un mapa, pero informando qué usuario realizó determinada operación en la fecha indicada. A motivos de simplificar su uso, ambos identificadores usuario y fecha son instancias de `String`.

```
public interface Cache<K, V> extends Map<K,V> {

   void put(String user, String date, K key, V value);

   V get(String user, String date, K key);

}
```

Además ya se cuenta con la implementación **`BaseCache`** que se muestra a continuación:

```
public class BaseCache<K, V> extends HashMap<K, V> implements Cache<K, V> {

   @Override
   public void put(String user, String date, K key, V value) {
       System.out.println(user + " put value " + value + " for key " + key + " on " + date);
       super.put(key, value);
   }

   @Override
   public V get(String user, String date, K key) {
       V value = super.get(key);
       System.out.println(user + " retrieved value " + value + " for key " + key + " on " +
date);
       return value;
   }

}
```

Se desea contar con una nueva implementación de la interfaz `Cache` que permita **<u>limitar el número de operaciones de lectura y escritura que cada usuario puede realizar por día</u>** según las siguientes dos políticas:

- **<u>Política ilimitada:</u>** no hay ninguna limitación diaria para ambas operaciones.
- **<u>Política limitada:</u>** Se permite un máximo de 2 (dos) lecturas y 1 (una) escritura, por día y por usuario.

Superado el límite por un usuario en un determinado día, si el usuario intenta realizar nuevamente la misma operación en el mismo día se debe arrojar la excepción `RateLimitedException`. Una vez llegado al límite, debe ser posible realizar operaciones en otras fechas y/o para otros usuarios y/o la otra operación para el mismo usuario en el mismo día (si es que aún le queda cuota).

Implementar todo lo necesario para que, con el siguiente programa de prueba

```
public class RateLimitedCacheTester {

   private static final String USER1 = "Alice";
   private static final String USER2 = "Bob";
   private static final String DATE1 = "11/11/2019";
   private static final String DATE2 = "12/11/2019";

   public static void main(String[] args) {
       RateLimitedCache<Integer, String> cache = new RateLimitedCache<>();

       System.out.println(cache.size());

       cache.register(USER1, QuotaType.LIMITED);

       System.out.println("Testing puts with a limited quota (maximum 1 per date)");
       cache.put(USER1, DATE1, 1, "1");
       try {
           cache.put(USER1, DATE1, 1, "2");
       } catch (RateLimitedException e) {
           System.out.println("Cannot put 1->\"2\" on  " + DATE1);
       }
       System.out.println("-----");
```

```
        System.out.println(cache.get(USER1, DATE1, 1));
        System.out.println("-----");

        System.out.println("Testing reads with a limited quota (maximum 2 per date)");
        cache.put(USER1, DATE2, 2, "2");
        System.out.println("-----");
        System.out.println(cache.get(USER1, DATE2, 1));
        System.out.println("-----");
        System.out.println(cache.get(USER1, DATE2, 2));
        System.out.println("-----");
        try {
            System.out.println(cache.get(USER1, DATE2, 3));
        } catch (RateLimitedException e) {
            System.out.println("Cannot read 1 on " + DATE2);
        }

        cache.register(USER2, QuotaType.UNLIMITED);

        System.out.println("Testing puts with a unlimited");
        cache.put(USER2, DATE1, 3, "3");
        cache.put(USER2, DATE1, 3, "4");

        System.out.println("Testing reads with a unlimited");
        System.out.println("-----");
        System.out.println(cache.get(USER2, DATE1, 3));
        System.out.println("-----");
        System.out.println(cache.get(USER2, DATE1, 3));
        System.out.println("-----");
        System.out.println(cache.get(USER2, DATE1, 3));
        System.out.println("-----");

        System.out.println(cache.size());
    }

}
```

se obtenga la siguiente salida

```
0
Testing puts with a limited quota (maximum 1 per date)
Alice put value 1 for key 1 on 11/11/2019
Cannot put 1->"2" on  11/11/2019
-----
Alice retrieved value 1 for key 1 on 11/11/2019
1
-----
Testing reads with a limited quota (maximum 2 per date)
Alice put value 2 for key 2 on 12/11/2019
-----
Alice retrieved value 1 for key 1 on 12/11/2019
1
-----
Alice retrieved value 2 for key 2 on 12/11/2019
2
-----
Cannot read 1 on 12/11/2019
Testing puts with a unlimited
Bob put value 3 for key 3 on 11/11/2019
Bob put value 4 for key 3 on 11/11/2019
Testing reads with a unlimited
-----
Bob retrieved value 4 for key 3 on 11/11/2019
4
-----
Bob retrieved value 4 for key 3 on 11/11/2019
4
-----
Bob retrieved value 4 for key 3 on 11/11/2019
4
-----
3
```