TP N°9: Diseño Avanzado en Java

Ejercicio 1

Un iterador cíclico es un iterador que, a partir de una colección, permite recorrerla infinitamente. Al igual que un iterador convencional permite recorrer todos los elementos de la colección, pero la diferencia es que, una vez consumido todos los elementos de la colección, vuelve a recorrer los mismos elementos.

Se desea implementar un iterador cíclico que, ante cada invocación a `next`, retorne un par de elementos de la colección.

El método `next` lanza una `NoSuchElementException` cuando una invocación a `hasNext` hubiera retornado `false`.

Implementar todo lo necesario para que, con el siguiente programa:

```
package ar.edu.itba.poo.tp9.cyclic;

import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class PairCyclicIteratorTester {

   public static void main(String[] args) {
       List<String> list = Arrays.asList("hola", "que", "tal", "todo", "bien");
       PairCyclicIterator<String> listIterator = new PairCyclicIterator<>(list);
       for(int i = 0; listIterator.hasNext() && i < 4; i++) {
           System.out.println(listIterator.next());
       }
       System.out.println("----------");
       Set<Integer> set = new HashSet<>();
       PairCyclicIterator<Integer> setIterator = new PairCyclicIterator<>(set);
       System.out.println(setIterator.hasNext());
       setIterator.next();
   }

}
```

se obtenga la siguiente salida

```
# hola + que #
# tal + todo #
# bien + hola #
# que + tal #
----------
false
Exception in thread "main" java.util.NoSuchElementException
```

Ejercicio 2

Se pide implementar la clase `ConcactIterator` la cual, recibe dos iteradores y genera la concatenación de ambos.

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

```
package ar.edu.itba.poo.tp9.concat;

import java.util.Arrays;
import java.util.Iterator;
import java.util.List;

public class ConcatIteratorTester {

   public static void main(String[] args) {
       List<String> list1 = Arrays.asList("a", "b", "c", "d");
       List<String> list2 = Arrays.asList("1", "2", "3", "4");
       List<String> list3 = Arrays.asList("alpha", "beta");
       Iterator<String> iterator = new ConcatIterator<>(
               new ConcatIterator<>(list1.iterator(), list2.iterator()),
               list3.iterator());
       while(iterator.hasNext()) {
           System.out.print(iterator.next());
       }
   }

}
```

se obtenga la siguiente salida

```
abcd1234alphabeta
```

Ejercicio 3

Dada la siguiente clase `Person`:

```
package ar.edu.itba.poo.tp9.person;

import java.util.Date;

public class Person {

   private String firstName;
   private String lastName;
   private Date bornDate;

   public Person(String firstName, String lastName, Date bornDate) {
       this.firstName = firstName;
       this.lastName = lastName;
       this.bornDate = bornDate;
   }

   @Override
   public boolean equals(Object o) {
       if (this == o) return true;
       if (!(o instanceof Person))
           return false;
       Person person = (Person) o;
       if (!firstName.equals(person.firstName))
           return false;
       return lastName.equals(person.lastName);
   }
```

```
    @Override
    public int hashCode() {
        int result = firstName.hashCode();
        result = 31 * result + lastName.hashCode();
        return result;
    }

    @Override
    public String toString() {
        return "Person{" +
                "firstName='" + firstName + '\'' +
                ", lastName='" + lastName + '\'' +
                '}';
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

}
```

implementar la siguiente interfaz `PersonCollection`:

```
package ar.edu.itba.poo.tp9.person;

import java.util.List;

public interface PersonCollection {

    void addPerson(Person aPerson);

    List<Person> findByLastName(String lastName);

    Person findByName(String firstName, String lastName);

}
```

Ejercicio 4

Se está implementando un sistema para rankear libros. Luego, dicha información se puede recuperar para generar un listado de los mismos ordenados por ranking. Además, los libros se pueden asociar a un género. También se puede generar un listado por género, también ordenado. Si un libro no tiene género, se lo puede agregar luego.

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

```
package ar.edu.itba.poo.tp9.books;

public class RankerTester {

    public static void main(String args[]) {
```

```
        Ranker ranker = new Ranker();

        Genre fantasy = new Genre("Fantasy");
        Genre crimeFiction = new Genre("Crime Fiction");
        Genre drama = new Genre("Drama");

        Book hp7 = new Book("Harry Potter and the Deadly Hallows", "JK Rowling");
        Book t2t = new Book("The Two Towers", "JRR Tolkien");
        Book theHobbit = new Book("The Hobbit", "JRR Tolkien");
        Book studyInScarlet = new Book("A Study in Scarlet", "Arthur Conan Doyle");
        Book hamlet = new Book("Hamlet", "William Shakespeare");
        Book prejudice = new Book("Pride and Prejudice", "Jane Austen");
        Book eragon = new Book("Eragon", "Christopher Paolini");

        ranker.add(fantasy, hp7);
        ranker.add(fantasy, theHobbit);
        ranker.add(fantasy, t2t);
        ranker.add(crimeFiction, studyInScarlet);
        ranker.add(drama, hamlet);
        ranker.add(drama, prejudice);

        ranker.rateUp(hp7);
        ranker.rateUp(hp7);
        ranker.rateUp(hp7);
        ranker.rateUp(theHobbit);
        ranker.rateUp(theHobbit);
        ranker.rateUp(hamlet);
        ranker.rateUp(new Book("Eragon", "Christopher Paolini"));
        ranker.rateUp(eragon);

        ranker.printRanking(fantasy);
        System.out.println("---------------------------------");
        ranker.printRanking(drama);
        System.out.println("---------------------------------");
        ranker.printRanking(new Genre("Non-Fiction"));
        System.out.println("---------------------------------");
        ranker.printRanking();
    }

}
```

se obtenga la siguiente salida:

```
Ranking of Fantasy
Harry Potter and the Deadly Hallows : 3
The Hobbit : 2
The Two Towers : 0
---------------------------------
Ranking of Drama
Hamlet : 1
Pride and Prejudice : 0
---------------------------------
Ranking of Non-Fiction
---------------------------------
General Ranking
Harry Potter and the Deadly Hallows : 3
Eragon : 2
The Hobbit : 2
```

```
Hamlet : 1
A Study in Scarlet : 0
Pride and Prejudice : 0
The Two Towers : 0
```

Ejercicio 5

Se cuenta con la interfaz `IterableBag` que extiende a la interfaz `Bag` del ***Ejercicio 7*** del ***TP N°8*** y permite iterar por los elementos de la bolsa de dos formas:

```java
package ar.edu.itba.poo.tp9.bag;

import ar.edu.itba.poo.tp8.bag.Bag;

public interface IterableBag<E> extends Bag<E> {

   /**
    * Para iterar, en orden descendente, por todos los elementos que hay en la bolsa.
    */
   Iterable<E> elements();

   /**
    * Para iterar, en orden descendente, por todos los elementos distintos
    * que hay en la bolsa.
    */
   Iterable<E> elementsDistinct();

}
```

Realizar el diagrama de clases. Implementar todo lo necesario para que, con el siguiente programa:

```java
package ar.edu.itba.poo.tp9.bag;

public class IterableBagTester {

   public static void main(String[] args) {
       IterableBag<String> stringBag = new IterableBagImpl<>();
       System.out.println(stringBag.contains("hola"));
       stringBag.add("hola");
       System.out.println(stringBag.contains("hola"));
       stringBag.add("que");
       stringBag.add("que");
       System.out.println(stringBag.count("que"));
       stringBag.add("tal");
       System.out.println(stringBag.count("que"));
       System.out.println(stringBag.sizeDistinct());
       for(String elem : stringBag.elementsDistinct()) {
           System.out.println(elem);
       }
       System.out.println(stringBag.size());
       for(String elem : stringBag.elements()) {
           System.out.println(elem);
       }
       stringBag.remove("tal");
```

```
        System.out.println(stringBag.sizeDistinct());
        System.out.println("----------");
        IterableBag<Integer> integerBag = new IterableBagImpl<>();
        integerBag.remove(2);
    }

}
```

se obtenga la siguiente salida:

```
false
true
2
2
3
tal
que
hola
4
tal
que
que
hola
2
----------
Exception in thread "main" java.util.NoSuchElementException
```

¿Es necesario establecer alguna restricción sobre el tipo genérico `E`?

Ejercicio 6

Un multimapa es un mapa que admite más de un valor para una misma clave. Se cuenta con la siguiente interfaz `MultiMap`.

```
package ar.edu.itba.poo.tp9.multimap;

public interface MultiMap<K, V>{

    /**
     * Agrega un par key,value al multimapa si el par no existe.
     */
    void put(K key, V value);

    /**
     * Cantidad de valores del multimapa.
     */
    int size();

    /**
     * Cantidad de valores del multimapa para la clave key.
     */
    int size(K key);

    /**
     * Elimina la clave del multimapa (con todos sus valores) si existe.
```

```
     */
    void remove(K key);

    /**
     * Elimina el valor value de la clave key si existe.
     */
    void remove(K key, V value);

    /**
     * Colección ordenada descendentemente de valores de clave key.
     */
    Iterable<V> get(K key);

}
```

Implementar todo lo necesario (pudiéndose modificar la interfaz) para que, con el siguiente programa,

```
package ar.edu.itba.poo.tp9.multimap;

public class MultiMapTester {

    public static void main(String[] args) {
        MultiMap<String,Integer> m = new MultiMapImpl<>();
        m.put("hola", 4);
        m.put("hola", 3);
        m.put("hola", 2);
        m.put("chau", 4);
        m.put("chau", 5);
        m.put("adios", 6);
        System.out.println(m.size());
        System.out.println(m.get("hola"));
        m.remove("adios");
        m.remove("hola", 2);
        System.out.println(m.get("hola"));
        System.out.println(m.get("adios"));
        System.out.println(m.size());
    }

}
```

se obtenga la siguiente salida

```
6
[4, 3, 2]
[4, 3]
null
4
```

Ejercicio 7

Se cuenta con la interfaz `SortedMap` que modela a un mapa ordenado por las claves y permite obtener la clave mayor y el valor asociado a la clave mayor con los siguientes métodos:

```java
package ar.edu.itba.poo.tp9.sortedMap;

import java.util.Map;

public interface SortedMap<K,V> {

   K higherKey();

   V higherValue();

}
```

Implementar todo lo necesario (pudiéndose modificar la interfaz) para que con el siguiente programa de prueba

```java
package ar.edu.itba.poo.tp9.sortedMap;

public class SortedMapTester {

   public static void main(String[] args) {
       SortedMap<MyComparableClass, String> sm = new SortedMapImpl<>();
       MyComparableClass obj1 = new MyComparableClass("A");
       MyComparableClass obj2 = new MyComparableClass("B");
       MyComparableClass obj3 = new MyComparableClass("C");
       MyComparableClass obj4 = new MyComparableClass("D");
       obj1.sortableIdentifier = "Key 1";
       obj2.sortableIdentifier = "Key 2";
       obj3.sortableIdentifier = "Key 3";
       obj4.sortableIdentifier = "Key 4";
       sm.put(obj1, "Value 1");
       sm.put(obj2, "Value 2");
       sm.put(obj3, "Value 3");
       sm.put(obj4, "Value 4");
       System.out.println("----------");
       for(Map.Entry<MyComparableClass, String> each : sm) {
           System.out.println(each.getKey());
       }
       System.out.println("----------");
       System.out.println(sm.higherKey());
       System.out.println(sm.higherValue());

       sm.remove(obj1);
       sm.remove(obj2);
       sm.remove(obj4);
       sm.remove(obj3);
       obj1.sortableIdentifier = "Key 4";
       obj2.sortableIdentifier = "Key 3";
       obj3.sortableIdentifier = "Key 2";
       obj4.sortableIdentifier = "Key 1";
       sm.put(obj1, "Value 1");
       sm.put(obj2, "Value 2");
       sm.put(obj3, "Value 3");
       sm.put(obj4, "Value 4");
       System.out.println("----------");
       for(Map.Entry<MyComparableClass, String> each : sm) {
           System.out.println(each.getKey());
       }
```

```
        System.out.println("----------");
        System.out.println(sm.higherKey());
        System.out.println(sm.higherValue());
    }

}
```

se obtenga la siguiente salida:

```
----------
A - (Key 1)
B - (Key 2)
C - (Key 3)
D - (Key 4)
----------
A - (Key 1)
Value 1
----------
D - (Key 1)
C - (Key 2)
B - (Key 3)
A - (Key 4)
----------
D - (Key 1)
Value 4
```

Analizando el programa de prueba responder:

- La interfaz `SortedMap` ¿debería extender de alguna otra interfaz? ¿Por qué?
- ¿Por qué en el programa de prueba se eliminan y se vuelven a agregar los datos una vez cambiado el `sortableIdentifier`? Justificar

Ejercicio 8

Implementar la clase `Bank` que almacena un conjunto de cuentas (para ello utilizar la familia de clases `BankAccount` del ***Ejercicio 5*** del ***TP Nº5)***.

El banco deberá almacenar el conjunto de cuentas que le pertenecen. Se deben ofrecer métodos para operar con las cuentas, como se muestra en el siguiente fragmento de código:

```
Bank bank = new Bank();
BankAccount c1 = new CheckingAccount(1234, 5000);
BankAccount c2 = new CheckingAccount(3462, 5000);
bank.addAccount(c1);
bank.addAccount(c2);
System.out.println(bank.accountSize());
System.out.println(bank.totalAmount());
c1.deposit(100);
c2.deposit(200);
bank.removeAccount(c2);
System.out.println(bank.accountSize());
System.out.println(bank.totalAmount());
```

que produce la siguiente salida:

```
2
0.0
1
100.0
```

Se desea contar con un nuevo tipo de cuenta para modelar a los clientes especiales. La misma ofrece descuentos de manera automática para las compras realizadas en ciertos locales. Los locales que participan de dichas promociones son obtenidos del `ShopDiscountsProvider` mediante el método `getDiscount`. Dicho método devuelve el porcentaje de descuento a aplicar al importe a debitar.

A continuación se presenta la implementación del `ShopDiscountsProvider`:

```
package ar.edu.itba.poo.tp9.bank;

import java.util.HashMap;
import java.util.Map;

public class ShopDiscountsProvider {

   private Map<String, Double> discounts = new HashMap<>();

   public void addShop(String shop, double discount) {
       this.discounts.put(shop, discount);
   }

   public double getDiscount(String shop) {
       if (discounts.containsKey(shop)) {
           return discounts.get(shop);
       }
       return 0;
   }

}
```

Realizar la implementación de la clase `PremiumAccount` que modele las cuentas para los clientes especiales. El siguiente programa demuestra la manera de uso de `PremiumAccount`:

```
package ar.edu.itba.poo.tp9.bank;

public class PremiumAccountTester {

   public static void main(String[] args) {
       ShopDiscountsProvider shopDiscountsProvider = new ShopDiscountsProvider();
       shopDiscountsProvider.addShop("Falabella", 0.1D);
       shopDiscountsProvider.addShop("Nike", 0.15D);
       shopDiscountsProvider.addShop("Garbarino", 0.3D);
       PremiumAccount premiumAccount = new PremiumAccount(9999, 5000,
shopDiscountsProvider);
       premiumAccount.deposit(1000);
       premiumAccount.extract(150, "Nike");
       premiumAccount.extract(250, "Lacoste");
       premiumAccount.extract(50, "Starbucks");
       premiumAccount.extract(150, "Nike");
       premiumAccount.showMovements();
   }
```

```
}
```

Al ejecutar el programa se obtiene por consola:

```
Movements for Account 9999
Deposit $1000.0
Extraction $127.5 for shop Nike
Extraction $250.0 for shop Lacoste
Extraction $50.0 for shop Starbucks
Extraction $127.5 for shop Nike
```

Como se ve en el ejemplo, ahora es necesario guardar un registro de todos los movimientos realizados en las cuentas para consultarlo con el método `showMovements()`. ¿Cómo lo implementaría?

Ejercicio 9

Se cuenta con las siguientes clases que modelan un cajero automático. Este cajero es especial porque entrega dinero a cualquier usuario que se lo pida. Al solicitar una suma de dinero el cajero determina cuál es la mínima cantidad de billetes de cada valor que debe entregar, considerando el stock de billetes disponibles.

```
package ar.edu.itba.poo.tp9.cashier;

import java.util.ArrayList;
import java.util.List;

public class NoteDispenser {

   private Cashier cashier;
   private List<Integer> notes = new ArrayList<>();

   /**
    * Cancela la transaccion actual devolviendo los billetes al cajero.
    */
   public void reset() {
       cashier.loadMoney(notes);
       notes = new ArrayList<>();
   }

   /**
    * Acumula una cantidad de billetes de cierto valor en la transaccion.
    */
   public void storeNote(Integer count, Integer value) {
       for (int i=0; i<count; i++)
           notes.add(value);
   }

   /**
    * Entrega al usuario los billetes acumulados en la transaccion.
    */
   public void deliver() {
       for (Integer i: notes) {
```

```
            System.out.println("Billete de " + i );
        }
    }

    public void setCashier(Cashier cashier) {
        this.cashier = cashier;
    }

}
```

```
package ar.edu.itba.poo.tp9.cashier;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class Cashier {

    private NoteDispenser dispenser;
    public static Integer values[] = {100,50,20,10,5,2,1};
    private Map<Integer, Integer> cash = new HashMap<>();

    public Cashier(NoteDispenser dispenser) {
        this.dispenser = dispenser;
        this.dispenser.setCashier(this);
        for (Integer i: values) {
            cash.put(i, 0);
        }
    }

    /**
     * Este método extrae del cajero la cantidad mínima de billetes
     * de cada valor necesarios para formar el importe solicitado.
     * @throws NoCashException cuando no tiene billetes suficientes.
     */
    public void getMoney(Integer amount) {
        int i = 0;
        while (amount != 0 && i < values.length) {
            if (amount >= values[i]) {
                int neededCount = amount / values[i];
                int realCount = getMoney(neededCount, values[i]);
                amount -= realCount * values[i];
            }
            i++;
        }
        if (amount != 0) {
            dispenser.reset();
            throw new NoCashException();
        }
        dispenser.deliver();
    }

    /**
     * Extrae del cajero cierta cantidad de billetes de cierto valor. De no
     * contar con dicha cantidad extrae lo existente.
     */
    protected Integer getMoney(Integer count, Integer value) {
        int realCount = cash.get(value);
```

```
        if (realCount < count) {
            count = realCount;
        }
        if (count > 0) {
            cash.put(value, realCount - count);
            dispenser.storeNote(count, value);
        }
        return count;
    }

    /**
     * Carga en el cajero una lista de billetes.
     */
    public void loadMoney(List<Integer> newCash) {
        for (Integer i: newCash) {
            cash.put(i, cash.get(i) + 1);
        }
    }

    public NoteDispenser getDispenser() {
        return dispenser;
    }

}
```

```
package ar.edu.itba.poo.tp9.cashier;

public class NoCashException extends RuntimeException {

    private static final String MESSAGE = "No hay dinero disponible";

    public NoCashException() {
        super(MESSAGE);
    }

}
```

Se pide implementar un nuevo tipo de cajero `ChangeCashier` que utilice un criterio diferente para decidir la cantidad de billetes a entregar de cada valor, en líneas generales se pretende que de cambio.

El criterio es el siguiente: se debe procesar cada uno de los valores posibles, ordenados ascendentemente. El proceso consiste en verificar que haya en stock al menos un billete de dicho valor, en cuyo caso su importe se le resta al importe solicitado, y luego se procede igual con cada uno de los siguientes valores. Luego de procesar el billete de 100, este proceso vuelve a comenzar, hasta tanto se complete el importe solicitado, o bien no haya suficientes billetes del valor necesario, en cuyo caso se cancela el proceso y se opera con el criterio ya implementado en la clase `Cashier`.

Ejemplos:

Si se piden $16

Billetes	*Stock inicial*	*Entregado*
1	5	2
2	5	2
5	5	2
10	5	
20	5	
50	5	
100	5	

Si se piden $20

Billetes	*Stock inicial*	*Entregado*
1	5	3
2	5	1
5	5	1
10	5	1
20	5	
50	5	
100	5	

Si se piden $16

Billetes	*Stock inicial*	*Entregado*
1	5	2
2	5	2
5	0	
10	5	1
20	5	
50	5	
100	5	

Si se piden $4

Billetes	*Stock inicial*	*Entregado*
1	1	
2	5	2*
5	5	
10	5	
20	5	
50	5	
100	5	

(*) Entrega los billetes según el criterio de la clase ya implementada, porque no lo puede hacer según el nuevo criterio.

Se pide implementar la clase `ChangeCashier.`

Ejercicio 10

Implementar una jerarquía de clases para modelar los distintos tipos de preguntas que pueden aparecer en un juego de preguntas y respuestas. Los tipos son los siguientes:

Pregunta con respuesta de texto simple

La respuesta consiste en un texto, que debe coincidir con el ingresado por el usuario, ignorando mayúsculas y minúsculas.

Ejemplo:
¿Cuál es la capital de La Pampa?
Santa Rosa

Pregunta con respuesta numérica

La respuesta debe ser un valor numérico entero.

Ejemplo:
¿En qué año fue la Revolución de Mayo?
1810

Pregunta de opciones múltiples con una única respuesta

Una pregunta que contiene varias opciones posibles para responder, y solamente una de ellas es correcta.

Ejemplo:
¿Cuál es la capital de Australia?

a) Sydney, b) Canberra, c) Melbourne
Respuesta: b) Canberra

Pregunta de opciones múltiples con varias respuestas

Una pregunta que contiene varias opciones posibles, en donde la respuesta está formada por un subconjunto de estas opciones

Ejemplo:
¿Cuáles de las siguientes son ciudades capitales de países?
a) Buenos Aires, b) Punta del Este, c) Natal, d) Montevideo, e) Santiago de Chile
Respuesta: a) Buenos Aires, d) Montevideo, e) Santiago de Chile

Pregunta con respuesta verdadero/falso

Una pregunta en donde la respuesta consiste en indicar si la afirmación es verdadera o falsa.
Ejemplo:
La capital de Argentina es La Plata.
Respuesta: Falso

Realizar el diagrama de clases. Implementar además un programa de prueba con las preguntas y respuestas de los ejemplos anteriores.