

## TP N°8: Colecciones

La siguiente guía cubre los contenidos vistos en las clases teórica 7. **Anotaciones y Colecciones**

### ~~Ejercicio 1~~

Indicar cuáles de las siguientes asignaciones son válidas:

```
package ar.edu.itba.poo.tp8;

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

public class Ej1 {

    public static void main(String[] args) {
        List<Integer> list1 = new ArrayList();
        List<Integer> list2 = new ArrayList<Integer>(); Se puede sacar el Integer de la izquierda
        List<Number> list3 = new ArrayList<Integer>(); X Tipos incompatibles
        List list4 = new ArrayList<Integer>();
        List<? extends Number> list5 = new ArrayList<? extends Number>(); X el wildcard no se puede instanciar directamente
        List<?> list6 = new ArrayList(); No es necesario poner el ?
    }
}
```

### ~~Ejercicio 2~~

Explicar por qué no compila el siguiente fragmento de código:

```
package ar.edu.itba.poo.tp8;

import ar.edu.itba.poo.tp7.list.LinearList;
import ar.edu.itba.poo.tp7.list.LinearListImpl;

public class Ej2 {

    public static void main(String[] args) {
        LinearList<String> list = new LinearListImpl<>();
        list.add("hola");
        list.add("mundo");
        for (String str : list) {
            System.out.println(str);
        }
    }
}
```

Realizar las modificaciones necesarias en la interfaz y en la implementación para que el fragmento anterior funcione correctamente.

~~Ejercicio 3~~

Modificar la implementación del **Ejercicio 4** del **TP N°6** para que el intervalo sea iterable. Actualizar el diagrama de clases. Escribir un programa de prueba.

~~Ejercicio 4~~

El siguiente fragmento de código intenta crear una lista de números enteros y luego eliminar aquellos que son pares. Explicar el error obtenido en tiempo de ejecución. Proponer una solución.

```
package ar.edu.itba.poo.tp8;

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

public class Ej4 {

    public static void main(String[] args) {
        List<Integer> list = new ArrayList<>();
        list.add(1);
        list.add(6);
        list.add(8);
        list.add(10);
        for (Integer i : list) {
            if (i % 2 == 0) {
                list.remove(i);
            }
        }
    }
}
```

~~Ejercicio 5~~

Implementar un método que dada una colección de objetos retorne el mayor (recorriéndola elemento a elemento, esto es, sin utilizar `Collections.max()` o similar). ¿El método debería ser de instancia o de clase? Utilizar tipos genéricos para exigir en tiempo de compilación que los objetos de la colección recibida implementen la interfaz `Comparable`. ¿El tipo genérico debe estar en la clase, en el método o en ambos?

**Ejercicio 6**

Modificar la implementación del **Ejercicio 3** del **TP N°6** para que en lugar de componer dos funciones permita componer N funciones, como se muestra en el siguiente ejemplo:

```
package ar.edu.itba.poo.tp8.function;

import ar.edu.itba.poo.tp6.function.LinearFunction;
import ar.edu.itba.poo.tp6.function.QuadraticFunction;
import ar.edu.itba.poo.tp6.function.SineFunction;

public class FunctionTester {

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

```

// armamos la funcion (2x)^2 como la composicion de 2x con x^2
CompositeFunction f1 = new CompositeFunction();
f1.addFunction(new LinearFunction(2, 0));           // y = 2x
f1.addFunction(new QuadraticFunction(1, 0, 0));     // y = x^2
System.out.println(f1.evaluate(1));                 // 4
System.out.println(f1.evaluate(2));                 // 16
// armamos la funcion 3 * sin(2x) componiendo 2*x con sin(x) y con 3*x
CompositeFunction f2 = new CompositeFunction();
f2.addFunction(new LinearFunction(2, 0));           // y = 2*x
f2.addFunction(new SineFunction());                 // y = sin(x)
f2.addFunction(new LinearFunction(3, 0));           // y = 3*x
System.out.println(f2.evaluate(0));                 // 0
System.out.println(f2.evaluate(Math.PI / 4));       // 3
// si no especificamos ninguna funcion, se lanza una excepcion.
CompositeFunction f3 = new CompositeFunction();
System.out.println(f3.evaluate(1));                 // IllegalStateException
}
}

```

### Ejercicio 7

Diseñar la interfaz **Bag** que define una colección de elementos no ordenados que admite repeticiones. Dado el siguiente programa de prueba, deducir la definición de la interfaz **Bag** y realizar la implementación **BagImpl**.

```

package ar.edu.itba.poo.tp8.bag;

public class BagTester {

    public static void main(String[] args) {
        Bag<String> bag = new BagImpl<>();
        bag.add("Hola");
        bag.add("Hola");
        System.out.println(bag.count("Hola"));        // 2
        bag.add("Chau");
        System.out.println(bag.size());                // 3
        System.out.println(bag.sizeDistinct());        // 2
        System.out.println(bag.contains("Chau"));      // true
        bag.remove("Hola");
        System.out.println(bag.count("Hola"));        // 1
        bag.remove("Hola");
        System.out.println(bag.contains("Hola"));      // false
        bag.remove("Hola");                           // NoSuchElementException
    }
}

```

### Ejercicio 8

Escribir la interfaz **FilterList** que extiende de **List** agregando un método que dada cierta condición (implementada a través de una nueva interfaz **Criteria**) retorne una nueva lista con los elementos de la original que satisfacen dicha condición. Implementar la clase **ArrayFilterList** que herede de **ArrayList** e implemente la nueva interfaz **FilterList**.

Implementar además un pequeño programa de prueba `FilterListTester` que genere una lista de objetos `Integer` y luego utilice el filtro para obtener aquellos que son pares.

### Ejercicio 9

Se desea implementar un conjunto genérico de elementos, en donde a cada elemento se le agrega un horario (horas y minutos), para luego poder consultar aquellos objetos que pertenezcan a un determinado rango horario. Realizar una implementación de la siguiente interfaz:

```
package ar.edu.itba.poo.tp8.timeSet;

import java.util.Set;

/**
 * Conjunto no ordenado de elementos, en donde a cada elemento se le asigna
 * un horario al momento de la inserción, en formato horas y minutos.
 *
 * @param <T> Tipo de los objetos almacenados.
 */
public interface TimeSet<T> {

    /**
     * Agrega un elemento al conjunto, junto con un horario especificado.
     * Si el elemento a agregar ya existía, se modifica el horario que tenía
     * anteriormente.
     *
     * @throws IllegalArgumentException Si el horario no es valido.
     */
    void add(T elem, int hours, int minutes);

    /**
     * Elimina un objeto del conjunto.
     *
     * @param elem Elemento a ser eliminado.
     */
    void remove(T elem);

    /**
     * Obtiene la cantidad de elementos del conjunto.
     */
    int size();

    /**
     * Determina si un objeto pertenece o no al conjunto.
     */
    boolean contains(T elem);

    /**
     * Obtiene todos los objetos que pertenezcan al rango
     * horario especificado.
     *
     * @throws IllegalArgumentException Si el limite superior es menor al
     * inferior, o los horarios no son validos.
     */
    Set<T> retrieve(int hoursFrom, int minutesFrom, int hoursTo, int minutesTo);
}
```

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

```
package ar.edu.itba.poo.tp8.timeSet;

public class TimeSetTester {

    public static void main(String[] args) {
        TimeSet<String> timeSet = new TimeSetImpl<>();
        timeSet.add("Taller P00", 16, 0);
        timeSet.add("Taller PI", 13, 0);
        timeSet.add("Clase POD", 18, 0);
        System.out.println(timeSet.size());
        System.out.println(timeSet.contains("Taller P00"));
        System.out.println(timeSet.retrieve(12, 0, 16, 0));
        timeSet.remove("Clase POD");
        System.out.println(timeSet.size());
        System.out.println(timeSet.retrieve(17, 30, 18, 30));
    }
}
```

se obtenga la siguiente salida

```
3
true
[Taller PI, Taller P00]
2
[]
```

### Ejercicio 10

Se cuenta con la interfaz `MultiSortedCollection<T>` que representa a una colección de elementos **sin repetidos** que mantiene a sus elementos ordenados según varios criterios simultáneos, en base a comparadores.

```
package ar.edu.itba.poo.tp8.multisortedCollection;

import java.util.Comparator;

public interface MultiSortedCollection<T> {

    /**
     * Agrega un comparador a la colección.
     */
    void add(Comparator<T> comp);

    /**
     * Agrega un elemento a la colección.
     * Arroja IllegalStateException si no se agrego al menos un comparador.
     */
    void add(T elem);

    /**
     * Elimina un elemento de la colección. Si el mismo no existe, no hace nada.
     */
}
```

```
    */  
    void remove(T elem);  
  
    /**  
     * Devuelve un objeto iterable en base al comparador pedido.  
     * Arroja IllegalArgumentException si el comparador no pertenece a la colección.  
     */  
    Iterable<T> iterable(Comparator<T> comp);  
}
```

**Se busca que la implementación sea eficiente en cuanto al tiempo de acceso a los elementos, por encima del uso de memoria.**

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

```
package ar.edu.itba.poo.tp8.multisortedCollection;  
  
import java.util.Comparator;  
  
public class MultiSortedCollectionTester {  
    public static void main(String[] args) {  
        MultiSortedCollection<Person> m = new MultiSortedCollectionImpl<>();  
        Comparator<Person> nameComparator = (o1, o2) ->  
o1.getName().compareTo(o2.getName());  
        m.add(nameComparator);  
        Comparator<Person> ageComparator = (o1, o2) -> o1.getAge() - o2.getAge();  
        m.add(ageComparator);  
        m.add(new Person("Ana", 15));  
        m.add(new Person("Juan", 20));  
        m.add(new Person("Pedro", 10));  
        for(Person person: m.iterable(nameComparator)) {  
            System.out.println(person);  
        }  
        System.out.println();  
        for(Person person: m.iterable(ageComparator)) {  
            System.out.println(person);  
        }  
        System.out.println();  
        m.remove(new Person("Ana", 15));  
        for(Person person: m.iterable(ageComparator)) {  
            System.out.println(person);  
        }  
    }  
}
```

se obtenga la siguiente salida

```
Ana # 15  
Juan # 20  
Pedro # 10  
  
Pedro # 10  
Ana # 15
```

Juan # 20

Pedro # 10

Juan # 20