

Sesión 18: Genéricos

Hoja de problemas


Programación 2

Ángel Herranz

aherranz@fi.upm.es

Universidad Politécnica de Madrid

2022-2023

 **Ejercicio 1.** Repasa las transparencias de la sesión de genéricos. En esta hoja de ejercicios sólo vamos a pedirte que conviertas en genéricos todas las clases relacionadas con cadenas enlazadas e implementaciones de listas.

`Node<T>` **Ejercicio 2.** En la hoja de ejercicios de cadenas enlazadas te toca eliminar la clase `NodoStr` y cambiarla por la siguiente clase genérica:

```
class Node<T> {
    T data;
    Node<T> next;
}
```

Operaciones **Ejercicio 3.** Ahora debes rehacer toda la hoja de ejercicios para utilizar la clase genérica `Node<T>` para que los nodos que antes eran de tipo `NodoStr` pasen a ser ahora `Node<String>`.

`IList<T>` **Ejercicio 4.** Convierte en genérico el interfaz `IListStr` de esta forma:

```
public interface IList<T> {
    void add(int index, T elem);
    void add(T elem);
    T get(int index);
    int size();
    void set(int index, T elem);
    int indexOf(T elem);
    void remove(int index);
    void remove(T elem);
    IListStr subList(int start, int end);
}
```

Ejercicio 5. Convierte en genérica la clase `LinkedListStr` de esta forma:

`LinkedList<T>` **Ejercicio 5.** Convierte en genérica la clase `LinkedListStr` de esta forma:

```
public class LinkedList<T> implements IList<T> {
    ...
}
```

Y completa toda la implementación.

`ListIntTest` **Ejercicio 6.** Implementa los tests de la nueva clase genérica `LinkedList<T>` utilizando en ellos listas de enteros (es decir, los tipos `IList<Integer>` y `LinkedList<Integer>`).

`Tupla` **Ejercicio 7.** Completa la implementación de `Tupla` con los métodos `toString` y `equals`.

Nota: Ten en cuenta que las variables de tipo `T1` y `T2` pueden ser cualquier clase pero no sabes cuál. Pero lo que es seguro es que sus objetos implementan los métodos de la clase `Object`. Por lo que puedes usar `toString` o `equals` de datos de esos tipos `T1` y `T2` sin ningún problema.

`Optional` **Ejercicio 8.** Existe un tipo muy especial en muchos lenguajes de programación que permiten evitar el típico caso en el que hay un `null` en una variable para representar que *no hay dato*. Dicho tipo suele llamarse `Optional` (aunque en algunos lenguajes se denomina `Maybe`). A continuación se ofrece un esqueleto:

```

public class Optional<T> {
    // TODO: definir aquí el/los atributos

    public Optional() {
        // TODO: completar el constructor para "no hay dato"
    }

    public Optional(T x) {
        // TODO: completar el constructor con dato
    }

    public T get() {
        // TODO: completar el observador que devuelve el dato
        // (si lo hay) o dispara una excepción si no hay dato
        return null;
    }

    public boolean isPresent() {
        // TODO: completar el observador que decide si hay dato
        return false;
    }

    public String toString() {
        // TODO: completar el observador toString
        return "";
    }
}

```

Tu tarea es escribir los tests y la implementación del tipo anterior.

Nota: para más información sobre este tipo ver

<https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html>