Trabajo práctico Nro. 3

|  |  |  |
| --- | --- | --- |
| Logo Departamento 2.jpg | **Asignatura: Programación II** | |
|  | |
| **Cursado:** Segundo semestre | **Horas semanales**: |
|  | **Horas semestrales:**  *Cantidad estimada de horas*  *semestrales/anuales.* |
| **Carrera**: *Tecnicatura Universitaria en*  *Programación* | **Nivel (Año):**  1°  2°  3° |
| **Ciclo Lectivo: 2024** |

**Integrantes de la Cátedra:**

* **DOCENTES:**

|  |  |  |
| --- | --- | --- |
| **Nombre del Profesor** | **Periodo** | **Cantidad horas materia** |
| **Cinthia Rigoni** |  | 6 horas |

* 1. **¿Qué es una relación involutiva en java?**

En Java, una relación involutiva se refiere a una relación entre dos clases en la que ambas clases tienen una referencia a la otra. Es decir, cada clase es un atributo de la otra. Esto crea una relación bidireccional en la que ambas clases pueden acceder y comunicarse entre sí.

* 1. **De un ejemplo de relación involutiva.**

Por ejemplo, considera dos clases ClaseA y ClaseB. Si ClaseA tiene una referencia a ClaseB como un atributo, y a su vez, ClaseB tiene una referencia a ClaseA como un atributo, entonces tenemos una relación involutiva entre ClaseA y ClaseB.

public class ClaseA {

private ClaseB objetoB;

// Constructor, getters y setters

}

public class ClaseB {

private ClaseA objetoA;

// Constructor, getters y setters

}

En este ejemplo, ClaseA y ClaseB tienen una relación involutiva porque cada una contiene una referencia a la otra. Esto permite que ambas clases interactúen entre sí de manera bidireccional.

* 1. **¿Qué métodos comunes se encuentran en la interfaz Collection y qué funcionalidades proporcionan?**

La interfaz Collection en Java define un conjunto de métodos comunes que son implementados por todas las clases que representan colecciones de elementos. Algunos de los métodos más comunes incluyen:

**1. int size():** Devuelve el número de elementos en la colección.

**2. boolean isEmpty():** Devuelve `true` si la colección está vacía, `false` de lo contrario.

**3. boolean contains(Object o):** Devuelve `true` si la colección contiene el elemento especificado, `false` de lo contrario.

**4. boolean add(E e):** Agrega el elemento especificado a la colección. Devuelve `true` si la colección cambió como resultado de la operación, `false` de lo contrario.

**5. boolean remove(Object o):** Elimina la primera aparición del elemento especificado de la colección, si está presente. Devuelve `true` si la colección cambió como resultado de la operación, `false` de lo contrario.

**6. void clear():** Elimina todos los elementos de la colección.

**7. Iterator<E> iterator():** Devuelve un iterador sobre los elementos en la colección.

**8. Object[] toArray():** Devuelve un array que contiene todos los elementos de la colección.

**9. boolean addAll(Collection<? extends E> c):** Agrega todos los elementos de la colección especificada a esta colección.

**10. boolean removeAll(Collection<?> c):** Elimina todos los elementos de la colección que están contenidos en la colección especificada.

**11. boolean retainAll(Collection<?> c):** Retiene solo los elementos de la colección que están contenidos en la colección especificada.

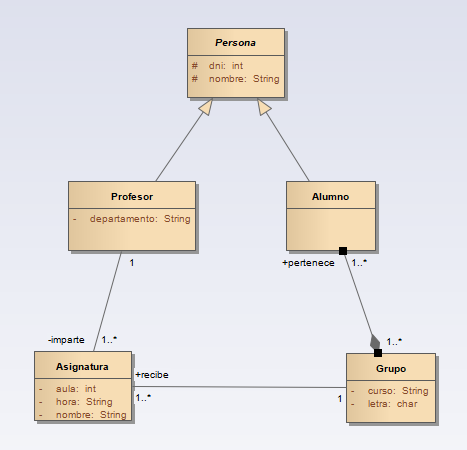
**12. boolean containsAll(Collection<?> c):** Devuelve `true` si la colección contiene todos los elementos de la colección especificada, `false` de lo contrario.

Estos son solo algunos de los métodos comunes definidos en la interfaz Collection. Proporcionan funcionalidades para manipular y trabajar con colecciones de elementos de manera genérica en Java.

* 1. **Explica el propósito de la interfaz Iterable en Java y cómo se utiliza.**
  2. **¿Qué ventajas ofrece el uso de la interfaz Iterable en comparación con simplemente iterar sobre una colección utilizando un bucle for estándar?**

A codificar!

1. Teniendo el siguiente diagrama:



1. Cree todas las clases del modelo, con todos los métodos necesarios.
2. Complete un objeto de cada clase, asignando algún valor a todos sus atributos, inclusive los de relación.
3. Realice las siguientes operaciones sobre el modelo:
   1. Teniendo un objeto de Profesor, muestre el aula de la 3ra. asignatura.
   2. Teniendo un profesor, muestre todos los nombres de las asignaturas que imparte.
   3. Teniendo una asignatura, muestre nombre y dni de los alumnos del grupo que recibe.
   4. Teniendo un alumno, muestre la letra del 3er. grupo al que pertenece.
   5. Teniendo un alumno, muestre todas las asignaturas recibidas por el 1er. grupo al que pertenece.
   6. Teniendo un objeto de Alumno y, pidiendo por pantalla, una asignatura existente, muestre desde él, el profesor que la imparte.
   7. Teniendo un objeto de Profesor, muestre los nombres de todos los alumnos inscriptos en el Grupo de la 2da. Asignatura.
   8. Responda: ¿En qué se diferencian una asociación de una composición y de una agregación?
4. En una empresa se desea modelar la relación entre los empleados y sus supervisores. Cada empleado tiene un nombre y puede tener un supervisor, que a su vez es otro empleado. Implementa la clase Empleado en Java con los siguientes requisitos:

La clase Empleado debe tener los siguientes atributos:

* + nombre de tipo String.
  + supervisor de tipo Empleado.

La clase Empleado debe tener los siguientes métodos:

* + Un constructor que reciba el nombre del empleado.
  + Un método asignarSupervisor(Empleado supervisor) que permita asignar un supervisor al empleado.
  + Un método obtenerNombreSupervisor() que retorne el nombre del supervisor del empleado, o "Sin supervisor" si no tiene uno.

En el método main, crea al menos dos instancias de la clase Empleado, asigna un supervisor a uno de ellos y muestra por pantalla el nombre del supervisor del primer empleado.

1. Crea una lista que contenga los días de la semana, luego:

* Inserta en la posición 4 el elemento “Juernes”.
* Muestra el contenido de las posiciones 3 y 4 de la lista.
* Muestra el primer elemento y el último de la lista.
* Elimina el elemento que contenga “Juernes” de la lista y comprueba que haya sido eliminado.
* Crea un iterador y muestras uno a uno los valores de la lista.
* Busca si existe en la lista un elemento que se denomine “Lunes”.
* Busca si existe en la lista un elemento que se denomine “Lunes”. No importa si está en mayúscula o minúscula.
* Busca si existe en la lista un elemento que se denomine “Lunes”. No importa si está en mayúscula o minúscula.
* Borra todos los elementos de la lista.
* En cada paso realizado debes mostrar por pantalla que la acción se haya realizado.

1. Escribe un programa para gestionar una lista de alumnos de una escuela. Cada alumno estará representado por una clase Alumno, que contendrá los siguientes atributos:

* Nombre (String)
* Edad (int)
* Nota (double)

Debes implementar las siguientes funcionalidades en tu programa:

* Permitir al usuario agregar nuevos alumnos a la lista.
* Mostrar la lista completa de alumnos, incluyendo su nombre, edad y nota.
* Calcular y mostrar la media de notas de todos los alumnos.
* Mostrar el alumno con la nota más alta.
* Permitir al usuario buscar un alumno por nombre y mostrar su información (nombre, edad y nota).

Utiliza una colección adecuada de Java para almacenar los objetos Alumno. Puedes elegir entre ArrayList, LinkedList, HashSet, HashMap, u otra colección según consideres más conveniente para el ejercicio.

Además, asegúrate de manejar las entradas del usuario de manera adecuada, proporcionando mensajes claros y opciones para cada funcionalidad.

1. Escribe un programa en Java para gestionar una lista de tareas pendientes. Cada tarea estará representada por una clase Tarea, que contendrá los siguientes atributos:

* Descripción (String)
* Prioridad (int) - donde valores más altos indican mayor prioridad

Debes implementar una clase llamada ListaDeTareas que implemente la interfaz Iterable<Tarea>. Esta clase debe permitir agregar nuevas tareas, eliminar tareas existentes y obtener el número total de tareas en la lista.

Además, la clase ListaDeTareas debe proporcionar un método para iterar sobre todas las tareas almacenadas, en orden de prioridad descendente. Esto significa que al iterar sobre la lista, las tareas con prioridad más alta deberían aparecer primero.

Finalmente, escribe un programa (main) que permita al usuario realizar las siguientes operaciones:

* Agregar una nueva tarea a la lista, especificando su descripción y prioridad.
* Mostrar todas las tareas de la lista, en orden de prioridad descendente.
* Eliminar una tarea específica de la lista, especificando su descripción.
* Salir del programa.