|  |
| --- |
| **Laboratorio II - Ciclo lectivo 2023** |

|  |  |
| --- | --- |
| **TP1: Relaciones en Java. Ingenieria directa.** | |
| **Unidad** | **1** |
| Tema: | **Clases Genéricas - Java** |
| **Resultados de Aprendizajes** | RA1:    RA2: |
| **Objetivo** | El objetivo de este trabajo práctico es que los estudiantes adquieran un entendimiento medio de clases genéricos |
| **Fecha de inicio** | 9 de mayo de 2024- |
| **Fecha de entrega:** | 16 de Mayo de 2024 - |
| **Comisiones** |  |
| **Modalidad de entrega :** | Actividad agendada en aula virtual |
| **Modalidad del Trabajo Práctico** | Desarrollo grupal – Entrega en aula virtual |
|  | |
| Enunciados  **Ejercicio 1**  Escribe una interfaz genérica Operable, que sea genérica y que declare las cuatro operaciones básicas: suma, resta, producto y división.  *Como puede hacerse*  Se escribe la interfaz Operable, con un tipo genérico E. Todos los métodos de esta interfaz reciben un objeto de tipo E y devuelven un objeto de tipo E. De esta manera, se opera el objeto que llame al método con el que se recibe por parámetro y se devuelve el resultado. Ambos, operandos y el resultado, son del mismo tipo.  **Ejercicio 2**  Implementa una pila utilizando como atributos un array genérico y un entero que cuente el número de objetos insertados. La clase se debe llamar PilaArray y tiene los mismos métodos que la pila del ejercicio anterior.  *Como puede hacerse*  La clase PilaArray tiene un parámetro genérico E.  Esta clase tiene como atributos un array del mismo tipo genérico E que la clase y un entero que sirve de contador de objetos. El constructor recibe por parámetro el tamaño máximo de la pila. El método estaVacia() comprueba si el contador es 0.  El método añadir() recibe por parámetro un objeto de tipo E, comprueba que hay espacio libre y, si es así, lo añade en la celda que indica el contador. Posteriormente incrementa el valor del contador. Si se ha añadido, devuelve true. Si no se ha podido añadir, devuelve false.  primero(): si está vacía, lanza NoSuchElementException. Si no está vacía, devuelve el elemento que está en el contador. El método se declara de tipo E, ya que los objetos que va a devolver son del tipo parámetro.  extraer(): si está vacía, lanza NoSuchElementException. Si no está vacía, decrementa el contador y devuelve el elemento que está en la celda indicada por el contador después de decrementarse. Extraer se declara también de tipo E. Recuerda que NoSuchElementException hereda de RunTimeException, por lo que no se declara en la cláusula throws.  Se sobrescribe el método toString()  **Ejercicio 3**  En una aplicación de gestión de productos, se desea implementar una clase Almacen que pueda manejar diferentes tipos de productos. Cada producto tiene un nombre y un precio.  Define una clase Producto con los siguientes atributos:  nombre (String): representa el nombre del producto.  precio (double): representa el precio del producto.  Define una clase Almacen que pueda almacenar productos. Esta clase debe tener un método para añadir un producto al almacén y otro método para imprimir los productos almacenados.  Implementa el método imprimirProductos de la clase Almacen utilizando un comodín <? extends Producto> para permitir imprimir una lista de productos sin importar el tipo específico.  Crea una clase Main que pruebe el funcionamiento de la clase Almacen. En esta clase, crea una instancia de Almacen, agrega varios productos de diferentes tipos (por ejemplo, alimentos, electrónicos, etc.) y luego imprime los productos almacenados.  **Ejercicio 4**  Se desea implementar una clase Almacen que pueda almacenar elementos de cualquier tipo que sean superclases de Producto. Esta clase debe tener un método para añadir elementos al almacén y otro método para imprimir los elementos almacenados.  Implementa el método imprimirElementos de la clase Almacen utilizando un comodín <? super T> para permitir imprimir una lista de elementos sin importar el tipo específico. | |
| Desarrollo:  Crear un proyecto que contenga un paquete por cada ejercicio a realizar.  Subir todo al repositorio github | |
| **Aula Invertida y recursos:**  En esta sección, se presenta el aula invertida sobre una clase de Persistencia en Sprint. Se propone el estudio previo de la siguiente lista de reproducción presentada en el aula virtual. La misma comprende la creación de un proyecto y de entidades explicando todas las anotaciones de JPA y sus relaciones.  **Actividad Asíncrona**  **Comenzar con el estudio de. ( Aula Invertida)**  De la presente lista de reproducción presentada como actividad en el aula virtual de la cátedra: | |
| **Conclusiones:**  Los estudiantes deben resumir lo que aprendieron durante la realización del trabajo práctico y destacar cómo la implementación de la lógica de creación de objetos y el mapeo de relaciones confirmaron que el modelo funciona correctamente en función de su descripción. Deben discutir cualquier problema encontrado y cómo lo resolvieron. | |
| Forma de Presentación | El proyecto se presentar con un link de un repositorio de proyecto en github.  Se debe crear en el repo un archivo readme con las características de ejecución del mismo.  El link del repo se subirá al aula virtual. |
| Tipo de Evaluación | Sumativa - Individual |
| **Modalidad de Evaluación** | |  |  |  |  | | --- | --- | --- | --- | |  | **>90%** | **90% y 60%** | **<60 %** | | **Clases** |  |  |  | | **Atributos** |  |  |  | | **Relaciones** |  |  |  | | **Cardinalidades** |  |  |  | | **RESULTADOS** | **10** | **6** | **4** |   **Se establece una sumatoria de cada ítem y se determina el porcentaje individual de cada uno. Posteriormente se establece la sumatoria de todos los ítems y se lo divide por la cantidad de los mismos para determinar en cuál de los valores de la escala se encuadra.**  **Para aprobar el alumno debe obtener una nota igual o superior a 6.**  **Para aprobar el estudiante debe obtener una nota igual o superior a 6.** |
| Bibliografía | Listas de reproducción de videos de la cátedra y material teórico del aula virtual |
| Profesores responsables del TP. | |  |  |  |  | | --- | --- | --- | --- | | Apellido y nombre | | Apellido y nombre | | |  | |  | | |  |  | |  | |  |  | |

**Trabajo Práctico:** Clases Genéricas - **Grupo DevHype**

Conclusiones:

Durante la realización de los ejercicios prácticos sobre clases genéricas, hemos aprendido cómo utilizar este poderoso concepto de la programación orientada a objetos para crear estructuras y métodos flexibles y reutilizables que pueden funcionar con diferentes tipos de datos. A través de la implementación de interfaces genéricas y el uso de comodines, pudimos diseñar soluciones que permiten una mayor flexibilidad en el manejo de datos.

Al enfrentarnos a cada ejercicio, confirmamos la eficacia del modelo genérico al observar cómo nuestras implementaciones se ajustaban a la descripción y a los requisitos establecidos en cada caso. En particular, la capacidad de crear una calculadora genérica capaz de operar con diferentes tipos numéricos y un almacén capaz de almacenar y manipular elementos de cualquier tipo que sea una superclase de Producto, demostraron la versatilidad y utilidad de las clases genéricas en la práctica.

En el transcurso de estos ejercicios, también encontramos desafíos, como la gestión de excepciones y la correcta manipulación de las listas de objetos genéricos. Sin embargo, gracias a la comprensión de los conceptos subyacentes y la aplicación de técnicas adecuadas, pudimos resolver estos problemas y crear soluciones funcionales y sólidas.

En resumen, los ejercicios prácticos sobre clases genéricas nos han brindado una comprensión más profunda de este concepto fundamental en Java, mostrándonos cómo aplicarlo de manera efectiva para crear código más flexible, reutilizable y robusto.