UTN AVELLANEDA
Tecnicatura Universitaria en Programación

Materia:	Programación II		
Nivel:	2º Cuatrimestre		
Tipo de Examen:	Final		
Apellido(1):		Fecha:	
Nombre/s(1):		Docente a cargo(2):	
División(1):		Nota(2):	
DNI(1):		Firma(2):	

(1) Campos a completar solo por el estudiante en caso de imprimir este enunciado en papel.
(2) Campos a completar solo por el docente en caso de imprimir este enunciado en papel.

Objetivo:
Desarrollar un sistema en Java que permita gestionar entidades relacionadas a un contexto definido por el alumno, siguiendo los principios de la Programación Orientada a Objetos y utilizando los conocimientos adquiridos durante la cursada.
El proyecto debe incluir las siguientes características técnicas y funcionales.

Requisitos del Proyecto

1. **Contexto del Proyecto**
   Se podrá elegir un contexto temático para el sistema, el cual debe estar relacionado con una de las siguientes áreas:
   - Gestión de productos (negocio, inventario, stock, etc.)
   - Gestión de personas (empleados, estudiantes, clientes, etc.)
   - Gestión de vehículos (autos, barcos, aviones, etc.)
   - Gestión de entidades creativas (naves, monstruos, aliens, personajes, etc.)

Nota: *La jerarquía de clases elegida debe poseer como mínimo dos niveles con una clase base y al menos tres clases derivadas. Ejemplo: Si se eligen "vehículos", podrían implementar una clase abstracta Vehiculo con derivadas como Auto, Camion y Moto.*

2. **Clases y Herencia**
   - Implementar una **clase abstracta base** que defina, al menos tres atributos y tres comportamientos comunes a todas las entidades del sistema.

- Crear al menos **tres clases derivadas** de esta clase base, que contengan cómo mínimo, dos atributos.
- Todas las clases deben contar con su respectivo constructor sobrecargado para:
  - Recibir tantos parámetros como atributos tenga la clase.
  - Otra sobrecarga que reciba un parámetros menos que el anterior.
  - Otra sobrecarga a su elección.
- Algunas de las clases derivadas deben implementar una **interfaz con al menos un método abstracto**.
- Utilización de enumerados. Utilizarlos cómo atributos y cómo parámetros de métodos.

3. **CRUD (Crear, Leer, Actualizar, Eliminar)**
   - Implementar una **clase de gestión** que contenga una lista genérica para administrar las entidades del sistema.
   - Esta clase debe implementar una **interfaz genérica** que incluya métodos para realizar las operaciones del CRUD.
   - Proveer la funcionalidad para iterar sobre la lista utilizando un **Iterator** personalizado.

4. **Ordenamiento y Filtrado**
   - Las entidades deben ser ordenables por un criterio **natural** (usando *Comparable*) y por al menos **dos criterios diferentes** (usando *Comparator*).
   - Implementar una funcionalidad para **filtrar** la lista utilizando **wildcards** en los métodos.

5. **Modificaciones con Interfaces Funcionales**
   - Aplicar cambios a la lista mediante el uso de **interfaces funcionales** (*Consumer*, *Function*, etc.).
     Ejemplo: Incrementar un atributo numérico (como precio o edad) o modificar un estado.

6. **Persistencia de Datos**
   - Implementar la funcionalidad para **serializar y deserializar** la lista de entidades a un archivo.
   - Implementar la opción para **guardar y recuperar** los datos en formato **CSV** y **JSON**.
   - Implementar una funcionalidad para **exportar un listado filtrado a un archivo de texto (.txt)**. Este archivo debe:

- Incluir un encabezado descriptivo.
- Ser legible por un ser humano (organizado para su impresión o lectura directa).

7. **Interfaz Gráfica**
   - Crear una interfaz gráfica básica utilizando **JavaFX** que permita:
     - Agregar, listar, actualizar y eliminar entidades.
     - Ordenar y filtrar las entidades desde la interfaz gráfica.
     - Mostrar los resultados de las modificaciones realizadas con interfaces funcionales.
     - Guardar, cargar y exportar los datos desde la interfaz gráfica.

8. **Uso de Wildcards**
   - Incorporar al menos un método en el sistema que utilice **wildcards** con **límites superiores** (**? extends**) y otro con **límites inferiores** (**? super**).

9. **Manejo de excepciones**
   - Realizar al menos dos excepciones propias.

10. **Diagrama UML**
    - Presentar un **diagrama de clases UML** que represente la estructura del proyecto, incluyendo clases, interfaces, relaciones de herencia e implementación, y métodos clave.

Nota: *El alumno deberá investigar aquellos temas que se soliciten en el proyecto final y que no se hayan visto en su cursada.*

Instancia Oral: Defensa del Proyecto.

Se deberá realizar la defensa del proyecto en una instancia oral, abordando los siguientes puntos:

1. **Justificación del Diseño**
    - Explicar la elección del contexto y la estructura de las clases.
    - Detallar cómo se implementaron el ordenamiento, filtrado, modificaciones, persistencia e iteración.

2. **Explicación Técnica**
    - Demostrar el uso correcto de **Iterator**, **wildcards**, **interfaces funcionales**, **serialización**, **archivos CSV, archivos JSON** y **JavaFX**.
    - Explicar el flujo de trabajo entre la interfaz gráfica, la lógica del sistema y la persistencia de datos.

3. **Preguntas Técnicas**
    - Se evaluará el entendimiento de conceptos fundamentales como herencia, interfaces, polimorfismo, uso de genéricos y persistencia de datos, entre otros.

Documentación: Entrega.

Se deberá crear un repositorio en Github, nombrado cómo ***Apellido.Nombre.Final.Java.2024***.

Completar el archivo **README.md** que se encuentra en dicho repositorio con la siguiente información:

- **Título**: Ponerle un nombre a la aplicación. *Ejemplo: CRUD - Personas*
- **Sobre mí**: Presentarse brevemente.
- **Resumen**: Explicar qué hace la aplicación y cómo se usa a grandes rasgos. Esto debe incluir: capturas de pantalla o demostración de la interfaz gráfica.
- **Diagrama de clases UML**: Adjuntar una imagen con el diagrama UML del proyecto.
- **Archivos generados** (.dat, .csv, .json, .txt) con ejemplos prácticos de cada funcionalidad.

Nota: *Para trabajar con este archivo se deberá utilizar el lenguaje de marcado Markdown.*

Criterios de Evaluación

- **Precisión funcional**: El sistema cumple los requisitos establecidos.
- **Diseño**: Uso adecuado de POO, genéricos, interfaces funcionales, wildcards, Iterator y persistencia.
- **Documentación**: Diagrama UML claro y detallado.
- **Código limpio y organizado**: Nombres descriptivos, comentarios y uso eficiente de los recursos de Java.
- **Interfaz gráfica funcional**.
- **Defensa**: Capacidad para justificar decisiones técnicas y resolver preguntas.