Programación orientada a objetos
Módulo: conceptos básicos de la programación orientada a objetos
Proyecto Directorio telefónico clínica veterinaria

Objetivos

Practicar los conceptos de clase, atributo, método y objeto.
Practicar el mecanismo de paso de mensajes entre objetos de una clase.
Entender la relación entre diagramas de clase UML y código
Practicar el uso de constructores
Practicar el uso de mapas y vectores
Practicar el modelado de relaciones de muchos a muchos con entidades intermedias.

Enunciado

Usted ha sido contratado para crear un software de directorio telefónico para una compañía veterinaria del país. El área de tecnología de la organización ya tiene definido el diseño del sistema de información. Su labor principal es implementarlo para satisfacer los siguientes requisitos:

- El sistema almacenar el nombre completo, email, documento de identidad y teléfono de cada propietario responsable.
- El sistema debe almacenar la información de cada mascota. Esta información consiste en la raza, el tipo: perro | gato | otro, el peso, la edad, tipo de sangre, el nombre, la identificación, status: viva muerta. Si la mascota ha fallecido también registra la fecha de defunción.
- Una persona puede tener varias mascotas y cada mascota puede tener varios propietarios responsables.
- El sistema debe permitir agregar/modificar/eliminar y listar el contacto de los propietarios responsables. El número de identificación es el campo que el sistema utiliza para buscar a las personas antes de modificar o eliminar.
- El sistema debe mostrar cuántos propietarios tiene registrados el directorio.
- El sistema debe mostrar la información de los propietarios registrados el directorio.
- El sistema debe mostrar la información de las mascotas registradas el directo
- El sistema debe permitir consultar para una mascota dada cuáles son sus propietarios responsables
- El sistema debe permitir consultar para un propietario dado cuales son las mascotas relacionadas.
- El sistema debe permitir asociar nuevas mascotas a un propietario.
- El sistema debe permitir asociar nuevos propietarios a una mascota.
- El sistema debe permitir cambiar el status de una mascota: de viva a muerta.
- El sistema debe permitir eliminar un propietario de una mascota

Reglas de negocio

- Cada propietario debe estar registrado solo una vez. No pueden existir dos propietarios con el mismo número de identidad.
- Cada mascota debe estar registrada una sola vez. No pueden existir dos mascotas con el mismo número de identidad.

Requisitos no funcionales

- Debe usar C++ para el desarrollo del proyecto
    - Use Maps como la estructura de datos para almacenar la información de los propietarios

Mínimos esperados

Todos los entregables deben ser subidos a un repositorio de git en una carpeta llamada ***VETERIANA+iniciales***. En Brightspace uno de los miembros del equipo sube la URL de repositorio donde quedaron subidos los archivos. Puede subir la información a su repositorio máximo hasta las 11:55 pm del 7 de marzo del 2022. Si prefiere dar la nota de su compañero de manera anónima puede comunicarse conmigo. ***Los documentos deben tener una calidad de redacción, ortografía y presentación esperadas a nivel universitario***. Las sustentaciones serán el 1 de 8 de manera grupal e individual en el horario de clase.

Durante la clase del 24 de febrero solucionaremos dudas asociadas al proyecto. También puede contactarme en otros momentos para solucionar estas dudas.

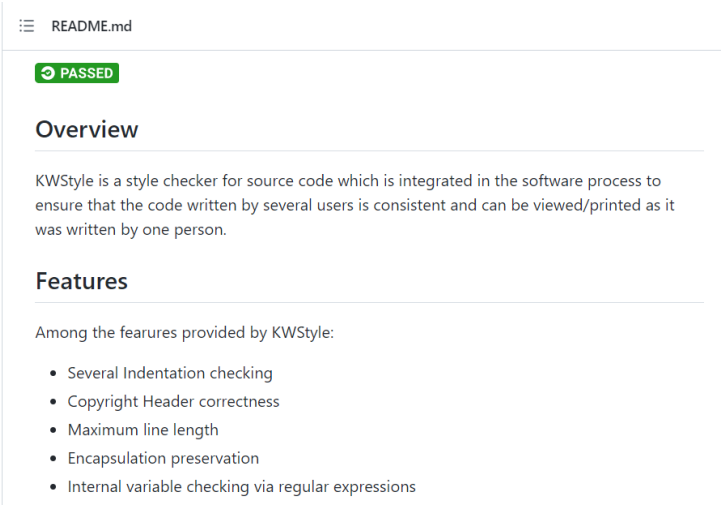
sperados (no cuentan en la nota, pero sin esto no califico el proyecto):

En la elaboración de su programa debe considerar:

- Uso continuo de git para mantener el histórico de avance de su proyecto con comentarios claros sobre los cambios de su programa.
- Tener un menú usando do while y switch case para acceder a las diferentes opciones del programa.
- Tener un makefile para compilar el programa.
- Cumplir con el estándar de codificación lowerCamelCase, en el que las operaciones inician con un verbo en infinitivo. Las palabras compuestas inician en minúsculas y la segunda palabra tiene la primera letra en mayúsculas Ejm: *consultarPropietarios.* Las variables tienen nombres que semánticamente se relacionan con la función que cumplen. El cumplimiento de este estándar es obligatorio.
- Buena indentación y organización del código.
- Código documentado para hacerlo claro para cualquier lector. En la documentación recuerde que sus comentarios deben explicar por qué se hacen las cosas más que solo describir literamente las líneas de código.
- Buenas prácticas de programación: buen nombramiento, no números mágicos, uso de constantes.

Entregables

- **Código fuente del programa.**
- **Informe de autoevaluación:** documento en el que explique qué problemas tuvo al hacer su proyecto, qué aprendió, qué le gustó, que no le gustó, qué hizo cada uno de los miembros del equipo, qué nota se pondrían de manera individual y qué nota le pondrían al compañero junto con la justificación. Cada persona del equipo debe entregar un informe individual.
- **README en el repositorio [Manual técnico]:** Con presentación general del proyecto, principales funcionales, y explicación de las principales funcionalidades de la aplicación: métodos, entradas y salidas y el diagrama UML (uno por equipo). La siguiente figura muestra una imagen de ejemplo de un README disponible en https://github.com/Kitware/KWStyle. Tenga en cuenta que para documentar el README en Github debe usar MarkDown (aquí puede encontrar más info https://www.markdownguide.org).


README.md

PASSED

Overview

KWStyle is a style checker for source code which is integrated in the software process to ensure that the code written by several users is consistent and can be viewed/printed as it was written by one person.

Features

Among the fearures provided by KWStyle:

- Several Indentation checking
- Copyright Header correctness
- Maximum line length
- Encapsulation preservation
- Internal variable checking via regular expressions

Todos los entregables deben ser subidos a un repositorio de git en una carpeta llamada ***VETERINARIA+iniciales***. En Brightspace uno de los miembros del equipo sube la URL de repositorio donde quedaron subidos los archivos. Puede subir la información a su repositorio máximo hasta las 11:55 pm del 5 de marzo del 2022. Si prefiere dar la nota de su compañero de manera anónima puede comunicarse conmigo. ***Los documentos deben tener una calidad de redacción , ortografía y presentación esperadas a nivel universitario***. Las sustentaciones serán el 8 y 10 de marzo de manera grupal e individual en el horario de clase.
En las clases solucionaremos dudas sobre el proyecto y también puede contactarme en otros momentos para solución de dudas.

Calificación

- Autoevaluación: 10 %
- Calificación del compañero: 10%. En trabajos individuales la autoevaluación tendrá un 20% de peso.
- Calificación de la profesora compuesta por entregables, diseño, funcionalidad, estilo de codificación y mejores prácticas: 80%. La rubríca de evaluación explica los elementos que se consideran para la calificación del proyecto.
- Propiedad intelectual: valor entre 0 y 1 que multiplica la calificación total. Se demuestra durante la sustentación.
- Nota final = (criterios evaluacion%) * propiedadIntelectual

Rúbrica de evaluación

La nota de la profesora será dada al finalizar la sustentación según los criterios que se describen a continuación y su capacidad para sustentar su trabajo.

	5	4	3	2	1	0
Entregables (10%)	Los informes contienen toda la información solicitada y tiene alta calidad en	Se entregaron todos los informes. En términos de contenido están completos pero	Se entregaron todos los informes. En términos de contenido están completos pero	Faltan algunos de los informes pero los entregados tiene buen estilo y formato	Faltan algunos de los informes y los entregados necesitan mejoras de estilo y formato	No se entregaron los informes

	cuanto a estilo y formato.	podría mejorar en cuanto a estilo o formato	podría mejorar en cuanto a estilo Y formato			
Diseño (30%)	El diseño responde a los requisitos. Se detectaron todas las clases importantes y para cada clase se detectaron los atributos y métodos importantes. Usa las relaciones correctas	El diseño responde a los requisitos. Se detectaron todas las clases importantes y para cada clase se detectaron los atributos y métodos importantes. Tiene algunas relaciones incorrectas o faltantes en el diseño de los métodos como por ejemplo los atributos.	El diseño responde a los requisitos. Faltó detectar algunas clases importantes. Faltó detectar algunos de los atributos y métodos importantes. Tiene algunas relaciones incorrectas. El diseño no corresponde a lo codificado.	Faltó detectar la mayoría de clases importantes Faltó detectar muchos de los atributos y métodos importantes. Tiene muchas relaciones incorrectas	El diseño no satisface los requisitos	No se entregó el diseño
Funcionalidad (30%)	Cumplió con todos los requisitos.	Fueron desarrollados mínimo el 75% de los requisitos	El diseño responde al 75% de los requisitos / podría mejorarse	Fueron desarrollados mínimo el 25% de los requisitos	Fueron desarrollados menos del 25% de los requisitos	La funcionalidad no responde a lo solicitado
Estilo de codificación (5%)	El código se encuentra correctamente indentado, los nombres de los atributos y las funciones cumplen con el estándar de nombramiento. El código tiene documentación interna para facilitar la revisión.	La mayoría del código se encuentra correctamente indentado- La mayoría de los nombres de los atributos y las funciones cumplen con el estándar de nombramiento. La mayoría del código tiene documentación interna para facilitar la revisión	Falta alguno de los ítems de calidad del estilo de codificación o alguna se cumple con mala calidad	Faltan dos de los ítems de calidad del estilo de codificación o dos se cumple con mala calidad	No hay código fuente suficiente para evaluar el estilo de codificación.	No cumple con el estándar de nombramiento No se encuentra correctamente indentado No está divido adecuadamente
Mejores prácticas (5%)	El código muestra mejores prácticas de desarrollo siempre. Reúso, separación de operaciones, buen manejo de ciclos, simplicidad, programación defensiva validaciones.	El código muestra mejores prácticas de desarrollo en la mayoría de los casos, pero falta mejorar algunos de los siguientes aspectos Reúso, separación de operaciones, buen manejo de	El código muestra buenas prácticas de desarrollo pero falta mejorar dos de los siguientes aspectos Reúso, separación de operaciones, buen manejo de	El código aplica pocas buenas prácticas de desarrollo. Falta mejorar tres de los siguientes aspectos: Reúso, separación de operaciones, buen manejo de ciclos, simplicidad,	No hay código fuente suficiente para evaluar las buenas prácticas.	No se entregó proyecto o el proyecto no cubre al menos el 25% de las funcionalidades. No es posible evaluarlo .

		ciclos, simplicidad, programación defensiva validaciones	ciclos, simplicidad, programación defensiva validaciones	programación defensiva validaciones.		

Rúbrica de propiedad intelectual

	1	0.8	0.6	0.4	0.2	0
Sustentación	Es evidente que el estudiante entiende el código que desarrolló lo explica con claridad y responde correctamente a las preguntas.	La sustentación es buena pero se evidenció inseguridad del estudiante para explicar algunas partes del trabajo desarrollado o para responder algunas preguntas.	La sustentación es aceptable se evidencia que el estudiante desarrolló el código pero le cuesta trabajo explicar aspectos del código.	La sustentación es regular se evidenció inseguridad del estudiante para explicar gran parte del trabajo desarrollado o para responder muchas de las preguntas. Parece que el código no hubiera sido desarrollado por el estudiante.	El estudiante demuestra que entiende partes del código, pero no tiene claro cómo se relacionan con la funcionalidad solicitada.	Se evidencia que el estudiante no entiende el código desarrollado, no es capaz de responder a las preguntas formuladas de manera correcta.