Laboratorio 4. Polimorfismo a través de Interfaces

Competencias a desarrollar:

- Identifica los requisitos funcionales del sistema a desarrollar a partir del problema planteado.
- Elabora el análisis y el diseño del programa produciendo un diagrama de clases en UML.
- Diseña la interfaz o interfaces que necesitará para generalizar su programa.

Problema a resolver:

La compañía Mercedes-Benz le ha contratado para implementar el *software* de los radios para la línea de vehículos que está a punto de lanzar. No todos los modelos tienen las mismas prestaciones, por lo que es posible que no tengan todas las opciones disponibles. Sin embargo, quieren unificar todas las opciones en una interfaz para que el radio sea intercambiable directamente en línea de producción y no sea necesario reprogramar el *software* de control.

Le han pedido que desarrolle un simulador de cómo debería funcionar un radio que implementa la interfaz, pero…. ¡no se ha diseñado esa interfaz! Siguiendo el principio de segregación de interfaces el equipo de desarrollo ha decidido hacer tres: una para carros clase S, otra para carros clase A y otra para carros clase C.

Funcionalidades esperadas de los radios:

- Encender/apagar
- Cambiar volumen (en intervalos de ±1)
- Modo Radio
    - Cambiar de FM a AM
    - Cambiar emisoras, se cambiará en intervalos de 0.5
    - Guardar emisoras. Se podrán guardar hasta 50 estaciones de radio.
    - Cargar emisora
- Modo Reproducción (sirve para CD, MP3, o Spotify)
    - Seleccionar lista de reproducción (se tienen configuradas algunas para prueba)
    - Cambiar canción (tanto para adelante como para atrás)
    - Escuchar canción (debe mostrar el nombre, la duración, el autor y el género de la canción que se está escuchando)
- Modo teléfono
    - Conectar/Desconectar teléfono
    - Mostrar contactos
    - Llamar a contacto
    - Finalizar llamada
    - Cambiar a bocinas o auriculares (Clase S)
    - Llamar al último contacto con el que se habló (Clase A)
    - Cambiar a llamada en espera (Clase C)
- Modo Productividad
    - Planificar Viajes (Clase S)
    - Ver tarjetas de presentación (Clase A)
    - Ver pronóstico del tiempo (Clase C)

- En todos los casos debe mostrarse en pantalla el modo del radio, con toda la información que le resulte de utilidad al usuario; por ejemplo:
    - Si está en modo radio, la banda (AM/FM) la frecuencia de radio y la estación. Si la frecuencia está guardada, la posición o número en la que está.
    - Si está en modo reproducción, la información de la canción que se está reproduciendo, etc.

Observaciones y excepciones:

- Si el radio está apagado no se debe permitir ningún cambio de estado, únicamente encender.
- Si el carro es de una clase en específico no debe mostrar las funcionalidades de otras clases.
- Muestre mensajes de error descriptivos cuando sea necesario.

Instrucciones:

Se deben conformar grupos de dos o tres personas. El grupo debe desarrollar en conjunto las interfaces que se implementarán y la jerarquía de clases abstractas y concretas de las que puedan depender cada miembro. Luego, las implementaciones específicas de cada clase de vehículo y del programa principal (Vista, Controlador) deben ser desarrolladas individualmente por diferentes miembros, apoyándose en la parte diseñada de manera grupal **sin hacerle modificaciones** (en caso una persona necesite hacer modificaciones al diseño realizado de forma grupal, debe discutirlo y realizarlo en acuerdo con el resto del grupo).

Realicen el análisis del problema planteado y elaboren el diagrama de clases (UML) del sistema. Diseñen la interfaz (o interfaces, en caso de que necesite más de una; tenga en cuenta el Principio de Segregación de Interfaces en su diseño) que se usará para presentar las diferentes formas de interacción con el usuario. Desarrollen las clases que modelan cada vehículo, así como el programa principal y cualquier otra clase necesaria que corresponda al modelo (en referencia al patrón MVC). Empleen un sistema de versionamiento de código para desarrollar su programa e identificar las partes que desarrolló cada persona del grupo.

Material a entregar y fechas de entrega:

- Archivo .pdf con el análisis del sistema
- Archivo de imagen (.png, .jpg) con el diagrama de clases diseñado
- Archivos .java con el sistema desarrollado.
- Enlace a repositorio con el código desarrollado y el historial de cambios.

Evaluación:

Análisis **(29 puntos)**:

- Identificación de Clases, atributos y métodos (**12 puntos**):
  - (**6 puntos**) Se identifican correctamente las clases que se necesitan para resolver el problema.
  - (**3 puntos**) Se identifica correctamente la interfaz o interfaces que deben diseñarse para resolver el problema planteado.
  - (**3 puntos**) Se explica correctamente y de forma lógica el propósito de cada una de las clases.
- Atributos de las clases (**10 puntos**):
  - (**7 puntos**) Se identifican correctamente todos los atributos de cada una de las clases.
  - (**3 puntos**) Se explica correctamente y de forma lógica el propósito de cada uno de los atributos.
- Métodos de las clases (**7 puntos**):
  - Se identifican correctamente los métodos necesarios para resolver la situación planteada. Se explica correctamente y de forma lógica el propósito de cada uno de los métodos de las clases.

Diseño **(23 puntos)**:

- Diagrama de Clases en UML
  - (**5 puntos**) Se representa correctamente la jerarquía de clases y/o interfaces identificadas, utilizando las relaciones correctas.
  - (**4 puntos**) Se representan correctamente el resto de las relaciones entre las clases identificadas.
  - (**5 puntos**) Todos los atributos y métodos tienen correctamente representada la visibilidad que poseen.
  - (**4 puntos**) Los métodos tienen todos los parámetros y valor de retorno correctamente representados.
  - (**5 puntos**) El diseño sigue el patrón MVC y respeta los principios de OO.

Implementación **(48 puntos)**

  - (**20 puntos**) Están correctamente implementados los requisitos funcionales identificados. Se da solución con ellos al problema planteado.
  - (**5 puntos**) El sistema es completamente reactivo a entradas incorrectas, mostrando errores amigables con el usuario.
  - (**20 puntos**) Está correctamente implementado el polimorfismo basado en interfaces. El código sigue principios y buenas prácticas de OO.
  - (**3 puntos)** Está totalmente documentado usando el estándar JavaDoc.

Para que sea calificada la implementación deben cumplirse los siguientes requisitos:

- El sistema no tiene errores de sintaxis ni en tiempo de compilación que impidan la revisión del mismo.
- La implementación debe coincidir con el diseño. Los cambios que surjan deben ser menores y deben ser presentados claramente en un re-diseño entregado con el código. Los cambios también deben estar documentados en el código con comentarios y justificaciones.