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 utilizando un diagrama de clases en UML.
- Diseña la interface que necesitará para generalizar su programa.

Problema a resolver:

La compañía Mercedes-Benz lo ha contratado para implementar el software de sus radios para la línea de vehículos que está a punto de lanzar. Todos los modelos no 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 interface para que el radio pueda ser intercambiable directamente en línea de producción y no sea necesario reprogramar el software de control.

Le ha pedido que haga un simulador de como debería funcionar un radio utilizando la interface, pero…. ¡No se ha diseñado esa interface! Siguiendo el principio de segregación de interfaces el equipo de desarrollo ha decidido hacer 3, una para un carro clase A, otra para un carro clase B y otra para un carro clase C.

Funcionalidades esperadas de los radios:

Funcionalidades:

- 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 speaker o auriculares (ClaseA)
    - Llamar al último contacto con el que se habló (Clase B)
    - Cambiar a llamada en espera (Clase C)
- Modo Productividad

  - Planificar Viajes (Clase A)
  - Ver tarjetas de presentación (Clase B)
  - Ver pronóstico del tiempo (Clase C)
- En todos los casos debe mostrarse en pantalla el estado 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 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.

Tareas:

Tarea #1. Análisis: Realice el análisis del problema planteado. Tenga en cuenta que parte de su programa va a ser utilizado por otro compañero de clases, por lo que debe hacerlo lo más general y claro posible.

Tarea #2. Diseño: Elabore el diagrama de clases (UML) del sistema.

Tarea #3. Interface: Diseñe la interface (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 intercambiar la GUI, o la clase que interactúa con el usuario.

Tarea #4. Intercambio: Intercambie su clase de interacción con el usuario con otro compañero. A la clase que reciba solo puede cambiarle **una línea de código** para que funcione con su programa.

Tarea #5. Reflexión: Describa el proceso de diseño que tuvo que hacer en conjunto con el resto del grupo para poder cambiar solo una línea de código con el otro compañero. Describa los tropiezos que tuvo y los cambios que tuvo que hacer para poder usar la clase intercambiada con su sistema.

Instrucciones:

Serán divididos de forma aleatoria en grupos. Los grupos que hacen el simulador para cada clase de vehículo (A,B o C) deben ponerse de acuerdo en cuanto a los métodos a poner en la interface/interfaces. Cada grupo debe hacer su versión del programa y probar intercambiar la clase de interfaz de usuario con otro de los grupos que implementan la misma clase de vehículo (A,B,C). Los auxiliares intercambiarán la clase con cualquiera de un grupo que implemente la misma clase de vehículo y todo tiene que funcionar cambiando solo una línea de código. En la tabla siguiente se puede observar la clase de vehículo que va a implementar cada Grupo.

Grupo	Clase de Vehículo
1	A
2	B
3	C
4	A

5	B
6	C
7	A
8	B
9	C
10	A
11	B
12	C

Nota: Cada integrante del grupo tendrá nota individual basado en sus contribuciones al repositorio de github.

Material a entregar y fechas de entrega:

- **Primer día de clases a las 23:59**.
  - Archivo .pdf con el análisis del sistema
  - Archivo de imagen (.png, .jpg) con el diagrama de clases diseñado
  - Archivos .java con la interface que pactaron en su grupo implementada
  - link del github que utilizaron para construir el ejercicio
- **Segundo día de clase a las 23:59**.
  - Archivo .pdf con la descripción de los cambios que hay que realizarle al sistema para que funcione con otra interfaz
  - Archivos .java con el sistema incluyendo la clase de interacción del usuario del compañero con el que probó intercambiar, asegúrese de identificar correctamente que no es una clase propia sino de otra persona.
  - link del github que utilizaron para construir el ejercicio

Evaluación:

Requisitos Funcionales **(5 puntos)**:
- Identifica correctamente todo lo que debe hacer el programa según la situación planteada.

Análisis **(24 puntos)**:
- Identificación de Clases, atributos y métodos (**9 puntos**):
  - (**3 puntos**) Se identifican correctamente las clases que se necesitan para resolver el problema. El número de clases identificadas es suficiente para darle solución a la situación planteada.
  - (**3 puntos**) Se identifica correctamente la interface que debe 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**):
  - (**3 puntos**) Se identifican correctamente todos los atributos de cada una de las clases seleccionadas. Son los necesarios para resolver el problema planteado.
  - (**4 puntos**) Se identifican correctamente los atributos y/o variables polimórficos.
  - (**3 puntos**) Se explica correctamente y de forma lógica el propósito de cada uno de los atributos.
- Métodos de las clases (**5 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 **(18 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.

Implementación **(53 puntos)**

- (**20 puntos**) Están correctamente implementados los requisitos funcionales identificados. Se da solución con ellos al problema planteado.
- (**10 puntos**) El sistema es completamente reactivo a entradas incorrectas, mostrando errores amigables con el usuario.
- (**23 puntos**) Está correctamente implementado el polimorfismo basado en interfaces. Se logró cumplir el objetivo de intercambiar con éxito la clase de interacción con el usuario cambiando solo una línea de código.
- (**5 puntos)** Está totalmente documentado usando el estándar JavaDoc.

Nota: 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 en un 95% con el diseño. Los cambios que surjan deben ser mínimos y deben estar documentados en el código.