Programación Orientada a Objetos 1

Práctica Calificada 3

Pregrado

2022-I

Profesor: Jonathan Silva

Lab: 2.05

Indicaciones específicas:

- Esta evaluación contiene 6 páginas (incluyendo esta página) con 1 pregunta. El total de puntos son 20.
- El tiempo límite para la evaluación es 100 minutos.
- El código debe consistir de un **main.cpp** y archivos de encabezado **.h**. Si lo requiere, tambien archivos **.cpp** adicionales
- Deberá subir estos archivos directamente a www.gradescope.com, por separado o en un **.zip**

Competencias:

- Para los alumnos de la carrera de Ciencia de la Computación

  Aplicar conocimientos de computación apropiados para la solución de problemas definidos y sus requerimientos en la disciplina del programa. (nivel 2)

  Diseñar, implementar y evaluar soluciones a problemas complejos de computación.(nivel 2)

  Crear, seleccionar, adaptar y aplicar técnicas, recursos y herramientas modernas para la práctica de la computación y comprende sus limitaciones. (nivel 2)

- Para los alumnos de las carreras de Ingeniería

  Aplicar conocimientos de ingeniería en la solución de problemas complejos de ingeniería (nivel 2).

  Diseñar soluciones relacionados a problemas complejos de ingeniería (nivel 2)

  Crear, seleccionar y utilizar técnicas, habilidades, recursos y herramientas modernas de la ingeniería y las tecnologías de la información, incluyendo la predicción y el modelamiento, con la comprensión de sus limitaciones (nivel 2)

- Para los alumnos de Administración y Negocios Digitales

  Analizar información verbal y/o lógica proveniente de distintas fuentes, encontrando relaciones y presentándola de manera clara y concisa (nivel 2)

  Analizar y evaluar el comportamiento del consumidor y el desarrollo de estrategias comerciales (nivel 2)

  Trabajar de manera efectiva con equipos multidisciplinarios y diversos en género, nacionalidad, edad, etc. (nivel 2)

Calificación:

Tabla de puntos (sólo para uso del professor)

Question	Points	Score
1	20	
Total:	20	

1. (20 points) **App de Movilidad:**

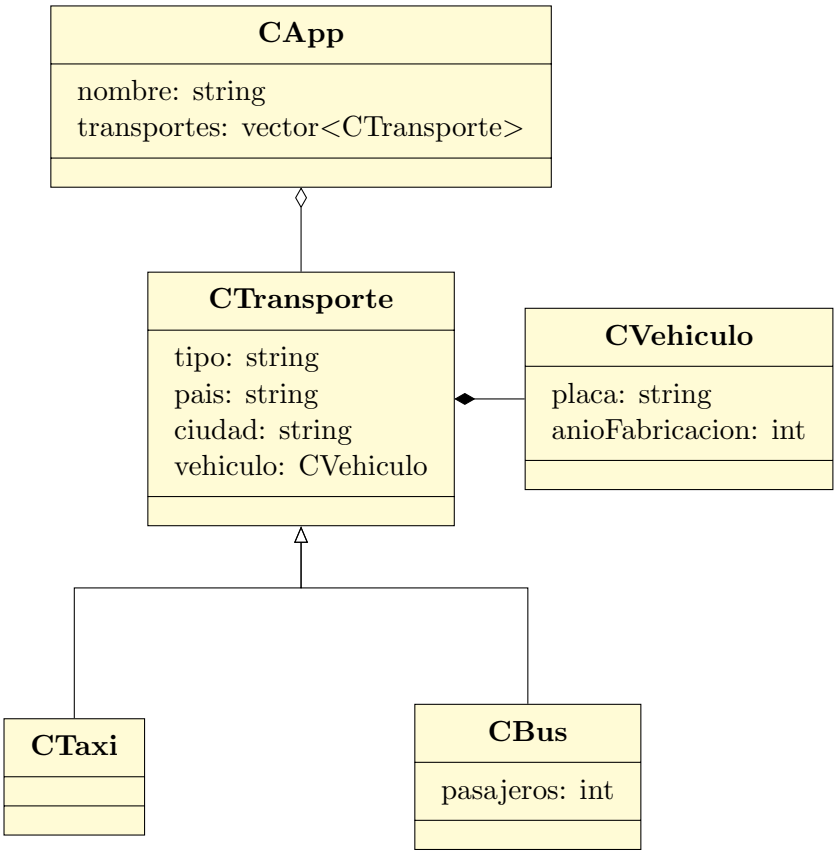
   Tema a evaluar: Relaciones entre clases, Herencia - polimorfismo, y sobrecarga de operadores

   Una app de movilidad permite registrar dos tipos de transportes (Taxi y Bus) en distintas ciudades, y requiere conocer cuantos vehiculos tiene en cada ciudad. Para eso se le solicita desarrollar las clases necesarias para almacenar los datos de los vehiculos para dos tipos de transporte y luego consultar los datos de los vehiculos por ciudad.

   Su programa se debe desarrollar utilizando el paradigma de **Programación Orientado a Objetos**. Su implementación debe implementar al menos las siguiente clases:

   - Cree una super clase Transporte que implemente herencia a las sub clases Taxi y Bus. Los objetos transporte deben contener el tipo de transporte (Taxi o Bus), el pais y la ciudad. Además, los buses tienen un dato especifico sobre la cantidad de pasajeros que pueden llevar.
   - Cree una clase App que almacene el nombre de la App y un vector polimorfico con objetos de la clase Taxi y Bus.
   - Cree una clase Vehiculo que tenga una relación de composición con la clase Transporte, y almacene la placa y año de fabricación.

   Un diagrama de las relaciones y atributos recomendados se muestra a continuación.



Además, su programa debe implementar la sobrecarga de operadores "<<" y "()":

- Agregar un transporte al vector de transportes con el operador "<<". Por ejemplo: para agregar un objeto taxi al vector polimorfico del objeto App:

Listing 1: Ejemplo de operador <<

```
CApp objApp();
CTaxi objTaxi();
objApp << objTaxi;
```

- Imprimir los vehículos para una ciudad específica con el operador "()". Por ejemplo: para imprimir la lista de los vehículos para la ciudad de Lima:

Listing 2: Ejemplo de operador ()

```
objApp("Lima");
```

Los criterios para la evaluación son los siguientes:

1. Implementar las relaciones entre las clases: Composición y Agregación **(7 pts)**.
2. Implementar la relación de Herencia entre clases para conseguir almacenar los distintos objetos Taxi y Bus en un vector usando polimorfismo **(7 pts)**.
3. Implementar la sobrecarga de dos operadores: "<<" y "( )" **(6 pts)**.

A continuación se muestra algunos ejemplos de la ejecución correcta del código:

Listing 3: Ejemplo de resultado 1

```
Ingresar nombre de la App:Flash
Cantidad de transportes:6
Ingrese los 6 transportes:
Taxi Peru Lima AAA111 2013
Taxi Colombia Bogota RRR444 2017
Taxi Peru Lima CCC555 2017
Taxi Colombia Bogota III666 2013
Bus Peru Lima QWE938 2013 35
Bus Colombia Bogota TYU485 2017 40

Ingrese la ciudad a buscar: Bogota

Vehiculo [1] Tipo: Taxi, Pais: Colombia, Ciudad: Bogota,
   Placa: RRR444, Anio: 2017
Vehiculo [2] Tipo: Taxi, Pais: Colombia, Ciudad: Bogota,
   Placa: III666, Anio: 2013
Vehiculo [3] Tipo: Bus, Pais: Colombia, Ciudad: Bogota,
   Placa: TYU485, Anio: 2017, Pasajeros: 40
La app Flash tiene 3 transportes que ofrecer
```

Listing 4: Ejemplo de resultado 2

```
Ingresar nombre de la App:Flash
Cantidad de transportes:8
Ingrese los 8 transportes:
Taxi Peru Lima AAA111 2013
Taxi Colombia Bogota RRR444 2017
Taxi Peru Lima BBB999 2018
Taxi Peru Lima CCC555 2017
Taxi Colombia Bogota III666 2013
Bus Peru Lima QWE938 2013 35
Bus Colombia Bogota TYU485 2017 40
Bus Peru Lima WER120 2018 30

Ingrese la ciudad a buscar: Lima

Vehiculo [1] Tipo: Taxi, Pais: Peru, Ciudad: Lima, Placa:
   AAA111, Anio: 2013
Vehiculo [2] Tipo: Taxi, Pais: Peru, Ciudad: Lima, Placa:
   BBB999, Anio: 2018
Vehiculo [3] Tipo: Taxi, Pais: Peru, Ciudad: Lima, Placa:
   CCC555, Anio: 2017
Vehiculo [4] Tipo: Bus, Pais: Peru, Ciudad: Lima, Placa:
   QWE938, Anio: 2013, Pasajeros: 35
Vehiculo [5] Tipo: Bus, Pais: Peru, Ciudad: Lima, Placa:
   WER120, Anio: 2018, Pasajeros: 30
La app Flash tiene 5 transportes que ofrecer
```

Los criterios en la rúbrica (y el puntaje respectivo) se condicionan a que la solución presentada corresponda al problema planteado. El 100% corresponde al puntaje indicado en cada punto

Criterio	Excelente	Adecuado	Mínimo	Insuficiente
Ejecución	El diseño del algoritmo es ordenado y claro, siguiendo buenas prácticas en programación. La ejecución es correcta (100%)	El diseño del algoritmo es ordenado y claro. La ejecución es correcta (70%)	El diseño tiene algunas deficiencias pero la ejecución es correcta (30%).	El diseño es deficiente y la ejecución no es correcta (0%)
Sintaxis	No existen errores sintácticos o de compilación (100%)	Existen algunos errores sintácticos de menor relevancia, que no afectan el resultado (50%).	Existen errores sintácticos en la forma de ejecución, que no afectan el resultado (30%).	El código tiene errores de sintáxis que afectan el resultado (10%)
Optimizacion	El código es óptimo y eficiente. De buen performance e interacción con el usuario (100%)	El código es de buen performance durante la ejecución (70%)	El código no está optimizado pero la ejecución no es deficiente(30%)	El código no está optimizado y la ejecución es deficiente (0%)