Programación Orientada a Objetos 1

Práctica Calificada 3

Pregrado

2022-I

Profesor: Jonathan Silva

Lab: 1.04

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) **Ranking en servicio de delivery:**

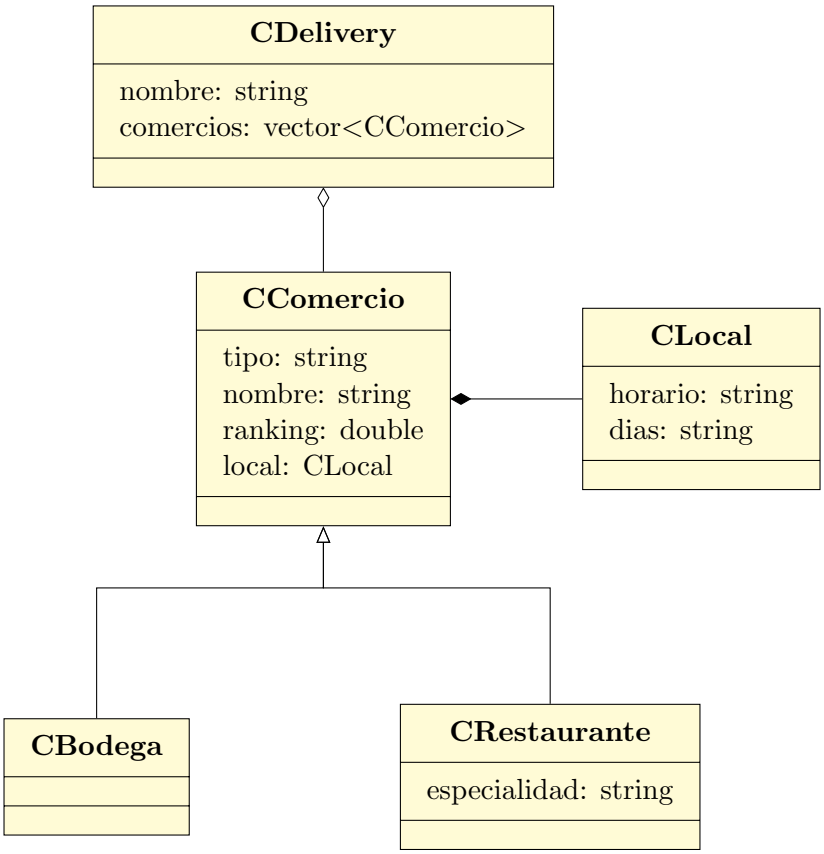
   Tema a evaluar: Relaciones entre clases, Herencia - polimorfismo, y sobrecarga de operadores

   Una app que ofrece servicio de delivery permite registrar dos tipos de comercios y mostrar su ranking (calificación de usuarios), y requiere conocer cuantos comercios tiene por ranking. Para eso se le solicita desarrollar las clases necesarias para almacenar los datos de los locales para dos tipos de comercio y luego consultar los datos de los locales por ranking.

   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 Comercio que implemente herencia a las sub clases Bodega y Restaurante. Los objetos comercio deben contener el tipo de comercio (B: Bodega, o R: Restaurante), el nombre y el ranking (números entre 1 y 5). Además, los restaurantes tienen un dato específico sobre la especialidad de la comida que preparan.
   - Cree una clase Delivery que almacene el nombre de la App y un vector polimorfico con objetos de la clase Bodega y Restaurante.
   - Cree una clase Local que tenga una relación de composición con la clase Comercio, y almacene el horario y días de atención.

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



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

- Agregar un comercio al vector de comercios con el operador "<<". Por ejemplo: para agregar un objeto comercio al vector polimorfico del objeto Delivery:

Listing 1: Ejemplo de operador <<

```
CDelivery objDelivery();
CComercio objComercio();
objDelivery << objComercio;
```

- Imprimir los comercios para un ranking con el operador "()". Tenga en cuenta que los rankings son números con punto flotante y el uso del operador debe realizarse con un valor entero. Por ejemplo: para mostrar los comercios con ranking 4:

Listing 2: Ejemplo de operador ()

```
objApp(4);
```

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 del Delivery:Ahora
Cantidad de comercios:7
Ingrese los 7 comercios:
R Velero 5.0 12-16 M-D Pescados
R Wu_Fan 4.2 11-24 M-D Oriental
R Ristorante 4.5 12-23 M-D Pastas
R Verde 3.9 12-18 M-D Vegetariano
B Timbo 3.2 8-23 L-D
B Xoox 4.1 9-24 L-D
B Samm 3.6 10-21 L-D

Ingrese el raning a buscar: 3
Comercio [1] Tipo: Restaurante, Nombre: Verde, Ranking: 3.9,
    Horario: 12-18, Dias: M-D, Especialidad: Vegetariano
Comercio [2] Tipo: Bodega, Nombre: Timbo, Ranking: 3.2,
   Horario: 8-23, Dias: L-D
Comercio [3] Tipo: Bodega, Nombre: Samm, Ranking: 3.6,
   Horario: 10-21, Dias: L-D
El delivery Ahora tiene 3 comercios para ofrecer
```

Listing 4: Ejemplo de resultado 2

```
Ingresar nombre del Delivery:Ahora
Cantidad de comercios:8
Ingrese los 8 comercios:
R Velero 5.0 12-16 M-D Pescados
R Wu_Fan 4.2 11-24 M-D Oriental
R Ristorante 4.5 12-23 M-D Pastas
R Verde 3.9 12-18 M-D Vegetariano
B Timbo 3.2 8-23 L-D
B Xoox 4.1 9-24 L-D
B Samm 3.6 10-21 L-D
B Once 4.4 00-24 L-D

Ingrese el ranking a buscar:4
Comercio [1] Tipo: Restaurante, Nombre: Wu_Fan, Ranking:
   4.2, Horario: 11-24, Dias: M-D, Especialidad: Oriental
Comercio [2] Tipo: Restaurante, Nombre: Ristorante, Ranking:
    4.5, Horario: 12-23, Dias: M-D, Especialidad: Pastas
Comercio [3] Tipo: Bodega, Nombre: Xoox, Ranking: 4.1,
   Horario: 9-24, Dias: L-D
Comercio [4] Tipo: Bodega, Nombre: Once, Ranking: 4.4,
   Horario: 00-24, Dias: L-D
El delivery Ahora tiene 4 comercios para 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%)