Programación Orientada a Objetos 1

Práctica Calificada 3

Pregrado

2022-I

Profesor: Jonathan Silva

Lab: 1.05

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	

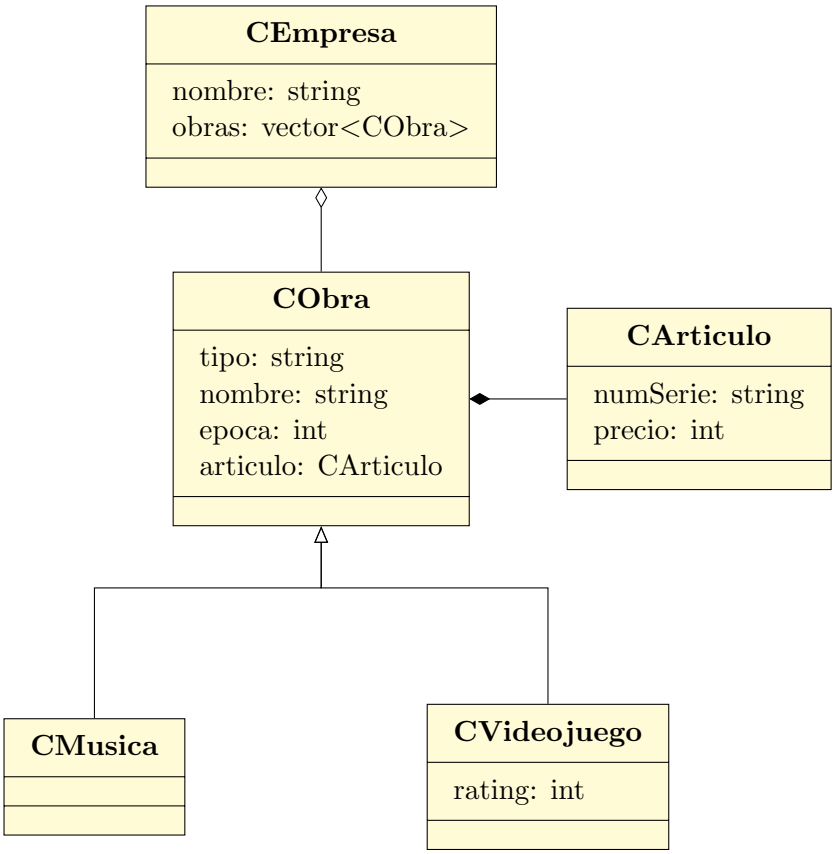
1. (20 points) **Articulos de colección por época**

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

   Una empresa que vende articulos de colección sobre musica ha adquirido otra empresa que vende videojuegos de colección y requiere guardar la información de todos los articulos que ofrece. Para eso se le solicita desarrollar las clases necesarias para almacenar los datos de los articulos para dos tipos de obras (musica y videojuegos) y luego consultar los datos de las obras por epoca (Ej: 1980, 1990, 2000, etc.).
   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 Obra que implemente herencia a las sub clases Musica y Videojuego. Los objetos obra deben contener el tipo de obra (Musica o Videojuego), el nombre de la obra y la epoca en la que fue publicada. Además, los videojuegos tienen un dato especifico sobre el rating de la edad adecuada (Ej: E, PG13, PG18).
   - Cree una clase Empresa que almacene el nombre de la empresa y un vector polimorfico con objetos de la clase musica y videojuego.
   - Cree una clase Articulo que tenga una relación de composición con la clase Obra, y almacene la el número de serie del articulo y el precio.

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



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

- Agregar un articulo al vector de articulos con el operador "<<". Por ejemplo: para agregar un objeto videojuego al vector polimorfico del objeto empresa:

Listing 1: Ejemplo de operador <<

```
CEmpresa objEmpresa();
CVideojuego objVideojuego();
objEmpresa << objVideojuego;
```

- Imprimir los articulos para una epoca específica con el operador "()". Por ejemplo: para imprimir la lista de los articulos para la epoca 1990:

Listing 2: Ejemplo de operador ()

```
objApp(1990);
```

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 Empresa:Epicos
Cantidad de obras:7
Ingrese las 7 obras:
V Atlantis_II 1990 A1212 6000 PG7
V Super_Mario_Bros 1980 S2323 33000 E
V Suikoden_II  2000 S3434 500 T
M Rage_Against_the_Machine 1990 R9988 550
M Fix_You 2000 T8877 440
M Nevermind 1990 N7766 5000
M Invincible 2000 I6655 1450

Ingrese la epoca a buscar: 2000
Obra [1] Tipo: Videojuego, Nombre: Suikoden_II, Epoca: 2000,
    Serie: S3434, Precio: 500, Rating: T
Obra [2] Tipo: Musica, Nombre: Fix_You, Epoca: 2000, Serie:
   T8877, Precio: 440
Obra [3] Tipo: Musica, Nombre: Invincible, Epoca: 2000,
   Serie: I6655, Precio: 1450
La empresa Epicos tiene 3 obras que ofrecer
```

Listing 4: Ejemplo de resultado 2

```
Ingresar nombre de la Empresa:Epicos
Cantidad de obras:8
Ingrese las 8 obras:
V Atlantis_II 1990 A1212 6000 PG7
V Super_Mario_Bros 1980 S2323 33000 E
V Hercs_Adventures 1990 H3434 700 PG13
V Suikoden_II  2000 S3434 500 T
M Rage_Against_the_Machine 1990 R9988 550
M Fix_You 2000 T8877 440
M Nevermind 1990 N7766 5000
M Invincible 2000 I6655 1450

Ingrese la epoca a buscar: 1990
Obra [1] Tipo: Videojuego, Nombre: Atlantis_II, Epoca: 1990,
    Serie: A1212, Precio: 6000, Rating: PG7
Obra [2] Tipo: Videojuego, Nombre: Hercs_Adventures, Epoca:
   1990, Serie: H3434, Precio: 700, Rating: PG13
Obra [3] Tipo: Musica, Nombre: Rage_Against_the_Machine,
   Epoca: 1990, Serie: R9988, Precio: 550
Obra [4] Tipo: Musica, Nombre: Nevermind, Epoca: 1990, Serie
   : N7766, Precio: 5000
La empresa Epicos tiene 4 obras 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%)