Programación Orientada a Objetos 1

Práctica Calificada 3

Pregrado

2021-II

Profesor: Rubén Rivas

Lab: 104

Indicaciones específicas:

- Esta evaluación contiene 5 páginas (incluyendo esta página) con 1 preguntas. El total de puntos son 20.
- El tiempo límite para la evaluación es 100 minutos.
- El código debe incluir OBLIGATORIAMENTE los siguientes archivos: **PC3.cpp** y **PC3.h**
- Deberá subir estos archivos directamente a www.gradescope.com.
- Se solicita activar cámara durante la evaluación. En caso de contingencia, justifique por correo electrónico rrivas@utec.edu.pe

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)

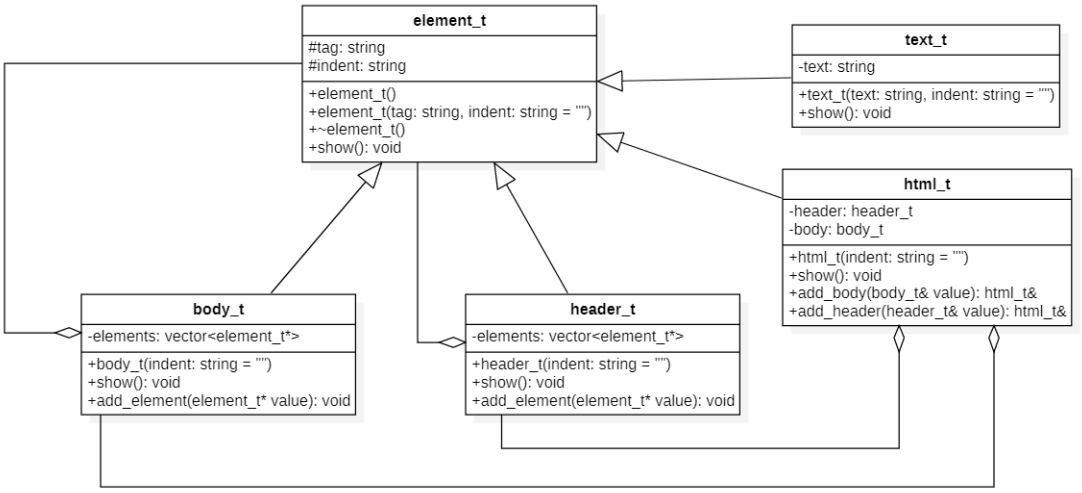
Calificación:

Tabla de puntos (sólo para uso del professor)

Question	Points	Score
1	20	
Total:	20	

1. (20 points) **document html**

   Desarrollar el siguiente modelo orientado a objetos que permite simular la elaboracion de un document html:



   Considerar que el valor del atributo **tag** debera ser actualizado en el constructor de cada clase con el nombre de la clase sin "**_t**" (ejemplo para la clase html_t el valor del **tag** seria "html"). excepto en el caso de la clase **text_t** donde el valor debe ser blanco y solo debera inprimirse en el metodo **show()** el texto. el valor del atributo **tag** se utilizar para inprimir los tags al inicio y al final de cada clase en el metodo **show()**, ejemplo:

```
void show() {
  cout << indent << "<" << tag << ">" << endl;
  for (auto &e : elements) {
    e->show();
  }
  cout << indent << "</" << tag << ">" << endl;
}
```

   En caso de **text_t** el metodo **show()** solo imprime el texto, ejemplo:

```
void show() {
  cout << indent << "<" << text << ">" << endl;
}
```

   En caso de **html_t** el metodo **show()** imprime, ejemplo:

```
void show() {
  cout << indent << "<" << tag << ">" << endl;
  header.show();
  body.show();
  cout << indent << "</" << tag << ">" << endl;
}
```

Luego de desarrollado debera de probarlo con el siguiente codigo, para lo cual en el caso de la clase **html_t** debera de sobrecargarse el operador << para imprimir el texto:

```
// document html
html_t html;

// cabecera de html
header_t header("\t");
text_t text_1("\t\tTexto␣1\n");
header.add_element(&text_1);

// cuerpo de html
body_t body("\t");
text_t text_2("\t\tTexto␣2\n");
body.add_element(&text_2);

// construyendo html
html.add_body(body);
html.add_header(header);

// imprimiendo
cout << html;
```

Resultado:

```
<html>
    <header>
        Texto 1
    </header>
    <body>
        Texto 2
    </body>
<html>
```

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%)