Nombre: ..... Fecha: / 12 / 2008

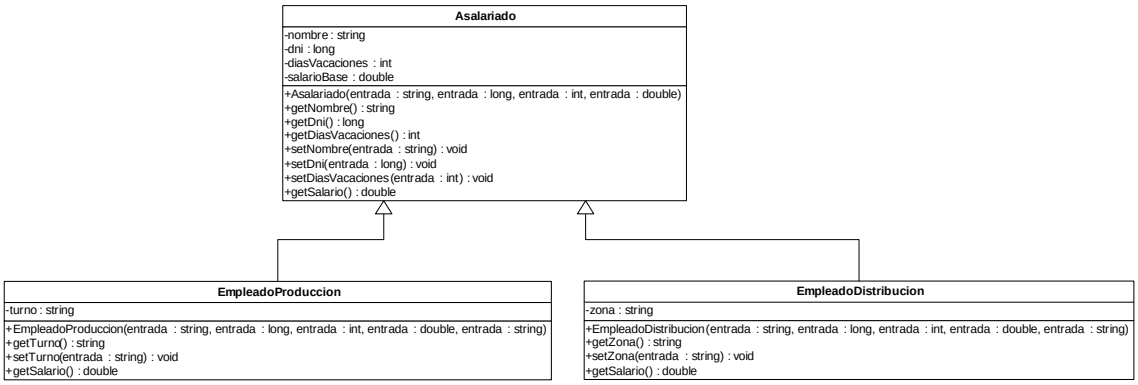
Grupo: 1 ☐ 2 ☐ 3 ☐ 4 ☐

PRÁCTICA 9
USO DE MÉTODOS ABSTRACTOS Y CLASES ABSTRACTAS EN C++ Y JAVA

En esta práctica recuperaremos uno de los ejemplo que introdujimos en el Tema 3 para ilustrar la redefinición de métodos para tratar de enriquecerlo por medio del uso de métodos abstractos y clases abstractas tanto en C++ como en Java.

Parte 1. Métodos abstractos y clases abstractas en C++ y Java.

Recuperamos el siguiente diagrama de clases en UML que introdujimos en el Tema 3.



Los métodos especificados tienen el comportamiento esperado. El método "getSalario(): double" en la clase "Asalariado" devuelve el atributo "salarioBase: double". En las clases derivadas ("EmpleadoProduccion", "EmpleadoDistribucion") devuelve el valor de "salarioBase: double" más una bonificación del 15% y del 10% respectivamente.

1. Programa el anterior diagrama de clases en C++ y Java. Observa que el método "getSalario(): double" está redefinido y por tanto debe tener comportamiento *polimorfo*. Tenlo en cuenta a la hora de declararlo en C++ en el fichero "Asalariado.h".

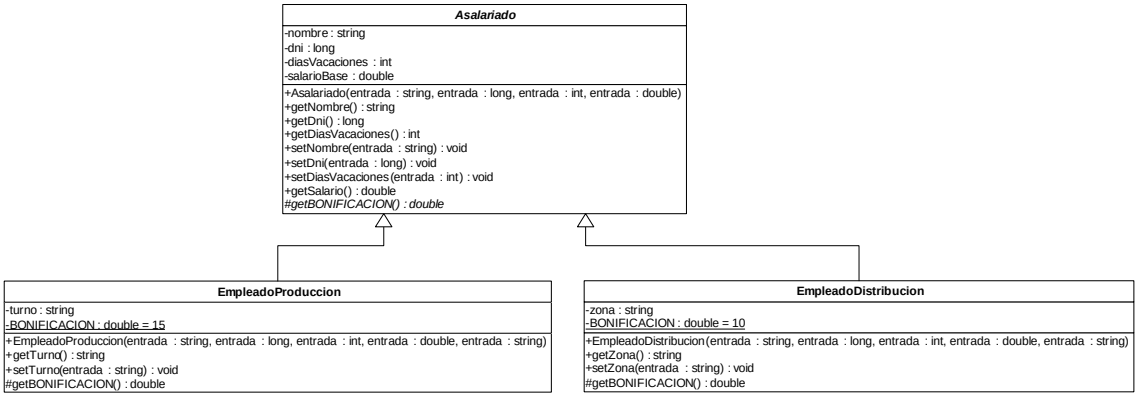
2. Declara un programa cliente del anterior sistema de clases en forma de una función "main" (créala en un fichero de nombre "principal_prac9") tanto en Java como en C++ que realice las siguientes operaciones:

a) Declara un objeto "emplead1" de la clase "Asalariado"
b) Crea el anterior objeto por medio del constructor "Asalariado" con nombre "Manuel Cortina", DNI "12345678", días de vacaciones igual a 28 y salario base igual a 1200
c) Declara un objeto "emplead2" de la clase "Asalariado"
d) Crea el anterior objeto por medio del constructor "EmpleadoProduccion" con nombre "Juan Mota", DNI "55333222", días de vacaciones igual a 30, salario base igual a 1200 y turno "noche"
e) Declara un objeto "emplead3" de la clase "Asalariado"
f) Crea el anterior objeto por medio del constructor "EmpleadoDistribucion" con nombre "Antonio Camino", DNI "55333666", días de vacaciones igual a 35, salario base igual a 1200 y región "Granada"
g) Comprueba que el método "getSalario(): double" se comporta de modo polimorfo para los tres objetos, "emplead1", "emplead2" y "emplead3"

3. Comprueba que, en Java, es posible hacer uso del modificador "abstract" para clases. Declara la clase "Asalariado" como "abstract" y comprueba qué ha sucedido. ¿Cómo ha afectado esto a la construcción de los objetos? ¿Cómo ha afectado a la declaración de los mismos?

Pon entre comentarios la declaración y la construcción (y la llamada al método "getSalario(): double") del objeto "emplead1".

4. Modifica la declaración y definición de las clases "Asalariado", "EmpleadoProduccion" y "EmpleadoDistribucion" en Java y en C++ para que se ajusten al siguiente diagrama UML:

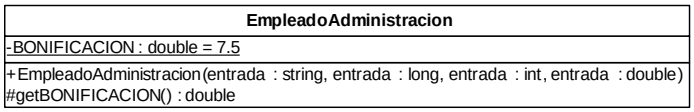


La constante de clase "BONIFICACION: double" tomará los valores correspondientes a la bonificación por encima del salario base que obtiene cada uno de los tipos de empleado (en esta caso, 10% ó 15%). El método "*getBONIFICACION(): double*" en la clase "*Asalariado*" debe ser declarado como abstracto, y por tanto la clase también lo será. En las clases "EmpleadoProduccion" y "EmpleadoDistribucion", el método "getBONIFICACION(): double" devolverá el valor correspondiente a la constante de clase "BONIFICACION: double".

Observa también que el método "getSalario(): double" ha desaparecido de las clases "EmpleadoProduccion" y "EmpleadoDistribucion" y ya sólo existe una versión del mismo, en la clase abstracta "*Asalariado*". Ten en cuenta que debes modificar la definición de "getSalario(): double" en "*Asalariado*" para que devuelva el resultado de añadir al atributo "salarioBase: double" el resultado de "getBONIFICACION(): double", que es abstracto en "*Asalariado*".

5. Vuelve a ejecutar el cliente de las clases anteriores (la función o método "main") que has definido en el ejercicio 2 (sin hacer uso del objeto "emplead1") en Java y C++. Comprueba que su comportamiento es el mismo a pesar de los cambios realizados.

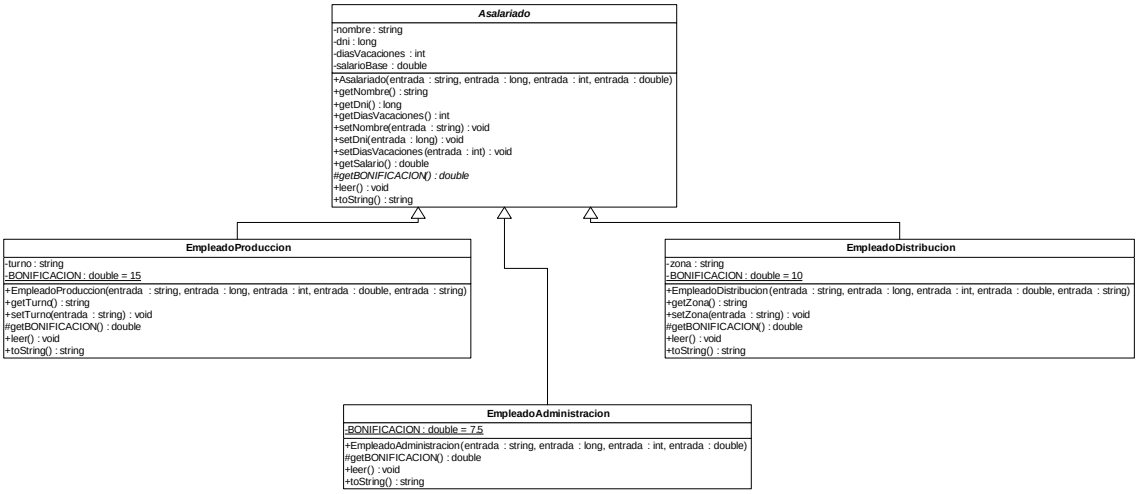
6. Define una nueva clase "EmpleadoAdministracion" que herede de la clase abstracta "*Asalariado*" y que se corresponda con el siguiente diagrama UML:



Declara un objeto "emplead4" de la clase "*Asalariado*" por medio del constructor de la clase "EmpleadoAdministracion" con nombre "Manuel Cortina", DNI "12345678", días de vacaciones igual a 28 y salario base igual a 1200. Comprueba el resultado de invocar al método "getSalario(): double" sobre el mismo desde la función "main".

Repasaremos ahora las funciones de lectura desde la consola estándar y escritura en Java y C++.

Modifica el diagrama de clases anterior añadiendo un método "leer(): void", que, invocado sobre un objeto cualquiera, lea los atributos del mismo (y los actualice), y un método "toString(): string" que los muestre por pantalla. El diagrama de clases deberá quedar de la siguiente forma:



El método "leer(): void" en la clase "*Asalariado*" debe ser capaz de leer desde el teclado los atributos "nombre: string", "dni: long", "diasVacaciones: int" y "salarioBase: double" y asignárselos al objeto desde el que ha sido invocado.

De igual modo, "leer(): void" en las clases derivadas "EmpleadoProduccion", "EmpleadoDistribucion" y "EmpleadoAdministracion" debe invocar al método "leer(): void" de la clase base "*Asalariado*" y debe leer los atributos restantes.

El método "toString(): string" en la clase "*Asalariado*" debe devolver una cadena de texto que contenga la siguiente información:

```
La clase a la que pertenece el objeto es ...
El nombre del asalariado es ...
El DNI del asalariado es ...
El número de días de vacaciones es ...
//Observa que el siguiente método se debe comportar de modo polimorfo:
El salario base del mismo es ...
//Y el siguiente método es abstracto
La bonificación obtenida por este trabajador es ...
```

En las clases derivadas, el método "toString(): string" debe llamar al método "toString(): string" de la clase "*Asalariado*" y mostrar los parámetros adicionales (el "turno: string" o la "zona: string")

7. Comprobamos ahora que la clase abstracta "*Asalariado*" puede ser utilizada como tipo base para definir estructuras genéricas. Define, tanto en Java como en C++, dentro del "main", un "array" de nombre "array_empl" de tipo "*Asalariado*" con 10 componentes. Ten en cuenta que tanto en Java como en C++ los métodos de las componentes del "array" van a tener que comportarse de modo polimorfo, y las consecuencias que eso tiene en la declaración del mismo.

Construye las 10 componentes por medio de un bucle, en el cual se le debe preguntar al usuario qué tipo de "Asalariado" quiere construir (por ejemplo, "1" puede ser para "EmpleadoProduccion", "2" para "EmpleadoDistribucion" y "3" para "EmpleadoAdmisnitracion"). Recuerda que, de una clase abstracta, no se puede construir objetos. La estructura del bucle podría ser:

```
for (int i = 0; i < 10; i++){
        Mostrar ("Qué tipo de empleado quieres construir:");
        Leer (tipo);
        Si (tipo == 1){
                array_empl [i] = new EmpleadoProduccion(...);
        }
        Si (tipo == 2){
                array_empl [i] = new EmpleadoDistribucion(...);
        }
        Si (tipo == 3){
                array_empl [i] = new EmpleadoAdministracion(...);
        }
        array_empl [i].leer();
```

}

8. Define un nuevo bucle que recorra las 10 componentes de "array_empl" y muestre cada una de sus componentes por pantalla, por medio del método "toString(): string".

ENTREGAR los archivos *Asalariado.cpp*, *Asalariado.h*, *EmpleadoProduccion.cpp*, *EmpleadoProduccion.h*, *EmpleadoDistribucion.cpp*, *EmpleadoDistribucion.h*, *EmpleadoAdministracion.cpp*, *EmpleadoAdministracion.h*, *principal_prac9.cpp*, *Asalariado.java*, *EmpleadoProduccion.java*, *EmpleadoDistribucion.java*, *EmpleadoAdministracion.java*, *principal_prac9.java*. Los archivos deben contener **todos los cambios** realizados durante los ejercicios de la práctica. La entrega es a través de Belenus en el repositorio de Julio Rubio. También deben contener unas cabeceras que permitan identificarte:

/* Nombre: ___
Grupo: _____
Nombre del fichero: ____
Descripción: _____*/

Guarda una copia de los ficheros para las prácticas sucesivas