Programación Orientada a Objetos. Curso 2016-17
Práctica 2. Clases, herencia y ficheros en C++

La clase Persona gestiona el DNI, nombre, apellidos, dirección, localidad, provincia y país de una persona (todos datos de tipo string).
Codificar la clase Persona en los ficheros `persona.h` y `persona.cc` con los siguientes métodos:

1. Modificadores (set) y observadores (get) para cada dato (setDNI(), getDNI(), setNombre(), getNombre(), etc.).
2. Un constructor que recibe como parámetro obligatorio el DNI y como parámetro opcional el resto de datos en el orden indicado con un valor por defecto igual a "" (cadena vacía).
3. Un método getApellidosyNombre() que devuelve un string con el formato: "apellidos, nombre". Usar la concatenación (operador +) de la clase string.

Ten en cuenta que las funciones/métodos que sean muy breves debes hacerlas **inline**.

En esta práctica también vamos a comprobar como se copian objetos y como se asigna un objeto a otro. Para copiar un objeto se puede hacer al declararlo poniendo entre paréntesis el objeto del cual copiar:

```
Persona p;
. . .
Persona q(p); // q es una copia de p
```

O bien con el operador igual:

```
Persona p;
. . .
Persona q;
. . .
q=p; // asignamos el objeto p al objeto q
```

Vamos a comprobar esto y a probar el resto de la clase *Persona* mediante una serie de test que se encuentran en el fichero `persona_unittest.cc` que se proporciona y que tiene los siguientes tests:

a) test del constructor con parámetros por defecto
b) test del constructor con parámetros obligatorios
c) test del constructor de copia (se usará el constructor de copia por defecto)
d) test del operador = (se usará el operador = por defecto)

Codificar la clase Persona de forma que pase dichos tests.

El constructor de copia y el operador = que proporciona C++ funcionan bien para el caso de la clase *Persona*. Hay casos en los que tendremos que definir nuestros propio constructor de copia y nuestro propio operador =.

La clase Crupier hereda de la clase *Persona* y le añade un código alfanumérico de empleado del casino. El constructor debe recibir forzosamente DNI y código de empleado, y el resto de parámetros tienen un valor por defecto igua a "". Tener en cuenta que se le deben pasar los parámetros correspondientes al constructor de la clase base (*Persona*)

mediante iniciadores de la clase base como se indica aquí:

```
Crupier(string DNI, string codigo, string nombre="", string
apellidos="" . . .) : Persona(DNI, nombre, apellidos, . . . ,
pais) {codigo_=codigo;};
```

Añadir también los métodos getCodigo() y setCodigo().

Realizar un test unitario en el fichero `crupier_unittest.cc` análogo al de la clase Persona pero ampliado para la clase *Crupier*, es decir, con test análogos e incluyendo tests para el código del empleado y un test diferente para cada uno de sus métodos.
NOTA: Al final de la práctica se proporcionará un fichero `crupier_unittest.cc` para probar definitivamente este ejercicio.

Es importante observar que el constructor de copia y operador = de C++ funcionan también bien en el caso de la clase Crupier.

La clase Jugador hereda de la clase *Persona*, tiene un dinero en euros para apostar (dinero_, de tipo *int*, no se admiten fracciones de euro en las apuestas) y tiene un código alfanumérico de jugador. Además, cada jugador tiene una lista de apuestas (apuestas_).

Cada elemento de la lista tiene tres campos: un *int* para el tipo de apuesta, un *string* para el valor de la apuesta y un *int* para la cantidad apostada. Los tipos de apuesta son según su código:

1. Apuesta sencilla. Se apuesta a un número entre 0 y 36, y si sale, se gana 35 a 1 (se puede apostar y ganar al 0).
2. Apuesta rojo o negro. Se apuesta a un color y se paga 1 a 1. Si sale el cero, se pierde.
3. Apuesta par o impar. Se apuesta par o impar y se paga 1 a 1. Si sale el cero, se pierde.
4. Apuesta alto o bajo. Bajo es entre 1 y 18; alto entre 19 y 36. Se paga 1 a 1. Si sale el cero, se pierde.

La clase Jugador debe cumplir los siguientes requisitos:

1. El constructor debe recibir forzosamente DNI y código de jugador, el resto de parámetros tienen un valor por defecto igua a “”. El dinero debe ser inicializado siempre a 1000.
2. Observadores y modificadores para código de jugador (get/setCodigo())y dinero (get/setDinero()).
3. La lista de apuestas debe ser dinámica usando la clase *list* de la STL de C++ (C++ STL *list*) donde cada elemento de la lista guarda los tres datos antes mencionados: tipo, valor y cantidad.
4. Un método, *getApuestas()*, que devuelve la lista de apuestas (también podría hacerse pasando como parámetro una referencia a una lista de apuestas a la que se le asignan las apuestas del jugador, pero no es necesario ya que un objeto *list* puede devolverse en una función y se copia bien sin problemas en otro objeto de tipo *list* que reciba el valor devuelto).
5. Un método, *setApuestas()*, que borra las apuestas actuales y lee del fichero DNI.txt las nuevas apuestas, siendo DNI el DNI del jugador. El fichero tiene formato texto y la siguiente estructura:
   CÓDIGO-APUESTA,VALOR,CANTIDAD

CÓDIGO-APUESTA,VALOR,CANTIDAD
…
CÓDIGO-APUESTA,VALOR,CANTIDAD

Un ejemplo de este fichero en el formato indicado sería:

1,21,30
2,rojo,15
3,par,20
4,alto,12

6. Usar un editor de texto plano para crear un fichero de texto con algunas apuestas siguiendo el formato descrito.

Hacer tests para probar la clase *Jugador* en el fichero `jugador_unittest.cc` y un pequeño programa principal (`jugador-ppal.cc`) que pida el DNI y los datos personales de un jugador, lea del fichero de texto correspondiente sus apuestas y las muestre por pantalla. En realidad, si se diseñan bien los test, el programa principal de prueba no sería necesario (analizar esta afirmación).
Para hacer la lista REVISAR ANTES el ejemplo de STL list proporcionado en la web de la asignatura.
Igualmente, para el manejo de ficheros en C++ REVISAR ANTES los ejemplos y documentación aportada en la web de la asignatura sobre ficheros en C++ y la función getline() para leer un fichero texto con delimitadores, como es el caso del fichero del apartado 5 de la práctica.

NOTAS:

- Al final de la práctica se proporcionará un fichero `jugador_unittest.cc` para probar definitivamente este ejercicio.
- Todos los ejercicios deben hacerse con los estándares de calidad que estamos aprendiendo en las clases de teoría. Esto es más importante que la correcta ejecución del ejercicio en sí.
- Usar los identificadores de las clases y los métodos exactamente como aparecen en el enunciado ya que luego pasaremos unos tests en los que usaremos dichos identificadores.