Examen Práctico de la Asignatura Programación Orientada a Objetos
Grado en Ingeniería Informática. Febrero 2023

NORMAS DE EXAMEN:

1. El examen práctico se hace en una cuenta anónima de examen que se entrega a cada alumno con su usuario y su contraseña. Entrar, usar o conectarse a otra cuenta supone el suspenso de la asignatura.
2. La corrección del examen se hace ejecutando los tests de cada ejercicio. Si no sabes hacer alguna función, defínela y déjala vacía y/o devolviendo un valor literal para que el programa al menos compile y puedan ejecutarse el resto de los tests. Aunque algún test falle, puntuarán los test correctos. Si un ejercicio/test no compila, se considerará mal ese ejercicio/test completo.
3. Si codificas una clase utilizando solamente el fichero .h, crea su correspondiente fichero .cc dejándolo vacío para que funcione bien el Makefile. Ten cuidado de poner los nombres adecuados a los ficheros tal y como se indica en el enunciado. Si tienes alguna duda consulta el Makefile.
4. Una vez codificados los ejercicios puedes compilar el primer ejercicio ejecutando `make test1`, el segundo ejercicio ejecutando `make test2`, y así sucesivamente. Ejecutando simplemente `make`, se compilan todos los ejercicios.
5. Una vez compilado, se puede ejecutar el test del ejercicio 1 ejecutado `./test1`, el test del ejercicio 2 ejecutando `./test2`, etc.
6. Modificar el Makefile, los tests o el enunciado del examen supondrá suspenso.
7. Toda la actividad del alumno será *logueada* y cualquier actividad diferente a la del uso del editor, compilador, programa unzip, depurador y programa make supondrá el suspenso total de la asignatura tanto de la teoría como de la práctica. Por tanto, queda prohibido abrir un navegador o cualquier otro programa que no sea los que se han indicado.
8. En el directorio HOME se encuentra un subdirectorio que se llama `examen` en el que debes realizar todo el examen (para la corrección no se mirará en ningún otro directorio). Crear otro directorio y/o **cambiar a cualquier otro directorio**, a otra cuenta, o conectarse a internet supondrá el suspenso definitivo de la asignatura tanto de la teoría como de la práctica.
9. Cada ejercicio tiene su puntuación indicada si se superan todos los tests. Si en un ejercicio falla algún test, puntuarán los test correctos proporcionalmente (salvo alguna excepción). Por tanto, si ejecutas los tests y haces la suma, sabrás tu calificación y también sabrás qué ejercicios tienes mal y el motivo.

EJERCICIOS (hay 5 ejercicios cada uno con su puntuación):

ej1.cc - Visual Studio... | pooP23-02.pdf

EJERCICIOS (hay 5 ejercicios cada uno con su puntuación):

1.- (test1) La clase **Product** representa un producto de una tienda con dos campos de tipo cadena de caracteres '*name*' y 'id', y un campo de tipo número real 'price'. Codifica el fichero product.h y product.cc para la clase *Product* con la siguiente funcionalidad:

- Constructor con 3 parámetros. El primer parámetro será 'id' y debe ser obligatorio. El segundo parámetro será 'name' y tendrá como valor por defecto 'no-name'. El tercer parámetro será 'price' con valor por defecto igual a 0.0. Controla los siguientes errores: si 'id' llega como la cadena vacía tendrá que iniciarse a la cadena "error". Si 'name' llega como una cadena vacía, deberán ponerse a su valor por defecto antes mencionado. Si el precio llega como parámetro con un valor inferior a 0, debe ponerse a 0.0.
- Modificador setId() que recibe como parámetro el nuevo 'id'. Si el nuevo 'id' es la cadena vacía, no se hará el cambio y se devolverá *false*. En caso contrario se hará el cambio y se devolverá *true*.
- Modificador setName() que recibe como parámetro el nuevo 'name'. Si el nuevo 'name' es la cadena vacía, no se hará el cambio y se devolverá *false*. En caso contrario se hará el cambio y se devolverá *true*.
- Modificador setPrice() que recibe como parámetro el nuevo precio que debe ser positivo, en cuyo caso se devuelve *true*. Si es menor que cero no se cambia y se devuelve *false*.
- Observadores getPrice(), getId() y getName().

- Observador getIdName() que devuelve una cadena que comienza con 'id', seguido de una coma y un espacio, seguido de 'name'. Ejemplo: si id="PC1" y model="Xy7", entones devolverá: "PC1, Xy7".
- Observador getStatus() que devuelve la cadena "ok" (en minúsculas) si el campo 'id' está asignado a un valor distinto de 'error' y el campo 'name' a un valor distinto a "no-name" y el precio es mayor que 0. En caso contrario, debe devolver la cadena "error" (también en minúsculas).
- Observador writeData() de tipo void que escriba en un fichero de salida (data.txt) una cadena que comienza con 'id', seguido de una coma y un espacio, seguido de 'name'.

PUNTUACIÓN: 2 puntos

2.- (test2) La clase **Cart** representa una cesta de la compra de una tienda online y tiene un campo de tipo entero "id" y una lista de objetos de la clase *Product* del ejercicio anterior (para la lista **usar el tipo list de la STL** de C++). Codifica los ficheros cart.h y cart.cc con la siguiente funcionalidad:

a) Constructor que recibe como parámetro obligatorio el *id* de la compra.
b) Observador getId() que devuelve el id.
c) Observador getN() que devuelve el tamaño de la lista.
d) El método addProduct() de tipo bool que recibe un objeto de tipo *Product*. Si no existe un producto en la lista con ese mismo 'id', entonces lo añade a la cesta y devuelve true. Si existe en la lista un producto con ese mismo id, no lo añade y devuelve false.
e) El método deleteProduct() que recibe un string con un el 'id' de un producto. El método debe borrar el elemento de la lista cuyo 'id' sea igual al pasado como parámetro. Si no existe ningún elemento con ese 'id', no hace nada y devuelve *false*. Si lo encuentra, lo borra y devuelve *true*.
f) Observador getTotal() que devuelve la suma de los precios de los productos en la cesta.
g) El método print(), de tipo void, que muestra en pantalla un producto por línea con su posición en la lista seguida de "/", seguido del tamaño de la lista, seguido de dos puntos y espacio, seguido de su 'id', una coma, 'name', una coma y, finalmente, el precio de ese producto. A continuación una nueva línea que comience por "TOTAL=" seguido de la suma total de los precios de la cesta.

1/tamaño_lista: id,nombre,precio
2/tamaño_lista: id,nombre,precio
3/tamaño_lista: id,nombre,precio
TOTAL=suma

Durante la ejecución del test2 deberá mostrarse en pantalla:
1/3: ID1,N1,111.1
2/3: ID2,N2,222.2
3/3: ID3,N3,444.4
TOTAL=777.7

PUNTUACIÓN: 2 puntos

3.- (test3) La clase **Computer** deriva de la clase Product del ejercicio anterior y además gestiona una variable de tipo *int* denominada "ram". Escribe el código de la clase *Computer* en los ficheros computer.h y computer.cc con la siguiente funcionalidad:

- Constructor que recibe como parámetros obligatorios 'ram' de la clase *Computer*, y 'id', 'name' y 'price' de la clase base también obligatorios. Si el valor de 'ram' recibido es menor o igual a cero se asignará a 'ram' el valor 1.
- Observador getRam().
- Modificador setRam() que recibe el nuevo valor para 'ram'. Si el parámetro es menor o igual a 0 debe devolver *false* y no asignarlo. Si por el contrario es positivo, debe asignarlo y devolver *true*.
- Observador getIdNameRam() que devuelve una cadena que comienza con 'id', seguido de una coma y un espacio, seguido de 'name', seguido de una coma y un espacio, seguido de 'ram'.

Ejemplo: si id="PC1", model="Xy7" y 'ram'=1024, entones devolverá: "PC1, Xy7, 1024".

PUNTUACIÓN: 1 puntos

4.- (test4) La clase *Mapa* tiene un mapa de nombre 'ciudades' de pares key-value donde key es de tipo string (representa el nombre de una ciudad) y value es de tipo int (representa los habitantes de esa ciudad). Para el mapa **usar el tipo map de la STL** de C++.
Declara la clase Mapa en los ficheros mapa.h y mapa.cc con las siguientes funciones:

a) Constructor sin parámetros.
b) getN() que devuelve el número de ciudades en el mapa.
c) setCiudad() con dos parámetros. El primero 'ciudad' es un string con el nombre de la ciudad y el segundo 'n' es el número de habitantes de esa ciudad. Si la ciudad no existe en el mapa, debe añadirse una nueva entrada al mapa con esa ciudad y el número de sus habitantes, y la función debe devolver true. Si la ciudad es la cadena vacía, no hace nada y debe devolver false. Si la ciudad existe, debe cambiar su número de habitantes y devolver true.
d) getCiudad() de tipo int que recibe como parámetro el nombre de una ciudad y devuelve sus habitantes si existe, o el valor -1 si no existe.
e) deleteCiudad() que recibe el nombre de la ciudad. Si existe la ciudad, la borra del mapa y devuelve true. Si no existe, no borra nada y devuelve false.

PUNTUACIÓN: 2 puntos

5.- (test5) La clase Punto dispone de un entero para la coordenada *x* y otro para la coordenada *y* de un punto en el plano. Codifica en los ficheros *punto.h* y *punto.cc* los siguientes métodos para esta clase:

a) Constructor con los valores iniciales de ***x*** e ***y*** como parámetros obligatorios.
b) Observador getX() y getY() para cada coordenada del punto.
c) Sobrecarga el operador + para que devuelva un punto cuyas coordenadas sean la suma de las coordenadas de los dos puntos.
d) Sobrecarga el operador ++ (tanto prefijo como sufijo) para que incremente en uno cada coordenada del punto sobre el que se aplica el operador.
e) Sobrecarga del operador >> (extractor) que saque en pantalla las coordenadas de un punto entre paréntesis. Ejemplo (saldrá en pantalla en la ejecución del test5):
(555,888)
f) Sobrecarga del operador << (insertador) que pida por teclado las coordenadas del punto de la siguiente forma (saldrá en pantalla en la ejecución del test test5):

Introduce coordenada X: *(el usuario la introduce por teclado)*
Introduce coordenada Y: *(el usuario la introduce por teclado)*

En la ejecución del test5 se usarán todos los operadores, y para que este ejercicio esté completamente bien deben ejecutar bien los test y la entrada y salida en pantalla de los puntos. Por tanto, realiza las comprobaciones necesarias.

PUNTUACIÓN: 3 puntos