EXAMEN JULIO 2022
PROGRAMACIÓN DE SISTEMAS DE TIEMPO-REAL

GRADO DE INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA

industriales
etsii UPCT

Universidad Politécnica de Cartagena

EJERCICIO 1

Se pide que diseñe parte de la infraestructura de una aplicación de alquiler de habitaciones y emplazamientos partiendo del siguiente código:

```
struct Alojamiento{
    virtual double calcular_precio() const = 0;
    friend ostream& operator<< (ostream &os, const Alojamiento &a);
    Alojamiento (string direccion) : direccion_{direccion} {}
    string get_dir() const { return direccion_; }
protected :
    virtual void print(ostream &os) const = 0;
private:
    string direccion_;
};
ostream& operator<< (ostream &os, const Alojamiento &a) {
    a.print(os);
    return os;
};
```

Se pide que implemente las siguientes clases, que comparten una jerarquía de herencia:

Hotel

- Hereda de Alojamiento.
- Tiene como **atributos** el número de estrellas, el precio por noche (se supone que es siempre el mismo), número máximo de huéspedes que se pueden aceptar por reserva, y el mínimo y máximo de días que se puede realizar una reserva.
- Tiene un **constructor** que recibe como parámetros valores para inicializar todos los atributos, y otro constructor (delegado), que asume que el valor mínimo de días es 1 y el máximo 7.
- Implementa los dos **métodos virtuales** de Alojamiento. Para calcular el precio, el método tiene que preguntar el número de huéspedes y el número de días que se quedarán, asumiendo que las habitaciones son individuales. Si algún valor es negativo o está fuera del rango que establecen los atributos (número máximo de huéspedes que se pueden aceptar por reserva, y el mínimo y máximo de días que se puede realizar una reserva), el método retorna el código de error -1. Una posible implementación del método print podría generar: "*Hotel en Cartagena de 4 estrellas, precio por noche: 85*".

Vivienda

- Hereda de Alojamiento.
- Tiene como **atributos** el número mínimo de días que se puede alquilar (no hay máximo), la cantidad de huéspedes que caben en una vivienda (se supone que son todas iguales) y el precio de alquiler de la vivienda por día.
- Tiene un **constructor** que recibe como parámetros valores para inicializar todos los atributos.
- Implementa los dos **métodos virtuales** de Alojamiento. Para calcular el precio, el método tiene que preguntar el número de huéspedes y el número de días que se

quedarán. A partir del número de huéspedes y el valor del atributo cantidad-de-huéspedes-que-caben-en-una-vivienda el método calcula la cantidad de viviendas que hay que reservar. Si algún valor es negativo o está fuera del rango que establecen los atributos, el método retorna el código de error -1. Una posible implementación del método print podría generar: "*Vivienda en La Manga para 5 personas, precio por dia: 150*".

Castillo

- Hereda de Alojamiento.
- Tiene como **atributos** el número máximo de días que se puede alquilar (el mínimo es 1), el precio de alquiler del castillo, y un mapa de string a double que establece el precio de cada estancia del castillo que se quiera alquilar (van por separado del alquiler del propio castillo).
- Tiene un **constructor** que recibe como parámetros valores para inicializar todos los atributos.
- Implementa los dos **métodos virtuales** de Alojamiento. Para calcular el precio, el método tiene que preguntar si se quieren alquilar también las estancias especiales del castillo y el número de días que se alquilarán. Un ejemplo de definición de estas estancias es `{{"Trono", 1000}, {"Salon Banquetes", 2000}, {"Habitaciones reales", 3000}}`. Si algún valor es negativo o está fuera del rango que establecen los atributos, el método retorna el código de error -1. Una posible implementación del método print podría generar: "*Castillo en Madrid con 3 estancias, precio minimo por hacer reserva: 4000*".

Puede ampliar el número de funciones y métodos como estime necesario. Escriba una función **main** de prueba corta para comprobar el funcionamiento de las clases desarrolladas y que el polimorfismo funciona adecuadamente. Para ello, cree un objeto de cada clase, lo asigna a un puntero de tipo Alojamiento y luego lo imprime por consola a través del puntero. Solo puede utilizar la funcionalidad que proporciona la librería estándar de C++.

EJERCICIO 2

Se pide programar la infraestructura básica para gestionar los datos recogidos por sensores en una planta industrial. Para ello, se pide implementar la clase **`Centro_Datos`** que contenga los atributos necesarios para cumplir los siguientes requisitos:

1. Se deben poder almacenar todos los valores, de tipo *double*, enviados por los sensores instalados en la planta. Es decir, se guarda el histórico de valores, no únicamente el último que se ha recibido. Cada sensor está identificado por un *string*, que no contiene espacios en blanco.
2. Para evitar problemas de sensores con códigos erróneos, la clase debe programarse para aceptar solo los códigos de los sensores que se suministren al construir el objeto. De esta

forma, si llega un valor de un sensor cuyo código de identificación no se ha suministrado al construir el objeto, debe ser ignorado.

En cuanto a la funcionalidad de la clase, se piden los siguientes métodos:

- **`Constructor`** que recibe una lista de códigos de identificación de sensores, para implementar el segundo requisito. En una posible implementación, puede inicializar el vector de valores asociado al código del sensor invocando, por ejemplo, el método **`reserve()`**.
- Método **`actualizar_datos()`**, que recibe como parámetro un string con los datos a procesar y no retorna nada. Otra parte del código, que no se pide en este ejercicio, es la encargada de recibir los datos de los sensores por un canal de comunicación, crear un string con todos ellos, e invocar este método para añadir los nuevos datos a Centro_Datos. La estructura básica del string es "código-sensor valor-double", repetida tantas veces como haga falta. Es decir, la cantidad de valores que contiene el parámetro de este método es variable. Un ejemplo puede ser: `string datos_2 {"s1 11 s2 11 s3 1 s2 21 s1 21 s1 31"}`. La implementación de este método tiene que tener en cuenta el segundo requisito.
- Método **`datos_sensor()`**, que recibe como parámetro el código de identificación de un sensor y retorna todos los valores que se han almacenado para dicho sensor. Si el código no estaba registrado, retorna un vector vacío.
- Método **`últimos_valores()`**, que retorna el último valor que se ha almacenado para cada sensor. Tenga cuidado con los sensores registrados, pero para los que todavía no se ha recibido ningún valor.
- El **`operator<<`** para imprimir el contenido de Centro_Datos por consola. Por ejemplo, el siguiente código mostraría la salida por consola que aparece en el cajetín (es un ejemplo de código, la implementación de su clase no tiene que coincidir al 100%):

```
Centro_Datos cd {{"s1", "s3"}};
string datos_1 {"s1 1 s2 1 s2 2 s1 2 s1 3"};
cd.actualizar_datos(datos_1);
cout << cd << '\n';
string datos_2 {"s1 11 s2 11 s3 1 s2 21 s1 21 s1 31"};
cd.actualizar_datos(datos_2);
cout << cd << '\n';
```

```
s1 -> {1, 2, 3, }
s3 -> {}

s1 -> {1, 2, 3, 11, 21, 31, }
s3 -> {1, }
```

Escriba una función **main** de prueba corta para comprobar el funcionamiento de la clase desarrollada. Solo puede utilizar la funcionalidad que proporciona la librería estándar de C++.

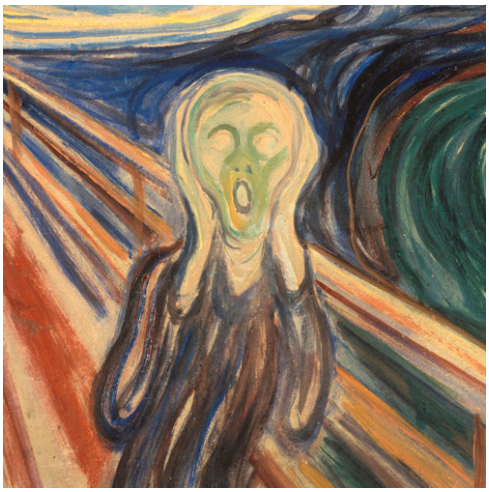
EJERCICIO 3: CONCURRENCIA

Se quiere realizar un simulador de un cruce a nivel, que se va a implementar mediante una clase que se comporta como un monitor, **Cruce_Nivel**. El paso a nivel es cruzado por coches y trenes, que se van a implementar como hilos en el simulador. El comportamiento es el siguiente. Mientras haya un coche o un tren cruzando el paso a nivel, es posible que otro coche o tren cruce el paso a nivel sin bloquear. Si un coche o tren intenta cruzar cuando un tren o un coche está cruzando, debe bloquear hasta que el cruce a nivel no esté siendo cruzado por ningún vehículo. En el simulador, el coche tarda 150 milisegundos en atravesar el paso a nivel, mientras que un tren tarda 300ms. Los trenes utilizan el método **cruza_tren()** del monitor, y los coches usan el método **cruza_coche()** para interactuar con el cruce a nivel. Cada vez que un vehículo (coche o tren) cruza el paso a nivel, imprime un mensaje por consola indicando cuántos vehículos lo están cruzando a la vez. Un ejemplo de salida por consola puede ser el siguiente (suponiendo que todos los hilos intentan acceder al paso a nivel a la vez):

```
Pasando tren 1, total cruzando 1
Pasando tren 0, total cruzando 2
Pasando tren 2, total cruzando 3
Pasando tren 3, total cruzando 4
Pasando tren 4, total cruzando 5
Tren 1 ha pasado, total cruzando 4
Tren 2 ha pasado, total cruzando 3
Tren 0 ha pasado, total cruzando 2
Tren 4 ha pasado, total cruzando 1
Tren 3 ha pasado, total cruzando 0
Pasando coche 3, total cruzando 1
Pasando coche 1, total cruzando 2
Pasando coche 2, total cruzando 3
Pasando coche 4, total cruzando 4
Pasando coche 0, total cruzando 5
Coche 3 ha pasado, total cruzando 4
Coche 1 ha pasado, total cruzando 3
Coche 2 ha pasado, total cruzando 2
Coche 4 ha pasado, total cruzando 1
Coche 0 ha pasado, total cruzando 0
```

Haga una función **main** que cree un objeto del tipo **Cruce_Nivel**, 10 hilos que se comportan como coche y otros 10 como trenes. Solo puede utilizar la funcionalidad que proporciona la librería estándar de C++.

Las soluciones vienen a continuación.
Consultar solo si ya está un poco desesperad@, o para comparar soluciones



Puede que las soluciones sean un poco más generales o haya más detalles de los que se piden en el enunciado del ejercicio. Se proporcionan para que el alumno tenga un código más completo. Como sabe, en programación **NO** existe la solución única a un ejercicio.

EJERCICIO 1

```
#include <iostream>
#include <string>
#include <map>

using namespace std;

struct Alojamiento{
    virtual double calcular_precio() const = 0;
    friend ostream& operator<< (ostream &os, const Alojamiento &a);
    Alojamiento (string direccion) : direccion_{direccion} {}
    string get_dir() const { return direccion_; }
protected :
    virtual void print(ostream &os) const = 0;
private:
    string direccion_;
};
ostream& operator<< (ostream &os, const Alojamiento &a) {
    a.print(os);
    return os;
};

int num_huespedes(const int max) {
    if (max<1) return -1;
    int res;
    do {
        cout << "Introduzca num huespedes, max " << max <<": ";
        cin >> res;
        if (!cin) {
            cin.clear(); // elimina error
            cin.ignore(9999, '\n');
        }
    } while (res<0 or res>max);
    return res;
}

int num_dias(const int min, const int max) {
    if (max<min) return -1;
    if (max<1 or min<1) return -1;
    int res;
    do {
        cout << "Introduzca num dias de la estancia, entre " << min << " y "
             <<  max <<": ";
        cin >> res;
        if (!cin) {
            cin.clear(); // elimina error
            cin.ignore(9999, '\n');
        }
    } while (res<min or res>max);
    return res;
}
```

```
class Hotel : public Alojamiento {
    int estrellas_;
    double precio_noche_;
    int max_huespedes_reserva_;
    int min_dias_, max_dias_;
    const int huespedes_hab=2;
public:
    Hotel(string dir, int est, double precio, int max_huespedes, int min_dias,
int max_dias): Alojamiento(dir), estrellas_{est}, precio_noche_{precio},
max_huespedes_reserva_{max_huespedes}, min_dias_{min_dias}, max_dias_{max_dias}
{}

    Hotel(string dir, int est, double precio, int max_huespedes): Hotel (dir,
est, precio, max_huespedes, 1, 7) {}

    double calcular_precio() const override {
        int huespedes = num_huespedes(max_huespedes_reserva_);
        int dias = num_dias(min_dias_, max_dias_);
        return huespedes*dias*precio_noche_/huespedes_hab;
    }


protected:
    void print(ostream &os) const override {
        os << "Hotel en " << get_dir() << " de " << estrellas_ ;
        os << " estrellas, precio por noche: " << precio_noche_ ;
    }
};


class Vivienda : public Alojamiento {
    int min_dias_, huespedes_vivienda_;
    double precio_vivienda_;
public:
    Vivienda(string dir, int min_dias, int huespedes_vivienda, double precio):
Alojamiento(dir), min_dias_{min_dias}, huespedes_vivienda_{huespedes_vivienda},
precio_vivienda_{precio} {}

    double calcular_precio() const override {
        int huespedes = num_huespedes(9999);
        int dias = num_dias(min_dias_, 9999);
        int viviendas = huespedes/huespedes_vivienda_;
        if (huespedes%huespedes_vivienda_ != 0) ++viviendas;
        return viviendas*precio_vivienda_*dias;
    }

protected:
    void print(ostream &os) const override {
        os << "Vivienda en " << get_dir() << " para " << huespedes_vivienda_;
        os << " personas, precio por dia: " << precio_vivienda_;
    }
};

class Castillo : public Alojamiento {
    int max_dias_;
    double precio_reserva_;
```

```
    map<string, double> estancias_;
public:
    Castillo(string dir, int max_dias, double precio, map<string, double>
estancias): Alojamiento(dir), max_dias_{max_dias}, precio_reserva_{precio},
estancias_ {estancias}{}

    double calcular_precio() const override {
        int dias = num_dias(1, max_dias_);
        double precio_total=precio_reserva;
        for (const auto &[est, precio] : estancias_) {
            cout << "Quiere reservar " << est << "? (s/n): ";
            char resp;
            cin >> resp;
            if (resp=='s' or resp=='S')
                { precio_total+=precio; }
        }
        return (precio_total*dias);
    }


protected:
    void print(ostream &os) const override {
        os << "Castillo en " << get_dir() << " con " << estancias_.size() ;
        os << " estancias, precio minimo por hacer reserva: "<<precio_reserva_ ;
    }
};

int main() {
    Alojamiento *p;
    Hotel h1 {"Cartagena", 4, 85,5};
    cout << h1 << endl;
    p = &h1;
    cout << *p << endl;
    Vivienda v1 {"La Manga", 14, 5, 150};
    cout << v1 << endl;
    p = &v1;
    cout << *p << endl;
    map<string, double> estancias {{"Trono", 1000}, {"Salon Banquetes", 2000},
           {"Habitaciones reales", 3000}};
    Castillo c1 {"Madrid", 3, 4000, estancias};
    cout << c1 << endl;
    p = &c1;
    cout << *p << endl;
    cout << p->calcular_precio() << "€ \n";
    return 0;
}
```

EJERCICIO 2

```
#include <iostream>
#include <string>
#include <map>
#include <vector>
#include <sstream>
```

```
using namespace std;

class Centro_Datos {
    map<string, vector<double>> datos_;
public:
    Centro_Datos (const vector<string> &sensores_permitidos) {
        for (const auto &v : sensores_permitidos) {
            datos_[v].reserve (10);
        }
    }

    void actualizar_datos (const string &txt) {
        istringstream iss {txt};
        while (!iss.eof()) {
            string codigo_sensor;
            double valor_sensor;
            iss >> codigo_sensor;
            if (iss) {
                iss >> valor_sensor;
                if (iss and datos_.count(codigo_sensor)!=0)
                      { datos_[codigo_sensor].push_back(valor_sensor); }
            }
            else { iss.clear(); }
        }
    }

    friend ostream& operator<< (ostream &os, const Centro_Datos &cd) {
        for (const auto &[k, vv] : cd.datos_){
            os << k << " -> {";
            for (const auto &v : vv) {
                os << v << ", ";
            }
            os << "}\n";
        }
        return os;
    }

    vector <double> datos_sensor (const string &codigo_sensor){
        vector <double> res;
        if (datos_.count(codigo_sensor) != 0) {
            res = datos_[codigo_sensor];
        }
        return res;
    }

    map<string, double> ultimos_valores () {
        map<string, double> res;
        for (const auto &[k, vv] : datos_){
            if (not vv.empty())
                { res[k] = vv.back(); }
        }
        return res;
    }
};
```

```
ostream& operator<< (ostream &os, const map<string, double>  &m) {
    for (const auto &[k, v] : m){
        os << k << " -> " << v << '\n';
    }
    return os;
}

int main() {
    Centro_Datos cd {{"s1", "s3"}};
    string datos_1 {"s1 1 s2 1 s2 2 s1 2 s1 3"};
    cd.actualizar_datos(datos_1);
    cout << cd << '\n';
    string datos_2 {"s1 11 s2 11 s3 1 s2 21 s1 21 s1 31"};
    cd.actualizar_datos(datos_2);
    cout << cd << '\n';
    auto r = cd.datos_sensor("s1");
    cout << r.size() << '\n';
    r = cd.datos_sensor("si1");
    cout << r.size() << '\n';
    auto ultimos {cd.ultimos_valores()};
    cout << ultimos;
    return 0;
}
```

EJERCICIO 3: CONCURRENCIA

```
#include <iostream>
#include <string>
#include <thread>
#include <mutex>
#include <chrono>
#include <condition_variable>
#include <vector>

using namespace std;

class Paso_Nivel {
    mutex m_, consola_;
    condition_variable hay_coche_, hay_tren_;
    int n_coches_, n_trenes_;
public:
    Paso_Nivel(): n_coches_{0}, n_trenes_{0} {}

    void atraviesa_coche(const int id, const chrono::milliseconds &t) {
        unique_lock<mutex> ul {m_};
        hay_tren_.wait(ul, [this](){return n_trenes_==0;}  );
        ++n_coches_;
        cout << "Pasando coche " << id <<", total cruzando "
             << n_coches_ << '\n';
        ul.unlock();
        this_thread::sleep_for(t);
        ul.lock();
        --n_coches_;
        cout << "Coche " << id <<" ha pasado, total cruzando "
             << n_coches_ << '\n';
        if (n_coches_ == 0)
            { hay_coche_.notify_all(); }
    }

    void atraviesa_tren(const int id, const chrono::milliseconds &t) {
        unique_lock<mutex> ul {m_};
        hay_coche_.wait(ul, [this](){return n_coches_==0;}  );
        ++n_trenes_;
        cout << "Pasando tren " << id <<", total cruzando "
             << n_trenes_ << '\n';
        ul.unlock();
        this_thread::sleep_for(t);
        ul.lock();
        --n_trenes_;
        cout << "Tren " << id <<" ha pasado, total cruzando "
             << n_trenes_ << '\n';
        if (n_trenes_ == 0)
            { hay_tren_.notify_all(); }
    }
};

int main() {
```

```
    Paso_Nivel p;
    vector<thread> coches, trenes;
    for (int i=0; i<5; ++i) {
        trenes.emplace_back(&Paso_Nivel::atraviesa_tren, &p, i,300ms);
        coches.emplace_back(&Paso_Nivel::atraviesa_coche, &p, i, 150ms);

    }
    for (int i=0; i<5; ++i) {
        coches[i].join();
        trenes[i].join();
    }
    return 0;
}
```