

1 – Singleton Pattern

2 – The singleton design pattern is used when you only want one object of any kind to ever be instantiated within the program. I used this to instantiate an in memory database of the us population.

Since I never want two in memory databases I was comfortable making use of the singleton design pattern to achieve this goal.

//a.File Name - Singleton.h

//b.Author - Joseph Rossitto

//c.Date - 11-27-20

//d.Compiler Used - Visual Studio

//e.Brief Description of the file - Singleton Pattern.

#pragma once

#include <iostream>

#include <fstream>

#include <string>

#include <map>

using namespace std;

class Singleton

{

private:

Singleton();

map<string, int> capitals;

public:

Singleton(Singleton const&) = delete;

void operator=(Singleton const&) = delete;

static Singleton& get();

void print();

map<string, int> load();

};

#include "Singleton.h"

Singleton::Singleton()

{

std::cout << "Initalizing database" << endl;

ifstream stream("capitals.txt");

string name;

string population;

while (getline(stream, name))

{

getline(stream, population);

capitals[name] = stoi(population);

}

}

Singleton& Singleton::get()

{

static Singleton db;

return db;

}

void Singleton::print()

{

for (auto it = capitals.cbegin(); it != capitals.cend(); ++it)

{

std::cout << "{" << (\*it).first << ": " << (\*it).second << "}\n";

}

}

map<string, int> Singleton::load()

{

return capitals;

}

1 – The Adaptor Pattern

2 – The adaptor is used when you want to change or adapt one data type to another

3 – I used the design pattern to convert my int datatype to a string data type displaying Million instead of the int I read from the in memory database of to change how it was displayed from 31 to 31 Million

//a.File Name - Adaptor.h

//b.Author - Joseph Rossitto

//c.Date - 11-27-20

//d.Compiler Used - Visual Studio

//e.Brief Description of the file - Adaptor Pattern.

#pragma once

#include <vector>

#include <stack>

#include <string>

#include <algorithm>

#include <sstream>

#include <iterator>

using namespace std;

class Adaptor

{

private:

string adaptingString;

int adaptingInt;

public:

Adaptor();

string toMillions(int number) const;

string get();

};

#include "Adaptor.h"

#include <iostream>

//using namespace std;

Adaptor::Adaptor()

{

}

string Adaptor::toMillions(int number) const

{

string millions{ adaptingString };

millions = to\_string(number) + " Million";

return { millions };

}

string Adaptor::get()

{

cout << this->adaptingString << endl;

return this->adaptingString;

}

1 – The State Pattern

2 – The state pattern is used when ever you want to switch between defined states in your program

3 – I chose to use this to switch between the allowing immigrants state and the not allowing immigrants state therefore when its on people can flow into your communities. When its off the population of the communities will no longer change.

//a.File Name - State.h

//b.Author - Joseph Rossitto

//c.Date - 11-27-20

//d.Compiler Used - Visual Studio

//e.Brief Description of the file - State Pattern.

#include <iostream>

#include <string>

#include <map>

#include <vector>

using namespace std;

enum class State

{

On,

Off

};

ostream& operator<<(ostream& os, const State& s);

enum class Trigger

{

Welcome,

Closed

};

ostream& operator<<(ostream& os, const Trigger& t);

#include "State.h"

ostream& operator<<(ostream& os, const State& s)

{

switch (s)

{

case State::On:

os << "immigrants welcome";

break;

case State::Off:

os << "Closed to new immigrants";

break;

}

return os;

}

ostream& operator<<(ostream& os, const Trigger& t)

{

switch (t)

{

case Trigger::Welcome:

os << "All immigrants welcome";

break;

case Trigger::Closed:

os << "Sorry closed for immigration";

break;

default: break;

}

return os;

}