**Source Code Of Token Based Single Sign On**

#include <iostream>

#include <vector>

#include <string>

using namespace std;

**// DevelopmentKits is an abstract class to be Usb-Stick object or Smart Card object to connect a port.**

class DevelopmentKits

{

public:

virtual string getKitName() = 0;

};

**// UsbKit Class is a concreate class of DevelopmentKits. It instantiates a UsbKit object which is refered Usb-Stick.**

class UsbKit : public DevelopmentKits

{

public:

UsbKit() : \_kitname("usb-stick") {}

string getKitName() { return \_kitname; }

private:

string \_kitname;

};

**//ScKit Class is a concreate class of DevelopmentKits. It instantiates a ScKit object which is refered Smart-Card.**

class ScKit : public DevelopmentKits

{

public:

ScKit() : \_kitname("smard-card") {}

string getKitName() { return \_kitname; }

private:

string \_kitname;

};

**//Port class is based on ports in the computer to connect smart-card object or usb-stick.**

class Port {

public:

Port(string portName, int usbvalid = 0, int scvalid = 0) : \_portName(portName), USBvalid(usbvalid), SCvalid(scvalid) { cout<<portName<<" is ready to use "<<endl;};

void setPort(DevelopmentKits \* kit) { \_kit = kit; };

**//It return the a kit in the port if It has a any connection with a kit.**

DevelopmentKits \* getPort() { return \_kit; }

string getPortName() { return \_portName; };

**//When we put a usb-stick. Usbvalid will set USBvalid=1. It shows us that this connected with a port.**

void setUSBvalid(int usbvalid) { USBvalid = usbvalid; }

**//When we put a smart card reader. setSCvalid will set SCvalid=1. It shows us that this connected with a port.**

void setSCvalid(int scvalid) { SCvalid = scvalid; }

**//It returns us that Usb-sticker has connection with a port or not.**

int getUSBvalid() { return USBvalid; }

**//It returns us that smart card reader has connection with a port or not.**

int getSCvalid() { return SCvalid; }

~Port(){}

private:

DevelopmentKits \*\_kit;

string \_portName;

int USBvalid;

int SCvalid;

};

**// AbstractIterator is a abstract class to iterate UsbKit or ScKit on the ports to read or write them.**

class AbstractIterator {

public:

virtual void First() = 0;

virtual void Next() = 0;

virtual bool IsDone() const = 0;

virtual Port\* CurrentItem() const = 0;

~AbstractIterator(){}

protected:

AbstractIterator() {};

};

class Collection;

**// CollectionIterator is a concrete class of AbstractIterator to iterate UsbKit or ScKit on the ports to read or write them.**

class CollectionIterator : public AbstractIterator {

public:

CollectionIterator(const Collection \*collection);

void First();

void Next();

Port\* CurrentItem() const;

bool IsDone()const;

private:

const Collection \*\_collection;

int \_current;

};

**//AbstractAggregate Class is an abstract class to collect the ports.**

class AbstractAggregate {

public:

virtual ~AbstractAggregate() {};

virtual AbstractIterator\* CreateIterator() = 0;

virtual void add(Port \*) = 0; // Not needed for iteration.

virtual int getCount() const = 0; // Needed for iteration.

virtual Port \* get(int) const = 0; // Needed for iteration.

protected:

AbstractAggregate() {};

};

**//Collection Class is an concreate class of AbstractAggregate Class to collect the ports.**

class Collection : public AbstractAggregate {

private:

vector<Port\*> \_ports;

public:

CollectionIterator\* CreateIterator() {

return new CollectionIterator(this);

}

int getCount() const { return \_ports.size(); }

void add(Port \*port) { \_ports.push\_back(port); };

Port \* get(int index) const { return \_ports[index]; };

};

#include "Avkan-Kafkas-Arslanparcasi-Project.h"

class ConnectionInterface;

class USB;

class SC;

class StickConnector;

class KitAdaptor;

class AbstractTemplateAPI;

class Manage;

class ManageSC;

class ManageUSB;

class DriverManager;

**// AbstractTemplateAPI is an abstract class read and write function collection for USB and Smart** Card

class AbstractTemplateAPI

{

public:

virtual void waitCardIsertion() = 0;

virtual void readData() = 0;

virtual void writeData() = 0;

virtual void encryptData() = 0;

virtual void decryptData() = 0;

virtual void deleteFile() = 0;

virtual void readFunction() = 0;

virtual void writeFunction() = 0;

};

**// AbstractTemplateAPI is an abstract class read and write function collection for USB**

class USBTemplate : public AbstractTemplateAPI

{

public:

void waitCardIsertion() { cout << "Wait for USB Token Insertion" << endl; }

void openFile() { cout << "Opening file in USB" << endl; }

void closeFile() { cout << "Closing file in USB" << endl; }

void readData() { cout << "Reading data from USB" << endl; }

void writeData() { cout << "Writing data to USB" << endl; }

void encryptData() { cout << "Encrypting data of USB" << endl; }

void decryptData() { cout << "Decrypting data of USB" << endl; }

void deleteFile() { cout << "Deleting data of USB" << endl; }

void readFunction() { waitCardIsertion(); openFile(); readData(); closeFile(); decryptData(); }

void writeFunction() { waitCardIsertion(); openFile(); encryptData(); writeData(); closeFile(); }

};

**// AbstractTemplateAPI is an abstract class read and write function collection for Smart Card**

class SCTemplate : public AbstractTemplateAPI

{

public:

void waitCardIsertion() { cout << "Wait for Card Insertion" << endl; }

void verifyPIN() { cout << "Verifying the PIN of Smart Card" << endl; }

void selectFile() { cout << "Selecting a file from Smart Card" << endl; }

void readData() { cout << "Reading data from Smart Card" << endl; }

void writeData() { cout << "Writing data to Smart Card" << endl; }

void encryptData() { cout << "Encrypting data of Smart Card" << endl; }

void decryptData() { cout << "Decrypting data of Smart Card" << endl; }

void deleteFile() { cout << "Deleting data of Smart Card" << endl; }

void readFunction() { waitCardIsertion(); verifyPIN(); selectFile(); readData(); decryptData(); }

void writeFunction() { waitCardIsertion(); verifyPIN(); selectFile(); encryptData(); writeData(); } };

**//Manage Class is a abstract class to send read or write function commend to AbstractTemplateAPI class**

class Manage {

public:

virtual void read() = 0;

virtual void write() = 0;

protected:

Manage() {};

};

**//Concrete ManageSC Command Class to send read or write function commend to AbstractTemplateAPI class for Smart Card**

class ManageSC : public Manage {

public:

ManageSC(AbstractTemplateAPI \* tempAPI, string kitName) :\_tempAPI(tempAPI), \_kitName(kitName) {}

**// It executes the in Smart Card template for reading**.

void read() {

cout<<"Read process is initializing... "<<endl;

\_tempAPI->readFunction();

}

**// It executes the in Smart Card template for writing.**

void write() {

cout<<"Write process is initializing... "<<endl;

\_tempAPI->writeFunction();

}

private:

AbstractTemplateAPI \* \_tempAPI;

string \_kitName;

};

**//Concrete ManageUSB Command Class to send read or write function commend to AbstractTemplateAPI class for Usb-Stick**

class ManageUSB : public Manage {

public:

ManageUSB(AbstractTemplateAPI \* tempAPI, string kitName) :\_tempAPI(tempAPI), \_kitName(kitName) {}

**// It executes the in Usb template for reaading.**

void read() {

cout<<"Read process is initializing... "<<endl;

\_tempAPI->readFunction();

}

**// It executes the in Usb template for writing**.

void write() {

cout<<"Write process is initializing... "<<endl;

\_tempAPI->writeFunction();

}

private:

AbstractTemplateAPI \* \_tempAPI;

string \_kitName;

};

**//ConnectionInterface is a abstract class to be adaptee Usb-Stick or Smart Card Reader for usb connection or smart card.**

class ConnectionInterface

{

public:

virtual void adapteeConnect() {};

virtual void applySituation(string situation) {};

};

**//Adaptee Class is a concrete class of ConnectionInterface. It adaptees to USB**

class USB :public ConnectionInterface

{

public:

USB() {}

**// It show the change of Usb-Stick.**

void adapteeConnect() { cout << "Connection has adapted to USB" << endl; }

private:

};

**//Adaptee Class is a concrete class of ConnectionInterface. It adaptees to SC**

class SC :public ConnectionInterface

{

public:

SC(){}

**// It show the change of Smard-Card.**

void adapteeConnect() { cout << "Connection has adapted to SC" << endl; }

};

**//KitAdaptor is a class. It converts and acquires connection between Usb-Sticks Smart-Card Reader and USB ,Smart Card**

class KitAdaptor

{

private:

ConnectionInterface \* \_adaptee;

public:

**//It connects the adaptee class object to convert Usb or Smard Card.**

ConnectionInterface \* getAdaptee() { return \_adaptee; }

**//It convert the tokens to Usb or Smard Card.**

KitAdaptor(ConnectionInterface \* adaptee) { \_adaptee = adaptee; connect(); }

**//It show the situation of adapter**.

void connect() { cout << "Adapter has initialized" << endl; \_adaptee->adapteeConnect(); } };

CollectionIterator::CollectionIterator(const Collection \*collection) :

\_collection(collection), \_current(0) {

}

void CollectionIterator::First() {

\_current = 0;

}

void CollectionIterator::Next() {

\_current++;

}

Port \*CollectionIterator::CurrentItem() const {

return (IsDone() ? NULL : \_collection->get(\_current));

}

bool CollectionIterator::IsDone() const {

return \_current >= \_collection->getCount();

}

/**/StickConnector class is a target class between adaptor and client.**

class StickConnector

{

public:

**// It decides the tokens in the ports If it is usb-stick or smard-card.**

StickConnector(DevelopmentKits \* kit) {

cout << "StickConnector has connected" << endl;

if (kit->getKitName() == "usb-stick")

{

KitAdaptor \* adapter = new KitAdaptor(new USB());

\_adapter=adapter;

}

else if (kit->getKitName() == "smard-card")

{

KitAdaptor \* adapter = new KitAdaptor(new SC());

\_adapter=adapter;

}

else

{

cout << "Nothing has initialized" << endl;

}

}

~StickConnector(){delete \_adapter;}

private:

KitAdaptor \* \_adapter;

};

**//DriveManager is a Facade Class. It converts one interface to another so that it matches what the client is expecting.**

class DriverManager {

public:

**//It initializes 4 ports to be ready to put in usb stick or smard card reader.**

DriverManager() {

aggregate = new Collection();

aggregate->add(new Port("Port1"));

aggregate->add(new Port("Port2"));

aggregate->add(new Port("Port3"));

aggregate->add(new Port("Port4"));

iterator = aggregate->CreateIterator();

}

**//It has a convert mechanism for usb-stick tokens to usb and smart-card reader tokens to smart card.**

void convertDeviceToSystem()

{

string kitname = searchKit();

if (kitname == "usb-stick")

{

\_temp = new USBTemplate();

\_manage = new ManageUSB(\_temp, "usb-stick");

}

else if (kitname == "smard-card")

{ \_temp = new SCTemplate();

\_manage = new ManageSC(\_temp, "smard-card");

}

else

{

cout << "There is no valid kit!!!" << endl;

}

\_connector = new StickConnector(\_kit);

}

**// It sends the situations of the process to Manage Class object to write or read the usb-stick or smart card.**

void process(string situation){

if(situation=="read")

{

\_manage->read();

}

else if(situation=="write")

{

\_manage->write();

}

}

**//It does the port in which can be used, ready to put the usb kit or smart card reader kit.**

void connectPort(AbstractIterator& i, string portName, DevelopmentKits \* kit) {

for (i.First(); !i.IsDone();i.Next()) {

if (i.CurrentItem()->getPortName() == portName)

{

if (kit->getKitName()=="usb-stick")

{

i.CurrentItem()->setPort(kit);

i.CurrentItem()->setUSBvalid(1);

cout<<kit->getKitName()<<" is inserted to "<<i.CurrentItem()->getPortName()<<endl;

break;

}

else if (kit->getKitName() == "smard-card")

{

i.CurrentItem()->setPort(kit);

i.CurrentItem()->setSCvalid(1);

cout<<kit->getKitName()<<" is inserted to "<<i.CurrentItem()->getPortName()<<endl;

break;

}

else

{

cout << "There is no valid kit!!!" << endl;

break;

}

}

}

}

**//It searchs all ports to find which ports have a connection with a smart-card or usb-stick**

string searchPortToKit(AbstractIterator& i) {

for (i.First(); !i.IsDone(); i.Next()) {

if (i.CurrentItem()->getUSBvalid() == 1)

{

return i.CurrentItem()->getPort()->getKitName();

}

else if (i.CurrentItem()->getSCvalid() == 1)

{

return i.CurrentItem()->getPort()->getKitName();

}

}

return "no-kit";

}

**//It deletes all ports in the system.**

void deletePorts(AbstractIterator& i) {

for (i.First(); !i.IsDone(); i.Next()) {

delete i.CurrentItem();

}

}

**//We select and put a usb kit or smart card reader in any port whatever we want.**

void setPort(string portname,DevelopmentKits \* kit) { \_kit = kit; connectPort(\*iterator, portname,kit); }

string searchKit() { return searchPortToKit(\*iterator); }

~DriverManager(){ deletePorts(\*iterator); delete \_kit,iterator,aggregate,\_manage,\_temp,\_connector; cout<<"All objects have been deleted"<<endl;}

private:

DevelopmentKits \* \_kit;

AbstractIterator \* iterator;

AbstractAggregate \* aggregate;

Manage \* \_manage;

AbstractTemplateAPI \* \_temp;

StickConnector \* \_connector;

};

int main()

{

DriverManager \* drive = new DriverManager();

DevelopmentKits \* kit = new ScKit();

DevelopmentKits \* kit2 = new UsbKit();

drive->setPort("Port3",kit);

drive->convertDeviceToSystem();

cout<<endl;

cout<<endl;

drive->process("read");

cout<<endl;

cout<<endl;

drive->process("write");

cout<<endl;

cout<<endl;

drive->setPort("Port2",kit2);

drive->convertDeviceToSystem();

cout<<endl;

cout<<endl;

drive->process("read");

cout<<endl;

cout<<endl;

drive->process("write");

delete drive;

return 0;

}