**Book – Time Project**

**Class 3 – Time.cpp:**

//Time class member-function definitions

#include "stdafx.h"

#include <iomanip> //for setw and setfill stream manipulators

#include <stdexcept> //for invalid\_argument exception class

#include <sstream> //for ostringstream class

#include <string>

#include "Time.h" //include definition of class Time from Time.h

//set new Time value using universal interval

void Time::setTime(int h, int m, int s) {

//validate hour, minute and second

if ((h>=0&&h<24)&&(m>=0&&m<60)&&(s>=0&&s<60)) {

hour = h;

minute = m;

second = s;

}

else {

throw std::invalid\_argument("hour, minute and/or second was out of range");

}

}

//return Time as a string in universal-time format (HH:MM:SS)

std::string Time::toUniversalString() const {

std::ostringstream output;

output << std::setfill('0') << std::setw(2) << hour << ":"

<< std::setw(2) << minute << ":" << std::setw(2) << second;

return output.str(); //returns the formatting string

}

//return Time as a string in standard-time format (HH:MM:SS AM or PM)

std::string Time::toStandardString() const {

std::ostringstream output;

output << ((hour == 0 || hour == 12) ? 12 : (hour % 12)) << ":"

<< std::setfill('0') << std::setw(2) << minute << ":" << std::setw(2)

<< second << (hour < 12 ? " AM" : " PM");

return output.str(); //returns the formatting string

}

**Class 2 – Time.h:**

#include <string>

//prevent mulptiple inclusions of header

#ifndef TIME\_H

#define TIME\_H

//Time class definition

class Time

{

public:

void setTime(int, int, int);

std::string toUniversalString() const; //24-hour time format string

std::string toStandardString() const; //12-hour time format string

private:

unsigned int hour{0}; //0-23

unsigned int minute{0}; //0-59

unsigned int second{0}; //0-59

};

#endif

**Class 1 – ConsoleApplication5.cpp:**

#include "stdafx.h"

#include <iostream>

#include <stdexcept> //invalid\_argument exception class

#include "Time.h"

// displays a Time in 24-hour and 12-hour formats

void displayTime(const std::string& message, const Time& time) {

std::cout << message << "\nUniversal time: " << time.toUniversalString()

<< "\nStandard time: " << time.toStandardString() << "\n\n";

}

int main()

{

Time t;

displayTime("Initial time:", t); //display t's initial value

t.setTime(13, 27, 6); //change time

displayTime("After setTime:", t);

//attempt to set the time with invalid values

try {

t.setTime(99, 99, 99);

}

catch (std::invalid\_argument& e) {

std::cout << "Exception: " << e.what() << "\n\n";

}

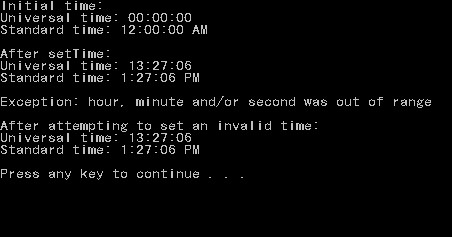
//display t's value after attempting to set an invalid time

displayTime("After attempting to set an invalid time:", t);

return 0;

}

**Result:**



**Important notes:**

* Always use guard – **#ifndef**, **#define** and **#endif** – in your header classes to not include in other classes more than once
* Remember to catch exceptions by reference to improve performance
* Look below to learn about the **<sstream>** library

#include "stdafx.h"

#include <iostream>

#include <sstream>

#include <fstream>

int main()

{

std::stringstream ss;

ss << 1234 << " text " << 'c';

std::cout << "Printing out ss: " << ss.str() << std::endl;

ss.str(""); //resets string

ss.clear(); //resets errors

std::cout << "Printing out ss: " << ss.str() << std::endl;

std::string line{" "};

std::fstream my\_stream;

std::cout << "\nOpening Information.txt:" << "\n";

my\_stream.open("Information.txt");

while (getline(my\_stream, line)) {

ss << line << "\n";

}

my\_stream.close();

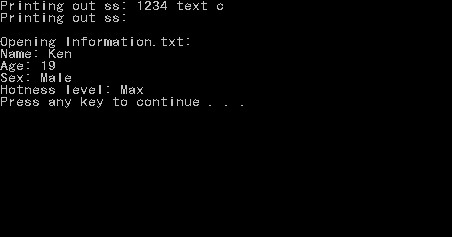
std::cout << ss.str();

system("pause");

return 0;

}

**Result:**



**Important notes:**

* To clear the string stream, we used ‘**ss.str(“”)**’, and to reset all the errors (or whatever the heck that means) we used ‘**ss.clear()**’