# Assignment Description

Modify the Stereo Receiver class created in Module 1. If the user attempts to create or modify a stereo receiver object when any of the values passed are invalid, an invalid argument exception should be thrown with appropriate text:

* Manufacturer, Model, or Serial Number not provided
* Inappropriate values for Wattage, Number of Channels, Band , Frequency , Volume, or Power

The constructor and mutator methods should throw invalid parameter exceptions if invalid parameters are provided.

Create a program that utilizes this class to create and display stereo receivers based on user input. Utilize try-catch blocks to validate the user input. Ensure that the exception thrown includes appropriate text to describe the problem (e.g. “Invalid Value – volume cannot exceed 10” or “Invalid Frequency”). Demonstrate usage of the class and its embedded exception handling in a program that prompts users for initial values, creates a receiver object, and then prompts the user to change the various values.

# 1 Readme Documentation

This program allows the user to create a stereo using the stereoReciever class. The user will be asked for manufacturer, model, serial number, wattage, and number of channels. Then, the user can modify any of these values as well as current band, volume, bass, treble, and radio frequency. If the user inputs an invalid value, an exception will be thrown and an error message will be displayed.

# 2 Flowchart Screen Shots

# 3 UML and Use Case Diagrams

# 4 Source Code of All files (.h, .cpp)

#include *<iostream>*

#include *<iomanip>*

#include *<string>*

#include *<cctype>*

#include *<stdexcept>*

**using** **namespace** **std**;

*/\**

*Name: Stereo Reciever with Exception Handling*

*Author: Wesley Hixon*

*Date Last Updated: 11/03/2024*

*Purpose: User can create a stereo by inputting*

*stereo characteristics which will be stored in stereoReciever object.*

*Error handling is now implemented using try/catch blocks and exception classes.*

*\*/*

*// Stereo Class*

**class** **stereoReciever**{

**private**:

string manufacturer;

string model;

int serialNumber;

int wattage;

int numChannels;

string band = "AM";

double fmFrequency = 100.1;

int amFrequency = 1000;

int volume = 5;

bool power = false;

int bass = 0;

int treble = 0;

**public**:

*// Setter Methods*

void setManufacturer(string manufacturerInput){

*// No try/catch because manufacturer string can be anything*

manufacturer = manufacturerInput;

}

void setModel(string modelInput){

*// No try/catch because model string can be anything*

model = modelInput;

}

void setSerialNum(int serialNumInput){

**try**{

**if**(serialNumInput < 1) **throw**(invalid\_argument("Invalid Value - Serial Number cannot be negative"));

serialNumber = serialNumInput;

}

*// Error is rethrown to be handled further up the chain*

*// This is the same for all further setter methods*

**catch**(**const** invalid\_argument& e){

**throw** e;

}

}

void setWattage(int wattageInput){

**try**{

*// Check wattage greater than 0*

**if**(wattageInput < 1) **throw**(invalid\_argument("Invalid Value - Wattage must be greater than 0"));

wattage = wattageInput;

}

*// Throw error further up the chain*

**catch**(**const** invalid\_argument& e){

**throw** e;

}

}

void setNumChannels(int numChannelsInput){

**try**{

*// Check num channels greater than 1*

**if**(numChannelsInput < 1) **throw**(invalid\_argument("Invalid Value - Number of channels must be at least 1"));

numChannels = numChannelsInput;

}

*// Throw error further up the chain*

**catch**(**const** invalid\_argument& e){

**throw** e;

}

}

void setFMFrequency(double frequencyInput){

**try**{

*// Check FM frequency is within the valid range for US regulations*

**if**(frequencyInput < 88 || frequencyInput > 108) **throw**(invalid\_argument("Invalid Value - FM Frequency must be between 88 and 108 mHz"));

fmFrequency = frequencyInput;

}

*// Throw error further up the chain*

**catch**(**const** invalid\_argument& e){

**throw** e;

}

}

void setAMFrequency(int frequencyInput){

**try**{

*// Check AM frequency is within the valid range for US regulations*

**if**(frequencyInput < 540 || frequencyInput > 1700) **throw**(invalid\_argument("Invalid Value - AM Frequency must be between 540 and 1700 kHz"));

amFrequency = frequencyInput;

}

*// Throw error further up the chain*

**catch**(**const** invalid\_argument& e){

**throw** e;

}

}

void setVolume(int volumeInput){

**try**{

*// Check volume between 0 and 10*

**if**(volumeInput > 10 || volumeInput < 0) **throw**(invalid\_argument("Invalid Value - Volume must be between 0 and 10"));

volume = volumeInput;

}

*// Throw error further up the chain*

**catch**(**const** invalid\_argument& e){

**throw** e;

}

}

void setBass(int bassInput){

**try**{

*// Check bass between -5 and 5*

**if**(bassInput < -5 || bassInput > 5) **throw**(invalid\_argument("Invalid Value - Bass must be between -5 and 5"));

bass = bassInput;

}

*// Throw error further up the chain*

**catch**(**const** invalid\_argument& e){

**throw** e;

}

}

void setTreble(int trebleInput){

**try**{

*// Check treble between -5 and 5*

**if**(trebleInput < -5 || trebleInput > 5) **throw**(invalid\_argument("Invalid Value - Treble must be between -5 and 5"));

treble = trebleInput;

}

*// Throw error further up the chain*

**catch**(**const** invalid\_argument& e){

**throw** e;

}

}

*// Switches between AM and FM*

void changeBand(){

**if**(band == "AM"){

band = "FM";

}

**else**{

band = "AM";

}

}

void togglePower(){

**if**(power){

power = false;

}

**else**{

power = true;

}

}

*// Getter Methods*

string getManufacturer(){

**return** manufacturer;

}

string getModel(){

**return** model;

}

int getSerialNum(){

**return** serialNumber;

}

int getWattage(){

**return** wattage;

}

int getNumChannels(){

**return** numChannels;

}

string getBand(){

**return** band;

}

double getFMFrequency(){

**return** fmFrequency;

}

int getAMFrequency(){

**return** amFrequency;

}

int getVolume(){

**return** volume;

}

bool getPower(){

**return** power;

}

int getBass(){

**return** bass;

}

int getTreble(){

**return** treble;

}

*// Default constructor just initializes with no default values*

stereoReciever(){

}

stereoReciever(string manufacturerInput, string modelInput, int serialNumberInput, int wattageInput, int numChannelsInput){

*// Set values user provided*

**try**{

setManufacturer(manufacturerInput);

setModel(modelInput);

setSerialNum(serialNumberInput);

setWattage(wattageInput);

setNumChannels(numChannelsInput);

*// Everything else is default*

}

**catch**(**const** exception& e){

**throw** e;

}

}

};

*// Function prototypes*

stereoReciever buildNewStereo();

void displayStereo(stereoReciever stereo);

void changeBand(stereoReciever& stereo);

bool getBoolInput(string prompt);

void changeFrequency(stereoReciever& stereo);

void changeVolume(stereoReciever& stereo);

void changeBass(stereoReciever& stereo);

void changeTreble(stereoReciever& stereo);

int main(){

stereoReciever userStereo; *// Initializing stereo*

cout << "Welcome to the stereo reciever simulation!" << endl

<< "Please create your new stereo reciever." << endl;

*// Building stereo*

userStereo = buildNewStereo();

*// Turn stereo on*

userStereo.togglePower();

bool menu = true;

bool valid;

*// Menu where you can edit stereo attributes*

**while**(menu){

int menuChoice;

cout << endl << "Input a number between 1 and 7 to select any of the following options:" << endl

<< "1. View Stereo Attrubutes" << endl

<< "2. Change Band (AM / FM)" << endl

<< "3. Change Radio Frequency" << endl

<< "4. Change Volume" << endl

<< "5. Change Bass" << endl

<< "6. Change Treble" << endl

<< "7. Power Off" << endl;

valid = false;

*// Get menu choice*

**while**(!valid){

**try**{

**if**(!(cin >> menuChoice) || (menuChoice < 1 && menuChoice > 7)){

cin.clear();

cin.ignore(10000, '\n');

**throw**(runtime\_error("Invalid input. Please enter a number between 1 and 7."));

}

valid = true;

}

**catch**(**const** exception& e){

cerr << endl << e.what() << endl;

}

}

*// Switch statement for menu*

**switch**(menuChoice){

**case** 1:

displayStereo(userStereo);

**break**;

**case** 2:

changeBand(userStereo);

**break**;

**case** 3:

*// Change frequency*

changeFrequency(userStereo);

**break**;

**case** 4:

*// change Volume*

changeVolume(userStereo);

**break**;

**case** 5:

*// Change bass*

changeBass(userStereo);

**break**;

**case** 6:

*// Change treble*

changeTreble(userStereo);

**break**;

**case** 7:

bool input = getBoolInput("Are you sure you would like to exit? Type (y/n)");

**if**(input == true){

userStereo.togglePower();

menu = false;

}

**break**;

}

}

**return** 0;

}

*// Prompts for characteristics of a new stereo and returns it*

stereoReciever buildNewStereo(){

stereoReciever newStereo;

*// Get manufacturer*

cout << "Who is the manufacturer of your new stereo? E.g. Yamaha, JBL, Bose, etc." << endl;

bool valid = false;

string manufacturer;

**while**(!valid){

**try**{

**if**(!(cin >> manufacturer)){

cin.clear();

cin.ignore(10000, '\n');

**throw**(runtime\_error("Invalid input. Please try again."));

}

newStereo.setManufacturer(manufacturer);

valid = true;

}

**catch**(**const** exception& e){

cerr << endl << e.what() << endl;

}

}

*// Get model*

cout << "Enter the model of your new stereo." << endl;

valid = false;

string model;

**while**(!valid){

**try**{

**if**(!(cin >> model)){

cin.clear();

cin.ignore(10000, '\n');

**throw**(runtime\_error("Invalid input. Please try again."));

}

newStereo.setModel(model);

valid = true;

}

**catch**(**const** exception& e){

cerr << endl << e.what() << endl;

}

}

*// Get serial number*

cout << "Enter the serial number of your new stereo." << endl;

int serialNum;

valid = false;

**while**(!valid){

**try**{

**if**(!(cin >> serialNum)){

cin.clear();

cin.ignore(10000, '\n');

**throw**(runtime\_error("Invalid input. Please try again."));

}

newStereo.setSerialNum(serialNum);

valid = true;

}

**catch**(**const** exception& e){

cerr << endl << e.what() << endl;

}

}

*// Get wattage*

cout << "Enter the wattage of your new stereo." << endl;

int wattage;

valid = false;

**while**(!valid){

**try**{

**if**(!(cin >> wattage)){

cin.clear();

cin.ignore(10000, '\n');

**throw**(runtime\_error("Invalid input. Please try again."));

}

newStereo.setWattage(wattage);

valid = true;

}

**catch**(**const** exception& e){

cerr << endl << e.what() << endl;

}

}

*// Get number of channels*

cout << "Enter the number of channels for your new stereo." << endl;

int numChannels;

valid = false;

**while**(!valid){

**try**{

**if**(!(cin >> numChannels)){

cin.clear();

cin.ignore(10000, '\n');

**throw**(runtime\_error("Invalid input. Please try again."));

}

newStereo.setNumChannels(numChannels);

valid = true;

}

**catch**(**const** exception& e){

cerr << endl << e.what() << endl;

}

}

*// return new stereo*

**return** newStereo;

}

*// Displays all stereo characteristics*

void displayStereo(stereoReciever stereo){

cout << endl << "This is your current stereo's attributes:" << endl

<< "Manufacturer: " << stereo.getManufacturer() << endl

<< "Model: " << stereo.getModel() << endl

<< "Serial Number: " << stereo.getSerialNum() << endl

<< "Wattage: " << stereo.getWattage() << endl

<< "Number of Channels: " << stereo.getNumChannels() << endl

<< "Current Band: " << stereo.getBand() << endl

<< "Current Volume: " << stereo.getVolume() << endl

<< "Current Bass Level: " << stereo.getBass() << endl

<< "Current Treble Level: " << stereo.getTreble() << endl;

**return**;

}

*// Changes from AM to FM and vice versa*

void changeBand(stereoReciever& stereo){

string band = stereo.getBand();

cout << endl << "The current band is set to " << band << endl;

bool input = getBoolInput("Would you like to switch bands? (Type y/n)");

**if**(input == true){

stereo.changeBand();

}

}

*// Gets a yes or no input given a prompt*

bool getBoolInput(string prompt){

bool valid = false;

char input;

cout << endl << prompt;

**while**(!valid){

**try**{

**if**(!(cin >> input) || (tolower(input) != 'y' && tolower(input) != 'n')){

cin.clear();

cin.ignore(100000, '\n');

**throw**(runtime\_error("Invalid input. Please enter y or n."));

}

valid = true;

}

**catch**(**const** exception& e){

cerr << endl << e.what() << endl;

}

}

**if**(tolower(input) == 'y'){

**return** true;

}

**else**{

**return** false;

}

}

*// Changes the AM/FM frequency, depending on which one is enabled*

void changeFrequency(stereoReciever& stereo){

double frequency;

*// Gets the frequency which corresponds to the radio band*

**if**(stereo.getBand() == "AM"){

frequency = stereo.getAMFrequency();

}

**else**{

frequency = stereo.getFMFrequency();

}

*// Outputting current frequency*

cout << endl << stereo.getBand() << " radio frequency is currently set to " << frequency << endl;

bool input = getBoolInput("Would you like to change it? (Type y/n)");

**if**(input == false){

**return**;

}

bool valid = false;

double frequencyInput;

*// Getting desired frequency*

cout << "Enter your desired " << stereo.getBand() << " radio frequency: ";

**while**(!valid){

**try**{

**if**(!(cin >> frequencyInput)){

cin.clear();

cin.ignore(10000, '\n');

**throw**(runtime\_error("Invalid input. Please try again."));

}

**if**(stereo.getBand() == "AM"){

stereo.setAMFrequency(frequencyInput);

}

**else**{

stereo.setFMFrequency(frequencyInput);

}

valid = true;

}

**catch**(**const** exception& e){

cerr << endl << e.what() << endl;

}

}

*// Changing frequency on stereo*

}

*// Prompt to change volume*

void changeVolume(stereoReciever& stereo){

*// Outputting current volume*

cout << "The volume is currently set to " << stereo.getVolume() << "." << endl;

bool input = getBoolInput("Would you like to change it? (Type y/n) ");

**if**(input == false){

**return**;

}

*// Getting desired volume*

cout << "Enter your desired volume: ";

int volumeInput;

bool valid = false;

**while**(!valid){

**try**{

**if**(!(cin >> volumeInput)){

cin.clear();

cin.ignore(10000, '\n');

**throw**(runtime\_error("Invalid input. Please try again."));

}

stereo.setVolume(volumeInput);

valid = true;

}

**catch**(**const** exception& e){

cerr << endl << e.what() << endl;

}

}

}

*// Prompt to change bass*

void changeBass(stereoReciever& stereo){

*// Outputting current bass level*

cout << "The bass level is currently set to " << stereo.getBass() << "." << endl;

bool input = getBoolInput("Would you like to change it? (Type y/n) ");

**if**(input == false){

**return**;

}

*// Getting desired bass level*

cout << "Enter your desired bass level between -5 and 5: ";

int bassInput;

bool valid = false;

**while**(!valid){

**try**{

**if**(!(cin >> bassInput)){

cin.clear();

cin.ignore(10000, '\n');

**throw**(runtime\_error("Invalid input. Please try again."));

}

stereo.setBass(bassInput);

valid = true;

}

**catch**(**const** exception& e){

cerr << endl << e.what() << endl;

}

}

}

*// Prompt to change treble*

void changeTreble(stereoReciever& stereo){

*// Outputting current treble level*

cout << "The treble level is currently set to " << stereo.getTreble() << "." << endl;

bool input = getBoolInput("Would you like to change it? (Type y/n) ");

**if**(input == false){

**return**;

}

*// Getting desired treble level*

cout << "Enter your desired treble level between -5 and 5: ";

int trebleInput;

bool valid = false;

**while**(!valid){

**try**{

**if**(!(cin >> trebleInput)){

cin.clear();

cin.ignore(10000, '\n');

**throw**(runtime\_error("Invalid input. Please try again."));

}

stereo.setTreble(trebleInput);

valid = true;

}

**catch**(**const** exception& e){

cerr << endl << e.what() << endl;

}

}

}

# 5 Three Use Case Screen Shots







