# UML:

Timeline

Description automatically generated

# Screenshots:

Text

Description automatically generated

Contents

[UML: 1](#_Toc68513555)

[Screenshots: 1](#_Toc68513556)

[Source.cpp 2](#_Toc68513557)

[Pirate.h 3](#_Toc68513558)

[Pirate.cpp 3](#_Toc68513559)

[CaptainPirate.h 4](#_Toc68513560)

[CaptainPirate.cpp 5](#_Toc68513561)

[ZombiePirate.h 5](#_Toc68513562)

[ZombiePirate.cpp 6](#_Toc68513563)

**Code:**

# Source.cpp

/\*

Edward Alvarado

Class: CIS247C

Date: 4/4/2021

Week 6 Lab

\*/

#include <iostream>

#include <string>

#include <conio.h>

#include "ZombiePirate.h"

#include "CaptainPirate.h"

using namespace std;

void displayPirate(Pirate\* ptrPirate);

/// Entry point to the applicatrion

int main()

{

#if defined(DEBUG) | defined(\_DEBUG)

\_CrtSetDbgFlag(\_CRTDBG\_ALLOC\_MEM\_DF | \_CRTDBG\_LEAK\_CHECK\_DF);

#endif

Pirate\* pirates[3];

pirates[0] = new CaptainPirate("Jack", "Sparrow");

pirates[1] = new CaptainPirate("Red Beard", "Gum Gum");

pirates[2] = new ZombiePirate("Bob", 5);

for (int i = 0; i < 3; i++)

cout << pirates[i]->speak() << endl;

for (int i = 0; i < 3; i++)

{

delete pirates[i];

}

// Pause

cout << "\nPress any key to continue...";

\_getch();

return 0;

}

void displayPirate(Pirate\* ptrPirate)

{

CaptainPirate\* cp = dynamic\_cast<CaptainPirate\*>(ptrPirate); // NULL if it does not work

if (cp != NULL)

{

cout << "Pet: " << cp->getPet() << endl;

}

}

# Pirate.h

#pragma once

#include <string>

using namespace std;

class Pirate

{

protected:

string name;

public:

Pirate();

Pirate(string name);

virtual ~Pirate();

virtual string speak() = 0; // pure virtual method -- abstract method

virtual string toString();

string getName();

void setName(string name);

};

# Pirate.cpp

#include "Pirate.h"

using namespace std;

Pirate::Pirate()

{

name = "unknown";

}

Pirate::Pirate(string name)

{

setName(name);

}

Pirate::~Pirate(){}

string Pirate::toString()

{

return "Name: " + name;

}

string Pirate::getName()

{

return name;

}

void Pirate::setName(string name)

{

if (name.length() > 0)

this->name = name;

else

this->name = "unknown";

}

# CaptainPirate.h

#pragma once

#include "Pirate.h"

#include <string>

using namespace std;

class CaptainPirate :

public Pirate

{

private:

string pet;

public:

CaptainPirate();

CaptainPirate(string name, string pet);

~CaptainPirate();

string speak();

string toString();

string getPet();

void setPet(string pet);

};

# CaptainPirate.cpp

#include "CaptainPirate.h"

CaptainPirate::CaptainPirate()

{

name = "unknown";

pet = "unknown";

}

CaptainPirate::CaptainPirate(string name, string pet):Pirate(name)

{

setName(name);

setPet(pet);

}

CaptainPirate::~CaptainPirate(){}

string CaptainPirate::speak()

{

return "Yaaarrr! It be a " + pet + "! \nYaarrr Scallywags! Swab that poop deck!";

}

string CaptainPirate::toString()

{

return Pirate::toString() + ", pet: " + pet;

}

string CaptainPirate::getPet()

{

return pet;

}

void CaptainPirate::setPet(string pet)

{

if (pet.length() > 0)

this->pet = pet;

else

this->pet = "unknown";

}

# ZombiePirate.h

#pragma once

#include "Pirate.h"

#include <string>

using namespace std;

class ZombiePirate :

public Pirate

{

private:

short brainHunger;

public:

ZombiePirate(void);

ZombiePirate(string name, short brainHunger);

~ZombiePirate(void);

string speak(void);

string toString(void);

short getBrainHunger(void);

void setBrainHunger(short brainHunger);

};

# ZombiePirate.cpp

#include "ZombiePirate.h"

#include <string>

#include <sstream>

using namespace std;

ZombiePirate::ZombiePirate()

{

name = "unknown";

brainHunger = 0;

}

ZombiePirate::ZombiePirate(string name, short brainHunger):Pirate(name)

{

setName(name);

setBrainHunger(brainHunger);

}

ZombiePirate::~ZombiePirate(void){}

string ZombiePirate::speak(void)

{

// say something based on the hunger level

switch (brainHunger)

{

case 0:

return "Yum. I just ate a brain!";

break;

case 1:

return "I'm getting a little hungry... Are there any brains out there?";

break;

case 2:

return "My tummy is rumbling... anyone have some brains?";

break;

case 3:

return "I'm getting very hungry!! I need a brain to eat!!";

break;

case 4:

return "This is poor service! Zero brains! 1 star Yelp review!";

break;

case 5:

return "BRAINS!!!! GIVE ME BRAINS NOW!!! I NEED TO EAT BRAINS NOW!!!";

break;

default:

return "Error. Something went wrong.";

break;

}

}

string ZombiePirate::toString(void)

{

stringstream ss;

ss << "Brain Hunger: " << brainHunger << ", " << Pirate::toString();

return ss.str();

}

short ZombiePirate::getBrainHunger(void)

{

return brainHunger;

}

void ZombiePirate::setBrainHunger(short brainHunger)

{

if (brainHunger > 0)

this->brainHunger = brainHunger;

else

this->brainHunger = 0;

}