САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Домашнее задание

«Реализация программной модели инфокоммуникационной системы»

Выполнил: Мороз И.О.

Группа К3120

Проверил: Осипов Н.А.

Санкт-Петербург

2018 г.

# Задача

Создать программное обеспечение системы обработки данных.

Было решено создать программу для учёта животных в ветеринарной клинике. Врач может добавлять и удалять животных из базы данных, изменять их свойства. Например, если животное выздоровело, то врач устанавливает ему флаг «Здоров». При этом интерфейс должын быть понятен конечному пользователю, чтобы UX был на максимальном уровне.

# Диаграмма классов

Так как все животные имеют схожие свойства, то для большей гибкости приложения будет создан абстактный класс Animal, от которого будут наследоваться все остальные классы животных.

Из-за сложностей работы с датами в C++ было решено создать свой класс для работы с датами Calendar. Объект данного класса будет хранить введённую дату и расчитывать время в годах до текущей даты.

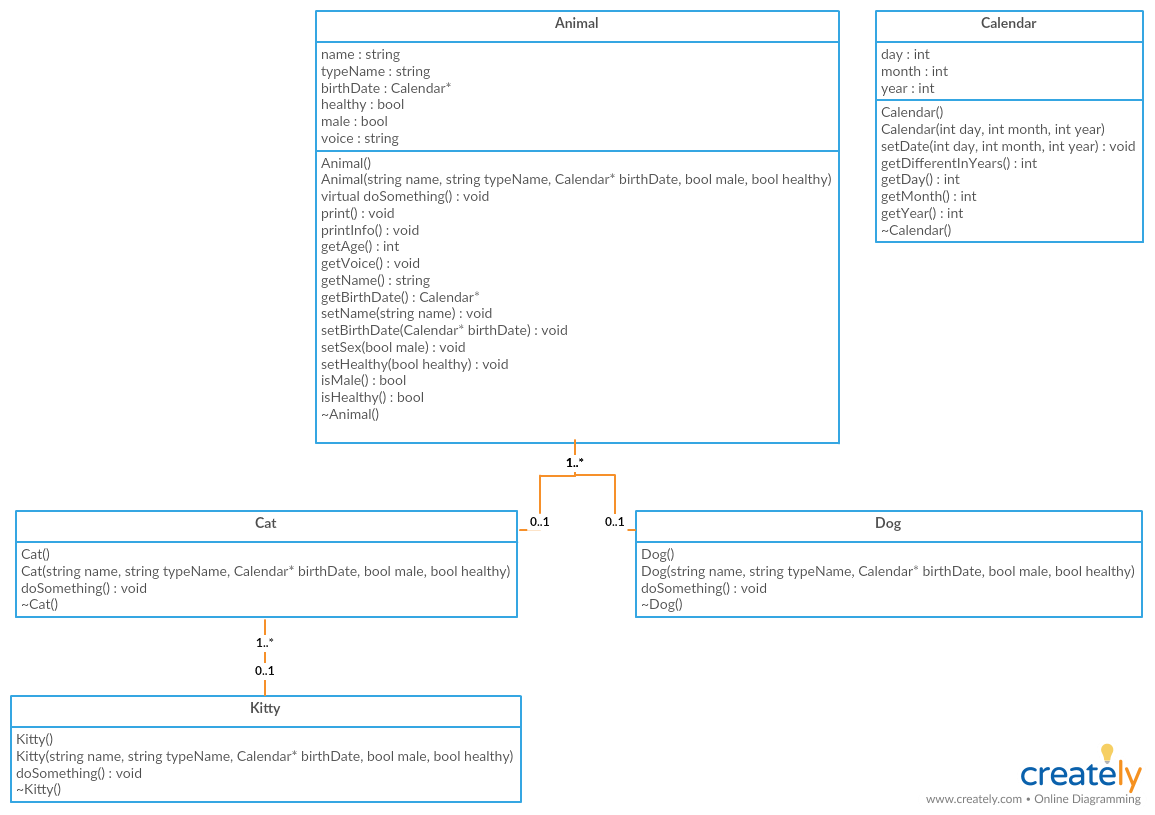


Рисунок 1 – диаграмма классов

На данный момент в программе представлены три вида животных: Cat и Dog, наследуемых от Animal, и Kitty, наследуемого от Cat. При этом класс Calendar имеет ассоциативные отношения с классом Animal.

# Класс Animal

Класс Animal является абстрактным классом. Поэтому в нём необходимо реализовать основные интерфейсы использования классов-наследников от данного класса. В основном это методы get и set (инкапсуляция), но также были реализованы классы для вывода информации об объекте класса в консоль print() и printInfo(). Листинг данного класса:

#include "Animal.h"

Animal::Animal() {

this->name = "";

this->birthDate = new Calendar();

this->healthy = true;

this->male = true;

this->voice = "";

}

Animal::Animal(string name, string typeName, Calendar\* birthDate, bool male, bool healthy) {

this->name = name;

this->typeName = typeName;

this->birthDate = birthDate;

this->male = male;

this->healthy = healthy;

}

void Animal::doSomething() {

}

void Animal::print() {

cout << "Name: " << this->name << endl;

cout << "Class: " << this->typeName << endl;

cout << "Age: " << this->getAge() << endl;

cout << "Sex: " << (this->male ? "male" : "female") << endl;

cout << "Healthy: " << (this->healthy ? "yes" : "no") << endl;

cout << "Voice: " << this->voice << endl;

this->doSomething();

}

void Animal::printInfo() {

cout << "Name: " << this->name << "; Age: " << this->getAge() << "; Healthy: " << (this->healthy ? "yes" : "no") << endl;

}

int Animal::getAge() {

int age = this->birthDate->getDifferentInYears();

return age;

}

void Animal::getVoice() {

cout << this->name << " say " << this->voice;

}

string Animal::getName() {

return this->name;

}

Calendar\* Animal::getBirthDate() {

return this->birthDate;

}

bool Animal::isMale() {

return this->male;

}

bool Animal::isHealthy() {

return this->healthy;

}

void Animal::setName(string name) {

this->name = name;

}

void Animal::setBirthDate(Calendar\* birthDate) {

this->birthDate = birthDate;

}

void Animal::setSex(bool male) {

this->male = male;

}

void Animal::setHealthy(bool healthy) {

this->healthy = healthy;

}

Animal::~Animal() {

}

# Класс Dog

Класс Dog является наследником класса Animal. Он использует все методы класса-родителя, при необходимости используя полиморфизм и переопределяя методы класса-родителя, в том числе виртуального метода doSomething(). При вызове конструктора Dog(), вызывается конструктор класса Animal(). Листинг данного класса:

#include "Dog.h"

Dog::Dog() : Animal() {

}

Dog::Dog(string name, string typeName, Calendar\* birthDate, bool male, bool healthy) : Animal(name, typeName, birthDate, male, healthy) {

this->voice = "Woof";

}

void Dog::doSomething() {

cout << this->getName() << " is barking on couch..." << endl;

}

Dog::~Dog() {

}

# Класс Cat

Класс Cat является наследником класса Animal. Он схож с классом Dog: происходит схожее переопределение методов класса-родителя (см. Класс Dog). Листинг данного класса:

#include "Cat.h"

Cat::Cat() : Animal() {

}

Cat::Cat(string name, string typeName, Calendar\* birthDate, bool male, bool healthy) : Animal(name, typeName, birthDate, male, healthy) {

this->voice = "Meow";

}

void Cat::doSomething() {

cout << this->getName() << " is sleeping..." << endl;

}

Cat::~Cat() {

}

# Класс Kitty

Данный класс является наследником класса Cat. При вызове конструктора Kitty() происходит вызов родительского класса Cat(), куда передаются аргументы. Также используется полиморфизм для метода doSomething(). Листинг данного класса:

#include "Kitty.h"

Kitty::Kitty()

{

}

Kitty::Kitty(string name, string typeName, Calendar\* birthDate, bool male, bool healthy) : Cat(name, typeName, birthDate, male, healthy) {

this->voice = "meow-meow";

}

void Kitty::doSomething() {

cout << this->getName() << " acting in video with kitties..." << endl;

}

Kitty::~Kitty()

{

}

# Класс Calendar

Класс Calendar используется для работы с датами. В конструкторе получает значения даты (день, месяц, год). Описан метод для изменения даты уже существующего объекта класса. Основным методом класса является метод getDifferentInYears(). Класс имеет ассоциативные отношения с классом Animal. Листинг данного класса:

#include "Calendar.h"

Calendar::Calendar()

{

}

Calendar::Calendar(int day, int month, int year)

{

this->day = day;

this->month = month;

this->year = year;

}

void Calendar::setDate(int day, int month, int year)

{

this->day = day;

this->month = month;

this->year = year;

}

int Calendar::getDifferentInYears() {

time\_t seconds = time(NULL);

tm\* timeinfo = localtime(&seconds);

Calendar \*currentDate = new Calendar(timeinfo->tm\_mday, timeinfo->tm\_mon + 1, timeinfo->tm\_year+1900);

int diffYear;

diffYear = currentDate->year - this->year;

if (currentDate->month - this->month < 0) {

diffYear--;

}

if ((currentDate->month - this->month <= 0) && (currentDate->day - this->day < 0)) {

diffYear--;

}

return diffYear;

}

Calendar::~Calendar()

{

}

# Исполняемый файл main

В файле были описаны методы для работы со списком указателей на объекты класса Animal:

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| void printMenu(); | Выводит в консоль меню |
| Animal\* createAnimal(); | Создает и возвращает указатель на объект класса Animal |
| void printAnimals(list<Animal\*> animals); | Выводит всех животных из списка |
| int printInfoAboutAnimal(list<Animal\*> animals); | Выводит информацию об одном определённом животном в подробном формате |
| void changeInfoAboutAnimal(list<Animal\*> animals); | Изменяет информацию об определённом животном |
| list<Animal\*> deleteAnimal(list<Animal\*> animals); | Удаляет определённое животное из списка |

Таблица 1 – перечисление методов исполняемого файла

Подробный листинг файла main.cpp:

#include "Animal.h"

#include "Cat.h"

#include "Kitty.h"

#include "Dog.h"

#include <iostream>

#include <list>

using namespace std;

void printMenu();

Animal\* createAnimal();

void printAnimals(list<Animal\*> animals);

int printInfoAboutAnimal(list<Animal\*> animals);

void changeInfoAboutAnimal(list<Animal\*> animals);

list<Animal\*> deleteAnimal(list<Animal\*> animals);

int main() {

system("mode con cols=56 lines=25");

list<Animal\*> animals;

while (true) {

printMenu();

int choice = 0;

cin >> choice;

if (choice == 0) break;

switch (choice)

{

case 1:

animals.push\_back(createAnimal());

break;

case 2:

printAnimals(animals);

system("pause");

break;

case 3:

printInfoAboutAnimal(animals);

system("pause");

break;

case 4:

changeInfoAboutAnimal(animals);

break;

case 5:

animals = deleteAnimal(animals);

break;

default:

break;

}

}

return 0;

}

void printMenu() {

system("cls");

cout << "~~~~~~~~~~~~~~~~~~~~VetHospital v1.0~~~~~~~~~~~~~~~~~~~~\n";

cout << "1 - add animal\n";

cout << "2 - print animals\n";

cout << "3 - get all info about animal\n";

cout << "4 - change info about animal\n";

cout << "5 - delete animal\n\n";

cout << "0 - exit\n";

}

Animal\* createAnimal() {

system("cls");

cout << "Create (1 - Cat, 2 - Dog, 3 - Kitty): ";

int userChoice;

try {

cin >> userChoice;

if (userChoice != 1 && userChoice != 2 && userChoice != 3) {

throw;

}

}

catch (exception ex) {

cout << "Error! Invalid input choice" << endl;

}

Animal \*animal = new Animal();

string name;

string typeName;

Calendar\* birthDate = new Calendar();

char birthDateS[100];

string sex;

bool isMale;

string healthy;

bool isHealthy;

cout << "Enter name: ";

cin >> name;

cout << "Enter birth date (DD.MM.YYYY): ";

try {

cin >> birthDateS;

int day;

int month;

int year;

sscanf\_s(birthDateS, "%d.%d.%d", &day, &month, &year);

if ((day < 1) || (day > 31) || (month < 1) || (month > 12)) {

throw;

}

birthDate->setDate(day, month, year);

}

catch (exception ex) {

cout << "Error! Invalid input date" << endl;

}

cout << "Enter sex (m/f): ";

try {

cin >> sex;

if ((sex != "m") && (sex != "f")) {

throw;

}

}

catch (exception ex) {

cout << "Error! Invalid input sex" << endl;

}

isMale = (sex == "m") ? true : false;

cout << "Is healthy (y/n)? ";

try {

cin >> healthy;

if ((healthy != "y") && (healthy != "n")) {

throw;

}

}

catch (exception ex) {

cout << "Error! Invalid input health" << endl;

}

isHealthy = (healthy == "y") ? true : false;

switch (userChoice)

{

case 1:

typeName = "Cat";

animal = new Cat(name, typeName, birthDate, isMale, isHealthy);

break;

case 2:

typeName = "Dog";

animal = new Dog(name, typeName, birthDate, isMale, isHealthy);

break;

case 3:

typeName = "Kitty";

animal = new Kitty(name, typeName, birthDate, isMale, isHealthy);

break;

default:

break;

}

return animal;

}

void printAnimals(list<Animal\*> animals) {

system("cls");

list<Animal\*>::iterator iter = animals.begin();

int i = 1;

while (iter != animals.end())

{

cout << i++ << ". ";

(\*iter++)->printInfo();

cout << "\n";

}

}

int printInfoAboutAnimal(list<Animal\*> animals) {

printAnimals(animals);

int number;

cout << "Enter animals number: ";

cin >> number;

list<Animal\*>::iterator animal = animals.begin();

std::advance(animal, --number);

Animal\* a = \*animal;

a->print();

return number;

}

void changeInfoAboutAnimal(list<Animal\*> animals) {

int number = printInfoAboutAnimal(animals);

list<Animal\*>::iterator animal = animals.begin();

std::advance(animal, number);

Animal\* a = \*animal;

int number1;

string name;

Calendar\* birthDate = new Calendar();

char birthDateS[100];

string sex;

string healthy;

cout << "Change: ";

cin >> number1;

system("cls");

switch (number1)

{

case 1:

cout << "Change name (" << a->getName() << "): ";

cin >> name;

a->setName(name);

break;

case 2:

cout << "You can't change it";

break;

case 3:

cout << "Enter birth date (DD.MM.YYYY): ";

try {

cin >> birthDateS;

int day;

int month;

int year;

sscanf\_s(birthDateS, "%d.%d.%d", &day, &month, &year);

if ((day < 1) || (day > 31) || (month < 1) || (month > 12)) {

throw;

}

birthDate->setDate(day, month, year);

a->setBirthDate(birthDate);

}

catch (exception ex) {

cout << "Error! Invalid input date" << endl;

}

break;

case 4:

cout << "Enter sex (m/f): ";

try {

cin >> sex;

if ((sex != "m") && (sex != "f")) {

throw;

}

a->setSex((sex == "m") ? true : false);

}

catch (exception ex) {

cout << "Error! Invalid input sex" << endl;

}

break;

case 5:

cout << "Is healthy (y/n)? ";

try {

cin >> healthy;

if ((healthy != "y") && (healthy != "n")) {

throw;

}

a->setHealthy((healthy == "y") ? true : false);

}

catch (exception ex) {

cout << "Error! Invalid input health" << endl;

}

break;

case 6:

cout << "You can't change it";

break;

default:

break;

}

}

list<Animal\*> deleteAnimal(list<Animal\*> animals) {

int number = printInfoAboutAnimal(animals);

string sure;

cout << "Are you sure? (y/n): ";

try {

cin >> sure;

if ((sure != "y") && (sure != "n")) {

throw;

}

if (sure == "y") {

list<Animal\*>::iterator it = animals.begin();

advance(it, number);

animals.erase(it);

}

return animals;

}

catch (exception ex) {

cout << "Error! Invalid input health" << endl;

}

}

При получении некоторой информации из консоли был использован обработчик исключений. С его помощью проверяется введённая пользователем информация на корректность.

# Алгоритм работы программы

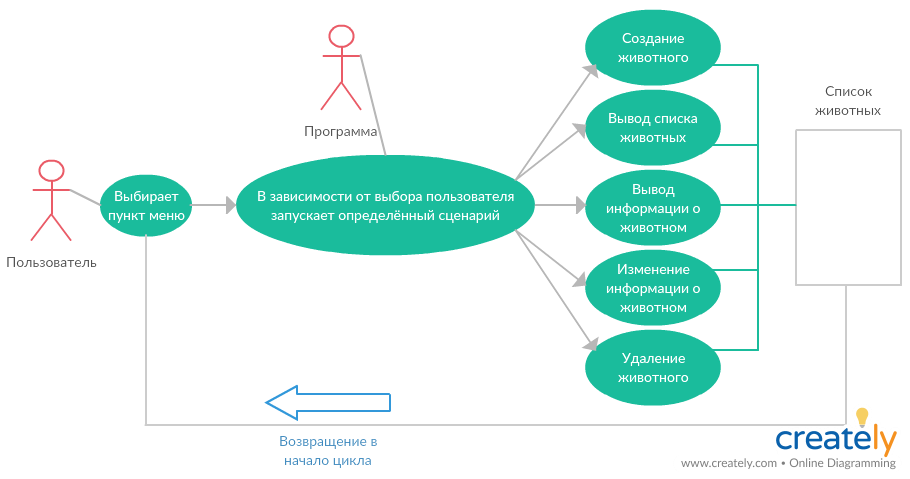


Рисунок 2 – описание алгоритма работы программы

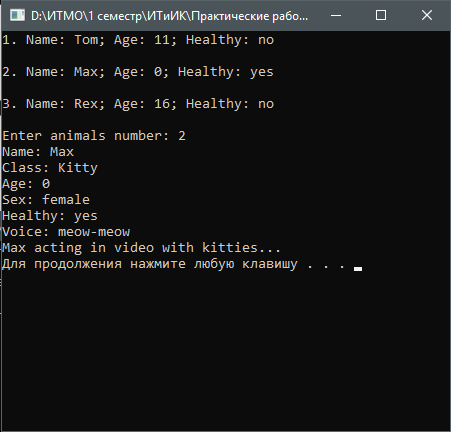
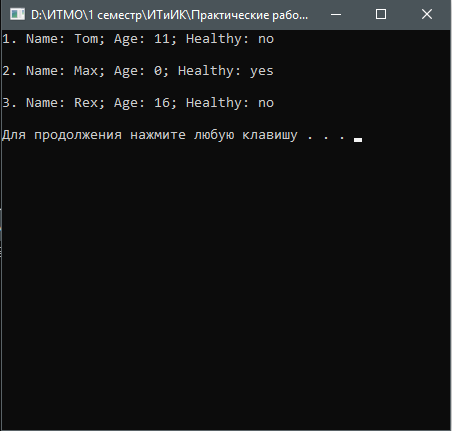
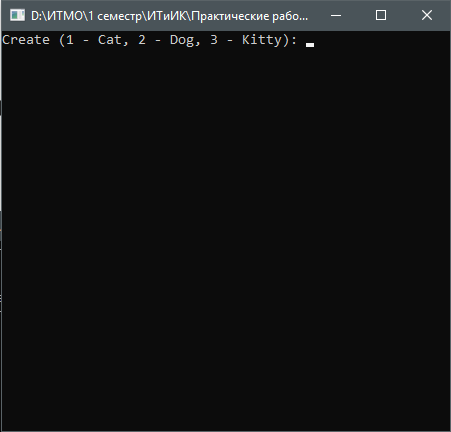
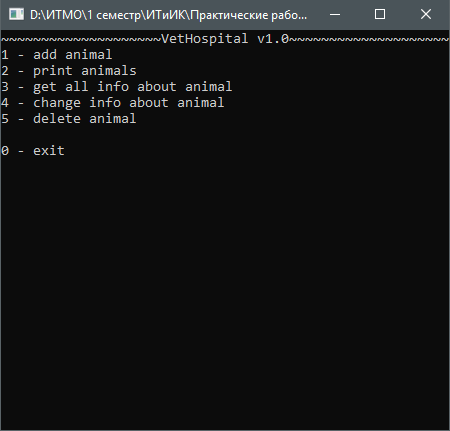


Рисунок 3, 4, 5, 6 – пример работы программы

# Вывод

В ходе выполнения данного домашнего задания были использованы на практике знания, полученные при выполнении практических заданий. Программа основана на базовых понятиях ООП (инкапсуляция, наследование, полиморфизм). Для обработки исключений была использована конструкция try-catch. Кроме использования наследования в программе также реализованы ассоциативные отношения между классами (Calendar – Animal). Полученная инфокоммуникационная система может быть применена в ветеринарных клиниках для облегчения работы со списками животных, находящихся в клинике. В будущем планируется добавление большего кол-ва видов животных и написание GUI-интерфейса.

# Контрольные вопросы