|  |  |
| --- | --- |
| Институт (факультет) | Институт информационных технологий |
| Кафедра | Кафедра математического и программного обеспечения ЭВМ |

# РАЗРАБОТКА АБСТРАКТНЫХ ТИПОВ ДАННЫХ

|  |  |
| --- | --- |
| **Дисциплина:** | ООП |
| **Темы:** | **Реализация взаимодействия между классами через компонентные функции** |

**Среда разработки:** Microsoft Visual Studio

**Язык программирования:** C++

**Тип проекта:** Консольное приложение

**ТРЕБОВАНИЯ К РАЗРАБОТКЕ**

1. Каждый класс должен быть оформлен в отдельных файлах: заголовочный (.h) и файл с кодом (.cpp).
2. Запрещается использовать обработку исключительных ситуаций и генерировать исключения.
3. Придерживайтесь принципа DRY (Don’t repeat yourself).
4. Обязательно наличие комментариев.

**Файл source.cpp**

**#include** "airTransport.h"

#include "List.h"

#include "List.cpp"

#include <iostream>

using namespace std;

enum class State {

Happy,

Peace,

Stress,

Depresion

};

class Programmer;

class Computer;

class Programmer {

private:

State mood;

double alertness;

public:

Programmer(State mood, double alertness) {

if (alertness < 0) {

this->alertness = 0;

}

this->alertness = alertness;

this->mood = mood;

}

State getMood() {

return mood;

}

double getAlertness() {

return alertness;

}

void setMood(State state) {

mood = state;

}

void setAlertness(double alertness) {

if (alertness < 0) {

cout << "Бдительность не может быть отрицательной!" << endl;

return;

}

this->alertness = alertness;

}

bool startWork(Computer& computer);

int fixBug(Computer& computer, int countFixedBugs);

};

class Computer {

private:

bool isWork;

int countBug;

public:

Computer() {

isWork = false;

countBug = 0;

}

bool getIsWork() {

return isWork;

}

void setIsWork(bool isWork) {

this->isWork = isWork;

}

int getCountBug() {

return countBug;

}

void setCountBug(int count) {

if (countBug < 0) {

cout << "Бдительность не может быть отрицательной!" << endl;

return;

}

countBug = count;

}

int working(Programmer& programmer);

State findBug(Programmer& programmer);

};

void StateToString(State mood) {

switch (mood)

{

case State::Happy:

cout << "Happy" << endl;

break;

case State::Peace:

cout << "Peace" << endl;

break;

case State::Stress:

cout << "Stress" << endl;

break;

case State::Depresion:

cout << "Depresion" << endl;

break;

default:

break;

}

}

bool Programmer::startWork(Computer& computer) {

computer.setIsWork(true);

return computer.getIsWork();

}

int Programmer::fixBug(Computer& computer, int countFixedBugs) {

computer.setCountBug(computer.getCountBug() - countFixedBugs);

return computer.getCountBug();

}

int Computer::working(Programmer& programmer) {

programmer.setAlertness(programmer.getAlertness() - 1);

return programmer.getAlertness();

}

State Computer::findBug(Programmer& programmer) {

if (countBug > 5) {

programmer.setMood(State::Depresion);

}

else {

programmer.setMood(State::Stress);

}

countBug++;

return programmer.getMood();

}

int main()

{

setlocale(LC\_ALL,"rus");

Programmer programmer(State::Happy, 20);

Computer computer;

cout << "Бдительность программиста до работы: " << programmer.getAlertness() << endl;

cout << "Компьютер включен?" << computer.getIsWork() << endl;

cout << "Настроение программиста до работы: ";

StateToString(programmer.getMood());

cout << "Начали работать: " << programmer.startWork(computer) << endl;

cout << "Бдительность программиста на работе: " << computer.working(programmer) << endl;

computer.findBug(programmer);

computer.findBug(programmer);

computer.findBug(programmer);

computer.findBug(programmer);

cout << "Настроение после нахождения бага: ";

StateToString(computer.findBug(programmer));

cout << computer.getCountBug() << endl;

cout << programmer.fixBug(computer, 3) << endl;

return 1;

}

Тесты:

