

**ЛАБОРАТОРНАЯ РАБОТА №2 ПО ПРЕДМЕТУ
«ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ
ПРОГРАММИРОВАНИЕ»
ТЕМА «ОБРАБОТКА ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ В
ЯЗЫКЕ C++»**

ЦЕЛЬ РАБОТЫ: изучить особенности обработки исключительных ситуаций на языке C++.

ИСКЛЮЧЕНИЯ И ОБРАБОТКА ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ

Исключения (*exceptions*) – это возникновение непредвиденных условий времени выполнения, например, деление на ноль, невозможность выделения динамической памяти для создания нового объекта, потеря подключения к базе данных или ввод непредвиденных данных, которые нарушают нормальное функционирование программы. Обычно эти условия завершают выполнение программы с системной ошибкой. Язык C++ позволяет **восстанавливать** программу из этих условий и **продолжать** ее выполнение.

Исключение в C++ с точки зрения разработки программного обеспечения – это объект системного или пользовательского класса, создаваемого операционной системой или кодом программы в ответ на обстоятельства, либо не допускающие дальнейшего нормального выполнения программы, либо определенные пользователем. Обработка возникших исключений в приложении позволяет корректно продолжить работу программы.

Обработка исключений обычно используется в случае, когда какая-то часть программы обнаруживает проблему, с которой она не может справиться, причем проблема такова, что обнаружившая ее часть программы не может продолжить выполнение. В таких случаях обнаруживший проблему участок программы нуждается в способе сообщить о случившемся и о том, что он неспособен продолжить выполнение. **Способ сообщения о проблеме не подразумевает знания о том, какая именно часть программы будет справляться с создавшейся ситуацией.** Сообщив о случившемся, обнаружившая проблему часть кода **завершает** работу.

Каждой части программы, способной **передать** исключение, соответствует другая часть, код которой способен **обработать** исключение, независимо от того, что произошло. Например, если проблема в недопустимом вводе данных, то часть кода обработки могла бы попросить пользователя ввести правильные (корректные) данные. Если потеряна связь с базой данных, то часть кода обработки могла бы предупредить об этом пользователя и предложить создать соединение вновь. Исключения обеспечивают **взаимодействие** частей программы, обнаруживающих проблему и решающих ее.

Механизм обработки исключительных ситуаций (*exception handling*) является неотъемлемой частью языка C++. Исключения позволяют отделять код обнаружения проблемы от кода ее решения. Часть программы, ответственная за обнаружение проблемы, может передать информацию о возникшей ситуации другой части программы, которая специально предназначена для решения подобных проблем. Механизм обработки исключительных ситуаций предоставляет программисту средство реагирования на нештатные события и позволяет преодолеть ряд принципиальных недостатков следующих **возможных методов обработки ошибок**:

- **возврат функцией кода ошибки как возвращаемого значения или через выходные аргументы функций;**

- **использование глобальных переменных для хранения ошибок;**

- **использование оператора безусловного перехода *goto* или функций *setjmp/longjmp*.** Функция *setjmp* сохраняет значения окружения для возврата управления программой в точку сохранения. Эта функция принимает аргумент и наполняет его значениями состояний переменных окружения в этой точке кода, для того, чтобы, в случае необходимости, можно было восстановить значения переменных окружения в более позднем вызове функции *longjmp*;

- **использование макроса препроцессора *assert*.** Макрос *assert()* добавляет к программе процедуру диагностики. Если выражение ложно, то есть, результат сравнения равен нулю, макрос *assert()* пишет информацию о вызове в поток *stderr* и вызывает функцию *abort()* для завершения программы.

Возврат функцией кода ошибки является самым обычным и широко применяемым методом. Однако этот метод имеет существенные недостатки. Во-первых, нужно помнить численные значения кодов ошибок. Эту проблему можно обойти, используя перечисляемые типы (*enumerations*). Но в некоторых случаях функция может возвращать широкий диапазон допустимых (неошибочных) значений, и тогда сложно найти диапазон для возвращаемых кодов ошибки. Также при использовании такого механизма сигнализации об ошибках вся ответственность по их обработке ложится на программиста и могут возникнуть ситуации, когда серьезные ошибки могут остаться необработанными.

Возврат кода ошибки через аргумент функции или использование глобальной переменной ошибки являются возможными способами отслеживания ошибок, но использование глобальных переменных для обработки исключительных ситуаций является опасным и приводящим к непредсказуемым проблемам, т.к. неконстантные глобальные переменные может изменить любая вызываемая функция. Также глобальные переменные делают программу менее модульной и гибкой. Функция, которая использует только свои параметры и не имеет побочных эффектов, является идеальной в

плане модульности. Модульность помогает понять структуру программы, что она делает и как можно повторно использовать определённые участки кода в другой программе. Глобальные переменные значительно уменьшают эту возможность.

Использование оператора безусловного перехода в любых ситуациях является нежелательным, кроме того, оператор *goto* позволяет переходить в любое место в пределах функции (с некоторыми оговорками), однако переход между функциями с его помощью невозможен. Оператор *goto* иногда используется в языках *Basic*, *Fortran*, *C*. Однако в *C++* *goto* почти никогда не используется, поскольку любой код, написанный с ним, можно более эффективно переписать с использованием других объектов *C++*, таких как циклы, обработчики исключений или деструкторы. Стоит избегать использования операторов *goto*, если на это нет веской причины.

Пара функций *setjmp/longjmp* является мощным средством, однако и этот метод имеет серьезнейший недостаток: он не обеспечивает вызов деструкторов локальных объектов при выходе из области видимости, что влечет за собой утечку памяти.

Макрос *assert()* является средством для проверки корректности работы программы, а не средством обработки нештатных событий, возникающих в процессе использования программы.

Таким образом, необходим некий другой корректный механизм обработки ошибок, который учитывает объектно-ориентированную парадигму. Таким способом является механизм обработки исключительных ситуаций языка *C++*.

Обработка исключительных ситуаций *C++* лишена недостатков вышеназванных методов реагирования на ошибки. **Обработка исключительных ситуаций позволяет использовать для представления информации об ошибке объект любого типа.** Поэтому можно создать иерархию классов, которая будет предназначена для обработки различных типов аварийных событий. Это упростит, структурирует программу.

Обработка исключений в языке *C++* использует следующий процесс:

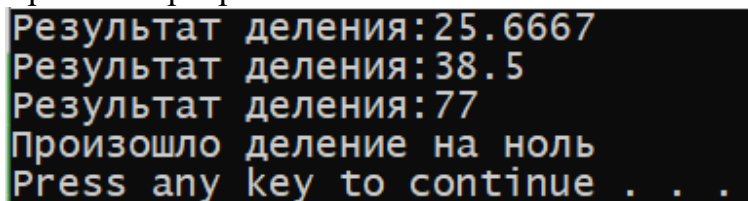
1. Оператор *throw* используется частью кода, обнаружившего проблему, с которой он не может справиться. Об операторе *throw* говорят, что он передает, генерирует (*raise*) исключение.

2. Блок *try* начинается с ключевого слова *try* и завершается одной или несколькими директивами *catch* (*catch clause*). Исключения, переданные из кода, расположенного в блоке *try*, как правило, обрабатываются в одном из блоков *catch*. Поскольку разделы *catch* обрабатывают исключение, то их называют также обработчиками исключений (*exception handler*). Набор определенных в библиотеке классов исключений (*exception classes*) используется для передачи информации о произошедшем событии между операторами *throw* и соответствующими разделами *catch*.

Рассмотрим пример обработки исключительных ситуаций. Функция *division()* возвращает частное от деления чисел, принимаемых в качестве аргументов. Если делитель равен нулю, то генерируется исключительная ситуация.

```
//Пример №1. Возникновение и обработка исключительной ситуации
#include<iostream>
using namespace std;
float division(float divisible, float divisor) {
    if (divisor == 0) throw 1;
    return divisible / divisor;
}
void Ex1() {
    float result, value = 3;
    while (true) {
        try {
            result = division(77., value);
            cout << "Результат деления:" << result << endl;
            value--;
        }
        catch (int) { cout << "Произошло деление на ноль" << endl;
            break;
        }
    }
}
```

Результат работы программы:



```
Результат деления:25.6667
Результат деления:38.5
Результат деления:77
Произошло деление на ноль
Press any key to continue . . .
```

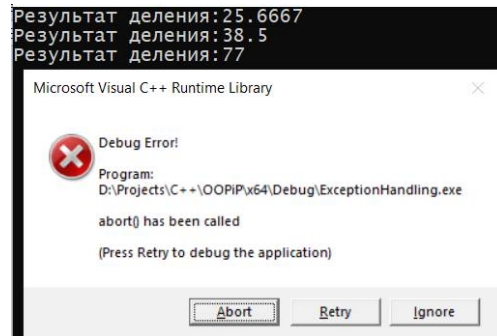
В данном примере необходимо выделить три ключевых элемента. Во-первых, вызов функции *division()* заключен внутри блока, который начинается с ключевого слова *try*. Этот блок указывает, что внутри него могут происходить исключительные ситуации. По этой причине код, заключенный внутри блока *try*, иногда называют **охранным**.

Далее за блоком *try* следует блок *catch*, называемый обычно **обработчиком исключительной ситуации**. Если возникает исключительная ситуация, то выполнение программы переходит к *catch*-блоку, тип которого соответствует типу созданного исключения. Хотя в этом примере имеется единственный обработчик, их в программах может быть значительно больше, и они способны обрабатывать множество различных типов исключительных ситуаций.

Еще одним элементом процесса обработки исключительных ситуаций является оператор *throw* (в данном случае он находится внутри функции *division()*). Оператор *throw* генерирует объект исключительной ситуации,

сигнализируя об исключительной ситуации, который перехватывается соответствующим обработчиком *catch*. Этот процесс называется **созданием исключительной ситуации**. В рассматриваемом примере исключительная ситуация имеет тип целого числа, однако программы могут генерировать любой тип исключительной ситуации.

Если в инструкции *if(divisor==0) throw 1* значение 1 заменить на 1.0, то при выполнении будет выдана ошибка об отсутствии соответствующего обработчика *catch* (так как генерируется исключительная ситуация типа *double*).



ОПЕРАТОР THROW

Обнаружившая исключение часть программы использует оператор *throw* для создания исключения. Оператор *throw* состоит из ключевого слова *throw*, сопровождаемого выражением, которое **определяет тип передаваемого исключения**. Оператор *throw*, как правило, завершается точкой с запятой, что делает его выражением.

Рассмотрим программу, в которой суммируются два объекта класса *SalesItem*. Программа проверяет, относятся ли обе записи к одной книге. Если нет, программа отображает сообщение об ошибке и завершает работу. Для проверки используется международный стандартный книжный номер (*International Standard Book Number, ISBN*) — уникальный номер книжного издания, необходимый для распространения книги в торговых сетях и автоматизации работы с изданием.

//Пример №2. Возникновение и обработка исключительной ситуации

```
#include<iostream>
using namespace std;
class SalesItem {
    int isbn;
    int count;
public:
    SalesItem(int isbn) { this->isbn = isbn; count = 5; }
    int getIsbn() { return isbn; }
    friend istream& operator>> (istream& stream, SalesItem&
salesItem) {
        cin >> salesItem.isbn;
        return stream;
    }
}
```

```

    friend ostream& operator<< (ostream& stream, const SalesItem&
salesItem) {
    cout << "ISBN=" << salesItem.isbn;
    return stream;
}
const SalesItem operator+ (const SalesItem& obj) const {
    // складываем данные о продажах книги
    return (count + obj.count);
}
};
int Ex2() {
    SalesItem item1(123), item2(456);
    //представляют ли объекты item1 и item2 одну и ту же книгу
    if (item1.getIsbn() == item2.getIsbn()) {
        cout << item1 + item2 << endl;
        return 0; //свидетельство успеха
    }
    else {
        cerr << "Книги должны иметь одинаковый номер ISBN" << endl;
        return -1; //свидетельство отказа
    }
}

```

Результаты работы программы:

```

Книги должны иметь одинаковый номер ISBN
Press any key to continue . . .

```

Суммирующая объекты часть кода могла бы быть отделена от части, обеспечивающей взаимодействие с пользователем. В таком случае проверяющую часть можно было бы переписать так, чтобы она передавала исключение, а не возвращала свидетельство отказа.

```

//представляют ли объекты item1 и item2 одну и ту же книгу
if (item1.getIsbn() != item2.getIsbn()) {
    throw runtime_error("Книги должны иметь одинаковый номер ISBN");
}
//если управление здесь, значит, ISBN совпадают
cout << item1 + item2 << endl

```

Если *ISBN* окажутся разными, то в систему обработки исключений будет передан объект исключения типа *runtime_error*. **Передача исключения завершает работу текущей функции и передает управление обработчику, способному справиться с этой ошибкой.**

Тип *runtime_error* является одним из типов исключения, определенных в заголовке *stdexcept* стандартной библиотеки. Объект класса *runtime_error* следует инициализировать объектом класса *string* или символьной строкой в стиле *C*. Эта строка представляет дополнительную информацию о проблеме.

БЛОК TRY-CATCH

Блок *try-catch* имеет следующую структуру:

```
try {
    //операторы_программы
}
//необходимое количество блоков catch для обработки различных типов
исключительных ситуаций
catch (объявление исключения) {
    //операторы_обработчика
}
```

Блок *try* начинается с ключевого слова *try*, за которым следует блок кода, заключенный в фигурные скобки. Блок *try* сопровождается одним или несколькими блоками *catch*. Блок *catch* состоит из трех частей: ключевого слова *catch*, объявления объекта (возможно, безымянного) в круглых скобках (называется объявлением исключения (*exception declaration*)) и операторного блока. **Когда тип исключения в блоке *catch* совпадает с типом сгенерированного исключения, выполняется блок, связанный с этим *catch*. При завершении выполнения кода соответствующего обработчика управление переходит к оператору, находящемуся после всех блоков *catch*.**

Операторы программы в блоке *try* являются обычными программными операторами, реализующими ее логику. Подобно любым другим блокам кода, блоки *try* способны содержать любые операторы языка C++, включая объявления переменных, объектов, вызовы функций и методов. **Объявленные в блоке *try* переменные или объекты недоступны вне этого блока, даже в блоке *catch*.**

СОЗДАНИЕ ОБРАБОТЧИКА

В приведенном выше примере, чтобы избежать суммирования двух объектов класса *SalesItem*, представляющих разные книги, использовался оператор *throw*. Предположим, что суммирующая объекты класса *SalesItem* часть программы отделена от части, взаимодействующей с пользователем. Эта часть могла бы содержать примерно такой код обработки исключения, переданного в блоке сложения.

```
//Пример №3. Возникновение и обработка исключительной ситуации
#include<iostream>
using namespace std;
class SalesItem {
    int isbn;
    int count;
public:
    SalesItem(int isbn) { this->isbn = isbn; count = 5; }
    int getIsbn() { return isbn; }
    friend istream& operator>> (istream& stream, SalesItem&
salesItem) {
        cin >> salesItem.isbn;
```



```

        return stream;
    }
    friend ostream& operator<< (ostream& stream, const SalesItem&
salesItem) {
        cout << "ISBN=" << salesItem.isbn;
        return stream;
    }
    const SalesItem operator+ (const SalesItem& obj) const {
        // складываем данные о продажах книги
        return (count + obj.count);
    }
};
void Ex3() {
    SalesItem item1(123), item2(456);
    //cin >> item1 >> item2;
    ////представляют ли объекты item1 и item2 одну и ту же книгу
    //if (item1.getIsbn() == item2.getIsbn()) {
    //    cout << item1 + item2 << endl;
    //    return 0; //свидетельство успеха
    //}
    //else {
    //    cerr << "Книги должны иметь одинаковый номер ISBN" <<
endl;
    //    return -1; //свидетельство отказа
    //}
    ///////
    while (1) {
        try {
            //код, который складывает объекты класса SalesItem.
            Если произойдет сбой, передается исключение runtime_error
            if (item1.getIsbn() != item2.getIsbn())
                throw runtime_error("Книги должны иметь
одинаковый номер ISBN");
            //если управление здесь, значит, ISBN совпадают
            cout << item1 + item2 << endl;
        }
        catch (runtime_error err) {
            //ISBN слагаемых объектов должны совпадать
            cout << err.what() << "\nПовторить ввод данных?
Введите 'y' или 'n'" << endl;
            char userChoice;
            cin >> userChoice;
            if (userChoice == 'n') break; //выход из цикла while
        }
        cout << "Введите данные о ISBN для двух книг:" << endl;
        cin >> item1;
        cin >> item2;
    }
}

```

Результаты работы программы:


```

Книги должны иметь одинаковый номер ISBN
Повторить ввод данных? Введите 'y' или 'n'
y
Введите данные о ISBN для двух книг:
111
222
Книги должны иметь одинаковый номер ISBN
Повторить ввод данных? Введите 'y' или 'n'
y
Введите данные о ISBN для двух книг:
222
222
ISBN=10
Введите данные о ISBN для двух книг:

```

В блоке *try* расположена обычная логика программы. Это сделано потому, что данная часть программы способна передать исключение типа *runtime_error*.

Данный блок *try* обладает одним блоком *catch*, который обрабатывает исключение типа *runtime_error*. Операторы в блоке после ключевого слова *catch* определяют действия, выполняемые в случае, если код в блоке *try* передаст исключение типа *runtime_error*. В данном случае обработка подразумевает отображение сообщения об ошибке и запрос у пользователя разрешения на продолжение выполнения программы. Когда пользователь вводит символ 'n', цикл *while* завершается, в противном случае он продолжается и считывает два новых объекта класса *SalesItem*.

В сообщении об ошибке используется текст, возвращенный функцией *err.what()*. **В каждом из библиотечных классов исключений определена функция-элемент *what()*, которая не получает никаких аргументов и возвращает символьную строку в стиле C (т.е. *const char**). В случае класса *runtime_error* эта строка является копией строки, использованной при инициализации объекта класса *runtime_error*.**

ПРИ ПОИСКЕ ОБРАБОТЧИКА ВЫПОЛНЕНИЕ ФУНКЦИЙ ПРЕРЫВАЕТСЯ

В сложных системах программа может пройти через несколько блоков *try* прежде, чем встретится с кодом, который передает исключение. Например, в блоке *try* может быть вызвана функция, в блоке *try* которой содержится вызов другой функции с ее собственным блоком *try*, и т.д.

Поиск обработчика осуществляется по цепочке обращений в обратном порядке. Сначала поиск обработчика исключения осуществляется в той функции, в которой оно было сгенерировано. Если соответствующего раздела *catch* не найдено, то работа функции завершается, а поиск продолжается в той функции, которая вызвала функцию, в которой было создано исключение. Если и здесь соответствующий раздел *catch* не найден, то и эта функция также завершается, а поиск продолжается по цепочке вызовов дальше, пока обработчик исключения соответствующего типа не будет найден.

Если блок *catch* с соответствующим типом данных так и не будет найден, то управление перейдет к библиотечной функции *terminate()*, которая определена в заголовке *exception*. Поведение этой функции зависит от системы, но обычно она завершает выполнение программы.

Исключения, которые были созданы в программах, не имеющих блоков *try*, обрабатываются аналогично: в конце концов, без блоков *try* не может быть никаких обработчиков и ни для каких исключений. В таком случае исключение приводит к вызову функции *terminate()*, которая (как правило) и завершает работу программы.

Написание устойчивого к исключениям кода — важная задача при проектировании и разработке ООП программ. Важно понимать, что **исключения прерывают нормальный ход выполнения программы**. В месте, где происходит исключение, некоторые из действий, ожидаемых вызывающей стороной, могут быть выполнены, а другие нет. Как правило, пропуск части программы может означать, что объект останется в недопустимом или неполном состоянии, либо что ресурс не будет освобожден и т.д. Программы, которые правильно "очищают" используемые ресурсы во время обработки исключений, называют **устойчивыми к исключениям (*exception safe*)**.

ПЕРЕДАЧА ИСКЛЮЧЕНИЙ

В языке C++ исключение генерируется выражением *throw*. Тип выражения *throw*, вместе с текущей цепочкой вызова, определяет, какой обработчик (*handler*) будет обрабатывать текущее исключение. Выбирается ближайший обработчик в цепочке вызовов, соответствующий типу переданного объекта. Тип и содержимое этого объекта позволяют передающей части программы сообщать о том, что пошло не так.

Когда выполняется оператор *throw*, расположенные после него операторы игнорируются. Оператор *throw* передает управление соответствующему блоку *catch*. Блок *catch* может быть локальным для той же функции или функции, непосредственно или косвенно вызвавшей ту, в которой произошла ошибка, приведшая к передаче исключения. Тот факт, что управление передается из одного места в другое, имеет два важных следствия:

- функции можно преждевременно покидать по цепочке вызовов.
- по достижении обработчика созданные цепочкой вызова объекты будут уничтожены.

Поскольку операторы после оператора *throw* не выполняются, он похож на оператор *return*: он обычно является частью условного оператора или последним (или единственным) оператором функции.

ОБРАБОТЧИКИ СATCH

Обработчики исключительных ситуаций являются важнейшей частью всего механизма обработки исключений, так как именно они определяют

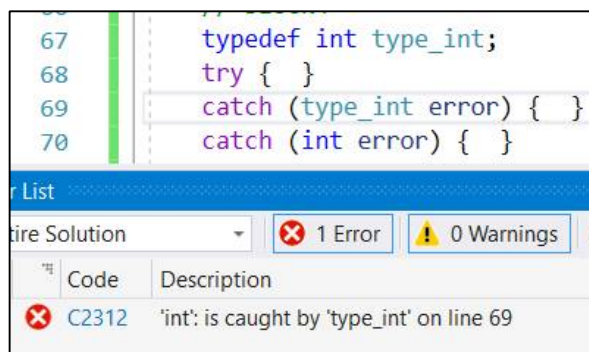
поведение программы после генерации и перехвата исключительной ситуации. Синтаксис блока *catch* имеет следующий вид:

```
catch (тип1 имя аргумента) {  
    //тело обработчика  
}  
catch (тип2 имя аргумента) {  
    //тело обработчика  
}  
//...  
catch (типN имя аргумента) {  
    //тело обработчика  
}
```

После ключевого слова *catch* должен следовать составной оператор, заключенный в фигурные скобки. **В аргументах обработчика можно указать только тип исключительной ситуации, необязательно объявлять имя объекта, если не требуется его использование.**

У каждого блока *try* может быть множество обработчиков, каждый из которых должен иметь свой уникальный тип исключительной ситуации. Неправильной будет следующая запись:

```
typedef int type_int;  
try {  
    // code  
}  
catch (type_int error) {  
    // code  
}  
catch (int error) {  
    // code  
}
```



В этом случае *type_int* и *int* один и тот же тип данных. А следующий пример верен:

```
class FileException {  
public:  
    int i;
```

```
};
try {
    // code
}
catch (FileNotFoundException i1) {
    // code
}
catch (int i) {
    // code
}
```

Абсолютный обработчик совместим с любым типом исключительной ситуации. Для создания абсолютного обработчика надо в качестве его аргументов написать многоточие (эллипсис).

```
catch (...) {
    //блок обработки исключения
}
```

Использование абсолютного обработчика рассмотрим на примере программы, в которой происходит генерация исключительной ситуации типа *char**, но обработчик такого типа отсутствует. В этом случае управление передается абсолютному обработчику.

```
//Пример №4. Использование абсолютного обработчика
#include <iostream>
using namespace std;
void intExceptionDemo(int i) {
    if (i > 100) throw 1; //генерация исключения типа int
}
void stringExceptionDemo() {
    throw "Error"; // генерация исключения типа char*
}
void Ex4() {
    try { //в блоке возможна обработка одного из двух исключений
        intExceptionDemo(99); // возможно исключение типа int
        stringExceptionDemo(); // возможно исключение типа char*
    }
    catch (int) {cout << "Сработал обработчик для типа int" << endl;}
    catch (...) {cout << "Сработал абсолютный обработчик " << endl; }
}
```

Результат работы программы:

```
Сработал абсолютный обработчик
Press any key to continue . . .
```

Так как абсолютный обработчик перехватывает исключительные ситуации любых типов, то он должен стоять в списке обработчиков последним.

Исключительная ситуация типа дочернего класса может быть обработана обработчиком с типом базового класса. Рассмотрим особенности выбора соответствующего обработчика на следующем примере. Пусть имеется класс *EOFException*, являющийся производным от классов *Exception* и *IOException*. Рассмотрим какими обработчиками может быть перехвачена исключительная ситуация типа *EOFException* и типа указателя на *EOFException*.

```
//Пример №5. Срабатывание соответствующего типа обработчика
#include<iostream>
using namespace std;
class Exception {};
class IOException :Exception {};
class EOFException : public IOException {};
void calculate(int value) {
    if (value) throw EOFException();//создание исключительной
ситуации типа объект класса EOFException
    else throw new EOFException;//создание исключительной ситуации
типа указатель класса EOFException
}
void Ex5() {
    int value;
    while (true) {
        try {
            cin >> value;
            calculate(value);
        }
        catch (EOFException*) { cout << "Обработчик типа \"указатель
класса EOFException\"" << endl; }
        catch (EOFException) { cout << "Обработчик типа \"объект
класса EOFException\"" << endl; } //warning: 'EOFException' is caught
by base class IOException
        catch (IOException&) { cout << "Обработчик типа \"ссылка на
объект класса IOException\"" << endl; } //warning: 'EOFException' is
caught by base class IOException
        catch (Exception) { cout << "Обработчик типа \"объект класса
Exception\"" << endl; }
        catch (Exception*) { cout << "Обработчик типа \"указатель
класса Exception\"" << endl; }
        catch (void*) {cout << "Обработчик типа \"void*\"" << endl;}
    }
}
```

Результат работы программы:

```

2
Обработчик типа "объект класса EOFException"
0
Обработчик типа "указатель класса EOFException"
-2
Обработчик типа "объект класса EOFException"

```

В данном примере исключительная ситуация класса *EOFException* может быть направлена любому из обработчиков *Exception*, *IOException* или *EOFException*, поэтому выбирается обработчик, стоящий первым в списке. Аналогично для исключительной ситуации, имеющей тип указателя на объект класса *EOFException*, выбирается первый подходящий обработчик *Exception** или *EOFException**. Эта ситуация также может быть обработана так же обработчиками *void**. Так как к типу *void** может быть приведен любой указатель, то обработчик этого типа будет перехватывать любые исключительные ситуации типа указателя.

Рассмотрим еще один пример.

```

//Пример №6. Выбор соответствующего типа обработчика
#include <iostream>
using namespace std;
class Exception {};
class IOException : public Exception {};
void fun() {
    IOException objDerived;
    Exception& objBase = objDerived;
    throw objBase;
}
void Ex6() {
    try { fun(); }
    catch (IOException) { cout << "Обработчик для типа
\"IOException\" << endl; }
    catch (Exception) { cout << "Обработчик для типа \"Exception\"
<< endl; }
}

```

Результат работы программы:

```

Обработчик для типа "Exception"
Press any key to continue . . .

```

В примере генерируется исключение, имеющее тип *Exception*, несмотря на то, что ссылка *objBase* имеет тип *Exception*, а ссылается на объект класса *IOException*. Так происходит потому, что статический тип ссылки *objBase* равен *Exception*, а не *IOException*.

ПЕРЕНАПРАВЛЕНИЕ ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ

Иногда возникает ситуация, при которой необходимо обработать исключительную ситуацию сначала на более низком уровне вложенности блока *try*, а затем передать ее на более высокий уровень для продолжения обработки. Для этого необходимо использовать оператор

throw без аргументов. В этом случае исключительная ситуация будет перенаправлена к следующему подходящему обработчику (подходящий обработчик не ищется ниже в текущем списке, а сразу осуществляется поиск на более высоком уровне). Рассмотрим пример организации такой передачи. Программа содержит вложенный блок *try* и соответствующий блок *catch*. Сначала происходит первичная обработка, затем исключительная ситуация перенаправляется на более высокий уровень для дальнейшей обработки.

```
//Пример №7. Перенаправление исключительной ситуации
#include<iostream>
using namespace std;
void calculate(int argument) {
    try { if (argument) throw "Ошибка при получении аргумента"; }
    catch (const char* message) {
        cout << message << "- выполняется первый обработчик" << endl;
        throw;
    }
}
void Ex7() {
    try { calculate(1); }
    catch (const char* message) {
        cout << message << "- выполняется второй обработчик" << endl;
    }
}
```

Результат работы программы:

```
Ошибка при получении аргумента- выполняется первый обработчик
Ошибка при получении аргумента- выполняется второй обработчик
Press any key to continue . . .
```

Если ключевое слово *throw* без аргументов используется вне блока *catch*, то автоматически будет вызвана функция *terminate()*, которая по умолчанию завершает программу.

ИСКЛЮЧИТЕЛЬНАЯ СИТУАЦИЯ, ГЕНЕРИРУЕМАЯ ОПЕРАТОРОМ NEW

Некоторые компиляторы поддерживают генерацию исключений в случае ошибки выделения памяти с помощью оператора *new*, в частности исключения типа *bad_alloc*. В следующем примере рассмотрены особенности генерации и обработки исключительных ситуаций типа *bad_alloc*. Искусственно вызывается ошибка выделения памяти и перехватывается исключительная ситуация.

```
//Пример №8. Создание ошибки выделения памяти
#include<iostream>
using namespace std;
void Ex8() {
    double* p;
    int counter = 0;
```



```

try {
    while (1) {
        p = new double[100]; //ошибка выделения памяти
        counter++;
    }
}
catch (bad_alloc exепt) {
    cout << counter << endl;
    cout << "Возникло исключение " << exепt.what() << endl;
}
}

```

Результаты работы программы:

```

2404548
Возникло исключение bad allocation

```

Можно исключение типа *bad_alloc* создать искусственно:

```

bad_alloc exепt;
try {
    if (!(p = new double[100000000])) //память не выделена, p=NULL
        throw exепt; // генерация ошибки выделения памяти
}
catch (bad_alloc exепt) {
    cout << "Возникло исключение " << exепt.what() << endl;
}

```

ГЕНЕРИРОВАНИЕ ИСКЛЮЧЕНИЙ В КОНСТРУКТОРАХ

Механизм обработки исключительных ситуаций очень удобен для обработки ошибок, возникающих в конструкторах. Так как конструктор не возвращает значения, то соответственно нельзя вернуть код ошибки из конструктора. В этом случае наилучшим решением является генерация и обработка исключений. **При генерации исключения внутри конструктора процесс создания объекта прекращается.** Если к этому моменту были вызваны конструкторы базовых классов, то будет обеспечен и вызов деструкторов базовых классов.

Рассмотрим генерацию исключительной ситуации внутри конструктора. Пусть имеется класс *ArrayIterator*, производный от класса *Iterator* и содержащий в качестве поля объект класса *File*. В конструкторе класса *ArrayIterator* генерируется исключительная ситуация.

```

//Пример №9. Создание исключений в конструкторах
#include<iostream>
using namespace std;
class File {
public:
    File() { cout << "Constructor of the File class" << endl; }
    ~File() { cout << "Destructor of the File class" << endl; }
}

```

```

};
class Iterator {
public:
    Iterator() { cout << "Constructor of the Iterator class" << endl; }
    ~Iterator() { cout << "Destructor of the Iterator class" << endl; }
};
class ArrayIterator : public Iterator {
public:
    File ob;
    ArrayIterator(int i) {
        cout << "Constructor of the ArrayIterator class" << endl;
        if (i) throw 1;
    }
    ~ArrayIterator() { cout << "Destructor of the ArrayIterator
class" << endl; }
};
void Ex9() {
    try { ArrayIterator ob(1); }
    catch (int) { cout << "int exception handler"; }
}

```

Результат работы программы:

```

Constructor of the Iterator class
Constructor of the File class
Constructor of the ArrayIterator class
Destructor of the File class
Destructor of the Iterator class
int exception handlerPress any key to continue . . .

```

В программе при создании объекта производного класса *ArrayIterator* сначала вызываются конструкторы базового класса *Iterator*, затем класса *File*, который является компонентом класса *ArrayIterator*. После этого вызывается конструктор класса *ArrayIterator*, в котором генерируется исключительная ситуация. Видно, что при этом для всех ранее созданных объектов вызваны деструкторы, а для объекта класса *ArrayIterator* деструктор не вызывается, так как конструирование этого объекта не было завершено.

ЗАДАНИЕ СОБСТВЕННОЙ ФУНКЦИИ ЗАВЕРШЕНИЯ

Если программа не может найти подходящий обработчик для сгенерированной исключительной ситуации, то будет вызвана процедура завершения *terminate()* (ее также называют обработчиком завершения). Можно установить собственный обработчик завершения, используя функцию *set_terminate()*, единственным аргументом которой является указатель на новую функцию завершения (функция, принимающая и возвращающая *void*), а возвращаемое значение – указатель на предыдущий обработчик.

Ниже приведен пример установки собственного обработчика завершения и генерации исключительной ситуации, для которой не может быть найден обработчик.

```

//Пример №10. Задание собственной функции завершения
#include <iostream>
using namespace std;
void termFunc();
void Ex10() {
    int i = 10, j = 0, result;
    set_terminate(termFunc);
    try {
        if (j == 0) throw "Деление на ноль!";
        else result = i / j;
    }
    catch (int) {
        cout << "Обработка исключения типа int.\n";
    }
    cout << "Эта строка не будет выведена на экран.\n";
}
void termFunc() {
    cout << "Функция termFunc() вызвана функцией terminate().\n";
    // операторы освобождения ресурсов
    exit(-1);
}

```

Результат работы программы:

```

Функция termFunc() вызвана функцией terminate().
Press any key to continue . . .

```

КОНТРОЛЬНЫЕ ВОПРОСЫ К ЛАБОРАТОРНОЙ РАБОТЕ:

1. Опишите как выглядят и для чего используются блоки *try*, *catch*. Что произойдет, если исключение будет сгенерировано вне блока *try*? Допустимо ли нахождение между блоками *try* и *catch* какого-либо кода?
2. Опишите как пишется и для чего используется оператор *throw*.
3. Опишите предназначение функции *terminate()*. Приведите примеры классов исключений стандартной библиотеки.
4. Что произойдет, если после блока *try* отсутствует список обработчиков исключений?
5. Для чего необходимо использовать абсолютный обработчик?
6. Что будет, если в блоке *catch* использовать *throw* без параметра?
7. Что произойдет, если ни один из обработчиков не соответствует типу сгенерированного исключения?
8. Если в блоке *try* не генерируются никакие исключения, куда передается управление после выполнения блока *try*?
9. Прекращается ли процесс создания объекта при генерации исключения внутри конструктора?
10. Доступны ли переменные, объявленные в блоке *try*, в блоках *catch*? Если да, то в каких блоках *catch*?

ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ:

1. Изучить теоретические сведения, полученные на лекции и лабораторной работе, ознакомиться с соответствующими материалами литературных источников.

2. Ответить на контрольные вопросы лабораторной работы.

3. Разработать алгоритм программы по индивидуальному заданию.

4. Написать, отладить и проверить корректность работы созданной программы.

5. Написать электронный отчет по выполненной лабораторной работе.

Отчет должен быть оформлен по стандарту БГУИР ([Стандарт предприятия СТП 01-2017 "Дипломные проекты \(работы\). Общие требования"](#)) и иметь следующую структуру:

1. титульный лист

2. цель выполнения лабораторной работы

3. теоретические сведения по лабораторной работе

4. формулировка индивидуального задания

5. весь код решения индивидуального задания, разбитый на необходимые типы файлов

6. скриншоты выполнения индивидуального задания

7. выводы по лабораторной работе

В РАМКАХ ВСЕГО КУРСА «ООП» ВСЕ ЛАБОРАТОРНЫЕ РАБОТЫ НА ЯЗЫКЕ C++ ДОЛЖНЫ ХРАНИТЬСЯ В ОДНОМ РЕШЕНИИ (SOLUTION), В КОТОРОМ ДОЛЖНЫ БЫТЬ СОЗДАНЫ ОТДЕЛЬНЫЕ ПРОЕКТЫ (PROJECTS) ДЛЯ КАЖДОЙ ЛАБОРАТОРНОЙ РАБОТЫ. ВО ВСЕХ ПРОЕКТАХ ПОЛЬЗОВАТЕЛЬ ДОЛЖЕН САМ РЕШАТЬ ВЫЙТИ ИЗ ПРОГРАММЫ ИЛИ ПРОДОЛЖИТЬ ВВОД ДАННЫХ. ВСЕ РЕШАЕМЫЕ ЗАДАЧИ ДОЛЖНЫ БЫТЬ РЕАЛИЗОВАНЫ, ИСПОЛЬЗУЯ НЕОБХОДИМЫЕ КЛАССЫ И ОБЪЕКТЫ.

ВАРИАНТЫ ИНДИВИДУАЛЬНЫХ ЗАДАНИЙ К ЛАБОРАТОРНОЙ РАБОТЕ №2:

1. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Цветочный магазин». Функционал программы должен позволить **собрать заказ**. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

2. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Книжный магазин». Функционал программы должен позволить собрать заказ. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

3. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Аппаратное обеспечение компьютера». Функционал программы должен позволить собрать компьютер. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

4. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Продовольственный магазин». Функционал программы должен позволить собрать заказ. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

5. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Канцелярские принадлежности». Функционал программы должен позволить собрать заказ. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский)

класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

6. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Музыкальный магазин». Функционал программы должен позволить собрать заказ. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

7. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Железнодорожные пассажироперевозки». Функционал программы должен позволить собрать персонал для поездки. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

8. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Аптека». Функционал программы должен позволить собрать заказ. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

9. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Магазин стройматериалов». Функционал программы должен позволить собрать заказ. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать

собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

10. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Магазин парфюмерии». Функционал программы должен позволить собрать заказ. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

11. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Магазин игрушек». Функционал программы должен позволить собрать заказ. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

12. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Жилищно-эксплуатационная служба». Функционал программы должен позволить собрать персонал для обслуживания территории. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

13. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Оркестр». Функционал программы должен позволить собрать персонал для концерта. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*.

Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

14. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Медицинское учреждение». Функционал программы должен позволить собрать персонал для обслуживания отделения. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

15. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Авиакомпания». Функционал программы должен позволить собрать персонал для обслуживания рейса. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

16. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Высшее учебное заведение». Функционал программы должен позволить собрать данные о студентах. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

17. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Магазин электроники». Функционал программы должен позволить собрать данные о заказе. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных

ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

18. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Магазин одежды». Функционал программы должен позволить собрать данные о заказе. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

19. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Выставка». Функционал программы должен позволить собрать данные об экспозиции. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

20. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Программное обеспечение компьютера». Функционал программы должен позволить собрать данные о необходимом программном обеспечении. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

21. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Тренажерный зал». Функционал программы должен позволить собрать

данные о необходимых тренажерах. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

22. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Спортивная сборная». Функционал программы должен позволить собрать данные спортсменах на соревнования. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

23. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Услуги оператора связи». Функционал программы должен позволить собрать данные об услугах. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

24. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Ювелирный магазин». Функционал программы должен позволить собрать данные о заказе. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

25. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области

«Магазин посуды». Функционал программы должен позволить собрать данные о заказе. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

26. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Станция технического обслуживания автомобилей». Функционал программы должен позволить собрать данные о заказе. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

27. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Телевизионный канал». Функционал программы должен позволить собрать данные команду для проведения телепередач. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

28. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «HR-менеджмент». Функционал программы должен позволить собрать данные команду для разработки проекта. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный

(пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

29. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Каталог мобильных устройств». Функционал программы должен позволить собрать данные заказе. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.

30. Разработать набор классов (минимум 5 классов, связи между классами: агрегация, композиция, наследование) по предметной области «Конвертор файлов». Функционал программы должен позволить собрать данные о проведенных конвертациях и времени их выполнения. Сгенерировать минимум пять типов исключительных ситуаций. Реализовать перенаправление исключительных ситуаций. Сгенерировать минимум одну исключительную ситуацию с оператором *new*. Создать исключительную ситуацию в конструкторе и продемонстрировать вызов конструкторов и деструкторов. Задать собственную функцию завершения. Создать собственный (пользовательский) класс исключения, сгенерировать исключение этого типа и обработать его. В отчете отобразить созданную диаграмму классов.