Федеральное государственное автономное образовательное учреждение

высшего образования

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа «Киберфизические системы и управление»

**Отчет №11**

по дисциплине «Системный подход к разработке программного обеспечения»

|  |  |  |
| --- | --- | --- |
| Выполнил:  студент гр. 3530902/00201 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Позолотин О. В. |
|  | <*подпись*> |  |
|  |  |  |
|  |  |  |
| Руководитель:  Кандидат т.н. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Нестеров С. А. |
|  | <*подпись*> |  |

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2022 г.

Санкт-Петербург

2022

**Ход работы**

**По заданию 8.2:**

Изображение выглядит как текст

Автоматически созданное описание

Создаём ArrayList String-ов и добавляем в него 4 элемента, далее добавляем ещё 2 по индексу и убираем элемент по индексу.

Изображение выглядит как текст

Автоматически созданное описание

Создаём ArrayList с объектами Integer и работаем с ним с помощью итератора.

**По заданию 8.3:**

Изображение выглядит как текст

Автоматически созданное описание

Исправленный блок программы, не выкидывающий исключения о выходе за границы массива.

Изображение выглядит как текст

Автоматически созданное описание

Метод divide может выбросить исключение.

Изображение выглядит как текст

Автоматически созданное описание

Обрабатываем исключение.

**Конспект по книге А. А. Дубакова:**

Exception (Исключение) - это событие, которое происходит во

время выполнения программы, нарушая нормальный ход выполнения ее

команд. Обработка исключений - важная часть Java-программирования,

которая позволяет помещать потенциально опасный код в блок

try (что означает: "попробуем так и посмотрим, вызовет ли это исключение") и добавлять средства для обнаружения исключений (ошибок)

различного типа.

Блоки try, catch и finally вместе образуют "капкан для

ловли исключений". Вначале код, который является потенциально опасным и может вызвать исключение, заключается в состав оператора try.

Если предполагаемое исключение происходит, управление сразу передается соответствующему блоку catch, который обрабатывает исключение. Когда один или оба оператора выполнены, управление передается блоку finally, который выполняется всегда, вне зависимости от

того, имело место исключение или нет. "Поймав" исключение, можно

попытаться аккуратно обойти его или же выйти из программы (или метода).

Язык Java поддерживает иерархию исключений, состоящую из

большого количества типов исключений, сгруппированных в две основные категории:

Контролируемые исключения проверяются компилятором (то

есть компилятор проверяет, что ваш код где-то обрабатывает их).

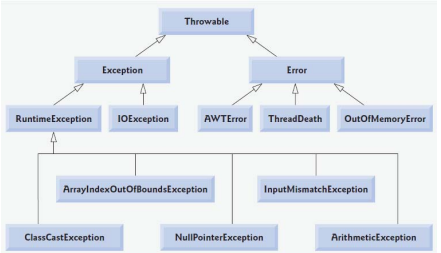
Неконтролируемые исключения (или исключения времени выполнения) не проверяются компилятором.

Контролируемые исключения представляют собой ошибки, которые можно и нужно обрабатывать в программе, к этому типу относятся

все потомки класса Exception. Они проверяются компилятором (то

есть компилятор проверяет, что ваш код где-то обрабатывает их, а

Eclipse будет этому способствовать).



Exception - это подкласс в иерархии, на который при программировании на Java следует уделять основное внимание. Этот уровень

иерархии также разделяется на две ветви: исключения, производные от

класса RuntimeException, и IOExeption. Исключения типа

RuntimeException возникают вследствие ошибок программирования. Исключения типа IOExeption являются следствием непредвиденного стечения обстоятельств, возникающих, как правило, из-за

ошибок ввода-вывода, при выполнении вполне корректных программ.

Ниже представлены некоторые, в том числе, и не изображенные

на Рис. 66 типы исключений, наследованные от RuntimeException:

**IndexOutOfBoundsException** - выбрасывается (thrown),

когда указанный индекс некоторого элемента в структуре данных (массив/коллекция) не попадает в диапазон существующих

индексов.

**NullPointerException** - выбрасывается, когда ссылка

на объект, к которому выполняется обращение не определен и

содержит значение null.

**ClassCastException** – выбрасывается, когда происходит ошибка приведения типов. Всякий раз при приведении типов выполняется проверка на возможность приведения (проверка осуществляется с помощью оператора instanceof.

**ArithmeticException** - выбрасывается, когда выполняются недопустимые арифметические операции, например деление на ноль.

Типы исключений, наследованные от класса Error:

**ThreadDeath** - вызывается при неожиданной остановке потока посредством метода Thread.stop().

**StackOverflowError** - ошибка переполнение стека. Часто

возникает в рекурсивных функциях из-за неправильного условия выхода.

**OutOfMemoryError** - ошибка переполнения памяти.

Когда программа вызывает исключение, говорят, что она выбрасывает (throws) его. Если метод способен выбрасывать исключения, которые он сам не обрабатывает, он должен объявить об этом, чтобы вызывающие его методы могли обеспечить обработку этих исключений. Для определения списка исключений, которые могут выбрасываться методом, используется оператор throws. Если метод в яв ном виде (т.е. с помощью оператора throw) выбрасывает исключение соответствующего класса, тип класса исключений должен быть указан в операторе throws в объявлении этого метода.

Оператор throw используется для выбрасывания исключения «вручную». Для этого нужно иметь объект подкласса класса Throwable, который может быть получен как параметр оператора catch, либо создать с помощью оператора new. Общая форма оператора throw: throw <объект тип Throwable >.

Блоков catch , перехватывающих различные типы ошибок, может быть несколько, но они должны быть определенным образом структурированы. Если какие-то исключения являются подклассами других исключений, то дочерние классы размещаются перед родительскими классами в порядке следования блоков catch. Рассмотрим следующий фрагмент кода:

Изображение выглядит как текст

Автоматически созданное описание

В этом примере NullPointerException - это класс, наследующий класс исключения Exception, так что его следует поместить

перед более общим catch блоком Exception.

Вообще говоря, в Java имеется пять ключевых операторов для работы с исключениями:

**try** - данное ключевое слово используется для отметки начала

блока кода, который потенциально может привести к ошибке.

**catch** - ключевое слово для отметки начала блока кода, предназначенного для перехвата и обработки исключений.

**finally** - ключевое слово для отметки начала блока кода, которое является дополнительным. Этот блок помещается после

последнего блока catch. Управление обычно передаётся в

блок finally в любом случае.

**throw** - служит для генерации исключений. Оператор throw

используется для возбуждения исключения «вручную».

**throws** - ключевое слово, которое прописывается в сигнатуре

метода и обозначающее, что метод потенциально может выбросить исключение с указанным типом.

Оператор finally предназначен для того, чтобы обеспечить гарантированное выполнение какого-либо фрагмента кода, вне зависимости от того, возникла ли исключительная ситуация. Если блок finally завершается ненормально, то весь try завершится ненормально по той же причине. Даже в тех случаях, когда в методе нет соответствующего возбужденному исключению раздела catch, блок finally будет выполнен до того, как управление перейдет к операторам, следующим за разделом try. У каждого раздела try должен быть, по крайней мере, один раздел catch или блок finally. Блок finally очень удобен для закрытия файлов и освобождения любых других ресурсов, захваченных для временного использования в начале выполнения метода.

**Примеры из книги Васильева:**

Изображение выглядит как текст

Автоматически созданное описание

Пример 1, вывод биномиальных коэффициентов.

Изображение выглядит как текст

Автоматически созданное описание

Пример 2

Изображение выглядит как текст

Автоматически созданное описание

Пример 3, случайные числа

Изображение выглядит как текст

Автоматически созданное описание

Пример 4, чётные и нечётные массивы

Изображение выглядит как текст

Автоматически созданное описание

Пример 5, таблица умножения

Изображение выглядит как текст

Автоматически созданное описание

Пример 6, массивы разной размерности

Изображение выглядит как текст

Автоматически созданное описание

Пример 7, массив со строками разной длины

Изображение выглядит как текст

Автоматически созданное описание

Пример 8, умножение вектора на матрицу