НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Объектно-ориентированное программирование на языке Java

## Часть 5. Исключения.

Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua
Tatyana Smykodub, NUoS
tgsmyk@gmail.com

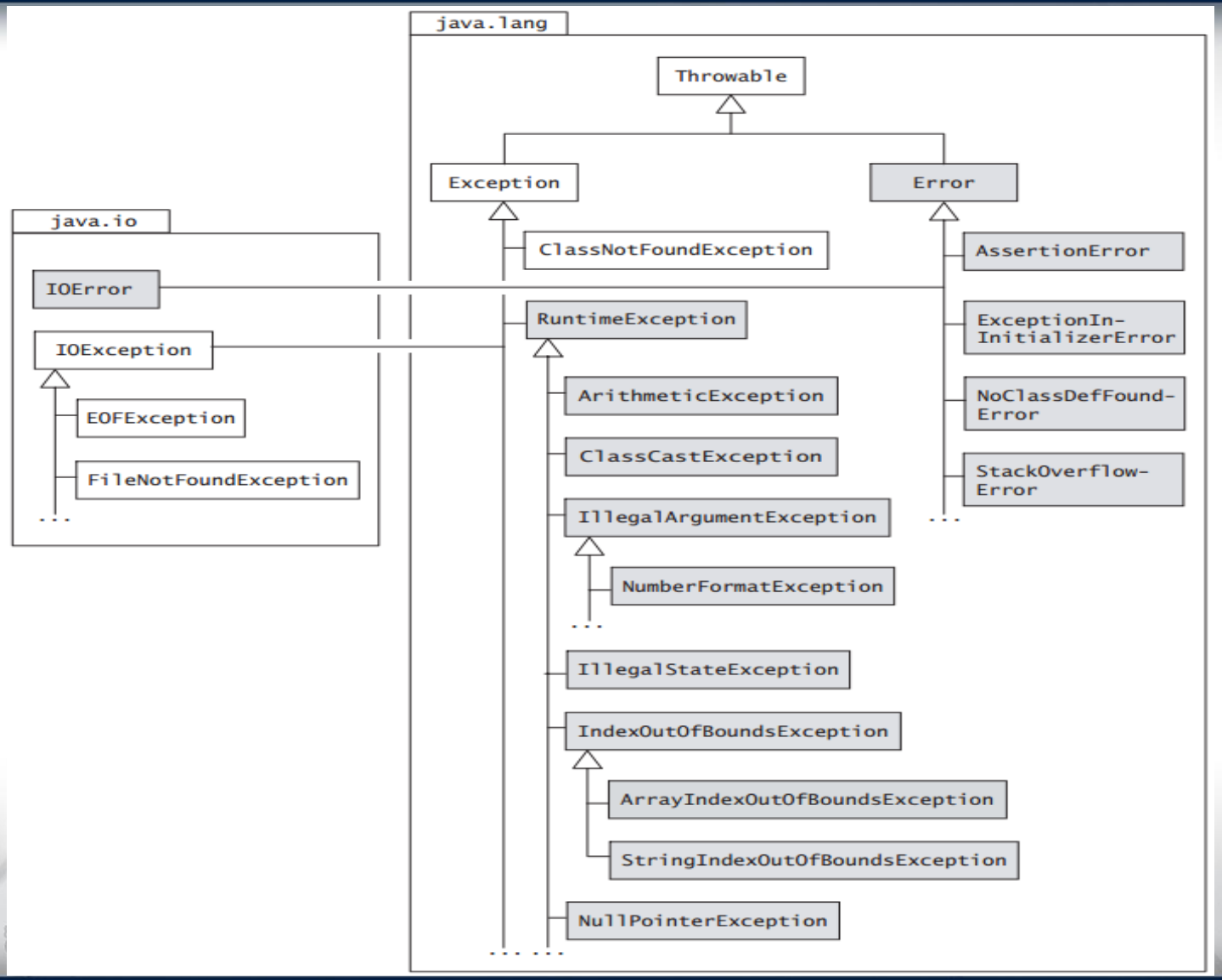
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Исключения

- Исключения в Java - это объекты. Все исключения производятся из класса java.lang.Throwable.
- Иерархия наследования исключений сразу же разделяется на две ветви: Error и Exception , хотя общим предшественником для всех исключений является класс Throwable.
- Иерархия класса Error описывает внутренние ошибки и ситуации, возникающие в связи с нехваткой ресурсов в исполняющей системе Java.
- Класс **Exception** представляет исключения, которые обычно хотят запрограммировать. Его подкласс RuntimeException представляет множество распространенных ошибок программирования, которые могут проявляться во время выполнения. Другие подклассы класса Exception определяют другие категории исключений, например связанные с I / O исключения в пакете java.io (IOException, FileNotFoundException, EOFException, IOError)

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Исключения



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Класс RuntimeException

- Исключения типа **RuntimeException** возникают вследствие ошибок программирования. И являются подклассами класса java.lang.RuntimeException, который является подклассом класса Exception.
- Исключения, производные от класса **RuntimeException,** связаны со следующими программными ошибками.
  - Неверное приведение типов.
  - Выход за пределы массива.
  - Попытка обратиться к объекту по пустой ссылке **null**
- Исключения типа **RuntimeException** практически всегда возникают по вине программиста. Так, исключения типа **ArrayIndexOutBoundsException** можно избежать, если всегда проверять индексы массива. А исключение **NullPointerException** никогда не возникнет, если перед тем, как воспользоваться переменной, проверить, не содержит ли она пустое значение **null**

ArithmeticException, ArrayIndexOutOfBoundsException, ClassCastException, IllegalArgumentException, NumberFormatException, IllegalStateException, NullPointerException

# Класс Error

- Класс Error и его подклассы определяют ошибки, которые явно не пойманы и обычно не восстанавливаются.
- Исключения типа Error относятся к ошибкам, возникающим в виртуальной машине Java, а не в прикладной программе. Контролировать такие исключения невозможно.

*AssertionError*, *ExceptionInInitializerError*, *IOError*, *NoClassDefFoundError*, *StackOverflowError*

# Проверяемые и непроверяемые исключения

- Кроме RuntimeException, Error и их подклассов, все исключения называются *проверяемыми*.
- Возможность возникновения проверяемого исключения может быть отслежена еще на этапе компиляции кода. Компилятор проверяет, может ли данный метод генерировать или обрабатывать исключение.
- Проверяемые исключения должны быть обработаны в методе, который может их генерировать, или включены в **throws**-список метода для дальнейшей обработки в вызывающих методах.

# Проверяемые и непроверяемые исключения

- Исключения, определенные классами Error и RuntimeException, и их подклассы являются непроверяемыми исключениями, это означает, что метод не обязан разбираться с этими исключениями. (Компилятор не проверяет, может ли генерировать и/или обрабатывать метод эти исключения. )
- Они либо безвозвратные (например, класс Error), и программа не должна пытаться справиться с ними, или
- Они представляют собой ошибки программирования (например, класс RuntimeException) и обычно должны рассматриваться как таковые, а не как исключения.

# Обработка исключений

На практике используется один из трех способов обработки исключений:

1. перехват и обработка исключения в блоке **try-catch** метода;
2. объявление исключения в секции **throws** метода и передача вызывающему методу (в первую очередь для проверяемых исключений);
3. использование собственных исключений.

# try и catch

Они используются совместно. Это означает, что в коде нельзя указать ключевое слово catch, не указав ключевое слово try.

```
try {
     // блок кода, в котором должны отслеживаться ошибки
    }
catch (тип_исключения_1 объект_исключения){
// обработчик исключения тип исключения 1
}
catch (тип_исключения_2 объект_исключения){
// обработчик исключения тип исключения 2
}
finally {
  // закрытие открытых ранее ресурсов
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Пример

```
// Демонстрация обработки исключений
class ExcDemol {
public static void main(String args[]){
int nums[]= new int[4];
try { //Создается блок try
        System.out.println("До генерации исключения"};
        nums[7] = 10;// Попытка использовать индекс, выходящий за
        //границы массива
        System.out.println("Эта строка не будет отображаться"};
}
catch (ArrayindexOutOfBoundsException exc) {
// Перехват исключений, обусловленных выходом за границы массива
System.out.println("Выход за границы массива!"};
}
System.out.println("После оператора catch"};
}
}
```

# trows

- Иногда исключения нецелесообразно обрабатывать в том методе, в котором они возникают.
- В таком случае их следует указывать с помощью ключевого слова throws, т.е. обязанности по обработке данного исключения поручаются вызывающему методу.

```
возвращаемый_тип имя_метода(список_параметров) throws
список_исключений {
// Тело метода
}
```

В списке исключений через запятую указываются исключения, которые может генерировать метод.

# trows пример

```
// Использование ключевого слова throws
class ThrowsDemo {
  public static char prompt(String str) throws java.io.IOException
{ // Обратите внимание на оператор throws в объявлении метода
    System.out.print(str + ": ");
    return (char) System.in.read();
}
public static void main(String args[]) {
 char ch;
 try {// В методе prompt () может быть
    ch = prompt("Enter a letter") // сгенерировано исключение,
    // поэтому его вызов следует поместить в блок try
}
catch(java.io.IOException exc) {
    System.out.println("Произошло исключение ввода-вывода");
    ch = 'X';}
System.out.println("Вы нажали клавишу " + ch);
}}
```

# trows пример

- Но это исключение не обрабатывается в методе prompt ().
- Вместо этого в объявлении метода указан оператор throws, т.е. обязанности по обработке данного исключения поручаются вызывающему методу.
- В данном случае вызывающим является метод main (), в котором и перехватывается исключение.

# JAVA 7

Начиная с JDK 7 в Java появилась очень полезная и удобная разновидность оператора try – он называется *try с ресурсами* или *try с автоматическим управлением ресурсами*.

Данный оператор применяется к ресурсам, реализующим интерфейс AutoCloseable (в нём определён метод *close()*), и отпадает необходимость заботиться о ручном закрытии ресурсов. Компилятор сам неявно создаёт секцию *finally* в которой и происходит закрытие без участия разработчика.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# try с ресурсами

Оператор **try с ресурсами** имеет следующую сигнатуру:

```
try (создание ресурса) {
     // использование ресурса
}
```

Если предполагается использование нескольких ресурсов, то они отделяются точкой с запятой.

```
try (создание ресурса1;
            создание ресурса2) {
    // использование ресурсов
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# И еще…

Интересная статья про правильное освобождение ресурсов:

https://habr.com/ru/post/178405/