OTUS

ОНЛАЙН-ОБРАЗОВАНИЕ

33– Handling Exceptions (Часть 3)

Дмитрий Коган

Как меня слышно и видно?

Если нет – напишите, если слышите – смайлик в чат.

Цели :

- **Изучим пессимистичный механизм**
- **Научимся пугать правильно**

Начинаем?

Темы экзамена

- ❑ Java Basics
- ❑ Working with Java Data Types
- ❑ Using Operators and Decision Constructs
- ❑ Creating and Using Arrays
- ❑ Using Loop Constructs
- ❑ Working with Methods and Encapsulation
- ❑ Working with Inheritance
- ❑ Handling Exceptions
- ❑ Working with Selected classes from the Java API

Подтемы экзамена

Handling Exceptions

- Differentiate among checked exceptions, unchecked exceptions, and Errors
- Create a try-catch block and determine how exceptions alter normal program flow
- Describe the advantages of Exception handling
- Create and invoke a method that throws an exception
- Recognize common exception classes (such as NullPointerException, ArithmeticException, ArrayIndexOutOfBoundsException, ClassCastException)

Пессимистичный механизм

Использование throws

```
public class App {
    public static void main(String[] args) {
        throw new Exception(); // тут ошибка компиляции
    }
}

>> COMPILATION ERROR: unhandled exception: java.lang.Exception
```

Использование throws

```
import java.io.IOException;

public class App {
    public static void main(String[] args) throws IOException {
        throw new Exception(); // тут ошибка компиляции
    }
}

>> COMPILATION ERROR: unhandled exception: java.lang.Exception
```

Использование throws

```java
public class App {
    public static void main(String[] args) throws Exception { // предупреждаем о Exception
        throw new Exception(); // и кидаем Exception
    }
}
```

Использование throws

```
public class App {
    public static void main(String[] args) throws Throwable { // предупреждаем "целом" Throw
able
        throw new Exception(); // а кидаем только Exception
    }
}
```

Использование throws

```
public class App {
    public static void main(String[] args) throws Exception { // пугаем
        // но ничего не бросаем
    }
}
```

Использование throws

```
public class App {
    public static void main(String[] args) {
        f(); // тут ошибка компиляции
    }

    public static void f() throws Exception {
    }
}

>> COMPILATION ERROR: unhandled exception: java.lang.Exception
```

Использование throws

```java
public class App {
    // они пугают целым Throwable
    public static void main(String[] args) throws Throwable {
        f();
    }
    // хотя мы пугали всего-лишь Exception
    public static void f() throws Exception {
    }
}
```

Предчувствие будущего

```
public class InternetDownloader {
    public static byte[] (String url) throws IOException {
        return "<html><body>Nothing! It's stub!</body></html>".getBytes();
    }
}
```

Непроверяемые исключения

```
public class App {
    public static void main(String[] args) {
        f();
    }
    public static void f() throws RuntimeException {
    }
}
```

Множественные исключения

```java
import java.io.EOFException;
import java.io.FileNotFoundException;

public class App {
    // пугаем ОБОИМИ исключениями
    public static void main(String[] args) throws EOFException, FileNotFoundException {
        if (System.currentTimeMillis() % 2 == 0) {
            throw new EOFException();
        } else {
            throw new FileNotFoundException();
        }
    }
}
```

Множественные исключения

```java
import java.io.EOFException;
import java.io.FileNotFoundException;

public class App {
    // пугаем ОБОИМИ исключениями
    public static void main(String[] args) throws EOFException, FileNotFoundException {
        f0();
        f1();
    }
    public static void f0() throws EOFException {...}
    public static void f1() throws FileNotFoundException {...}
}
```

Множественные исключения

```
import java.io.EOFException;
import java.io.FileNotFoundException;
import java.io.IOException;

public class App {
    // пугаем ПРЕДКОМ исключений
    public static void main(String[] args) throws IOException {
        if (System.currentTimeMillis() % 2 == 0) {
            throw new EOFException();
        } else {
            throw new FileNotFoundException();
        }
    }
}
```

Множественные исключения

```java
import java.io.EOFException;
import java.io.FileNotFoundException;

public class App {
    // пугаем ПРЕДКОМ исключений
    public static void main(String[] args) throws IOException {
        f0();
        f1();
    }
    public static void f0() throws EOFException {...}
    public static void f1() throws FileNotFoundException {...}
}
```

Множественные исключения

```java
import java.io.EOFException;
import java.io.FileNotFoundException;

public class App {
    public static void main(String[] args) throws IOException, InterruptedException {
        f0();
        f1();
        f2();
    }
    public static void f0() throws EOFException {...}
    public static void f1() throws FileNotFoundException {...}
    public static void f2() throws InterruptedException {...}
}
```

Catch-or-Declare

```
public static void main(String[] args)
      throws NoMoreCarrotsException { // declare exception
   eatCarrot();
}

public static void main(String[] args) {
   try {
      eatCarrot();
   } catch (NoMoreCarrotsException e ) { // handle exception
      System.out.print("sad rabbit");
   }
}


class NoMoreCarrotsException extends Exception {}

private static void eatCarrot() throws NoMoreCarrotsException {
}
```

Ошибки перехвата

```
public class App {
    public static void main(String[] args) {
        try {
            throw new Throwable();
        } catch (Exception e) {
            // ...
        }
    }
}

>> COMPILATION ERROR: unhandled exception: java.lang.Throwable
```

Ошибки перехвата

```
public class App {
    public static void main(String[] args) {
        try {
            throw new Exception();
        } catch (Error e) {
            // ...
        }
    }
}

>> COMPILATION ERROR: unhandled exception: java.lang.Exception
```

Ultima ratio

```
public class TestEx {
    public static void main(String[] args) {
        try {
            FileWriter fw = new FileWriter( fileName: "NonExistent.txt");
        }
        catch (IOException e) {
            System.out.println("The File could not be found");
            throw new Exception(e); // Без finally с return; здесь вылетит ошибка компиляции
        }
        finally {
            return;
        }
    }
}
```

Вопросы?

Код недоступен

Только декларация

```
public void bad() {
   try {
      eatCarrot();
   } catch (NoMoreCarrotsException e ) { // DOES NOT COMPILE
      System.out.print("sad rabbit");
   }
}

public void good() throws NoMoreCarrotsException {
   eatCarrot();
}

private void eatCarrot() { }
```

Доступные исключения

```
class Test {
    public static void main(String[] args) {
        try { }

     //  catch (IOException e){}  // комперр: 'exception IOException is never
                                   // thrown in body of corresponding try stat'

        catch (RuntimeException rte){}     // ни в одном из них нет проблем с
        catch (Error err){}                // компиляцией, хотя «затыкание рта»
        catch (Exception e){}              // исключениям с помощью пустых
        catch (Throwable t){}              // блоков - практика порочная
    }
}
```

Точная проверка

```
public class App2 {
    public void run(String[] args) {
        try {
            split();
        }
        catch (Ex2 ex2) {
            ex2.printStackTrace();
        }

        /* Компилируется, хотя и излишне с catch выше
        catch (Ex1 ex1)
        {}
        */

        /* Не компилируется. Компилятор уверен, что этого исключения не будет
        catch (IOException io)
        {}
        */
    }

    private void split() throws Ex2 {
    }
}

class Ex1 extends Exception {}

class Ex2 extends Ex1 {}
```

К слову

- ✓ Если метод что-то делает с файлами, нужно проверить наличие в **его сигнатуре** throws IOException.
- ✓ **Если в сигнатуре метода есть** throws IOException**, нужно проверить наличие в классе** import java.io.

Вопросы?

Поведение компилятора

Sic JVM creatus est

Необходимо понимать, что

— проверка на checked исключения происходит в момент компиляции (compile-time checking)

— перехват исключений (catch) происходит в момент выполнения (runtime checking)

Мимо

```
public class App {
    // пугаем Exception
    public static void main(String[] args) throws Exception {
        Throwable t = new Exception(); // и лететь будем Exception
        throw t; // но тут ошибка компиляции
    }
}

>> COMPILATION ERROR: unhandled exception: java.lang.Throwable
```

Снова мимо

```
public class App {
    // пугаем Exception
    public static void main(String[] args) throws Exception {
        try {
            Throwable t = new Exception(); // и лететь будет Exception
            throw t; // но тут ошибка компиляции
        } catch (Exception e) {
            System.out.println("Перехвачено!");
        }
    }
}

>> COMPILATION ERROR: unhandled exception: java.lang.Throwable
```

Есть!

```
public class App {
    // ТЕПЕРЬ пугаем Throwable
    public static void main(String[] args) throws Throwable {
        try {
            Throwable t = new Exception(); // а лететь будет Exception
            throw t;
        } catch (Exception e) { // и мы перехватим Exception
            System.out.println("Перехвачено!");
        }
    }
}

>> Перехвачено!
```

Упражнение

```
class WeightOutOfBoundsException extends Exception {
    @Override
    public String toString(){ return "Oh no... "; }
}

class GoodMorning {
    private int weight;

    void standOnScale(GoodMorning gm) throws Exception {
        gm.weight = (int)(Math.random()*21 + 90);          // generates random
                                                           // integer numbers
                                                           // from 90 to 110 (kg)
        if (weight > 100)
            throw new WeightOutOfBoundsException();
        else {
            System.out.print("I'm ordering pizza tonight! ");
        }
    }

    public static void main(String[] args) {
        GoodMorning gm = new GoodMorning();
        try {
            gm.standOnScale(gm);
        } catch (WeightOutOfBoundsException woobe) {
            System.out.print(woobe);
        }
        finally { System.out.println("Finally!"); }
    }
}
```

What is the result?

A. Oh no…
B. Oh no… Finally!
C. I'm ordering pizza tonight!
D. I'm ordering pizza tonight! Finally!
E. Compilations fails

Ответ: E

Вопросы?

Переопределение

Нельзя пугать больше

```
class CanNotHopException extends Exception { }
class Hopper {
  public void hop() { }
}
class Bunny extends Hopper {
  public void hop() throws CanNotHopException { } // DOES NOT COMPILE
}
```

Меньше - можно

```
class Hopper {
   public void hop() throws CanNotHopException { }
}
class Bunny extends Hopper {
   public void hop()  { }
}
```

Так тоже можно

```
class Hopper {
   public void hop() throws Exception { }
}
class Bunny extends Hopper {
   public void hop() throws CanNotHopException { }
}
```

Unchecked exceptions

```
class Hopper {
   public void hop() { }
}
class Bunny extends Hopper {
   public void hop() throws IllegalStateException { }
}
```

Упражнение

What will be the output of this program code?

```
1. public class Fruit {
2.   public void eatMe() throws Exception {
3.       System.out.print("Eat a fruit");
4.   }
5.   public static void main (String [] args) {
6.       Fruit f = new Mango();
7.       f.eatMe();
8.   }
9. }
10. class Mango extends Fruit {
11.   public void eatMe() {
12.       System.out.println("Eating a mango");
13.   }
14. }
```

Please select :

- *A. Compilation fails due to an Error at line 3*
- *B. Compilation fails due to an Error at line 5*
- *C. Compilation fails due to an Error at line 7*
- *D. Compilation fails due to an Error at line 11*
- *E. Eat a fruit*
- *F. Eating a mango*
- *G. Compilation succeeds but an error is thrown at runtime*

Ответ: C

Что можно вставить?

```
class Parent {
    void doStuff() throws Exception{
        throw new Exception();
    }
}
class Child extends Parent {
    public void doStuff(){ }
    public static void main(String[] args) {
        _________ obj = new _________();
        obj.doStuff();
    }
}
```

Только это

```
class Parent {
    void doStuff() throws Exception{
        throw new Exception();
    }
}
class Child extends Parent {
    public void doStuff(){ }
    public static void main(String[] args) {
        _________ obj = new _________();
        obj.doStuff();
    }
}
```

Child - Child

Или так

```
class Parent {
    void doStuff() throws Exception{
        throw new Exception();
    }
}
class Child extends Parent {
    public void doStuff(){ }
    public static void main(String[] args) {
        _________ obj = new _________();
        obj.doStuff();
    }
}
```

Child - Child

Или сделать так: ((Child)obj).doStuff();

Интерфейсы

- ✓ Когда класс имплементирует несколько интерфейсов, его переопределяющий метод должен удовлетворять правилам по каждому из переопределенных методов.

Интерфейсы

```
interface I1 {void run() throws IOException, IllegalArgumentException;}

interface I2 {void run() throws FileNotFoundException, InterruptedException;}

class C implements I1, I2{
// декларация ниже НЕ ВАЛИДНА, так как C хочет переопределить run() в I2,
// но тот бросает FNFE, а IOE шире, чем FNFE:

    public void run() throws IOException {}                    // INVALID

// а вот эта декларация НЕ ВАЛИДНА, потому что run() в I1 не бросает IE:
//    public void run() throws InterruptedException {}         // INVALID
}
```

Конструкторы

- ✓ С точки зрения исключений, правило переопределения методов работает противоположным образом в случае конструкторов.
- ✓ Конструкторы могут возбуждать исключения подобно методам, но правила оказываются противоположными: в то время как перегружающему методу нельзя бросать новое Checked Exception или его суперкласс, конструктору в подклассе запрещено бросать подкласс этого Checked Exception, если только оно не задекларировано в throws-секции.
- ✓ Отметим, что как и в случае с методами, это правило не распространяется на Runtime Exceptions.

Конструкторы

```
1 class A {
2     A() throws IOException {}
3     A(int a) throws IOException {}
4     A(double d) throws IndexOutOfBoundsException {}
5 }
6
7 class B extends A{
8     B() throws FileNotFoundException {
9         super();
10    }
11    B(int b) throws Exception { super(1);}
12    B(double d) throws ArrayIndexOutOfBoundsException { super(1.0); }
13 }
14
15 class C extends A{
16    C() throws FileNotFoundException {
17        try {
18            super();
19        }
20        catch(IOException ioe){}
21    }
22 }
```

Конструкторы

```
1 class A {
2     A() throws IOException {}
3     A(int a) throws IOException {}
4     A(double d) throws IndexOutOfBoundsException {}
5 }
6
7 class B extends A{
8     B() throws FileNotFoundException {
9         super();                                          // INVALID
10    }
11    B(int b) throws Exception { super(1);}
12    B(double d) throws ArrayIndexOutOfBoundsException { super(1.0); }
13 }
14
15 class C extends A{
16    C() throws FileNotFoundException {                   // INVALID
17        try {
18            super();                                      // INVALID
19        }
20        catch(IOException ioe){}
21    }
22 }
```

Вопросы?

Вывод исключений

Три способа

```
public static void main(String[] args) {
   try {
      hop();
   } catch (Exception e) {
      System.out.println(e);
      System.out.println(e.getMessage());
      e.printStackTrace();
   }
}
private static void hop() {
   throw new RuntimeException("cannot hop");
}
```

```
java.lang.RuntimeException: cannot hop
cannot hop
java.lang.RuntimeException: cannot hop
   at Handling.hop(Handling.java:15)
   at Handling.main(Handling.java:7)
```

Цепочка исключений

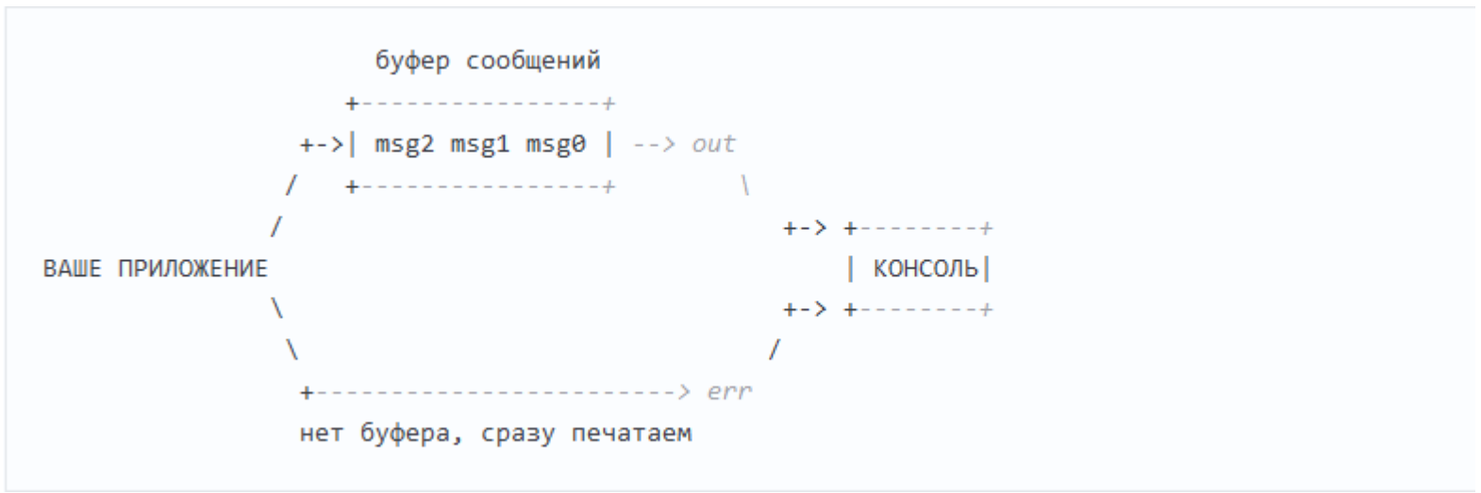
```java
public class ChainOfExceptions {
    public static void main(String[] args) {
        try {
            // creating an exception
            ArithmeticException e = new ArithmeticException("Apparent cause");
            // set the cause of an exception
            e.initCause(new NullPointerException("Actual cause"));
            // throwing the exception
            throw e;
        }
        catch (ArithmeticException e) {
            // Getting the actual cause of the exception
            System.out.println(e.getCause());
            System.out.println("---");
            e.printStackTrace();
        }
    }
}
```

```
java.lang.NullPointerException: Actual cause
---
java.lang.ArithmeticException: Apparent cause
    at dik.ocajp.webinar33.ChainOfExceptions.main(ChainOfExceptions.java:11) <5 internal calls>
Caused by: java.lang.NullPointerException: Actual cause
    at dik.ocajp.webinar33.ChainOfExceptions.main(ChainOfExceptions.java:13)
    ... 5 more
```

По порядку

✓ StackTrace **выводится после** finally.

Для чистоты эксперимента надо использовать System.err.println(). System.out — buffered-**поток вывода, а** System.err — **нет.** StackTrace необработанного исключения выводится через System.err, что позволяет им обгонять «обычные» сообщения.



Пример

```
public class StackTrace {
    public static void main(String[] args) {
        try {
            throw new NullPointerException("Ой-ё-ёй!");
        }
        finally {
            System.err.println("Финальный аккорд");
        }
    }
}
```

```
Финальный аккорд
Exception in thread "main" java.lang.NullPointerException: Ой-ё-ёй!
    at dik.ocajp.webinar33.StackTrace.main(StackTrace.java:10) <5 internal calls>
```

Упражнение

Which lines can fill in the blank to make the following code compile? (Choose all that apply.)

```
void rollOut() throws ClassCastException {}

  public void transform(String c) {
   try {
      rollOut();
   } catch (IllegalArgumentException | _____________________) {
   }
}
```

A. `IOException a`

B. `Error b`

C. `NullPointerException c`

D. `RuntimeException d`

E. `NumberFormatException e`

F. `ClassCastException f`

G. None of the above. The code contains a compiler error regardless of what is inserted into the blank.

Ответ: BF

Вопросы?

Домашнее задание

Тест

Пожалуйста, пройдите опрос

https://otus.ru/polls/17839/

Спасибо за внимание!

Никогда ничего не пугайтесь!