

A background image showing a complex network of interconnected nodes and lines, rendered in shades of blue. The nodes are represented by small circles, and the lines are thin, connecting the nodes in a web-like structure. The overall color scheme is monochromatic blue.

## Лекция 7. Исключения

**NetCracker**®

# Источник ошибок

и

## Внешние:

- Неверные данные (от пользователя или другого агента)
- Некритичные сбои оборудования и соединений

## Внутренние:

- Выполнение кода с ошибками
- Нарушение ограничений среды

## Критичные:

- Ошибки работы JVM
- Критичные сбои оборудования и нехватка

ресурсов

# Методы обработки ошибок

## Возвращение кода ошибки

```
int errNum = firstMethod();  
if (errNum == ERR_SIGN1) {  
    // обработка 1-ой ошибки  
}  
else if (errNum == ERR_SIGN2) {  
    // обработка 2-ой ошибки  
}  
else  
    secondMethod();
```

## Встроенный механизм проверки

```
try {  
    firstMethod();  
    secondMethod();  
}  
catch(Exception1 e1) {  
    // обработка 1-ой ошибки  
}  
catch(Exception2 e2) {  
    // обработка 2-ой ошибки  
}
```

# Обработка исключительных ситуаций

(try-catch-finally)

```
try {  
    // Код, который может выбрасывать           исключени  
                                                    я  
}  
catch(SpecificException1 e) {  
    // обработка специфической ошибки  
}  
  
catch(SpecificException2 e) {           ошибк  
                                                    и  
    // обработка другой специфической  
    // обработка более широкого класса  
} ошибок  
catch(WiderException e) {  
catch(Exception e) {  
    // и если исключение не подошло под выш  
    шаблоны е  
}  
  
finally {  
    // выполнится в любом случае  
}
```

# Использование оператора

throw

**Пример генерации искусственного  
исключения**

**сами  
М**

**программистом:**

....

```
public int calculate(int theValue) throws Exception {  
    if (theValue < 0) {  
        throw new Exception(  
            "Параметр для вычисления не должен быть отрицательным"  
        );  
    }  
}
```

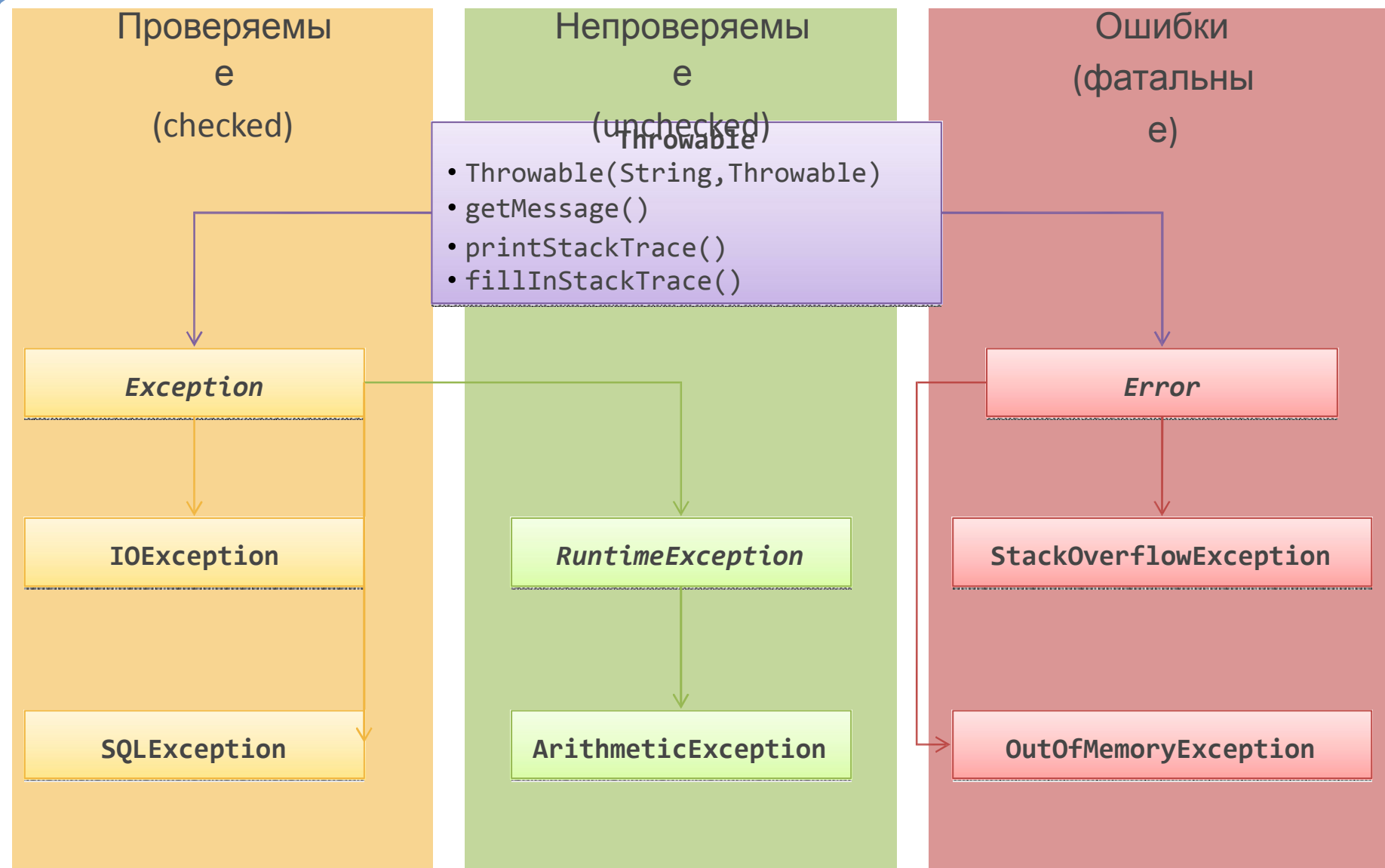
# Использование оператора

## throw

**Передача исключительной ситуации на более высокий уровень иерархии с промежуточной обработкой:**

```
try {  
:  
} catch (IOException ex) {  
:  
// Обработка исключительной ситуации  
:  
// Повторное возбуждение исключительной ситуации  
throw ex;  
}
```

# Типы исключений



# Создание своих классов исключений

Допускается создание собственных классов исключений.

Для этого достаточно создать свой класс, унаследовав его от любого наследника `java.lang.Throwable` (или от самого `Throwable`):

```
public class UserException extends Exception {  
  
    public UserException() {  
        super();  
    }  
  
    public UserException(String descry) {  
        super(descry);  
    }  
  
    public UserException(Throwable cause) {  
        super(cause);  
    }  
  
    public UserException(String descry, Throwable cause) {  
        super(descry, cause);  
    }  
}
```

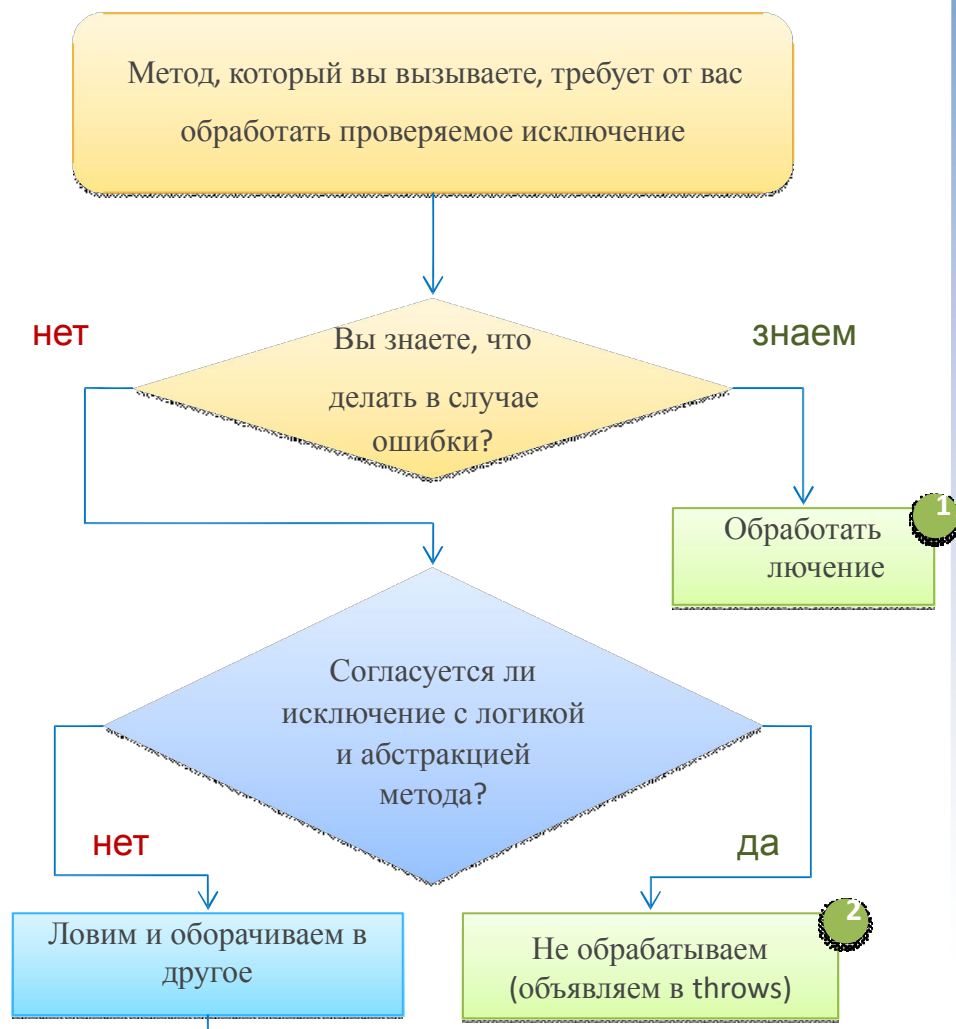
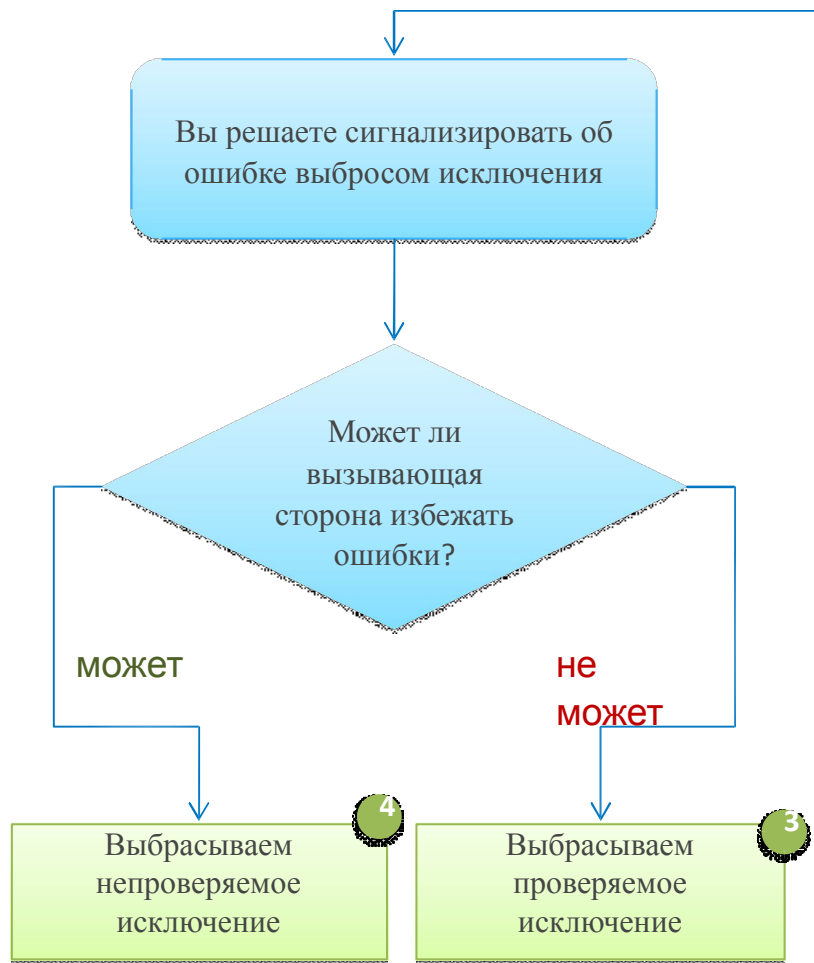


# Переопределение исключений

При **переопределении методов** следует помнить, что если переопределяемый метод объявляет список возможных исключений, то переопределяющий метод **не может расширять** этот список, но **может его сужать**.

```
public class BaseClass {  
    public void method () throws IOException {  
        ...  
    }  
}  
public class IllegalOne extends BaseClass {  
    public void method () throws IOException, IllegalAccessException {  
        ...  
    }  
}
```

# Общая схема работы с исключениями



# Примеры

1-2

## 1. Мы знаем что делать в случае ошибки

```
int readInt(String filename, int defaultValue) {  
    try {  
        return readIntFrom(filename); // throws IOException  
    }  
    catch (IOException e) {  
        return defaultValue;  
    }  
}
```

## 2. Не знаем что делать, но исключение соответствует логике метода

```
void sendMessage(String host, int port, String message) throws NetworkException {  
    Connection connection = new Connection(host, port);  
    try {  
        connection.send(message); // throws NetworkException  
    }  
    finally {  
        connection.close();  
    }  
}
```

# Пример

3

3. Не знаем что делать, но исключение не соответствует уровню абстракции метода.

```
Record readRecord(String source) throws StorageException {  
  
    try {  
        XmlReader xml = openXml(source);  
        try {  
            return xml.readRecord(); // throws XmlException  
        }  
        finally {  
            xml.close();  
        }  
    }  
  
    catch (XmlException e) {  
        throw new StorageException(e);  
    }  
}
```

# Пример

## 4

### 4. Исключения может избежать вызывающая сторона.

```
/**
 * Evaluates maximum element of array.
 *
 * @param elements must not be null and must contain at least one element
 * @return maximum element of given array
 *
 * @throws ArithmeticException if given array is empty
 * @throws NullPointerException if elements is null
 */
double maximum(double[] elements) {
    if (elements.length == 0)
        throw new ArithmeticException("maximum of empty array doesn't make sense");
    double max = elements[0];
    for (double current : elements)
        if (current > max)
            max = current;
    return max;
}
```

A blue background with a white network diagram consisting of interconnected nodes and lines, resembling a mesh or web structure.

**Thank you!**

