

Основни езикови елементи

Стоян Велев, Петър Петров / SAP Labs Bulgaria
15 октомври 2014

Public



Съдържание

Типове

Условия и разклонения

Итерация / Цикли

Масиви

Исключения

Примитивни типове

Primitive type	Size	Minimum	Maximum	Wrapper type
boolean	—	—	—	Boolean
char	16 bits	Unicode 0	Unicode $2^{16}-1$	Character
byte	8 bits	-128	+127	Byte
short	16 bits	-2^{15}	$+2^{15}-1$	Short
int	32 bits	-2^{31}	$+2^{31}-1$	Integer
long	64 bits	-2^{63}	$+2^{63}-1$	Long
float	32 bits	IEEE754	IEEE754	Float
double	64 bits	IEEE754	IEEE754	Double
void	—	—	—	Void



range at full precision	precision*
$\pm 1.18 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$	approx. 7 decimal digits
$\pm 2.23 \times 10^{-308}$ to $\pm 1.80 \times 10^{308}$	approx. 15 decimal digits

Autoboxing

```
char c = 'a';
```

```
Character ch = new Character('a');
```

```
Character ch = 'x'; //autoboxing
```

```
char c = ch; //autoboxing
```

Числа с голяма точност

`BigInteger`

`BigDecimal`

Scoping

```
{  
    int x = 12;  
    // Only x available  
    {  
        int q = 96;  
        // Both x & q available  
    }  
    // Only x available  
    // q is "out of scope"  
}
```

Литералите: какво ново в Java 7?

Числови литерали с подчертавка

```
int thousand = 1_000;  
int million = 1_000_000;  
long hexWords = 0xCAFE_BABE;
```

Числови литерали в двоична бройна система

```
int one = 0b1;  
int mask = 0b1010_1010_1010;
```

Низове

String

StringBuilder

StringBuffer ← thread safe

Операции с низове

`length()`

`charAt()`

`toCharArray()`

`equals()`, `equalsIgnoreCase()`

`compareTo()`

`contains()`

`startsWith()`

`endsWith()`

`indexOf()`, `lastIndexOf()`

Операции с низове

`substring()`

`replace()`

`toLowerCase()`

`toUpperCase()`

`trim()`

Булеви изрази

`true` и `false`

за разлика от C/C++, не може да ползвате число
вместо булев израз

Simple & compound statements

if-else

```
if (boolean_expression) {  
    statement  
}
```

ИЛИ

```
if (boolean_expression) {  
    statement  
} else {  
    statement  
}
```

? :

condition ? statement1 : statement2;

```
if (condition) {  
    statement1  
} else {  
    statement2  
}
```

Итерация

```
while (boolean_expression) {  
    statement  
}
```

Итерация

```
do {  
    statement  
} while (boolean_expression)
```

Итерация

```
for (initialization; boolean_expression; step)
{
    statement
}
```

Операторът ,

Итерация

Enhanced for-loop

```
for (declaration : Iterable) {  
    statement  
}
```

Iterable: array, String, Collection

Unconditional branching

`return [value]`

`break [label]`

`continue [label]`

No “`goto`” ← keyword reserved but not used

Switch

```
switch (selector) {  
    case value1 : statement; break;  
    case value2 : statement; break;  
    case value3 : statement; break;  
    // ...  
    default: statement;  
}
```

Масиви

```
int[] a;
```

```
int a[];
```

```
int a[] = {1, 2, 3, 4};
```

```
int b[] = new int[7];
```

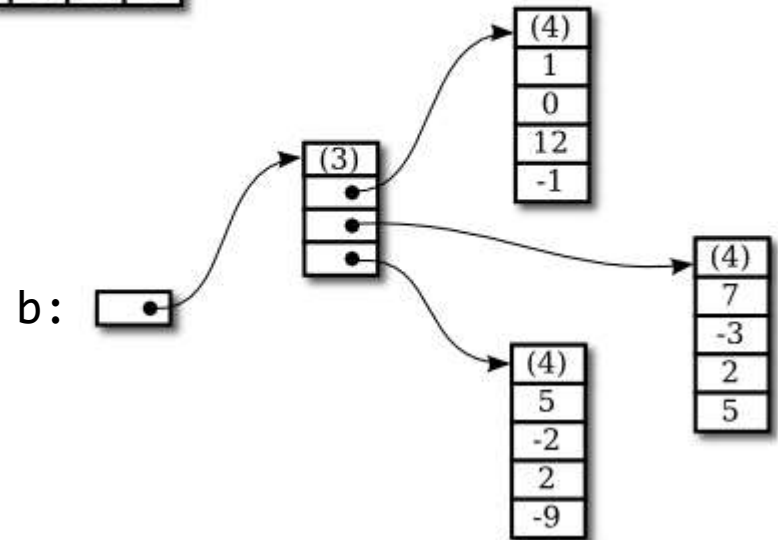
```
b.length;
```

Многомерни масиви

```
int[][] a;
```

```
a = new int[3][4];
```

1	0	12	-1
7	-3	2	5
-5	-2	2	-9



```
int[][] b = { { 1, 0, 12, -1 },  
              { 7, -3, 2, 5 },  
              { -5, -2, 2, -9 }  
            };
```

Многомерни масиви

```
double[][] matrix = new double[7][];  
// rows have not yet been created!  
  
for (int i = 0; i < 7; i++) {  
    // Create row i with i + 1 elements.  
    matrix[i] = new double[i+1];  
}
```

Стандартни операции с масиви

```
System.arraycopy(from_arr, offset_from,  
to_arr, offset_to, num_elements);
```

```
Arrays.equals(arr1, arr2);
```

```
Arrays.deepEquals(arr1, arr2);
```

```
Arrays.fill(arr, value);
```

```
Arrays.toString(arr);
```

Стандартни операции с масиви

```
Arrays.sort(arr);
```

```
Arrays.sort(a, Collections.reverseOrder());
```

```
Arrays.sort(sa,  
String.CASE_INSENSITIVE_ORDER);
```

```
Arrays.binarySearch(arr, value);
```

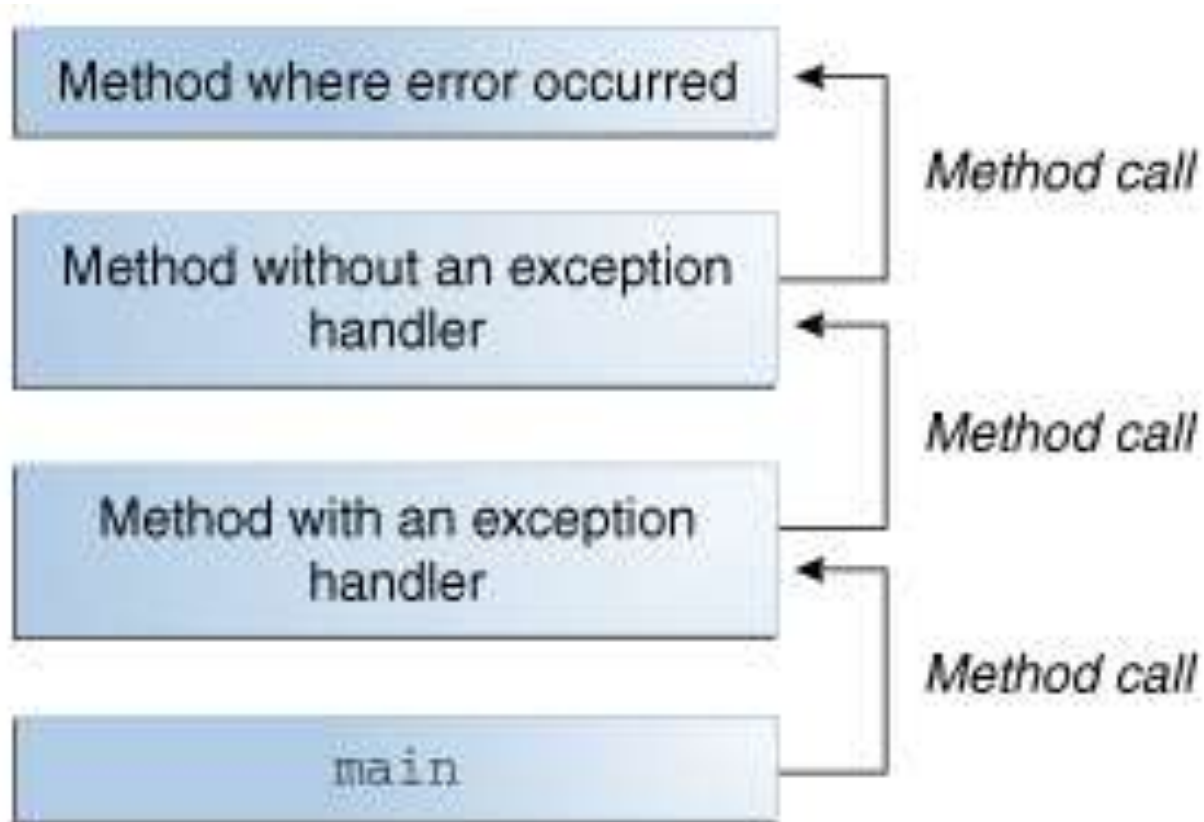
```
Arrays.asList(arr);
```

Исключения (Exceptions)

Definition:

An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

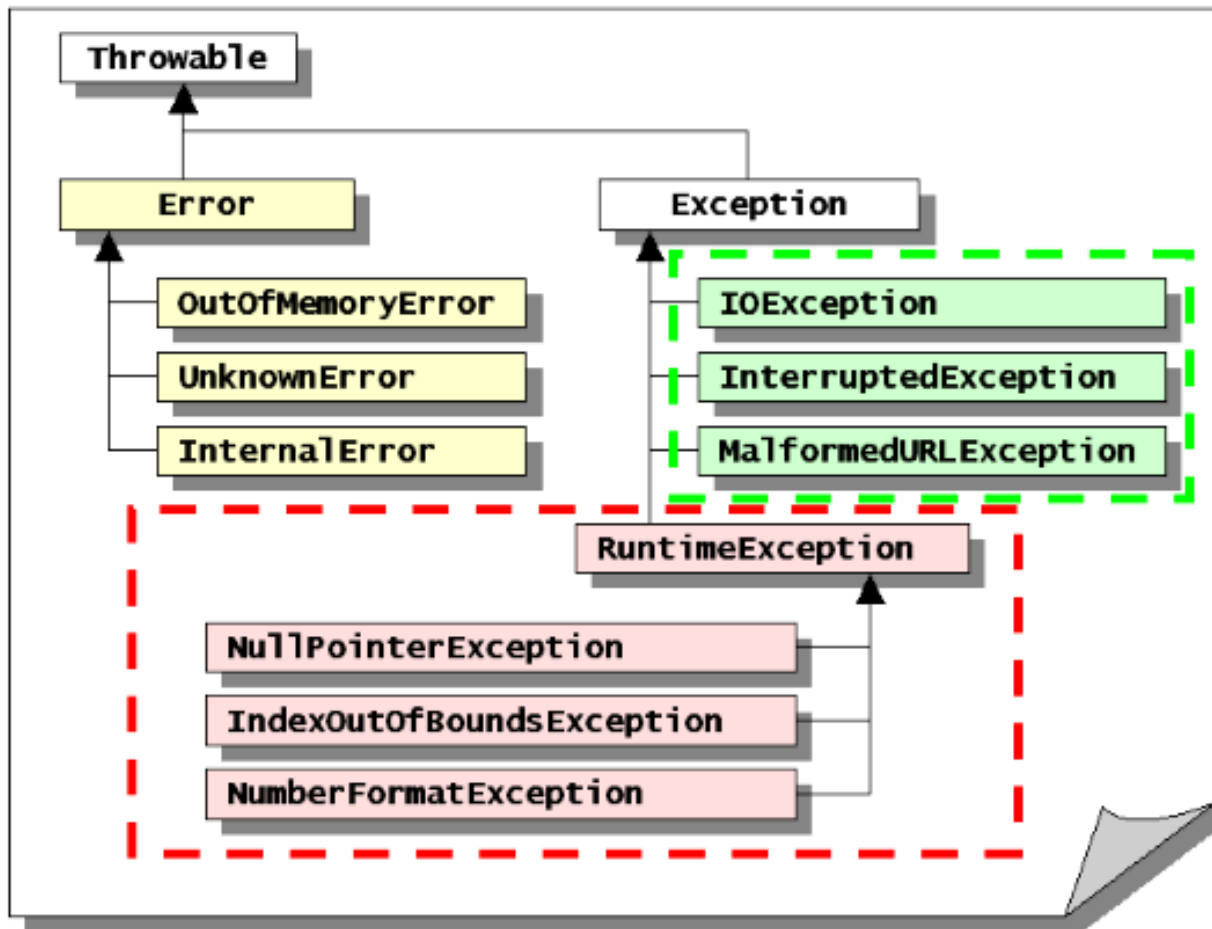
Стек на извикванията (Call stack)



Трите вида изключения

- *Checked Exceptions*
- *Errors*
- *Runtime Exception*

Class Hierarchy



Try block

The first step in constructing an exception handler is to enclose the code that might throw an exception within a try block.

```
try {  
    code  
} catch and finally blocks . . . .
```

Catch block

You associate exception handlers with a try block by providing one or more catch blocks directly after the try block. No code can be between the end of the try block and the beginning of the first catch block.

```
try {  
      
} catch (ExceptionType name) {  
      
} catch (ExceptionType name) {  
      
}
```

Catch block

```
try {  
    ..  
} catch (IndexOutOfBoundsException e) {  
    System.err.println("IndexOutOfBoundsException:  
n: " + e.getMessage());  
} catch (IOException e) {  
    System.err.println("Caught IOException: " +  
e.getMessage());  
}
```

Catch block – new in java 7

```
catch (IOException | SQLException ex) {  
    logger.log(ex);  
    throw ex;  
}
```

Finally block

The finally block *always* executes when the try block exits. This ensures that the finally block is executed even if an unexpected exception occurs

Finally - more than just exception handling

- it allows the programmer to avoid having cleanup code accidentally bypassed by a return, continue, or break
- cleanup code in a finally block is always a good practice, even when no exceptions are anticipated

try-with-resources – new in java 7

- The try-with-resources statement is a try statement that declares one or more resources.
- The try-with-resources statement ensures that each resource is closed at the end of the statement.
- Resource can be any object that implements `java.lang.AutoCloseable`, which includes all objects which implement `java.io.Closeable`.

try-withOUT-resources example

```
static String readFirstLine(String path)
throws IOException {
    BufferedReader br = new
    BufferedReader(new FileReader(path));
    try {
        return br.readLine();
    } finally {
        if (br != null) {br.close();}
    }
}
```

try-with-resources – example

```
static String readFirstLineFromFile(String  
path) throws IOException {  
    try (BufferedReader br = new  
BufferedReader(new FileReader(path))) {  
        return br.readLine();  
    }  
}
```

Exceptions Thrown by a Method

```
public void writeList() throws IOException,  
IndexOutOfBoundsException {
```

```
public void writeList() throws IOException  
{
```

How to Throw Exceptions

```
public Object pop() {  
    if (size == 0) {  
        throw new EmptyStackException();  
    }  
    ...  
}
```

Chained Exceptions

```
try {  
} catch (IOException e) {  
    throw new SampleException("Other  
        IOException", e);  
}
```

Finally block

The finally block *always* executes when the try block exits. This ensures that the finally block is executed even if an unexpected exception occurs

Защо изключения?

- Separating Error-Handling Code from "Regular" Code
- Propagating Errors Up the Call Stack
- Grouping and Differentiating Error Types



Благодарим за вниманието!

За контакти:

stoyan.vellev@sap.com

p.petrov@sap.com

ИСПОЛЗВАНА ЛИТЕРАТУРА

- *Thinking in Java* by Bruce Eckel

[link to free online edition](#)

- *Highlights of Technology Changes in Java SE 7*

<http://www.oracle.com/technetwork/java/javase/jdk7-relnotes-418459.html>