

# Програмування-1

Лекція 13

Exceptions

# Exception

- Терміни:
  - Виняток
    - дослівний переклад, не вірно по суті
  - Виняткова ситуація
    - вірно по суті, але довго вимовляти
  - Експеншн
    - суржик, але зручно 😊

# Навіщо потрібні Exceptions?

- Якщо десь щось пішло не так, програма має про це знати
- За допомогою **Exception** сервіс може сповістити про проблему



# Способи сповістити про проблему

- Застарілі підходи
  - встановлення/перевірка пропорця помилки
  - повертання/перевірка коду помилки
- Сучасні підходи
  - checked exceptions (Java)
  - unchecked exceptions (Java, C#, C++, ...)

# Пропорець помилки (приклад на мові Pascal)

Вірно	Невірно
<pre>procedure WriteToFile(s: string); var   f : textfile; begin   AssignFile(f, 'TargetComp.txt');   Rewrite(f);   Error := IOResult;   if Error then     ShowMessage('Error opening file = ' + inttostr(Error))   else     begin       WriteLn(f, s);       Error := IOResult;       if Error then         ShowMessage('Error writing to file = ' + inttostr(Error));     end;   Closefile(f); end;</pre>	<pre>procedure WriteToFile(s: string); var   f : textfile; begin   AssignFile(f, 'TargetComp.txt');   Rewrite(f);   WriteLn(f, s);   Closefile(f); end;</pre>

# Повертання коду помилки (приклад на мові С)

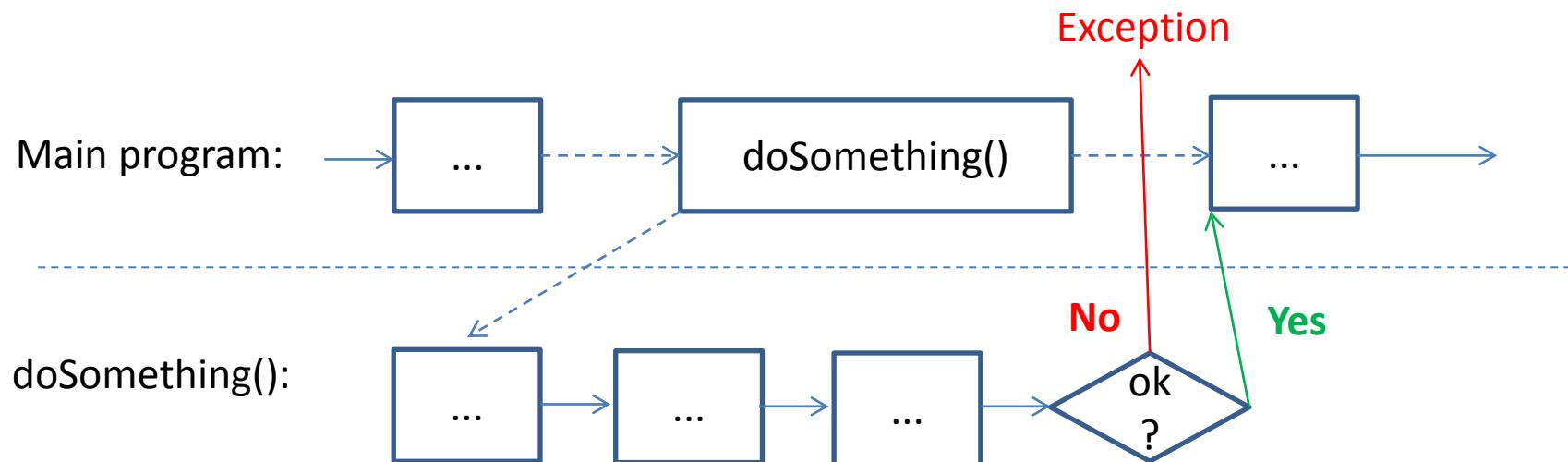
Вірно	Невірно
<pre>#include &lt;stdio.h&gt; int main () { FILE * pFile; pFile = fopen ("myfile.txt","wt"); if (pFile == NULL) { // NULL return value: File Open Error     return -1; } if (fprintf (pFile, "example") &lt; 0) { // Negative return value: Write Error     fclose (pFile);     return -1; } if (fclose (pFile) == EOF) { // return value == EOF: Close Error     return -1; } return 0; }</pre>	<pre>#include &lt;stdio.h&gt; int main () { FILE * pFile; pFile = fopen ("myfile.txt", "wt"); fprintf (pFile, "example"); fclose (pFile); return 0; }</pre>

# Недоліки застарілих підходів

- Програміст може забути або полінуватися написати код для обробки помилки
- Перемішування коду для позитивного та негативного сценаріїв робить код менш читабельним

# Exception

- За допомогою Exception сервіс може повідомити основну програму про помилку



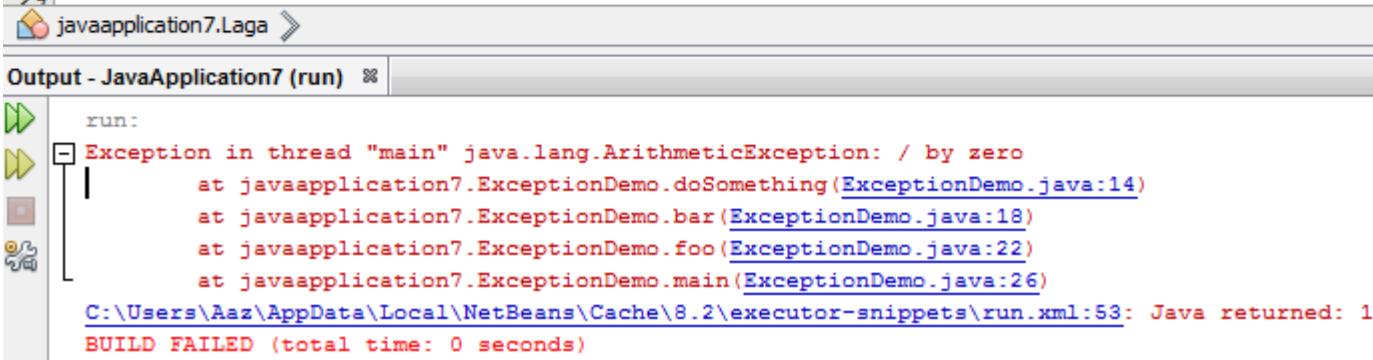
# Exception

```
11  public class ExceptionDemo {  
12  
13      static void doSomething() {  
14          int a=1/0;  
15      }  
16  
17      public static void main(String[] args) {  
18          doSomething();  
19      }  
20  }  
21  
javaapplication7.Laga >  
  
Output - JavaApplication7 (run) ✘  
run:  
Exception in thread "main" java.lang.ArithmetricException: / by zero  
    at javaapplication7.ExceptionDemo.doSomething(ExceptionDemo.java:14)  
    at javaapplication7.ExceptionDemo.main(ExceptionDemo.java:18)  
C:\Users\Aaz\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1  
BUILD FAILED (total time: 0 seconds)
```

# Exception propagation

- Якщо було згенеровано exception, який не було «спіймано», то даний метод завершується аварійно
- При цьому exception «вилітає» у той метод, з якого був здійснений виклик даного методу
- Якщо його і там не було «спіймано», exception вилітає ще вище
- Якщо його не спіймали у main(), то програма завершується аварійно

# Exception propagation

```
11     public class ExceptionDemo {  
12  
13         static void doSomething() {  
14             int a=1/0;  
15         }  
16  
17         static void bar() {  
18             doSomething();  
19         }  
20  
21         static void foo() {  
22             bar();  
23         }  
24  
25         public static void main(String[] args) {  
26             foo();  
27         }  
28     }  
29  


The screenshot shows the NetBeans IDE interface. At the top, there's a code editor window titled "javaapplication7.Laga" containing the Java code above. Below it is an "Output - JavaApplication7 (run)" tab showing the execution results. The output window displays the following:  
run:  
Exception in thread "main" java.lang.Arithm... / by zero  
at javaapplication7.ExceptionDemo.doSomething(...)  
at javaapplication7.ExceptionDemo.bar(...)  
at javaapplication7.ExceptionDemo.foo(...)  
at javaapplication7.ExceptionDemo.main(...)  
C:\Users\Aaz\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1  
BUILD FAILED (total time: 0 seconds)


```

# Exception propagation

```
11  public class ExceptionDemo {  
12  
13      static void doSomething() {  
14          int a=1/0; X  
15      }  
16  
17      static void bar(){  
18          doSomething();  
19      }  
20  
21      static void foo() {  
22          bar();  
23      }  
24  
25      public static void main(String[] args) {  
26          foo();  
27      }  
28  }
```

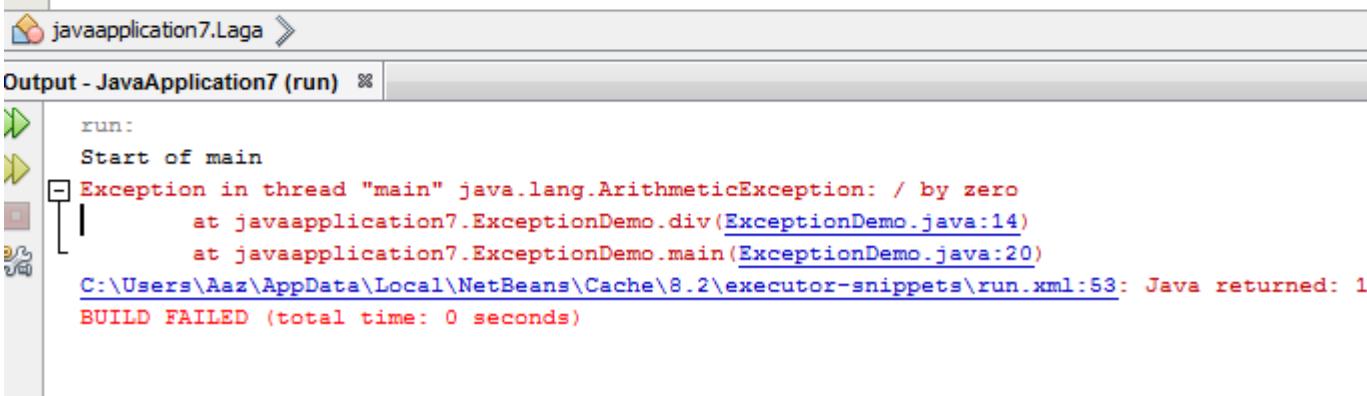
Diagram illustrating exception propagation:

- Entry Point:** The `main` method at line 25.
- Program terminated:** The program ends at line 28 with an error.
- Call Stack:** The stack shows the sequence of method calls: `main` calls `foo`, which calls `bar`, which calls `doSomething`.
- Exception:** An `ArithmeticException` is thrown at line 14 due to division by zero (`a=1/0`).
- Stack Trace:** The output window shows the full stack trace from the exception to the entry point.

```
javaapplication7.Laga  
Output - JavaApplication7 (run)  ✘ Stack Trace  
run:  
Exception in thread "main" java.lang.Arithmeti...  
at javaapplication7.ExceptionDemo.doSomething(...)  
at javaapplication7.ExceptionDemo.bar(...)  
at javaapplication7.ExceptionDemo.foo(...)  
at javaapplication7.ExceptionDemo.main(...)  
C:\Users\Aaz\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1  
BUILD FAILED (total time: 0 seconds)
```

# Exception propagation

```
11  public class ExceptionDemo {  
12  
13      static int div(int a, int b) {  
14          return a / b;  
15      }  
16  
17      public static void main(String[] args) {  
18          System.out.println("Start of main");  
19  
20          int result = div(1, 0);  
21          System.out.println("Result:" + result);  
22  
23          System.out.println("End of main");  
24      }  
25  }  
26
```



The screenshot shows the NetBeans IDE interface with the code editor and the Output window.

**Output - JavaApplication7 (run)**

```
run:  
Start of main  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
    at javaapplication7.ExceptionDemo.div(ExceptionDemo.java:14)  
    at javaapplication7.ExceptionDemo.main(ExceptionDemo.java:20)  
C:\Users\Aaz\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1  
BUILD FAILED (total time: 0 seconds)
```

# Exception propagation

The diagram illustrates the flow of an exception from a method call to its caller. A green curved arrow starts at the call to `div(1, 0)` (line 20) and points to the `System.out.println("Result:" + result);` statement (line 21). Handwritten numbers 1 through 5 are placed along this arrow: 1 at the start of the call, 2 at the call itself, 3 at the start of the main block, 4 at the `System.out.println` statement, and 5 at the end of the main block. A red curved arrow originates from the `return a / b;` statement (line 14) and points to the `System.out.println` statement (line 21). A red 'X' is placed at the `/ by zero` part of the stack trace in the output window, indicating an error. A green checkmark is placed next to the `System.out.println` statement in the code editor.

```
11  public class ExceptionDemo {  
12  
13      static int div(int a, int b) {  
14          return a / b; X  
15      }  
16  3  public static void main(String[] args) {  
17      System.out.println("Start of main");  
18      1 2 int result = div(1, 0);  
19      System.out.println("Result:" + result);  
20      X 4  
21      System.out.println("End of main");  
22  }  
23  
24  }  
25 }  
  
javaapplication7.Laga  
Output - JavaApplication7 (run) X  
run:  
Start of main  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
    at javaapplication7.ExceptionDemo.div(ExceptionDemo.java:14)  
    at javaapplication7.ExceptionDemo.main(ExceptionDemo.java:20)  
C:\Users\Aaz\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1  
BUILD FAILED (total time: 0 seconds)X
```

# try-catch

```
try {
```

<Небезпечний код>;

```
} catch (<ExceptionType> ex) {
```

<Обробка виняткової ситуації>;

```
}
```



# try-catch

```
11  public class ExceptionDemo {  
12  
13      static int div(int a, int b) {  
14          return a / b;  
15      }  
16  
17      public static void main(String[] args) {  
18          System.out.println("Start of main");  
19  
20          try {  
21              int result = div(10, 2);  
22              System.out.println("Result: " + result);  
23          } catch (Exception e) {  
24              System.out.println("Catched !!!");  
25          }  
26  
27          System.out.println("End of main");  
28      }  
29  }
```



Output - JavaApplication7 (run) ✘

```
run:  
Start of main  
Result: 5  
End of main  
BUILD SUCCESSFUL (total time: 0 seconds)
```

# try-catch

```
11 public class ExceptionDemo {  
12     static int div(int a, int b) {  
13         return a / b; }  
14     public static void main(String[] args) {  
15         System.out.println("Start of main");  
16         try {  
17             int result = div(10, 2);  
18             System.out.println("Result: " + result);  
19         } catch (Exception e) {  
20             System.out.println("Catched !!!");  
21         }  
22         System.out.println("End of main");  
23     }  
24 }  
25  
26  
27  
28  
29 }
```

Output - JavaApplication7 (run) ✘

```
run:  
Start of main  
Result: 5  
End of main  
BUILD SUCCESSFUL (total time: 0 seconds)
```

# try-catch

```
11  public class ExceptionDemo {  
12  
13      static int div(int a, int b) {  
14          return a / b;  
15      }  
16  
17      public static void main(String[] args) {  
18          System.out.println("Start of main");  
19  
20          try {  
21              int result = div(10, 0);  
22              System.out.println("Result: " + result);  
23          } catch (Exception e) {  
24              System.out.println("Exception Caught !!!");  
25          }  
26  
27          System.out.println("End of main");  
28      }  
29  }
```

javaapplication7.Laga >

Output - JavaApplication7 (run) ✘

```
run:  
Start of main  
Exception Caught !!!  
End of main  
BUILD SUCCESSFUL (total time: 0 seconds)
```

# try-catch

```
11  public class ExceptionDemo {  
12  
13      static int div(int a, int b) {  
14          return a / b;  15      }  
16  
17      public static void main(String[] args) {  
18          System.out.println("Start of main");  
19           20          try {  
21              int result =  div(10, 0);  22              System.out.println("Result: " + result);  
23          } catch (Exception e) {  
24              System.out.println("Exception Caught !!!");  25          }  
26           27          System.out.println("End of main");  28      }  
29  }
```

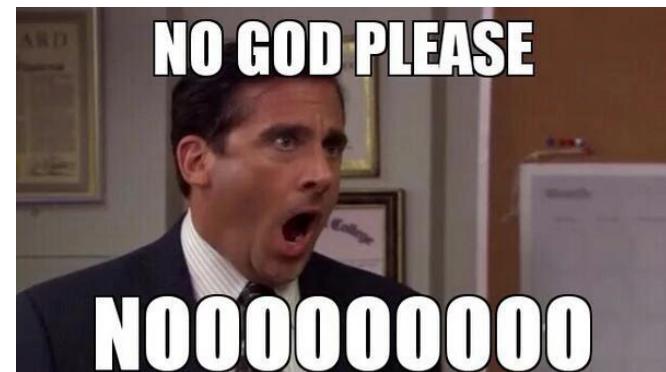
javaapplication7.Laga

Output - JavaApplication7 (run)

```
run:  
Start of main  
Exception Caught !!!  
End of main  
BUILD SUCCESSFUL (total time: 0 seconds)
```

# try-catch: Never do this!

```
try {  
    //  
    // Небезпечний код  
    //  
} catch (Exception ex) {}
```



# try-catch vs if

- Ніколи не використовуйте try-catch у якості операторів управління потоком виконання (if, break, return, ...) при реалізації позитивних сценаріїв виконання
- try-catch працює значно повільніше ніж if
- Тому try-catch потрібно використовувати для обробки лише виняткових ситуацій

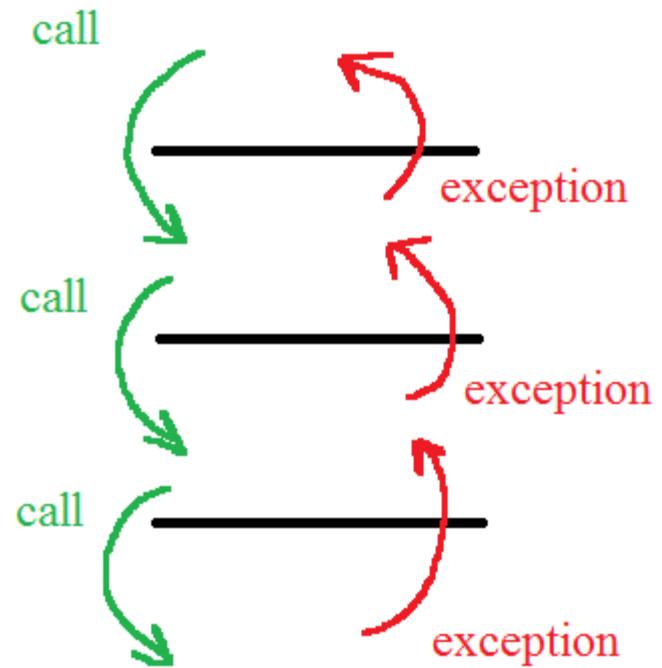
# try-catch vs if

```
class Something {  
    void doSomething() {  
    }  
}  
  
class SomeClass {  
  
    public static void main(String[] args) {  
  
        Something s = null;  
        // ...  
  
        // NEVER do this:  
        try {  
            s.doSomething();  
        } catch (NullPointerException npe) {  
            System.out.println("Oh no! Another NullPointerException");  
        }  
  
        // Do this:  
        if (s !=null) {  
            s.doSomething();  
        }  
    }  
}
```



# Root cause

- Root cause –  
виняткова ситуація,  
яка стала причиною  
виникнення іншої  
виняткової ситуації



# Root cause

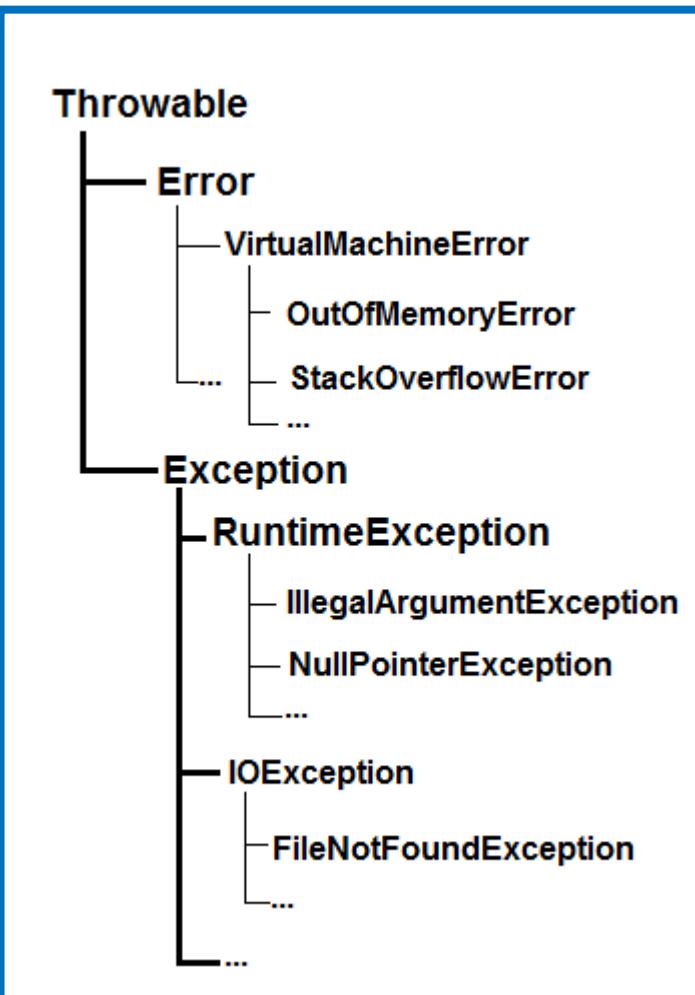
```
public class SomeClass {
    void doSomeIO() throws IOException {
        // ... some IO
        throw new IOException("Oops, IO not working...");
    }
    void doBusinessLogic() {
        // ... some complex business logic
        try {
            doSomeIO();
        } catch (IOException e) {
            throw new RuntimeException("Oops, can't do business logic", e);
        }
    }
    public static void main(String[] args) {
        SomeClass sm = new SomeClass();
        sm.doBusinessLogic();
    }
}
```

The diagram illustrates the flow of an exception from its source in the code to its final presentation in the terminal output. A red curved arrow originates from the line 'throw new IOException("Oops, IO not working...");' and points to the corresponding error message in the terminal. A green curved arrow originates from the line 'throw new RuntimeException("Oops, can't do business logic", e);' and points to the same error message in the terminal. Both arrows highlight the specific lines of code that caused the exception.

```
out - JavaApplication7 (run) ✘
run:
Exception in thread "main" java.lang.RuntimeException: Oops, can't do business logic
|   at javaapplication7.SomeClass.doBusinessLogic(SomeClass.java:27)
|   at javaapplication7.SomeClass.main(SomeClass.java:33)
Caused by: java.io.IOException: Oops, IO not working...
|   at javaapplication7.SomeClass.doSomeIO(SomeClass.java:18)
|   at javaapplication7.SomeClass.doBusinessLogic(SomeClass.java:25)
... 1 more
C:\Users\Aaz\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 1 second)
```

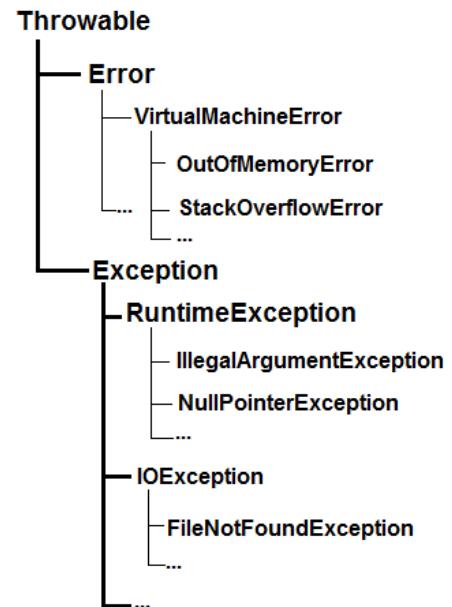
# Класи виняткових ситуацій

- Їх багато
  - насправді дуже багато 😊
  - Кожний описує певний тип проблем
- Перед тим, як вигадувати свій - краще заглянути у JavaDoc
  - Якщо стандартного не знайшлося, можна вигадувати свій власний



# Класи виняткових ситуацій

- Можна відловлювати виключення більш широкого типу
- Але краще не потрібно

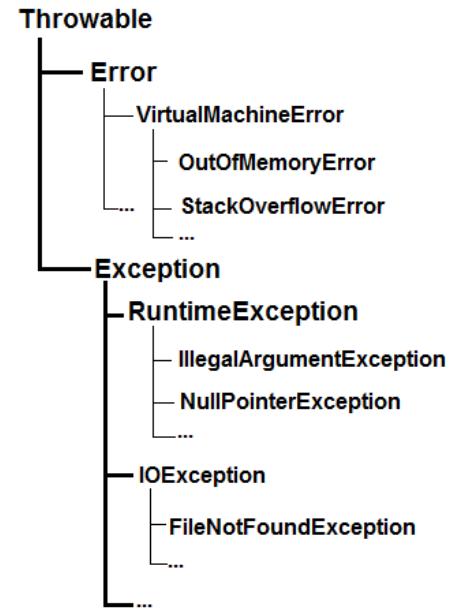


# Класи виняткових ситуацій

```
try {  
    FileInputStream fis = new FileInputStream("MyFile.txt");  
} catch (FileNotFoundException ex) {  
    // ...  
}
```

```
try {  
    FileInputStream fis = new FileInputStream("MyFile.txt");  
} catch (IOException ex) {  
    // ...  
}
```

```
try {  
    FileInputStream fis = new FileInputStream("MyFile.txt");  
} catch (Exception ex) {  
    // ...  
}
```



# Класи виняткових ситуацій

- У одного блока **try** може бути **кілька catch** для різних типів

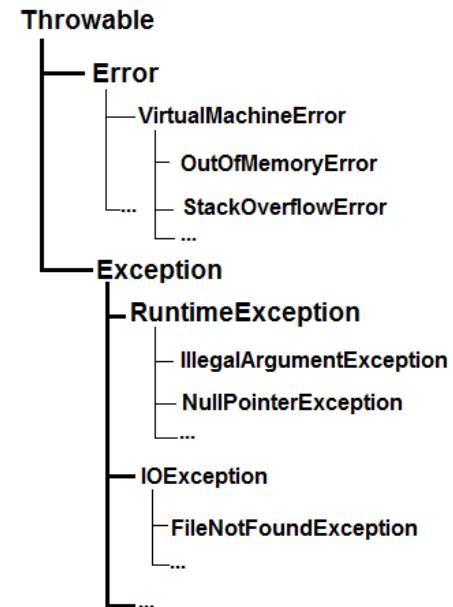
```
try {  
    // code that might throw one or more exceptions  
}  
catch (MyException e1) {  
    // code to execute if a MyException exception is thrown  
}  
catch (MyOtherException e2) {  
    // code to execute if a MyOtherException exception is thrown  
}  
catch (Exception e3) {  
    // code to execute if any other exception is thrown  
}
```



# Класи виняткових ситуацій

- Якщо у одного `try` є кілька `catch` для виняткових ситуацій, класи яких знаходяться у відношенні IS-A, вони мають йти «від конкретного до загального»
  - інакше помилка компіляції

```
try {  
    FileInputStream fis = new FileInputStream("MyFile.txt");  
} catch (FileNotFoundException ex) {  
    // ...  
} catch (IOException ex) {  
    // ...  
} catch (Exception ex) {  
    // ...  
}
```



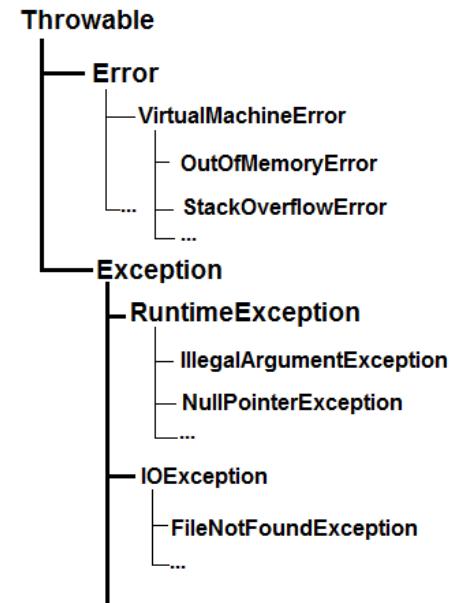
# Класи виняткових ситуацій

```
try {  
    FileInputStream fis = new FileInputStream("MyFile.txt");  
  
} catch (Exception ex) {  
    // ...  
}  
catch (FileNotFoundException ex) {  
    // ...  
}  
catch (IOException ex) {  
    // ...  
}
```

Помилка компіляції :

```
exception java.io.FileNotFoundException  
has already been caught
```

```
exception java.io.IOException  
has already been caught
```



# try-catch-finally

```
try {
```

*<Небезпечний код>;*

```
} catch (<ExceptionType> ex) {
```

*<Обробка виняткової ситуації>;*

```
} finally {
```

*<Фрагмент коду, який має виконатись незалежного від того був exception чи ні>;*

```
}
```

# try-catch-finally

- **Можна робити:**
  - try-catch
  - try-finally
  - try-catch-finally
- **finally спрацьовує завжди**
  - навіть при break, continue, return, throw
  - може не спрацювати при зупинці ВМ
    - `System.exit(1);`
  - може не доробити до кінця, якщо всередині finally буде викинуто новий Exception

# finally

```
20 public static void main(String[] args) {  
21     System.out.println("Hi");  
22  
23     int a = 1 / 0;  
24     System.out.println("Ok");  
25  
26     System.out.println("Bye");  
27 }  
28  
29 }
```

javaapplication7.Laga >

Output - JavaApplication7 (run) \*

```
run:  
Hi  
Exception in thread "main" java.lang.Arithmeti  
at javaapplication7.ExceptionDemo.main(Except  
C:\Users\Aaz\AppData\Local\NetBeans\Cache\8.2\executo  
BUILD FAILED (total time: 0 seconds)
```

```
20 public static void main(String[] args) {  
21     System.out.println("Hi");  
22     try {  
23         int a = 1 / 0;  
24         System.out.println("Ok");  
25     } finally {  
26         System.out.println("Bye");  
27     }  
28 }  
29  
30
```

javaapplication7.ExceptionDemo >

Output - JavaApplication7 (run) \*

```
run:  
Hi  
Bye  
Exception in thread "main" java.lang.Arithmeti  
at javaapplication7.ExceptionDemo.main(Except  
C:\Users\Aaz\AppData\Local\NetBeans\Cache\8.2\executo  
BUILD FAILED (total time: 0 seconds)
```

# finally

```
20 public static void main(String[] args) {  
21     System.out.println("Hi");  
22     try {  
23         int a = 1 / 1;  
24         System.out.println("Ok");  
25     } catch (Exception e) {  
26         System.out.println("NOT Ok");  
27     } finally {  
28         System.out.println("Bye");  
29     }  
30 }  
31 }  
32
```

javaapplication7.ExceptionDemo > div >

Output - JavaApplication7 (run) ✘

```
run:  
Hi  
Ok  
Bye  
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
20 public static void main(String[] args) {  
21     System.out.println("Hi");  
22     try {  
23         int a = 1 / 0;  
24         System.out.println("Ok");  
25     } catch (Exception e) {  
26         System.out.println("NOT Ok");  
27     } finally {  
28         System.out.println("Bye");  
29     }  
30 }  
31 }  
32
```

javaapplication7.ExceptionDemo > div > if(a < 0 || b < 0) >

Output - JavaApplication7 (run) ✘

```
run:  
Hi  
NOT Ok  
Bye  
BUILD SUCCESSFUL (total time: 0 seconds)
```

# throw

- **throw** – «викидає» exception назовні
  - слід використовувати у разі якщо щось пішло не так (аварійні випадки)
  - ніколи не використовуйте **throw** у якості оператора управління потоком виконання
    - для цього є `if`, `break`, `return` тощо

# throw - приклад

- Наприклад, хочемо заборонити ділити від'ємні значення

```
11  public class ExceptionDemo {  
12  
13      static int div(int a, int b) {  
14          if (a < 0 || b < 0) {  
15              throw new ArithmeticException();  
16          }  
17          return a / b;  
18      }  
19  }
```

# throw - приклад

```
11  public class ExceptionDemo {  
12  
13      static int div(int a, int b) {  
14          if (a < 0 || b < 0) {  
15              throw new ArithmeticException();  
16          }  
17          return a / b;  
18      }  
19  
20      public static void main(String[] args) {  
21          System.out.println("10 / 2 = ");  
22          System.out.println(div(10, 2));  
23  
24          System.out.println("10 / -2 = ");  
25          System.out.println(div(10, -2));  
26      }  
27  }  
28 }
```

The screenshot shows the NetBeans IDE interface. At the top, there's a toolbar with icons for file operations like New, Open, Save, and Run. Below the toolbar is a menu bar with options like File, Edit, View, Tools, Window, and Help. The main workspace contains two tabs: 'javaapplication7.Laga' (which is currently selected) and 'Output - JavaApplication7 (run)'. The code editor tab shows the Java code for 'ExceptionDemo'. The output tab shows the run command 'run:' followed by the program's output: '10 / 2 =' followed by the value '5'. Then it shows '10 / -2 =' followed by the error message 'Exception in thread "main" java.lang.ArithmeticException' with a stack trace. The stack trace indicates the exception was thrown at line 15 of 'ExceptionDemo.java' in the 'div' method, and at line 25 in the 'main' method. The output ends with 'Java returned: 1' and 'BUILD FAILED (total time: 0 seconds)'.

```
run:  
10 / 2 =  
5  
10 / -2 =  
Exception in thread "main" java.lang.ArithmeticException  
|  at javaapplication7.ExceptionDemo.div(ExceptionDemo.java:15)  
|  at javaapplication7.ExceptionDemo.main(ExceptionDemo.java:25)  
C:\Users\Aaz\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1  
BUILD FAILED (total time: 0 seconds)
```

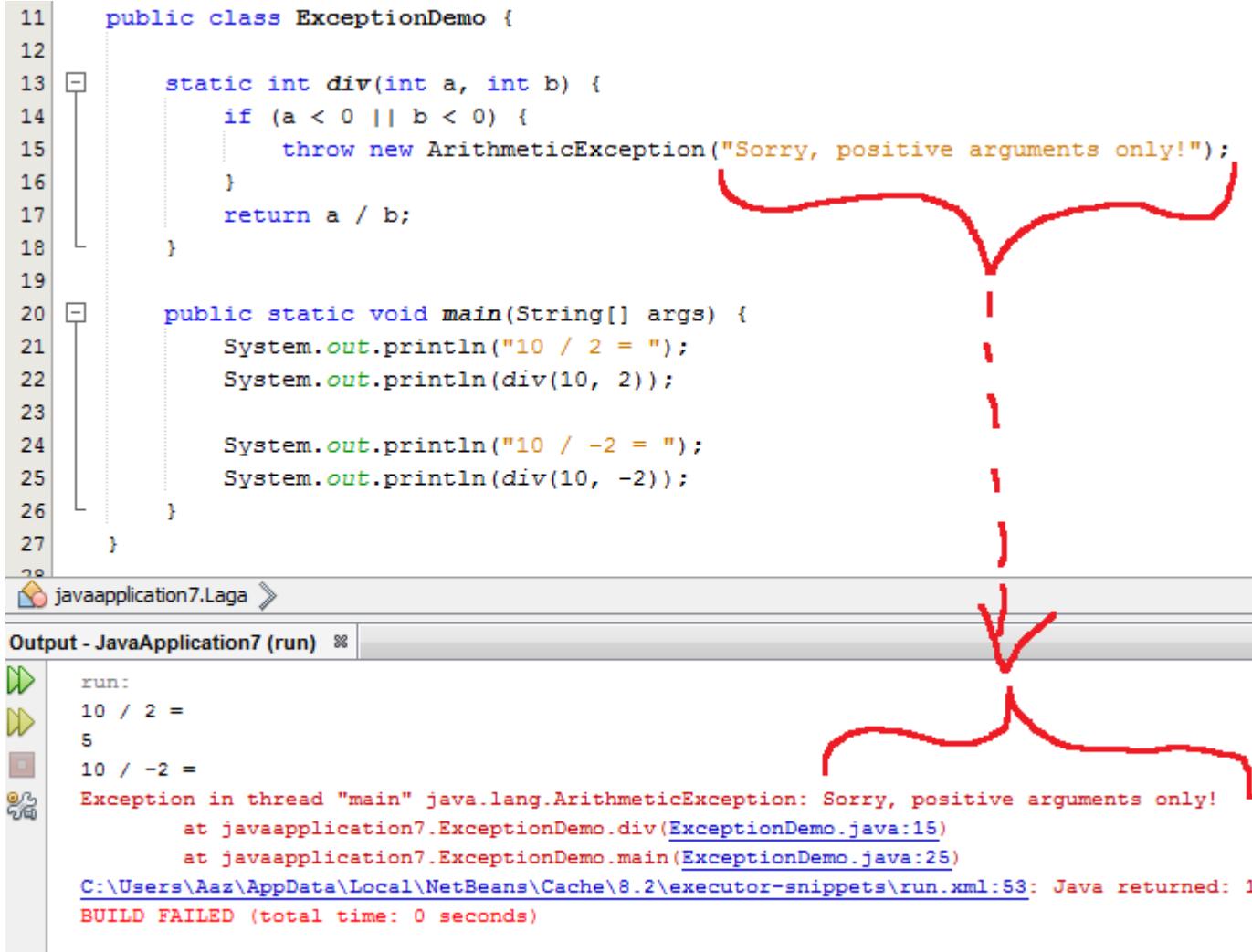
# throw new Exception(...)

- У конструктор Exception можна передати String, який буде відображеного у stack trace

```
static int div(int a, int b) {  
    if (a < 0 || b < 0) {  
        throw new ArithmeticException("Sorry, positive arguments only!");  
    }  
    return a / b;  
}
```

# throw new Exception(...)

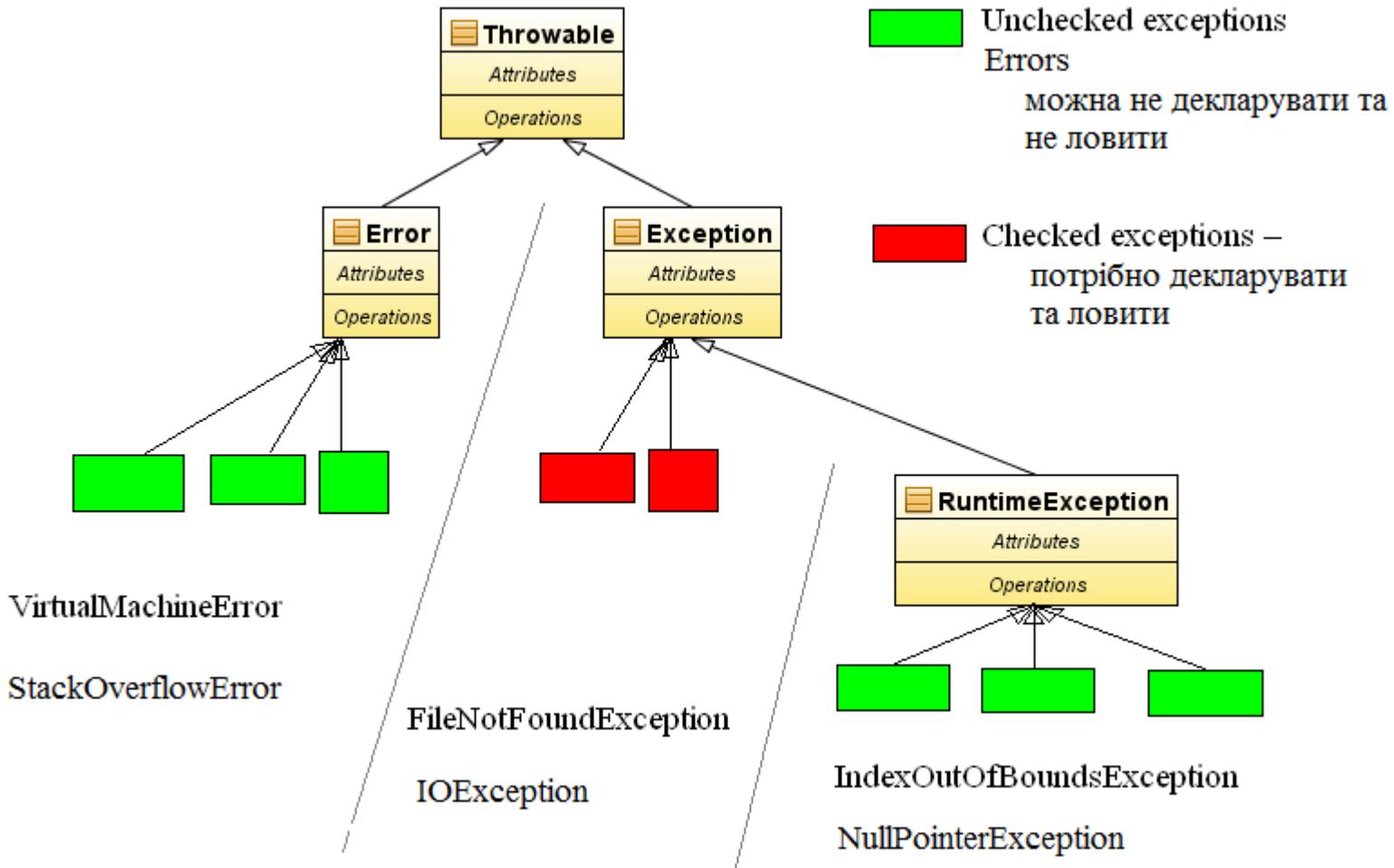
```
11  public class ExceptionDemo {  
12  
13      static int div(int a, int b) {  
14          if (a < 0 || b < 0) {  
15              throw new ArithmeticException("Sorry, positive arguments only!");  
16          }  
17          return a / b;  
18      }  
19  
20      public static void main(String[] args) {  
21          System.out.println("10 / 2 = ");  
22          System.out.println(div(10, 2));  
23  
24          System.out.println("10 / -2 = ");  
25          System.out.println(div(10, -2));  
26      }  
27  }  
  
javaapplication7.Laga  
Output - JavaApplication7 (run) ✘  
run:  
10 / 2 =  
5  
10 / -2 =  
Exception in thread "main" java.lang.ArithmeticException: Sorry, positive arguments only!  
    at javaapplication7.ExceptionDemo.div(ExceptionDemo.java:15)  
    at javaapplication7.ExceptionDemo.main(ExceptionDemo.java:25)  
C:\Users\Aaz\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1  
BUILD FAILED (total time: 0 seconds)
```



# Класифікація помилкових ситуацій

- Контрольовані (checked exceptions)
  - Компілятор гарантує, що якщо виникне checked exception, то десь у коді він буде підманий та оброблений
    - Це дозволяє писати хороші програми, які добре працюють у поганих умовах (IOException, FileNotFoundException)
  - Якщо з метода хочемо викинути checked exception, то маємо це задекларувати
  - Якщо викликаємо метод, що викидає checked exception, маємо його обробити або задекларувати
- Неконтрольовані (unchecked exceptions)
  - Компілятор не перевіряє обробку unchecked exceptions
    - Зазвичай вони виникають через «крайні руки» програмістів (NullPointerException, ArrayIndexOutOfBoundsException)
  - Можемо викидати unchecked exceptions з методів без попередньої декларації
  - Навіть якщо задекларувати unchecked exceptions, це ні на що не впливає
- Помилки (Error)
  - Компілятор не перевіряє обробку помилок
    - Зазвичай у разі помилки нічого зробити неможливо (VirtualMachineError, StackOverflowError)
    - Тому немає сенсу ловити

# Класифікація помилкових ситуацій



# Checked Exception Demo

```
11
12     class CheckedDemo {
13
14         void checkedBad() {
15             throw new IOException(); // ERROR: Checked Exception no declaration
16         }
17
18         void checkedGood() throws IOException {
19             throw new IOException(); // Checked Exception with declaration
20         }
21
22         void foo() {
23             try {
24                 checkedGood();
25             } catch (IOException e) {
26                 System.out.println("Catched!");
27             }
28         }
29
30         void bar() throws IOException {
31             checkedGood(); // No catch. Declaration needed!
32         }
33
34         void baz() {
35             checkedGood(); // unreported exception IOException; must be caught or declared to be thrown
36             // (Alt-Enter shows hints)
37         }
38     }
```

# Unchecked Exception Demo

```
11
12     public class UncheckedDemo {
13
14         void uncheckedGood() {
15             throw new RuntimeException(); // Ok: Unchecked Exception no declaration
16         }
17
18         void uncheckedLol() throws RuntimeException {
19             throw new RuntimeException(); // Useless declaration
20         }
21
22         void foo() {
23             try {
24                 uncheckedGood();
25             } catch (RuntimeException e) {
26                 System.out.println("Catched!"); // We can catch
27             }
28         }
29
30         void bar() throws RuntimeException {
31             uncheckedGood(); // Or we can declare
32         }
33
34         void baz() {
35             uncheckedGood(); // Or we can just call. No check at compile time
36         }
37     }
```

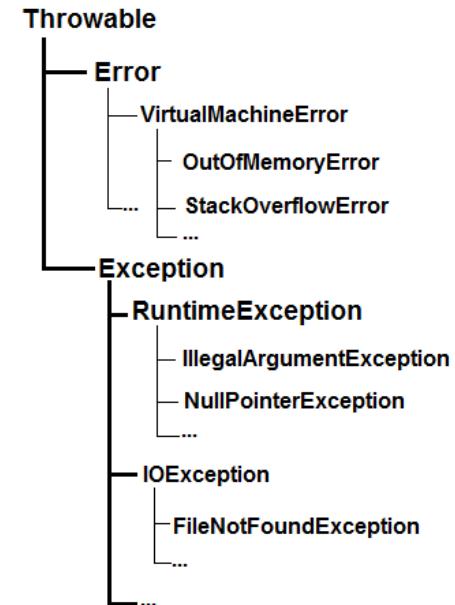
# Декларування винятків при наслідуванні

- При перевизначенні «throws-метода» **можна**:
  - вказати ті самі класи виняткових ситуацій;
  - конкретизувати (вказати класи-нащадки);
  - зовсім прибрати throws-декларацію.
- При перевизначенні «throws-метода» **неможна**:
  - узагальнювати (вказувати предків);
  - додавати нові виняткові ситуації.



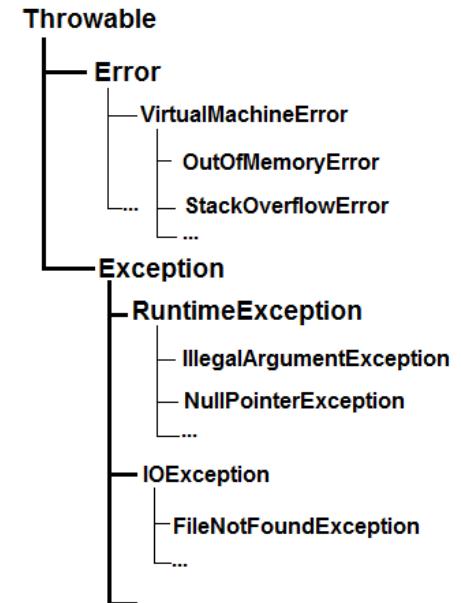
# Декларування винятків при наслідуванні

```
class A {  
    public void openFile() throws IOException {  
        FileInputStream fis = new  
            FileInputStream("MyFile.txt");  
    }  
}  
  
class B extends A {  
    public void openFile() throws FileNotFoundException {  
        FileInputStream fis = new  
            FileInputStream("MyFile.txt");  
    }  
}  
  
class C extends B {  
    public void openFile() {  
        try {  
            FileInputStream fis = new  
                FileInputStream("MyFile.txt");  
        } catch (FileNotFoundException e) {  
            // somehow solved  
        }  
    }  
}
```



# Декларування винятків при наслідуванні

```
class A {  
    public void openFile() throws FileNotFoundException {  
        FileInputStream fis = new  
            FileInputStream("MyFile.txt");  
    }  
}  
  
class B extends A {  
    public void openFile() throws IOException { // ERROR  
        FileInputStream fis = new  
            FileInputStream("MyFile.txt");  
    }  
}  
  
class C extends A {  
    public void openFile() throws FileNotFoundException, SQLException { // E  
        FileInputStream fis = new  
            FileInputStream("MyFile.txt");  
    }  
}
```



# Декларування винятків при наслідуванні

- Чому так?



- Підказка: **S O L I D**

# try-with-resources (JDK 7+)

Дуже часто до появи JDK 7 писали код типу:

```
try {  
    <Просимо у системи ресурси>;  
    <Працюємо з ресурсами>;  
  
} catch (<ExceptionType> ex) {  
    <Обробляємо виняткові ситуації>;  
  
} finally {  
    <Звільняємо ресурси>;  
    // викликаємо у них метод close();  
}
```

# try-with-resources (JDK 7+)

У JDK 7+ можна писати:

```
try(<Створюємо ресурси>) {  
    <Працюємо з ресурсами>;  
}  
} catch (<ExceptionType> ex) {  
    <Обробляємо виняткові ситуації>;  
}
```

- Код для вивільнення ресурсів руками писати не потрібно
- Метод **close()** виконується автоматично для всіх ресурсів, створених у **try(...)** якщо в них є інтерфейс **java.lang.AutoCloseable** або **java.io.Closeable**

# try-with-resources (JDK 7+)

```
// JDK 6
static String readFirstLineFromFileWithFinallyBlock(String path) throws IOException {

    BufferedReader br = new BufferedReader(new FileReader(path));

    try {
        return br.readLine();
    } finally {
        if (br != null) br.close();
    }
}
```

```
//JDK 7+
static String readFirstLineFromFile(String path) throws IOException {

    try (BufferedReader br = new BufferedReader(new FileReader(path))) {
        return br.readLine();
    }
}
```

# try-with-resources (JDK 7+)

Якщо ресурсів декілька, то через «;»:

```
try (
    java.util.zip.ZipFile zf = new java.util.zip.ZipFile(zipFileName);
    java.io.BufferedWriter writer = java.nio.file.Files.newBufferedWriter(outputFilePath, charset)
) {
    // Enumerate each entry
    for (java.util.Enumeration entries = zf.entries(); entries.hasMoreElements();) {

        // Get the entry name and write it to the output file
        String.newLine = System.getProperty("line.separator");
        String zipEntryName = ((java.util.zip.ZipEntry)entries.nextElement()).getName() + newLine;
        writer.write(zipEntryName, 0, zipEntryName.length());
    }
}
```