



# NIO 2.0 reference



Core Java: Day 4

Student Guide



# Exception and Error Handling, Input / Output, and File IO

# Exception and Error Handling, Input / Output, and File IO

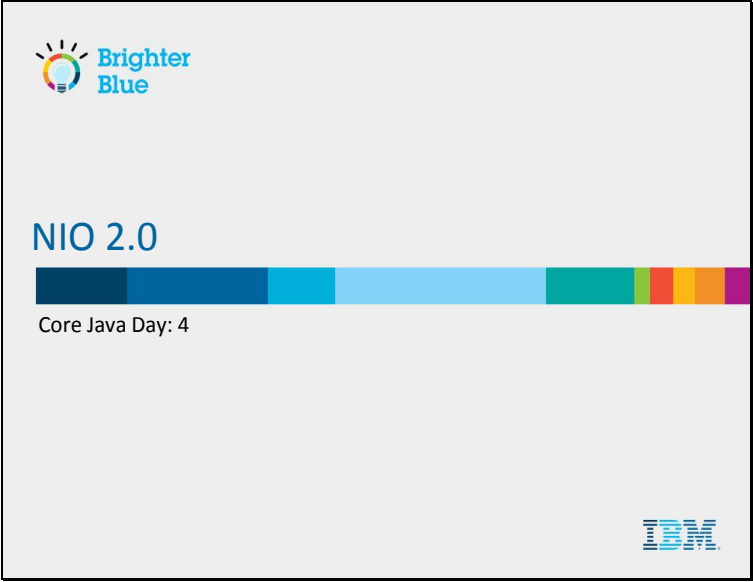
## Contents

**NIO 2.0**



---

**4**

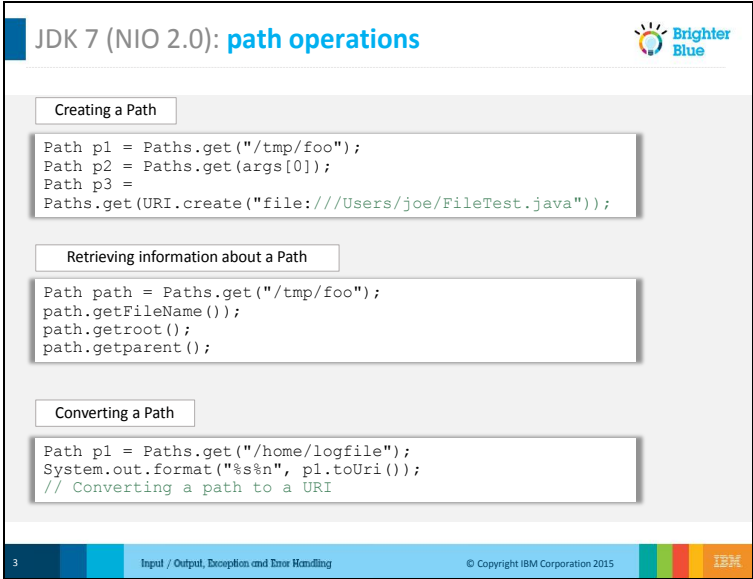
# Exception and Error Handling, Input / Output, and File IO

Slide Content	Use this space for your own notes
<p>Slide 1</p>  <p>Brighter Blue</p> <p>NIO 2.0</p> <p>Core Java Day: 4</p> <p>IBM</p>	

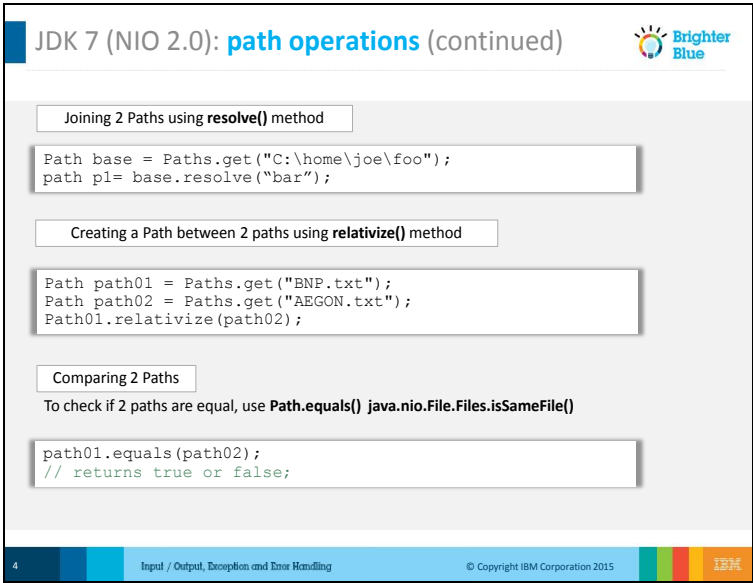
# Exception and Error Handling, Input / Output, and File IO

Slide Content	Use this space for your own notes
<p>Slide 2</p> <div data-bbox="205 430 949 1003">  <h2>JDK 7 (NIO 2.0): The new path class and file system</h2> <ul style="list-style-type: none"> <li>NIO 2.0 introduces a new abstract class – <code>java.nio.file.Path</code></li> <li>It supports two types of operations: <ul style="list-style-type: none"> <li><i>syntactic operations</i></li> <li><i>operations over files</i></li> </ul> </li> <li>A Path resides in file system.</li> </ul> <p><b>The old way:</b></p> <pre>import java.io.File;... File file = new File("index.html");</pre> <p><b>Java 7 way:</b></p> <pre>import java.nio.file.Path; import java.nio.file.Paths; ... Path path = Paths.get("index.html");</pre> <ul style="list-style-type: none"> <li>Its important methods are: <code>get Default ()</code> <code>getFileSystem (URI uri)</code></li> </ul> <p>2    Input / Output, Exception and Error Handling    © Copyright IBM Corporation 2015    </p> </div>	

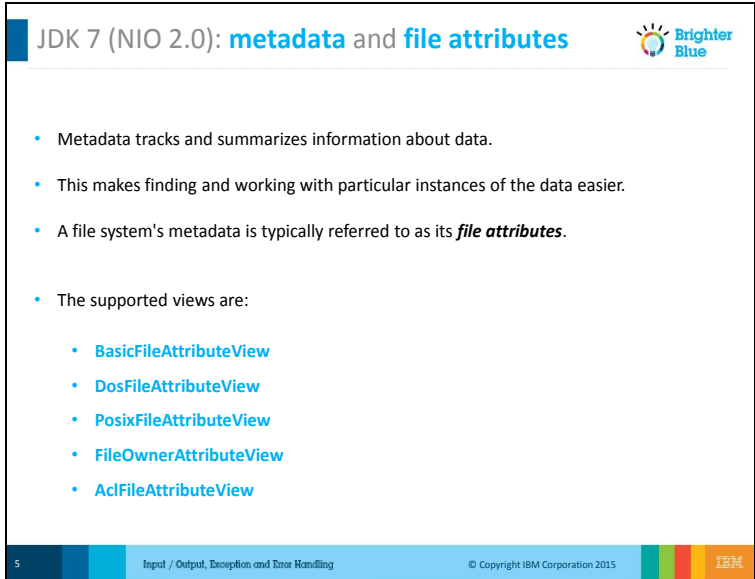
# Exception and Error Handling, Input / Output, and File IO

Slide Content	Use this space for your own notes
<p>Slide 3</p> <div data-bbox="201 480 949 1055">  <p><b>JDK 7 (NIO 2.0): path operations</b></p> <p><b>Creating a Path</b></p> <pre>Path p1 = Paths.get("/tmp/foo"); Path p2 = Paths.get(args[0]); Path p3 = Paths.get(URI.create("file:///Users/joe/FileTest.java"));</pre> <p><b>Retrieving information about a Path</b></p> <pre>Path path = Paths.get("/tmp/foo"); path.getFileName(); path.getRoot(); path.getParent();</pre> <p><b>Converting a Path</b></p> <pre>Path p1 = Paths.get("/home/logfile"); System.out.format("%s\n", p1.toUri()); // Converting a path to a URI</pre> <p>3 Input / Output, Exception and Error Handling © Copyright IBM Corporation 2015 IBM</p> </div>	

# Exception and Error Handling, Input / Output, and File IO

Slide Content	Use this space for your own notes
<p>Slide 4</p> <div data-bbox="201 418 949 997">  <p>JDK 7 (NIO 2.0): <b>path operations</b> (continued)</p> <p>Joining 2 Paths using <b>resolve()</b> method</p> <pre>Path base = Paths.get("C:\home\joe\foo"); path p1= base.resolve("bar");</pre> <p>Creating a Path between 2 paths using <b>relativize()</b> method</p> <pre>Path path01 = Paths.get("BNP.txt"); Path path02 = Paths.get("AEGON.txt"); Path01.relativize(path02);</pre> <p>Comparing 2 Paths</p> <p>To check if 2 paths are equal, use <b>Path.equals()</b> <b>java.nio.File.Files.isSameFile()</b></p> <pre>path01.equals(path02); // returns true or false;</pre> <p>4 Input / Output, Exception and Error Handling © Copyright IBM Corporation 2015 IBM</p> </div>	

# Exception and Error Handling, Input / Output, and File IO



Slide Content	Use this space for your own notes
<p>Slide 5</p> <div data-bbox="199 440 949 1016">  <p>JDK 7 (NIO 2.0): <b>metadata</b> and <b>file attributes</b></p> <ul style="list-style-type: none"> <li>• Metadata tracks and summarizes information about data.</li> <li>• This makes finding and working with particular instances of the data easier.</li> <li>• A file system's metadata is typically referred to as its <b>file attributes</b>.</li> <li>• The supported views are: <ul style="list-style-type: none"> <li>• <code>BasicFileAttributeView</code></li> <li>• <code>DosFileAttributeView</code></li> <li>• <code>PosixFileAttributeView</code></li> <li>• <code>FileOwnerAttributeView</code></li> <li>• <code>AclFileAttributeView</code></li> </ul> </li> </ul> <p>5 Input / Output, Exception and Error Handling © Copyright IBM Corporation 2015 IBM</p> </div>	





# Exception and Error Handling, Input / Output, and File IO

Slide Content	Use this space for your own notes
<p>Slide 6</p> <div data-bbox="199 440 949 1016"> <p>JDK 7 (NIO 2.0): <b>symbolic</b> and <b>hard links</b></p> <div data-bbox="291 565 459 594">Symbolic Link</div> <div data-bbox="701 565 869 594">Hard Link</div> <div data-bbox="275 613 478 678"> <p>Is a special type of file that contains a <b>reference</b> to another <b>file</b> or <b>directory</b></p> </div> <div data-bbox="688 613 892 678"> <p>Is a <b>directory entry</b> that <b>associates a name</b> with a <b>file</b> on a <b>file system</b></p> </div> <div data-bbox="256 704 491 919"> <ul style="list-style-type: none"> <li>Can link to a file or directory</li> <li>Can exist across file systems</li> <li>Existence of a target is not compulsory</li> <li>On removing the original file, the attached symbolic link becomes useless</li> <li>More flexible, as its target may not even exist</li> </ul> </div> <div data-bbox="659 704 894 919"> <ul style="list-style-type: none"> <li>Can't link to directories</li> <li>Can't exist across file systems</li> <li>Existence of the target is compulsory</li> <li>On removing the original file, the hard link still provides the content of the file</li> <li>Looks and behaves like regular files, so, hard to find</li> </ul> </div> </div> <p>As mentioned previously, the <code>java.nio.file</code> package, and the <code>Path</code> class in particular, is "link aware."</p>	


# Exception and Error Handling, Input / Output, and File IO

Slide Content	Use this space for your own notes										
<p>Slide 7</p> <div> <p><b>Important methods:</b> symbolic and hard links </p> <table> <tr> <th>Methods</th><th>Representation of links with examples</th></tr> <tr> <td><code>createSymbolicLink(Path, Path, FileAttribute&lt;?&gt;)</code></td><td><b>Creating a symbolic link:</b> <code>Files.createSymbolicLink(newLink, target);</code></td></tr> <tr> <td><code>createLink(Path, Path)</code></td><td><b>Creating a hard link to an existing file:</b> <code>Files.createLink(newLink, existingFile);</code></td></tr> <tr> <td><code>isSymbolicLink(Path)</code></td><td><b>Detecting a symbolic link:</b> To determine whether a Path instance is a symbolic link. <code>boolean isSymbolicLink = Files.isSymbolicLink(file);</code></td></tr> <tr> <td><code>readSymbolicLink(Path)</code></td><td><b>Finding the target of a link:</b> <code>Files.readSymbolicLink(link)</code></td></tr> </table> </div> <p>7 Input / Output, Exception and Error Handling © Copyright IBM Corporation 2015 </p>	Methods	Representation of links with examples	<code>createSymbolicLink(Path, Path, FileAttribute&lt;?&gt;)</code>	<b>Creating a symbolic link:</b> <code>Files.createSymbolicLink(newLink, target);</code>	<code>createLink(Path, Path)</code>	<b>Creating a hard link to an existing file:</b> <code>Files.createLink(newLink, existingFile);</code>	<code>isSymbolicLink(Path)</code>	<b>Detecting a symbolic link:</b> To determine whether a Path instance is a symbolic link. <code>boolean isSymbolicLink = Files.isSymbolicLink(file);</code>	<code>readSymbolicLink(Path)</code>	<b>Finding the target of a link:</b> <code>Files.readSymbolicLink(link)</code>	
Methods	Representation of links with examples										
<code>createSymbolicLink(Path, Path, FileAttribute&lt;?&gt;)</code>	<b>Creating a symbolic link:</b> <code>Files.createSymbolicLink(newLink, target);</code>										
<code>createLink(Path, Path)</code>	<b>Creating a hard link to an existing file:</b> <code>Files.createLink(newLink, existingFile);</code>										
<code>isSymbolicLink(Path)</code>	<b>Detecting a symbolic link:</b> To determine whether a Path instance is a symbolic link. <code>boolean isSymbolicLink = Files.isSymbolicLink(file);</code>										
<code>readSymbolicLink(Path)</code>	<b>Finding the target of a link:</b> <code>Files.readSymbolicLink(link)</code>										



# Exception and Error Handling, Input / Output, and File IO

Slide Content	Use this space for your own notes
<p>Slide 8</p> <div data-bbox="201 443 949 1018"> <p><b>FileVisitor interface:</b> walking the file tree (1 of 3) </p> <ul style="list-style-type: none"> <li>• A <b>FileVisitor</b> needs to be implemented before walking a file tree.</li> <li>• The <b>FileVisitor</b> specifies the required behavior at key points in the traversal process.</li> </ul> <p><b>Methods Implemented:</b></p> <pre>preVisitDirectory(T dir, BasicFileAttributes attrs) throws IOException</pre> <pre>postVisitDirectory(T dir, IOException exc) throws IOException</pre> <pre>visitFile(T file, BasicFileAttributes attrs) throws IOException</pre> <pre>visitFileFailed(T file, IOException exc) throws IOException</pre> <p>8    Input / Output, Exception and Error Handling    © Copyright IBM Corporation 2015    </p> </div>	

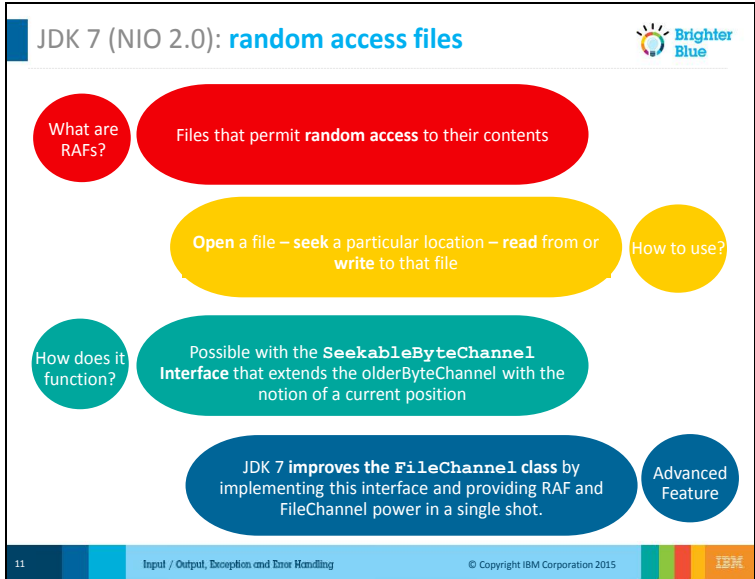
# Exception and Error Handling, Input / Output, and File IO

Slide Content	Use this space for your own notes
<p>Slide 9</p> <div data-bbox="201 443 951 1018"><div><div></div><div><b>FileVisitor interface:</b> walking the file tree (2 of 3)</div><div>Brighter Blue</div></div><p>Enum constant details of the <b>FileVisitResult</b>:</p><div><div>FileVisitResult.<b>CONTINUE</b></div><div>FileVisitResult.<b>SKIP_SIBLINGS</b></div><div>FileVisitResult.<b>SKIP_SUBTREE</b></div><div>FileVisitResult.<b>TERMINATE</b></div></div><div><div>9</div><div>Input / Output, Exception and Error Handling</div><div>© Copyright IBM Corporation 2015</div><div></div></div></div>	

# Exception and Error Handling, Input / Output, and File IO

Slide Content	Use this space for your own notes
<p>Slide 10</p> <div data-bbox="201 443 949 1018">  <p><b>FileVisitor interface:</b> walking the file tree (3 of 3)</p> <p>Consider traversing a file tree and list the names of all directories:</p> <pre> class ListTree extends SimpleFileVisitor&lt;Path&gt; {     @Override     public FileVisitResult postVisitDirectory(Path, dir, IOException exc) {         System.out.println("Visited directory:" + dir.toString());         return FileVisitResult.CONTINUE;     }     @Override     public FileVisitResult visitFileFailed(Path file, IOException exc) {         System.out.println(exc);         return FileVisitResult.CONTINUE;     } } </pre> <p>10    Input / Output, Exception and Error Handling    © Copyright IBM Corporation 2015    </p> </div>	

# Exception and Error Handling, Input / Output, and File IO

Slide Content	Use this space for your own notes
<p>Slide 11</p> <div data-bbox="199 440 949 1016">  <p><b>JDK 7 (NIO 2.0): random access files</b></p> <p><b>What are RAFs?</b> Files that permit <b>random access</b> to their contents</p> <p><b>Open a file – seek a particular location – read from or write to that file</b> <b>How to use?</b></p> <p><b>How does it function?</b> Possible with the <b>SeekableByteChannel Interface</b> that extends the older <b>ByteChannel</b> with the notion of a current position</p> <p><b>JDK 7 improves the FileChannel class</b> by implementing this interface and providing RAF and FileChannel power in a single shot. <b>Advanced Feature</b></p> <p>11 Input / Output, Exception and Error Handling © Copyright IBM Corporation 2015 IBM</p> </div>	