Correction Java IO File (NIO.2) - 0

C - B\C\Book.java

Root folder or drive is not considered in count and indexing. In the given path A is at 0th index, B is at 1st index, C is at 2nd index and Book.java is at 3rd index.

In ‘subpath(int beginIndex, int endIndex)’ method beginIndex in inclusive and endIndex is exclusive. So, in the given question, starting index is 1 and end index is 3.

So, ‘path.subpath(1,4)’ returns ‘B.java’.

Correction Java IO File (NIO.2) - 1

B - Above program executes successfully and prints below lines on to the console:

```
F:Parent\a.txt
F:Parent\b.txt
```

String class has endsWith method, and the lambda expression '(p,a) -> p.toString().endsWith("txt")' will return all the paths ending with "txt".

Signature of find method is:

Stream find(Path start, int maxDepth, BiPredicate<Path, BasicFileAttributes> matcher, FileVisitOption... options)

and in the code, following syntax is used: Files.find(root, 2, predicate).

root refers to 'F:' and maxDepth is 2. This means look out for all the files under F: (depth 1) and all the files under the directories whose immediate parent is F: (depth 2).

So in this case, F: and Parent directory are searched for the matching files. 'F:Parent\a.txt' and 'F:Parent\b.txt' are printed on to the console.

Correction Java IO File (NIO.2) - 2

D - Program executes successfully and prints ‘Welcome!’ on to the console

First of all I would like to tell you that Windows shortcut and symbolic links are different.

Shortcut is just a regular file and symbolic link is a File System object.

To create symbolic link used in this question, I used below command:

C:\TEMP>mklink msg .\Parent.txt

And below message was displayed on to the console for the successful creation of symbolic link ‘symbolic link created for msg <<===>> .\Parent\Child\Message.txt’.

Files class has methods such as newInputStream(…), newOutputStream(…), newBufferedReader(…) and newBufferedWriter(…) for files reading and writing.

Give code doesn’t cause any compilation error.

path refers to ‘C\TEMP\msg’, which is a symbolic link and hence Files.newBufferedReader(path) works with ‘C:\TEMP\Parent\Child\Message.txt’.

Given code successfully reads the file and prints ‘Welcome!’ on to the console.

Correction Java IO File (NIO.2) - 3

B - Files.readAllLines(Paths.get(“F:\Book.java”)).forEach(System.out::println);

C - Files.readAllLines(Paths.get(“F:\Book.java”)).stream().forEach(System.out::println);

D - Files.lines(Paths.get(“F:\Book.java”)).forEach(System.out::println);

Below are the declarations of lines and readAllLines methods from Files class:

public static Stream lines(Path path) throws IOException {…}

public static List readAllLines(Path path) throws IOException {…}

‘Files.lines(Paths.get(“F:\Book.java”))’ returns Stream object. Hence forEach() can be invoked but stream() can’t be invoked.

‘Files.readAllLines(Paths.get(“F:\Book.java”))’ returns List object. Hence both forEach() and stream() methods can be invoked. List has both the methods. But converting list to stream() and then invoking forEach() method is not required but it is a legal syntax and prints the file contents.

Correction Java IO File (NIO.2) - 4

D - It will only print the paths of directories and files under HOME directory.

Files.list(Path) returns the object of Stream containing all the paths (files and directories) of current directory. It is not recursive.

For recursive access use overloaded Files.walk() methods.

Correction Java IO File (NIO.2) - 5

C -

```
..\..
B\C
```

For ‘path1.relativize(path2)’ both path1 and path2 should be of same type. Both should either be relative or absolute.

In this case, path1 refers to ‘F:\A\B\C’ and path2 refers to ‘F:\A’.

To easily tell the resultant path, there is a simple trick. Just remember this for the exam.

path1.relativize(path2) means how to reach path2 from path1. It is by doing ‘cd ..\..’ so, 1st output is ‘..\..’

path2.relativize(path1) means how to reach path1 from path2. It is by doing ‘cd B’ so, 2nd output is ‘B’.

Correction Java IO File (NIO.2) - 6

C - An exception is thrown at runtime

For ‘path1.relativize(path2)’ both path1 and path2 should be of same type. Both should either be relative or absolute.

In this case, path1 refers to ‘C:’ and path2 refers to ‘D:’.

Even though both paths are absolute but their roots are different, hence IllegalArgumentException is thrown at runtime.

Correction Java IO File (NIO.2) - 7

A - F:\A\B\Book.java

toAbsolutePath() method doesn’t care if given path elements are physically available or not. It just returns the absolute path.

As file already refers to absolute path, hence the same path is printed on to the console.

Correction Java IO File (NIO.2) - 8

A - System.out.println(Files.getAttribute(path, “isDirectory”));

B - System.out.println(path.toFile().isDirectory());

C - System.out.println(Files.isDirectory(path));

‘B’ is a directory.

java.nio.file.Files (not java.io.File) class has static method isDirectory(Path) to check if the farthest element of given path is directory or not. ‘Files.isDirectory(path)’ returns true.

Interface Path has toFile() method to return the java.io.File object representing this path. And java.io.File class has isDirectory() method to check if given File object is directory or not. ‘path.toFile().isDirectory()’ returns true.

Files.getAttribute(Path path, String attribute, LinkOption… options) returns the value corresponding to passed attribute. IllegalArgumentException is thrown if attribute is not spelled correctly.

Files.getAttribute(path, “isDirectory”) returns true.

There is no static method, isDirectory(Path) in java.io.File class, hence ‘File.isDirectory(path)’ causes compilation error.

There is no constructor of File class accepting Path, hence new File(path) causes compilation error.

Correction Java IO File (NIO.2) - 9

B - true:false

First of all I would like to tell you that Windows shortcut and symbolic links are different.

Shortcut is just a regular file and symbolic link is a File System object.

To create symbolic link used in this question, I used below command:

C:\TEMP>mklink msg .\Parent\Child\Message.txt

And below message was displayed on to the console for the successful creation of symbolic link ‘symbolic link created for msg <<===>> .\Parent\Child\Message.txt’.

When copy() method is used for symbolic link, then by default target file is copied and not the symbolic link.

So, ‘src’ is a symbolic link an ‘tgt’ is a regular file. ‘true:false’ is printed on to the console.

NOTE: For the job, if you want to copy symbolic link, then use ‘Files.copy(src, tgt, LinkOption.NOFOLLOW_LINKS);’ but make sure that user should have ‘symbolic’ LinkPermission.

Correction Java IO File (NIO.2) - 10

A - An exception is thrown at runtime.

‘Files.createDirectory(path);’ creates the farthest directory element but all parents must exist.

In this case, createDirectory method tries to create ‘Z’ directory under F:\X\Y.

F:\X exists but F:\X\Y doesn’t exist and hence NoSuchFileException is thrown at runtime.

Correction Java IO File (NIO.2) - 11

E - 3, C, C

Root folder or drive is not considered in count and indexing. In the given path A is at 0th index, B is at 1st index and C is at 2nd index.

path.getName(2) returns ‘C’.

path.getNameCount() returns 3 (A,B,C) and path.getFileName() returns the last name of the path, which is ‘C’.

Given methods doesn’t need actual path to physically exist and hence no exception is thrown at Runtime.

Correction Java IO File (NIO.2) - 12

A - java.nio.file.FileAlreadyExistsException is thrown at runtime

Path is an abstract path and farthest element can be a file or directory.

src refers to Path object, F:\A\B\C\Book.java and its farthest element, ‘Book.java’ is a file (and it exists).

tgt refers to Path object, F:\A\B and its farthest element, ‘B’ is a directory (and it exists).

Files.copy(src, tgt) copies the farthest element. ‘Book.java’ can’t be copied to ‘B’ as it is a directory and it is not allowed to have file and directory of same name to be present at same location. Hence, java.nio.file.FileAlreadyExistsException is thrown at runtime.

If you change tgt to Paths.get(“F:\A\C\”); then ‘Files.copy(src, tgt);’ will successfully copy ‘Book.java’ to ‘C’ (‘C’ will be a file in that case, containing the contents from ‘Book.java’ file).

Correction Java IO File (NIO.2) - 13

C - C:\classes\Book.java

Paths.get(“Book.java”); represents a relative path. This means relative to current directory.

‘Test.class’ file is available under “C:\classes” directory, hence path of ‘Book.java’ is calculated relative to C:\classes.

file.toAbsolutePath() returns ‘C:\classes\Book.java’.

NOTE: toAbsolutePath() method doesn’t care if given path elements are physically available or not. It just returns the absolute path.

Correction Java IO File (NIO.2) - 14

C - HELLO will be printed once and FAILED will be printed twice

Files.walk(Paths.get(“F:\process”)) returns the object of Stream containing “F:\process”, “F:\process.txt”, “F:\process\file.docx” and “F:\process.pdf”.

paths.filter(path -> !Files.isDirectory(path)) filters out the directory “F:\process” and process the 3 files.

Files.readAllLines method reads all the lines from the files using StandardCharsets.UTF_8 but as pdf and docx files use different Charset, hence exception is thrown for reading these files.

FAILED will be printed twice for pdf and docx files. And HELLO (content of file.txt) will be printed once.

Files.readAllLines(path, StandardCharsets.ISO_8859_1) may allow you to read all the 3 files without Exception but these files store lots of other information font, formatting etc. so output may not make any sense.

Correction Java IO File (NIO.2) - 15

C - The code executes successfully and deletes symbolic link file ‘msg’

According to the javadoc comment of delete method, if the file is a symbolic link then the symbolic link itself, not the final target of the link, is deleted.

Correction Java IO File (NIO.2) - 16

C - Directory Y will be created under X and directory Z will be created under Y

‘Files.createDirectories(path);’ creates a directory by creating all nonexistent parent directories first.

So this method first creates directory Y under X and then directory Z under Y.

Correction Java IO File (NIO.2) - 17

A - F:\training..

path –> {F:\user\..\training..}. path.normalize() will return {F:\training..}.

NOTE: double dot with ‘training’ is not removed as these are not path symbol.

Correction Java IO File (NIO.2) - 18

A - true

path1 –> [F:\Other\Logs].

path2 –> [..\..\Child.lnk\Message.txt].

path1.resolve(path2) –> [F:\Other\..\..\Shortcut\Child.lnk\Message.txt].

path3 –> [F:\Shortcut\Child.lnk\Message.txt].

path1.resolveSibling(path2) –>
[F:\Other\..\..\Shortcut\Child.lnk\Message.txt].

path4 –> [F:\Shortcut\Child.lnk\Message.txt].

This is interesting, if you are at the root directory, and give the command cd .., then nothing happens, you stay at the root only.

System.out.println(Paths.get("F:\..\..\..\..").normalize()); would print F:\.

This is the reason, why path4 is referring to
[F:\Shortcut\Child.lnk\Message.txt] and no exception is thrown at runtime.

As path3 and path4 refer to same location, hence path3.equals(path4) returns true.

Correction Java IO File (NIO.2) - 19

D -

```
F:\A\B\C\Book.java
F:\A\B\Book.java
```

file1.resolve(file2) resolves file2 against file1. file1 is an absolute path and file2 is a relative path, hence resolve method returns Path object referring to 'F:\A\B\C\Book.java'.

file1.resolveSibling(file2) resolves file2 against parent path of file1. Parent path of file1 is: 'F:\A\B\', hence resolveSibling method returns Path object referring to 'F:\A\B\Book.java'.

Correction Java IO File (NIO.2) - 20

B - F:\A\B\C\Book.java

toRealPath() returns the path of an existing file. It returns the path after normalizing.

Let's first normalize the path.

"F:\A\.\B\C\D\..\Book.java"

can be normalized to "F:\A\B\C\D\..\Book.java" [Single dot is for current directory, hence it is redundant].

can be further normalized to "F:\A\B\C\Book.java" [Double dot is for going to parent directory, hence dir 'D' is removed].

'F:\A\B\C\Book.java' exists on the file system, hence no exception.

Correction Java IO File (NIO.2) - 21

C - true

path refers to ‘F:\A’, path.getRoot() refers to ‘F:\’ and path.getParent() refers to ‘F:\’.

Hence result is ‘true’.

Correction Java IO File (NIO.2) - 22

C -

```
false
true
```

‘Files.copy(src, tgt);’ copies ‘F:\A\B\C\Book.java’ to ‘F:\A\B\Book.java’ and returns the Path of copied element.

src refers to ‘F:\A\B\C\Book.java’.

tgt refers to ‘F:\A\B\Book.java’.

copy refers to ‘F:\A\B\Book.java’.

Files.isSameFile(Path path1, Path path2) returns true if both the paths locate the same physical file.

src and copy refer to different physical files, hence ‘Files.isSameFile(src, copy)’ returns false.

tgt and copy refer to same physical file, hence ‘Files.isSameFile(tgt, copy)’ returns true.

Correction Java IO File (NIO.2) - 23

B - Exception is thrown at runtime

Root folder or drive is not considered in count and indexing. In the given path A is at 0th index, B is at 1st index, C is at 2nd index and Book.java is at 3rd index.

In ‘subpath(int beginIndex, int endIndex)’ method beginIndex in inclusive and endIndex is exclusive.

So, in the given question, starting index is 1 and end index is 4. In the given path there is no element at the 4th index, hence an exception is thrown at runtime.

In fact, subpath(int beginIndex, int endIndex) throws IllegalArgumentException if ‘beginIndex >= No. of path elements’, ‘endIndex > No. of path elements’ and ‘endIndex <= beginIndex’.

Correction Java IO File (NIO.2) - 24

A -

```
for(int i = 6; i < path.getNameCount(); i++) {
  System.out.println(path.getName(i));
}
```

B -

```
Iterator<Path> iterator = path.iterator();
while(iterator.hasNext()) {
  System.out.println(iterator.next());
}
```

C -

```
for(Path p : path) {
  System.out.println(p);
}
```

D -

```
path.forEach(System.out::println);
```

All 4 are the correct way to iterate through path elements.

Root folder or drive is not considered in count and indexing. In the given path A is at 0th index, B is at 1st index, C is at 2nd index and Book.java is at 3rd index.

path.getNameCount() returns 4.

Correction Java IO File (NIO.2) - 25

E - Compilation error

endsWith method is overloaded in Path interface:

boolean endsWith(Path other);

boolean endsWith(String other);

‘p.endsWith(null)’ causes compilation error as it is an ambiguous method call.

Correction Java IO File (NIO.2) - 26

B - Above program executes successfully and prints nothing on to the console

endsWith method is overloaded in Path interface:

boolean endsWith(Path other);

boolean endsWith(String other);

Even though endsWith(String) accepts String but it should evaluate to pathname, such as “Child” OR “a.txt” but not just a part of pathname, such as “txt”.

p.endsWith(“txt”) will return false for all the available paths and hence nothing will get printed on to the console.

NOTE: If you want to find the files ending with “txt” then use ‘p.toString().endsWith(“txt”)’ in the lambda expression.

Correction Java IO File (NIO.2) - 27

C - true

static method Files.size(path) method is equivalent to instance method length() defined in java.io.File class.

Both returns length of the file, in bytes.

Correction Java IO File (NIO.2) - 28

A - Compilation Error

toRealPath() returns the path of an existing file. It returns the path after normalizing.

As toRealPath() works with existing file, there is a possibility of I/O error hence toRealPath() method declares to throw IOException, which is a checked exception.

Given code doesn't handle or declare IOException and that is why 'file.toRealPath()' causes compilation error.

Correction Java IO File (NIO.2) - 29

B - Contents of Book.java are printed on to the console.

Files class has methods such as newInputStream(…), newOutputStream(…), newBufferedReader(…) and newBufferedWriter(…) for files reading and writing.

Given code doesn’t cause any compilation error.

As Book.java is a text file and accessible, hence its contents are printed on to the console.

Correction Java IO File (NIO.2) - 30

D - F:\A\.\B\C\D\..\Book.java

Implementations of Path interface are immutable, hence path.normalize() method doesn't make any changes to the Path object referred by reference variable 'path'.

System.out.println(path); prints the original path, 'F:\A\.\B\C\D\..\Book.java' on to the console.

If you replace 'path.normalize();' with 'path = path.normalize();', then 'F:\A\B\C\Book.java' would be printed on to the console.

Correction Java IO File (NIO.2) - 31

B - System.out.println(Files.readAttributes(path, BasicFileAttributes.class).creationTime());

C - System.out.println(Files.getAttribute(path, “creationTime”));

D - System.out.println(Files.readAttributes(path, “*”).get(“creationTime”));

Files.getAttribute(Path path, String attribute, LinkOption… options) returns the value corresponding to passed attribute. IllegalArgumentException is thrown if attribute is not spelled correctly.

Files.getAttribute(path, “creationTime”) returns an object containing value for ‘creationTime’ attribute.

Files.readAttributes is overloaded method:

public static Map<String,Object> readAttributes(Path path, String attributes, LinkOption… options) throws IOException {…}

public static A readAttributes(Path path, Class type, LinkOption… options) throws IOException {…}

If 2nd parameter is of String type, readAttributes method returns Map<String, Object> and if 2nd parameter is of Class type, it returns A. And A should pass IS-A test for BasicFileAttributes type.

To retrieve value from Map object, use get(key) method.

Files.readAttributes(path, “*”).get(“creationTime”) returns an object containing value for ‘creationTime’ attribute.

Files.readAttributes(path, “*”).creationTime() causes compilation error as creationTime() method is not defined in Map interface.

Files.readAttributes(path, BasicFileAttributes.class) returns an instance of BasicFileAttributes class and it has creationTime() method to return the creation time.

But BaseFileAttributes class doesn’t have get(String) method, so ‘Files.readAttributes(path, BasicFileAttributes.class).get(“creationTime”)’ causes compilation error.

NOTE: There are other important methods in BaseFileAttributes class which you should know for the OCP exam: size(), isDirectory(), isRegularFile(), isSymbolicLink(), creationTime(), lastAccessedTime() and lastModifiedTime().

Correction Java IO File (NIO.2) - 32

A - An exception is thrown at runtime

Files.move(Path source, Path target, CopyOption... options) method throws following exceptions-

[Copied from the Javadoc]

1. UnsupportedOperationException - if the array contains a copy option that is not supported
2. FileAlreadyExistsException - if the target file exists but cannot be replaced because the REPLACE_EXISTING option is not specified (optional specific exception)
3. DirectoryNotEmptyException - the REPLACE_EXISTING option is specified but the file cannot be replaced because it is a non-empty directory (optional specific exception)
4. AtomicMoveNotSupportedException - if the options array contains the ATOMIC_MOVE option but the file cannot be moved as an atomic file system operation.
5. IOException - if an I/O error occurs
6. SecurityException - In the case of the default provider, and a security manager is installed, the checkWrite method is invoked to check write access to both the source and target file.

As target directory is not empty and StandardCopyOption.REPLACE_EXISTING is used hence DirectoryNotEmptyException is thrown at runtime.

Correction Java IO File (NIO.2) - 33

A - true

As file2 refers to an absolute path and not relative path, hence both ‘file1.resolve(file2)’ and ‘file1.resolveSibling(file2)’ returns Path object referring to ‘F:\A\B\C\Book.java’.

equals method returns true in this case.

Correction Java IO File (NIO.2) - 34

C - F:\A\.\B\C\D\..\Book.java

toAbsolutePath() method doesn’t care if given path elements are physically available or not and hence it doesn’t declare to throw IOException.

It just returns the absolute path without any normalization.

‘F:\A\.\B\C\D\..\Book.java’ is displayed on to the console.

Correction Java IO File (NIO.2) - 35

C -

```
Files.createDirectories(path.getParent());
Files.createFile(path);
```

Files.createFile(path); => throws IOException as parent directories don't exist.

Files.createDirectories(path); => Creates the directories 'A', 'B' and 'File.txt'. Path after creation is: 'F:\A\B\File.txt\'.

Files.createFile(path); => FileAlreadyExistsException as directory with the same name already exists.

path.getParent() returns 'F:\A\B', which is an absolute path and path.getFileName() returns 'File.txt', which is a relative path.

Files.createDirectories(path.getParent()); => Creates the directories 'A' and 'B'. Path after creation is: 'F:\A\B\'.

Files.createFile(path.getFileName()); => Creates the file under current directory, Path after creation is: 'C:\classes\com\training\ocp\File.txt'.

Files.createDirectories(path.getParent()); => Creates the directories 'A' and 'B'. Path after creation is: 'F:\A\B\'.

Files.createFile(path); => Creates the file, 'File.txt' under 'F:\A\B\'. Path after creation is: 'F:\A\B\File.txt'.