Chapter 19
Java File I/O (NIO.2)

THE OCP EXAM TOPICS COVERED IN THIS PRACTICE TEST INCLUDE THE FOLLOWING:

- ✓**Java File I/O (NIO.2)**
- Use Path interface to operate on file and directory paths
- Use Files class to check, read, delete, copy, move, manage metadata of a file or directory
- Use Stream API with NIO.2

1. Fill in the blanks: A(n)_____ is a file that contains a reference to another file or directory, while a(n)_____ is a file that contains content.

1. irregular file, regular file
2. regular file, opaque file
3. symbolic link, regular file
4. symbolic link, symbolic directory

2. Which methods listed below are found in the NIO.2 `Path` interface?

1. `getRoot()`
2. `isDirectory()`
3. `listFiles()`
4. `toRealPath()`

1. I only
2. I, II, and III
3. I and IV
4. II and III

3. Assuming the file `/secret/hide.txt` exists and is marked hidden, what is result of executing the following program?

```
package hidden;
import java.nio.file.*;
public class Finder {
   public void findHiddenFile(Path p) throws Exception {
      if(File.isHidden(p)) {
         System.out.print("Found!");
      }
   }
   public static void main(String[] folders) throws Exception {
      final Finder f = new Finder();
      f.findHiddenFile(Paths.get("/secret/hide.txt"));
   }
}
```

3. `Found!` is printed at runtime.
4. Nothing is printed at runtime.

4. Fill in the blanks: `Files.walk()` performs a _____ traversal, while `Files.find()` performs a _____ traversal.

1. breadth-first, breadth-first
2. breadth-first, depth-first
3. depth-first, breadth-first
4. depth-first, depth-first

5. When reading file information, what is an advantage of using an NIO.2 attribute interface rather than reading the values individually from `Files` methods?

1. Costs fewer round-trips to the file system
2. Guarantees performance improvement
3. Has support for symbolic links
4. Reduces memory leaks

6. What is the result of compiling and executing the following program? Assume the current directory is `/stock` and the path `/stock/sneakers` does not exist prior to execution.

```
package shoe;
import java.io.*;
import java.nio.file.*;
public class Sneaker {
   public void setupInventory(Path desiredPath) throws Exception {
      Path suggestedPath = Paths.get("sneakers");
      if(Files.isSameFile(suggestedPath, desiredPath)  // j1
            && !Files.exists(suggestedPath))
         Files.createDirectories(desiredPath);  // j2
   }
   public static void main(String[] socks) throws Exception {
      Path w = new File("/stock/sneakers").toPath();  // j3
      new Sneaker().setupInventory(w);
   }
}
```

1. The directory `/stock/sneakers` is created.
2. Line `j1` does not compile or produces an exception at runtime.
3. Line `j2` does not compile or produces an exception at runtime.
4. Line `j3` does not compile or produces an exception at runtime.

7. Assuming the path referenced below exists and contains a symbolic link that references `/again`, what is the expected result of executing the following code snippet?

```
System.out.print(Files.walk(Paths.get("/again/and/again")).count())
```

1. An exception is thrown at runtime.
2. A number is printed at runtime.
3. The process hangs indefinitely.
4. The result cannot be determined with the information given.

8. Which method in the NIO.2 `Files` class is equivalent to the `java.io.File` method `length()`?

1. `length()`
2. `size()`
3. `getLength()`
4. None of the above

9. Assuming the current working directory is `/home`, then what is the output of the following program?

```
1:  package magic;
2:  import java.nio.file.*;
3:  public class Magician {
4:     public String doTrick(Path path) {
5:        return path.subpath(2,3)
6:           .getName(1)
7:           .toAbsolutePath()
8:           .toString();
9:     }
10:    public static void main(String... cards) {
11:       final Magician m = new Magician();
12:       System.out.print(m.doTrick(
13:          Paths.get("/bag/of/tricks/.././disappear.txt")));
14:    } }
```

1. `/home/tricks`
2. `/home`
3. The code does not compile.
4. The code compiles but prints an exception at runtime.

10. Which methods listed below are found in the NIO.2 `Files` class?

I. `isSameFile()`
II. `length()`
III. `relativize()`
IV. `mkdir()`

1. I only
2. I, II, and IV
3. II and III
4. IV only

11. The following code snippet, which attempts to move a file system record from `oldHardDrivePath` to `newHardDrivePath`, results in an exception at runtime. Which of the following is the most likely type of exception to be thrown?

```
Files.move(oldHardDrivePath,newHardDrivePath,REPLACE_EXISTING);
```

1. `AtomicMoveNotSupportedException`
2. `DirectoryNotEmptyException`
3. `FileAlreadyExistsException`
4. None of the above since the line of code does not compile

12. Which of the following can be filled into the blank that would allow the method to compile?

```
public String getPathName(String fileName) {
   final Path p = ____________________;
   return p.getFileName();
}
```

I. `new File(fileName).toPath()`
II. `new Path(fileName)`
III. `FileSystems.getDefault().getPath(fileName)`

1. I and II
2. I and III
3. II
4. None of the above

13. Which statement about the following class is correct?

```
package clone;
import java.io.*;
import java.nio.file.*;
public class Rewriter {
   public static void copy(Path source, Path target) throws Exceptio
      try (BufferedReader r = Files.newBufferedReader(source);
            Writer w = Files.newBufferedWriter(target)) {
         String temp = null;
         while((temp = r.readLine()) != null) {
            w.write(temp);
         }
```

```
      }
   }
   public static void main(String[] tooMany) throws Throwable {
      Rewriter.copy(Paths.get("/original.txt"),
         FileSystems.getDefault().getPath("/","unoriginal.txt"));
   }
}
```

---

1. The class compiles without issue.
2. The class never throws an exception at runtime.
3. The implementation correctly copies a regular file.
4. All of the above

14. Fill in the blanks: The `Files.____________` method returns a `List`, while the `Files.____________` method returns a `Stream`.

1. `lines()`, `readAllLines()`
2. `lines()`, `readLines()`
3. `readAllLines()`, `lines()`
4. `readLines()`, `lines()`

15. What is the output of the following application?

```
1:  package yellow;
2:  import java.nio.file.*;
3:  public class Road {
4:     public boolean findHome() {
5:        Path oftenTraveled = Paths.get("/highway/street/spot.txt"
6:        Path lessTraveled = Paths.get("/highway/street/house/../.
7:        lessTraveled.resolve("spot.txt");
8:        return oftenTraveled.equals(lessTraveled.normalize());
9:     }
10:    public static void main(String... emerald) {
11:       System.out.print("AM I HOME? "
12:             +(new Road().findHome() ? "yes" : " no"));
13:    }
14: }
```

---

1. `AM I HOME? no`
2. `AM I HOME? yes`
3. The class does not compile.
4. The class compiles but throws an exception at runtime.

16. Which of the following is not an advantage of using an NIO.2 `Path` instead of a `java.io.File` to work with files?

1. Contains built-in support for symbolic links
2. Has ability to read operating-system-specific attributes
3. Provides a single method for deleting a directory tree
4. Provides efficient access of file metadata

17. What is the result of executing the following program? Assume the path `/driveway` exists and is non-empty, and the directory tree is fully accessible within the file system.

```
package weather;
import java.io.*;
import java.nio.file.*;
public class Snow {
   public static boolean removeSnow(Path flake) throws IOException
      if(!Files.isDirectory(flake) && !Files.isSymbolicLink(flake))
         return Files.delete(flake);
      else return true;
   }
   public static void main(String[] cones) throws IOException {
      File driveway = new File("/driveway");
      for(File f : driveway.listFiles()) {
         System.out.println(removeSnow(f.toPath()));
      }
   }
}
```

---

1. The program prints a list of only `true` values.

2. The program prints a mix of `true` and `false` values.
3. The code does not compile.
4. The code compiles but prints an exception at runtime.

18. Which interface name inserted into the blank below allows the code snippet to compile?

```
Path file = Paths.get("/data/movie.txt");
BasicFileAttributes b = Files.readAttributes(file, _________);
```

1. `BasicFileAttributes.class`
2. `DosFileAttributes.class`
3. `PosixFileAttributes.class`
4. All of the above

19. What is the output of the following code snippet? Assume that the current directory is the root path.

```
Path p1 = Paths.get("./locks");
Path p2 = Paths.get("/found/red.zip");
System.out.println(p1.relativize(p2));
System.out.println(p2.relativize(p1));
```

1. `../found/red.zip`

   `../../locks`
2. `../../locks`

   `../found/red.zip`
3. `locks/../found/red.zip`

   `../found/locks`
4. None of the above

20. What is the output of the following code snippet? Assume that the current directory is the root path.

```
Path p1 = Paths.get("./found/../keys");
Path p2 = Paths.get("/lost/blue.txt");
System.out.println(p1.resolve(p2));
System.out.println(p2.resolve(p1));
```

1. `/lost/blue.txt`

   `./found/../keys`
2. `/found/../keys/./lost/blue.txt`

   `/lost/blue.txt/keys`
3. `/lost/blue.txt`

   `/lost/blue.txt/./found/../keys`
4. None of the above

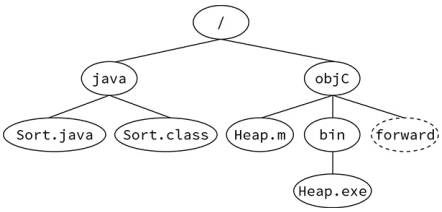
21. What is the output of the following application? Assume the application is called with a valid path that exists and is accessible within the file system.

```
package charity;
import java.nio.file.*;
public class Roster {
   protected void printRoster(Path p) {
      for(Path f : Files.list(p)) {  // n1
         if(f.toString().endsWith(".per"))  // n2
            System.out.print(f);
      }
   }
   public static void main(String... volunteers) {
      new Roster().printRoster(Paths.get(volunteers[0]));
   }
}
```

1. A list of file names is printed at runtime.

2. The class does not compile due to line `n1`.
3. The class does not compile due to line `n2`.
4. None of the above

22. Given the following file system diagram, in which `forward` is a symbolic link to the `java` directory, which value does not print `/java/Sort.java` at runtime?




```
Path p = Paths.get("/", "objC", "bin");
System.out.println(p.resolve("_________").toRealPath());
```

1. `../backwards/../forward/Sort.java`
2. `../forward/./Sort.java`
3. `../java/./forward/Sort.java`
4. `../../java/Sort.java`

23. Using the file system diagram from the previous question, including the symbolic link from `forward` to `java`, how many calls to `Files.delete()` would need to be made before the following line could be executed without throwing an exception?

```
Files.delete(Paths.get("/objC"));
```

1. One
2. Four
3. Seven
4. None of the above. The symbolic link needs to be removed with `Files.deleteSymbolicLink()` first.

24. Assuming the `course.txt` file exists and is readable, what is the result of executing the following application?

```
package schoolwork;
import java.io.*;
import java.nio.file.*;
public class Notes {
   public void printNotes() {
      try (OutputStream out = System.out) {  // y1
         Files.copy(out, Paths.get("course.txt"));
      } catch (IOException e) {
         throw new RuntimeException(e);
      }
   }
   public static void main(String[] coursework) {
      new Notes().printNotes();
   }
}
```

1. The code compiles but prints an exception at runtime.
2. The class does not compile due to line `y1`.
3. The code does not compile for some other reason.
4. The program prints the contents of the `course.txt` file.

25. When reading file information, what is an advantage of loading a `BasicFileAttributeView` over a `BasicFileAttributes`?

1. Allows the hidden attribute to be set
2. Allows the last modified date to be changed
3. All of the file information is read in a single round-trip.
4. There is no advantage.

26. The `Rose` application is run with an input argument of `/flower`. The `/flower` directory contains five subdirectories, each of which contains five files. How many `Path` values does the following application print?

```
import java.nio.file.*;
public class Rose {
   public void tendGarden(Path p) throws Exception {
      Files.walk(p,1)
         .map(p -> p.toRealPath())
         .forEach(System.out::println);
   }
   public static void main(String... thorns) throws Exception {
      new Rose().tendGarden(Paths.get(thorns[0]));
   }
}
```

1. None
2. One
3. Six
4. Thirty-one

27. Which of the following statements, when run independently, produces a `NullPointerException` at runtime?

I. `Paths.get("../sang").getParent().getParent()`

II. `Paths.get("/sing").getParent().getRoot()`

III. `Paths.get("/song").getRoot().getRoot()`

IV. `Paths.get("../sung").getRoot().getParent()`

1. I and III
2. I and IV
3. II and III
4. IV only

28. Which statement about the following `Finalize` class is correct?

```
1: package end;
2: import java.nio.file.*;
3: public class Finalize {
4:    public Path makeAbsolute(Path p) {
5:       if(p!=null && !p.isAbsolute())
6:       return p.toAbsolutePath();
7:       return p;
8:    }
9: }
```

1. It does not compile because `IOException` is neither handled nor declared.
2. It does not compile for some other reason.
3. The method compiles and returns a `Path` value that is always equivalent to the input argument.
4. The method compiles and returns a `Path` value that may not be equivalent to the input argument.

29. Which of the following is a difference between the `createDirectory()` and `createDirectories()` methods found in the NIO.2 `Files` class?

1. One takes multiple `Path` arguments; the other does not.
2. One throws an exception if a file already exists at the directory path; the other does not.
3. One declares a checked exception; the other does not.
4. One creates a single directory while the other may create many directories.

30. Assuming the current working directory is `/hail`, what is the expected output of executing the following code snippet?

```
Path w1 = Paths.get("../jungle/.././rain..")
      .toAbsolutePath().normalize();
System.out.print(w1.resolve("snow.txt"));
```

1. `/jungle/snow.txt`

2. `/hail/rain../snow.txt`
3. `/rain../snow.txt`
4. An exception is printed at runtime.

31. What is the output of the following application?

```
package med;
import java.nio.file.*;
public class Surgeon {
   public Path rebuild(Path p) {
      Path v = null;
      for(int i=0; i<p.getNameCount(); i++)
         if(v==null) v = p.getName(i);
         else v = v.resolve(p.getName(i));
      return v;
   }
   public static void main(String... tools) {
      final Surgeon al = new Surgeon();
      Path original = Paths.get("/tissue/heart/chambers.txt");
      Path repaired = al.rebuild(original);
      System.out.print(original.equals(repaired));
   }
}
```

1. `false`
2. `true`
3. The code does not compile.
4. The code compiles but prints an exception at runtime.

32. Under which circumstances does `Files.deleteIfExists()` not throw an exception?

1. The file system suddenly becomes unavailable.
2. The path does not exist.
3. The path represents a non-empty directory.
4. The process does not have write access to a path.

33. What is the output of the following code snippet? Assume all referenced paths exist within the file system.

```
Path v1 = Path.get("/./desert/./").resolve(Paths.get("sand.doc"));
Path v2 = new File("/desert/./cactus/../sand.doc").toPath();
System.out.print(Files.isSameFile(v1,v2));
System.out.print(" "+v1.equals(v2));
System.out.print(" "+v1.normalize().equals(v2.normalize()));
```

1. `false false false`
2. `true false true`
3. `true true true`
4. None of the above

34. How many lines of the following program contain compilation errors?

```
public class Song {
   public static void organize(Path folder, Path file) throws IOExc
      Path p = folder.resolve(file);
      BasicFileAttributeView vw = Files.getFileAttributeView(p,
         BasicFileAttributes.class);
      if(vw.creationTime().toMillis()<System.currentTimeMillis()) {
         vw.setTimes(FileTime.fromMillis(System.currentTimeMillis()
            null,null);
      }
   }
   public static void main(String[] audio) throws Exception {
      Song.organize(Paths.get("/", "pub"),new File("/songs").toPath
   }
}
```

1. None
2. One
3. Two
4. Three

35. What is the output of the following application?

```
package stars;
import java.nio.file.*;
public class Sun {
   public void printInfo() {
      Path halleysComet = Paths.get("stars/./rocks/../m1.meteor")
      Path lexellsComet = Paths.get("./stars/../solar/");
      lexellsComet = lexellsComet.subpath(0, 2)
            .resolve("m1.meteor")
            .normalize();
      System.out.print(halleysComet.equals(lexellsComet)
            ? "Same!" : "Different!");
   }
   public static void main(String... emerald) {
      Sun s = new Sun();
      s.printInfo();
   }
}
```

1. `Different!`
2. `Same!`
3. The class does not compile.
4. The class compiles but throws an exception at runtime.

36. Assuming the directory `/eclipse/projects` exists and its contents are accessible, which statement about the following code snippet is correct?

```
Path p = Paths.get("/eclipse/projects");

Files.walk(p)
   .map(z -> z.toAbsolutePath().toString())
   .filter(s -> s.endsWith(".java"))
   .collect(Collectors.toList()).forEach(System.out::println);

Files.find(p,Integer.MAX_VALUE,
      (w,a) -> w.toAbsolutePath().toString().endsWith(".java"))
   .collect(Collectors.toList()).forEach(System.out::println);
```

1. The first stream statement does not compile.
2. The second stream statement does not compile.
3. Both statements compile but are unlikely to print the same results at runtime.
4. None of the above

37. Assuming the file referenced below exists and is significantly large, which statement about the following program is correct?

```
public class SpeedRead {
   public void jenniferReads(Path p) {
      Files.lines(p);
   }
   public void jonReads(Path p) {
      Files.readAllLines(p);
   }
   public static void main(String[] pages) {
      Path p = Paths.get("/bookshelf/mobydick.txt");
      final SpeedRead r = new SpeedRead();
      r.jenniferReads(p);
      r.jonReads(p);
   }
}
```

1. The code does not compile.
2. The method `jenniferReads()` is likely to take longer to run.
3. The method `jonReads()` is likely to take longer to run.
4. It is not possible to know which method will take longer to run.

38. What is the result of executing the following program? Assume the files referenced in the application both exist and are fully accessible within the file system.

```
package duplicate;
import static java.nio.file.StandardCopyOption.*;
import static java.nio.file.Files.*;
import java.io.*;
import java.nio.file.*;
public class CopyOfACopy {
   public void main(String[] items) throws Exception {
      final Path s = new File("apples.zip").toPath();
```

```
      final Path t = FileSystems.getDefault().getPath("oranges.zip"
      copy(s,t,REPLACE_EXISTING);  // q1
      copy(Files.newBufferedReader(t),t,ATOMIC_MOVE);  // q2
   }
}
```

1. Line `q1` does not compile.
2. Line `q1` produces an exception at runtime.
3. Line `q2` does not compile.
4. Line `q2` produces an exception at runtime.

39. Which of the following `Files` methods requires the enclosing method to handle or declare a checked exception?

1. `exists()`
2. `isDirectory()`
3. `isSameFile()`
4. `isSymbolicLink()`

40. What is the output of the following application? Assume `/all-data` exists and is accessible within the file system.

```
package numbers;
import java.nio.file.*;
import java.util.stream.Stream;
public class TheCount {
   public static Stream<String> readLines(Path p) {
      try {
         return Files.lines(p);
      } catch (Exception e) {
         throw new RuntimeException(e);
      }
   }
   public static long count(Path p) throws Exception {
      return Files.list(p)
         .filter(w -> Files.isRegularFile(w))
         .flatMap(s -> readLines(s))
         .count();
   }
   public final static void main(String[] day) throws Exception {
      System.out.print(count(Paths.get("/all-data")));
   }
}
```

1. The number of lines in all files in a directory tree
2. The number of lines in all files in a single directory
3. The code does not compile.
4. The code compiles but prints an exception at runtime.

PREV
Chapter 18 Java I/O Fundamentals

NEXT
Chapter 20 Java Concurrency