21.7 Creating Directory Entries

The `Files` class provides methods for creating files and directories. These methods can accept a variable arity parameter of type `FileAttribute<?>`. The interface `java.nio.file.attribute.FileAttribute<T>` defines an object that encapsulates the value of a file attribute that can be set when a file or a directory is created by these methods. For a POSIX-based file system, the `PosixFilePermissions.asFileAttribute()` method creates a `FileAttribute` that encapsulates a set of type `PosixFilePermission` (p. 1325).

The methods for creating regular files and directories throw a `FileAlreadyExistsException` if the directory entry with that name already exists. All methods for creating directory entries can throw an `IOException`, and should be called in a `try` block or the exception should be specified in a `throws` clause.

The steps to check whether the directory entry exists and to create the new directory entry if it does not exist are performed as a single *atomic operation*.

The following code calls the `printDirEntryInfo()` method in the utility class `FileUtils` on a path to print what kind of directory entry it denotes and its file permissions.

```
public static void printDirEntryInfo(Path path) throws IOException {
  String fmt = Files.isSymbolicLink(path)? "Symbolic link: %s%n":
      Files.isRegularFile(path)? "File: %s%n":
        Files.isDirectory(path)? "Directory: %s%n":
          "Directory entry: %s%n";
  out.printf(fmt, path);
  Set<PosixFilePermission> perms = Files.getPosixFilePermissions(path);
  String permStr = PosixFilePermissions.toString(perms);
  out.println(permStr);
}
```

Creating Regular and Temporary Files

The following methods of the `Files` class can be used to create regular and temporary files, and symbolic links.

```
static Path createFile(Path path, FileAttribute<?>... attrs)
                       throws IOException
```

Creates an empty file that is denoted by the `path` parameter, but fails by throwing a `FileAlreadyExistsException` if the file already exists. It does *not* create nonexistent parent directories, and throws a `NoSuchFileException` if any parent directory does not exist.

```
static Path createTempFile(String prefix, String suffix,
                           FileAttribute<?>... attrs) throws IOException
static Path createTempFile(Path dir, String prefix, String suffix,
                           FileAttribute<?>... attrs) throws IOException
```

Create an empty file in the *default temporary-file directory* or in the *specified directory* denoted by the `Path` object, respectively, using the specified `prefix` and `suffix` to generate its name. The default temporary-file directory and naming of temporary files is file-system-specific.

```
static Path createSymbolicLink(Path symbLink, Path target,
                               FileAttribute<?>... attrs) throws IOException
```

Creates a symbolic link to a target (*optional operation*). When the target is a *relative path* then file system operations on the target are relative to the path of the symbolic link. The target of the symbolic link need not exist. Throws a `FileAlreadyExistsException` if a directory entry with the same name as the symbolic link already exists.

```
static Path readSymbolicLink(Path symbLink) throws IOException
```

Returns a `Path` object that denoted the target of a symbolic link (*optional operation*). The target of the symbolic link need not exist.

For creating files, we will use the following `FileAttribute` object denoted by the `fileAttr` reference (p. 1325). It specifies read and write permissions for the owner, but only read access for the group and others.

```
Set<PosixFilePermission> filePerms = PosixFilePermissions.fromString("rw-r--r--");
FileAttribute<Set<PosixFilePermission>> fileAttr =
        PosixFilePermissions.asFileAttribute(perms);
```

Creating Regular Files

The `createFile()` method of the `Files` class fails to create a regular file if the file already exists, or the parent directories on the path do not exist. The code below at (1) creates a file with the path `project/docs/readme.txt` relative to the current directory under the assumption that the file name does not exist and the parent directories exist on the path. However, running this code repeatedly will result in a `FileAlreadyExistsException`, unless the file is deleted (e.g., calling `Files.deleteIfExists(regularFile)`) before rerunning the code.

```
try {
  Path regularFile  = Path.of("project", "docs", "readme.txt");
  Path createdFile1 = Files.createFile(regularFile, fileAttr);        // (1)
  FileUtils.printDirEntryInfo(createdFile1);
} catch (NoSuchFileException | FileAlreadyExistsException fe) {
  fe.printStackTrace();
} catch (IOException ioe) {
  ioe.printStackTrace();
}
```

Possible output from the code:

```
File: project/docs/readme.txt
rw-r--r--
```

Creating Temporary Files

Programs usually create temporary files during execution, and delete such files when done in the interest of good housekeeping. The JVM defines a system property for the default temporary-file directory where temporary files are created, if a specific location is not specified. This location is printed by the code below at (1).

How the name of the temporary file is generated is file-system specific. The code at (2) creates a temporary file under the default temporary-file directory, that has default file permissions.

The code at (3) creates temporary files under a specific directory. How the specification of file name prefix and suffix affect the generated file name can be seen in the output, where the `null` value as the prefix omits the prefix, and the `null` value as the suffix appends the default file name extension `".tmp"`.

The NIO.2 API does not define a method to request that a file be deleted when the JVM terminates.

```
try {
  // System property that defines the default temporary-file directory.      (1)
  String tmpdir = System.getProperty("java.io.tmpdir");
  System.out.println("Default temporary directory: " + tmpdir);

  // Create under the default temporary-file directory.                      (2)
  Path tmpFile1 = Files.createTempFile("events", ".log");
  FileUtils.printDirEntryInfo(tmpFile1);

  // Create under a specific directory:                                      (3)
  Path tmpFileDir = Path.of("project");
  Path tmpFile2 = Files.createTempFile(tmpFileDir, "proj_", ".dat", fileAttr);
  Path tmpFile3 = Files.createTempFile(tmpFileDir, "proj_", null,   fileAttr);
  Path tmpFile4 = Files.createTempFile(tmpFileDir, null,    ".dat", fileAttr);
  Path tmpFile5 = Files.createTempFile(tmpFileDir, null,    null,   fileAttr);
  FileUtils.printDirEntryInfo(tmpFile2);
  FileUtils.printDirEntryInfo(tmpFile3);
  FileUtils.printDirEntryInfo(tmpFile4);
  FileUtils.printDirEntryInfo(tmpFile5);
} catch (IOException ioe) {
  ioe.printStackTrace();
}
```

Possible output from the code (*edited to fit on the page*):

```
Default temporary directory: /var/folders/cr/wk7fqcjx07z95d9vxcgjnrtc0000gr/T/
File:
/var/folders/cr/wk7fqcjx07z95d9vxcgjnrtc0000gr/T/events4720093907665196131.log
rw-------
File: project/proj_6062790522710209175.dat
rw-r--r--
File: project/proj_957015125593453845.tmp
rw-r--r--
File: project/3032983609251590109.dat
rw-r--r--
File: project/8205471872222375044.tmp
rw-r--r--
```

Creating Symbolic Links

The code below illustrates creating symbolic links to directory entries. We assume that the symbolic link at (1) below does not exist; otherwise, a resounding `FileAlreadyExistsException` will be thrown by the `createSymbolicLink()` method of the `Files` class. However, the target path need *not* exists, but that might limit the file operations that can be performed using the symbolic link. We assume that the target path at (2) exists when the code below is run.

The `createSymbolicLink()` method can be called with a relative path or the absolute path of the target, as shown at (3a) and (3b), respectively. In both cases, the symbolic link will be created with the default file permissions, as we have not specified the optional `FileAttribute` variable arity parameter. Note that the symbolic link is created to a file or a directory, depending on the kind of the target.

The method `readSymbolicLink()` method of the `Files` class returns the path of the target denoted by the symbolic link.

```
try {
  Path symbLinkPath  = Path.of(".", "readme_link");                    // (1)
  Path targetPath    = Path.of(".", "project", "backup", "readme.txt"); // (2)

  Path symbLink = Files.createSymbolicLink(symbLinkPath, targetPath);   // (3a)
//Path symbLink = Files.createSymbolicLink(symbLinkPath,
//                                         targetPath.toAbsolutePath());// (3b)
  Path target = Files.readSymbolicLink(symbLink);                       // (4)

  FileUtils.printDirEntryInfo(symbLink);
  FileUtils.printDirEntryInfo(target);
} catch (FileAlreadyExistsException fe) {
  fe.printStackTrace();
} catch (IOException ioe) {
  ioe.printStackTrace();
}
```

Possible output from the code:

```
Symbolic link: ./readme_link
rw-r--r--
File: ./project/backup/readme.txt
rw-r--r--
```

Creating Regular and Temporary Directories

The `Files` class provides methods to create regular and temporary directories.

```
static Path createDirectory(Path dir, FileAttribute<?>... attrs)
                            throws IOException
```

Creates a new directory denoted by the `Path` object. It does *not* create nonexistent parent directories, and throws a `NoSuchFileException` if any parent directory does not exist. It also throws a `FileAlreadyExistsException` if the directory entry with that name already exists.

```
static Path createDirectories(Path dir, FileAttribute<?>... attrs)
           throws IOException
```

Creates a directory by creating all nonexistent parent directories first. It does not throw an exception if any of the directories already exist. If the method fails, some directories may have been created.

```
static Path createTempDirectory(String prefix, FileAttribute<?>... attrs)
                                throws IOException
static Path createTempDirectory(Path dir, String prefix,
                                FileAttribute<?>... attrs)
                                throws IOException
```

Create a new directory in the *default temporary-file directory* or in the *specified directory* denoted by the `Path` object, using the mandatory non-`null` `prefix` to append to its generated name. A `NoSuchFileException` is thrown by the second method if the specified location does not exist.

For creating directories, we will use the following `FileAttribute` object denoted by the `dirFileAttr` reference (p. 1325). It specifies read, write, and execute permissions for all users: the owner, the group, and others—often called *full permissions*.

```
Set<PosixFilePermission> dPerms = PosixFilePermissions.fromString("rwxrwxrwx");
FileAttribute<Set<PosixFilePermission>> dirFileAttr =
        PosixFilePermissions.asFileAttribute(dPerms);
```

Creating Regular Directories

The `Files` class provides two methods to create regular directories. The `createDirectory()` method fails to create a regular directory if the directory already exists, or if the parent directories on the path do not exist. The code below creates a directory with the path `project/bin` relative to the current directory, under the assumption that the directory name `bin` does not exist and the parent directory `./project` exists on the path. The directory is first created with the default file permissions at (1), then deleted at (2) and created again at (3) with specific file permissions. The second call to the `createDirectory()` method at (3) would throw a `FileAlreadyExistsException` if we did not delete the directory before creating it with specific file permissions.

The astute reader will have noticed from the output that the *write* permission for the group and others is not set at creation time by the `createDirectory()` method, regardless of the file permissions specified. This can be remedied by setting the file permissions explicitly after the directory has been created, as at (4).

```
try {
  Path regularDir  = Path.of("project", "bin");
  Path createdDir = Files.createDirectory(regularDir);                    // (1)
  FileUtils.printDirEntryInfo(createdDir);

  if (Files.deleteIfExists(regularDir)) {                                 // (2)
    System.out.printf("Directory deleted: %s%n", regularDir);
  }
  Path newDir = Files.createDirectory(regularDir, dirFileAttr);           // (3)
  FileUtils.printDirEntryInfo(newDir);
```

```
    Files.setPosixFilePermissions(newDir, dPerms);                          // (4)
    FileUtils.printDirEntryInfo(newDir);
  } catch (NoSuchFileException | FileAlreadyExistsException fe) {
    fe.printStackTrace();
  } catch (IOException ioe) {
    ioe.printStackTrace();
  }
```

Possible output from the code:

```
Directory: project/bin
rwxr-xr-
Directory deleted: project/bin
Directory: project/bin
rwxr-xr-x
Directory: project/bin
rwxrwxrwx
```

The method `createDirectories()` first creates all nonexistent parent directories in the path if necessary, and only creates the directory if it does not exist—a convenient way to ensure that a directory always exists. In the code below at (5), the parent directories (in the path `project/branches/maintenance`) are created from top to bottom, relative to the current directory if necessary, and the directory `versions` is only created if it does not exist from before. A `FileAlreadyExistsException` at (6) is thrown if a directory entry named `versions` exists, but is not a directory.

```
try {
  Path regularDir2  = Path.of("project", "branches", "maintenance", "versions");
  Path createdDir2 = Files.createDirectories(regularDir2, dirFileAttr);  // (5)
  FileUtils.printDirEntryInfo(createdDir2);
} catch (FileAlreadyExistsException fe) {                                 // (6)
  fe.printStackTrace();
} catch (IOException ioe) {
  ioe.printStackTrace();
}
```

Output from the code:

```
Directory: project/branches/maintenance/versions
rwxrwxrwx
```

Creating Temporary Directories

Analogous to temporary files, temporary directories can be created by the `createTempDirectory()` methods of the `Files` class. Again, these directories can be created in the default temporary-file directory, or in a specific location if one is specified in the method call, with either the default file permissions or specific permissions.

The code at (7) creates a temporary directory named `"log_dir"` under the default temporary-file directory. A temporary directory with the same name is also created under a specific location (the `./project` directory) at (8). A `NoSuchFileException` is thrown if the specified location does not exist. A prefix can be specified for the directory name or it can be `null`.

```
try  {
  // Create under the default temporary-file directory.                    (7)
  Path tmpDirPath1 = Files.createTempDirectory("log_dir", dirFileAttr);
  FileUtils.printDirEntryInfo(tmpDirPath1);

  // Create under a specific location:                                     (8)
  Path tmpDirLoc = Path.of("project");
  Path tmpDirPath2 = Files.createTempDirectory(tmpDirLoc, "log_dir", dirFileAttr);
  Path tmpDirPath3 = Files.createTempDirectory(tmpDirLoc, null, dirFileAttr);
  FileUtils.printDirEntryInfo(tmpDirPath2);
  FileUtils.printDirEntryInfo(tmpDirPath3);

  Files.setPosixFilePermissions(tmpDirPath3, dPerms);                   // (9)
  FileUtils.printDirEntryInfo(tmpDirPath3);
} catch (NoSuchFileException nsfe) {
  nsfe.printStackTrace();
} catch (IOException ioe) {
  ioe.printStackTrace();
}
```

Possible output from the code (*edited to fit the page*):

```
Directory:
/var/folders/cr/wk7fqcjx07z95d9vxcgjnrtc0000gr/T/log_dir18136008118052819366
rwxr-xr-x
Directory: project/log_dir14908011762674796217
rwxr-xr-x
Directory: project/18428782809319921018
rwxr-xr-x
Directory: project/18428782809319921018
rwxrwxrwx
```

We notice from the output that the *write* permission for the group and others is not set at creation time by the `createTempDirectory()` method, regardless of the file permissions specified. Again, this can be remedied by setting the file permissions explicitly after the temporary directory has been created, as at (9).