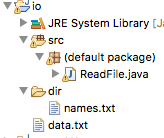
We now look at methods that can be used to deal with a path to a particular directory or file on your hard drive. Three classes/interfaces come into play; all are in package java.nio.file:

Interface Path: has instance methods that give information about or operate on a path.  
 Class Files: has static methods to operate on the file/directory given by a Path object.   
 Class Paths: has static methods to create a Path object corresponding to a String.

Note the difference between Path and Files: Path has methods for dealing with paths; Files has methods for dealing with the file/directory given by a path.

**Creating a Path object**

Consider the Eclipse project shown to the right. In classes in directory src, to create a Path pd for file data.txt and a Path pn for file names.txt, use:

Path pd= Paths.get("data.txt");  
 Path pn= Paths.get("dir", "names.txt");

Note that Path is an interface, and you do not know the actual class of pd! Its class will depend on the operating system on which your computer is running. You can print the value of pd.getClass() to find out. Try it, and compare what you get with what friends who own a different kind of computer get. This illustrates how useful interfaces are. We don’t *care* what the class of pd is. We just need to know what methods it has.

Method Paths.get can have any number of names as arguments. They usually form a relative path starting at some node of the tree of your hard drive directory. It is possible to give an absolute path, starting at the root of your hard drive directory, but don’t do that. If you do, your program won’t work on someone else’s computer because the file structure will be different. More on this later.

**First time reading this? No need to read further now**

You do not have to digest the following material on methods in interface Path and class Files if this is your first time looking at I/O and you just want to know how to read and write files. You can skip to item (3) in the JavaHyperText entry for I/O. But this information properly belongs here as a reference.

**Information about a Path**

Interface Path has several functions for extracting information about a Path p. You may never need them, but it’s nice to know about them. Below are some of them. We give examples of what they do assuming p contains four names, "s1", "s2", "s3", and "s4".

p.toString(): The path as a string. It will have system-specific name separators (usually / or \). Example: "s1/s2/s3/s4".

p.getFileName(): A Path object that contains only the last name on p. Example: a Path for "s4".

p.getParent(): The Path of the parent (null if no parent). Example: a Path containing "s1", "s2", and "s4".

p.getRoot(): The Path consisting of the root. This is **null** for a relative path because the root is missing.

p.getNameCount(): The number of names in p. Example: 4.

p.isAbsolute(): True if p is an absolute path, false if not.

p.toAbsolute(): The absolute Path object corresponding to relative path p. This will typically be done by resolving the path against the file system default directory. So, starting from the root of the default directory, a tree search can be made for the node begins path p.

Thus given a relative Path, one can construct a corresponding absolute path.

There are other methods. If you need to process a Path in some fashion, visit the API documentation for interface Path and look through its methods.

**Information about the file or directory described by a Path**

As you now know, class Path has methods for finding out about a Path object p. Class Files, on the other hand, contains static methods that give you information about the actual file or directory that is described by p. You can find out whether it is a file or a dictionary, whether it exists, and whether it is readable or writable. You can even create the file or directory if it doesn’t exist, you can copy the file to another place, you can delete it, and you can move it or rename it.

Finally, you can obtain a BufferedReader or BufferedWriter for it, allowing you to read or write the file. This is the subject of item (3) in the JavaHyperText entry for I/O.

Here are just a few of the static methods in class Files.

Files.exists(p): true if the file/directory described by p exists.

Files.isDirectory(p): true if p describes a directory and not a file.

Files.isReadable(p): true if p is readable.

Files.isWritable(p): true if p is writable.

Files.createDirectory(p): Create the directory described by Path p, but fail if it already exists. Read the spec carefully before using this method.

Files.createFile(p): Create the file described by Path p, but fail if it already exists. Read the spec carefully before using this method.

Files.move(p1, p2): move files described by Path p1 to the placed described by Path p2. Read the spec carefully before using this method.