SCC0504 - Programação Orientada a Objetos

Arquivos - Parte 02

Prof.: Leonardo Tórtoro Pereira

leonardop@usp.br

Arquivos

- Em Java, existem 2 pacotes principais para leitura e escrita de dados. A leitura pode ser dividida em Streams e File
  - I/O Streams (java.io)
  - File I/O (java.nio e em menor parte na java.io)

File I/O

File I/O

- ➔ É um pacote que contém suporte para entrada e saída de arquivos e acessar o sistema de arquivos padrão
- ➔ É possível acessar a árvore de arquivos do SO, buscar arquivos com expressões regulares, checar, deletar, copiar e mover arquivos, entre outros.

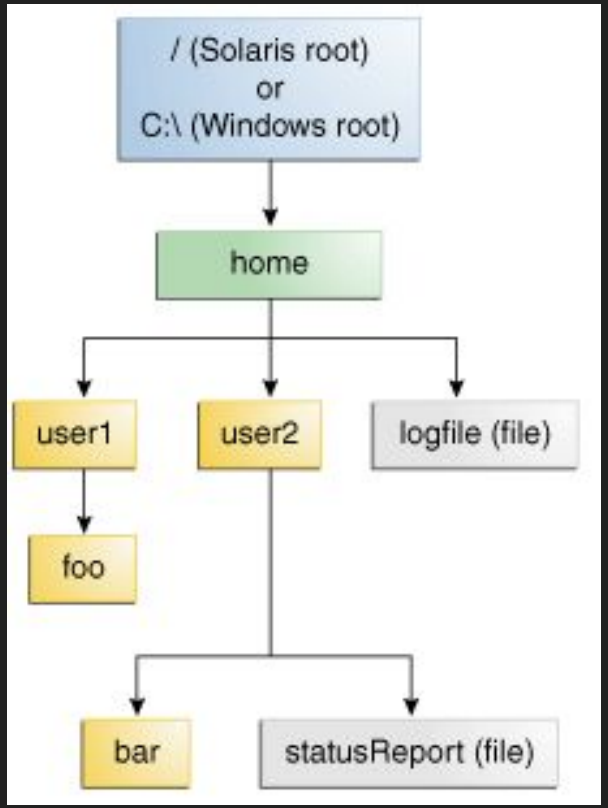
Caminho (*Path*)

Caminho (Path)

- Um sistema de arquivos armazena e organiza arquivos em alguma forma de mídia (geralmente um ou mais discos rígidos) de forma que eles sejam facilmente acessados
- A maioria dos sistemas de arquivos atuais usam armazenam os arquivos em uma estrutura de árvore
  - Também chamada de estrutura hierárquica

Caminho (Path)

- No topo da árvore ficam um (ou mais) nós raízes
- Embaixo de cada um ficar arquivos e diretórios (pastas)
- Cada diretório contém arquivos e subdiretórios
  - Estes podem conter arquivos e subdiretórios
  - E assim por diante
- Um arquivo é identificado pelo caminho dele através do sistema de arquivos, começando do nó raiz



Fonte: https://docs.oracle.com/javase/tutorial/essential/io/path.html

Caminho (Path)

- Caminho do arquivo *statusReport* no sistema Solaris:
  - /home/sally/statusReport
- No Windows:
  - C:\home\sally\statusReport
- Cada sistema usa um caractere para separar os nomes dos diretórios (também conhecido como delimitador)
  - Solaris - ‘/’
  - Windows ‘\’

Caminho (Path)

- Um caminho pode ser relativo ou absoluto
- Absoluto
  - Contém o nó raiz e a lista completa de diretórios para localizar o arquivo
  - Ex: */home/sally/statusReport*

Caminho (Path)

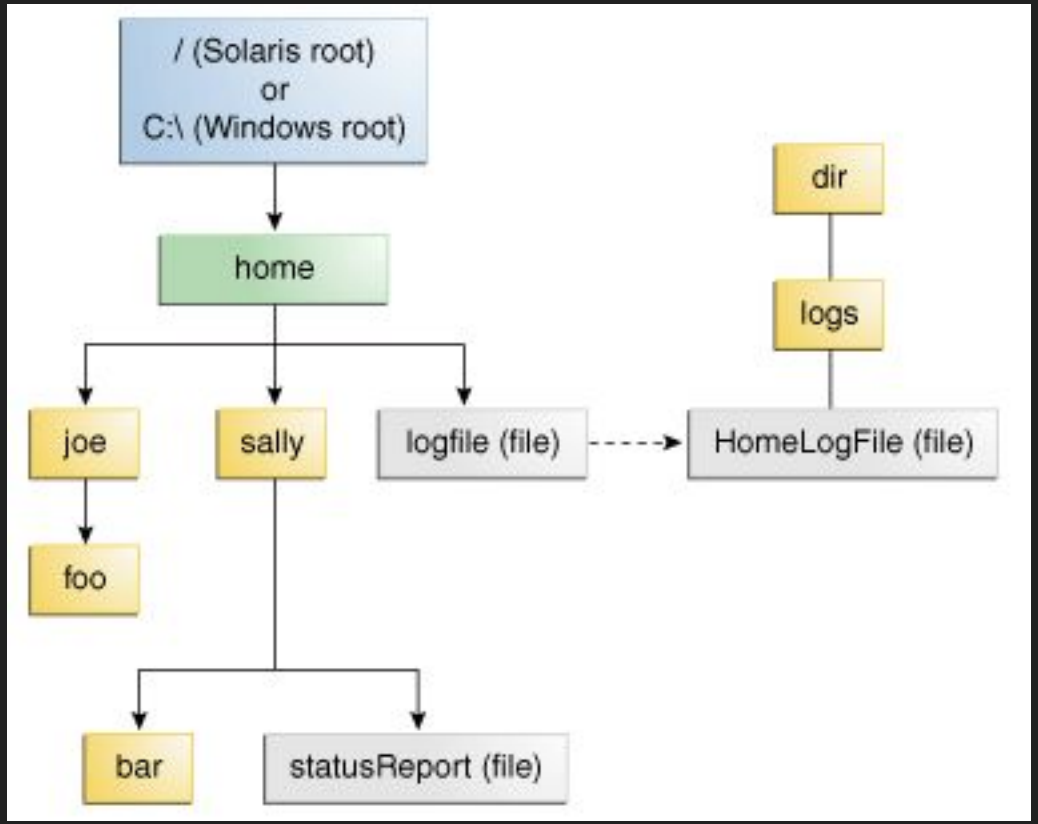
- Relativo
  - Precisa ser combinado com outro para acessar um arquivo
  - Ex: *joe/foo*
  - Sem mais informações

Caminho (Path)

- Alguns sistemas de arquivos suportam a noção de links simbólicos (*symbolic link*)
  - São também conhecidos como symlink ou *soft link*
- É um arquivo especial que serve como referência para outro arquivo

Caminho (Path)

- Na maioria das vezes, são transparente para as aplicações, e operações neles são automaticamente redirecionadas ao alvo do *link* (o arquivo ao qual ele aponta)
  - Exceções são quando o link é deletado ou renomeado, já que apenas o link sofre essas ações e não o alvo do link



Fonte: https://docs.oracle.com/javase/tutorial/essential/io/path.html

Caminho (Path)

- No exemplo anterior, *logFile* parece ser um arquivo normal para o usuário, mas é um link simbólico para o arquivo *dir/logs/HomeLogFile*
  - Nesse caso, *HomeLogFile* é o alvo do link
- Resolver o link significa substituir a localização atual do arquivo pelo link simbólico
- No exemplo, resolver *logFile* retorna *dir/logs/HomeLogFile*

A Classe Path

A Classe Path

- Representação de um caminho no sistema de arquivos
- Um objeto de Path contém um nome de arquivo e lista de diretório usada para construir o caminho
  - É usada para examinar, localizar e manipular arquivos
- Reflete a plataforma em que executa (SO)
  - Não é independente do sistema!
    - Não é possível igualar um Path de um sistema de arquivos Solaris de um do Windows

A Classe Path

- Para criar um objeto Path pode-se usar um dos métodos get:
  - `Path p1 = Paths.get("/tmp/foo");`
  - `Path p2 = Paths.get(args[0]);`
  - `Path p3 = Paths.get(URI.create("file:///Users/joe/FileTest.java"));`
- Em que *Paths.get* é uma abreviação do código:
  - `Path p4 = FileSystems.getDefault().getPath("/users/sally");`

A Classe Path

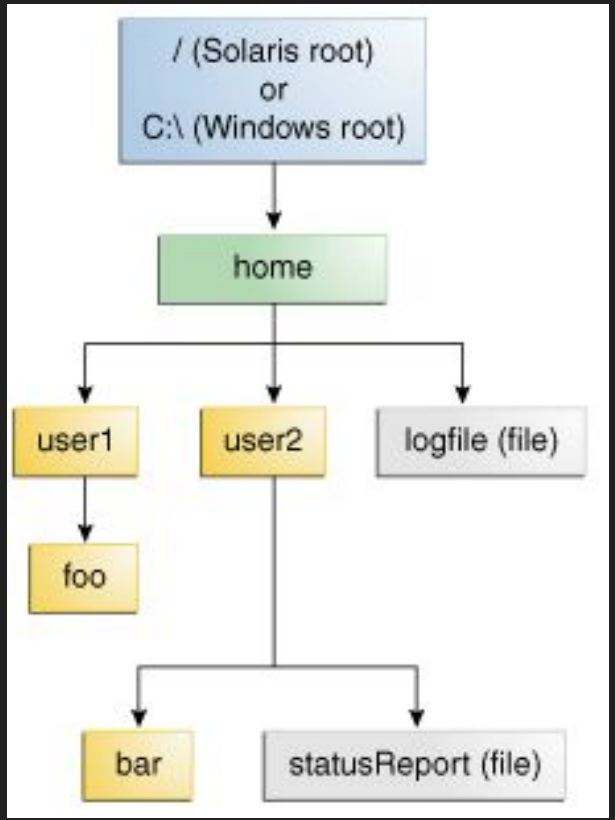
- O exemplo a seguir cria */u/joe/logs/foo.log* se seu diretório *home* é */u/joe*, ou *C:\joe\logs\foo.log* no Windows:

```
Path p5 =

    Paths.get(System.getProperty("user.home"),"logs", "foo.log");
```

A Classe Path

- Pode-se pensar como se o *Path* guardasse esses elementos de nome em sequência
  - O elemento mais alto na hierarquia tem o índice 0
  - O mais baixo, [n-1]
    - Sendo *n* o número de elementos de nome em Path
  - É possível recuperar elementos individuais usando esses índices com métodos de *Path*
- Vamos ver um exemplo:



Fonte:
https://docs.oracle.com/javase/tutorial/essential/io/pathOps.html

A Classe Path

```
// Microsoft Windows syntax
Path path = Paths.get("C:\\home\\joe\\foo");
// Solaris syntax
Path path = Paths.get("/home/joe/foo");

System.out.format("toString: %s%n", path.toString());
System.out.format("getFileName: %s%n", path.getFileName());
System.out.format("getName(0): %s%n", path.getName(0));
System.out.format("getNameCount: %d%n", path.getNameCount());
System.out.format("subpath(0,2): %s%n", path.subpath(0,2));
System.out.format("getParent: %s%n", path.getParent());
System.out.format("getRoot: %s%n", path.getRoot());
```

Method Invoked	Returns in the Solaris OS	Returns in Microsoft Windows	Comment
toString	/home/joe/foo	C:\home\joe\foo	Returns the string representation of the Path. If the path was created using Filesystems.getDefault().getPath(String) or Paths.get (the latter is a convenience method for getPath), the method performs minor syntactic cleanup. For example, in a UNIX operating system, it will correct the input string //home/joe/foo to /home/joe/foo.
getFileName	foo	foo	Returns the file name or the last element of the sequence of name elements.
getName(0)	home	home	Returns the path element corresponding to the specified index. The 0th element is the path element closest to the root.
getNameCount	3	3	Returns the number of elements in the path.
subpath(0,2)	home/joe	home\joe	Returns the subsequence of the Path (not including a root element) as specified by the beginning and ending indexes.
getParent	/home/joe	\home\joe	Returns the path of the parent directory.
getRoot	/	C:\	Returns the root of the path.

A Classe Path

➔ Exemplo com caminho relativo:

```
// Solaris syntax
Path path = Paths.get("sally/bar");
or
// Microsoft Windows syntax
Path path = Paths.get("sally\\bar");
```

Method Invoked	Returns in the Solaris OS	Returns in Microsoft Windows
toString	sally/bar	sally\bar
getFileName	bar	bar
getName(0)	sally	sally
getNameCount	2	2
subpath(0,1)	sally	sally
getParent	sally	sally
getRoot	null	null

A Classe Path

- ➔ Existem vários outros métodos da classe Path como retirar redundâncias, converter caminhos a partir de outras formatações de entrada (como string), juntar caminhos, criar caminhos entre caminhos, comparar dois caminhos, etc.
- ➔ Confiram a documentação :)

A Classe Files

A Classe Files

- A classe Files é a outra principal do pacote java.nio.file
- Oferece vários métodos estáticos para leitura, escrita e manipulação de arquivos e diretórios
- Trabalho em cima de objetos de Path

A Classe Files

➔ É possível verificar se um arquivo existe e se o programa tem permissão para executá-lo

```
Path file = ...;
boolean isRegularExecutableFile =
    Files.isRegularFile(file) &
    Files.isReadable(file) &
    Files.isExecutable(file);
```

A Classe Files

- ➔ É possível deletar arquivos, diretórios e links
  - ◆ Para links simbólicos, apenas o link é deletado
  - ◆ Se um diretório não estiver vazio, o método falha
- ➔ Existe o método `deleteIfExists(Path)`
  - ◆ Deleta o arquivo se existir
    - ● Se ele não existir, não joga nenhuma exceção
- ➔ O `delete(Path)` joga exceção caso não exista

A Classe Files

```
try {
    Files.delete(path);
} catch (NoSuchFileException x) {
    System.err.format("%s: no such" + " file or
directory%n", path);
} catch (DirectoryNotEmptyException x) {
    System.err.format("%s not empty%n", path);
} catch (IOException x) {
    // File permission problems are caught here.
    System.err.println(x);
}
```

A Classe Files

- Para ler de/escrever em arquivos, é possível ler/escrever todos os bytes de uma vez
  - Para arquivos pequenos!

```
Path file = ...;
byte[] fileArray;
fileArray = Files.readAllBytes(file);
byte[] buf = ...;
Files.write(file, buf);
```

A Classe Files

- Para arquivos maiores, podemos usar os Buffered Streams
  - BufferedReader
  - BufferedWriter

A Classe Files

```
Charset charset = Charset.forName("US-ASCII");
try (BufferedReader reader = Files.newBufferedReader(file,
charset)) {
    String line = null;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
} catch (IOException x) {
    System.err.format("IOException: %s%n", x);
}
```

A Classe Files

```
Charset charset = Charset.forName("US-ASCII");
String s = ...;
try (BufferedWriter writer = Files.newBufferedWriter(file,
charset)) {
    writer.write(s, 0, s.length());
} catch (IOException x) {
    System.err.format("IOException: %s%n", x);
}
```

A Classe Files

- Também é possível fazer a leitura e escrita sem o uso de buffers com as Input/Output Streams
- Existem mais algumas opções para leitura e escrita. Consultem a documentação :)

A Classe Files

```
Path file = ...;
try (InputStream in = Files.newInputStream(file);
     BufferedReader reader =
       new BufferedReader(new InputStreamReader(in))) {
     String line = null;
     while ((line = reader.readLine()) != null) {
         System.out.println(line);
     }
} catch (IOException x) {
     System.err.println(x);
}
```

A Classe Files

```
public class LogFileTest {
  public static void main(String[] args) {
    // Convert the string to a byte array.
    String s = "Hello World! ";
    byte data[] = s.getBytes();
    Path p = Paths.get("./logfile.txt");
    try (OutputStream out = new BufferedOutputStream(
      Files.newOutputStream(p, CREATE, APPEND))) {
      out.write(data, 0, data.length);
    } catch (IOException x) {
      System.err.println(x);
    }
  }
}
```

A Classe Files

- ➔ É possível fazer o acesso aleatório a arquivos também em java, através da interface SeekableByteChannel

A Classe Files

```
String s = "I was here!\n";
byte data[] = s.getBytes();
ByteBuffer out = ByteBuffer.wrap(data);
ByteBuffer copy = ByteBuffer.allocate(12);
try (FileChannel fc = (FileChannel.open(file, READ, WRITE))) {
    // Read the first 12 bytes of the file.
    int nread;
    do {
        nread = fc.read(copy);
    } while (nread != -1 && copy.hasRemaining());
    // Write "I was here!" at the beginning of the file.
    fc.position(0);
    while (out.hasRemaining())
        fc.write(out);
    out.rewind();
```

A Classe Files

```
    // Move to the end of the file.  Copy the first 12 bytes to
    // the end of the file.  Then write "I was here!" again.
    long length = fc.size();
    fc.position(length-1);
    copy.flip();
    while (copy.hasRemaining())
        fc.write(copy);
    while (out.hasRemaining())
        fc.write(out);
} catch (IOException x) {
    System.out.println("I/O Exception: " + x);
}
```

Referências

1. https://docs.oracle.com/javase/tutorial/essential/io/streams.html