Java

Archivos

I/O desde línea de comandos

Java soporta tres *Streams* estándar que son instancias de `PrintStream`:

- `System.in`: entrada estándar
- `System.out`: salida estándar
- `System.err`: salida de errores

```
public static void main(String[] args) {
   int inChar;
   System.out.println("Enter a Character:");
   try {
        inChar = System.in.read();
        System.out.print("You entered ");
        System.out.println(inChar);
   }
   catch (IOException e){
        System.out.println("Error reading from user");
   }
}
```

```
Enter a Character: a
You entered 97
```

I/O desde línea de comandos: Console

`Console` es más avanzado que Data Streams

```
Console c = System.console();
if (c == null) {
   // Error
}

String login = c.readLine("Login: ");
char [] oldPassword = c.readPassword("Password: ");
```

NIO (*non-blocking I/O*)

- Las APIs para I/O en Java presentaban problemas para los desarrolladores
  - No podían extenderse
  - Algunos métodos no eran consistentes en todos los S.O.
  - Muchas instancias para acceder archivos
- JDK 7 introduce NIO.2 para resolver estos problemas

Java 7: NIO.2

- package `java.nio.file`
- clase `Path` en reemplazo de `File`
- Mejor manejo de directorios
  - `list()` para iterar
  - filtro con expresiones regulares
- package `java.nio.file.attribute`
  - acceso a metadatos de un archivo

Path

Es una representación de un *path* en un *file system*

Contiene el nombre del archivo y el directorio

Se usa para examinar, ubicar, manipular archivos

Contiene métodos para obtener info de carpetas, acceder archivos, etc.

Path: operaciones

Crear una instancia de `Path`

```
Path p1 = Paths.get("/tmp/foo");
Path p2 = Paths.get(args[0]);
Path p3 = Paths.get(URI.create("file:///Users/joe/FileTest.java"));
Path p4 = FileSystems.getDefault().getPath("/users/sally");
Path p5 = Paths.get(System.getProperty("user.home"),"logs", "foo.log");
```

Path: operaciones

Obtener información

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

// Solaris syntax
Path path = Paths.get("/home/joe/foo");
```

Método aplicado a path	Respuesta Solaris OS	Respuesta Microsoft Windows
`toString`	`/home/joe/foo`	`C:\home\joe\foo`
`getFileName`	`foo`	`foo`
`getName(0)`	`home`	`home`
`getNameCount`	`3`	`3`
`subpath(0,2)`	`home/joe`	`home\joe`
`getParent`	`/home/joe`	`\home\joe`
`getRoot`	`/`	`C:\`

Path: conversiones

Convertir a un string que pueda usar un browser

```
Path p1 = Paths.get("/home/logfile");

System.out.format("%s%n", p1.toUri()); // => file:///home/logfile
```

El método `toAbsolutePath` obtiene un path absoluto en base a uno relativo.

Files

La clase `Files` implementa métodos estáticos para leer, escribir y manipular archivos y directorios

Método	
`exists(Path)`	`false si no existe o no es accesible`
`notExists(Path)`	`false si existe o no es accesible`
`isReadable(Path)`	
`isWritable(Path)`	
`isExecutable(Path)`	
`isSameFile(Path, Path)`	
`delete(Path)`	
`copy(Path, Path, CopyOption…)`	
`move(Path, Path, CopyOption…)`	

Files: atributos

Método	
`size(Path)`	`tamaño en bytes.`
`isDirectory(Path, LinkOption)`	`true si el Path representa un directorio.`
`isRegularFile(Path, LinkOption...)`	`true si es un archivo.`
`getLastModifiedTime(Path, LinkOption...)` `setLastModifiedTime(Path, FileTime)`	`retorna o cambia la fecha de última modificación.`
`getOwner(Path, LinkOption...)` `setOwner(Path, UserPrincipal)`	`retorna o cambia el owner.`
`...`	

Crear, leer, escribir archivos

Hay muchas formas de llevar a cabo estas acciones, sólo mencionaremos algunas.

Leer el archivo completo (sólo para archivos pequeños)

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

```
Path file = ...;
List<String> lines;
lines = Files.readAllLines(file);
```

Escribir todos los bytes (sólo para archivos pequeños)

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

```
Path file = ...;
byte[] buf = ...;
Files.write(file, buf,StandardOpenOption.CREATE,
     StandardOpenOption.APPEND);
```

Acceso “bufferizado”

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

```
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);
}
```

Acceso “bufferizado”

```
Charset charset = Charset.forName("US-ASCII");
try (BufferedReader reader = Files.newBufferedReader(file, charset)) {
         char[] buff = new char[MAX];


   int read;
         while ((read = reader.read(buff, 0, MAX)) != -1) {
        System.out.print(buff);
   }
   reader.close();
} catch (IOException ex) {
    System.err.format("IOException: %s%n", ex);
}
```

Acceso “bufferizado”

```
try {
   BufferedReader reader = Files.newBufferedReader(file, charset);

   int c;
   while ((c = reader.read()) != -1) {
       System.out.print((char)c);
   }
   reader.close();
} catch (IOException ex) {
   System.err.format("IOException: %s%n", ex);
}
```

Para más detalles sobre BufferedReader ver
https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html

Uso de flujo no bufferizado

`newInputStream` abre un archivo y retorna un *stream* para leer desde el archivo. El *stream* no se bufferiza y por lo tanto no soporta ciertas operaciones (mark, reset)

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

Crear archivos temporales

```
try {
    Path tempFile = Files.createTempFile("aed", ".txt");
    System.out.format("File created: %s\n", tempFile);
} catch (IOException e) {
    System.err.format("IOException: %s%n", e);
}
```

File created: /tmp/aed4185302790371 34.txt

Archivos binarios

```
Path file = Files.createTempFile("", ".bin");
String f = file.toString();

DataOutputStream out = new DataOutputStream(Files.newOutputStream(file));
out.writeDouble(1.5);
out.writeBoolean(true);
out.writeUTF("This is the end.");
out.close();

DataInputStream in = new DataInputStream(Files.newInputStream(file));
Double x = in.readDouble();
Boolean b = in.readBoolean();
String s = in.readUTF();
```

Uso de streams

```
public static void main(String args[]) {

       String fileName = "c://lines.txt";

       //read file into stream, try-with-resources
       try (Stream<String> stream =
          Files.lines(Paths.get(fileName))) {

                     stream.forEach(System.out::println);
             } catch (IOException e) {
                     e.printStackTrace();
             }
       }
```

Más sobre archivos

- Acceso aleatorio
- Creación de directorios
- Links simbólicos
- Recorrer un directorio
- Watch service