Ejercicio U8_B4_E1:

OJO: puedes tener problemas de permisos al ejecutar estos programas, usa directorios como por ejemplo el directorio donde guardas tus proyectos. No uses c:\windows o similares que suelen tener permisos extra que generan IOException al intentar ser accedidos

```
import java.io.*;
class App{
  public static void main(String[] args) {
      File dir = new File(args[0]);
      if(!dir.isDirectory()){
         System.out.println("El directorio NO existe");
      }
      else{
         for(String s:dir.list())
            System.out.println(s);

      }
  }
}
```

CON walk()

```
import java.io.*;
import java.nio.file.*;
import java.util.stream.Stream;
class App{
   public static void main(String[] args) throws IOException {
      Path dir =Paths.get(args[0]);
      if(!Files.exists(dir)){
         System.out.println("El directorio NO existe");
      }else{
       Stream<Path> listadir=Files.walk(dir,1);
        listadir.forEach(System.out::println);
      }
  }
}
```

Observa que contamos con que los objetos Path tienen sobreescrito el toString()

Ejercicio U8_B4_E2:

Ojo coge un un buen directorio de prueba:

- que no sea gigante. Si la lista a imprimir es muy grande hay contenidos que no veremos "con el scroll del cmd" porque excedimos la cantidad de memoria de buffer del cmd
- Si intentamos listar directorios para los que no tenemos permisos en el sistema de ficheros (por ejemplo algunos ficheros y directorios que hay dentro de c:\windows) se produce una IOException y list() devuelve null

En el ejemplo usamos getAbsolutePath que siempre devuelve la ruta absoluta. No usamos getPath() ya que puede devolver una ruta relativa si en el new File() usamos ruta relativa

```
import java.io.*;
class App {
   public static void main(String[] args) {
      File miruta = new File("C:\\Users\\donlo\\Documents\\NetBeansProjects\\App");
      if (!miruta.isDirectory()) {
         System.out.println("El directorio NO existe");
      } else {
         listarDirectorio(miruta);
      }
   }

   static void listarDirectorio(File f) {
```

```
        System.out.println(f.getAbsolutePath());
        if (f.isDirectory()) {
            for (String s : f.list()) {
                listarDirectorio(new File(f.getAbsolutePath() + "/" + s));
            }
        }
    }
}
```

CON walk()

```
import java.io.*;
import java.nio.file.*;
import java.util.stream.Stream;
class App{
    public static void main(String[] args) throws IOException {
        Path dir =Paths.get(args[0]);

        Stream<Path> listadir=Files.walk(dir);
        //listadir.forEach(p->System.out.println(p));
        listadir.forEach(System.out::println);
   }
}
```

Antes del java 7 no había walk y el que no utilizaba recursividad las pasaba canutas para hacer un listado del estilo anterior.
walk() tiene un tercer parámetro que permite configurar su salida, pero ya no nos interesa tanto detalle.

Ejercicio U8_B4_E3:

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class App {

    public static void main(String[] args) throws IOException {
        Stream<Path> stream = Files.list(Paths.get(""));
        String listaRutas = stream
                .filter(path->!Files.isDirectory(path))
                .map(String::valueOf)
                .sorted()
                .collect(Collectors.joining("; "));
        System.out.println("Lista de rutas: " + listaRutas);
    }
}
```

Recuerda que map() nos vale para tener un nuevo tipo de Stream, en el ejemplo a Map le entra un Stream<Path> y devuelve un Stream<String>
Si lo haces con lambda puede estar más claro

```
  .map(p->String.valueOf(p))
```

Ejercicio U8_B4_E4:

Observa que puede haber un directorio que acabe su nombre con las letras txt, aunque sea confuso puede ocurrir. Por eso mantenemos el filtro de directorios, y observa cuando filtro directorios el stream tiene que ser Stream<Path> y luego cuando filtro por strings el Stream ya después del map es un Stream<String>

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.stream.Collectors;
```

```
import java.util.stream.Stream;
public class App {
   public static void main(String[] args) throws IOException {
      Path comienzo = Paths.get("");//ruta de comienzo busqueda
      Stream<Path> stream = Files.walk(comienzo,1);
      String listaRutas = stream
            .filter(path->!Files.isDirectory(path))
            .map(String::valueOf)
            .filter(s->s.endsWith(".txt"))
            .sorted()
            .collect(Collectors.joining("; "));
      System.out.println("Lista de rutas: " + listaRutas);
   }
}
```

Ejercicio U8_B4_E5:

```
import java.io.IOException;
import java.util.zip.ZipFile;

public class App {

   public static void main(String[] args)  throws IOException {
      try (ZipFile zipFile = new ZipFile("mizip.zip")) {
         zipFile.stream().sorted((zpe1, zpe2) -> (zpe1.getSize() < zpe2.getSize()) ? 1 : -1)
               .forEach(zpe -> System.out.println(zpe.getName() + " " + zpe.getSize()));
      }

   }
}
```

recuerda que la lógica del comparador puede escribirse de varias formas por ejemplo:
`(zpe1,zpe2)-> zpe1.getSize()>zpe2.getSize()?-1:1`

esta última podríamos leerla : "si zpe1 es más grande que zpe2 entonces que vaya antes zpe1, es decir, devuelvo -1"