Section 14: Basic Input & Output including java.util

15 July 2021 19:23

Section 14: Basic Input & Output including java.util

Exceptions : Try, Catch, Throw

1) LBYL : Look before You Leave

```
private static int divideLBYL(int x, int y) {
   if(y != 0) {
      return x / y;
   } else {
      return 0;
   }
}
```

2) EAFP : Easy to Ask Forgiveness Permission

```
private static int divideEAFP(int x, int y) {
   try {
      return x / y;
   } catch(ArithmeticException e) {
      return 0;
   }
}
```

Throwing an exception :-

```
try {
        return x / y;
    } catch(ArithmeticException e) {
        throw new ArithmeticException("attempt to divide by zero");
    }
```

Writing content - FileWriter class and Finally block :

```
try
{
        locFile = new FileWriter("locations.txt");
        for(Location location : locations.values())
        {
           locFile.write(location.getLocationID() + "," + location.getDescription() + "\n");
        }
}
catch(IOException e)
{
        System.out.println("In catch block");
        e.printStackTrace();
}
finally
{
        System.out.println("in finally block");
        try
        {
           if(locFile != null)
           {
```

```
            System.out.println("Attempting to close locfile");
            locFile.close();
        }
    }
    catch(IOException e)
    {
        e.printStackTrace();
    }
}
```

https://docs.oracle.com/javase/7/docs/technotes/guides/language/try-with-resources.html

Java 7 added new feature :

```
try(FileWriter locFile = new FileWriter("locations.txt"))
{
    for(Location location : locations.values())
    {
        locFile.write(location.getLocationID() + "," + location.getDescription() + "\n");
    }
}
```

Reading data from a file :-

```
Scanner scanner = null;
    try
    {
        scanner = new Scanner(new FileReader("locations.txt"));
        scanner.useDelimiter(",");
        while(scanner.hasNextLine()) {
            int loc = scanner.nextInt();
            scanner.skip(scanner.delimiter());
            String description = scanner.nextLine();
            System.out.println("Imported loc: " + loc + ": " + description);
            Map<String, Integer> tempExit = new HashMap<>();
            locations.put(loc, new Location(loc, description, tempExit));
        }

    }

// Now read the exits
    try (BufferedReader dirFile = new BufferedReader(new FileReader("directions_big.txt"))) {
        String input;
        while((input = dirFile.readLine()) != null)
        { ……. }
```

The process of translating a data structure or object into a format that can be stored and recreated is called **Serialization**.

Java NIO Package : In Java 1.4, A new package was added to the Java SDK.
Called java.nio, the package was described as an improvement to Java I/O because the classes in the package perform I/O in a non-blocking manner.

// **Reading data**

```
try (BufferedReader dirFile = Files.newBufferedReader(dirPath))
{
```

```
        String input;
        while ((input = dirFile.readLine()) != null)
        {
                String[] data = input.split(",");
                int loc = Integer.parseInt(data[0]);
                String direction = data[1];
        }
}
catch (IOException e)
{
        e.printStackTrace();

}
```

Java NIO - Reading and Writing

```
 Path dataPath = FileSystems.getDefault().getPath("data.txt");
        Files.write(dataPath, "\nLine 5".getBytes("UTF-8"), StandardOpenOption.APPEND); //writing
         List<String> lines = Files.readAllLines(dataPath);
         for(String line : lines) {
            System.out.println(line);
         }
```

Writing data to a data.dat file :-

```
 try(FileOutputStream binFile = new FileOutputStream("data.dat");
         FileChannel binChannel = binFile.getChannel())
{

         byte[] outputBytes = "Hello World!".getBytes();
         ByteBuffer buffer = ByteBuffer.wrap(outputBytes);
         int numBytes = binChannel.write(buffer);
         System.out.println("numBytes written was: " + numBytes);
…….
}
```

File System :-

Each folder, which is also referred to as a directory, is also a node in a path. And then, of course, theirs the file itself.

Absolute Path : Specify root mode (starting from which location)
Relative Path : Doesn't specify a root node, it doesn't contain enough information to identify the file.

Reading a file using a given Path :-

```
   private static void printFile(Path path)
   {
      try(BufferedReader fileReader = Files.newBufferedReader(path))
      {
         String line;
         while((line = fileReader.readLine()) != null) {
            System.out.println(line);
         }
      }
      catch(IOException e)
      {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }
   }
```

Getting Path from a working project :-
Path path = FileSystems.getDefault().getPath("FileName.txt");
printFile(path);

For a random path :-
filePath = Paths.get("C:\\whole\\path\\fileName.txt");
printFile(path);

Get Current Path :-
filePath = Paths.get(".");
System.out.println(filePath.toAbsolutePath());

FileSystems.getDefault().getPath(".") is same as Paths.get(".");

Copy a File :-
Files.copy(sourceFilePath, copyFilePath);

Move a File :-
Files.move(fileToMove, destination);

Delete a File :-
Files.delete(fileToDelete);
Files.deleteIfExists(fileToDelete);

Create a File :-
Files.createFile(fileToCreate);

Create a Directory :-
Files.createDirectory(dirToCreate);

https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html
https://docs.oracle.com/javase/8/docs/api/java/nio/file/attribute/BasicFileAttributes.html

Read name of all files in a Directory:-

```
Path directory = FileSystems.getDefault().getPath("DIrectoryName\\Folder");
  try (DirectoryStream<Path> contents = Files.newDirectoryStream(directory))
                                        // instead of \\ we can use File.separator
  {
    for (Path file : contents)
    {
      System.out.println(file.getFileName());
    }

  }
  catch (IOException | DirectoryIteratorException e)
  {
    System.out.println(e.getMessage());
  }
```

A filter can also be pass on Files.newDirectoryStream as another argument to filter all files in the directory.
https://docs.oracle.com/javase/8/docs/api/java/nio/file/FileSystem.html#getPathMatcher-java.lang.String-

Walk the Directory :- To read all the files and directory, files/directories that are itself in a directory can be read.

Take away :- It's to use java.nio when working with a file system. But when it comes to reading and writing file contents, sometimes java.io streams are still the better choice.