PER SCHOLAS

Introduction to Java Programming

Working with Basic Input & Output

Java IO

- ❑ Students should be able to understand:
  - ➢ Absolute and Relative Reads
  - ➢ Chained Put Methods

Absolute and Relative

How java locates resources

- Java loads resources from the “environment”, in many cases it uses all jars in Classpath to retrieve the resource. Resource loading in java is called location independent because it is not relevant where your code is running, it just needs correct environment to find the resources.
- Given all jars and paths in Classpath, java will search relatively to each tone to find the resource you specified. You specify resources using resource names.
- The common way of accessing the file system in java is through the java.io.File API. In this tutorial, we explain how to use the File API to access a file using absolute and relative paths, we also describe the difference between **getPath()**, **getAbsolutePath()** and **getCanonicalPath()** methods provided by the API.

Absolute and Relative

How java locates resources

- In general, a path is a way to refer to a particular file or directory in a file system, there are 2 types of path: *absolute*and *relative*. One should understand the difference between these 2 types in order to successfully locate a file within a program.
- Absolute Path
  - Simply, a path is absolute if it starts with the root element of the file system. In windows, the root element is a drive e.g. C:\\, D:\\, while in unix it is denoted by "/" character.
  - An absolute path is complete in that no other information is required to locate the file, it usually holds the complete directory list starting from the root node of the file system till reaching the file or directory it denotes.
  - Since absolute path is static and platform dependent, it is a bad practice to locate a file using absolute path inside your program, since you will lose the ability to reuse your program on different machines and platforms.
- Relative Path
  - A relative path is a path which doesn't start with the root element of the file system. It is simply the path needed in order to locate the file from within the current directory of your program. It is not complete and needs to be combined with the current directory path in order to reach the requested file.
  - In order to construct a rigid and platform independent program, it is a common convention to use a relative path when locating a file inside your program.

Absolute and Relative

Checking if a path is relative

- The File.isAbsolute() method gives us a hand in this task, telling if the path is absolute. Example:
  - **File f1 = new File("..");**
    System.out.println(f1.isAbsolute()); //prints false

    File f2 = new File("c:\\temp");
    System.out.println(f2.isAbsolute()); //prints true
- Another useful method is getAbsolutePath(). It returns the absolute path to an instance of File.
  - Another example:
  - **File f1 = new File("\\directory\\file.xml");**
    System.out.println(f1.getAbsolutePath()); //prints C:\directory\file.xml

    File f2 = new File("c:\\directory\\file.xml");
    System.out.println(f2.getAbsolutePath()); //prints c:\directory\file.xml

Absolute and Relative

Features

Feature	**Description**
getParentFile	returns a File pointing to the directory that contains the current file or directory.
getAbsoluteFile	returns another instance of File with an absolute path.
toURI	returns a URI (*Universal Resource Identifier*) that begins with file:. It's useful to network operations.
isFile e **idDirectory**	tells if File points to a file or directory, respectively.
exists	tells if the file or directory really exists in filesystem.
canRead e **canWrite**	tells if you can read or write to the file, respectively.
createNewFile	creates a new blank file.
delete	removes the file or directory (if empty).

PER SCHOLAS

Absolute and Relative

Features

Feature	Description
length	returns the file size in bytes.
list e **listFiles**	lists files and directories, if File is a directory.
mkdir e **mkdirs**	creates a new directory, if File is a directory. The latter also creates parent directories if needed.
getFreeSpace	returns the available space in the device to where File is pointing to.
createTempFile	static method that returns an unique temporary file to be used by the application. The method deleteOnExit will delete the file at the termination of the program.

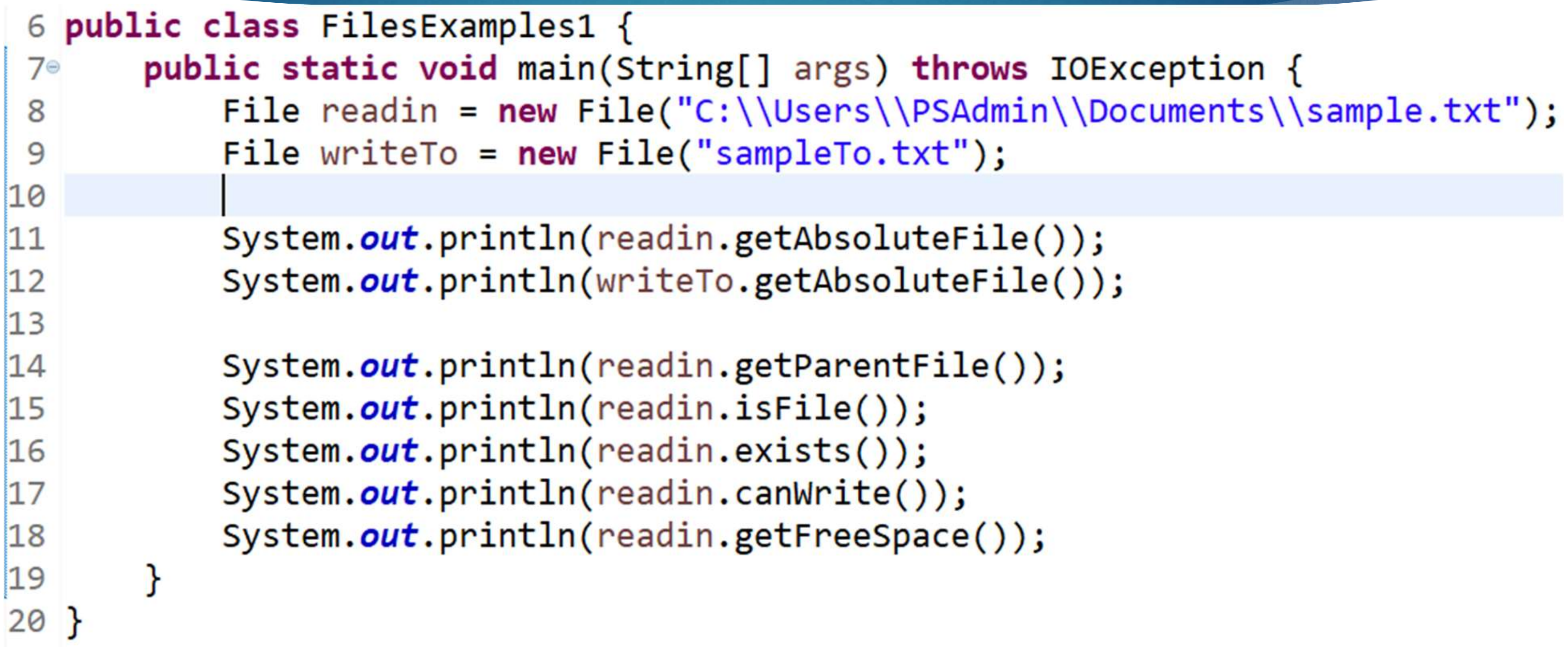
Absolute and Relative

Features

- The class File also contains some important static attributes useful to read and write files in different platforms:
    - **File.separator**: path-name separator character. On Unix and Linux it is /, while on Windows it is \.
    - **File.pathSeparator**: path-separator character, in order to create a path list with various directories, like *PATH* system variable. On Unix and Linux it is :, while on Windows it is ;.

Absolute and Relative

Working With Files



```java
public class FilesExamples1 {
    public static void main(String[] args) throws IOException {
        File readin = new File("C:\\Users\\PSAdmin\\Documents\\sample.txt");
        File writeTo = new File("sampleTo.txt");
        
        System.out.println(readin.getAbsoluteFile());
        System.out.println(writeTo.getAbsoluteFile());

        System.out.println(readin.getParentFile());
        System.out.println(readin.isFile());
        System.out.println(readin.exists());
        System.out.println(readin.canWrite());
        System.out.println(readin.getFreeSpace());
    }
}
```

Absolute and Relative

Working With Files

- Java NIO Files class contains static methods that is used for manipulating files and directories and those methods mostly works on Path object.
- Let's have a look at below methods of Files class:

Method Name	Description
copy(InputStream in, Path target, CopyOption… options)	This method copies all bytes from specified input stream to specified target file and it returns number of bytes read or written as long value.
copy(Path source, OutputStream out)	This method copies all bytes from specified source file to given output stream and it returns number of bytes read or written as long value.
copy(Path source, Path target, CopyOption… options)	This method copies given source file to specified target file and it returns path of target file.

PER SCHOLAS

Absolute and Relative

Working With Files

Method Name	Description
createDirectories(Path dir, FileAttribute<?>… attrs)	This method creates a directories using given path by creating all nonexistent parent directories first. This method will not throw an exception if the directory could not be created because it already exists. FileAttribute is an optional parameter to set automatically when creating the nonexistent directories and it returns the path of created directory.
createDirectory(Path dir, FileAttribute<?>… attrs)	This method creates a directory using given path, if it creates directory successfully it will returns the path of the created directory. If directory is already exists then it will throw **nio.file.FileAlreadyExistsException.**
createFile(Path path, FileAttribute<?>… attrs)	This method creates a new empty file using given path and returns path of newly created file if it creates it successfully. If file is already exists then it will throw **nio.file.FileAlreadyExistsException.**
createTempDirectory(Path dir, String prefix, FileAttribute<?>… attrs)	This method creates a Temporary directory using given path and it will generate the name of directory using given prefix. It will return the path of newly created temporary directory.

Absolute and Relative

Working With Files

Method Name	Description
createTempDirectory(String prefix, FileAttribute<?>… attrs)	This method creates a Temporary directory in the default temporary-file directory and generate the name of directory using given prefix. It will return the path of newly created temporary directory which is associated with the default File System.
createTempFile(Path dir, String prefix, String suffix, FileAttribute<?>… attrs)	This method creates a temporary file at specified directory and generates the file name using given prefix and suffix and returns the path of newly created file.
createTempFile(String prefix, String suffix, FileAttribute<?>… attrs)	This method creates a temporary file at default temporary-file directory and generates the file name using given prefix and suffix and returns the path of newly created file.
delete(Path path)	This is a void method and simply deletes the file from specified path. This method throws **NoSuchFileException** if the file is not exists at specified path and if the file is directory and it may not empty and cannot be deleted, in this case it will throws

Absolute and Relative

Working With Files

Method Name	Description
deleteIfExists(Path path)	This methods checks if file exists before delete the file and returns boolean value true if the file at the given path gets deleted successfully and returns false if the file is not exists at given path. If the file is directory and it may not empty and cannot be deleted, in this case it will throws
exists(Path path)	This method checks if file exists at specified path and if the file exists it will return true or else it returns false.
getLastModifiedTime(Path path, Linkoption… options)	This method returns a file's last modified time from given path as
getOwner(Path path, Linkoption… options)	This method returns **UserPrincipal**representing the owner of the file at given path.
isDirectory(Path path, Linkoption… options)	This method checks if the file is a directory from given path. It returns true if the file is directory and false if the file does not exists or is not a directory or it cannot be determined if the file is a directory or not.

PER SCHOLAS

Absolute and Relative

Working With Files

Method Name	Description
isExecutable(Path path)	This method checks whether file at given path is executable or not and it also checks that file exists and that this JVM has appropriate privilege to execute the file. It returns true if the file is exists at given path and is executable and false if the file does not exists or JVM has not sufficient privilege to execute file or access cannot be determined.
isHidden(Path path)	This method tells whether file at given is considered as hidden or not. The exact definition of hidden is platform or provider dependent. In case of UNIX system a file is considered hidden if the name of file is starts with period character ('.') And in case of WINDOWS file is considered as hidden if it is not a directory and DOS hidden attribute is set. It returns true if the file at given path is considered as hidden or else false.
isReadable(Path path)	This method tests whether the file at given path is readable or not. It reruns true if the file at specified path exists and is readable and false if the file does not exists or read access is denied because JVM does not has sufficient privilege or access cannot be determined.

PER SCHOLAS

Absolute and Relative

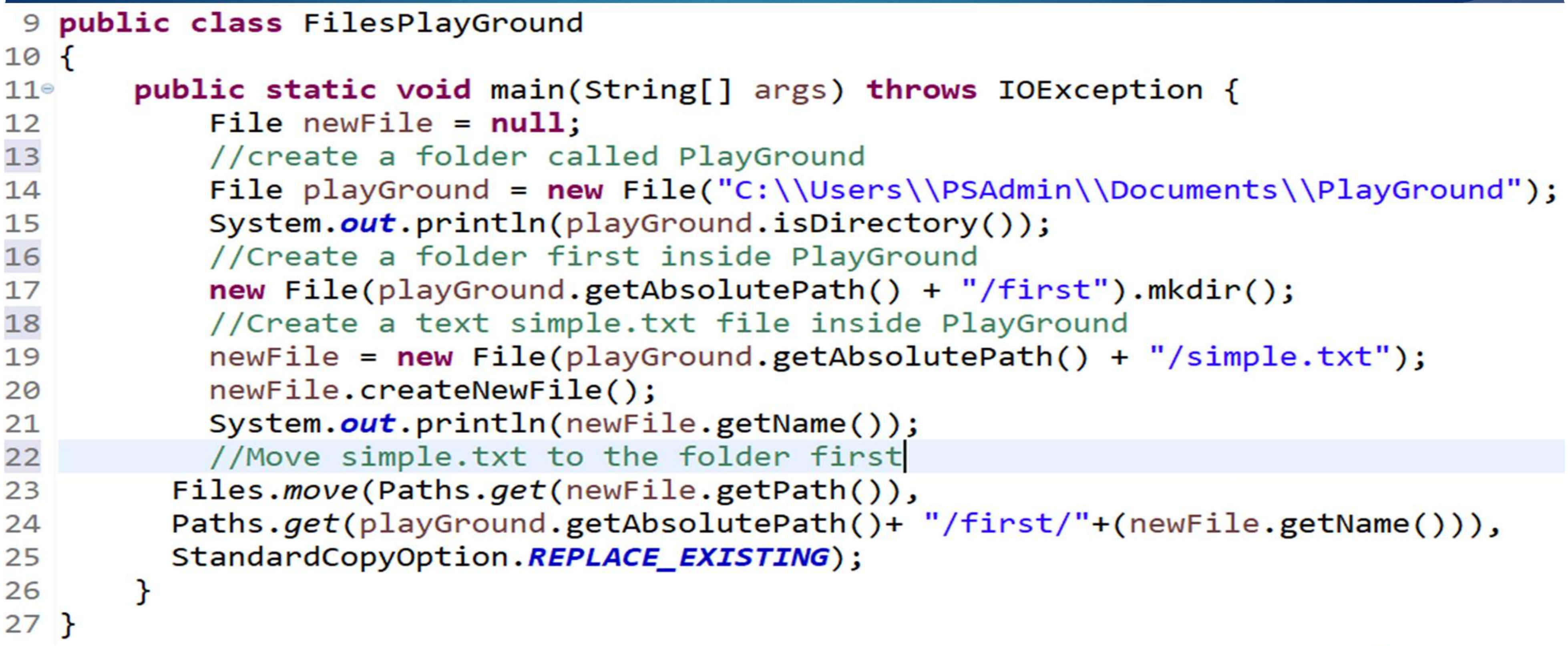
Working With Files

Method Name	Description
isWritable(Path path)	This method tests whether the file at given path is writable or not. It reruns true if the file at specified path exists and is writable and false if the file does not exists or write access is denied because JVM does not has sufficient privilege or access cannot be determined.
size(Path path)	This method returns the size of the file at specified path in bytes.
move(Path source, Path target, CopyOption… options)	his method move or rename a source file to target file and returns the path of target file. Option parameter may include following : **REPLACE_EXISTING:** It means if the target file exists then replaces it if it is not a non-empty directory. **ATOMIC_MOVE:** It means move is performed as atomic file system operation and all other options are ignored.

PER SCHOLAS

Absolute and Relative

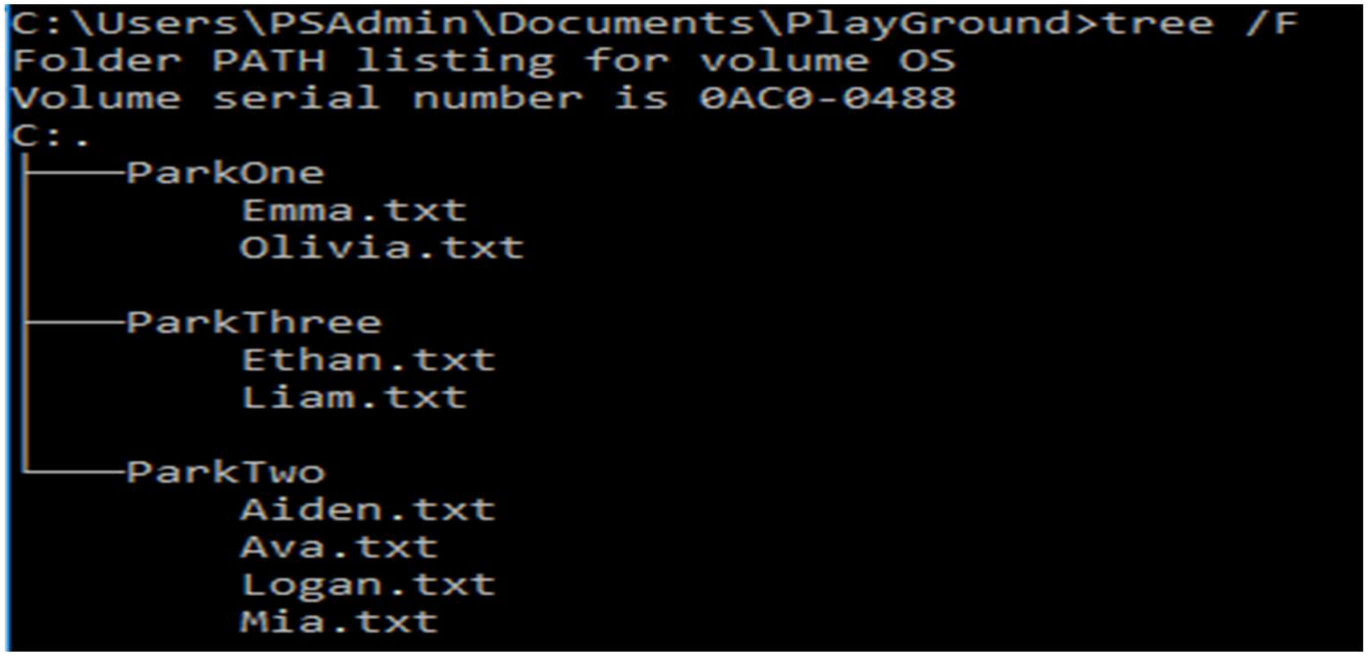
Working With Files



```java
public class FilesPlayGround
{
    public static void main(String[] args) throws IOException {
        File newFile = null;
        //create a folder called PlayGround
        File playGround = new File("C:\\Users\\PSAdmin\\Documents\\PlayGround");
        System.out.println(playGround.isDirectory());
        //Create a folder first inside PlayGround
        new File(playGround.getAbsolutePath() + "/first").mkdir();
        //Create a text simple.txt file inside PlayGround
        newFile = new File(playGround.getAbsolutePath() + "/simple.txt");
        newFile.createNewFile();
        System.out.println(newFile.getName());
        //Move simple.txt to the folder first
      Files.move(Paths.get(newFile.getPath()),
      Paths.get(playGround.getAbsolutePath()+ "/first/"+(newFile.getName())),
      StandardCopyOption.REPLACE_EXISTING);
    }
}
```

Absolute and Relative

Navigate to your Desktop and create a folder, name it: PlayGround.
Using Java, inside PlayGround create the following folder/file structure

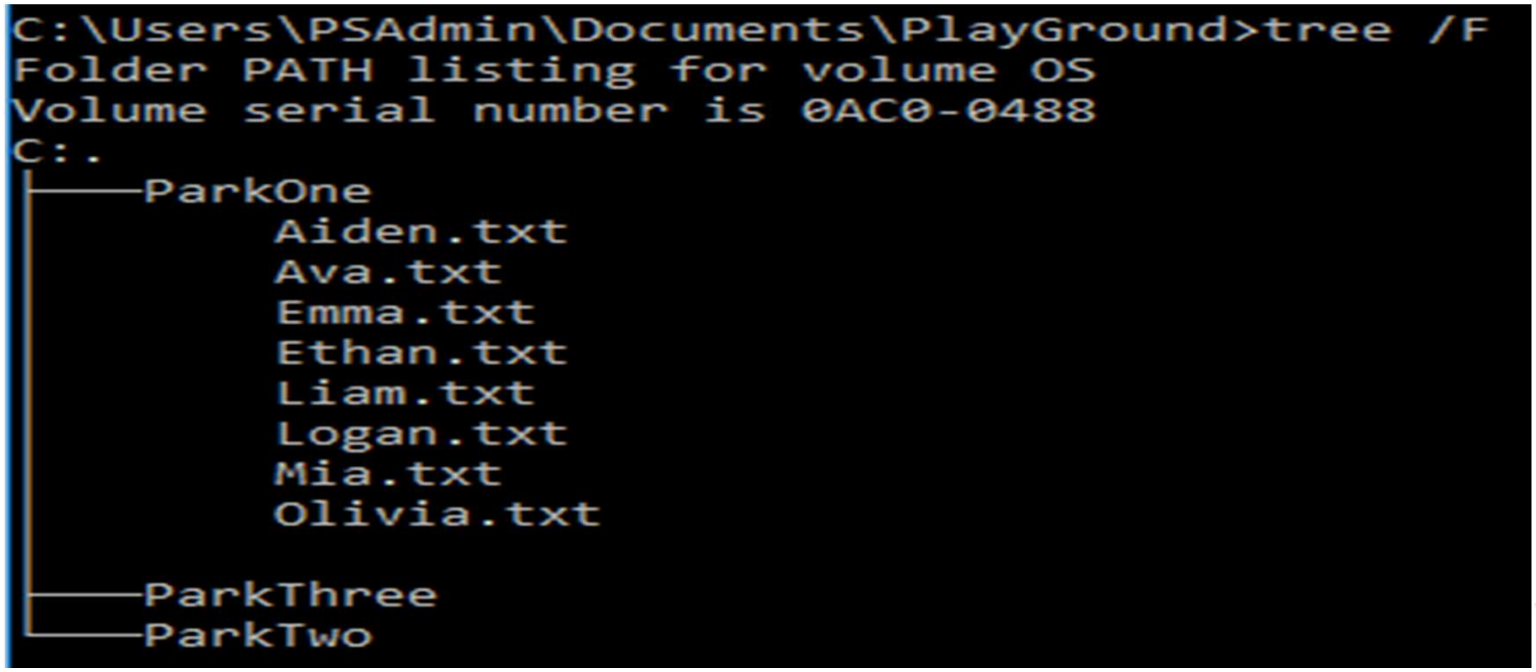


```
C:\Users\PSAdmin\Documents\PlayGround>tree /F
Folder PATH listing for volume OS
Volume serial number is 0AC0-0488
C:.
├───ParkOne
│       Emma.txt
│       Olivia.txt
│
├───ParkThree
│       Ethan.txt
│       Liam.txt
│
└───ParkTwo
        Aiden.txt
        Ava.txt
        Logan.txt
        Mia.txt
```

Absolute and Relative

Manipulating Files With Java

Using Java, Modified the structure inside the PlayGround to look like the following image:



```
C:\Users\PSAdmin\Documents\PlayGround>tree /F
Folder PATH listing for volume OS
Volume serial number is 0AC0-0488
C:.
├───ParkOne
│       Aiden.txt
│       Ava.txt
│       Emma.txt
│       Ethan.txt
│       Liam.txt
│       Logan.txt
│       Mia.txt
│       Olivia.txt
│
├───ParkThree
└───ParkTwo
```

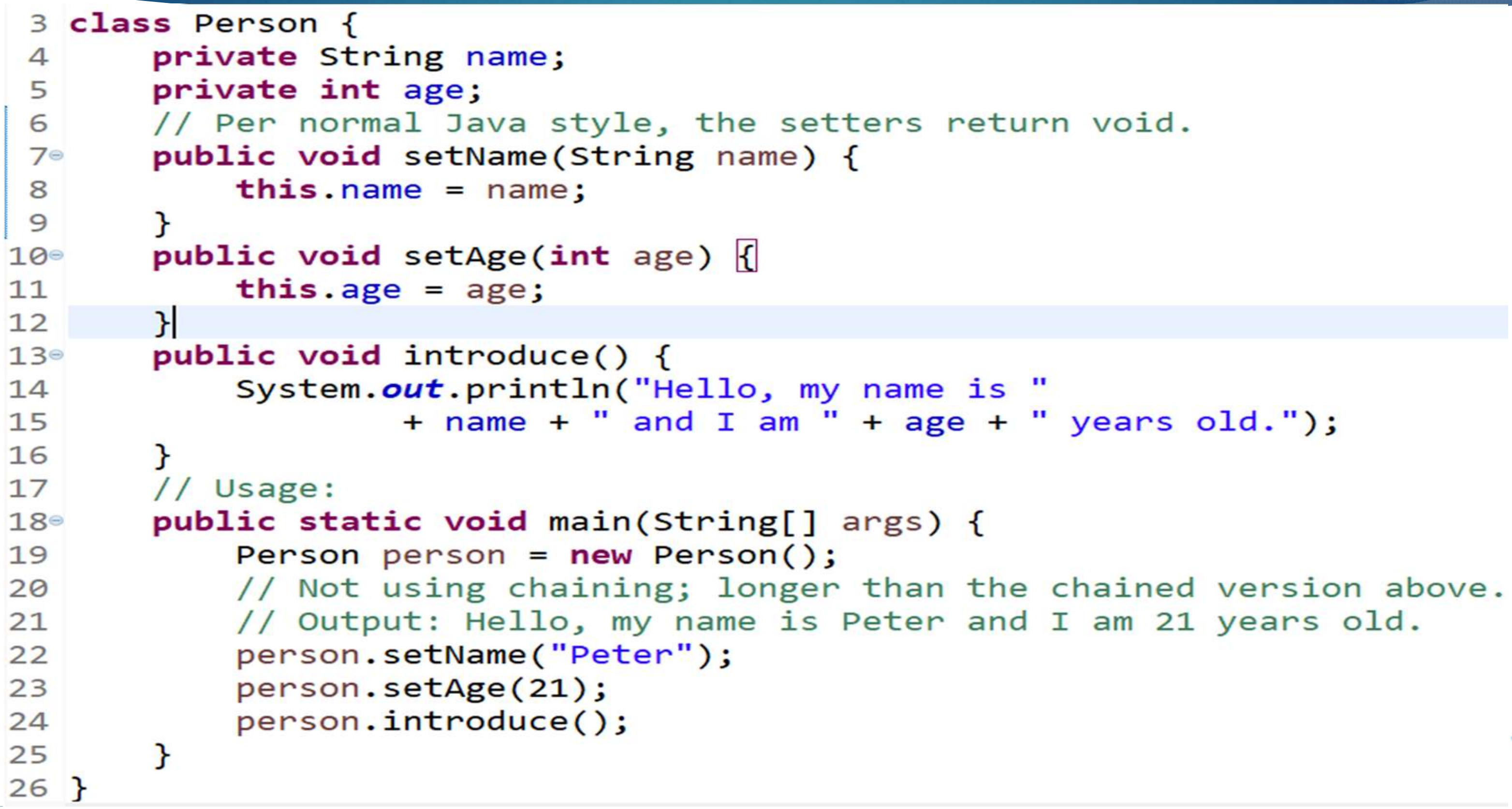
Chained Put Methods

Method Chaining

- Method chaining, also known as named parameter idiom, is a common syntax for invoking multiple method calls in object-oriented programming languages. Each method returns an object, allowing the calls to be chained together in a single statement. Chaining is syntactic sugar which eliminates the need for intermediate variables. A method chain is also known as a train wreck due to the increase in the number of methods that come one after another in the same line that occurs as more methods are chained together even though line breaks are often added between methods.

Chained Put Methods

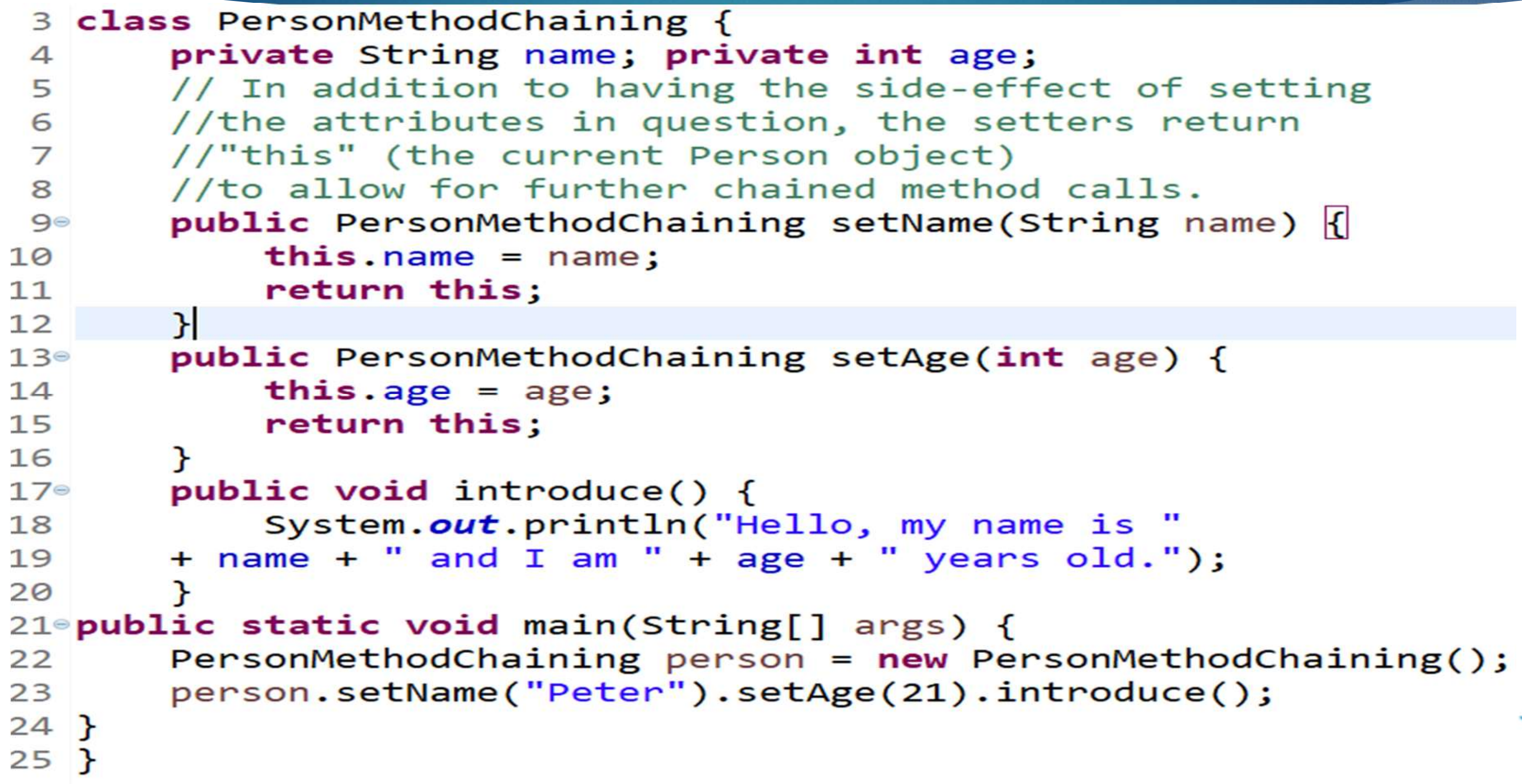
Without method chaining



```java
class Person {
     private String name;
     private int age;
     // Per normal Java style, the setters return void.
     public void setName(String name) {
          this.name = name;
     }
     public void setAge(int age) {
          this.age = age;
     }
     public void introduce() {
          System.out.println("Hello, my name is "
                    + name + " and I am " + age + " years old.");
     }
     // Usage:
     public static void main(String[] args) {
          Person person = new Person();
          // Not using chaining; longer than the chained version above.
          // Output: Hello, my name is Peter and I am 21 years old.
          person.setName("Peter");
          person.setAge(21);
          person.introduce();
     }
}
```

Chained Put Methods

With method chaining



```
class PersonMethodChaining {
    private String name; private int age;
    // In addition to having the side-effect of setting
    //the attributes in question, the setters return
    //"this" (the current Person object)
    //to allow for further chained method calls.
    public PersonMethodChaining setName(String name) {
        this.name = name;
        return this;
    }
    public PersonMethodChaining setAge(int age) {
        this.age = age;
        return this;
    }
    public void introduce() {
        System.out.println("Hello, my name is "
    + name + " and I am " + age + " years old.");
    }
public static void main(String[] args) {
    PersonMethodChaining person = new PersonMethodChaining();
    person.setName("Peter").setAge(21).introduce();
}
}
```

Chained Put Methods

Chaining Exercises

- Create a class called Calc and contains the following methods:
  - performSum(List<Integer>)
  - displaySum()
    - You should be able to use this method as follow:
      - classObj.performSum(List<Integer>).displaySum();
    - Sample:
    - List<Integer> num = Arrays.asList(3, 4, 3);
    - classObj.performSum(num).displaySum();
    - Output: 10

Chained Put Methods

Chaining Exercises

- Add the following methods to the Calc class:
  - numberTOPerformOn(int numA, int numB)
  - operation(String operator)
  - Example Use
    - classObj.numberToPerformOn(2, 3).operator("+")
      - OutPut: 5
    - classObj.numberToPerformOn(2, 3).operator("*")
      - OutPut: 6

PER SCHOLAS