Florida Institute of Technology

# CIS 5230 Operating Systems – Program 1

|  |
| --- |
| Requirements: |
| Class File (source: <http://docs.oracle.com/javase/6/docs/api/java/io/File.html>)  An abstract representation of file and directory pathnames.  User interfaces and operating systems use system-dependent *pathname strings* to name files and directories. This class presents an abstract, system-independent view of hierarchical pathnames. An *abstract pathname* has two components:   1. An optional system-dependent *prefix* string, such as a disk-drive specifier, "/" for the UNIX root directory, or "\\\\" for a Microsoft Windows UNC pathname, and 2. A sequence of zero or more string *names*.   The first name in an abstract pathname may be a directory name or, in the case of Microsoft Windows UNC pathnames, a hostname. Each subsequent name in an abstract pathname denotes a directory; the last name may denote either a directory or a file. The *empty* abstract pathname has no prefix and an empty name sequence.  Use the Java File class to demonstrate the following Operating Systems functionalities:   1. createNewFile()- Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist. 2. delete() - Deletes the file or directory denoted by this abstract pathname. If this pathname denotes a directory, then the directory must be empty in order to be deleted. 3. length() - Returns the length of the file denoted by this abstract pathname. The return value is unspecified if this pathname denotes a directory. 4. mkdir() - Creates the directory named by this abstract pathname. 5. renameTo(File dest) - Renames the file denoted by this abstract pathname.   **Note:** There are lots of sample programming examples available through Internet; for that, your program must specifically perform the above requirements. Points will be deducted for including extra code (exception – code to test your program) |
| Syntax/Logic Errors |
| Description of each syntax/logic error:  Syntax errors are errors the compiler finds. They may be incorrect spelling, incorrect punctuation, or undeclared variable. In these tests, the syntax errors will be in the file abstract path names. They may be incorrect or null.  Logic errors are errors in the program that give incorrect output. Logic errors in these tests would be a result that does not match the expected result.  Runtime errors (Exceptions) are listed in the Testing Section.    Syntax/Logic Errors of the Operating System Functionalities   1. createNewFile()- Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.   Syntax/Logic Errors:   1. Create a file that does not exists : Logic Error if it fails 2. Create a file that already exists : Logic Error 3. Create a file with an invalid path : Syntax Error 4. Create file in a read-only directory : Logic Error 5. Create a null file : Syntax Error 6. delete() - Deletes the file or directory denoted by this abstract pathname. If this pathname denotes a directory, then the directory must be empty in order to be deleted.   Syntax/Logic Errors:   1. Delete a file that exists : Logic Error if it fails 2. Delete a file that does not exist : Syntax Error (invalid path) 3. Delete a file that is read only : Logic error 4. Delete an empty directory : Logic error if it fails 5. Delete a directory that is not empty : Logic Error 6. Delete a read only directory : Logic error 7. Delete a null file : Syntax Error 8. length() - Returns the length of the file denoted by this abstract pathname. The return value is unspecified if this pathname denotes a directory.   Syntax/Logic Errors:   1. Length of a file that exists : Logic Error if Incorrect result 2. Length of a directory that exists: Logic error if it fails 3. Length of a file that does not exist : Syntax Error 4. Length of a directory that does not exist : Syntax error 5. Length of a read only file 6. Length of a read only directory 7. Length of a null file : Syntax error 8. mkdir() - Creates the directory named by this abstract pathname.   Syntax/Logic Errors:   1. Create a directory with valid path : Logic Error if it fails 2. Create a directory with a path already exists : Logic Error 3. Create a directory inside a read only directory : Logic Error 4. Creating a directory with an invalid path : syntax error 5. Create a directory with a null path : Syntax Error 6. renameTo(File dest) - Renames the file denoted by this abstract pathname.   Syntax/Logic Errors:   1. Rename from a valid path to a valid path : Logic error if it fails 2. Rename from an invalid from path : Syntax Error 3. Rename a file from a valid path to an invalid path : Syntax Error 4. Rename a file that is read only : Logic Error 5. Rename a directory with valid paths : logic error if it fails 6. Rename a directory that does not exist : syntax error 7. Rename a file to a null path : Syntax Error 8. Rename a file from a null path : syntax error |
| Source Programs |
| /\*  Programmer: Bernice Templeman  Class: CIS 5230 Operating Systems  Project: Program-1  Date: Jan 30 2016  Use the Java File class to demonstrate the following Operating Systems functionalities:  (1) createNewFile()- Atomically creates a new, empty file named by this abstract pathname  if and only if a file with this name does not yet exist.  (2) delete() - Deletes the file or directory denoted by this abstract pathname.  If this pathname denotes a directory,  then the directory must be empty in order to be deleted.  (3) length() - Returns the length of the file denoted by this abstract pathname.  The return value is unspecified if this pathname denotes a directory.  (4) mkdir() - Creates the directory named by this abstract pathname.  (5) renameTo(File dest) - Renames the file denoted by this abstract pathname.  Testing  include code to catch all potential exceptions,  and explain the catch for each exception.  \*/  package program1;  //import java.nio.file.Files;  import java.io.File;  import java.io.IOException;  import java.nio.file.Files;  import java.nio.file.Path;  import java.nio.file.Paths;  import java.nio.file.attribute.AclEntry;  import java.nio.file.attribute.AclEntryPermission;  import java.nio.file.attribute.AclEntryType;  import java.nio.file.attribute.AclFileAttributeView;  import java.nio.file.attribute.UserPrincipal;  import java.nio.file.attribute.UserPrincipalLookupService;  import java.util.Collections;  import java.util.EnumSet;  import java.util.Scanner;  // Use the Java File class to demonstrate Operating System functionalities  public class Program1  {  /\*\*  \* @param args the command line arguments  \*/    public static final String WELCOME\_MESSAGE = "Hello, welcome to the OS Functionality Demonstration using the Java File class";  public static final String OS\_VERSION = "This demonstration is on a Windows 10 OS and Java 8.";  public static final String PREREQUISITES = "This demonstration requires a Directory where a testing directory and files can be created and deleted.";  public static final String SECTION\_DIVIDER = "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";  public static final String READ\_ONLY\_DIRECTORY\_NAME = "readOnlyDir";  public static final String READ\_ONLY\_FILE\_NAME = "readOnlyFile";  public static final String FILE\_1 = "file1.txt";  public static final String FILE\_2 = "file2.txt";  public static final String FILE\_3 = "file3.txt";  public static final String FILE\_4 = "file4.txt";  public static final String DIRECTORY\_1 = "dir1";  public static final String DIRECTORY\_2 = "dir2";  public static final String DIRECTORY\_3 = "dir3";  public static final String DIRECTORY\_4 = "dir4";    public static void main(String[] args) throws IOException  {  Scanner keyboard = new Scanner(System.in);  String runTestPath = "n";  String testDirPath = "";  String readOnlyDirPath = "";  String readOnlyFilePath = "";  String myReadOnlyFilePath = "";    String runTest = "n";  System.out.println(SECTION\_DIVIDER);  System.out.println(WELCOME\_MESSAGE);  System.out.println(OS\_VERSION + "\n");  System.out.println(SECTION\_DIVIDER);    System.out.println(PREREQUISITES);  System.out.println("Do you have a directory where direcories & files can be created & deleted?");  System.out.println("For example: Create a directory such as: C:\\users\\bernice\\javaDirectory");  System.out.printf("Are you ready to continue with the Demonstration? y n : ");  runTest = keyboard.nextLine();    boolean myReadOnlyFileExists = false;  if(runTest.equals("y"))  {  // Checking for readonly file  System.out.println("\nThis demonstration also requires a read-only file in the directory.");  System.out.println("For example, in Windows,create the file: C:\\users\\bernice\\javaDirectory\\myReadOnlyFile.txt");  System.out.println("After creating the file in Windows, change the Windows file permissions to read only.");  System.out.printf("Do you want to use your own readonly file locaton? y n : ");  runTest = keyboard.nextLine();  if(runTest.equals("y"))  {  myReadOnlyFileExists = true;  }    // creating directories  System.out.println("\nCreating Directories to test in...\n");  // create testing directory  testDirPath = createTestDirectory();    if(myReadOnlyFileExists)  {  myReadOnlyFilePath = askMyReadOnlyFilePath();    }    // create readonly directory  readOnlyDirPath = createReadOnlyDirectory(testDirPath);    // create readOnly file  readOnlyFilePath = createReadOnlyFile(testDirPath);    System.out.printf("A read only file was also created in the esting directory : ");  System.out.printf("Please change the permissions to read only and enter y when complete: y : ");  runTest = keyboard.nextLine();  while(!(runTest.equals("y")))  {  System.out.printf("A read only file was also created in the testing directory : ");  System.out.printf("Please change the permissions to read only and enter y when complete: y : ");  runTest = keyboard.nextLine();  }    System.out.println(SECTION\_DIVIDER);  System.out.println(SECTION\_DIVIDER);    // These are the path locations we will be using for this demonstration:  System.out.println("These are the path names we are using for this test:");  System.out.println(testDirPath);  System.out.println(myReadOnlyFilePath);  System.out.println(getReadOnlyDirPath(testDirPath));  System.out.println(getReadOnlyFilePath(testDirPath));  System.out.println(getFilePath1(testDirPath));  System.out.println(getFilePath2(testDirPath));  System.out.println(getFilePath3(testDirPath));  System.out.println(getFilePath4(testDirPath));  System.out.println(getDirPath1(testDirPath));  System.out.println(getDirPath2(testDirPath));  System.out.println(getDirPath3(testDirPath));  System.out.println(getDirPath4(testDirPath));    System.out.println("\nBegining OS functionality demonstration.");  System.out.println(SECTION\_DIVIDER);  System.out.println(SECTION\_DIVIDER);    // Test without Security Manager and without try/catch  System.out.println("\n\nDemonstrating OS functionalies without Try/Catch & without Security Manager");  System.out.println(SECTION\_DIVIDER);    demonstrateCreateNewFile(testDirPath);  System.out.println(SECTION\_DIVIDER);    if(myReadOnlyFileExists)  demonstrateDelete(testDirPath,myReadOnlyFilePath);  else  demonstrateDelete(testDirPath,getReadOnlyFilePath(testDirPath));  System.out.println(SECTION\_DIVIDER);    demonstrateLength(testDirPath);  System.out.println(SECTION\_DIVIDER);    demonstrateMkDir(testDirPath);  System.out.println(SECTION\_DIVIDER);    if(myReadOnlyFileExists)  demonstrateRenameTo(testDirPath, myReadOnlyFilePath);  else  demonstrateRenameTo(testDirPath, getReadOnlyFilePath(testDirPath));  System.out.println(SECTION\_DIVIDER);  System.out.println(SECTION\_DIVIDER);    System.out.printf("Pausing to delete the test directories and files before the next set of tests : ");  System.out.printf("Please check the directory first and enter y when complete: y : ");  runTest = keyboard.nextLine();  while(!(runTest.equals("y")))  {  System.out.printf("Pausing to delete the test directories and files before the next set of tests : ");  System.out.printf("Please check the directory first and enter y when complete: y : ");  runTest = keyboard.nextLine();  }  deleteTestDirectories(testDirPath);    // test without Security Manager with try/catch  System.out.println("\n\nDemonstrating OS functionalities with Try/Catch & without Security Manager");  System.out.println(SECTION\_DIVIDER);    demonstrateCreateNewFileTryCatch(testDirPath);  System.out.println(SECTION\_DIVIDER);    if(myReadOnlyFileExists)  demonstrateDeleteTryCatch(testDirPath,myReadOnlyFilePath);  else  demonstrateDeleteTryCatch(testDirPath,getReadOnlyFilePath(testDirPath));    System.out.println(SECTION\_DIVIDER);    demonstrateLengthTryCatch(testDirPath);  System.out.println(SECTION\_DIVIDER);    demonstrateMkDirTryCatch(testDirPath);  System.out.println(SECTION\_DIVIDER);    if(myReadOnlyFileExists)  demonstrateRenameToTryCatch(testDirPath, myReadOnlyFilePath);  else  demonstrateRenameToTryCatch(testDirPath, getReadOnlyFilePath(testDirPath));  System.out.println(SECTION\_DIVIDER);  System.out.println(SECTION\_DIVIDER);    System.out.printf("Pausing to delete the test directories and files before the next set of tests : ");  System.out.printf("Please check the directory first and enter y when complete: y : ");  runTest = keyboard.nextLine();  while(!(runTest.equals("y")))  {  System.out.printf("Pausing to delete the test directories and files before the next set of tests : ");  System.out.printf("Please check the directory first and enter y when complete: y : ");  runTest = keyboard.nextLine();  }  deleteTestDirectories(testDirPath);    // create a security manager  SecurityManager sm = new SecurityManager();  // set the system security manager  System.setSecurityManager(sm);    //test with Security Manager with try/catch  System.out.println("\n\nDemonstrating OS functionalities with Try/Catch & with Security Manager");  System.out.println(SECTION\_DIVIDER);    demonstrateCreateNewFileTryCatch(testDirPath);  System.out.println(SECTION\_DIVIDER);    if(myReadOnlyFileExists)  demonstrateDeleteTryCatch(testDirPath, myReadOnlyFilePath);  else  demonstrateDeleteTryCatch(testDirPath, getReadOnlyFilePath(testDirPath));  System.out.println(SECTION\_DIVIDER);    demonstrateLengthTryCatch(testDirPath);  System.out.println(SECTION\_DIVIDER);    demonstrateMkDirTryCatch(testDirPath);  System.out.println(SECTION\_DIVIDER);    if(myReadOnlyFileExists)  demonstrateRenameToTryCatch(testDirPath, myReadOnlyFilePath);  else  demonstrateRenameToTryCatch(testDirPath, getReadOnlyFilePath(testDirPath));    System.out.println(SECTION\_DIVIDER);  System.out.println(SECTION\_DIVIDER);    //Test with Security Manager without try/catch  System.out.println("\n\nDemonstrating OS functionalities without Try/Catch & with Security Manager");  System.out.println(SECTION\_DIVIDER);  System.out.printf("\nCreate new file with SM: The next test causes a Security Exception\n "  + "Do you want to continue? y n : ");  runTest = keyboard.nextLine();  if(runTest.equals("y"))  {  demonstrateCreateNewFile(testDirPath);  // print a message if we passed the check  System.out.println("Allowed!");  }  else  System.out.println("Test did not run to demonstrate createNewFile with Security Manager without try/catch");    System.out.println(SECTION\_DIVIDER);    System.out.printf("\nDelete file with SM: The next test causes a Security Exception\n "  + "Do you want to continue? y n : ");  runTest = keyboard.nextLine();  if(runTest.equals("y"))  {  if(myReadOnlyFileExists)  demonstrateDelete(testDirPath,myReadOnlyFilePath);  else  demonstrateDelete(testDirPath,getReadOnlyFilePath(testDirPath));    // print a message if we passed the check  System.out.println("Allowed!");  }  else  System.out.println("Test did not run to demonstrate delete with Security Manager without try/catch");    System.out.println(SECTION\_DIVIDER);    System.out.printf("\nLength of file with SM: The next test causes a Security Exception\n "  + "Do you want to continue? y n : ");  runTest = keyboard.nextLine();  if(runTest.equals("y"))  {  demonstrateLength(testDirPath);  // print a message if we passed the check  System.out.println("Allowed!");  }  else  System.out.println("Test did not run to demonstrate length with Security Manager without try/catch");    System.out.println(SECTION\_DIVIDER);      System.out.printf("\nmkdir with SM: The next test causes a Security Exception\n "  + "Do you want to continue? y n : ");  runTest = keyboard.nextLine();  if(runTest.equals("y"))  {  demonstrateMkDir(testDirPath);  // print a message if we passed the check  System.out.println("Allowed!");  }  else  System.out.println("Test did not run to demonstrate mkdir with Security Manager without try/catch");  System.out.println(SECTION\_DIVIDER);      System.out.printf("\nrenameTo with SM: The next test causes a Security Exception\n "  + "Do you want to continue? y n");  runTest = keyboard.nextLine();  if(runTest.equals("y"))  {  if(myReadOnlyFileExists)  demonstrateRenameTo(testDirPath, myReadOnlyFilePath);  else  demonstrateRenameTo(testDirPath, getReadOnlyFilePath(testDirPath));  // print a message if we passed the check  System.out.println("Allowed!");  }  else  System.out.println("Test did not run to demonstrate reNameTo with Security Manager without try/catch");    System.out.println(SECTION\_DIVIDER);  System.out.println(SECTION\_DIVIDER);      System.out.println("\n\nTesting Completed.");    sm = null;    System.out.printf("\nDo you want to delete the testing directories and files? "  + " y n ");  runTest = keyboard.nextLine();  if(runTest.equals("y"))  {  // delete the directories  if (deleteTestDirectories(testDirPath))  System.out.println("Success deleting test directories & files");  else  System.out.println("Failed deleting test directories & files");  }  else  System.out.println("Test Directories & files were not deleted");  }//end if runTest  }//end main    //create test directories  public static String createTestDirectory()  {  Scanner keyboard = new Scanner(System.in);  boolean result = false;    //Get location to create testing directory  System.out.println(" Where do you want to create your testing directory?"  + " For Example: C:\\users\\bernice\\javaDirectory\\testDirectory "  + "\nEnter your directory path to create a new directory: ");  String dirPath = keyboard.nextLine();  File testDir = osFile(dirPath);  result = osMkDir(testDir);  while(!result)  {  System.out.println("Invalid path, test directory was not created");  System.out.println(" Where do you want to create your testing directory?"  + " For Example: C:\\users\\bernice\\javaDirectory\\testDirectory "  + "\nEnter your directory path to create a new directory: ");  dirPath = keyboard.nextLine();  testDir = osFile(dirPath);  result = osMkDir(testDir);  }    System.out.println("Your Testing Directory is: " + testDir.getPath());  if (testDir.canWrite())  {  System.out.println(testDir.getAbsolutePath() + "Can Write: ");  } else  {  System.out.println(testDir.getAbsolutePath() + " Cannot Write: ");  }  return dirPath;  }//end createTestDirectory    public static String askMyReadOnlyFilePath()  {  Scanner keyboard = new Scanner(System.in);  boolean result = false;    System.out.println("Example of an existing read-only file path: C:\\users\\bernice\\javaDirectory\\myReadOnlyFile.txt");  System.out.println("Enter your path: ");  String inputString = keyboard.nextLine();  File file = osFile(inputString);  result = osCreateNewFile(file);  result = file.exists();  while(!result)  {  System.out.println("Invalid file path.");  System.out.println("Example of an existing read-only file path: C:\\users\\bernice\\javaDirectory\\myReadOnlyFile.txt");  System.out.println("Enter your path: ");  inputString = keyboard.nextLine();  file = osFile(inputString);  result = osCreateNewFile(file);  result = file.exists();  }  return inputString;  }//askMyReadOnlyFilePath()    public static String createReadOnlyDirectory(String dirPath)  {  boolean result = false;  String readOnlyPath = dirPath + "\\"+ READ\_ONLY\_DIRECTORY\_NAME;    try {  File readOnlyDir = osFile(readOnlyPath);  result = osMkDir(readOnlyDir);  Path filepath = Paths.get(readOnlyPath);    AclFileAttributeView aclAttr = Files.getFileAttributeView(filepath, AclFileAttributeView.class);  System.out.println(aclAttr.getOwner());  for(AclEntry aclEntry : aclAttr.getAcl()){  System.out.println(aclEntry);  }  System.out.println();    UserPrincipalLookupService upls = filepath.getFileSystem().getUserPrincipalLookupService();  UserPrincipal user = upls.lookupPrincipalByName(System.getProperty("user.name"));  AclEntry.Builder builder = AclEntry.newBuilder();  builder.setPermissions( EnumSet.of(AclEntryPermission.READ\_DATA, AclEntryPermission.EXECUTE,  AclEntryPermission.READ\_ACL, AclEntryPermission.READ\_ATTRIBUTES, AclEntryPermission.READ\_NAMED\_ATTRS,  AclEntryPermission.WRITE\_ACL, AclEntryPermission.DELETE  ));  builder.setPrincipal(user);  builder.setType(AclEntryType.ALLOW);  aclAttr.setAcl(Collections.singletonList(builder.build()));  }  catch (IOException ex)  {  System.out.println("IO Exception: " + ex);  }  return readOnlyPath;  }//end createReadOnlyDirectory    public static String createReadOnlyFile(String testDirPath)  {  String readOnlyFileString = testDirPath + "\\"+READ\_ONLY\_FILE\_NAME ;  boolean result = false;    try  {  File file = new File(readOnlyFileString);  result = osCreateNewFile(file);  Path filepath = Paths.get(readOnlyFileString);  AclFileAttributeView aclAttr = Files.getFileAttributeView(filepath, AclFileAttributeView.class);  System.out.println(aclAttr.getOwner());  for(AclEntry aclEntry : aclAttr.getAcl()){  System.out.println(aclEntry);  }  System.out.println();    UserPrincipalLookupService upls = filepath.getFileSystem().getUserPrincipalLookupService();  UserPrincipal user = upls.lookupPrincipalByName(System.getProperty("user.name"));  AclEntry.Builder builder = AclEntry.newBuilder();  /\*builder.setPermissions( EnumSet.of(AclEntryPermission.READ\_DATA, AclEntryPermission.EXECUTE,  AclEntryPermission.READ\_ACL, AclEntryPermission.READ\_ATTRIBUTES, AclEntryPermission.READ\_NAMED\_ATTRS,  AclEntryPermission.WRITE\_ACL, AclEntryPermission.DELETE  ));  \*/  builder.setPermissions( EnumSet.of(AclEntryPermission.READ\_DATA,  AclEntryPermission.READ\_ACL, AclEntryPermission.READ\_ATTRIBUTES, AclEntryPermission.READ\_NAMED\_ATTRS  ));  builder.setPrincipal(user);  builder.setType(AclEntryType.ALLOW);  aclAttr.setAcl(Collections.singletonList(builder.build()));  }  catch (IOException ex)  {  System.out.println("IO Exception: " + ex);  }  return readOnlyFileString;  }//end createReadOnlyFile    public static String getFilePath1( String testDirectoryPath )  {  return testDirectoryPath + "\\" + FILE\_1;  }  public static String getFilePath2( String testDirectoryPath )  {  return testDirectoryPath + "\\" + FILE\_2;  }    public static String getFilePath3( String testDirectoryPath )  {  return testDirectoryPath + "\\" + FILE\_3;  }    public static String getFilePath4( String testDirectoryPath )  {  return testDirectoryPath + "\\" + FILE\_4;  }    public static String getDirPath1( String testDirectoryPath )  {  return testDirectoryPath + "\\" + DIRECTORY\_1;  }    public static String getDirPath2( String testDirectoryPath )  {  return testDirectoryPath + "\\" + DIRECTORY\_2;  }    public static String getDirPath3( String testDirectoryPath )  {  return testDirectoryPath + "\\" + DIRECTORY\_3;  }    public static String getDirPath4( String testDirectoryPath )  {  return testDirectoryPath + "\\" + DIRECTORY\_4;  }    public static String getReadOnlyDirPath( String testDirectoryPath )  {  return testDirectoryPath + "\\" + READ\_ONLY\_DIRECTORY\_NAME;  }    public static String getReadOnlyFilePath( String testDirectoryPath )  {  return testDirectoryPath + "\\" + READ\_ONLY\_FILE\_NAME;  }    // delete test directories and files  public static boolean deleteTestDirectories(String testDirPath)  {  boolean result = false;  boolean deleteDirSuccess = false;    String filename1 = getFilePath1(testDirPath);  String filename2 = getFilePath2(testDirPath);  String filename3 = getFilePath3(testDirPath);  String filename4 = getFilePath4(testDirPath);  String dirname1 = getDirPath1(testDirPath);  String dirname2 = getDirPath2(testDirPath);  String dirname3 = getDirPath3(testDirPath);  String dirname4 = getDirPath4(testDirPath);  String readOnlyFilePath = getReadOnlyFilePath(testDirPath);  String readOnlyDirPath = getReadOnlyDirPath(testDirPath);  String readOnlyDirPath2 = getReadOnlyDirPath(testDirPath)+"\\"+"testdir";  String invalidDirPath = testDirPath+FILE\_1+"\\"+FILE\_2;    File file1 = osFile(filename1);  File file2 = osFile(filename2);  File file3 = osFile(filename3);  File file4 = osFile(filename4);    File dir1 = osFile(dirname1);  File dir2 = osFile(dirname2);  File dir3 = osFile(dirname3);  File dir4 = osFile(dirname4);    File readOnlyFile = osFile(readOnlyFilePath);  File readOnlyDir = osFile(readOnlyDirPath);    osDelete(file1);  osDelete(file2);  osDelete(file3);  osDelete(file4);  osDelete(dir1);  osDelete(dir2);  osDelete(dir3);  osDelete(dir4);  osDelete(readOnlyFile);  osDelete(readOnlyDir);    return deleteDirSuccess;  }//deleteTestDirectories    /\* Testing with the File constructor with a try/catch  File(String pathname)  Creates a new File instance by converting the given pathname string into an abstract pathname.  \*/  // osFile includes a try/catch for a NullPointer exception  public static File osFile( String pathname)  {  File f1 = null;  try  {  //throws Null Pointer  f1 = new File(pathname);  }  catch( NullPointerException ex)  {  System.out.println(Program1.class.getName() + ex +  " in osFile: NullPointerError");  }  return f1;  }//end osFile    //(1) osCreateNewFile() includes try/catch code  // - Atomically creates a new,  // empty file named by this abstract pathname  // if and only if a file with this name does not yet exist.  //Returns: true if the named file does not exist and was successfully created;  // false if the named file already exists  // throws IOException  // can throw a Security Exception if security manager exists  // and its SecurityManager.checkRead(java.lang.String) method denies read access to the file  public static boolean osCreateNewFile(File f1)  {  boolean result = false;    try  {  // throws IOException | SecurityException  result = f1.createNewFile();    if(result)  System.out.println("CreateNewFile Success : " + f1.getName());  else  System.out.println("CreateNewFile Error: " + f1.getName());  }  catch (IOException ex)  {  System.out.println(Program1.class.getName() + ex + ": " + f1.getName() );  //System.out.println( " osCreateNewFile: IO Error: " + f1.getName());  }  catch ( SecurityException se)  {  System.out.println(Program1.class.getName() + se);  System.out.println(" osCreateNewFile: SE Error: " + f1.getName());  }  catch (NullPointerException np )  {  System.out.println(Program1.class.getName() + np);  System.out.println("NullPointer Error");  }  return result;  }//end osCreateNewFile    //(2) osDelete includes try/catch code  // Deletes the file or directory denoted by this abstract pathname.  // If this pathname denotes a directory,  // then the directory must be empty in order to be deleted.  // Returns: true if and only if the file or directory is successfully deleted;  // false otherwise  // delete throws SecurityException  public static boolean osDelete(File f2)  {  boolean result = false;    try  {  result = f2.delete();  if (result)  System.out.println("Delete success: " + f2.getName() );  else  System.out.println("Delete failed: " + f2.getName());  }//end try  catch(SecurityException se)  {  System.out.println(Program1.class.getName() + se);  System.out.println(" osDelete Security Exception Error" + f2.getName());  }  catch(NullPointerException np)  {  System.out.println(Program1.class.getName() + np);  System.out.println(" osDelete NullPointer Error");  }  return result;  }//end osDelete    //(3) osLength() - inclludes try/catch code  // Returns the length of the file denoted by this abstract pathname.  // The return value is unspecified if this pathname denotes a directory.  // length throws SecurityException  public static long osLength(File file)  {  long fileLength = -1;    try  {  fileLength = file.length();  }  catch( SecurityException ex)  {  System.out.println(Program1.class.getName() + ex);  System.out.println(" SecurityException: " + file.getName());  }  catch (NullPointerException np )  {  System.out.println(Program1.class.getName() + np);  System.out.println("NullPointer Error");  }  return fileLength;  }//osLength    //(4) osMkDir includes try catch code  // mkdir() - Creates the directory named by this abstract pathname.  // Returns: true if and only if the directory was created;  // false otherwise  // throws SecurityException  public static boolean osMkDir(File dir)  {  boolean result = false;    try  {  // throws SecurityException  result = dir.mkdir();    if (result)  {  System.out.println("mkDir Success: " + dir.getName());    }  else  {  System.out.println("mkdir fail: " + dir.getName());  }  }  catch(SecurityException ex)  {  System.out.println(Program1.class.getName() + ex);  System.out.println("dir.mkdir failed: Security execption");  }  catch (NullPointerException np )  {  System.out.println(Program1.class.getName() + np);  System.out.println("NullPointer Error");  }    return result;  }//end osMkDir    //(5) osRenameTo(File dest) - includes tr/catch code  //Renames the file denoted by this abstract pathname.  // Returns: true if and only if the renaming succeeded;  // false otherwise  // throws SecurityException | NullPointerException - If parameter dest is null  public static boolean osRenameTo(File d1, File d2)  {  boolean result = false;    try{  result = d1.renameTo(d2);  if (result)  {  System.out.println("Success renaming directory " + d1.getName() + " to " + d2.getName());  }  else  {  System.out.println("Error renaming directory " + d1.getName() + " to " + d2.getName());  }  }  catch (SecurityException ex )  {  System.out.println(Program1.class.getName() + ex);  System.out.println("SecurityException Error");  }  catch (NullPointerException np )  {  System.out.println(Program1.class.getName() + np);  System.out.println("NullPointer Error");  }  return result;  }//osRenameTo    // demonstrate createNewFile()- Atomically creates a new,  // empty file named by this abstract pathname  // if and only if a file with this name does not yet exist.  //Returns: true if the named file does not exist and was successfully created;  // false if the named file already exists  // throws IOException | SecurityException  public static void demonstrateCreateNewFile(String testDirPath) throws IOException  {  Scanner keyboard = new Scanner(System.in);  boolean result = false;    System.out.println("\nDemonstrate createNewFile())\n");    // added throws clause to method name & calling method name    //\*\*\*1. test createNewFile without try/catch - success - file dne  System.out.println("createNewFile Test 1: Creating a new file that does not exist");  String filename = getFilePath1(testDirPath);  File f1 = new File(filename);  //result = f1.delete(); //delete it if it exists    result = f1.createNewFile();  if(result)  System.out.println("createNewFile Test 1: Success " + filename + " created");  else  System.out.println("createNewFile Test 1: Failed " + filename + " not created");  System.out.println();    //\*\*\*2. test createNewFile without try/catch - fail - file exists  System.out.println("createNewFile Test 2: Creating a new file that exists");  if (result) //only do test if file exists ( test 1 was successful)  {  result = f1.createNewFile();  if(result)  System.out.println("createNewFile Test: Failed " + filename + " created");  else  System.out.println("createNewFile Test 2: Success " + filename  + " not created ");  }  else  System.out.println("createNewFile Test 2. Testing error. Test 1 failed and test 2 did not run.");  System.out.println();    //\*\*\*3. test createNewFile without try/catch - ioexception  System.out.println("createNewFile Test 3: Creating a new file with an invalid path");  System.out.printf("The next test throw an exception & cause the program to halt. "  + "Do you want to continue? y n : ");  String runTest = keyboard.nextLine();  if(runTest.equals("y"))  {  System.out.println("Test creating a file with an invalid path: ");  String filename2 = getFilePath2(testDirPath)+"\\A\\"+getFilePath2(testDirPath);  File f2 = new File(filename2);  result = f2.createNewFile();  if(result)  System.out.println("createNewFile Test 3: Failed " + filename2 + " created" );  else  System.out.println("createNewFile Test 3: Success " + filename2 + " not created");  }  else  System.out.println("createNewFile Test 3: did not run");  System.out.println();    //\*\*\*4. test createNewFile without try/catch - ioException  System.out.println("createNewFile Test 4: Creating a new file in a read only directory");  System.out.printf("The next test throw an exception & cause the program to halt. "  + "Do you want to continue? y n : ");  runTest = keyboard.nextLine();  if(runTest.equals("y"))  {  System.out.println("Test creating a file in a read only directory: ");  String filename3 = getReadOnlyDirPath(testDirPath)+"\\"+ READ\_ONLY\_FILE\_NAME;  File f3 = new File(filename3);  result = f3.createNewFile();  if(result)  System.out.println("createNewFile Test 4: Failed " + filename3 + " created" );  else  System.out.println("createNewFile Test 4: Success " + filename3 + " not created");  }  else  System.out.println("createNewFile Test 4: did not run");    System.out.println();    //\*\*\*4. test createNewFile without try/catch - ioException  System.out.println("createNewFile Test 5: Creating a null file");  System.out.printf("The next test throw an exception & cause the program to halt. "  + "Do you want to continue? y n : ");  runTest = keyboard.nextLine();  if(runTest.equals("y"))  {  System.out.println("Test creating a null file: ");  File nullfile = null;  result = nullfile.createNewFile();  if(result)  System.out.println("createNewFile Test 5: Failed nullfile created" );  else  System.out.println("createNewFile Test 5: Success nullfile not created");  }  else  System.out.println("createNewFile Test 5: did not run");  System.out.println();  }// end demonstrate create new file without try catch    public static void demonstrateCreateNewFileTryCatch(String testDirPath)  {  Scanner keyboard = new Scanner(System.in);  boolean result = false;    System.out.println("\nDemonstrate createNewFile() with try catch\n");    //\*\*\*1. test createNewFile with try/catch -success  System.out.println("createNewFile Test 1: Creating a new file that does not exist");  String filename2 = getFilePath2(testDirPath);  File f2 = osFile(filename2);  osDelete(f2);    // test creating a file that does not exist    result = osCreateNewFile(f2);  if (result)  System.out.println("createNewFile Test 1: Success: file created: " + f2.getName() + " created");  else  System.out.println("createNewFile Test 1: Failed: did not create " + f2.getName() + " not created");  System.out.println();    //\*\*\*2. test createNewFile with try/catch - fail - file exists  System.out.println("createNewFile Test 2: Creating a new file that exists");  if (osCreateNewFile(f2))  System.out.println("createNewFile Test 6: Failed " + f2.getName() + " created");  else  System.out.println("createNewFile Test 6: Success " + f2.getName() + " not created");  System.out.println();    //\*\*\*3. test createNewFile with try/catch - ioexception  // test creating a file with invalid path (IO Exception)  System.out.println("createNewFile Test 3: Creating a new file with an invalid path");  System.out.println("Test creating a file with an invalid path: ");  String filename3 = getFilePath2(testDirPath)+"\\A\\"+getFilePath2(testDirPath);  File f3 = osFile(filename3);  if (osCreateNewFile(f3))  System.out.println("createNewFile Test 3: Failed " + f3.getName() + " created");  else  System.out.println("createNewFile Test 3: Success " + f3.getName() + " not created");  System.out.println();    //\*\*\*8. test createNewFile with try/catch - ioException  // test Security Exception - read only directory  System.out.println("createNewFile Test 4: Creating a new file in a read only directory");  System.out.println("Test creating a file in a read only directory: ");  String filename4 = getReadOnlyDirPath(testDirPath)+"\\"+ READ\_ONLY\_FILE\_NAME;  File f4 = new File(filename4);  result = osCreateNewFile(f4);  if(result)  System.out.println("createNewFile Test 4: Failed " + f4.getName() + " created");  else  System.out.println("createNewFile Test 4: Success " + f4.getName() + " not created");  System.out.println();    //System.out.println("createNewFile Test 5: Creating a null file");  //System.out.printf("The next test throw an exception & cause the program to halt. "  // + "Do you want to continue? y n : ");  //runTest = keyboard.nextLine();  //if(runTest.equals("y")){  System.out.println("Test creating a null file: ");  File nullfile = null;  result = osCreateNewFile(nullfile);  if(result)  System.out.println("createNewFile Test 5: Failed nullfile created" );  else  System.out.println("createNewFile Test 5: Success nullfile not created");  // }else  // System.out.println("createNewFile Test 5: did not run");  }//end create new file with try catch      //(2) Demonstrate delete() - Deletes the file or directory denoted by this abstract pathname.  // If this pathname denotes a directory,  // then the directory must be empty in order to be deleted.  // Returns: true if and only if the file or directory is successfully deleted;  // false otherwise  // delete throws SecurityException  public static void demonstrateDelete(String testDirPath, String readOnlyFilePath) throws IOException, SecurityException  {  Scanner keyboard = new Scanner(System.in);  boolean result = false;  String runTest = "n";    System.out.println("\nDemonstrate delete() without try/catch");  System.out.println("It throws IOException, SecurityException\n");    //\*\*\*1. Test delete without try/catch : file exists  System.out.println("Delete Test 1: Deleting a file that exists");  String filename1 = getFilePath1(testDirPath);  File f1 = new File(filename1);    if(f1.exists())  {  //delete the file we created  result = f1.delete();  if (result)  System.out.println("Delete Test 1: Success " + f1.getName() + " was deleted");  else  System.out.println("Delete Test 1: Failed " + f1.getName() + " was not deleted");  }  else  System.out.println("Delete Test 1: incomplete because file was not created");    System.out.println();    //\*\*\*2 Test delete without try/catch : file does not exist  System.out.println("Delete Test 2: Deleting a file that does not exists");  result = f1.delete();  if (result)  System.out.println("Delete Test 2: Failed " + f1.getName() + " was deleted");  else  System.out.println("Delete Test 2: Success " + f1.getName() + " was not deleted");    System.out.println();    //test3 : delete read only file  System.out.println("Delete Test 3: Deleting a file that is read only");  File readOnlyFile = new File("C:\\users\\bernice\\javaDirectory\\myReadOnlyFile.txt");  readOnlyFile.createNewFile();  result = readOnlyFile.delete();  if (result)  System.out.println("Delete Test 3: Failed " + readOnlyFile.getName() + " was deleted");  else  System.out.println("Delete Test 3: Success " + readOnlyFile.getName() + " was not deleted");  //System.out.println("Delete Test 3: Did not run");  System.out.println();    //\*\*\*4 Test delete without try/catch : delete empty directory  System.out.println("Delete Test 4: Deleting an empty directory");  String dirname1 = getDirPath1(testDirPath);  File f4 = new File(dirname1);  result = f4.mkdir();    result = f4.delete(); //delete empty directory  if (result)  System.out.println("Delete Test 4: Success " + f4.getName() + " was deleted");  else  System.out.println("Delete Test 4: Failed " + f4.getName() + " was not deleted");  System.out.println();    //\*\*\*5 Test delete without try/catch : delete directory with files  System.out.println("Delete Test 5: Deleting a directory with files");  String filename5 = testDirPath;  File f5 = new File(filename5);  result = f5.delete();  if (result)  System.out.println("Delete Test 5: Failed " + f5.getName() + " was deleted");  else  System.out.println("Delete Test 5: Success " + f5.getName() + " was not deleted");  System.out.println();    //\*\*\*6 Test delete without try/catch : delete a read only directory  System.out.println("Delete Test 6: Deleting a read only directory");  String filename6 = getReadOnlyDirPath(testDirPath );  File f6 = new File(filename6);  result = f6.delete();  if (result)  System.out.println("Delete Test 6: Failed " + f6.getName() + " was deleted");  else  System.out.println("Delete Test 6: Success " + f6.getName() + " was not deleted");  System.out.println();    //\*\*\*7 Test delete without try/catch : File is null  System.out.println("Delete Test 7: Deleting a null file");  System.out.printf("The next test throws an exception & causes the program to halt. "  + "Do you want to continue? y n : ");  runTest = keyboard.nextLine();  if(runTest.equals("y"))  {  File f7 = null;  result = f7.delete();  if (result)  System.out.println("Delete Test 7: Failed nullfile was deleted");  else  System.out.println("Delete Test 7: Success nullfile was not deleted");  }  else  System.out.println("Delete Test 7: did not run.");  System.out.println();  }//end demonstrateDelete    public static void demonstrateDeleteTryCatch(String testDirPath, String myReadOnlyFilePath)  {  Scanner keyboard = new Scanner(System.in);  boolean result = false;  String runTest = "n";    System.out.println("\nDemonstrate delete with try/catch");    // create a file to delete  String filename1 = getFilePath1(testDirPath);  File f8 = osFile(filename1);  osCreateNewFile(f8);    ///\*\*\*1. Test delete with try/catch : file exists  System.out.println("Delete Test 1: Deleting a file that exists");  result = osDelete(f8);  if (result)  System.out.println("Delete Test 1: Success " + f8.getName() + " was deleted");  else  System.out.println("Delete Test 1: Failed " + f8.getName() + " was not deleted");  System.out.println();    //\*\*\*2 Test delete with try/catch : file does not exist  System.out.println("Delete Test 2: Deleting a file that does not exist");  result = osDelete(f8);  if (result)  System.out.println("Delete Test 2: Failed " + f8.getName() + " was deleted");  else  System.out.println("Delete Test 2: Success " + f8.getName() + " was not deleted" );  System.out.println();    //\*\*\*3 Test delete with try/catch : file is read only  System.out.println("Delete Test 3: Deleting a file that is readonly");  //File readOnlyFile = osFile("C:\\users\\bernice\\javaDirectory\\myReadOnlyFile.txt");  File readOnlyFile = osFile(myReadOnlyFilePath);  osCreateNewFile(readOnlyFile);  result = osDelete(readOnlyFile);  if (result)  System.out.println("Delete Test 3: Failed " + readOnlyFile.getName() + " was deleted");  else  System.out.println("Delete Test 3: Success " + readOnlyFile.getName() + " was not deleted");  //System.out.println("Delete Test 3: Did not run");  System.out.println();    //\*\*\*4 Test delete with try/catch : delete empty directory  // create directory  String dirname1 = getDirPath1(testDirPath);  File f4 = new File(dirname1);  osMkDir(f4);    System.out.println("Delete Test 4: Deleting an empty directory");  result = osDelete(f4); //delete empty directory  if (result)  System.out.println("Delete Test 4: Success " + f4.getName() + " was deleted");  else  System.out.println("Delete Test 4: Failed " + f4.getName() + " was not deleted");  System.out.println();    //\*\*\*5 Test delete with try/catch : delete directory with files  System.out.println("Delete Test 5: Deleting a directory with files");  File f12 = osFile(testDirPath);  result = osDelete(f12);  if (result)  System.out.println("Delete Test 5: Failed " + f12.getName() + " was deleted");  else  System.out.println("Delete Test 5: Success " + f12.getName() + " was not deleted");  System.out.println();    //\*\*\*6 Test delete with try/catch : delete a read only directory  System.out.println("Delete Test 6: Deleting a read only directory");  String filename13 = getReadOnlyDirPath(testDirPath);  File f13 = new File(filename13);  result = osDelete(f13);  if (result)  System.out.println("Delete Test 6: Failed " + f13.getName() + " was deleted");  else  System.out.println("Delete Test 6: Success " + f13.getName() + " was not deleted");  System.out.println();    //\*\*\*7 Test delete with try/catch : File is null  System.out.println("Delete Test 7: Deleting a null file");  File f14 = null;  result = osDelete(f14);  if (result)  System.out.println("Delete Test 7: Failed " );  else  System.out.println("Delete Test 7: Success ");  System.out.println();    System.out.println("\nDelete Test Complete\n");  }//end demonstrateDeleteTryCatch    //Demonstrate (3) length() - Returns the length of the file denoted by this abstract pathname.  // The return value is unspecified if this pathname denotes a directory.  //  //Throws: SecurityException - If a security manager exists and  //its SecurityManager.checkRead(java.lang.String) method denies read access to the file  public static void demonstrateLength(String testDirPath)  {  System.out.println("\nDemonstrate length() without try/catch\n");  long fileLength = 0;  Scanner keyboard = new Scanner(System.in);    String filename1 = getFilePath1(testDirPath);  String filename2 = getFilePath2(testDirPath);  String filename3 = getFilePath3(testDirPath);  String filename4 = getFilePath4(testDirPath);  String dirname1 = getDirPath1(testDirPath);  String dirname2 = getDirPath2(testDirPath);  String dirname3 = getDirPath3(testDirPath);  String dirname4 = getDirPath4(testDirPath);  String readOnlyFilePath = getReadOnlyFilePath(testDirPath);  String readOnlyDirPath = getReadOnlyDirPath(testDirPath);  String readOnlyDirPath2 = getReadOnlyDirPath(testDirPath)+"\\"+"testdir";  String invalidDirPath = testDirPath+FILE\_1+"\\"+FILE\_2;    File f1 = osFile(filename1);  File file2 = osFile(filename2);  File f3 = osFile(filename3);  File file4 = osFile(filename4);    File d1 = osFile(dirname1);  File dir2 = osFile(dirname2);  File dir3 = osFile(dirname3);  File dir4 = osFile(dirname4);    File readOnlyFile = osFile(readOnlyFilePath);  File readOnlyDir = osFile(readOnlyDirPath);  //test length without try/catch : valid file path  System.out.println("Length Test 1: Length of a valid file");  osCreateNewFile(f1);  fileLength = f1.length();  System.out.println("Length Test 1: "+ f1.getName()+": " + fileLength);  System.out.println();    //test length without try/catch : valid directory path  System.out.println("Length Test 2: Length of a valid directory");  osMkDir(d1);  fileLength = d1.length();  System.out.println("Length Test 2: "+ d1.getName()+": " + fileLength);  System.out.println();    //test length without try/catch : file invalid path  System.out.println("Length Test 3: Length of an invalid file");  String filename3x = testDirPath+"\\CInvalidpath\\"+testDirPath;  File file3x = new File(filename3x);  osCreateNewFile(file3x);  fileLength = file3x.length();  System.out.println("Length Test 3: "+ file3x.getName()+": " + fileLength);  System.out.println();    //test length without try/catch : directory invalid path  System.out.println("Length Test 4: Length of an invalid directory");  File file4x = new File(testDirPath+"\\a\\"+ READ\_ONLY\_DIRECTORY\_NAME);  osMkDir(file4x);  fileLength = file4x.length();  System.out.println("Length Test 4: "+ file4x.getName()+": " + fileLength);  System.out.println();    //test length without try/catch : file readonly  System.out.println("Length Test 5: Length of a read only valid file");  osCreateNewFile(readOnlyFile);  fileLength = readOnlyFile.length();  System.out.println("Length Test 5: "+ readOnlyFile.getName()+": " + fileLength);  System.out.println();    //test length without try/catch : directory readonly  System.out.println("Length Test 6: Length of a read only valid directory");  osMkDir(readOnlyDir);  fileLength = readOnlyDir.length();  System.out.println("Length Test 6: "+ readOnlyDir.getName()+": " + fileLength);  System.out.println();    //test length without try/catch : nullfile  System.out.println("Length Test 7: Length of a null file");  System.out.printf("The next test throws an exception & causes the program to halt. "  + "Do you want to continue? y n : ");  String runTest = keyboard.nextLine();  if(runTest.equals("y"))  {  File nullFile = null;  fileLength = nullFile.length();  System.out.println("Length Test 7: nullFile : " + fileLength);  System.out.println();  }  else  System.out.println("Length Test 7: Did not run");  }//end demonstrateLength    public static void demonstrateLengthTryCatch(String testDirPath)  {  boolean result = false;  long fileLength = 0;    String filename1 = getFilePath1(testDirPath);  String filename2 = getFilePath2(testDirPath);  String filename3 = getFilePath3(testDirPath);  String filename4 = getFilePath4(testDirPath);  String dirname1 = getDirPath1(testDirPath);  String dirname2 = getDirPath2(testDirPath);  String dirname3 = getDirPath3(testDirPath);  String dirname4 = getDirPath4(testDirPath);  String readOnlyFilePath = getReadOnlyFilePath(testDirPath);  String readOnlyDirPath = getReadOnlyDirPath(testDirPath);  String readOnlyDirPath2 = getReadOnlyDirPath(testDirPath)+"\\"+"testdir";  String invalidDirPath = testDirPath+FILE\_1+"\\"+FILE\_2;    File f1 = osFile(filename1);  File file2 = osFile(filename2);  File f3 = osFile(filename3);  File file4 = osFile(filename4);    File dir1 = osFile(dirname1);  File dir2 = osFile(dirname2);  File dir3 = osFile(dirname3);  File dir4 = osFile(dirname4);    File readOnlyFile = osFile(readOnlyFilePath);  File readOnlyDir = osFile(readOnlyDirPath);    System.out.println("\nDemonstrate length() with try/catch\n");    //test length with try/catch : valid path  System.out.println("Length Test 1: Length of a valid file");  File file1 = osFile(filename1);  result = osCreateNewFile(file1);  fileLength = osLength(file1);  System.out.println("Length Test 1: "+ file1.getName()+": " + fileLength);  System.out.println();    //test length with try/catch : valid path  System.out.println("Length Test 2: Length of a valid directory");  result = osMkDir(dir1);  fileLength = osLength(dir1);  System.out.println("Length Test 2: "+ dir1.getName()+": " + fileLength);  System.out.println();    //test length with try/catch : file invalid path  System.out.println("Length Test 3: Length of an invalid file");  String filename3x = testDirPath+"CInvalidpath"+testDirPath;  File file3x = new File(filename3x);  osCreateNewFile(file3x);  fileLength = osLength(file3x);  System.out.println("Length Test 3: "+ file3x.getName()+": " + fileLength);  System.out.println();    //test length with try/catch : directory invalid path  System.out.println("Length Test 4: Length of an invalid directory");  File file4x = new File(testDirPath+"\\a\\"+ READ\_ONLY\_DIRECTORY\_NAME);  result = osMkDir(file4x);  fileLength = osLength(file4x);  System.out.println("Length Test 4: "+ file4x.getName()+": " + fileLength);  System.out.println();    //test length with try/catch : file readonly  System.out.println("Length Test 5: Length of a valid read only file");  result = osCreateNewFile(readOnlyFile);  fileLength = osLength(readOnlyFile);  System.out.println("Length Test 5: " + fileLength);  System.out.println();    //test length with try/catch : directory readonly  System.out.println("Length Test 6: Length of a valid read only directory");  result = osMkDir(readOnlyDir);  fileLength = osLength(readOnlyDir);  System.out.println("Length Test 6: "+ readOnlyDir.getName()+": " + fileLength);  System.out.println();    //test length with try/catch : null file  System.out.println("Length Test 7: Length of a null file");  File nullFile = null;  fileLength = osLength(nullFile);  System.out.println("Length Test 7: nullFile : " + fileLength);  System.out.println();  }// end demonstrate length with try catch    //(4) demonstrate mkdir() - Creates the directory named by this abstract pathname.  // Returns: true if and only if the directory was created;  // false otherwisee  //Throws: SecurityException - If a security manager exists and its  //SecurityManager.checkWrite(java.lang.String) method does not permit the named directory to be created  public static void demonstrateMkDir(String testDirPath )  {  boolean result = false;    // set file & directory names  String dirname1 = getDirPath1(testDirPath);  String dirname2 = getDirPath2(testDirPath);  String dirname3 = getDirPath3(testDirPath);  String dirname4 = getDirPath4(testDirPath);  String readOnlyDirPath = getReadOnlyDirPath(testDirPath);  String readOnlyDirPath2 = getReadOnlyDirPath(testDirPath)+"\\"+"testdir";  String invalidDirPath = testDirPath+FILE\_1+"\\"+FILE\_2;    File dir1 = osFile(dirname1);  File dir2 = osFile(dirname2);  File dir3 = osFile(dirname3);  File dir4 = osFile(dirname4);  File readOnlyDir = osFile(readOnlyDirPath);  File invalidDir = osFile(invalidDirPath);    System.out.println();  System.out.println();  System.out.println("\nDemonstrate mkdir() without try/catch\n");    // mkdir test 1 : without try/catch : directory does not exist  System.out.println("mkdir Test 1: creating directory with valid path");    // mkdir() - Creates the directory named by this abstract pathname.  result = dir4.mkdir();  if (result)  System.out.println("mkdir Test 1: Success " + dir4.getName() + " was created");  else  System.out.println("mkdir Test 1: Failed " + dir4.getName() + " was not created");  System.out.println();      // mkdir test 2 : without try/catch : directory exists  System.out.println("mkdir Test 2: creating directory that already exists");  result = dir4.mkdir();  if (result)  System.out.println("mkdir Test 2: Failed " + dir4.getName() + " was created");  else  System.out.println("mkdir Test 2: Success " + dir4.getName() + " was not created");  System.out.println();    // mkdir test 3 : without try/catch : only read permissions directory  System.out.println("mkdir Test 3: creating directory in a readonly directory");  result = readOnlyDir.mkdir();  if (result)  System.out.println("mkdir Test 3: Failed " + readOnlyDir.getName() + " was created");  else  System.out.println("mkdir Test 3: Success " + readOnlyDir.getName() + " was not created");  System.out.println();    // mkdir test 4 : without try/catch : invalid path  System.out.println("mkdir Test 4: creating directory with an invalid path");  result = invalidDir.mkdir();  if (result)  System.out.println("mkdir Test 4: Failed " + invalidDir.getName() + " was created");  else  System.out.println("mkdir Test 4: Success " + invalidDir.getName() + " was not created");  System.out.println();    // mkdir test 5 : without try/catch : null file  System.out.println("mkdir Test 5: creating a null directory ");  System.out.printf("The next test throws an exception & causes the program to halt. "  + "Do you want to continue? y n : ");  Scanner keyboard = new Scanner(System.in);  String runTest = keyboard.nextLine();  if(runTest.equals("y"))  {  File nullfile = null;  result = nullfile.mkdir();  if (result)  System.out.println("mkdir Test 5: Failed nullfile was created");  else  System.out.println("mkdir Test 5: Success nullfile was not created");  }  else  System.out.println("mkdir Test 5: did not run");  System.out.println();  }// end demonstrate mkdir without try/catch    public static void demonstrateMkDirTryCatch(String testDirPath)  {  boolean result = false;    // set file & directory names  String dirname1 = getDirPath1(testDirPath);  String dirname2 = getDirPath2(testDirPath);  String dirname3 = getDirPath3(testDirPath);  String dirname4 = getDirPath4(testDirPath);  String readOnlyDirPath = getReadOnlyDirPath(testDirPath);  String readOnlyDirPath2 = getReadOnlyDirPath(testDirPath)+"\\"+"testdir";  String invalidDirPath = testDirPath+FILE\_1+"\\"+FILE\_2;  File dir1 = osFile(dirname1);  File dir2 = osFile(dirname2);  File dir3 = osFile(dirname3);  File dir4 = osFile(dirname4);  File readOnlyDir = osFile(readOnlyDirPath);  File invalidDir = osFile(invalidDirPath);    System.out.println();  System.out.println();  System.out.println("\nDemonstrate mkdir() with Try/Catch\n");    // mkdir test 1: with try/catch : directory does not exist  System.out.println("mkdir Test 1: creating directory with valid path");    // mkdir() - Creates the directory named by this abstract pathname.  result = osMkDir(dir4);  if (result)  System.out.println("mkdir Test 1: Success " + dir4.getName() + " was created");  else  System.out.println("mkdir Test 1: Failed " + dir4.getName() + " was not created");  System.out.println();    // mkdir test 6 : with try/catch : directory exists  System.out.println("mkdir Test 2: creating directory that already exists");  result = osMkDir(dir4);  if (result)  System.out.println("mkdir Test 2: Failed " + dir4.getName() + " was created");  else  System.out.println("mkdir Test 2: Success " + dir4.getName() + " was not created");  System.out.println();    // mkdir test 7 : with try/catch : only read permissions directory  System.out.println("mkdir Test 3: creating directory in a read only directory");  result = osMkDir(readOnlyDir);  if (result)  System.out.println("mkdir Test 3: Failed " + readOnlyDir.getName() + " was created");  else  System.out.println("mkdir Test 3: Success " + readOnlyDir.getName() + " was not created");  System.out.println();    // mkdir test 8 : with try/catch : invalid path  System.out.println("mkdir Test 4: creating directory with an invalid path");    result = osMkDir(invalidDir);  if (result)  System.out.println("mkdir Test 4: Failed " + invalidDir.getName() + " was created");  else  System.out.println("mkdir Test 4: Success " + invalidDir.getName() + " was not created");  System.out.println();    // mkdir test 5 : without try/catch : null file  System.out.println("mkdir Test 5: creating null directory");  File nullfile = null;  result = osMkDir(nullfile);  if (result)  System.out.println("mkdir Test 5: Failed nullfile was created");  else  System.out.println("mkdir Test 5: Success nullfile was not created");  System.out.println();  }// end demonstrate mkdir with try/catch    //Demonstrate (5) renameTo(File dest) - Renames the file denoted by this abstract pathname.  // Returns: true if and only if the renaming succeeded;  // false otherwise  // throws SecurityException | NullPointerException - If parameter dest is null  public static void demonstrateRenameTo(String testDirPath, String myReadOnlyFilePath)  {  boolean result = false;    System.out.println("\nDemonstrate renameTo() without try/catch\n");    // set file & directory names  String filename1 = getFilePath1(testDirPath);  String filename2 = getFilePath2(testDirPath);  String filename3 = getFilePath3(testDirPath);  String filename4 = getFilePath4(testDirPath);  String dirname1 = getDirPath1(testDirPath);  String dirname2 = getDirPath2(testDirPath);  String dirname3 = getDirPath3(testDirPath);  String dirname4 = getDirPath4(testDirPath);  String readOnlyFilePath = getReadOnlyFilePath(testDirPath);  String readOnlyDirPath = getReadOnlyDirPath(testDirPath);  String readOnlyDirPath2 = getReadOnlyDirPath(testDirPath)+"\\"+"testdir";  String invalidDirPath = testDirPath+FILE\_1+"\\"+FILE\_2;    File file1 = osFile(filename1);  File file2 = osFile(filename2);  File file3 = osFile(filename3);  File file4 = osFile(filename4);    File dir1 = osFile(dirname1);  File dir2 = osFile(dirname2);  File dir3 = osFile(dirname3);  File dir4 = osFile(dirname4);    File readOnlyFile = osFile(myReadOnlyFilePath);  File readOnlyDir = osFile(readOnlyDirPath);    osCreateNewFile(file1);  osCreateNewFile(file2);  osCreateNewFile(file3);  osCreateNewFile(readOnlyFile);  osMkDir(dir1);  osMkDir(dir2);    osDelete(file3);    System.out.println();  System.out.println();    // renameTo(File dest) - Renames the file denoted by this abstract pathname.  System.out.println("renameTo Test 1: renaming a file with valid paths");  result = file1.renameTo(file3);  if(result)  System.out.println("renameTo Test 1: Success " + file1.getName() + " was renamed to " + file3.getName());  else  System.out.println("renameTo Test 1: Failed " + file1.getName() + " was not renamed to "+ file3.getName());  System.out.println();    System.out.println("renameTo Test 2: renaming a file with not a valid from");  result = file1.renameTo(file3);  if(result)  System.out.println("renameTo Test 2: Failed " + file1.getName() + " was renamed to " + file3.getName());  else  System.out.println("renameTo Test 2: Success " + file1.getName() + " was not renamed to "+ file3.getName());  System.out.println();    System.out.println("renameTo Test 3: renaming a file with not a valid to (exists");  result = file3.renameTo(file2);  if(result)  System.out.println("renameTo Test 3: Failed " + file3.getName() + " was renamed to " + file2.getName());  else  System.out.println("renameTo Test 3: Success " + file3.getName() + " was not renamed to "+ file2.getName());  System.out.println();    System.out.println("renameTo Test 4: renaming a read only file ");  result = readOnlyFile.renameTo(file4);  if(result)  System.out.println("renameTo Test 4: Failed " + readOnlyFile.getName() + " was renamed to " + file4.getName());  else  System.out.println("renameTo Test 4: Success " + readOnlyFile.getName() + " was not renamed to "+ file4.getName());  System.out.println();      System.out.println("reNameTo Test 5: renaming a directory with valid paths");  result = dir2.renameTo(dir3);  if(result)  System.out.println("renameTo Test 5: Success " + dir2.getName() + " was renamed to " + dir3.getName());  else  System.out.println("renameTo Test 5: Failed " + dir2.getName() + " was not renamed to "+ dir3.getName());  System.out.println();    System.out.println("mkdir Test 6: renaming a directory that no longer exists");  result = dir2.renameTo(dir3);  if(result)  System.out.println("renameTo Test 6: Failed " + dir2.getName() + " was renamed to " + dir3.getName());  else  System.out.println("renameTo Test 6: Success " + dir2.getName() + " was not renamed to "+ dir3.getName());  System.out.println();    System.out.println("mkdir Test 7: renaming to a nullfile");  System.out.printf("The next test throws an exception & causes the program to halt. "  + "Do you want to continue? y n : ");  Scanner keyboard = new Scanner(System.in);  String runTest = keyboard.nextLine();  osMkDir(dir2);    if(runTest.equals("y"))  {  File nullfile = null;    result = dir2.renameTo(nullfile);  if(result)  System.out.println("renameTo Test 7: Failed " + dir2.getName() + " was renamed to nullfile");  else  System.out.println("renameTo Test 7: Success " + dir2.getName() + " was not renamed to nullfile");  }  else  System.out.println("renameTo Test 7: did not run");  System.out.println();    System.out.println("mkdir Test 8: renaming from a nullfile");  System.out.printf("The next test throws an exception & causes the program to halt. "  + "Do you want to continue? y n : ");  keyboard = new Scanner(System.in);  runTest = keyboard.nextLine();  if(runTest.equals("y"))  {  File nullfile = null;  result = nullfile.renameTo(dir2);  if(result)  System.out.println("renameTo Test 8: Failed nullfile was renamed to " + dir2.getName());  else  System.out.println("renameTo Test 8: Success nullfile was not renamed to" + dir2.getName());  System.out.println();  }  else  System.out.println("renameTo Test 8: did not run");  } //end demonstrate renameTo without try/catch    public static void demonstrateRenameToTryCatch(String testDirPath, String myReadOnlyFilePath)  {  boolean result = false;    System.out.println("\nDemonstrate renameTo() with try/catch\n");    // set file & directory names  String filename1 = getFilePath1(testDirPath);  String filename2 = getFilePath2(testDirPath);  String filename3 = getFilePath3(testDirPath);  String filename4 = getFilePath4(testDirPath);  String dirname1 = getDirPath1(testDirPath);  String dirname2 = getDirPath2(testDirPath);  String dirname3 = getDirPath3(testDirPath);  String dirname4 = getDirPath4(testDirPath);  String readOnlyFilePath = getReadOnlyFilePath(testDirPath);  String readOnlyDirPath = getReadOnlyDirPath(testDirPath);  String readOnlyDirPath2 = getReadOnlyDirPath(testDirPath)+"\\"+"testdir";  String invalidDirPath = testDirPath+FILE\_1+"\\"+FILE\_2;    File file1 = osFile(filename1);  File file2 = osFile(filename2);  File file3 = osFile(filename3);  File file4 = osFile(filename4);    File dir1 = osFile(dirname1);  File dir2 = osFile(dirname2);  File dir3 = osFile(dirname3);  File dir4 = osFile(dirname4);    File readOnlyFile = osFile(myReadOnlyFilePath);  File readOnlyDir = osFile(readOnlyDirPath);    osCreateNewFile(file1);  osCreateNewFile(file2);  osCreateNewFile(file3);  osCreateNewFile(readOnlyFile);  osMkDir(dir1);  osMkDir(dir2);    osDelete(file3);  osDelete(file4);    // renameTo(File dest) - Renames the file denoted by this abstract pathname.  System.out.println("renameTo Test 1: renaming a file with valid paths");  result = osRenameTo(file1,file3);  if(result)  System.out.println("renameTo Test 1: Success" + file1.getName() + " was renamed to " + file3.getName());  else  System.out.println("renameTo Test 1: Failed " + file1.getName() + " was not renamed to "+ file3.getName());  System.out.println();    System.out.println("renameTo Test 2: renaming a file with not a valid from");  result = osRenameTo(file1,file3);  if(result)  System.out.println("renameTo Test 2: Failed" + file1.getName() + " was renamed to " + file3.getName());  else  System.out.println("renameTo Test 2: Success " + file1.getName() + " was not renamed to "+ file3.getName());  System.out.println();    System.out.println("renameTo Test 3: renaming a file with not a valid to exists");  result = osRenameTo(file3, file2);  if(result)  System.out.println("renameTo Test 3: Failed" + file3.getName() + " was renamed to " + file2.getName());  else  System.out.println("renameTo Test 3: Success " + file3.getName() + " was not renamed to "+ file2.getName());  System.out.println();    System.out.println("renameTo Test 4: renaming a read only file ");  result = osRenameTo(readOnlyFile,file4);  if(result)  System.out.println("renameTo Test 4: Failed " + readOnlyFile.getName() + " was renamed to " + file4.getName());  else  System.out.println("renameTo Test 4: Success " + readOnlyFile.getName() + " was not renamed to "+ file4.getName());  System.out.println();    System.out.println("reNameTo Test 5: renaming a directory with valid paths");  result = osRenameTo(dir2,dir3);  if(result)  System.out.println("renameTo Test 5: Success" + dir2.getName() + " was renamed to " + dir3.getName());  else  System.out.println("renameTo Test 5: Failed " + dir2.getName() + " was not renamed to "+ dir3.getName());  System.out.println();    result = osRenameTo(dir2,dir3);  System.out.println("mkdir Test 6: renaming a directory that no longer exists");  if(result)  System.out.println("renameTo Test 6: Failed" + dir2.getName() + " was renamed to " + dir3.getName());  else  System.out.println("renameTo Test 6: Success " + dir2.getName() + " was not renamed to "+ dir3.getName());  System.out.println();    System.out.println("mkdir Test 7: renaming to a nullfile");  File nullfile = null;  osMkDir(dir2);  result = osRenameTo(dir2,nullfile);  if(result)  System.out.println("renameTo Test 7: Failed " + dir2.getName() + " was renamed to nullfile");  else  System.out.println("renameTo Test 7: Success " + dir2.getName() + " was not renamed to nullfile");  System.out.println();    System.out.println("mkdir Test 8: renaming from a nullfile");  result = osRenameTo(nullfile,dir2);  if(result)  System.out.println("renameTo Test 8: Failed nullfile was renamed to " + dir2.getName());  else  System.out.println("renameTo Test : Success nullfile was not renamed to " + dir2.getName());  System.out.println();  }//end demonstrate renameTo with try catch  }//end class Program1 |
| Output Results |
|  |
| Testing |
| **Description of Testing:**  4 sets of Tests were performed on each Operating System Functionality.   * Test Set1: without Exception code and without Security Manager * Test Set 2: with Exception Code and without Security Manager * Test Set 3: with Exception Code and with Security Manager * Test Set 4: without Exception Code and with Security Manager   Security Manager: a default Security Manager object was created to see the exceptions generated. No configuration was done on the Security Manager.   * without a try/catch block, it throws an exception on all tests.   Example:  createNewFile Test 1: Creating a new file that does not exist  Exception in thread "main" java.security.AccessControlException: access denied ("java.io.FilePermission" "C:\users\bernice\javaDirectory\testDirectory30\file1.txt" "write")  at java.security.AccessControlContext.checkPermission(AccessControlContext.java:457)  at java.security.AccessController.checkPermission(AccessController.java:884)  at java.lang.SecurityManager.checkPermission(SecurityManager.java:549)  at java.lang.SecurityManager.checkWrite(SecurityManager.java:979)  at java.io.File.createNewFile(File.java:1008)  at program1.Program1.demonstrateCreateNewFile(Program1.java:805)  at program1.Program1.main(Program1.java:267)  Java Result: 1   * with a try/catch block, the exception was caught on all tests and an error messaged printed.   Example:  createNewFile Test 1: Creating a new file that does not exist  program1.Program1java.security.AccessControlException: access denied ("java.io.FilePermission" "C:\users\bernice\javaDirectory\testDirectory30\file2.txt" "delete")  osDelete Security Exception Errorfile2.txt  program1.Program1java.security.AccessControlException: access denied ("java.io.FilePermission" "C:\users\bernice\javaDirectory\testDirectory30\file2.txt" "write")  osCreateNewFile: SE Error: file2.txt  createNewFile Test 1: Failed: did not create file2.txt not created  Exceptions Tests were performed:   * without try/catch blocks (using throws) * with try/catch blocks   Catch : The Catch Blocks in these tests print an error message.  From: <https://docs.oracle.com/javase/tutorial/essential/exceptions/catch.html>  You associate exception handlers with a try block by providing one or more catch blocks directly after the try block. No code can be between the end of the try block and the beginning of the first catch block.  Each catch block is an exception handler that handles the type of exception indicated by its argument. The argument type,*ExceptionType*, declares the type of exception that the handler can handle and must be the name of a class that inherits from the Throwable class. The handler can refer to the exception with *name*.  The catch block contains code that is executed if and when the exception handler is invoked. The runtime system invokes the exception handler when the handler is the first one in the call stack whose *ExceptionType* matches the type of the exception thrown. The system considers it a match if the thrown object can legally be assigned to the exception handler's argument.  Exception handlers can do more than just print error messages or halt the program.  They can do error recovery, prompt the user to make a decision, or propagate the error up to a higher-level handler using chained exceptions, as described in the[Chained Exceptions](https://docs.oracle.com/javase/tutorial/essential/exceptions/chained.html) section.  **Tests:**   1. **Description of Test: createNewFile()**   createNewFile()- Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.  The check for the existence of the file and the creation of the file if it does not exist are a single operation that is atomic with respect to all other filesystem activities that might affect the file.  Note: this method should *not* be used for file-locking, as the resulting protocol cannot be made to work reliably. The [FileLock](http://docs.oracle.com/javase/6/docs/api/java/nio/channels/FileLock.html) facility should be used instead.  Returns:  true if the named file does not exist and was successfully created;  false if the named file already exists  **Test Results**  createNewFile() Test 1: test without try/catch and without the security manager   1. Create a file that does not exists : Logic Error if it fails   createNewFile Test 1: Creating a new file that does not exist  createNewFile Test 1: Success C:\users\bernice\javaDirectory\testDirectory\file1.txt created   1. Create a file that already exists : Logic Error   createNewFile Test 2: Creating a new file that exists  createNewFile Test 2: Success C:\users\bernice\javaDirectory\testDirectory\file1.txt not created   1. Create a file with an invalid path : Syntax Error (throws exception)   The next test throw an exception & cause the program to halt. Do you want to continue? y n : y  Test creating a file with an invalid path:  Exception in thread "main" java.io.IOException: The filename, directory name, or volume label syntax is incorrect  at java.io.WinNTFileSystem.createFileExclusively(Native Method)  at java.io.File.createNewFile(File.java:1012)  at program1.Program1.demonstrateCreateNewFile(Program1.java:837)  at program1.Program1.main(Program1.java:155)  Java Result: 1   1. Create file in a read-only directory : Logic Error (throws exception)   createNewFile Test 4: Creating a new file in a read only directory  The next test throw an exception & cause the program to halt. Do you want to continue? y n : y  Test creating a file in a read only directory:  Exception in thread "main" java.io.IOException: Access is denied  at java.io.WinNTFileSystem.createFileExclusively(Native Method)  at java.io.File.createNewFile(File.java:1012)  at program1.Program1.demonstrateCreateNewFile(Program1.java:857)  at program1.Program1.main(Program1.java:155)  Java Result: 1  BUILD SUCCESSFUL (total time: 18 seconds)   1. Create a null file : Syntax Error (throws exception)   createNewFile Test 5: Creating a null file  The next test throw an exception & cause the program to halt. Do you want to continue? y n : y  Test creating a null file:  Exception in thread "main" java.lang.NullPointerException  at program1.Program1.demonstrateCreateNewFile(Program1.java:877)  at program1.Program1.main(Program1.java:155)  Java Result: 1  createNewFile() Test 2: test with try/catch and without the security manager   1. Create a file that does not exists : Logic Error if it fails   CreateNewFile Success : file2.txt  createNewFile Test 1: Success: file created: file2.txt created   1. Create a file that already exists : Logic Error   CreateNewFile Error: file2.txt  createNewFile Test 6: Success file2.txt not created   1. Create a file with an invalid path : Syntax Error   Test creating a file with an invalid path:  program1.Program1java.io.IOException: The filename, directory name, or volume label syntax is incorrect: file2.txt  createNewFile Test 3: Success file2.txt not created   1. Create file in a read-only directory : Logic Error   createNewFile Test 4: Creating a new file in a read only directory  Test creating a file in a read only directory:  program1.Program1java.io.IOException: Access is denied: readOnlyFile  createNewFile Test 4: Success readOnlyFile not created   1. Create a null file : Syntax Error   Test creating a null file:  program1.Program1java.lang.NullPointerException  NullPointer Error  createNewFile Test 5: Success nullfile not created  createNewFile() Exceptions:   * IOException - If an I/O error occurred * Null pointer – If the pathname argument is null * Security Exception -- If a security manager exists and   its SecurityManager.checkWrite(java.lang.String)  method denies write access to the file  Code to catch all potential exceptions:  public static boolean osCreateNewFile(File f1)  {  boolean result = false;  try{  // throws IOException | SecurityException  result = f1.createNewFile();    if(result)  System.out.println("CreateNewFile Success : " + f1.getName());  else  System.out.println("CreateNewFile Error: " + f1.getName());  }  catch (IOException ex)  {  System.out.println(Program1.class.getName() + ex + ": " + f1.getName() );  //System.out.println( " osCreateNewFile: IO Error: " + f1.getName());  }  catch ( SecurityException se)  {  System.out.println(Program1.class.getName() + se);  System.out.println(" osCreateNewFile: SE Error: " + f1.getName());  }  catch (NullPointerException np )  {  System.out.println(Program1.class.getName() + np);  System.out.println("NullPointer Error");  }  return result;  }//end osCreateNewFile   1. **Description of Test:** delete() –   Deletes the file or directory denoted by this abstract pathname. If this pathname denotes a directory, then the directory must be empty in order to be deleted.  Returns:   * true if and only if the file or directory is successfully deleted; * false otherwise   delete() Test 1: test without try/catch and without the security manager  Delete Tests for read only file are not valid. Different permission settings for deleted were tested, but further research is needed to find the correct Windows settings for a read only file and a “no delete” file.  Syntax/Logic Errors:   1. Delete a file that exists : Logic Error if it fails   Delete Test 1: Deleting a file that exists  Delete Test 1: Success file1.txt was deleted   1. Delete a file that does not exist : Syntax Error (invalid path)   Delete Test 2: Deleting a file that does not exists  Delete Test 2: Success file1.txt was not deleted   1. Delete a file that is read only : Logic error   Delete Test 3: Deleting a file that is read only  Delete Test 3: Failed myReadOnlyFile.txt was deleted   1. Delete an empty directory : Logic error if it fails   Delete Test 4: Deleting an empty directory  Delete Test 4: Success dir1 was deleted   1. Delete a directory that is not empty : Logic Error   Delete Test 5: Deleting a directory with files  Delete Test 5: Success testDirectory was not deleted   1. Delete a read only directory : Logic error   Delete Test 6: Deleting a read only directory  Delete Test 6: Success readOnlyDir was not deleted   1. Delete a null file : Syntax Error (throws an exception)   Delete Test 7: Deleting a null file  The next test throws an exception & causes the program to halt. Do you want to continue? y n : y  Exception in thread "main" java.lang.NullPointerException  at program1.Program1.demonstrateDelete(Program1.java:1061)  at program1.Program1.main(Program1.java:161)  Java Result: 1  delete() Test 2: test with try/catch and without the security manager  Syntax/Logic Errors:   1. Delete a file that exists : Logic Error if it fails   CreateNewFile Success : file1.txt  Delete Test 1: Deleting a file that exists  Delete success: file1.txt  Delete Test 1: Success file1.txt was deleted   1. Delete a file that does not exist : Syntax Error (invalid path)   Delete Test 2: Deleting a file that does not exist  Delete failed: file1.txt  Delete Test 2: Success file1.txt was not deleted   1. Delete a file that is read only : Logic error   Delete Test 3: Deleting a file that is readonly  CreateNewFile Success : readOnlyFile  Delete success: readOnlyFile  Delete Test 3: Failed readOnlyFile was deleted   1. Delete an empty directory : Logic error if it fails   mkDir Success: dir1  Delete Test 4: Deleting an empty directory  Delete success: dir1  Delete Test 4: Success dir1 was delete   1. Delete a directory that is not empty : Logic Error   Delete Test 5: Deleting a directory with files  Delete failed: testDirectory  Delete Test 5: Success testDirectory was not deleted   1. Delete a read only directory : Logic error   Delete Test 6: Deleting a read only directory  Delete failed: readOnlyDir  Delete Test 6: Success readOnlyDir was not deleted   1. Delete a null file : Syntax Error   Delete Test 7: Deleting a null file  program1.Program1java.lang.NullPointerException  osDelete NullPointer Error  Delete Test 7: Success  delete() Exceptions:   * Null pointer – If the pathname argument is null * Security Exception -- [SecurityException](http://docs.oracle.com/javase/6/docs/api/java/lang/SecurityException.html) - If a security manager exists and its [SecurityManager.checkDelete(java.lang.String)](http://docs.oracle.com/javase/6/docs/api/java/lang/SecurityManager.html#checkDelete(java.lang.String)) method denies delete access to the file   Code to catch all potential exceptions:  public static boolean osDelete(File f2)  {  boolean result = false;    try  {  result = f2.delete();  if (result)  System.out.println("Delete success: " + f2.getName() );  else  System.out.println("Delete failed: " + f2.getName());  }//end try  catch(SecurityException se)  {  System.out.println(Program1.class.getName() + se);  System.out.println(" osDelete Security Exception Error" + f2.getName());  }  catch(NullPointerException np)  {  System.out.println(Program1.class.getName() + np);  System.out.println(" osDelete NullPointer Error");  }  return result;  }//end osDelete   1. **Description of Test:** length() -   Returns the length of the file denoted by this abstract pathname.  The return value is unspecified if this pathname denotes a directory.  Returns:   * The length, in bytes, of the file denoted by this abstract pathname, * or 0L if the file does not exist. * Some operating systems may return0L for pathnames denoting system-dependent entities such as devices or pipes.   length() Test 1: test without try/catch and without the security manager  Syntax/Logic Errors:   1. Length of a file that exists : Logic Error if Incorrect result   CreateNewFile Success : file1.txt  Length Test 1: file1.txt: 0   1. Length of a directory that exists: Logic error if it fails   mkDir Success: dir1  Length Test 2: dir1: 0   1. Length of a file that does not exist : Syntax Error   Result: still gives a length  program1.Program1java.io.IOException: The filename, directory name, or volume label syntax is incorrect: testDirectory  Length Test 3: testDirectory: 0   1. Length of a directory that does not exist : Syntax error   mkdir fail: readOnlyDir  Length Test 4: readOnlyDir: 0   1. Length of a read only file   CreateNewFile Error: readOnlyFile  Length Test 5: readOnlyFile: 0   1. Length of a read only directory   mkdir fail: readOnlyDir  Length Test 6: readOnlyDir: 0   1. Length of a null file : Syntax error (throws exception)   Length Test 7: Length of a null file  The next test throws an exception & causes the program to halt. Do you want to continue? y n : y  Exception in thread "main" java.lang.NullPointerException  at program1.Program1.demonstrateLength(Program1.java:1255)  at program1.Program1.main(Program1.java:164)  Java Result: 1  length() Test 2: test with try/catch and without the security manager  Syntax/Logic Errors:   1. Length of a file that exists : Logic Error if Incorrect result   Length Test 1: Length of a valid file  CreateNewFile Success : file1.txt  Length Test 1: file1.txt: 0   1. Length of a directory that exists: Logic error if it fails   Length Test 2: Length of a valid directory  mkDir Success: dir1  Length Test 2: dir1: 0   1. Length of a file that does not exist : Syntax Error   Length Test 3: Length of an invalid file  program1.Program1java.io.IOException: The filename, directory name, or volume label syntax is incorrect: testDirectory  Length Test 3: testDirectory: 0   1. Length of a directory that does not exist : Syntax error   Length Test 4: Length of an invalid directory  mkdir fail: readOnlyDir  Length Test 4: readOnlyDir: 0   1. Length of a read only file   Length Test 5: Length of a valid read only file  CreateNewFile Success : readOnlyFile  Length Test 5: 0   1. Length of a read only directory   Length Test 6: Length of a valid read only directory  mkdir fail: readOnlyDir  Length Test 6: readOnlyDir: 0   1. Length of a null file : Syntax error   Length Test 7: Length of a null file  program1.Program1java.lang.NullPointerException  NullPointer Error  Length Test 7: nullFile : -1  length() Exceptions:   * Null pointer – If the pathname argument is null * Security Exception -- [SecurityException](http://docs.oracle.com/javase/6/docs/api/java/lang/SecurityException.html) - If a security manager exists and its [SecurityManager.checkRead(java.lang.String)](http://docs.oracle.com/javase/6/docs/api/java/lang/SecurityManager.html#checkRead(java.lang.String)) method denies read access to the file     Code to catch all potential exceptions:  public static long osLength(File file){  long fileLength = -1;  try {  fileLength = file.length();  }  catch( SecurityException ex)  {  System.out.println(Program1.class.getName() + ex);  System.out.println(" SecurityException: " + file.getName());  }  catch (NullPointerException np )  {  System.out.println(Program1.class.getName() + np);  System.out.println("NullPointer Error");  }  return fileLength;  }//osLength   1. **Description of Test:** mkdir() –   Creates the directory named by this abstract pathname.  Returns:   * true if and only if the directory was created; * false otherwise   mkdir() Test 1: test without try/catch and without the security manager  Syntax/Logic Errors:   1. Create a directory with valid path : Logic Error if it fails   mkdir Test 1: creating directory with valid path  mkdir Test 1: Success dir4 was created   1. Create a directory with a path already exists : Logic Error   mkdir Test 2: creating directory that already exists  mkdir Test 2: Success dir4 was not created   1. Create a directory inside a read only directory : Logic Error   mkdir Test 3: creating directory in a readonly directory  mkdir Test 3: Success readOnlyDir was not created   1. Creating a directory with an invalid path : syntax error   mkdir Test 4: creating directory with an invalid path  mkdir Test 4: Success file2.txt was not created   1. Create a directory with a null path : Syntax Error   (throws exception)  mkdir Test 5: creating a null directory  The next test throws an exception & causes the program to halt. Do you want to continue? y n : y  Exception in thread "main" java.lang.NullPointerException  at program1.Program1.demonstrateMkDir(Program1.java:1430)  at program1.Program1.main(Program1.java:167)  Java Result: 1  mkdir() Test 2: test with try/catch and without the security manager  Syntax/Logic Errors:   1. Create a directory with valid path : Logic Error if it fails   mkdir Test 1: creating directory with valid path  mkDir Success: dir4  mkdir Test 1: Success dir4 was created   1. Create a directory with a path already exists : Logic Error   mkdir Test 2: creating directory that already exists  mkdir fail: dir4  mkdir Test 2: Success dir4 was not created   1. Create a directory inside a read only directory : Logic Error   mkdir Test 3: creating directory in a read only directory  mkdir fail: readOnlyDir  mkdir Test 3: Success readOnlyDir was not created   1. Creating a directory with an invalid path : syntax error   mkdir Test 4: creating directory with an invalid path  mkdir fail: file2.txt  mkdir Test 4: Success file2.txt was not created   1. Create a directory with a null path : Syntax Error   mkdir Test 5: creating directory with an invalid path  program1.Program1java.lang.NullPointerException  NullPointer Error  mkdir() Exceptions:   * Null pointer – If the pathname argument is null * Security Exception -- [SecurityException](http://docs.oracle.com/javase/6/docs/api/java/lang/SecurityException.html) - If a security manager exists and its [SecurityManager.checkWrite(java.lang.String)](http://docs.oracle.com/javase/6/docs/api/java/lang/SecurityManager.html#checkWrite(java.lang.String)) method does not permit the named directory to be created   Code to catch all potential exceptions:  public static boolean osMkDir(File dir){  boolean result = false;  try {  // throws SecurityException  result = dir.mkdir();    if (result)  {  System.out.println("mkDir Success: " + dir.getName());  }  else  {  System.out.println("mkdir fail: " + dir.getName());  }  }  catch(SecurityException ex)  {  System.out.println(Program1.class.getName() + ex);  System.out.println("dir.mkdir failed: Security execption");  }  catch (NullPointerException np )  {  System.out.println(Program1.class.getName() + np);  System.out.println("NullPointer Error");  }  return result;  }//end osMkDir     1. **Description of Test:** renameTo(File dest) –   Renames the file denoted by this abstract pathname.  Many aspects of the behavior of this method are inherently platform-dependent: The rename operation might not be able to move a file from one filesystem to another, it might not be atomic, and it might not succeed if a file with the destination abstract pathname already exists.  The return value should always be checked to make sure that the rename operation was successful.  Parameters:   * dest - The new abstract pathname for the named file   Returns:   * true if and only if the renaming succeeded; * false otherwise   reNameTo() Test 1: test without try/catch and without the security manager  Syntax/Logic Errors:   1. Rename from a valid path to a valid path : Logic error if it fails   renameTo Test 1: renaming a file with valid paths  renameTo Test 1: Success file1.txt was renamed to file3.txt   1. Rename from an invalid from path : Syntax Error   renameTo Test 2: renaming a file with not a valid from  renameTo Test 2: Success file1.txt was not renamed to file3.txt   1. Rename a file from a valid path to an invalid path : Syntax Error   renameTo Test 3: renaming a file with not a valid to (exists  renameTo Test 3: Success file3.txt was not renamed to file2.txt   1. Rename a file that is read only : Logic Error   renameTo Test 4: renaming a read only file  renameTo Test 4: Success readOnlyFile was not renamed to file4.txt   1. Rename a directory with valid paths : logic error if it fails   reNameTo Test 5: renaming a directory with valid paths  renameTo Test 5: Success dir2 was renamed to dir3   1. Rename a directory that does not exist : syntax error   mkdir Test 6: renaming a directory that no longer exists  renameTo Test 6: Success dir2 was not renamed to dir3   1. Rename a file to a null path : Syntax Error (throws exception)   mkdir Test 7: renaming to a nullfile  The next test throws an exception & causes the program to halt. Do you want to continue? y n : y  mkDir Success: dir2  Exception in thread "main" java.lang.NullPointerException  at java.io.File.renameTo(File.java:1392)  at program1.Program1.demonstrateRenameTo(Program1.java:1627)  at program1.Program1.main(Program1.java:173)  Java Result: 1   1. Rename a file from a null path : syntax error (throws exception)   mkdir Test 8: renaming from a nullfile  The next test throws an exception & causes the program to halt. Do you want to continue? y n : y  Exception in thread "main" java.lang.NullPointerException  at program1.Program1.demonstrateRenameTo(Program1.java:1645)  at program1.Program1.main(Program1.java:173)  Java Result: 1  reNameTo() Test 2: test with try/catch and without the security manager  Syntax/Logic Errors:   1. Rename from a valid path to a valid path : Logic error if it fails   renameTo Test 1: renaming a file with valid paths  Success renaming directory file1.txt to file3.txt  renameTo Test 1: Successfile1.txt was renamed to file3.txt   1. Rename from an invalid from path : Syntax Error   renameTo Test 2: renaming a file with not a valid from  Error renaming directory file1.txt to file3.txt  renameTo Test 2: Success file1.txt was not renamed to file3.txt   1. Rename a file from a valid path to an invalid path : Syntax Error   renameTo Test 3: renaming a file with not a valid to exists  Error renaming directory file3.txt to file2.txt  renameTo Test 3: Success file3.txt was not renamed to file2.txt   1. Rename a file that is read only : Logic Error   renameTo Test 4: renaming a read only file  Success renaming directory readOnlyFile to file4.txt  renameTo Test 4: Failed readOnlyFile was renamed to file4.txt   1. Rename a directory with valid paths : logic error if it fails   reNameTo Test 5: renaming a directory with valid paths  Success renaming directory dir2 to dir3  renameTo Test 5: Successdir2 was renamed to dir3   1. Rename a directory that does not exist : syntax error   Error renaming directory dir2 to dir3  mkdir Test 6: renaming a directory that no longer exists  renameTo Test 6: Success dir2 was not renamed to dir3   1. Rename a file to a null path : Syntax Error   mkdir Test 7: renaming to a nullfile  mkDir Success: dir2  program1.Program1java.lang.NullPointerException  NullPointer Error  renameTo Test 7: Success dir2 was not renamed to nullfile   1. Rename a file from a null path : syntax error   mkdir Test 8: renaming from a nullfile  program1.Program1java.lang.NullPointerException  NullPointer Error  renameTo Test : Success nullfile was not renamed to dir2  reNameTo() Exceptions:   * Null pointer : If parameter dest is null * Security Exception: [SecurityException](http://docs.oracle.com/javase/6/docs/api/java/lang/SecurityException.html) - If a security manager exists and its [SecurityManager.checkWrite(java.lang.String)](http://docs.oracle.com/javase/6/docs/api/java/lang/SecurityManager.html#checkWrite(java.lang.String)) method denies write access to either the old or new pathnames   Code to catch all potential exceptions:  public static boolean osRenameTo(File d1, File d2) {  boolean result = false;  try{  result = d1.renameTo(d2);  if (result)  {  System.out.println("Success renaming directory " + d1.getName() + " to " + d2.getName());  }  else  {  System.out.println("Error renaming directory " + d1.getName() + " to " + d2.getName());  }  }  catch (SecurityException ex )  {  System.out.println(Program1.class.getName() + ex);  System.out.println("SecurityException Error");  }  catch (NullPointerException np )  {  System.out.println(Program1.class.getName() + np);  System.out.println("NullPointer Error");  }  return result;  }//osRenameTo |
| Grading: |
| |  | | --- | | Program Functionalities as described above (**50%**)  Sample Input / Output Screen shots to show all requirements (**25%**)  Test to verify all requirements (**25%**) | |