CS3231
Object Oriented Programming I

Name: _____ ( ) Date: _____

Chapter 11: File Input and Output

Lesson Objectives

After completing the lesson, the student will be able to

- *discover file/directory properties, to delete and rename files/ directories using File class*
- *create directories using the File class*
- *write and append data to a file using the PrintWriter class*
- *throw File IO exceptions*
- *use try-catch to handle file exceptions*
- *read data from a file using the Scanner class and BufferReader class*
- *use StringTokenizer class to tokenize a string*
- *use JFileChooser class to display a GUI for navigating a file system and choosing a file*

11.1 The File Class

The File class contains the methods for obtaining the properties of a file/directory and for renaming and deleting a file/directory. Data stored in the program are temporary; they are lost when the program terminates. To permanently store the data created in a program, you need to save them in a file on a disk or other permanent storage device. The file can then be transported and read later by other programs.

Every file is placed in a directory in the file system. **An absolute file name (or full name) contains a file name with its complete path and drive letter.** For example, `c:\book\Welcome.java` is the absolute file name for the file `Welcome.java` on the Windows operating system. Here `c:\book` is referred to as the directory path for the file. Absolute file names are machine dependent. On the UNIX platform, the absolute file name may be `/home/book/Welcome.java`, where `/home/book` is the directory path for the file `Welcome.java`.

A relative file name is in relation to the current working directory. The complete directory path for a relative file name is omitted. For example, Welcome.java is a relative file name. If the current working directory is `c:\book`, the absolute file name would be `c:\book\Welcome.java`.

The File class is intended to provide an abstraction that deals with most of the machine dependent complexities of files and path names in a machine-independent fashion. **The File class contains the methods for obtaining file and directory properties and for renaming and deleting files and directories**, as shown in Table 11.1. However, the File class does not contain the methods for reading and writing file contents.

The file name is a string. The File class is a wrapper class for the file name and its directory path. For example, `new File("c:\\book")` creates a File object for the directory `c:\book`, and `new File("c:\\book\\test.dat")` creates a File object for the file `c:\book\test.dat`, both on Windows operating systems. You can use the File class's `isDirectory()` method to check whether the object represents a directory, and the `isFile()` method to check whether the object represents a file.

java.io.File	Description
+File(pathname: String)	Creates a File object for the specified path name. The path name may be a directory or a file.
+File(parent: String, child: String)	Creates a File object for the child under the directory parent. The child may be a file name or a subdirectory.
+File(parent: File, child: String)	Creates a File object for the child under the directory parent. The parent is a File object. In the preceding constructor, the parent is a string.
+exists(): boolean	Returns true if the file or the directory represented by the File object exists.
+canRead(): boolean	Returns true if the file represented by the File object exists and can be read.
+canWrite(): boolean	Returns true if the file represented by the File object exists and can be written.
+isDirectory(): boolean	Returns true if the File object represents a directory.
+isFile(): boolean	Returns true if the File object represents a file.
+isAbsolute(): boolean	Returns true if the File object is created using an absolute path name.
+isHidden(): boolean	Returns true if the file represented in the File object is hidden. The exact definition of hidden is system-dependent. On Windows, you can mark a file hidden in the File Properties dialog box. On Unix systems, a file is hidden if its name begins with a period(.) character.
+getAbsolutePath(): String	Returns the complete absolute file or directory name represented by the File object.
+getCanonicalPath(): String	Returns the same as getAbsolutePath() except that it removes redundant names, such as "." and "..", from the path name, resolves symbolic links (on Unix), and converts drive letters to standard uppercase (on Windows).
+getName(): String	Returns the last name of the complete directory and file name represented by the File object. For example, new File("c:\\book\\test.dat").getName() returns test.dat.
+getPath(): String	Returns the complete directory and file name represented by the File object. For example, new File("c:\\book\\test.dat").getPath() returns c:\book\test.dat.
+getParent(): String	Returns the complete parent directory of the current directory or the file represented by the File object. For example, new File("c:\\book\\test.d at").getParent() returns c:\book.
+lastModified(): long	Returns the time that the file was last modified.
+length(): long	Returns the size of the file, or 0 if it does not exist or if it is a directory.
+listFile(): File[]	Returns the files under the directory for a directory File object.
+delete(): boolean	Deletes the file or directory represented by this File object.The method returns true if the deletion succeeds.
+renameTo(dest: File): boolean	Creates a directory represented in this File object. Returns true if the directory is created successfully.
+mkdir(): boolean	Same as mkdir() except that it creates directory along with its parent directories if the p arent directories do not exist.

Table 11.1 The File class can be used to obtain file and directory properties, to delete and rename files and directories,and to create directories.

Note: The directory separator for Windows is a backslash (\). The backslash is a special character in Java and should be written as \\ in a string literal.

Constructing a File instance does not create a file on the machine. You can create a File instance for any file name regardless whether it exists or not. You can invoke the exists() method on a File instance to check whether the file exists.

Do not use absolute file names in your program. If you use a file name such as `c:\\book\\Welcome.java`, it will work on Windows but not on other platforms. **You should use a file name relative to the current directory**. For example, you may create a File object using `new File("Welcome.java")` for the file Welcome.java in the current directory. You may create a File object using `new File("image/C.png")` for the file C.png under the image directory in the current directory. The forward slash (/) is the Java directory separator, which is the same as on UNIX. The statement new File("image/C.png") works on Windows, UNIX, and any other platform.

PROGRAM 11-1 demonstrates how to create a File object and use the methods in the File class to obtain its properties. The program creates a File object for the file `C.png`. This file is stored under the image directory in the current directory.

PROGRAM 11-1

```
1   import java.io.*;
2   public class TestFileClass {
3     public static void main(String[] args) {
4     File file = new File("image/C.png");
5     System.out.println("Does it exist? " + file.exists());
6     System.out.println("The file has " + file.length() + " bytes");
7     System.out.println("Can it be read? " + file.canRead());
8     System.out.println("Can it be written? " + file.canWrite());
9     System.out.println("Is it a directory? " + file.isDirectory());
10    System.out.println("Is it a file? " + file.isFile());
11    System.out.println("Is it absolute? " + file.isAbsolute());
12    System.out.println("Is it hidden? " + file.isHidden());
13    System.out.println("Absolute path is " +
14    file.getAbsolutePath());
15    System.out.println("Last modified on " +
16    new java.util.Date(file.lastModified()));
17    }
18  }
```

PROGRAM OUTPUT

```
Does it exist? true
The file has 104167 bytes
Can it be read? true
Can it be written? true
Is it a directory? false
Is it a file? true
Is it absolute? false
Is it hidden? false
Absolute path is C:\ECLIPSE WORKSPACE 2017\Test\image\C.png
Last modified on Wed Mar 01 12:34:30 SGT 2017
```

11.2 The File Input and Output

A File object encapsulates the properties of a file or a path, but it does not contain the methods for creating a file or for writing/reading data to/from a file (referred to as data input and output, or IO for short). In order to perform IO, you need to create objects using appropriate Java IO classes. The objects contain the methods for reading/writing data from/to a file.

There are two types of files: text (e.g. characters) and binary (e.g. images).

Byte streams are used to read/write raw bytes (i.e. binary files) serially from/to an external device. All the byte streams are derived from the abstract superclasses **InputStream** and **OutputStream**, as illustrated in the class diagram below (Figure 11.1).

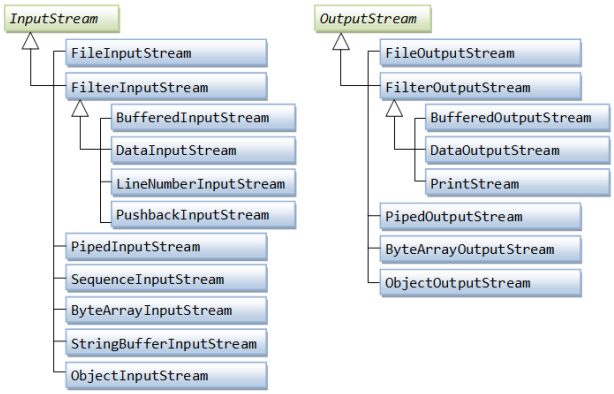


Figure 11.1

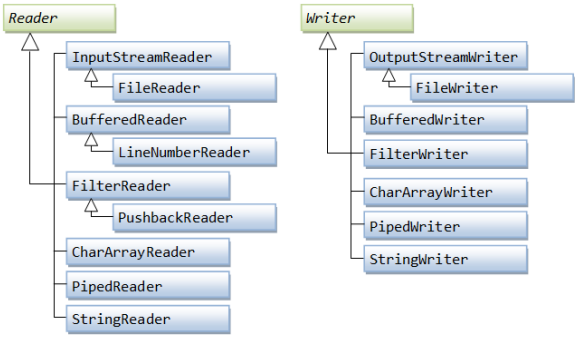


Figure 11.2

Java internally stores characters (char type) in 16-bit UCS-2 character set. But the external data source/sink could store characters in other character set (e.g., US-ASCII, ISO-8859-x, UTF-8, UTF-16, and many others), in fixed length of 8-bit or 16-bit, or in variable length of 1 to 4 bytes. **Hence, Java has to differentiate between byte-based I/O for processing 8-bit raw bytes, and character-based I/O for processing texts.** The character streams needs to translate between the character set used by external I/O devices and Java internal UCS-2 format. For example, the character '您' is stored as "60 A8" in UCS-2 (Java internal) and "E6 82 A8" in UTF8. If this character is to be written to a file that uses UTF-8, the character stream needs to translate "60 A8" to "E6 82 A8". The reserve takes place in a reading operation.

Other than the unit of operation and charset conversion (which is extremely complex), character-based I/O is almost identical to byte-based I/O. Instead of InputStream and OutputStream, we use Reader and Writer for character-based I/O (Figure 11.2).

This section introduces how to read/write strings and numeric values from/to a text file using the Scanner and PrintWriter classes (binary file IO will not be covered in this module).

11.2.1 Writing Data Using PrintWriter

The java.io.PrintWriter class can be used to create a file and write data to a text file. First, you have to create a PrintWriter object for a text file as follows:

```
PrintWriter output = new PrintWriter(filename);
```

Then, you can invoke the `print()`, `println()`, and `printf()` methods on the PrintWriter object to write data to a file. Table 11.2 summarizes frequently used methods in PrintWriter.

java.io.PrintWriter	Description
+PrintWriter(filename: String)	Creates a PrintWriter object for the specified file-name string.
+print(s: String): void	Writes a string to the file.
+PrintWriter(file: File)	Creates a PrintWriter object for the specified file object.
+print(c: char): void	Writes a character to the file.
+print(cArray: char[]): void	Writes an array of characters to the file.
+print(i: int): void	Writes an int value to the file.
+print(l: long): void	Writes a long value to the file.
+print(f: float): void	Writes a float value to the file.
+print(d: double): void	Writes a double value to the file.
+print(b: boolean): void	Writes a boolean value to the file.
+println(String x): void	A println method acts like a print method; additionally, it prints a line separator. The line-separator string is defined by the system. It is \r\n on Windows and \n on Unix.
+printf(String format, Object... args): PrintWriter	The printf method formats an output.

Table 11.2 The PrintWriter class contains the methods for writing data to a text file

PROGRAM 11-2 gives an example that creates an instance of PrintWriter and writes two lines to the file scores.txt. Each line consists of a first name (a string), a middle-name initial (a character), a last name (a string), and a score (an integer).

PROGRAM 11-2

```
1  import java.io.*;
2  public class WriteData {
3    public static void main(String[] args) throws IOException{
4    File file = new File("scores.txt");
5    if (file.exists()) {
6     System.out.println("File already exists");
7     System.exit(1);
8    }
9    PrintWriter output = new PrintWriter(file);
10
11   // Write formatted output to the file
12   output.print("John T Smith ");
13   output.println(90);
14   output.print("Eric K Jones ");
15   output.println(85);
16
17   output.close();
18  }
19 }
```

Lines 4–7 check whether the file scores.txt exists. If so, exit the program (line 6). **Invoking the constructor of PrintWriter will create a new file if the file does not exist. If the file already exists, the current content in the file will be discarded without verifying with the user. Invoking the constructor of PrintWriter may throw an IO exception.**

It is important to note that when you are doing file IO or network IO, something could go wrong. The file might not exist, it could be on a bad sector of the disk, the network could crash halfway through. These types of exception need to be checked. **Java forces you to write the code to deal with this type of exception. For simplicity, we declare throws IOException in the `main` method header (line 2).**

The reason that you need to do something about the IO Exception is that it is a checked exception. If you call a constructor or a function that throws a checked exception then you either

need to handle it, by catching it and taking appropriate actions. Or you need to tell the compiler that you know about the exception, but you do not plan to do anything about it, in which case you must declare throws IO Exception in your function definition.

Alternatively, you can use a `try.. catch` block to handle the throwing an exception. See PROGRAM 11-3

PROGRAM 11-3

```
1   import java.io.*;
2   public class FileIOWriteDemo {
3       public static void main(String[] args) {
4         try{
5            PrintWriter output
6                 = new PrintWriter(new FileOutputStream("stuff.txt"));
7
8            output.println("The quick brown fox");
9            output.println("jumped over the lazy dog.");
10           output.close();
11           System.out.println("File writing successfully completed.");
12
13        }
14        catch(IOException ex){
15              System.out.println("Error" + ex.getMessage());
16        }
17     }
   }
```

A file output stream is an output stream for writing data to a File, in this case, the file is a text file with name `"stuff.txt"`. **A stream is an object that allows for the flow of data between your program and some IO device or some file.**

- Input stream: data flow into your program (e.g. from keyboard, or file)
- Output stream: data flow out of your program (e.g to screen, or file)

Output streams are buffered, meaning that data is saved in a temporary location, known as a buffer first. When enough data is accumulated in the buffer, it is physically written to the file. If you study the Java API for PrintWriter, you will notice there exists a method named `flush()`.This method will cause a physical write to the file of any buffered data. The method `close()` also includes an invocation to `flush()`.

`System.out` is a standard Java object for the console output. You can create PrintWriter objects for writing text to any file using `print()`, `println()`, and `printf()` (lines 13–16). **The `close()` method must be used to close the file (line 19)**. If this method is not invoked, the data may not be saved properly in the file. **When the `close()` method is invoked, the system releases any resources used to connect the stream to the file and does any other housekeeping that is needed.** If your program does not close a file before the program ends, Java will close it for you when the program ends, but it is safest to close the file with an explicit call to close in case your Java program ends abnormally.

11.2.3 Appending File

So far, we have written the desired text to a file. If the file exists, we would have overwritten the contents of the existing file. Sometimes you might want to append text to the end of an existing file instead. You can set the writing mode to “append” as such:

```
output = new PrintWriter(new FileOutputStream("stuff.txt", true));
```

11.3 Reading Data Using Scanner

The `java.util.Scanner` class was used to read strings and primitive values from the console. A Scanner breaks its input into tokens delimited by whitespace characters. To read from the keyboard, you create a Scanner for `System.in`, as follows:

```
Scanner input = new Scanner(System.in);
```

To read from a file, create a Scanner for a file, as follows:

```
Scanner input = new Scanner(new File(filename));
```

java.util.Scanner	Description
+Scanner(source: File)	Creates a Scanner that scans tokens from the specified file.
+Scanner(source: String)	Creates a Scanner that scans tokens from the specified string.
+close()	Closes this scanner.
+hasNext(): boolean	Returns true if this scanner has more data to be read.
+next(): String	Returns next token as a string from this scanner.
+nextLine(): String	Returns a line ending with the line separator from this scanner.
+nextByte(): byte	Returns next token as a byte from this scanner.
+nextShort(): short	Returns next token as a short from this scanner.
+nextInt(): int	Returns next token as an int from this scanner.
+nextLong(): long	Returns next token as a long from this scanner.
+nextFloat(): float	Returns next token as a float from this scanner.
+nextDouble(): double	Returns next token as a double from this scanner.
+useDelimiter(pattern: String):	Sets this scanner's delimiting pattern and returns this scanner.

PROGRAM 11-4 gives an example that creates an instance of Scanner and reads data from the file scores.txt.

PROGRAM 11-4

```
1  import java.util.Scanner;
2  import java.io.*;
3  public class ReadData {
4    public static void main(String[] args) throws Exception {
5      File file = new File("scores.txt");
6      Scanner input = new Scanner(file); // Create a Scanner for the file
7      while (input.hasNext()) { // Read data from a file
8         String firstName = input.next();
9         String mi = input.next();
10        String lastName = input.next();
11        int score = input.nextInt();
12        System.out.println(
13        firstName + " " + mi + " " + lastName + " " + score);
14     }
15     input.close();
16   }
17 }
```

Note that `new Scanner(String)` creates a Scanner for a given string. To create a Scanner to read data from a file, you have to use the `File` class from java.io to create an instance of the File using the constructor `new File(filename)` (line 4), and use `new Scanner(File)` to create a Scanner for the file (line 5).

Invoking the constructor `new Scanner(File)` may throw an IO exception, so the `main` method declares throws Exception in line 3. Each iteration in the while loop reads the first name, middle initial, last name, and score from the text file (lines 7–10). The file is closed in line 14.

11.4 Reading Data Using BufferReader

If the file to read does not exist, the program will display the message as shown in Program Output 1. Otherwise, it will print the content of the file, see Program Output 2.

PROGRAM 11-4

```
1  import java.io.*;
2  import java.util.*;
3
4  public class FileIOReadDemoBufferedReader {
5
6      public static void main(String[] args) {
7        try{
8              BufferedReader inputStream =
9                      new BufferedReader(new FileReader("stuff.txt"));
10
11             String line = "dummy";
12             do {
13                   line = inputStream.readLine();
14                   if (line!=null) System.out.println(line);
15             }while (line!=null);
16
17             inputStream.close();
18
19             System.out.println("File reading successfully completed.");
20
21       }
22       catch(Exception ex){
23             System.out.println("Error " + ex.getMessage());
24       }
25    }
26 }
```

PROGRAM OUTPUT 1 (If file is not found in directory)

```
Error stuff.txt (The system cannot find the file specified)
```

PROGRAM OUTPUT 2 (If file exist, then content will be printed)

```
The quick brown fox
jumped over the lazy dog.
File reading successfully completed.
```

11.5 Tokenizing a String

Tokenization is the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements called **tokens**. In Java, you may use the .split() method in the String class to tokenizing the String (refer to PROGRAM 11-5).

PROGRAM 11-5

```
1  import java.io.BufferedReader;
2  import java.io.FileReader;
3  import java.io.IOException;
4
5  public class StringToken {
6      public static void main(String[] args) {
7          BufferedReader br = null;
8          try {
9              String line;
10             br = new BufferedReader(new FileReader("test.csv"));
11             while ((line = br.readLine()) != null) {
12                 System.out.println(line);
13                 String[] tokens = line.split("[|]");
14                 for(String i: tokens){
15                     System.out.println(i);
16                 }
17             }
18             br.close();
19         } catch (IOException e) {
20             e.printStackTrace();
21         }
22
23     }
24 }
```

In this example, we would like to split the content in String **`line`** by the delimiter |.
Thus in line 13, you invoke **`line.split("[|]");`** to delimit the content in line by |.

Content in test.csv

```
5|4.8|Apple
3|2.3|Orange
2|9.7|Watermelon
```

PROGRAM OUTPUT

```
5|4.8|Apple
5
4.8
Apple
3|2.3|Orange
3
2.3
Orange
2|9.7|Watermelon
2
9.7
Watermelon
```

Another possible way to tokenize a String is to use a Scanner.

PROGRAM 11-6

```
1  import java.io.BufferedReader;
2  import java.io.FileReader;
3  import java.io.IOException;
4  import java.util.Scanner;
5
6  public class StringToken2 {
7      public static void main(String[] args) {
8          BufferedReader br = null;
9          try {
10             String line;
11             br = new BufferedReader(new FileReader("test.csv"));
12             while ((line = br.readLine()) != null) {
13                 System.out.println(line);
14                 Scanner lineScanner = new Scanner(line);
15                 lineScanner.useDelimiter("|");
16                 while (lineScanner.hasNext()) {
17                     System.out.print(lineScanner.next());
18                 }
19             }
20             br.close();
21         } catch (IOException e) {
22             e.printStackTrace();
23         }
24     }
25 }
```

In line 14, create a Scanner object and pass in the String to be tokenized.
In line 15, invoke the useDelimiter() method to tokenize the String. The delimiter will be passed in as a String into the method.
Line 16 to 18 uses a while look to loop through the tokenized data. `lineScanner.hasNext()` will be true if there exists a next token.

11.6 JFileChooser class

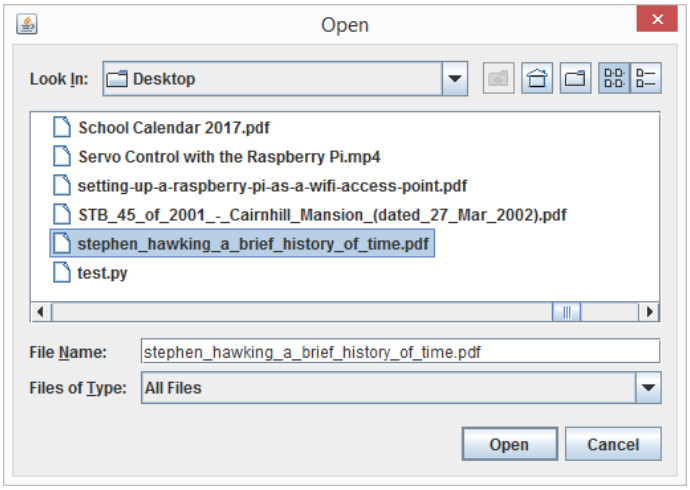
A file chooser provides a simple mechanism for the user to choose a file. File choosers provide a GUI for navigating the file system, and then allows either choosing a file, or entering the name of a file to be saved. However, note that JFileChooser belongs to Java Swing API. We will discuss about JFileChooser here, for a file dialog with a better aesthetic feel, using Java FX will be more appropriate (which will be discussed in chapter 13).

11.6.1 File Dialog() – Open a File

PROGRAM 11-7 shows how to use the JFileChooser to get the absolute path for the file the user wants to open or to get the location where the user wants to save the file:

PROGRAM 11-7

```
1  import java.io.File;
2  import javax.swing.JFileChooser;
3  import javax.swing.filechooser.FileSystemView;
4
5  public class FileChooser1 {
6
7    public static void main(String[] args) {
8      JFileChooser jfc = new
9  JFileChooser(FileSystemView.getFileSystemView().getHomeDirectory());
10     int returnValue = jfc.showOpenDialog(null);
11     if (returnValue == JFileChooser.APPROVE_OPTION) {
12        File selectedFile = jfc.getSelectedFile();
13        System.out.println(selectedFile.getAbsolutePath());
14     }
15   }
16 }
```



PROGRAM OUTPUT

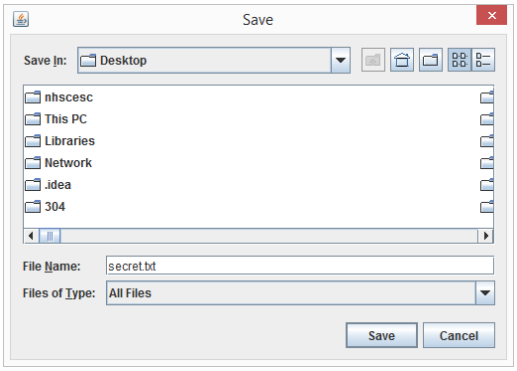
```
C:\Users\nhscesc\Desktop\stephen_hawking_a_brief_history_of_time.pdf
```

11.6.2 File Dialog() – Save a File

PROGRAM 11-8 shows how to use the JFileChooser to get the absolute path for the file the user wants to save or to get the location where the user wants to save the file:

PROGRAM 11-8

```
1  import java.io.File;
2  import javax.swing.JFileChooser;
3  import javax.swing.filechooser.FileSystemView;
4  public class FileChooser2 {
5    public static void main(String[] args) {
6      JFileChooser jfc = new
7      JFileChooser(FileSystemView.getFileSystemView().getHomeDirectory())
8  ;
9      int returnValue = jfc.showSaveDialog(null);
10     if (returnValue == JFileChooser.APPROVE_OPTION) {
11       File selectedFile = jfc.getSelectedFile();
12       System.out.println(selectedFile.getAbsolutePath());
13     }
14   }
   }
```



PROGRAM OUTPUT

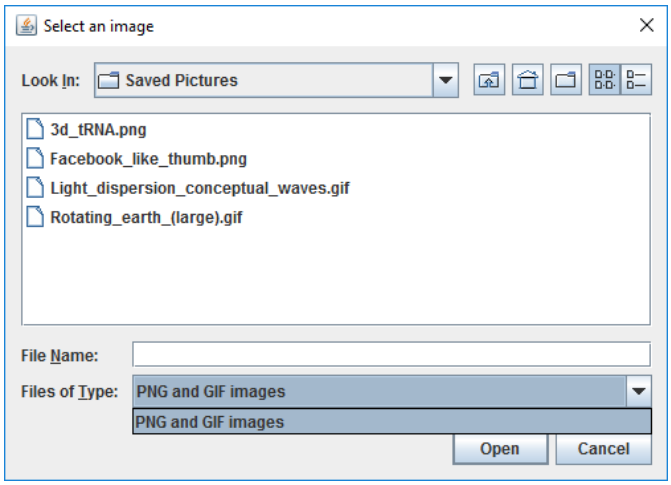
```
C:\Users\nhscesc\Desktop\secret.txt
```

11.6.3 Filters – Limit the set of files shown to the user

It is always handy to limit user selection to the program's needs. If for example your program requires png and gif images, it would be good practice to limit the user's selection to only that. PROGRAM 11-9 below shows how to achieve it using a custom FileNameExtensionFilter:

PROGRAM 11-9

```
1   import javax.swing.JFileChooser;
2   import javax.swing.filechooser.FileNameExtensionFilter;
3   import javax.swing.filechooser.FileSystemView;
4   public class FileChooser3 {
5     public static void main(String[] args) {
6      JFileChooser jfc = new
7      JFileChooser(FileSystemView.getFileSystemView().getHomeDirectory());
8      jfc.setDialogTitle("Select an image");
9      jfc.setAcceptAllFileFilterUsed(false);
10     FileNameExtensionFilter filter = new FileNameExtensionFilter("PNG
11  and
12     GIF images", "png", "gif");
13     jfc.addChoosableFileFilter(filter);
14     int returnValue = jfc.showOpenDialog(null);
15     if (returnValue == JFileChooser.APPROVE_OPTION)
16        System.out.println(jfc.getSelectedFile().getPath());
17    }
    }
```



[Reference]

[1] Introduction to Java Programming Comprehensive Version 10th Ed, Daniel Liang, 2016.
[2] https://www.mkyong.com/swing/java-swing-jfilechooser-example/

.