8. File Handling and I/O

8.1 Working with Files

The **java.io** package includes a **File** class that allows you to work with files. To start, create a **File** object and specify the path of the file in the constructor.

Escape Sequences

A character preceded by a backslash (\) is an *escape sequence* and has special meaning to the compiler. The following table shows the Java escape sequences:

Escape Sequence	Description
`\t`	Insert a tab in the text at this point.
`\b`	Insert a backspace in the text at this point.
`\n`	Insert a newline in the text at this point.
`\r`	Insert a carriage return in the text at this point.
`\f`	Insert a formfeed in the text at this point.
`\'`	Insert a single quote character in the text at this point.
`\"`	Insert a double quote character in the text at this point.
`\\`	Insert a backslash character in the text at this point.

```
import java.io.File;
...
File file = new File("C:\\data\\input-file.txt");
```

With the **exists**() method, you can determine whether a file exists.

```
import java.io.File;

public class MyClass {
 public static void main(String[ ] args) {
   File x = new File("C:\\myFolder\\test.txt");
   if(x.exists()) {
    System.out.println(x.getName() +  "exists!");
   }
   else {
    System.out.println("The file does not exist");
   }
 }
}
```

The code above prints a message stating whether or not the file exists at the specified path.

Note:
The **getName**() method returns the name of the file.
Note that we used double backslashes in the path, as one backslash should be escaped in the path String.

Q: Fill in the blanks to determine whether the file exists.

```
class A {
   public static void main(String args[ ]) {
   File file = _____ File("a.txt");
   if(file._______()) {
     System.out.println("Yes");
   }
}
}
```

8.2 Reading a File

Files are useful for storing and retrieving data, and there are a number of ways to read from a files.

One of the simplest ways is to use the **Scanner** class from the **java.util** package.
The constructor of the **Scanner** class can take a **File** object as input.

To read the contents of a text file at the path "C:\\myFolder\\test.txt", we would need to create a File object with the corresponding path and pass it to the Scanner object.

```
try {
 File x = new File("C:\\myFolder\\test.txt");
 Scanner sc = new Scanner(x);
}
catch (FileNotFoundException e) {

}
```

Note:
We surrounded the code with a try/catch block, because there's a chance that the file may not exist.

Q: Which class can be used for reading files?

- HashMap
- Set
- Scanner
- ArrayList

Reading a File

The **Scanner** class inherits from the **Iterator**, so it behaves like one.
We can use the Scanner object's **next**() method to read the file's contents.

```
try {
 File x = new File("C:\\myFolder\\test.txt");
 Scanner sc = new Scanner(x);
```

```
while(sc.hasNext()) {
  System.out.println(sc.next());
 }
 sc.close();
} catch (FileNotFoundException e) {
 System.out.println("Error");
}
```

The file's contents are output word by word, because the **next**() method returns each word separately.

Note:
It is always good practice to close a file when finished working with it. One way to do this is to use the Scanner's **close**() method.

Q: Fill in the blanks to read and print the content of the file a.txt, and then close it.

```
try {
   File f = new File("a.txt");
  Scanner sc = new _________(f);
  while (sc.hasNext()) {
      String a = ___.next();
      String b = sc._____();
      System.out.println(a + " " + b);
  }
  sc._______();
}
catch (Exception e) {
   System.out.println("Error");
}
```

8.3 Creating & Writing Files

Creating Files

Formatter, another useful class in the java.util package, is used to create content and write it to files.

Example:

```
import java.util.Formatter;

public class MyClass {
  public static void main(String[ ] args) {
 try {
   Formatter f = new Formatter("C:\\myFolder\\test.txt");
 } catch (Exception e) {
     System.out.println("Error");
 }
 }
}
```

This creates an empty file at the specified path. If the file already exists, this will overwrite it.

Note:

Again, you need to surround the code with a **try/catch** block, as the operation can fail.

Q: Which class is used to write content to files?

- Set
- Formatter
- ArrayList
- Scanner

Writing to Files

Once the file is created, you can write content to it using the same Formatter object's **format**() method.

Example:

```
import java.io.File;
```

```
import java.util.Scanner;
import java.util.Formatter;
Import java.io.File;

public class MyClass {
   public static void main(String[ ] args) {
      try {
         Formatter f = new Formatter("test.txt");
         f.format("%s %s %s", "1","John", "Smith \r\n");
         f.format("%s %s %s", "2","Amy", "Brown");
         f.close();

         File x = new File("test.txt");
         Scanner sc = new Scanner(x);
         while(sc.hasNext()) {
            System.out.println(sc.next());
         }
         sc.close();
      } catch (Exception e) {
      System.out.println("Error");
      }
   }
}
```

The **format**() method formats its parameters according to its first parameter.
%s mean a string and get's replaced by the first parameter after the format. The second %s get's replaced by the next one, and so on. So, the format **%s %s %s** denotes three strings that are separated with spaces.

Note: **\r\n** is the newline symbol in Windows.
The code above creates a file with the following content:

```
1 John Smith
2 Amy Brown
```

Note:

Don't forget to **close** the file once you're finished writing to it!

Q: Rearrange the code to write “Hi there” to the file.

1. Formattrer f = new Formatter(“a.txt”);
2. f.format(“%s”, “Hi”);
3. f.close();
4. f.format(“%s”, “there”);