OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

# CT326 Programming III

**LECTURE 16**

**RANDOM FILE ACCESS**

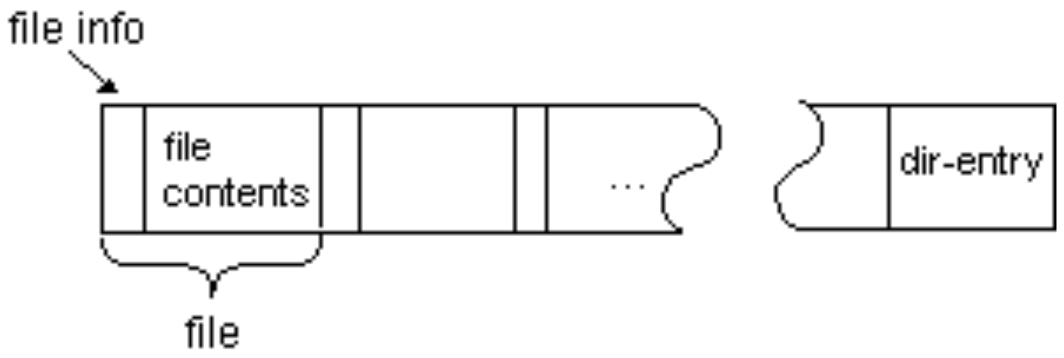
**DR ADRIAN CLEAR**
**SCHOOL OF COMPUTER SCIENCE**

# Random File Access

- Working with Random Access Files
  - The input and output streams in this lesson so far have been sequential access streams:
    - Streams whose contents must be read or written sequentially.
  - While still incredibly useful, sequential access files are a consequence of a sequential medium such as magnetic tape.
  - Random access files, on the other hand, permit non-sequential, or random, access to the contents of a file.

# Random File Access use case

- Consider the archive format known as "zip."
  - Zip archives contain files and are typically compressed to save space.
- Zip archives also contain a dir-entry at the end that indicates where the various files contained within the zip archive begin:



## Random File Access use case

- Suppose that you want to extract a specific file from a zip archive.
- If you use a sequential access stream, you have to do the following:
  - Open the zip archive.
  - Search through the zip archive until you located the file you wanted to extract.
  - Extract the file.
  - Close the zip archive.
- On average, using this algorithm, you'd have to read half the zip archive before finding the file that you wanted to extract.

# Random File Access use case

- You can extract the same file from the zip archive more efficiently using the seek feature of a random access file:
  - Open the zip archive.
  - Seek to the dir-entry and locate the entry for the file you want to extract from the zip archive.
  - Seek (backwards) within the zip archive to the position of the file to extract.
  - Extract the file and close the zip archive.
- This algorithm is more efficient because you only read the dir-entry and the file that you want to extract.

## `RandomAccessFile` class

- The `RandomAccessFile` class in the java.io package implements a random access file.
- Unlike the input and output stream classes in java.io, `RandomAccessFile` is used for both reading and writing files.
  - The `RandomAccessFile` class implements both the `DataInput` and `DataOutput` interfaces and therefore can be used for both reading and writing.
- You create a `RandomAccessFile` object with different arguments depending on whether you intend to read or write.

## Using `RandomAccessFile`

- `RandomAccessFile` is similar to `FileInputStream` and `FileOutputStream` in that you specify a file on the native file system to open.
  - You can do this with a filename or a `File` object.
- When you create a `RandomAccessFile`, you must indicate whether you will be just reading the file or also writing to it.
- The code creates a `RandomAccessFile` to read the file named farrago.txt:

```
new RandomAccessFile("farrago.txt", "r");
```

## Using `RandomAccessFile`

- This code opens the same file for both reading and writing:

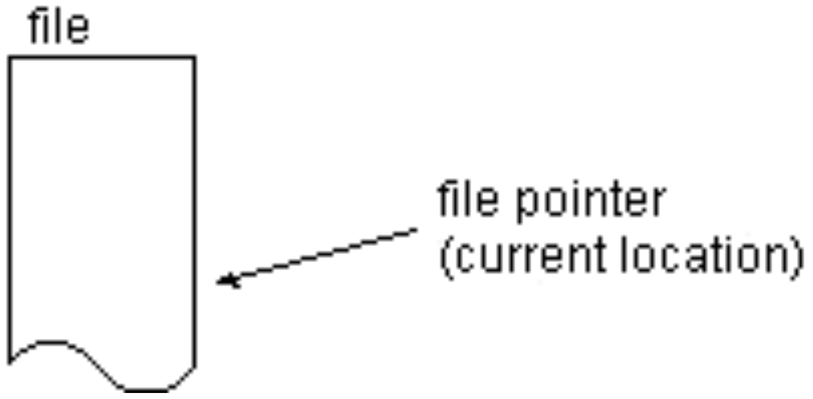
```
new RandomAccessFile("farrago.txt", "rw");
```

- After the file has been opened, you can use the common readXXX or writeXXX methods to perform I/O on the file.
- The `RandomAccessFile` class supports the notion of a file pointer.
  - This pointer indicates the current location in the file.
- When the file is first created, the file pointer is 0, indicating the beginning of the file.

# Using `RandomAccessFile`

- Calls to the readXXX and writeXXX methods adjust the file pointer by the number of bytes read or written e.g.

```
int data = file.readInt();
file.writeInt(data);
```



## Using `RandomAccessFile`

- In addition to the normal file I/O methods that implicitly move the file pointer when the operation occurs, `RandomAccessFile` also contains three methods for explicitly manipulating the file pointer.
  - `skipBytes()` - moves the file pointer forward the specified number of bytes.
  - `seek()` - positions the file pointer just before the specified byte.
  - `getFilePointer()` - returns the current byte location of the file pointer.

# Using `RandomAccessFile`

- `RandomAccessFile` is somewhat disconnected from the input and output streams in java.io - it doesn't inherit from the `InputStream` or `OutputStream`.
- This has some disadvantages in that you can't apply the same filters to `RandomAccessFile`s that you can to streams.
- However, `RandomAccessFile` does implement the `DataInput` and `DataOutput` interfaces:
  - If you design a filter that works for either `DataInput` or `DataOutput`, it will work on any `RandomAccessFile`.

## Next time…

- Collections