PersistentArray

**+ Create(arrayName : String, elementSize : int, userHeaderSize : int) : void**

let filePath be the call to GetPathToArrayName passing in arrayName

if file with filePath exists,

throw new ParameterException

let raf be the call to openRandomAccessFile passing in filePath

initialize file passing in raf, elementSize, and userHeaderSize

- GetPathToArrayName(arrayName : String)

return DB\_FOLDER + arrayName + “.dat”

- InitFile(RandomAccessFile : file, userHeaderSize : int, elementSize : int) : void

Seek to beginning of storage

Write to file userHeaderSize

Write to file elementSize

Write to file userHeaderSize

**+ Delete(arrayName : String) : void**

Let filename be the path to arrayName

if the file in filepath already exists,

delete the file in filepath

**+ Open(arrayName : String) : PersistentArray**

Let filename be the path to arrayName

if the file in filepath already exists,

let result be the call to PersistentArray’s ctor

return result

otherwise the file in filepath doesn’t exist,

throw PersistentArrayNotFoundException

//Assumes the file has been created and is initialized

- PersistentArray(filePath : String)

let storage be the result of the call to OpenRandomAccessFile passing in filePath

call refreshUserHeader()

- OpenRandomAccessFile(filename : String) : RandomAccessFile

Let file be the result of the constructor call to RandomAccessFilefile name and read/write permissions

return file

- refreshUserHeader()

Call SeekToBOF()

let header’s userHeaderSize be storage.read()

let header’s elementSize be storage.read()

let header’s nextIdex be storage.read()

let userHeader be a new byte[] of size userHeaderSize

call storage.read() passing in userHeader

let header’s userHeader be the userHeader variable

**+ Close() : void**

Call close() on storage

**+ GetUserHeaderSize() : int**

Return the call getUserHeaderSize() on header

**+ GetElementSize() : int**

Return the call getElementSize on header

- SeekToBOF() : void

Call seek() passing in 0 on storage

- SeekToFirstElement() : void

Call seek() on storage passing in the call to header’s getUserHeaderSize()

- SeekToElement(index : int) : void

Call seek() on storage passing in the call to header’s getElementSize() \* index + header’s getUserHeaderSize()

**+** **GetUserHeader() : byte[]**

Return the call getUserHeader() on header

**+** **GetNextIndex() : int**

Return getNextIndex() on header

**+ Get(elementIndex : int) : byte[]**

Let elementSize be the result of GetElementSize()

Call SeekToElement() passing elementIndex

Let result be a byte[] of size elementSize

call storage.read() passing in result

Return result

**+ Put(elementIndex : int, buffer : byte[]) : void**

Let elementSize be the result of GetElementSize()

If buffer size is greater than elementSize,

Throw a ParameterException

Otherwise buffer size is less than or equal to elementSize,

Call seekToElement() passing elementIndex

Call fillSpaceWithBuffer() passing in buffer and elementSize

**+** **PutUserHeader(userHeader : byte[]) : void**

Call updateUserHeader() passing in userHeader

Call setUserHeader() on header passing in userHeader

- updateUserHeader(userHeader : byte[]) : void

Let userHeaderSize be the call to getUserHeaderSize on header

If userHeader’s length is greater than userHeaderSize,

Throw a ParameterException

Call seekToUserHeader()

Call fillSpaceWithBuffer() passing in userHeader and userHeaderSize

- fillSpaceWithBuffer(buffer : byte[], spaceToFill : int) : void

Call write() on storage passing in buffer

Call writeZeros passing in userHeaderSize – buffer’s size

- writeZeros(count : int) : void

If count is less than 0,

Throw ParameterException

For 0 to count…

Call write() on storage

Header

**+ ctor(elementSize : int, headerData : byte[])**

If elementSize is less than or equal to 0,

Throw ParameterException

Otherwise elementSize is greater than 0,

Let this.elementSize be elementSize

Let this.nextIndex be 0

Let userHeader be a new UserHeader object

Let userHeader’s data be headerData

**+ getElementSize() : int**

Return elementSize

**+ getHeaderData() : byte[]**

Return headerData

**+ getHeaderDataSize() : int**

Return headerData’s size

**+ getHeaderSize() : int**

Return getHeaderDataSize() + 3

**+ setElementSize(elementSize : int) : void**

If elementSize is less than or equal to 0,

Throw ParameterException

Let this.elementSize be elementSize

**+ setHeaderData(data : byte[]) : void**

Let userHeader’s data be data

UserHeader