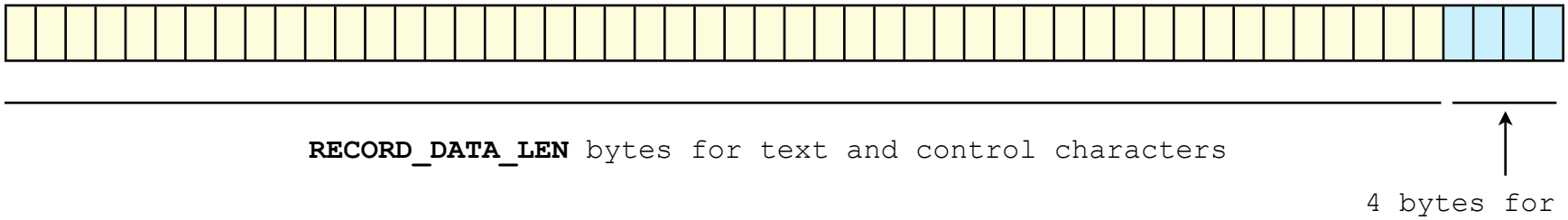
Record.java class

The record structure as outlined in this class is for the records stored in the **_messages.dat** file:



methods

public int readFromMessagesFile(int recordNumber)

Method that reads a record from **_messages.dat,** from a given record number position, and puts the record in the **data** byte array. The method returns a integer value that denotes whether the process was successful. The method also reads the integer for **next**

```
public int readFromMessagesFile(int recordNumber)
     move the file pointer to the beginning of the record recordNumber (you will need the
     constant RECORD_LEN to calculate the correct position)

     read the record with the following statement: int bytes = Globals.msg.read(data);
     bytes will contain the number of bytes that have been read from the file and
     the record will be placed in the data array

     read the integer that we store at the end of every record
          next = Globals.msg.readInt();
```

```
     if all went well, return PROCESS_OK
     else return PROCESS_ERROR
```

```
public int writeToMessagesFile(int recordNumber, int mode)
```

Method that writes a record to **_messages.dat**. The data is in the **data** array and is written in the record position recordNumber.

Parameter **mode** can take on one of two values:

```
     APPEND
     MODIFY
```

These need to be added to **Globals.java** as constants with the values 1 and 2 respectively. This parameter will determine whether the record will be added to the file or an existing record will be updated.

```
public int writeToMessagesFile(int recordNumber, int mode)
     move the file pointer to the beginning of the record recordNumber
     We write the entire data array on a single command: Globals.msg.write(data);
     Write the integer next with the method Globals.msg.writeInt(next);

     if mode equals APPEND then increment Globals.totalRecordsInMessageFile by 1

     if all went well, return PROCESS_OK
     else return PROCESS_ERROR
```

```
public String toString()
     return data as a string (use the method getData()) concatenated with next
```

EmailServer.java Testing the read/write methods of the Record class

test 1a

Write a main method in the EmailServer class to write a record to the messages file

```
main() method {
     open the messages file
     if file opens successfully {
          declare and instantiate an object of the Record class
               set the string parameter to a sentence of RECORD_DATA_LEN chars or less
               let the integer parameter be any integer

          make a call to the method writeToMessagesFile(0, Globals.APPEND)
          close the messages file
     }
     else {
          print error message
     }
}
```

Check that the messages file is exactly RECORD_LEN bytes long

test 1b

Write a main method in the EmailServer class to read a record from the messages file

```
main() method {
     open the messages file
     if file opens successfully {
          declare and instantiate a default object of the Record class
          make a call to the method readFromMessagesFile(0)
          print the contents of the record
          close the messages file
```

```
     }
     else {
          print error message
     }
}
```

test 2a

Modify the main method of test 1a so that at least 10 records are written to the messsages file. Note the following:

a) No direct calls to any file input/output methods should be made here. The method writeToMessagesFile() must handle the writing.

b) Be sure to delete the messages file prior to running this test. After running the test, check the size of the file. It should be a multiple of RECORD_LEN

test 2b

Modify the main method of **test 1b** so that all records of the messages file are read and printed on the screen. All that is required here is a loop that reads one record at a time and then its contents are printed. Once again, no direct calls to any file input/output methods should be made here. The method readFromMessagesFile() must handle the writing.