```python
"""A module containing FileRequest and FileResponse which are
subclasses of Record, which is a subclasses of Packet
(as defined in packet.py).  FileRequest and FileResponse arrange
packets (bytearrays) in the arrangement specific to each type.

The following is the inheritance tree.

          Packet
            |
            V
          Record
          |    |
          V    V
FileRequest    FileResponse

"""

from collections import OrderedDict
from packet import Packet
import math

BYTE_LEN = 8
BLOCK_SIZE = 4096
ENCODING_TYPE = "UTF-8"

FILE_REQUEST_MAGIC_NO = 0x497E   # Is it in network byte order?
FILE_REQUEST_TYPE = 1
MAX_FILENAME_LEN = 1024

FILE_RESPONSE_MAGIC_NO = 0x497E
FILE_RESPONSE_TYPE = 2



class Record(Packet):
    """A class between Packet and FileRequest, FileResponse
    that adds the header and payload to the internal
    bytearray (via append in Packet class)."""


    def __init__(self, header_dict, payload_bytes):
        payload_bit_len = len(payload_bytes) * BYTE_LEN

        packet_bit_len = sum(bit_len for bit_len, _ in header_dict.values()) \
            + payload_bit_len
        super().__init__(packet_bit_len)  # Call constructor of Packet
        self._append_header_dictionary(header_dict)
        self._append_payload(payload_bytes)

    def _append_header_dictionary(self, dictionary):
        """Helper function that takes an OrderedDict of values
        (bit_len, value).  Adds each value to the internal
        bytearray in order."""
        for bit_len, value in dictionary.values():
            self.append(value, bit_len)

    def _append_payload(self, payload_bytearray):
        """Takes each byte from payload_bytearray and adds
        it to the internal bytearray."""
        for byte in payload_bytearray:
            self.append(byte, BYTE_LEN)



class FileRequest(Record):
    '''An object that creates a bytearray representing a
    FileRequest.  Includes static methods for checking
    the validity of a recieved FileRequest, and extracting
    relevent fields.  The payload is of size FilenameLen bytes.
    includes a static OrderedDict of the header with structure:
    "FieldName" : [BitLen, Value]
    (Value is None for dynamic fields)
    By length in bits, a FileRequest header has the following
    fields:
    "MagicNo", 16
    "Type", 8
    "FilenameLen", 16
    ""
    '''

    HEADER_DICT = OrderedDict((
                ("MagicNo", [16, FILE_REQUEST_MAGIC_NO]),
                ("Type", [8, FILE_REQUEST_TYPE]),
                ("FilenameLen", [16, None]),
            ))

    def __init__(self, file_name):
        """Takes a filename string."""
        file_name_bytes = file_name.encode(ENCODING_TYPE)

        self.HEADER_DICT["FilenameLen"][-1] = len(file_name_bytes)

        super().__init__(self.HEADER_DICT, file_name_bytes)


    @staticmethod
    def get_filenameLen_from_header(packet_bytearray):
        """Takes a bytearray representing a FileRequest
        header.  Extracts the filenameLen."""
        pkt = Packet(len(packet_bytearray)*BYTE_LEN, packet_bytearray)
        try:
            MagicNo_len = FileRequest.HEADER_DICT["MagicNo"][0]
            Type_len = FileRequest.HEADER_DICT["Type"][0]
            FilenameLen_len = FileRequest.HEADER_DICT["FilenameLen"][0]
            FilenameLen = pkt.get_from_bits(
                1+MagicNo_len+Type_len, MagicNo_len+Type_len+FilenameLen_len
            )
            return FilenameLen
        except IndexError:
            raise ValueError("Invalid FileRequest header")


    @staticmethod
    def is_valid_header(packet_bytearray):
        """Takes a bytearray is checks if it is a valid FileRequest
        header.  For the method to return True:
        MagicNo == 0x497E,
        Type == 1,
        1 <= FilenameLen <= 1,024
        """
        is_valid = True
        pkt = Packet(len(packet_bytearray)*BYTE_LEN, packet_bytearray)

        MagicNo_len = FileRequest.HEADER_DICT["MagicNo"][0]
        Type_len = FileRequest.HEADER_DICT["Type"][0]
        FilenameLen_len = FileRequest.HEADER_DICT["FilenameLen"][0]

        MagicNo = pkt.get_from_bits(0, MagicNo_len)
        if MagicNo != FILE_REQUEST_MAGIC_NO:
            is_valid = False

        Type = pkt.get_from_bits(1+MagicNo_len, MagicNo_len+Type_len)
        if Type != FILE_REQUEST_TYPE:
            is_valid = False

        FilenameLen = pkt.get_from_bits(
            1+MagicNo_len+Type_len, MagicNo_len+Type_len+FilenameLen_len
        )
        if not (1 <= FilenameLen <= MAX_FILENAME_LEN):
            is_valid = False

        return is_valid


    @staticmethod
    def header_bit_len():
        """Returns the len of the header in bits."""
        try:
            return sum(bit_len for bit_len, _ in FileRequest.HEADER_DICT.values())
        except NameError:
            return 0

    @staticmethod
    def header_byte_len():
        """Returns the len of the header in bits."""
        return math.ceil(FileRequest.header_bit_len() / BYTE_LEN)


class FileResponse(Record):
    '''An object that creates a bytearray representing a
    FileResponse.  Includes static methods for checking
    the validity of a recieved FileResponse, and extracting
    relevent fields.  The payload is of size DataLen bytes.
    Includes a static OrderedDict of the header with structure:
    "FieldName" : [BitLen, Value]
    (Value is None for dynamic fields)

    By length in bits, a FileRequest header has the following
    fields:
    "MagicNo", 16
    "Type", 8
    "StatusCode", 8
    "DataLen", 32
    ""

    FileResponse is initialized with a file_name and a status
    code.  Unlike FileRequest, FileResponse does not read
    the payload into memory.  If .get_bytearray() is called,
    the whole file is read from file_name and the header and
    file are returned as a bytearray.  The method
    .read_byte_block() is an itterator that opens a file handle
    on file_name and returns an amount of bytes equal to
    BLOCK_SIZE.  First returns header + a part of the file as a
    bytearray, then returns an amount of the file equal to
    BLOCK_SIZE on each following itteration.  When the whole
    file is transfered, the file handle is closed.
    '''

    HEADER_DICT = OrderedDict((
            ("MagicNo", [16, FILE_RESPONSE_MAGIC_NO]),
            ("Type", [8, FILE_RESPONSE_TYPE]),
            ("StatusCode", [8, None]),
            ("DataLen", [32, None]),
        ))

    def __init__(self, file_name, status_code):
        """Takes a bytearray that containing a file, and a
        integer status_code.  If status_code == 0 then
        no payload is written to the packet."""

        self.HEADER_DICT["StatusCode"][-1] = status_code
        self.HEADER_DICT["DataLen"][-1] = self._file_len()

        super().__init__(self.HEADER_DICT, bytearray(0))



    @staticmethod
    def get_status_DataLen(packet_bytearray):
        """Takes a bytearray representing a FileRequest
        header.  Returns (StatusCode, DataLen)."""
        pkt = Packet(len(packet_bytearray)*BYTE_LEN, packet_bytearray)
        try:
            MagicNo_len = FileResponse.HEADER_DICT["MagicNo"][0]
            Type_len = FileResponse.HEADER_DICT["Type"][0]
            StatusCode_len = FileResponse.HEADER_DICT["StatusCode"][0]
            DataLen_len = FileResponse.HEADER_DICT["DataLen"][0]
            StatusCode = pkt.get_from_bits(
                1 + MagicNo_len + Type_len,
                MagicNo_len + Type_len + StatusCode_len
            )
            DataLen = pkt.get_from_bits(
                1 + MagicNo_len + Type_len + StatusCode_len,
                MagicNo_len + Type_len + StatusCode_len + DataLen_len
            )
            return StatusCode, DataLen
        except IndexError:
            raise ValueError("Invalid FileRequest header")

    @staticmethod
    def is_valid_FileResponse(packet_bytearray):
        """Not implemented yet"""
        return True

    @staticmethod
    def header_bit_len():
        """Returns the len of the header in bits."""
        try:
            return sum(bit_len for bit_len, _ in FileResponse.HEADER_DICT.values())
        except NameError:
            return 0

    @staticmethod
    def header_byte_len():
        """Returns the len of the header in bits."""
        return math.ceil(FileResponse.header_bit_len() / BYTE_LEN)
```