```
"""A class that handles low level bit and byte operations on an
internal bytearray."""
import math

BYTE_LEN = 8


class Packet(object):
    """Packet object that stores information in a bytearray, and
    has a method append to add to the bytearray bit by bit.
    Handles one byte overflowing into the next byte."""
    def __init__(self, bit_length, packet_bytearray=None):
        """Create a packet object of size
        ceil(bit_length / BYTE_LEN) bytes.  Also can take an
        initial bytearray."""
        total_bytes = int(math.ceil(bit_length / BYTE_LEN))
        self.byte_array = bytearray(total_bytes)
        self._curr_bit = 0

        if packet_bytearray is not None:
            assert len(packet_bytearray) * BYTE_LEN <= bit_length
            for byte in packet_bytearray:
                self.append(byte, BYTE_LEN)


    def append(self, value, bit_len):
        """Add a certain abount of bits to the end of
        the internal bytearray."""
        empty_bits = BYTE_LEN - self._curr_bit % BYTE_LEN
        byte_num = self._curr_bit // BYTE_LEN

        bit_mask = ((1 << min(bit_len, empty_bits)) - 1)

        if bit_len <= empty_bits:
            bits_to_add = (value & bit_mask) << max(empty_bits - bit_len, 0)
            self.byte_array[byte_num] |= bits_to_add

            self._curr_bit += bit_len
        else:
            value_right_shift = max((BYTE_LEN - empty_bits) + (bit_len - BYTE_LEN),
0)
            bits_to_add = ((value >> value_right_shift) & bit_mask) >> max(empty_bit
s - bit_len, 0)
            self.byte_array[byte_num] |= bits_to_add

            self._curr_bit += empty_bits
            next_bit_mask = ((1 << (bit_len - empty_bits)) - 1)
            self.append(value & next_bit_mask, bit_len - empty_bits)
            #add bits to end of current byte
            #add carry over bits to next byte recursivly


    def get_bytearray(self):
        """return the internal bytearray."""
        return self.byte_array


    def get_from_bytes(self, byte_start, byte_end):
        """Takes two indicies in acending order.
        Returns an integer that is the number represented by
        the bytes in a bytearray in big-endian."""
        pkt = self.byte_array

        num = 0
        j = 0
        for i in reversed(range(byte_start, byte_end)):
            num += pkt[i] << j
            j += BYTE_LEN

        return num


    def get_from_bits(self, bit_start, bit_end):
```

```
        """Get the integer representing the bits in the range
        from bit_start to bit_end."""
        pkt = self.byte_array

        total_packet = self.get_from_bytes(0, len(pkt))
        total_bit_len = len(pkt) * BYTE_LEN
        total_packet &= (1 << total_bit_len-bit_start) - 1
        total_packet = total_packet >> (total_bit_len-bit_end)
        return total_packet
```