Stack

The following classes implements a **StackOfCharacters**.

```
1 public class StackOfCharacters {
2
3   private Character [] list;
4   public static final int DEFAULT_INITIAL_CAPACITY = 16;
5   private int size;
6   private int capacity;
7
8   public StackOfCharacters( int capacity )
9   {
10    if (capacity <= 0 ) capacity = DEFAULT_INITIAL_CAPACITY;
11    list = new Character [capacity];
12    size = 0;
13    this.capacity = capacity;
14  }
15
16  public StackOfCharacters( )
17  {
18    this(DEFAULT_INITIAL_CAPACITY);
19  }
20
21  /**
22  * Determines if this stack is empty or not.
23  * @return true if this stack is empty, false otherwise
24  */
25  public boolean empty ( )
26  {
27    // TODO: implement this method
28  }
29
30  public int getSize()
31  {
32    return size;
33  }
34
35  public int getCapacity()
36  {
37    return capacity;
38  }
39
40  /**
41  *  TODO: provide the Javadoc comments for this method
42  */
43  public void push ( Character value )
44  {
45    //WARNING: this implementation has a bug
46    if (  full() )
47      makeLarger();
48    if (value != null) {
49      list[size] = value;
50    }
51  }
52
53
54  /**
```

```
55    *  TODO: provide the Javadoc comments for this method
56    */
57   public Character pop()
58   {
59     if ( empty() ) return null;
60     else {
61       size--;
62       return list[size];
63     }
64   }
65
66   public Character peek()
67   {
68     if ( empty() ) return null;
69     else {
70       return list[size-1];
71     }
72   }
73
74   private boolean full()
75   {
76     if (size == capacity )
77       return true;
78     else
79       return false;
80   }
81
82   private void makeLarger ()
83   {
84     Character [] newList = new Character [____________________ ];
85     for (int i = 0; i < capacity; i++)
86     {
87       _______________________________
88     }
89     list = newList;
90     capacity = ________________________________;
91
92   }
93
94   public String toString() {
95     StringBuilder s = new StringBuilder();
96     s.append("bottom [");
97     for (int i = 0; i < size-1; i++)
98       s.append(list[i]+", ");
99     s.append(list[size-1]);
100    s.append("] top");
101    return s.toString();
102  }
103}
```

Questions:

1. List all the data fields in the class **`StackOfCharacters`**.

2. Explain the difference between **`size`** and **`capacity`** of the stack.

3. Does the class have a default constructor? If so, which line is it on?

4. What are the values of ALL data fields right after a **`StackOfCharacters`** object is created (right after a call to a constructor)?

5. Implement the **`empty`** method on line 25.

6. The **`peek`** method on line 66, uses **`empty`** method in an **`if`** statement, and returns **`null`** if that method returns **`true`**. Why is this done?

7. The **`push`** method has a bug. Figure out what it is and suggest a fix.

8. Write the Javadoc comments for the **`push`** and the **`pop`** methods.

9. **`makeLarger`** should create a larger array when the capacity is reached. Complete the blanks in the code to achieve this.

10. Draw the content of the array representing a stack after the following sequence of operations:

```
StackOfCharacters s = new StackOfCharacters();
s.push('C');
s.push('S');
s.push(' ');
s.push('2');
s.pop();
s.push('1');
s.push('0');
s.peek();
s.push('1');
```