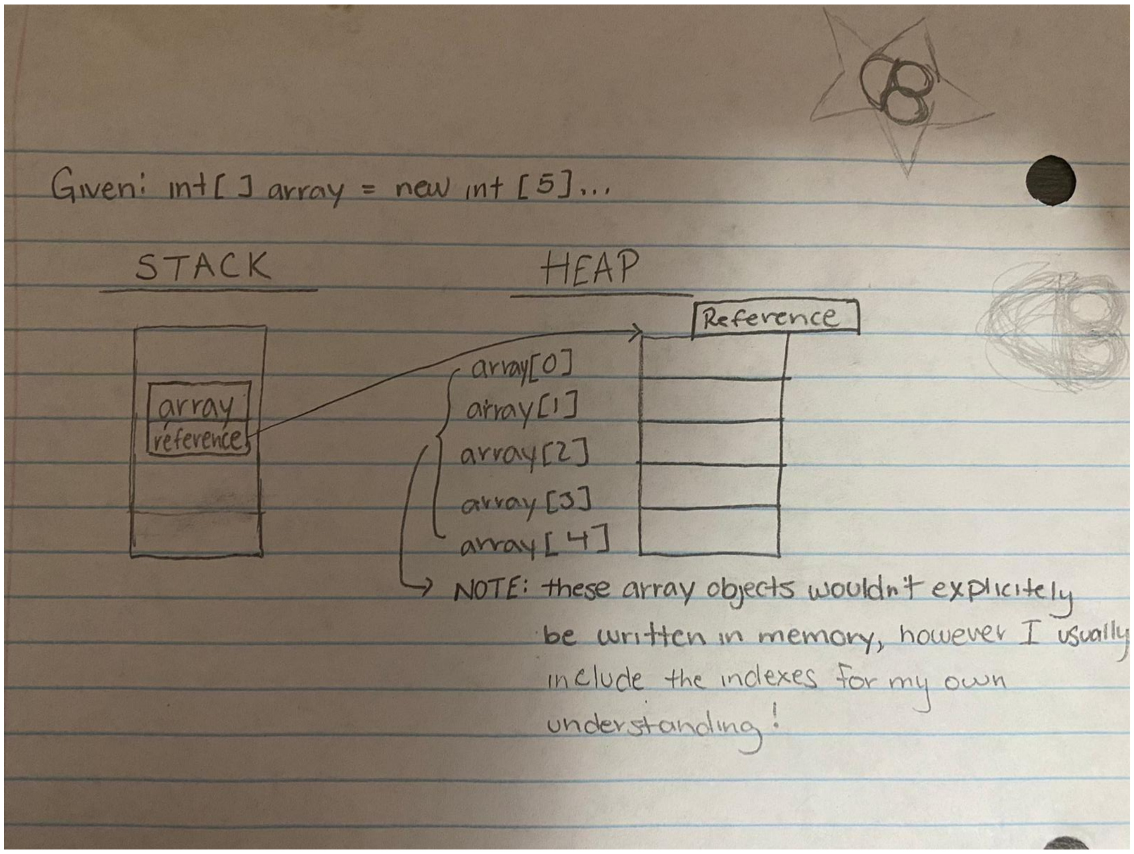
1.

(a) *Explain how an array is stored in memory. What are the advantages of this implementation? What are the disadvantages? How are 2D arrays stored in memory? It may be wise to use pictures here. If I have an array a such that: a = new double[1000][1000], explain why a[500][500] is quick to evaluate.*

An array will usually store either primitive values (int, char, Boolean, etc.) or references (a.k.a pointers) as objects. When an array is created by using "new", as demonstrated nicely by the example given within the question, a memory space is allocated in the heap space and a reference is returned as shown in the diagram below. For 2-d arrays, it follows a similar principle as it will be stored in the computer's memory one row following another. Assuming that an array has been defined by assigning "a = new double[1000][1000], determining a[500][500] would be an easy evaluation for the computer, as it doesn't have to go in order starting from the first index to search for this value. Instead, the computer will simply calculate the location of this cell given the two values of 500, making it just as easy to evaluate as let's say a[1][1], proving the importance of defining the amount of storage an array needs in the first place.


Given: int[ ] array = new int [5]...
STACK
HEAP
Reference
array reference
array[0]
array[1]
array[2]
array[3]
array[4]
NOTE: these array objects wouldn't explicitely be written in memory, however I usually include the indexes for my own understanding!

(b) *See the Java source code below. Make as many simplifications to the code as possible. For each simplification you make explain why it is a valid to do so.*

public static String f(int x, int y) {

~~if(x < 100) {~~ -> **The loop will never run, as x must be less than 100 but the loop runs when x is between 200 and 210., so the entire loop can be removed**

~~if(y < 0) {~~ -> The value of y is independent of the results in the for this statement, so this condition can be removed

~~for(int j = x; j > 200 && j < 210; j--) {~~ -> for loop should can just run such that 200 < j < 210, and simply return A after

~~if(j < 210) {~~

~~return "A";~~

~~}~~

~~}~~

~~}~~

~~}~~

~~else if(y >= 0 && x >= 100 && x < 200) {~~

~~if(x < 200) { -> line can be removed if we include a range from 100 to 200 in the else if statement above~~

~~return "B";~~

~~}~~

~~}~~ -> **Note that B will never be returned if x >=200, so this code for this statement can be removed as well. This is assumption is made because of the fact that this is an else if, which would assume that the pre-existing conditions of the initial of if statement was not met.**

else if(y >= 0 ~~|| x >= 100~~) { -> condition for variable x van be removed as it isn't being used by the for loop

for(int i = 0; i < (y + 5); i++) {

if(i == 1) {

return "C"; -> Note that if (y+5) <= 1 OR y <-3, C won't be returned. Therefore the for loop has unnecessary information which is further revised in the final code below

}

}

else {

return "D";

}

return "E";

}

<u>Final Edited Code (based on above comments):</u>

```
public static String f(int x, int y) {

   if((x < 100 || (y >= 0 && x >= 200 || y < -3)) {

      return "E";                    -> Assumption: x>=100 because this was defined in prior if statements

   else if (y >= 0)

      return "B";                    -> Assumption: x>=100 because this was defined in prior if statements

   return "C";                       -> Due to the prior condition, it is assumed that y < 0 in order for C to be returned

   }

}
```

<u>Summary of points and rationale</u> (majority of this will be repeats of the comments above, but I wanted to include this section to avoid any confusion!)

The code provided to us had a lot of room for simplification. This was often due to the repetitive nature of the conditions that were provided. I was able to determine that due to conflicting defined conditions, both the letter “A” and “D” would never be returned, so they wouldn’t need to be included in the final source code. However, the if statements that defined these particular return statements are still important, particularly because of the condition that x>=100. It can be seen that there are 3 distinct cases in which the code would return “E”, which are listed using || (or) to start the if statement. The only conditions that must be met for B to be returned are if y>=0, so the use of an else if statement ensure that this particular case is taken care of. Finally, since all factors have been considered, the only other option would be to return C, which is done as shown by the final line.

(c) *Review Stack.java. This is a standard implementation of a stack data structure. Critique the implementation. What behaviours are left unspecified? Is this particular implementation static or dynamic with regards to the stack size? Would this be easy to change? Specifically talk about the merits of encapsulation and use the Stack class as an example to demonstrate these merits. You do not need to change the code. Just discuss*

The stack.java file is a basic implementation of a stack data structure that defines functions such as push and print correctly, and quite efficiently. However, methods such as peek() and isEmpty() are missing, which would have led to increased functionality. Additionally, it is important that the stack redefines the height for both of these functions, as the value of height is either increased by 1 or decreased by 1 if a value is added or removed from the stack respectively. One thing to note however, is that the code, especially the print() method could be simplified by removing the second conditional due to the fact hat it will never be executed.

A feature that is good about this stack class is its attention to encapsulation, as it ensures that the user only has access to change variables to methods. This is done so that users are not able to modify the height, as this could cause the stack instance to collapse and become unstable. However, I believe that the user should be able to have access to what the height is without having the ability to modify the value, which could be implemented by defining a separate method for this.

One issue with the current stack file is that it was made under the assumption that the stack would only hold integers. However, the stack can not be used for storing multiple object types. Another downside to the current implementation of the stack is that it is static in regard to the stack size. If a user were to push on more than 100 items onto the stack, an out of bound error would occur. One possible fix is to resize this dynamically by dynamically changing the memory that is allocated as a user requires greater need for larger stacks. This could be done quite intuitively by defining a new method that resizes the stack once more space is needed. Specifically, a temporary array of a larger size can be created where the contents of the current stack can be transferred to this new temporary stack to dynamically increase the size.