What is system.limitexception:Apex heap size too large ?

You have probably seen this error before while doing some heavy lifting in Apex. If you are wondering how you got this error and how to remedy it, fear not, I will be outlining a number of simple changes that can help aid in fixing this error.

First of all, what does it mean when you hit this limit? It essentially means you are using too much memory. Since Salesforce is a shared tenant system, there are limits put in place so you do not use too much of any resource, memory being one of them. So to get around this error you need to be cognizant of your memory and how you are using it.

I have seen a number of people suggesting that declaring variables as transient will fix the heap size error but declaring a variable as transient is more just a way of decreasing the size of the ViewState on a VisualForce page that is using apex:forms. While it may have an effect, the effect would be insignificant and this is not normally the culprit for these types of errors.

From what I have seen, the biggest issue with heap size comes from querying an object outside of the scope of a loop. After this query the code will still iterate over that object to perform some sort of logic for variable placement, wrapper class built, etc. To get around this, put your queries in the loop. Below is an example of what not to do with the subsequent what to do. Feel free to run this code in an anonymous window to see for yourself the heap size difference between the two.

```
//Don't do this…
List<Account> accountList = [Select Id, Name FROM Account LIMIT 5000];
System.debug(LoggingLevel.ERROR, 'Heap Size: ' + Limits.getHeapSize() + '/' + Limits.getLimitHeapSize());
Map<Id, String> accountMap = new Map<Id, String>();
for(Account a : accountList){
    accountMap.put(a.Id, a.Name);
}
System.debug(LoggingLevel.ERROR, 'Heap Size: ' + Limits.getHeapSize() + '/' + Limits.getLimitHeapSize());


//This is much better
System.debug(LoggingLevel.ERROR, 'Heap Size: ' + Limits.getHeapSize() + '/' + Limits.getLimitHeapSize());
Map<Id, String> accountMap = new Map<Id, String>();
for(Account a : [Select Id, Name FROM Account LIMIT 5000]){
    accountMap.put(a.Id, a.Name);
}

System.debug(LoggingLevel.ERROR, 'Heap Size: ' + Limits.getHeapSize() + '/' + Limits.getLimitHeapSize());
```

Another pitfall comes from simply querying more fields than you need to from the object. The more fields, the more memory you will need to store those fields. So when you are building your queries, be sure to only grab what you need.

Another less common issue comes from lists and how you build them. This is especially prevalent when using recursion. For an example, let's assume that you are building a text string from a tree type structure. As you are iterating through the structure to build the string you need to call the same method from within the method. This is all well and good until you start adding more and more to the lists after you have already used them. A good way to get around this is to remove or null the specific instance of the object when you are done with it.

```
//Recursive example that removes objects when done
public String myRecursiveMethod(List<MyObject> myList){
    String myText = '';
    Integer size = myList.size();
    for(Integer i = 0; i < size; i++){
        if(myList.get(0) != null){
            myText += myRecursiveMethod(myList.get(0));
        }
        myList.remove(0);
    }
    return myText;
}


//Another example of nulling objects when done
for(MyObject mo : myObjectList){
    doStuffWithMyObject(mo);
    mo = null;
}
```