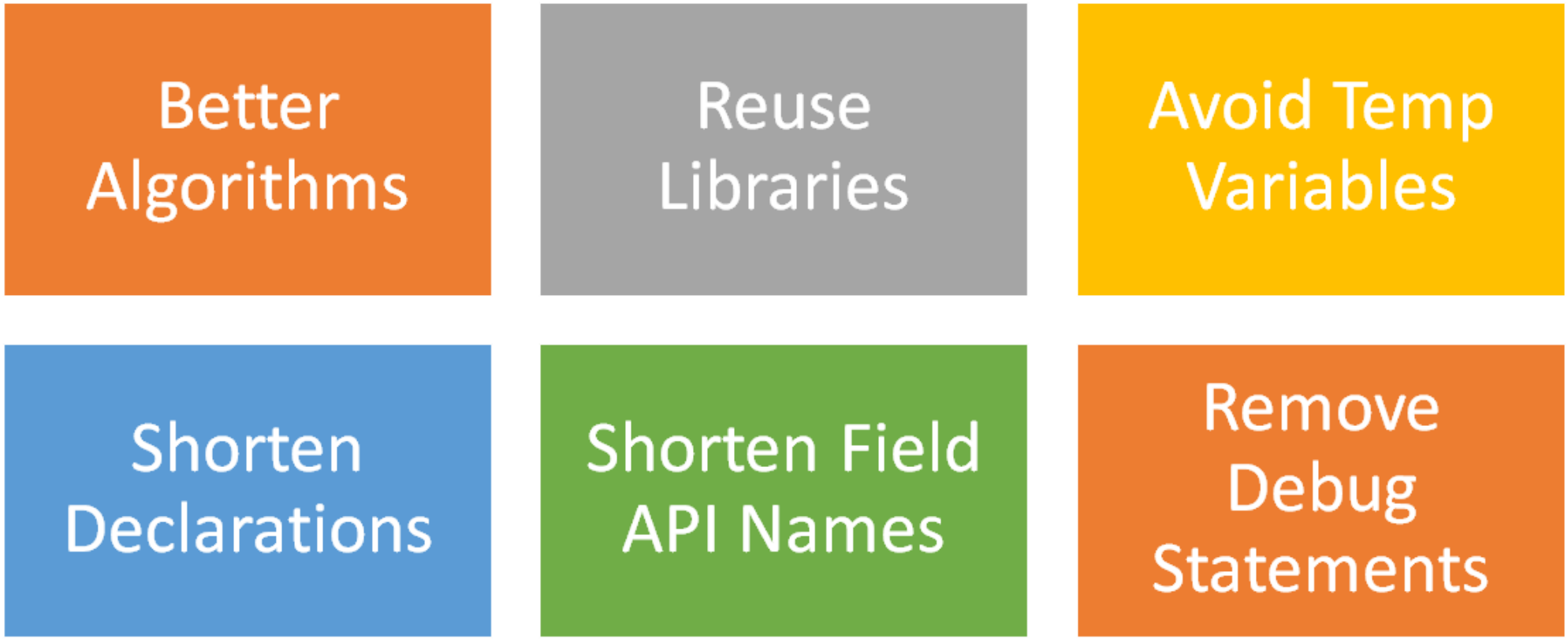
Heap Size: Elephant in the room

By Vernika Arora March 22nd, 2019

Heap size is a common, yet neglected problem whenever it comes to Salesforce Apex development. Heap size in actual is the count of memory used by the variables, object instances in an Apex class. In order to store these objects & variables, memory is allocated in Salesforce which is derived from allocated Heap. The Heap size truly depends on the type of transaction, Salesforce provides a heap size limit of 6MB for the synchronous transaction and 12 MB for the asynchronous transaction. Whenever too much data is stored during processing an error occurs prompting Apex heap size too large. You must have encountered this error at least once if you are a developer.
Here are a few tips to ensure that the heap limit does not exceed the maximum size and your code runs smoothly



Using Efficient Algorithms

Being the heart of any code, algorithms can make or break the running of your application. A good space-efficient algorithm can save a lot of your heap memory while simultaneously performing smoothly. Before developing any logic or writing any code snippet, sit, and think through the algorithms to use and choose the one that serves your purpose and makes use of the heap memory efficiently. Try not to split the parts of your algorithms into different functions; rather try to keep them in line when used only within that code.

Built-in Apex libraries

Try to leverage and make use of the inbuilt Apex libraries that Salesforce provides other than writing your custom logic for everything. For example, use the Math class for performing mathematical operations, use JSONGenerator class to create JSON structures, and many more. One line of method calling code is way better than rewriting the logic on your own. This not only saves you time but also the heap memory is managed efficiently.

Avoid using temporary variables

Creating temporary variables is a favorite practice of many developers. We unknowingly create such variables in our code without realizing that we can do without them. These variables take up unnecessary space and add up to the heap memory.

For example, instead of writing

```
List<Account> accLst= [Select id from Account Limit 10];
for (Account a: accList) {}
```

you should prefer,

```
for (Account a: [Select id from Account Limit 10]) {}
```

Shorten variable names & declarations

You can spare characters when naming temporary variables like loop counter variables as i instead of myCounter. You should keep meaningful names according to the business.

You can also combine variable declarations like:

```
Integer i = 0, j=1;
```

Instead of

```
Integer i = 0;
Integer j=1;
```

Shorten Field API Name

We all love to give meaningful names to the custom fields in Salesforce objects. Little do we know that a long API name impacts the code heap size. For example, it is counted in SOQL query size, code that references the field, and each time it is accessed.

Remove unnecessary debug statements

Since all the debug statements are counted against the code length, it is necessary to keep a cap on them in order to limit the consumed heap size of the Apex code. It is fine to use them while doing the apex development but should be removed or minimized in the final production version of the code.

Conclusion

With all these findings, you must have gained an insight into managing your heap size and its importance. Tests on various production orgs demonstrated that you could easily gain more than 10% on existing code by managing indentation (using tabs) and removing trailing white spaces on the line of code (this can be automated). By following Apex's best practices and using efficient algorithms, you can significantly minimize the heap size allocations for your Apex code.