HL/SL IA: How to get marks for 'complexity'

In 2014 the Computer Science was dramatically updated, but we are still coding Java and bits that were complex before, appear to still count as 'complex'. The rule of thumb used to be that a **HL project** had to have at **least 10** of the **following 18 aspects** to be counted as 'sufficiently complicated'. They are:

1. **Adding data to an instance of the RandomAccessFile** class by direct manipulation of the file pointer using the seek method

2. **Deleting data from an instance of the RandomAccessFile** class by direct manipulation of the file pointer using the seek method. (Data primitives or objects may be shuffled or marked as deleted by use of a flag field. Therefore files may be ordered or unordered)

3. **Searching** for specified data in a file

4. **Recursion**

5. **Merging two or more sorted data structures**

6. **Polymorphism**

7. **Inheritance**

8. **Encapsulation**

9. **Parsing a text file** or other data stream

10. **Implementing a hierarchical composite data structure**. A composite data structure in this definition is a class implementing a record style data structure. A hierarchical composite data structure is one that contains more than one element and at least one of the elements is a composite data structure. Examples are, an array or linked list of records, a record that has one field that is another record, or an array

11.Use of **additional libraries** (such as utilities and graphical libraries not included in appendix 2 Java Examination Tool Subsets)

12. **Inserting data into an ordered sequential file** without reading the entire file into RAM

13. **Deleting data from a sequential file** without reading the entire file into RAM

14. **Arrays of two or more dimensions**.

15.to 18. Up to four aspects can be awarded for the **implementation of abstract data types** (ADTs)

ADT Name	1 aspect	2 aspects	3 aspects	4 aspects
General criteria	An incomplete ADT is implemented.	An ADT is implemented with all key methods implemented.	An ADT is implemented that includes some error checking.	An ADT is implemented completely and robustly.
Lists, implemented using references (such as, a dynamically linked list).	A node style class with appropriate constructors and methods to set and get data elements.	Methods are implemented to add at / remove from the tail and the head of the list.	Proper checks are made for errors such as attempting to get an element from an empty list or inserting the same element twice.	All error conditions are checked for, and all appropriate methods are implemented. For a doubly linked list these could be: `size, isEmpty, first, last, before, after, insertHead, insertTail, insertBefore insertAfter`
Tree (simple, ordered binary tree is sufficient using arrays or dynamically linked object instances)	A class or interface with appropriate constructors and methods to set and get data elements.	Methods are implemented to add at / remove from the correct point in the tree.	Proper checks are made for errors such as attempting to get an element from an empty tree or not inserting the same element twice.	All error conditions are checked for, all appropriate methods are implemented. For a simple ordered, binary tree these could be: `size, isEmpty, root, parent leftChild, rightChild`
Stack implemented dynamically or statically.	A class or interface with appropriate constructors and methods to push and pop items.	Methods to test for full and empty stack are added.	Proper checks are made for errors such as attempting to get an element from an empty stack.	Probable methods `push, pop, top, isEmpty isFull, size`
Queue implemented dynamically or statically	A class or interface with appropriate constructors and methods to enqueue and dequeue items.	Methods to test for full and empty queue are added.	Proper checks are made for errors such as attempting to get an element from an empty queue.	Probable methods `enqueue, dequeue front, rear, isEmpty, isFull, size`
Hash table implemented in an array.	A class or interface with appropriate constructors and methods to insert and remove items.	Methods to test for full table and duplicate keys are added.	Proper checks are made for errors such as attempting to get a non-existent key, clashes are dealt with properly	Probable methods `hashFunction, insertKey removeKey, isDuplicate isEmpty, isFull, size`

The "**Non-trivial" principle** means that the programmer must demonstrate that the program benefits from the use of the aspect.
Where one aspect includes others, all are credited (always provided that the use is non-trivial, well-documented and appropriate).

Standard Level

The rules are slightly different for SL. The rule of thumb used to be that a project had to **have at least 10** of the **following 15 aspects** to be counted as 'sufficiently complicated'. They are

1. Arrays
2. User-defined objects
3. Objects as data records
4. Simple selection (if–else)
5. Complex selection (nested if, if with multiple conditions or switch)
6. Loops
7. Nested loops
8. User-defined methods
9. User-defined methods with parameters (the parameters have to be useful and used within the method body)
10. User-defined methods with appropriate return values (primitives or objects)
11. Sorting
12. Searching
13. File i/o
14. Use of additional libraries (such as utilities and graphical libraries not included in appendix 2 Java Examination Tool Subsets)
15. Use of sentinels or flags