ModuleCheck Class

As per the previous discussion, when a module has been developed by the module developer, he/she will inform the other module developers about the methods which he/she have declared publicly, and are ready to use by the other modules. The module developer needs to mention the exact names of the methods (along with case sensitivity), the functionality of those methods (what input does those methods take and what results they provide), the parameters that needs to be passed (in proper format and sequence), and format for the return type of the result.

Once a method has been declared publicly by the module developer (along with its functionality, inputs and outputs) to the other module developers as ready for use, that module's functionality can be improved anytime using overloading and/or overriding, but all attempts needs to be made to ensure that the previous methods (which have been declared ready for use) remains active along with its previous functionality. The previous declared methods should never be made obsolete as many modules may start using those methods and their code might break (stop working) if the functionality of the declared methods are reduced, altered or if those methods are made obsolete.

Thus, a method which has been declared as a standard public method and ready for use by the developer will remain active from that time onwards with at least its current functionality. The future modifications in the method should in no way hamper the way the method was previously used (that is how and what the method used to take as the inputs and what results it used to provide, and in what sequence). Only functionality can be added making the method "more functional" or "more useful", so that the method can be used in some other ways also.

But some methods, which now belong to some module, can later be moved to some other module. For example, Routing Manager contains 25 methods for example. Later, if the Routing Manager's workload increases (now let's say 23 new methods needs to be added/declared) and so, the module developer decides to separate the module in 3 parts (3 separate modules, RMmodule1, RMmodule2, and RMmodule3) for example, then those 25 old modules should be maintained, that is, all of them should be present in one of those 3 modules, or maybe they can also be moved to any other module. But once a method is declared ready for use, it should be present with some or the other module with at least its previous functionalities.

But how to figure out that which method now belongs to which module?

For that purpose we are developing this ModuleCheck class. This class will contain the information about which method belongs to which module. When a module needs some information, or needs some work to be done, it will send a query or a process to the Glue Code. That is, the module will call one of the methods from the pool of all the methods which are defined and declared by all the module developers of B4. The query/process will be send by the module to the Glue Code. The Glue Code will figure out that which method belongs to which module, and thus which query/process has to be responded by which module, with the help of the ModuleCheck class.
The module which is calling the methods does not need to have any idea as to which method belongs to which module. Glue Code will take care of it with the help of the ModuleCheck class. So, all the developers need to update all the information in the ModuleCheck class about each and every method present in their module which has been declared as public and are ready to be used by the other modules of B4.

MODULE CHECK

1. The modules will send only one query/process at a single time.
2. The query/publish have to be sent in the form of messages only, one query/process at a time.
3. There will be a separate Java class file (a separate class- ModuleCheck class file) which will be a part of the Glue Code.
4. The java class file will contain the information about which methods belongs to which module.
5. When the Glue Code receives a query/process, the Glue Code matches the query/process with the queries (methods) mentioned in the java class file and tries to find out that which query (method) belongs to which module, and then the Glue Code sends the received process message to the respective module or calls the query (API call) from the respective module.
6. The Glue Code will use the API call to get the response of the query from the module the method belongs to, and then send the response of the query to the corresponding module which initiated the query.