|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Option** | **Content** | **Note** |
| 1 |  | 1. Make sure that there shouldn't be any project warnings. |  |
| 2 |  | All unused usings need to be removed. Code cleanup for unnecessary code is always a good practice. |  |
| 3 |  | 1. 'null' check needs to be performed wherever applicable to avoid the *Null Reference Exception* at runtime. 2. – Or use nullable condition to avoid to the *Null Reference Exception* at runtime. |  |
| 4 |  | Naming conventions to be followed always. Generally for variables/parameters, follow Camel casing and for method names and class names, follow Pascal casing. |  |
| 5 |  | Code Reusability: Extract a method if the same piece of code is being used more than once or you expect it to be used in future. Make some generic methods for repetitive task and put them in a related class so that other developers start using them once you intimate them. Develop user controls for common functionality so that they can be reused across the project. |  |
| 6 |  | 1. Code Consistency: Let's say that an Int32 type is coded as int and String type is coded as string, then they should be coded in that same fashion across the application. But not like sometimes int and sometimes as Int32. |  |
| 7 |  | Disposing of Unmanaged Resources like File I/O, Network resources, etc. They have to be disposed of once their usage is completed. Use usings block for unmanaged code, if you want to automatically handle the disposing of objects once they are out of scope. |  |
| 8 |  | Proper implementation of Exception Handling (try/catch and finally blocks) and logging of exceptions. |  |
| 9 |  | 1. Making sure that methods have less number of lines of code. Not more than 30 to 40 lines. |  |
| 10 |  | Avoid nested for/foreach loops and nested if conditions as much as possible. |  |
| 11 |  | Use anonymous types if code is going to be used only once. |  |
| 12 |  | Try using LINQ queries and Lambda expressions to improve Readability. |  |
| 13 |  | Proper usage of var, object, and dynamic keywords. They have some similarities due to which most of the developers are confused or don’t know much about them and hence they use them interchangeably, which shouldn't be the case. |  |
| 14 |  | Use access specifiers (*private, public, protected, internal, protected internal*) as per the scope need of a method, a class, or a variable. Let's say if a class is meant to be used only within the assembly, then it is enough to mark the class as internal only. |  |
| 15 |  | Use interfaces wherever needed to maintain decoupling. Some design patterns came into existence due to the usage of interfaces. |  |
| 16 |  | Mark a class as sealed or static or abstract as per its usage and your need. |  |
| 17 |  | 1. Use a Stringbuilder instead of string if multiple concatenations are required, to save heap memory. |  |
| 18 |  | Check whether any unreachable code exists and modify the code if it exists. |  |
| 19 |  | Use constants and readonly wherever applicable. |  |
| 20 |  | Usage of 'out' and 'ref' keywords be avoided as recommended by Microsoft (*in the Code analysis Rules and guidelines*). These keywords are used to pass parameters by reference. Note that 'ref' parameter should be initialized in the calling method before passing to the called method but for 'out' parameter this is not mandatory. |  |

Refer:

1. <http://www.codeproject.com/Articles/593751/Code-Review-Checklist-and-Guidelines-for-Csharp-De>
2. <http://www.codeproject.com/Articles/524235/Codeplusreviewplusguidelines>