BBM384 -Software Engineering Laboratory

Sports Center Membership System Coding Standarts

Group 4:
Aycan ÖZMEN – 21427248
Muazzez Şule KARAŞLAR-21427066
İlayda KAYA- 21580836
Mustafa DANYILDIZ – 21426845
Uğurcan ÇİFTÇİ -21526863

1. Introduction

Anybody can write code. With a few months of programming experience, you can write 'working applications'. Making it work is easy, but doing it the right way requires more work, than just making it work.

Majority of the programmers write 'working code', but not ‘good code'. Writing 'good code' is an art and you must learn and practice it.

Everyone may have different definitions of the term ‘good code’. In our definition, the following are the characteristics of good code.

• Reliable

• Maintainable

• Efficient

If your code is not reliable and maintainable, you(and your team) will be spending a lot of time to identify issues, trying to understand code etc throughout the life of your application.

This document describes rules and recommendations for developing applications and class libraries using the C# Language. The goal is to define guidelines to enforce consistent style and formatting and help developers avoid common pitfalls and mistakes.

1.1. Flags

The following flags are used to help clarify or categorize certain statements:

[C#v2+]-> A flag to identify rules and statements that apply only to C# Language Specification v2.0 or greater.

2.Naming Conventions and Standards

Camel Case

A word with the first letter lowercase, and the first letter of each subsequent word-part capitalized.

Example: courseName

Pascal Case

A word with the first letter capitalized, and the first letter of each subsequent word-part capitalized.

Example: CourseName

-Use **PascalCasing** for **class names and method names**.

```
public class CourseController
{
        public static bool InsertCourse(Course course)
        {

              //…

        }

}
```

-Use **camelCasing** for **method arguments and local variables.**

```
public partial class Login : System.Web.UI.Page
{
        protected void LoginBtnClick(object sender, EventArgs e)
        {
            string username = userName.Value;
            string password = passwordInput.Value;
        }
}
```

- Do not use **Hungarian** notation or any other type identification in identifiers

```
// Correct
int counter;
string name;

// Avoid
int iCounter;
string strName;
```

-Do not use **Underscores** in identifiers. Exception: you can **prefix private static variables with an underscore.**

```
// Correct

public DateTime birthDate;


// Avoid


public DateTime customer_Appointment;


// Exception


private DateTime _birthDate;
```

-Use **prefix interfaces** with the letter **I**. Interface names are noun (phrases) or adjectives.

```
public class syscourseHandler : IHttpHandler
 {

 }
public interface Ishape
{
}
```

-Always use **singular noun** to define **enum**.

```
public  enum MailType
{
    Html,
    PlainText,
    Attachment
}
```

-Avoid using **Abbreviations**. Exceptions: abbreviations commonly used as names, such as **Id, Xml, Ftp, Uri**

```
// Correct

UserGroup userGroup;

Assignment employeeAssignment;

// Avoid

UserGroup usrGrp;

Assignment empAssignment;

// Exceptions

CustomerId customerId;

XmlDocument xmlDocument;

FtpHelper ftpHelper;

UriPart uriPart;
```

-Do **declare all member variables at the top of a class**, with static variables at the very top.

```
public class Course
{
      int _id;
      string _name;
      int _maxQuota;
      int _periodWeek;
      int _durationMinute;
      string _level;
      string _information;
      string _equipment;
      string _price;
}
```

-Do not use names that **begin with a numeric character.**

-Do not use **single character variable names** like i, n, s etc. Use names like index, temp.

-Use Pascal Case **for file names**.

-**File name** should match with class name.

For example, for the class **HelloWorld**, the file name should be **helloworld.cs**

3. Indentation and Spacing

-Use TAB for indentation. Do not use SPACES. Define the Tab size as 4.Spaces improve readability by decreasing code density.

-Use one blank line to separate logical groups of code.

-Do not use a space after the parenthesis and function arguments

-Do not use spaces between a function name and parenthesis.

-Use a single space after a comma between **function arguments**.

Right: public User(int id, string name, string surname, string username, string password)

Wrong: public User (int id,string name,string surname,string username,string password)

-Use a single space before and after **comparison operators, flow control statements.**

Right: while (x == y) if (x == y)

Wrong: while(x==y) if (x==y)

-Do vertically align curly brackets.

```
public partial class Login : System.Web.UI.Page
{
        protected void LoginBtnClick(object sender, EventArgs e)
      {
            string username = userName.Value;
            string password = passwordInput.Value;
       }
}
```

4. Comments

Do not write comments for every line of code and every variable declared. Write comments wherever required. Good, readable code will require very few comments. If all variables and method names are meaningful, that will make the code very readable and it will not need much commenting. Fewer lines of comments will make the code more elegant. However, if the code is not clean/readable and there are fewer comments, that is worse.

Write clean, readable code in such a way that it doesn't need any comments to understand it. Do a spell check on comments and also make sure that proper grammar and punctuation are used.

- Use // or /// for comments. Avoid using /* … */
- Begin comment text with an uppercase letter.
- End comment text with a period.
- Place the comment on a seperate line, not at the end of a line od code.

```
/// <summary>
/// Summary description for sysbranchHandler.
/// </summary>
```

5. Architecture

- Always **use multi layer** (N-Tier) architecture.
- Never access database from the UI pages. Always have a data layer class which performs all the database related tasks. This will help you support or migrate to another database back
  end easily.
- **Use try-catch** in your data layer to catch all database exceptions. This exception handler
  should record all exceptions from the database.
- **Separate your application** into multiple assemblies (MVC). Group all independent utility classes into a separate class library. All your database related files can be in another class library.

6.ASP.NET

1. Do not use session variables throughout the code. Use session variables only within the classes and expose methods to access the value stored in the session variables. A class can access the session using **System.Web.HttpContext.Current.Session**
2. **Do not store large objects in session**. Storing large objects in session may consume lot of server memory depending on the number of users.For exaple: store user id.

7.Exception Handling

Never do a "catch exception and do nothing." If you hide an exception, you will never know if the exception happened or not. In the case of exceptions, give a friendly message to the user, but log the actual error with all possible details about the error, including the time it occurred, the method and class name, etc. Always catch only the specific exception, not generic exceptions. Do not write very large try-catch blocks.Do not write try-catch in all your methods.

```
public class UserController
{
    public static User GetAllUser(string username, string password)
    {
        User user = null;
        using (var connection = Database.GetConnection())
        {
            try
            {
                var command = new MySqlCommand("SELECT * FROM users WHERE username=@username and password=@password", connection);

                command.Parameters.Add(new MySqlParameter("username", username));
                command.Parameters.Add(new MySqlParameter("password", password));
                connection.Open();

                using (MySqlDataReader reader = command.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        user = new User(reader.GetInt32(0), reader.GetString(1), reader.GetString(2), reader.GetString(3),
                                        reader.GetString(4), reader.GetString(5), reader.GetString(6),
                                        reader.GetString(7), reader.GetString(8)
                                        );
                    }
                }
            }
            catch(MySqlException e)
            {
                throw;
            }
            finally
            {
                connection.Close();
            }

        }
        return user;
    }
```