Практическая работа: Оценка соответствия стандартам кодирования

1. Обзор стандартов

Для оценки используется **Microsoft C# Coding Conventions** и **.NET Design Guidelines**.

2. Оценка кода по модулям

Модуль 1: App.xaml.cs

csharp

```
namespace CompanyPayrollApp
{
    public partial class App : Application
    {
        // ✅ Правильно: статическое поле с PascalCase
        public static string ConnectionString =
            @"Data Source=.\SQLEXPRESS;Initial Catalog=CompanyPayrollDB;Integrated Security=True;";

        // ✅ Правильно: приватное поле с camelCase
        public bool LoginSuccessful { get; set; }

        // ✅ Правильно: метод с PascalCase, статический
        public static string GetEmployeesViewQuery()
        {
            return @"SELECT e.id as EmployeeID, ...";
        }

        // ✅ Правильно: обработчик события с правильным именем
        private void Application_Startup(object sender, StartupEventArgs e)
        {
            // ... код
```

```
        }
    }
}
```

Оценка: ✅ Соответствует стандартам

Модуль 2: MainWindow.xaml.cs

```
csharp
namespace CompanyPayrollApp
{
    public partial class MainWindow : Window
    {
        // ✅ Правильно: обработчики событий с _Click суффиксом
        private void Employees_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                var page = new EmployeesPage();
                MainFrame.Navigate(page);
                StatusText.Text = "Модуль: Управление сотрудниками";
            }
            catch (System.Exception ex)  // ⚠ Проблема: System.Exception вместо просто Exception
            {
                MessageBox.Show($"Ошибка загрузки модуля сотрудников: {ex.Message}",
                    "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }

        // ✅ Правильно: приватный метод с PascalCase
```

```
        private void InitializeStatusBar()
        {
            StatusText.Text = "Готово";
        }
    }
}
```

Проблемы:

1. ⚠️ Использование System.Exception вместо Exception (лишнее пространство имен)
2. ✅ Обработка исключений с информативными сообщениями
3. ✅ Правильные имена методов обработчиков событий

Модуль 3: EmployeesPage.xaml.cs

```
csharp
namespace CompanyPayrollApp.Pages
{
    public partial class EmployeesPage : Page
    {
        // ✅ Правильно: приватное поле с camelCase
        private SqlConnection connection;

        // ✅ Правильно: конструктор с тем же именем, что и класс
        public EmployeesPage()
        {
            InitializeComponent();
            connection = new SqlConnection(App.ConnectionString);
            LoadEmployees();  // ✅ Вызов метода после инициализации
        }
```

```
// ✅ Правильно: приватный метод загрузки данных
private void LoadEmployees(string search = "")  // ✅ Необязательный параметр
{
    try
    {
        // ✅ Проверка состояния соединения
        if (connection.State != ConnectionState.Open)
            connection.Open();

        // ✅ Параметризованный запрос для защиты от SQL-инъекций
        string query = App.GetEmployeesViewQuery() +
            " WHERE (@search = '' OR e.last_name LIKE '%' + @search + '%')";

        SqlCommand cmd = new SqlCommand(query, connection);
        cmd.Parameters.AddWithValue("@search", search);  // ✅ Параметры

        // ... код
    }
    catch (Exception ex)
    {
        // ✅ Логирование ошибок
        MessageBox.Show($"Ошибка загрузки сотрудников: {ex.Message}", "Ошибка",
            MessageBoxButton.OK, MessageBoxImage.Error);
    }
    finally
    {
```

```
            // ✅ Гарантированное закрытие соединения
            if (connection.State == ConnectionState.Open)
                connection.Close();
        }
    }

    // ✅ Правильно: обработчики событий
    private void AddButton_Click(object sender, RoutedEventArgs e)
    {
        var window = new Windows.EmployeeEditWindow();
        // ✅ Использование лямбда-выражений
        window.Closed += (s, args) => LoadEmployees();
        window.ShowDialog();
    }
  }
}
```

Оценка: ✅ Хорошее соответствие стандартам

Модуль 4: CalculatePayrollWindow.xaml.cs

```
csharp
namespace CompanyPayrollApp.Windows
{
  public partial class CalculatePayrollWindow : Window
  {
    // ⚠️ Проблема: использование dynamic вместо конкретного типа
    private dynamic calculationResult;

    // ✅ Правильно: внутренний класс для результатов
    public class CalculationResult
```

```
    {
        // ✅ Свойства с PascalCase
        public string Item { get; set; }
        public decimal Amount { get; set; }
        public string Note { get; set; }
    }

    private void Calculate_Click(object sender, RoutedEventArgs e)
    {
        // ⚠️ Проблема: использование dynamic
        dynamic selectedEmployee = EmployeeCombo.SelectedItem;
        decimal positionSalary = selectedEmployee.position_salary;

        // ✅ Правильно: расчет с явным указанием типа decimal
        decimal baseSalary = (positionSalary / 22) * workDays;
        decimal totalGross = baseSalary + bonus;
        decimal tax = totalGross * 0.13m;  // ✅ Суффикс 'm' для decimal
        decimal netSalary = totalGross - tax;

        // ✅ Правильно: использование generic коллекции
        var results = new List<CalculationResult>
        {
            new CalculationResult { Item = "Оклад по должности", Amount = positionSalary },
            // ... другие элементы
        };

        CalculationGrid.ItemsSource = results;
```

```
    }

    private void Save_Click(object sender, RoutedEventArgs e)
    {
        if (calculationResult == null)
        {
            MessageBox.Show("Сначала выполните расчет", "Ошибка");
            return;
        }

        try
        {
            connection.Open();

            // ✅ Правильно: использование параметризованных запросов
            string accrualQuery = @"
                INSERT INTO Accruals
                (employee_id, month, year, work_days, base_salary, bonus, total)
                VALUES (@employeeId, @month, @year, @workDays, @baseSalary,
@bonus, @total);
                SELECT SCOPE_IDENTITY();";

            SqlCommand accrualCmd = new SqlCommand(accrualQuery, connection);
            accrualCmd.Parameters.AddWithValue("@employeeId",
calculationResult.EmployeeId);
            // ... другие параметры

            // ✅ Правильно: использование транзакционной логики
            int accrualId = Convert.ToInt32(accrualCmd.ExecuteScalar());
```

```
            // ... сохранение Deductions
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Ошибка сохранения: {ex.Message}", "Ошибка");
        }
        finally
        {
            connection.Close();  // ✅ Гарантированное закрытие
        }
    }
  }
}
```

Проблемы:

1. ⚠️ Использование dynamic вместо сильной типизации
2. ✅ Хорошая структура классов и методов
3. ✅ Корректная обработка исключений

Модуль 5: ReportDepartmentsPage.xaml.cs

```
csharp
namespace CompanyPayrollApp.Pages
{
  public partial class ReportDepartmentsPage : Page
  {
    // ✅ Правильно: использование using для работы с файлами
    private void ExportButton_Click(object sender, RoutedEventArgs e)
    {
        SaveFileDialog saveDialog = new SaveFileDialog
```

```
        {
            Filter = "CSV файлы (*.csv)|*.csv|Все файлы (*.*)|*.*",
            FileName = $"Отчет_отделы_{DateTime.Now:yyyyMMdd}.csv"
        };

        if (saveDialog.ShowDialog() == true)
        {
            try
            {
                // ✅ Правильно: using для автоматического закрытия ресурсов
                using (StreamWriter writer = new StreamWriter(saveDialog.FileName,
                    false, System.Text.Encoding.UTF8))
                {
                    writer.WriteLine("ID;Отдел;Кол-во сотрудников;Средний оклад;Общий оклад");

                    connection.Open();
                    string query = "SELECT * FROM vDepartmentReport ORDER BY department_name";
                    SqlCommand cmd = new SqlCommand(query, connection);

                    // ✅ Правильно: using для DataReader
                    using (SqlDataReader reader = cmd.ExecuteReader())
                    {
                        while (reader.Read())
                        {
                            writer.WriteLine(
                                $"{reader["id"]};" +
                                $"{reader["department_name"]};" +
```

```
                        $"{reader["employee_count"]};" +
                        $"{reader["avg_salary"]:N2};"  +    // ✅ Форматирование чисел
                        $"{reader["total_salary"]:N2}");
                    }
                }
            }

            MessageBox.Show($"Отчет сохранен: {saveDialog.FileName}", "Успешно",
                MessageBoxButton.OK, MessageBoxImage.Information);
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Ошибка экспорта: {ex.Message}", "Ошибка",
                MessageBoxButton.OK, MessageBoxImage.Error);
        }
        finally
        {
            connection.Close();
        }
    }
  }
}
```

Оценка: ✅ Отличное соответствие стандартам

3. Анализ соответствия стандартам

✅ Соответствующие практики:

1. Соглашения об именовании:

- **Классы:** PascalCase (EmployeesPage, CalculatePayrollWindow) ✅
- **Методы:** PascalCase (LoadEmployees, Calculate_Click) ✅
- **Параметры методов:** camelCase (sender, e) ✅
- **Локальные переменные:** camelCase (connection, saveDialog) ✅
- **Пространства имен:** CompanyPayrollApp.Pages, CompanyPayrollApp.Windows ✅

2. Структура кода:

- **Порядок членов класса:** поля → конструктор → методы ✅
- **Использование регионов:** не используется (хорошо, т.к. устарело) ✅
- **Длина строк:** в основном до 80-120 символов ✅

3. Обработка исключений:

- **Try-catch-finally:** везде используется ✅
- **Информативные сообщения:** есть во всех catch-блоках ✅
- **Гарантированное освобождение ресурсов:** using и finally ✅

4. Безопасность:

- **Параметризованные SQL-запросы:** везде используются ✅
- **Нет конкатенации строк в SQL:** проверено ✅
- **Проверка входных данных:** есть в CalculatePayrollWindow ✅

⚠️ **Проблемные места:**

1. Использование dynamic (CalculatePayrollWindow):

csharp

```
// ❌ Проблема
private dynamic calculationResult;


// ✅ Решение: создать класс
public class CalculationData
{
```

```
    public int EmployeeId { get; set; }
    public string EmployeeName { get; set; }
    public int Month { get; set; }
    // ... другие свойства
}
private CalculationData calculationResult;
```

2. Несогласованные имена переменных:

```
csharp
// В разных модулях по-разному:
private SqlConnection connection;  // ✅
private SqlConnection conn;        // ⚠️ в некоторых местах
```

3. Отсутствие комментариев XML Documentation:

```
csharp
// ❌ Отсутствует
private void LoadEmployees()

// ✅ Должно быть
/// <summary>
/// Загружает список сотрудников из базы данных
/// </summary>
/// <param name="search">Строка для поиска по фамилии</param>
private void LoadEmployees(string search = "")
```

4. Магические числа:

```
csharp
// ❌ Магическое число
decimal tax = totalGross * 0.13m;

// ✅ Константа
```

private const decimal TaxRate = 0.13m;

decimal tax = totalGross * TaxRate;

4. Оценка по критериям

Критерий	Оценка	Комментарий
Именование	8/10	В основном соответствует, есть мелкие несоответствия
Структура	9/10	Хорошая организация классов и методов
Безопасность	10/10	Отличная защита от SQL-инъекций
Обработка ошибок	9/10	Все исключения обрабатываются, есть finally
Читаемость	8/10	Хорошая, но можно добавить комментарии
Производительность	9/10	Использование using, правильное управление соединениями
Сопровождаемость	7/10	Динамические типы усложняют поддержку

5. Итоговая оценка

Общий балл: 8.5/10

Сильные стороны:

1. ✅ Отличная безопасность (параметризованные запросы)
2. ✅ Хорошая структура классов и методов
3. ✅ Правильное именование в большинстве случаев
4. ✅ Корректная обработка исключений и ресурсов

Слабые стороны:

1. ⚠️ Использование dynamic вместо строгой типизации
2. ⚠️ Недостаток XML-документации
3. ⚠️ Магические числа в коде
4. ⚠️ Несогласованность в именовании некоторых переменных