Министерство образования Республики Беларусь

Учреждение образования  
 БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
 ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Рефакторинг и оптимизация программного кода

Отчет

по практической работе №1

на тему:

**ОСНОВНЫЕ ПРОБЛЕМЫ И ОШИБКИ ПРИ НАПИСАНИИ ПРОГРАММНОГО КОДА**

|  |  |  |  |
| --- | --- | --- | --- |
| Проверил | |  | А.В. Шелест |
|  | (подпись) | |  |
| зачтено |  | |  |
|  | (дата защиты) | |  |
|  |  | |  |
| Выполнил |  | | Н.В. Верховодко  гр. 114302 |
|  | (подпись) | |  |

Минск, 2024

**Содержание**

[1 Основные проблемы и ошибки при написании программного кода 3](#_Toc177988985)

2 Список использованных источников……..………..…………………………12

# **1 ОСНОВНЫЕ ПРОБЛЕМЫ И ОШИБКИ ПРИ НАПИСАНИИ ПРОГРАММНОГО КОДА**

Программный код программы – это текст, выполненный на особом языке, понятном машине. Он может выполняться непосредственно по тексту с помощью интерпретатора или транслироваться в особый вид с помощью компилятора.

Сложные программные комплексы при сборке могут потребовать большого количества файлов, которое может исчисляться целыми сотнями. Компьютер не способен понять, как написан код для него, плохо или хорошо. Если он будет работоспособен и не содержит ошибок, то машина запустит его в любом случае. Плохой код может усложнить задачи сопровождения программного обеспечения. Особенно актуально это для больших проектов [1].

Рассмотрим основные виды ошибок:

1 Логическая ошибка. Наиболее серьезная из всех ошибок, когда написанная программа компилирует и работает правильно, но выдает неправильный вывод.

2 Синтаксическая ошибка. Каждый компьютерный язык имеет специфический синтаксис, в котором будет написан код. Когда программист не придерживаться «грамматики» спецификациями компьютерного языка, возникнет ошибка синтаксиса. Они легко устраняются на этапе компиляции.

3 Ошибка компиляции. Многие виды ошибок могут происходить на этом этапе, в том числе и синтаксические ошибки. Иногда, синтаксис исходного кода может быть безупречным, но ошибка компиляции все же может произойти.

4 Ошибки среды выполнения. Программный код успешно скомпилирован, и исполняемый файл был создан. Ошибки при выполнении программы могут возникнуть в результате аварии или нехватки ресурсов носителя.

5 Арифметическая ошибка. Арифметические ошибки возникают, когда компьютер не может справиться с проблемами, такими как «Деление на ноль», или ведущие к бесконечному результату.

6 Ошибки ресурса. Ошибка ресурса возникает, когда значение переменной переполняет максимально допустимое значение.

7 Ошибка взаимодействия. Они могут возникнуть в связи с несоответствием программного обеспечения с аппаратным интерфейсом или интерфейсом прикладного программирования.

Также, наряду с ошибками возникают и другие проблемы процесса программирования, такие, как:

– многопоточность;

– замыкания;

– использование больших данных;

– проблемы *NP*-полной задачи (полиномиальная для недетерминированной машины Тьюринга задача поиска и принятия решения);

– безопасность;

– управление идентификацией;

– шифрование;

– надежность измерения.

Теперь приведем примеры некачественного программного кода, характерные для *C#*, характерные ошибки и методы их устранения.

1 Длинные методы. Длинные методы затрудняют понимание и поддержку кода. Они часто выполняют несколько задач одновременно, что нарушает принцип единственной ответственности (*Single Responsibility Principle*).

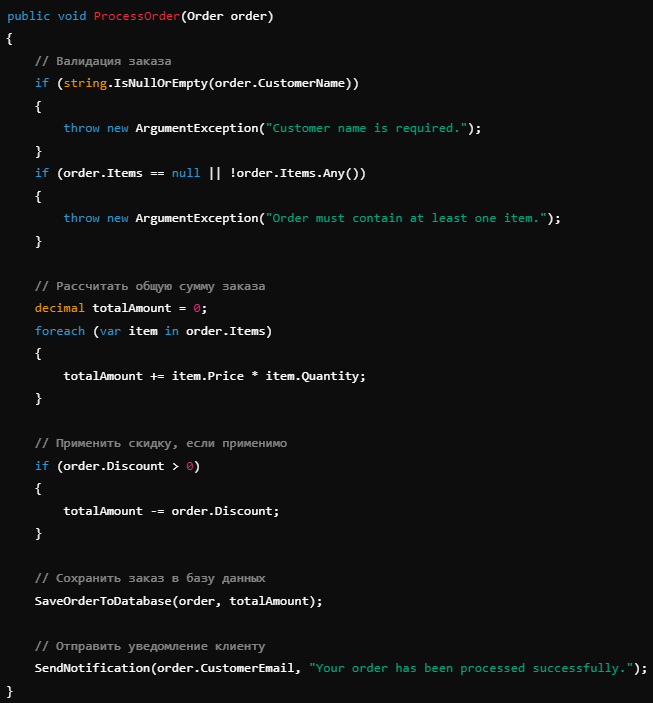


Рисунок 1.1 – Пример длинного метода

Решением для такой проблемы может стать разделение метода на более мелкие, отвечающие за конкретные задачи, представленное на рисунке 1.2.



Рисунок 1.2 – Решение длинного метода

2 Дублирование кода. Повторяющиеся участки кода усложняют его поддержку и увеличивают вероятность ошибок при внесении изменений.

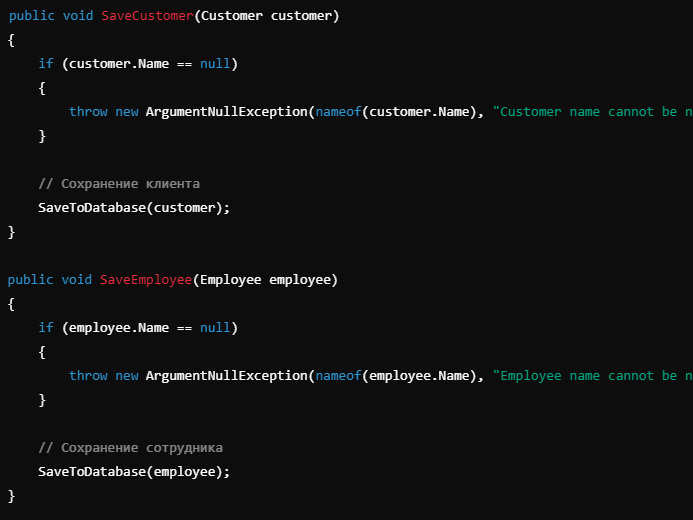


Рисунок 1.3 – Пример дублирования кода

Вынесение общего кода в отдельный метод или использование наследования может стать решением для такой ошибки.

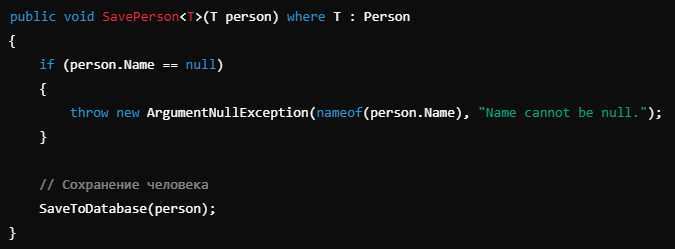


Рисунок 1.4 – Решение для дублирования кода

3 Плохие имена функций и переменных (*Poor Naming*). Неочевидные или непонятные имена функций и переменных усложняют понимание кода.

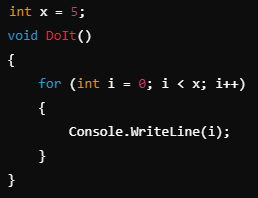


Рисунок 1.5 – Пример плохой имени функции

Использование осмысленных и описательных имен.

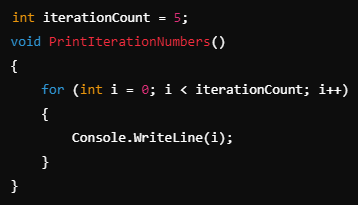


Рисунок 1.6 – Решение для плохого имени функции

4 Избыточные временные переменные (*Excessive Temporary Variables*). Лишние переменные делают код более громоздким и усложняют его чтение. Главное не переборщить и не потерять в читаемости кода.

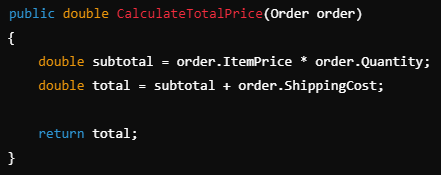


Рисунок 1.7 – Проблема избыточных временных переменных

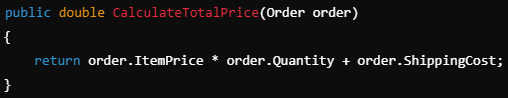


Рисунок 1.8 – Решение, удалением ненужных переменных

5 Утечка памяти (*Memory Leaks*). Неправильное управление ресурсами, такими как объекты или потоки, может привести к утечкам памяти.

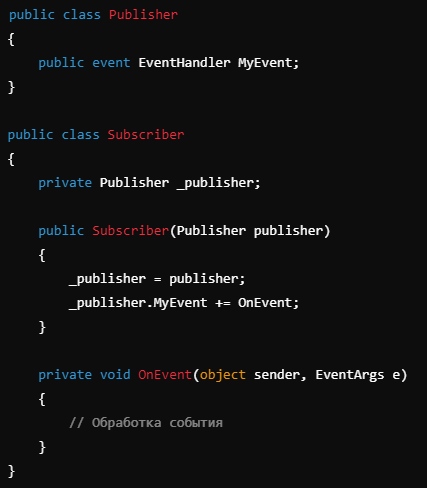


Рисунок 1.9 – Пример утечки памяти

Использование конструкции *try-with-resources* для автоматического закрытия ресурсов.

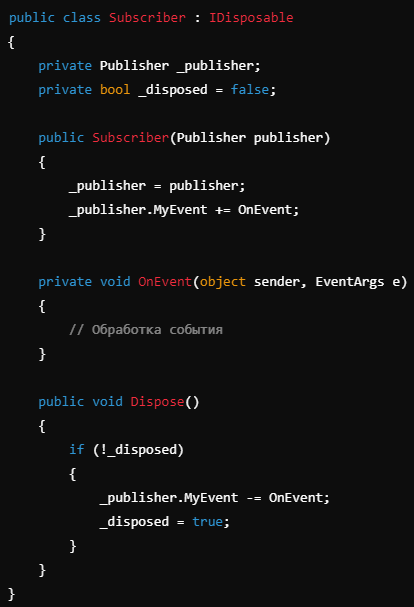


Рисунок 1.10 – Решение утечки памяти

6 Длительное выполнение операций (*Long-Running Operations*). Использование неоптимальных алгоритмов или неправильная реализация может привести к медленной работе приложения.

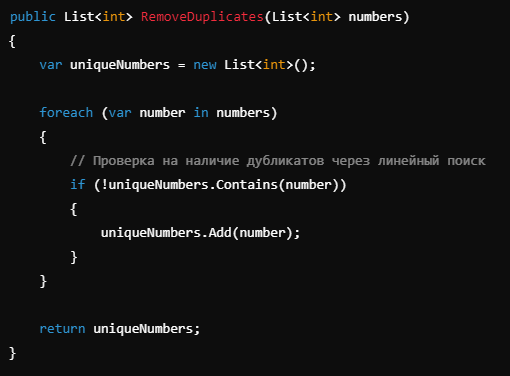


Рисунок 1.11 – Пример длительного выполнения операций

Использование более эффективных структур данных, таких как *HashSet*, для улучшения производительности.

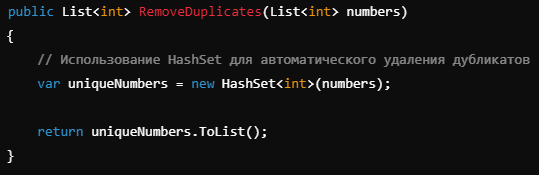


Рисунок 1.12 – Решение длительного выполнения операций

Кроме конкретных методов, для решения ошибок, рассмотрим еще общие методы, которые часто используют на практике:

1 Рефакторинг (*Refactoring*). Разбиение длинных методов на более мелкие и специализированные функции. Удаление дублирующегося кода путем вынесения общих частей в отдельные методы или классы. Переименование функций и переменных для улучшения читаемости.

2 Использование паттернов проектирования (*Design Patterns*). Применение паттернов, таких как *Singleton, Factory, Observer*, для решения общих проблем и улучшения структуры кода.

3 Автоматическое тестирование (*Automated Testing*). Модульные тесты (*Unit Tests)* для проверки отдельных компонентов. Интеграционные тесты (*Integration Tests*) для проверки взаимодействия между компонентами. Использование *CI/CD* для автоматизации процесса тестирования и деплоя.

4 Инструменты анализа кода (*Code Analysis Tools*). Линтеры (например, *Checkstyle, PMD*) для автоматической проверки стиля и обнаружения потенциальных ошибок. Статический анализ кода с помощью инструментов, таких как *SonarQube*, для выявления уязвимостей и проблем с производительностью. Профилировщики (например, *VisualVM, JProfiler*) для выявления узких мест в производительности и утечек памяти.

5 Оптимизация алгоритмов и структур данных. Выбор эффективных алгоритмов и структур данных для решения конкретных задач. Анализ сравнительной сложности алгоритмов и их влияние на производительность приложения.

6 Код-ревью (*Code Review*). Регулярные ревью кода с участием нескольких разработчиков для выявления и исправления ошибок на ранних стадиях. Использование инструментов для совместной работы (например, *GitHub Pull Requests*) для упрощения процесса ревью.

7 Применение предложений *IDE*. Для написания приложений на *C*# можно использовать *Rider* от *JetBrains*, он может автоматически рефакторить код.

Вывод:

Таким образом, понимание и умение выявлять некачественные участки кода – важный навык для любого разработчика. Регулярный рефакторинг, использование инструментов анализа кода, написание тестов и следование принципам чистого кода значительно улучшают качество программного обеспечения, облегчают его поддержку и развитие. Изучение предложенных примеров и методов устранения ошибок поможет повысить компетентность в разработке на языке *C*# и использовании соответствующих фреймворков.

Ссылка на репозиторий *GitHub*: https://github.com/NVERKHOVODKO/crypto-wallet-up

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Refactoring [Электронный ресурс]. – Режим доступа: https://refactoring.guru/refactoring. – Дата доступа: 18.10.2024.

[2] refactoring developer's guide [Электронный ресурс]. – Режим доступа: https://www.sonarsource.com/learn/refactoring/. – Дата доступа: 18.10.2024.