Хорошие практики кодирования

Юрий Литвинов
yurii.litvinov@gmail.com

13.07.2017г

Абстрактные типы данных

- currentFont.size = 16 — плохо
- currentFont.size = PointsToPixels(12) — чуть лучше
- currentFont.sizeInPixels = PointsToPixels(12) — ещё чуть лучше
- currentFont.SetSizeInPoints(sizeInPoints)
  currentFont.SetSizeInPixels(sizeInPixels) — совсем хорошо

Пример плохой абстракции

```
public class Program {
  public void InitializeCommandStack() { ... }
  public void PushCommand(Command command) { ... }
  public Command PopCommand() { ... }
  public void ShutdownCommandStack() { ... }
  public void InitializeReportFormatting() { ... }
  public void FormatReport(Report report) { ... }
  public void PrintReport(Report report) { ... }
  public void InitializeGlobalData() { ... }
  public void ShutdownGlobalData() { ... }
}
```

Пример хорошей абстракции

```
public class Employee {
  public Employee(
        FullName name,
        string address,
        string workPhone,
        string homePhone,
        TaxId taxIdNumber,
        JobClassification jobClass
  ) { ... }

  public FullName Name { get {...}; set {...} }
  public string Address { get {...}; set {...} }
  public string WorkPhone { get {...}; set {...} }
  public string HomePhone { get {...}; set {...} }
  public TaxId TaxIdNumber { get {...}; set {...} }
  public JobClassification JobClassification { get {...}; set {...} }
}
```

Уровень абстракции (плохо)

```
public class EmployeeRoster : MyList<Employee> {
  public void AddEmployee(Employee employee) { ... }
  public void RemoveEmployee(Employee employee) { ... }
  public Employee NextItemInList() { ... }
  public Employee FirstItem() { ... }
  public Employee LastItem() { ... }
}
```

Уровень абстракции (хорошо)

```
public class EmployeeRoster {
  public void AddEmployee(Employee employee) { ... }
  public void RemoveEmployee(Employee employee) { ... }
  public Employee NextEmployee() { ... }
  public Employee FirstEmployee() { ... }
  public Employee LastEmployee() { ... }
}
```

Общие рекомендации

- Про каждый класс знайте, реализацией какой абстракции он является
- Учитывайте противоположные методы (Add/Remove, On/Off, ...)
- Соблюдайте принцип единственности ответственности
  - Может потребоваться разделить класс на несколько разных классов просто потому, что методы по смыслу слабо связаны
- По возможности делайте некорректные состояния невыразимыми в системе типов
  - Комментарии в духе “не пользуйтесь объектом, не вызвав init()” можно заменить конструктором
- При рефакторинге надо следить, чтобы интерфейсы не деградировали

Ещё рекомендации

- Класс не должен ничего знать о своих клиентах
- Лёгкость чтения кода важнее, чем удобство его написания
- Опасайтесь семантических нарушений инкапсуляции
  - “Не будем вызывать ConnectToDB(), потому что GetRow() сам его вызовет, если соединение не установлено” — это программирование *сквозь* интерфейс
- Protected- и internal- полей тоже не бывает
  - На самом деле, у класса два интерфейса — для внешних объектов и для потомков (может быть отдельно третий, для классов внутри сборки, но это может быть плохо)

Наследование

- Включение лучше
  - Переконфигурируемо во время выполнения
  - Более гибко
  - Иногда более естественно
- Наследование — отношение “является”, закрытого наследования не бывает
  - Наследование — это наследование интерфейса (полиморфизм подтипов, subtyping)
- Хороший тон — явно запрещать наследование (sealed-классы)
- Не вводите новых методов с такими же именами, как у родителя
- Code smells:
  - Базовый класс, у которого только один потомок
  - Пустые переопределения
  - Очень много уровней в иерархии наследования

Пример

```
class Operation {
  private char sign = '+';
  private int left;
  private int right;
  public int Eval()
  {
     switch (sign) {
        case '+': return left + right;
     }
     throw new Exception();
  }
}
```

vs

```
abstract class Operation {
  private int left;
  private int right;
  protected int Left { get { return left; } }
  protected int Right { get { return right; } }
  abstract public int Eval();
}

class Plus : Operation {
  override public int Eval() {
     return this.Left + this.Right;
  }
}
```

Конструкторы

- Инициализируйте все поля, которые надо инициализировать
  - После конструктора должны выполняться все инварианты
- НЕ вызывайте виртуальные методы из конструктора
- private-конструкторы для объектов, которые не должны быть созданы (или одиночек)
- Deep copy предпочтительнее Shallow copy
  - Хотя второе может быть эффективнее
- Не создавайте ненужных объектов, не храните ссылки, если они не нужны

О дизайне классов и интерфейсов

- Минимизируйте видимость классов и методов
- Минимизируйте мутабельность
  - Лучше разделять методы на те, которые возвращают состояние и те, которые его меняют
- Не бойтесь принимать и возвращать функции
- Предпочитайте интерфейсы абстрактным классам
- Не используйте “тэги типов”
- Не используйте без нужды вложенные классы
- Проверяйте параметры public-методов на валидность
  - Помните о null-ах
- Fail fast

Общепрограммистские рекомендации

- Минимизируйте области видимости
- Не используйте числа с плавающей точкой там, где нужны точные значения
  - Сравнивать числа с плавающей точкой через == нельзя
- Знайте и используйте библиотеки
- Используйте исключения только в исключительных ситуациях
- Не игнорируйте исключения