nagarro

06

Clean Code

What Is the Purpose of the Following Code?

```
public List<string> SortForPrinting(List<Person> people)
{
    List<string> result = new List<string>();
    List<string> list = new List<string>();
    for (int i = 0; i < people.Count; i++)
    {
        Person person = people[i];
        list.Add(person.Birthday + " " + person.FamilyName);
    }
    list.Sort();

    try
    {
        for (int i = 0; i < list.Count; i++)
        {
            string str = list[i];
            result.Add(str.Substring(str.LastIndexOf(" ") + 1) + " " +
                              str.Substring(0, str.LastIndexOf(" ")));
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.StackTrace);
    }

    return result;
}
```

Look Again. Is this Better?

```csharp
public IEnumerable<string> GetSortedPrintRepresentation(List<Person> persons)
{
    var personsSorted = SortByAgeAndFamilyName(persons);
    var personsFormatted = FormatPersonsForPrinting(personsSorted);

    return personsFormatted;
}


private IOrderedEnumerable<Person> SortByAgeAndFamilyName(List<Person> persons)
{
    return persons.OrderBy(p => p.Birthday).ThenBy(p => p.FamilyName);
}

private IEnumerable<string> FormatPersonsForPrinting(IOrderedEnumerable<Person> persons)
{
    return persons.Select(p => p.FamilyName + " " + p.Birthday);
}
```

What Is the Purpose of the Following Code?

```
public List<int[]> GetThem()
{
    List<int[]> list1 = new List<int[]>();
    foreach (int[] x in TheList)

    {
        if (x[0] == 4)
        {
            list1.Add(x);
        }
    }
    return list1;
}
```

Look Again. Is this Better?

```
public List<int[]> GetFlaggedCells()
{
    var flaggedCells = new List<int[]>();
    foreach (int[] cell in GameBoard)
    {
        if (IsFlagged(cell))
        {
            flaggedCells.Add(cell);
        }
    }
    return flaggedCells;
}
```

…or Better Still?

```
public List<Cell> GetFlaggedCells()
{
    var flaggedCells = new List<Cell>();
    foreach (Cell cell in GameBoard)
    {
        if (cell.IsFlagged())
        {
            flaggedCells.Add(cell);
        }
    }
    return flaggedCells;
}
```

Clean Code

„Any fool can write code that computers understand. Good programmers write code than humans can understand.”

Martin Fowler

Clean Code

*„Clean code always looks like
it was written by someone who cares.“*

Dave Thomas

Clean Code

„Clean code can be read,
and enhanced by a developer
other than its original author.
It has unit and acceptance tests.
It has meaningful names…“

Dave Thomas

Clean Code

„Clean code is simple and direct.
Clean code reads like well-written prose.
Clean code never obscures the designer‘s intent
but rather is full of crisp abstractions
and straightforward lines of control.“

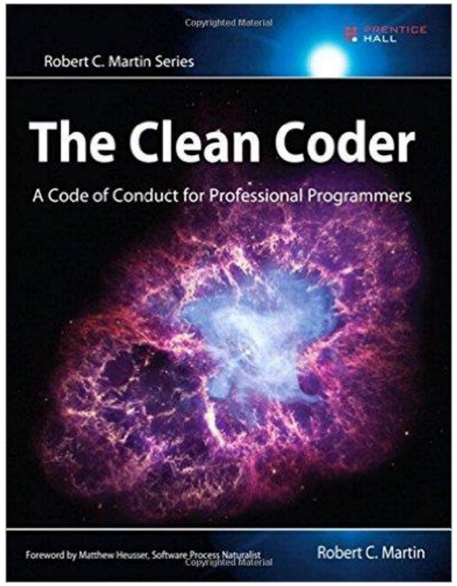
Grady Booch

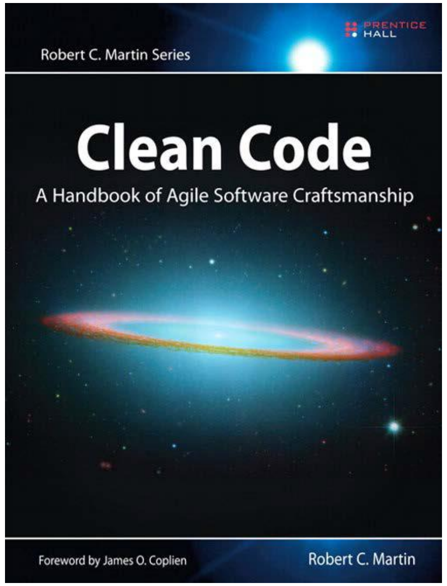
Origin

Two books by Robert C. Martin

- Clean Code: A Handbook of Agile Software Craftsmanship 2008
- The Clean Coder: A Code of Conduct for Professional Programmers 2011

These two book belong together and explain how and why a programmer should write clean code.




PRENTICE HALL
Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin

Write Unit Tests

Use meaningful names

Remove duplications (DRY)

Reduce complexity (KISS)

Clean Code Details

- **Meaningful names**
  Expressiveness of the code means that it has meaningful names. These names should express their intention. They should not mislead you.
  Self-explanatory: (similar to expressive => intention is clear), readable, has one level of abstraction

- **No duplications**
  It should comply with the DRY rule (Don't Repeat Yourself). When the DRY principle has successfully been applied, the modification of any single element of a system doesn't require a change in any other logically unrelated elements.

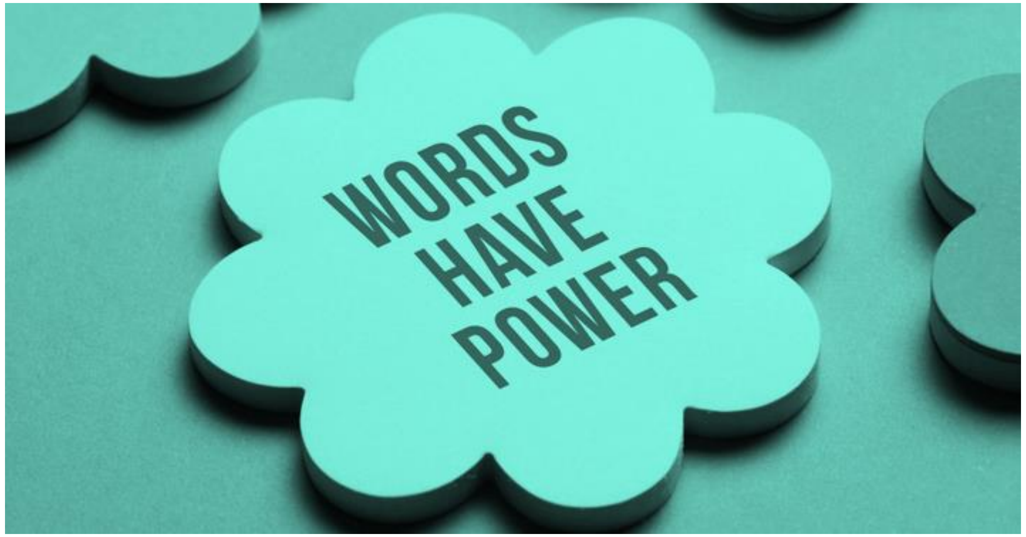
Clean Code Details

- **Small**
  Smaller is better. Code should be minimal. Both classes and methods should be short, preferably just a few lines of code. It should be well divided (also within one class). The better you divide your code the easier it becomes to read it.

- **Focused**
  A functional unit on a given level of abstraction should only be responsible for a single aspect of a system's requirements. Each class, method or any other entity should remain undisturbed. It should conform to SRP (Single Responsibility Principle).

Use Meaningful Names

Use self-explanatory names for classes, methods and variables



- No abbreviations
- Describe the intention – specific not general
- Pick one word per concept
- Use solution domain names wherever possible
- Use named constants instead of magic numbers/strings

Meaningful Names - Examples

Bad

```
static int STO = 4000;

DateTime modymdhms;

DoIt();

public class DtaRcrd102

int Calc(int x1, int x2)
```

Good

```
static int SocketTimeoutInMillisec = 4000;

DateTime modificationTimestamp;

PostPayment();

public class PremiumCustomer

int Power(int base, int exponent)
```

Clean Methods / Functions

Small (< 150 characters by line AND < 20 lines of code)

- Do one thing
- Same level of abstraction
- Few arguments
- Avoid side effects
- Command or Query

Same Level of Abstraction in Method – Example

Bad

```
public Report BuildReport(HashSet<CandidateResult> resultSet)
{
    Report report = new Report();
    foreach (CandidateResult result in resultSet)
    {
        ReportElement reportElement = new ReportElement();
        reportElement.SetEnglishScore(result.GetEnglishScore());
        reportElement.SetMathScore(result.GetMathScore());
        reportElement.SetMeanScore((result.GetEnglishScore() +
                                        result.GetMathScore()) / 2);
        report.Add(reportElement);
    }
    return report;
}
```

Same Level of Abstraction in Method – Example (cont.)

Good

```
public Report BuildReport(HashSet<CandidateResult> resultSet)
{
    Report report = new Report();
    foreach (CandidateResult result in resultSet)
    {
        ReportElement reportElement = BuildReportElement(result);
        report.Add(reportElement);
    }
    return report;
}

private ReportElement BuildReportElement(CandidateResult result)
{
    ReportElement element = new ReportElement();
    element.SetEnglishScore(result.GetEnglishScore());
    element.SetMathScore(result.GetMathScore());
    element.SetMeanScore((result.GetEnglishScore() + result.GetMathScore()) / 2);
    return element;
}
```

Clean Classes

Single Responsibiltiy

- Small
- Single Responsibiltiy
  - Class should have one, and only one, reason to change

Single Responsibility - Example

Bad

```
public class EmployeeManager {
    public Money CalculatePay();
    public void Save();
    public string ReportHours();
}
```

Good

```
public class EmployeeRepository {
     public Save(Employee e) {...}
 }

 public class EmployeeFinance {
     public Money CalculatePay(Employee e) {...}
 }

 public class EmployeeReporting {
     public string ReportHours(Employee e) {...}
 }
```

Single Responsibilty – Sample Explanation

- We can check Single Responsibility by thinking about who (i.e. stakeholder) is responsible for this module/class?
  When we look at C-level executives (CFO, CTO, COO), we will see that more than on stakeholder is responsible:
  - calculatePay(...) method implements the algorithms that determine how much a particular employee should be paid, based on that employee's contract, status, hours worked, etc
    - CFO’s team may have new requirements for this topic
  - save(...) method stores the data managed by the Employee object onto the enterprise database
    - CTO’s team may have new requirements for this topic
  - reportHours(...) method returns a string which is appended to a report that auditors use to ensure that employees are working the appropriate number of hours and are being paid the appropriate compensation.
    - COO’s team may have new requirements for this topic

Remove Duplications

- Reuse existing methods
- Obey common patterns and conventions
- Look out for same code
- Duplications are not just pattern matching

Write Unit Tests

Use meaningful names

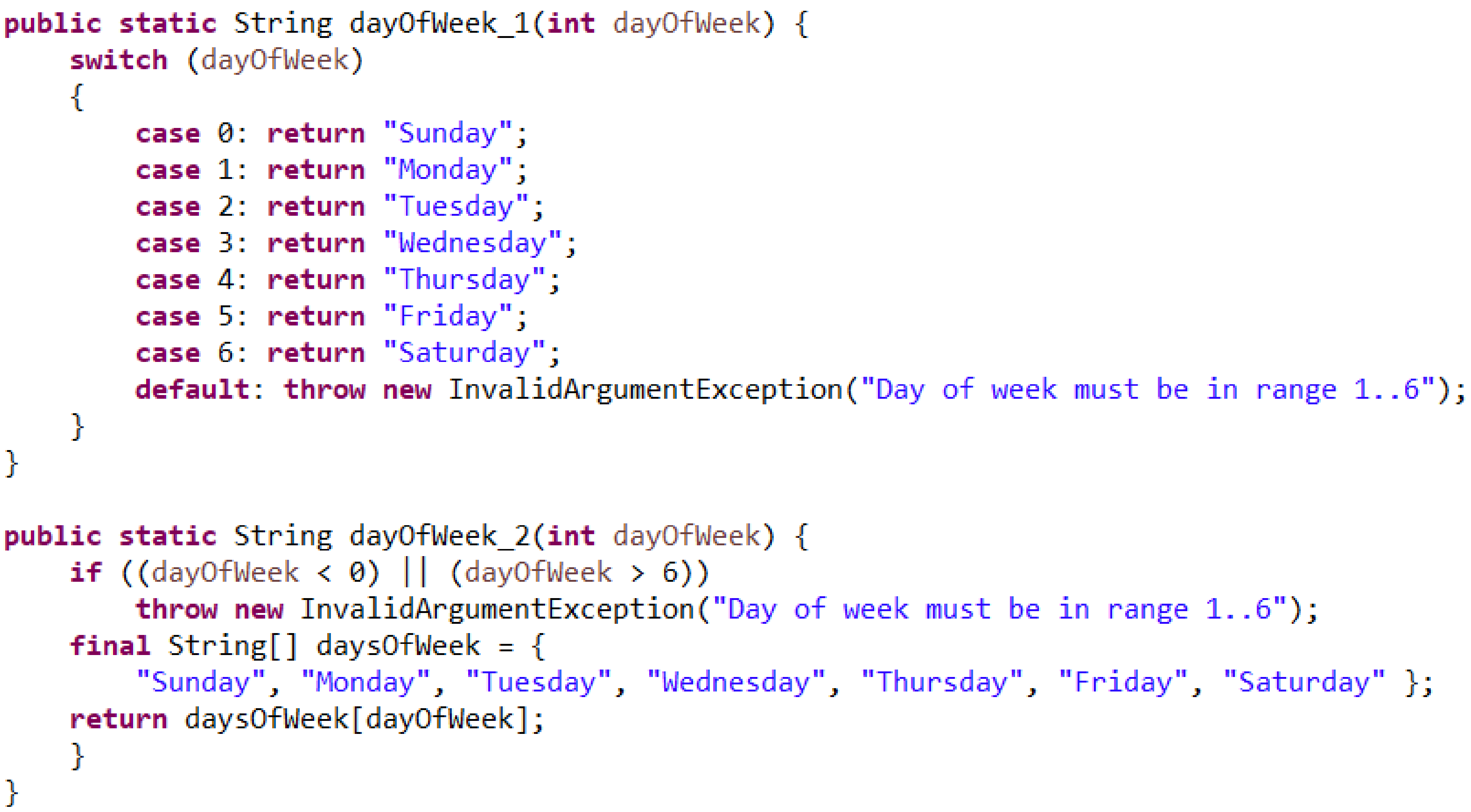
Remove duplications (DRY)

Reduce complexity (KISS)

Reduce complexity (KISS)

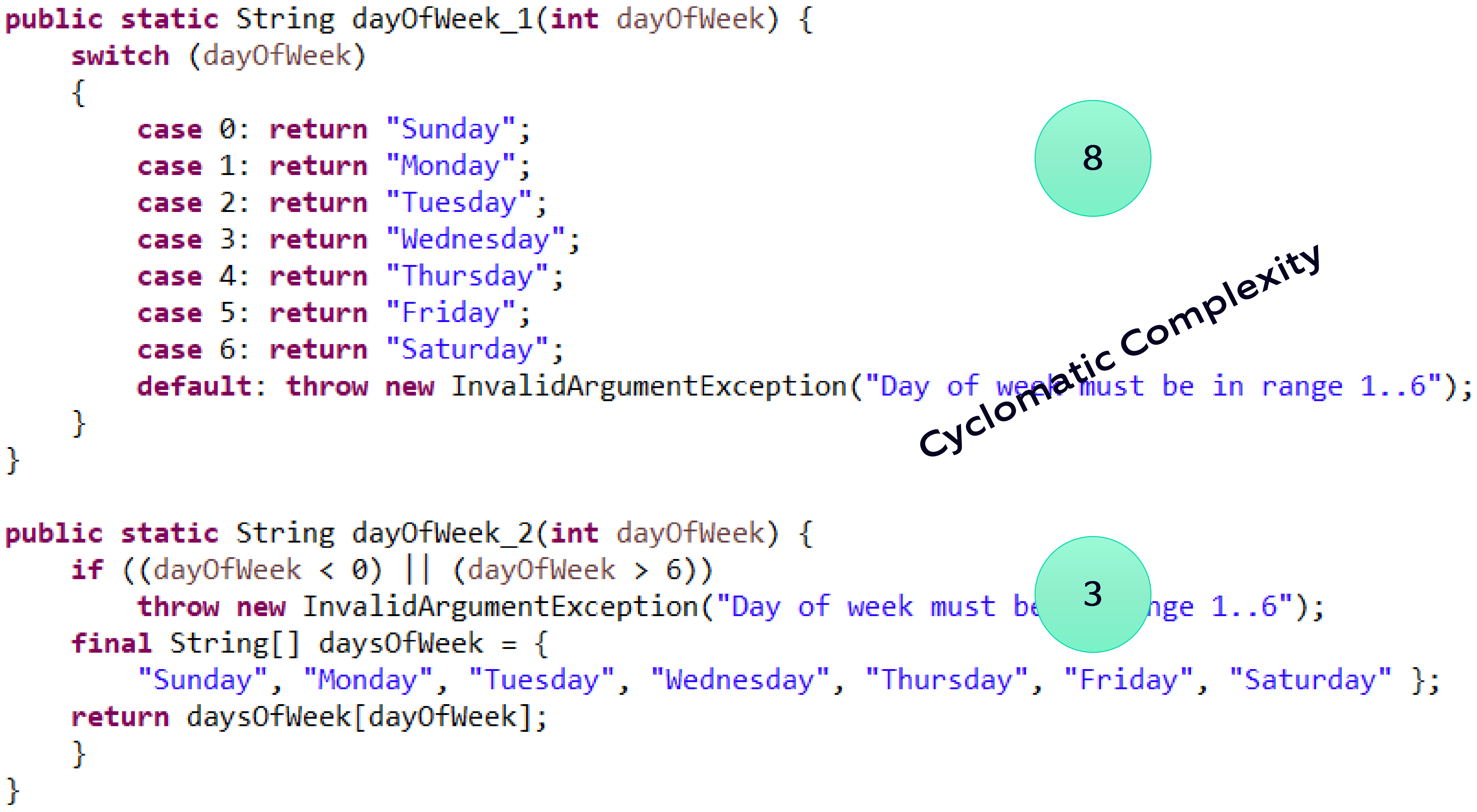
- Reduce your nesting levels
- Design your code with the same level of abstraction in mind
- Prefer to use simple concepts and language specific “short-hands”
- Avoid large conditional blocks
- Avoid negations in if-else statements
- Reduce the distance between declaration and usage
- Delete dead code

Complexity – Different Viewpoints



```
public static String dayOfWeek_1(int dayOfWeek) {
    switch (dayOfWeek)
    {
        case 0: return "Sunday";
        case 1: return "Monday";
        case 2: return "Tuesday";
        case 3: return "Wednesday";
        case 4: return "Thursday";
        case 5: return "Friday";
        case 6: return "Saturday";
        default: throw new InvalidArgumentException("Day of week must be in range 1..6");
    }
}

public static String dayOfWeek_2(int dayOfWeek) {
    if ((dayOfWeek < 0) || (dayOfWeek > 6))
        throw new InvalidArgumentException("Day of week must be in range 1..6");
    final String[] daysOfWeek = {
        "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday" };
    return daysOfWeek[dayOfWeek];
    }
}
```

Complexity – Different Viewpoints



```
public static String dayOfWeek_1(int dayOfWeek) {
    switch (dayOfWeek)
    {
        case 0: return "Sunday";
        case 1: return "Monday";
        case 2: return "Tuesday";
        case 3: return "Wednesday";
        case 4: return "Thursday";
        case 5: return "Friday";
        case 6: return "Saturday";
        default: throw new InvalidArgumentException("Day of week must be in range 1..6");
    }
}

public static String dayOfWeek_2(int dayOfWeek) {
    if ((dayOfWeek < 0) || (dayOfWeek > 6))
        throw new InvalidArgumentException("Day of week must be in range 1..6");
    final String[] daysOfWeek = {
        "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday" };
    return daysOfWeek[dayOfWeek];
    }
}
```

8

Cyclomatic Complexity

3

Cyclomatic Complexity vs. Cognitive Complexity

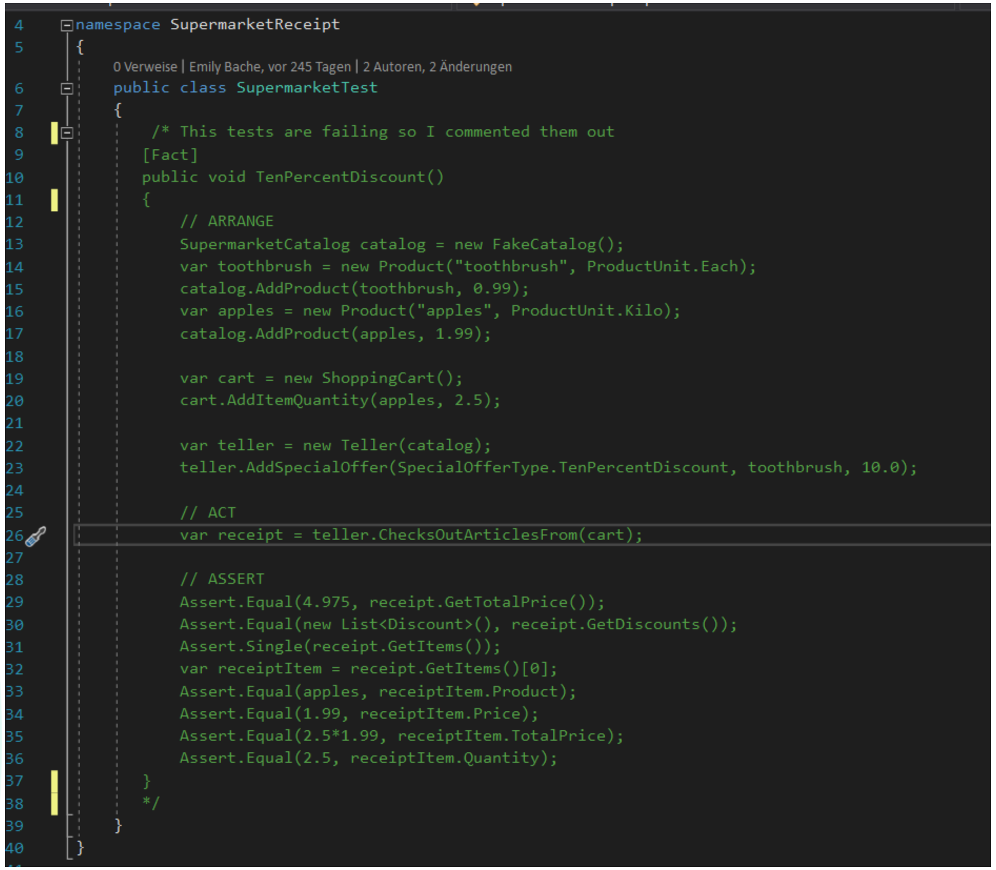
```
public double CalculateDiscount(double price, double discount, double taxRate)
{
    if (price <= 0 || discount <= 0 || taxRate <= 0)
        return 0.0;

    return (price * (discount / 100) * taxRate);
}

public double CalculateDiscount2(double price, double discount)
{
    if (! (price <= 0))
    {
        if (! (discount <= 0))
        {
            if (! (taxRate <= 0))
            {
                return (price * (discount / 100) * taxRate);
            }
            return 0.0;
        }
        return 0.0;
    }
    return 0.0;
}
```

Cyclomatic Complexity	Cognitive Complexity
4	3
4	6

Cognitive Complexity considers the *cognitive effort* required to understand the flows

Comments

Comments are some kind of dead code



```
namespace SupermarketReceipt
{
    0 Verweise | Emily Bache, vor 245 Tagen | 2 Autoren, 2 Änderungen
    public class SupermarketTest
    {
         /* This tests are failing so I commented them out
        [Fact]
        public void TenPercentDiscount()
        {
            // ARRANGE
            SupermarketCatalog catalog = new FakeCatalog();
            var toothbrush = new Product("toothbrush", ProductUnit.Each);
            catalog.AddProduct(toothbrush, 0.99);
            var apples = new Product("apples", ProductUnit.Kilo);
            catalog.AddProduct(apples, 1.99);

            var cart = new ShoppingCart();
            cart.AddItemQuantity(apples, 2.5);

            var teller = new Teller(catalog);
            teller.AddSpecialOffer(SpecialOfferType.TenPercentDiscount, toothbrush, 10.0);

            // ACT
            var receipt = teller.ChecksOutArticlesFrom(cart);

            // ASSERT
            Assert.Equal(4.975, receipt.GetTotalPrice());
            Assert.Equal(new List<Discount>(), receipt.GetDiscounts());
            Assert.Single(receipt.GetItems());
            var receiptItem = receipt.GetItems()[0];
            Assert.Equal(apples, receiptItem.Product);
            Assert.Equal(1.99, receiptItem.Price);
            Assert.Equal(2.5*1.99, receiptItem.TotalPrice);
            Assert.Equal(2.5, receiptItem.Quantity);
        }
        */
    }
}
```

- Use comments wisely
- Use comments to explain the why and not the what
- Avoid redundancy
- No disinformation
  - Priciple of Least Astonishment
- Don‘t use it as your history

Formatting / Layouting

- Use the same layout / code formatting in your code
- Your code tells a story: organize your code accordingly
  - members, constructors, public and in order of call sequence private methods

- Every programmer has his own favorite formatting rules
  - ...but if he works in a team then the team rules

Important Design Principles

S.O.L.I.D.

- Single Responsibiltiy Principle
- Open Closed Principle
- Liskov's Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle

Other Principles

- POLA or POLS (Principle of Least Astonishment/Surprise)
- Law of Demeter
- Simplicty
  - YAGNI (You Aren't Gonna Need It)
  - KISS (Keep It Simple, Stupid.)

Keep Your Code Clean

Boy Scout Rule

- Always invest a small percentage of your time
- If you don't care nobody else will care
- Never ask for permission to do your job correctly!

Broken Window Theory

- Take some action to prevent further damage as soon as it is discovered