

```
1 int d;  
2 // elapsed time in days  
3 int ds;  
4 int dsm;  
5 int faid;
```



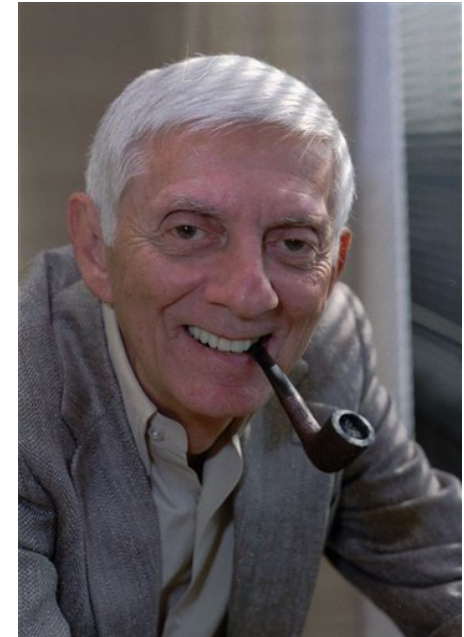
Names must reveal your intentions

```
1 int elapsedTimeInDays;  
2 int daysSinceCreation;  
3 int daysSinceModification;  
4 int fileAgeInDays;
```

```
1 Customer[] customerList;  
2 Table theTable;
```

Avoid Disinformation

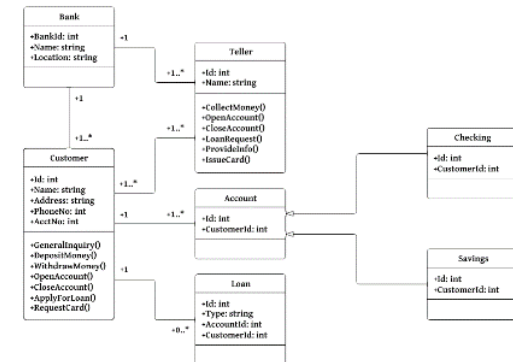
```
1 Customer[] customers;  
2 Table customers;
```



```
1 string addressCity;  
2 string addressHomeNumber;  
3 string addressPostCode;
```

Use meaningful names in their self context

```
1 class Address  
2 {  
3     string city;  
4     string homeNumber;  
5     string postCode;  
6 }
```



```
1 var theCustomersListWithAllCustomersIncludedWithoutFilter;  
2 bool visibleStateCheckWhenCustomerAccessGranted;
```

Good names length

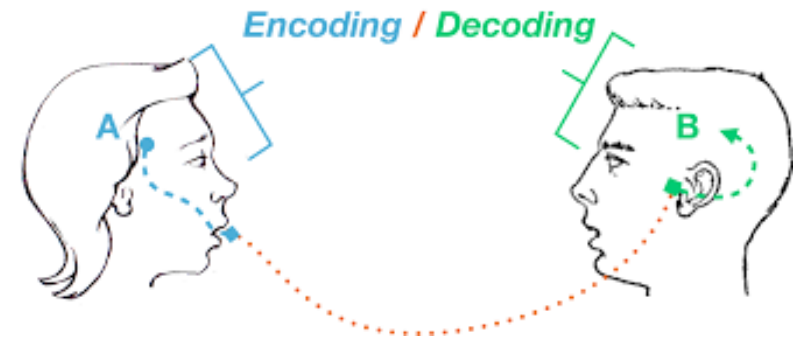
```
1 var allCustomers;  
2 bool isVisible;
```



```
1 private string m_strExePath;
```

Avoid Encodings

```
1 private string executablePath;
```



Class Names

- Classes and objects should have noun or noun-phrase names
- A class name should not be a verb.



Java class names from Spring Framework :

- SimpleBeanFactoryAwareAspectInstanceFactory
- AbstractInterceptorDrivenBeanDefinitionDecorator
- AbstractSingletonProxyFactoryBean

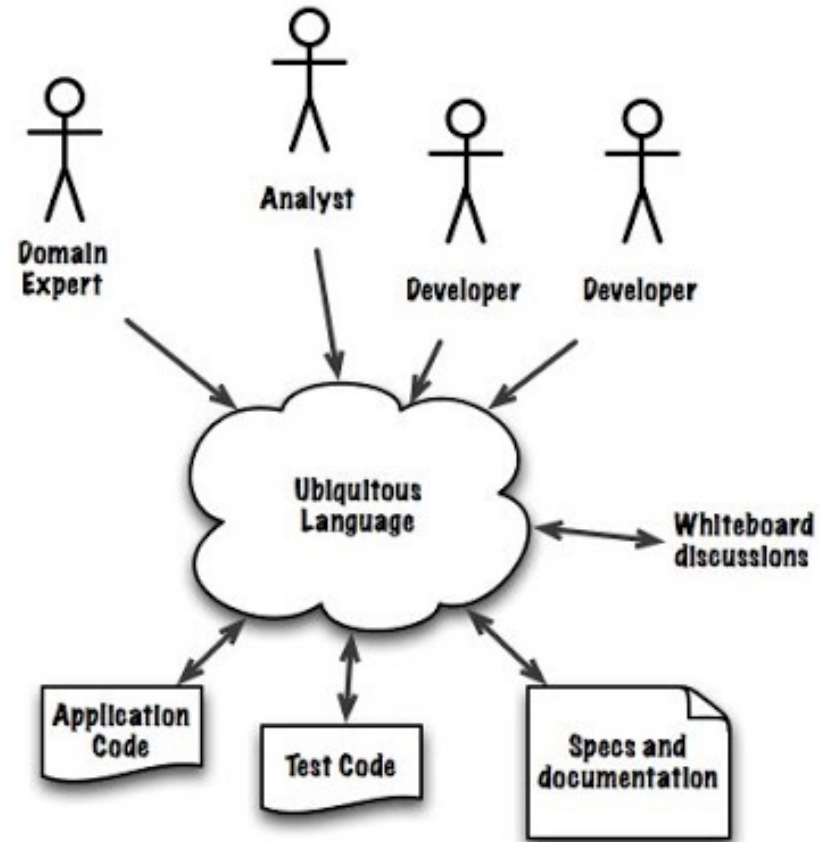
Use a ubiquitous language

Language used :

By all team members

By all business experts

In the source code as well



```
1 // The day of the year.  
2 int dayOfYear = DateTime.Today.DayOfYear;
```

```
1 /// <summary>  
2 /// Send email  
3 /// </summary>  
4 private void SendEmail()  
5 {  
6     // ...  
7     // ...  
8 }
```

Redundant comments


```
1 /// <summary>
2 /// Utility method that returns when this.closed is true.
3 /// </summary>
4 /// <param name="timeoutInMilliseconds"></param>
5 public void WaitForClose(long timeoutInMilliseconds)
6 {
7     if (!closed)
8     {
9         Wait(timeoutInMilliseconds);
10
11         if (!closed)
12         {
13             throw new Exception("Sender could not be closed");
14         }
15     }
16 }
```

Misleading comments
can contain lies

```
1 private void Persist<TEntity>(TEntity entity)
2 {
3     Save(entity);
4     //entity.Load();
5     //entity.Prepare();
6
7     /*if(state == true)
8     {
9
10    }*/
11 }
```

Don't leave commented out code in your codebase

```
1 private void Persist<TEntity>(TEntity entity)
2 {
3     Save(entity);
4 }
```

```
1 /*
2 2016-10-09: Remove dead code (Yot)
3 2016-11-01: Improve performance (AM)
4 2016-11-03: Makes the method async (AM)
5 */
6 private async Task Persist<TEntity>(TEntity entity)
7 {
8     await Save(entity);
9 }
```

Don't have journal comments

Use your source control instead



```
1 private async Task Persist<TEntity>(TEntity entity)
2 {
3     await Save(entity);
4 }
```

```
1 /// <summary>
2 /// This class describes a person.
3 /// </summary>
4 internal class Person
5 {
6     /// <summary>
7     /// Initializes a new Person instance.
8     /// </summary>
9     public Person()
10    {
11    }
12
13    /// <summary>
14    /// Gets or sets the person Identifier.
15    /// </summary>
16    public Guid Id { get; set; }
17
18    /// <summary>
19    /// Gets or sets the person Name.
20    /// </summary>
21    public string Name { get; set; }
22
23    // ...
24    // ...
25    // ...
26 }
```

Comments are noise

Public API comments

```
1 /// <summary>
2 /// Remove all items from this set. This clears the elements but not the underlying
3 /// buckets and slots array. Follow this call by TrimExcess to release these.
4 /// </summary>
5 public void Clear()
6 {
7     if (_lastIndex > 0)
8     {
9         //..
10    }
11    _version++;
12 }
```

Legal comments

```
1 // Licensed to the .NET Foundation under one or more agreements.  
2 // The .NET Foundation licenses this file to you under the MIT license.  
3 // See the LICENSE file in the project root for more information.
```

TODO comments

```
1 //TODO : Refactor this method to make it async
2 private void Persist<TEntity>(TEntity entity)
3 {
4     Save(entity);
5 }
```

Task List

Description

TODO : Refactor this method to make it async

Explanation of intent

```
1 public virtual void OnDeserialization(Object sender)
2 {
3     if (_siInfo == null)
4     {
5         // It might be necessary to call OnDeserialization from a container if the
6         // container object also implements OnDeserialization. We can return immediately
7         // if this function is called twice. Note we set _siInfo to null at the end of this method.
8         return;
9     }
10    //..
11 }
```


Use consistent capitalization

```

1 public class person
2 {
3     private const string CATEGORY = "P";
4     private string _name;
5
6     public void pay(decimal amount)
7     {
8         //...
9     }
10 }
11
12 public class Client
13 {
14     private const string category = "C";
15     private string name;
16
17     public void Pay(decimal amount)
18     {
19         //...
20     }
21 }

```

```

1 public class Person
2 {
3     private const string CATEGORY = "P";
4     private string name;
5
6     public void Pay(decimal amount)
7     {
8         //...
9     }
10 }
11
12 public class Client
13 {
14     private const string CATEGORY = "C";
15     private string name;
16
17     public void Pay(decimal amount)
18     {
19         //...
20     }
21 }

```

Unsustainable Spacing

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

- Individuals and interactions over process
- Working software over comprehensive d
- Customer collaboration over contract ne
- Responding to change over following a plan

Indentation can reveal things

```

1 public void Play()
2 {
3     var players = new List<Player>();
4
5     foreach(var player in players)
6     {
7         foreach(var game in player.Games)
8         {
9             foreach(var bet in game.Bets)
10            {
11                foreach(var sport in bet.Sports)
12                {
13                }
14            }
15        }
16    }
17 }
18 }

```

Cyclomatic complexity :

- Metric to indicate the complexity of a piece of code
- Measure the number of independent paths through our code

```
1 private string ArabicNumeralToRoman(int num)
2 {
3     string val = "";
4
5     while (num >= 1000)
6     {
7         val += "M";
8         num -= 1000;
9     }
10    if (num >= 900)
11    {
12        val += "CM";
13        num -= 900;
14    }
15    while (num >= 500)
16    {
17        val += "D";
18        num -= 500;
19    }
20    if (num >= 400)
21    {
22        val += "CD";
23        num -= 400;
24    }
25    while (num >= 100)
26    {
27        val += "C";
28        num -= 100;
29    }
30    if (num >= 90)
31    {
32        val += "XC";
33        num -= 90;
34    }
```



Functions Should be Small

(no longer than about 6 or so lines long)

```
1 public void RegisterUser(string email, string name)
2 {
3     var regularExpression = new Regex(@"/^([\w-]+(?:\.[\w-]+)*)@((?![\w-]+\.)*\w[\w-]{0,66})\.([a-z]{2,6}(?:\.[a-z]{2})?)$/i");
4
5     if(!regularExpression.IsMatch(email))
6     {
7         throw new InvalidOperationException("email invalid");
8     }
9
10    var user = new User(email, name);
11    Register(user);
12 }
```

Functions Should Do Only One Thing



```
1 public void RegisterUser(string email, string name)
2 {
3     CheckEmail(email);
4     var user = new User(email, name);
5     Register(user);
6 }
7
8 private void CheckEmail(string email)
9 {
10    var regularExpression = new Regex(@"/^([\w-]+(?:\.[\w-]+)*)@((?![\w-]+\.)*\w[\w-]{0,66})\.([a-z]{2,6}(?:\.[a-z]{2})?)$/i");
11
12    if (!regularExpression.IsMatch(email))
13    {
14        throw new InvalidOperationException("email invalid");
15    }
16 }
```

```
1 public double Convert(double value)
2 {
3     return (value - 32) * 5 / 9;
4 }
```

Use Descriptive Names



```
1 public double ConvertToCelcius(double fahrenheit)
2 {
3     return (fahrenheit - 32) * 5 / 9;
4 }
```

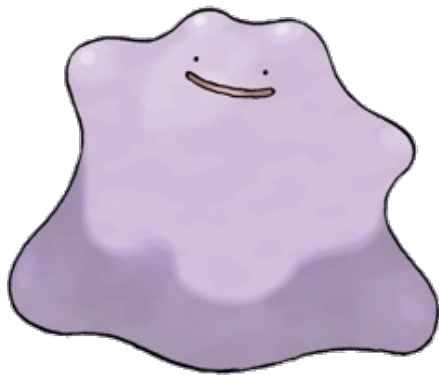
Command/Query Separation

```
1 private string CalculateRental()
2 {
3     string result = "";
4
5     foreach (var rental in this.rentals)
6     {
7         var thisAmount = rental.LineAmount();
8         this.FrequentRenterPoints += rental.CalculateFrequentPoints();
9         this.AmountOwed = this.AmountOwed + thisAmount;
10
11         result += FormatLine(rental, thisAmount);
12     }
13
14     return result;
15 }
```

Tell-Don't-Ask principle

Replace Conditional with Polymorphism

```
class Bird {  
    //...  
    double getSpeed() {  
        switch (type) {  
            case EUROPEAN:  
                return getBaseSpeed();  
            case AFRICAN:  
                return getBaseSpeed() - getLoadFactor() * numberOfCoconuts;  
            case NORWEGIAN_BLUE:  
                return (isNailed) ? 0 : getBaseSpeed(voltage);  
        }  
        throw new RuntimeException("Should be unreachable");  
    }  
}
```



```
abstract class Bird {  
    //...  
    abstract double getSpeed();  
}  
  
class European extends Bird {  
    double getSpeed() {  
        return getBaseSpeed();  
    }  
}  
  
class African extends Bird {  
    double getSpeed() {  
        return getBaseSpeed() - getLoadFactor() * numberOfCoconuts;  
    }  
}  
  
class NorwegianBlue extends Bird {  
    double getSpeed() {  
        return (isNailed) ? 0 : getBaseSpeed(voltage);  
    }  
}  
  
// Somewhere in client code  
speed = bird.getSpeed();
```


Prefer Exceptions to Returning Error Codes

Before

```
int Withdraw(int amount)
{
    if (amount > _balance)
    {
        return -1;
    }
    else
    {
        balance -= amount;
        return 0;
    }
}
```

After

```
void Withdraw(int amount)
{
    if (amount > _balance)
    {
        throw new BalanceException();
    }
    balance -= amount;
}
```

Or Monads...