```
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;
```

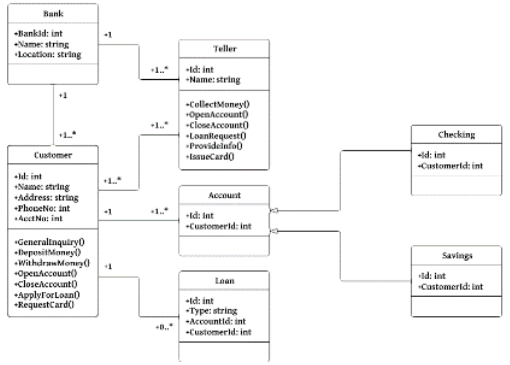
Avoid Disinformation

```
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 }
```



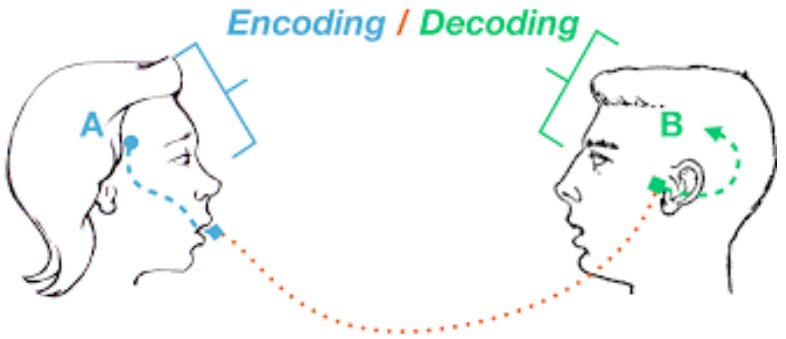
```
1 var theCustomersListWithAllCustomersIncludedWithoutFilter;
2 bool visibleStateCheckWhenCustomerAccessGranted;
```

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 :

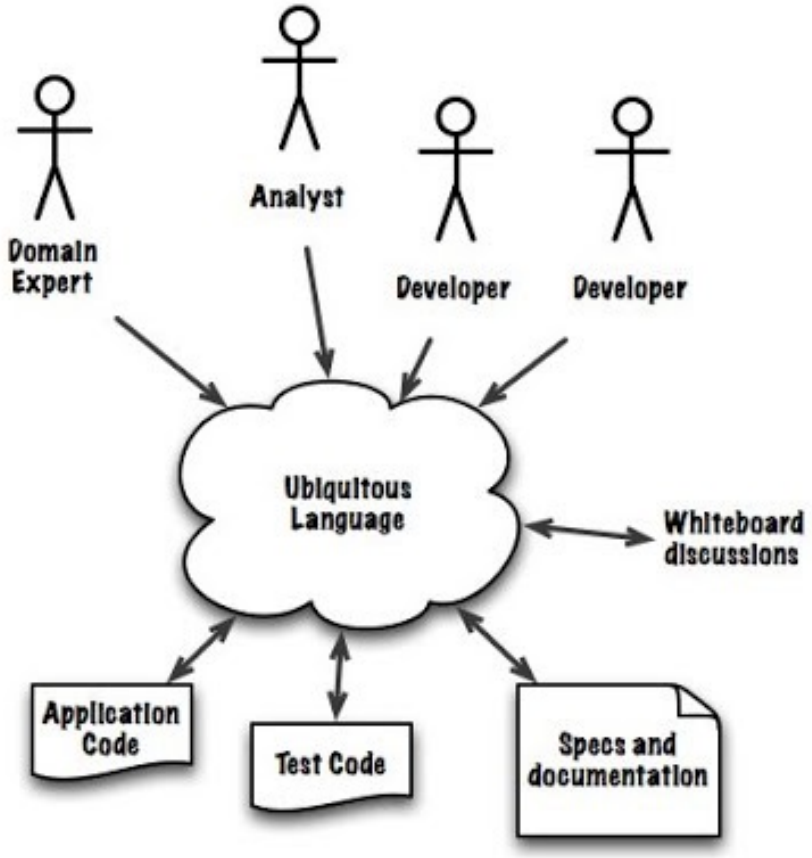
- SimpleBeanFactoryAwareAspectInstanceFactory
- AbstractInterceptorDrivenBeanDefinitionDecorator
- AbstractSingletonProxyFactoryBean

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

COMMENTS

```
private void Persist<TEntity>(TEntity entity)
{
    Save(entity);
    //entity.Load();
    //entity.Prepare();

    /*if(state == true)
    {

    }*/
}
```

Don't leave commented out code in your codebase

```
private void Persist<TEntity>(TEntity entity)
{
    Save(entity);
}
```

COMMENTS

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

Don't have journal comments

Use your source control instead

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

```
/// <summary>
/// This class describes a person.
/// </summary>
internal class Person
{
    /// <summary>
    /// Initializes a new Person instance.
    /// </summary>
    public Person()
    {
    }

    /// <summary>
    /// Gets or sets the person Identifier.
    /// </summary>
    public Guid Id { get; set; }

    /// <summary>
    /// Gets or sets the person Name.
    /// </summary>
    public string Name { get; set; }

    // ...
    // ...
    // ...
}
```

Comments are noise

COMMENTS

Public API comments

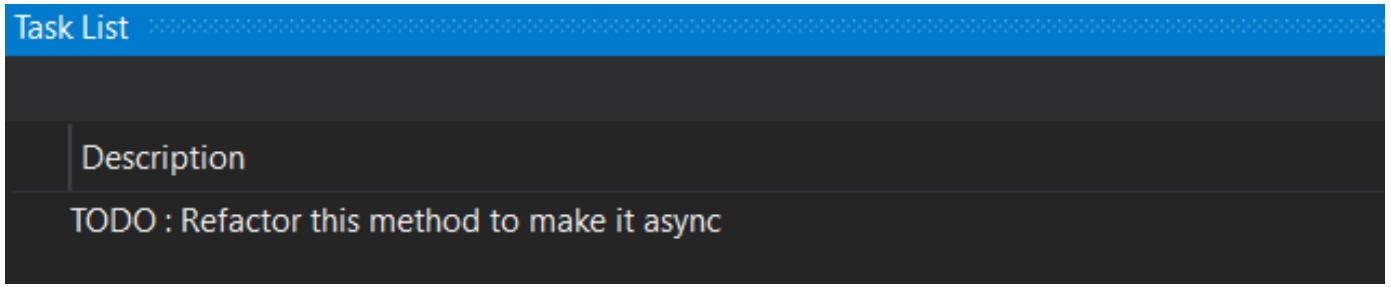
```
1  /// <summary>
2  /// Remove all items from this set. This clears the elements but not the underlying
3  /// buckets and slots array. Follow this call by TrimExcess to release these.
4  /// </summary>
5  public void Clear()
6  {
7      if (_lastIndex > 0)
8      {
9          //..
10     }
11  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



COMMENTS

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 }
```

```
public class person
{
    private const string CATEGORY = "P";
    private string _name;

    public void pay(decimal amount)
    {
      //...
    }
}

public class Client
{
    private const string category = "C";
    private string name;

    public void Pay(decimal amount)
    {
      //...
    }
}
```

Use consistent capitalization

```
public class Person
{
    private const string CATEGORY = "P";
    private string name;

    public void Pay(decimal amount)
    {
        //...
    }
}

public class Client
{
    private const string CATEGORY = "C";
    private string name;

    public void Pay(decimal amount)
    {
        //...
    }
}
```

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

```
private string ArabicNumeralToRoman(int num)
{
    string val = "";

    while (num >= 1000)
    {
        val += "M";
        num -= 1000;
    }
    if (num >= 900)
    {
        val += "CM";
        num -= 900;
    }
    while (num >= 500)
    {
        val += "D";
        num -= 500;
    }
    if (num >= 400)
    {
        val += "CD";
        num -= 400;
    }
    while (num >= 100)
    {
        val += "C";
        num -= 100;
    }
    if (num >= 90)
    {
        val += "XC";
        num -= (0;
    }
```

Functions Should be Small

(no longer than about 6 or so lines long)

```
public void RegisterUser(string email, string name)
{
    var regularExpression = new Regex(@"/^([\w-]+(?:\.[\w-]+)*)@((?:[\w-]+\.)*\w[\w-]{0,66})\.([a-z]{2,6}(?:\.[a-z]{2})?)$/i;");

    if(!regularExpression.IsMatch(email))
    {
        throw new InvalidOperationException("email invalid");
    }

    var user = new User(email, name);
    Register(user);
}
```

Functions Should Do Only One Thing


```
public void RegisterUser(string email, string name)
{
    CheckEmail(email);
    var user = new User(email, name);
    Register(user);
}

private void CheckEmail(string email)
{
    var regularExpression = new Regex(@"/^([\w-]+(?:\.[\w-]+)*)@((?:[\w-]+\.)*\w[\w-]{0,66})\.([a-z]{2,6}(?:\.[a-z]{2})?)$/i;");

    if (!regularExpression.IsMatch(email))
    {
        throw new InvalidOperationException("email invalid");
    }
}
```

```
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 }
```

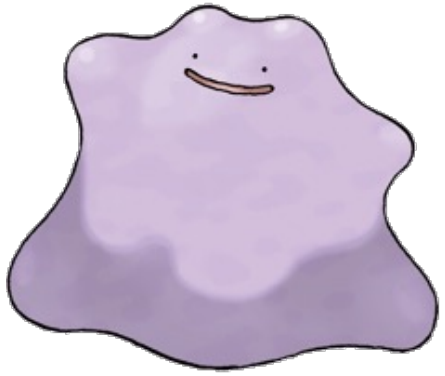
Command/Query Separation

```
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.CalculateFrecuentPoints();
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…