***Before reading this article, think about this – the product which you will provide, is what the customer sees, but as a programmers, we should care not only for the customer, but for our co-workers as well. We must work in way, that when someone looks our code, it should be clear and understandable.***

# **Style of writing.**

This topic is underestimated, but this is the basic of the clean code.

# **Noisy Code**

This is code, which contains a lot of **obsolete rows,** which don’t bring anything useful. The code should be clear and understandable, without things, which can confuse the person who read the code. Noise code are comments, dead code (code which we have, because the requirements have been changed many times, copy-paste, or just unused), variables which we don’t use any more, default property values, which are not in the constructor and etc.

# **Comments – code should explain itself**

Comments are something which is better to avoid when we can. A good point of this is that when we see bad code, and we ask the person who wrote the code to comment it to became understandable, something is wrong. Why? What we really want? We want from the person who wrote that bad code to write a good comment – I don’t think so. Most likely we will see the same mess in the comments like the mess in the code because the code is the understanding of this person of the problem. Another good point is that the comments aren’t checked by the compiler and they can be wrong. So, you cannot rely that the comments are correct. Most of the programmers read comments when they have a problem, not before that. And if they made a change, most of them don’t update the comments as well.

# **Code alignment**

This is something which is specific. I can recommend you the following. **Don’t favour a particular style, favour a properties and benefits which brings you this style**. Let’s look this closer. Let’s look method definitions and methods invocations because we make a lot of problems there.

**Code must not be breakable (it should stand the changes).** Changing of a method name for example, should not break the alignment of the params. Look the next example:

**public bool VeryImportantMethod (int userId,**

**string username)**

**{**

**//some code**

**}**

Its possible if we remove userid or we move the userid to the next row, when we try to merge, to bring merge conflicts with this change. Or if two developers merge their changes into the codebase, the developer which have a version of the code without userId (and it should be the correct code), to merge the version with the userId, or to switch two string params or something else.

Here is an example how we can refactor this code, in a way which can help us with that kind of problems.

**public bool VeryImportantMethod (**

**int userId,**

**string username)**

**{**

**//some code**

**}**

Same advice is working for invoking a method as well.

**bool success = VeryImportantMethod(userId, username);**

**VS**

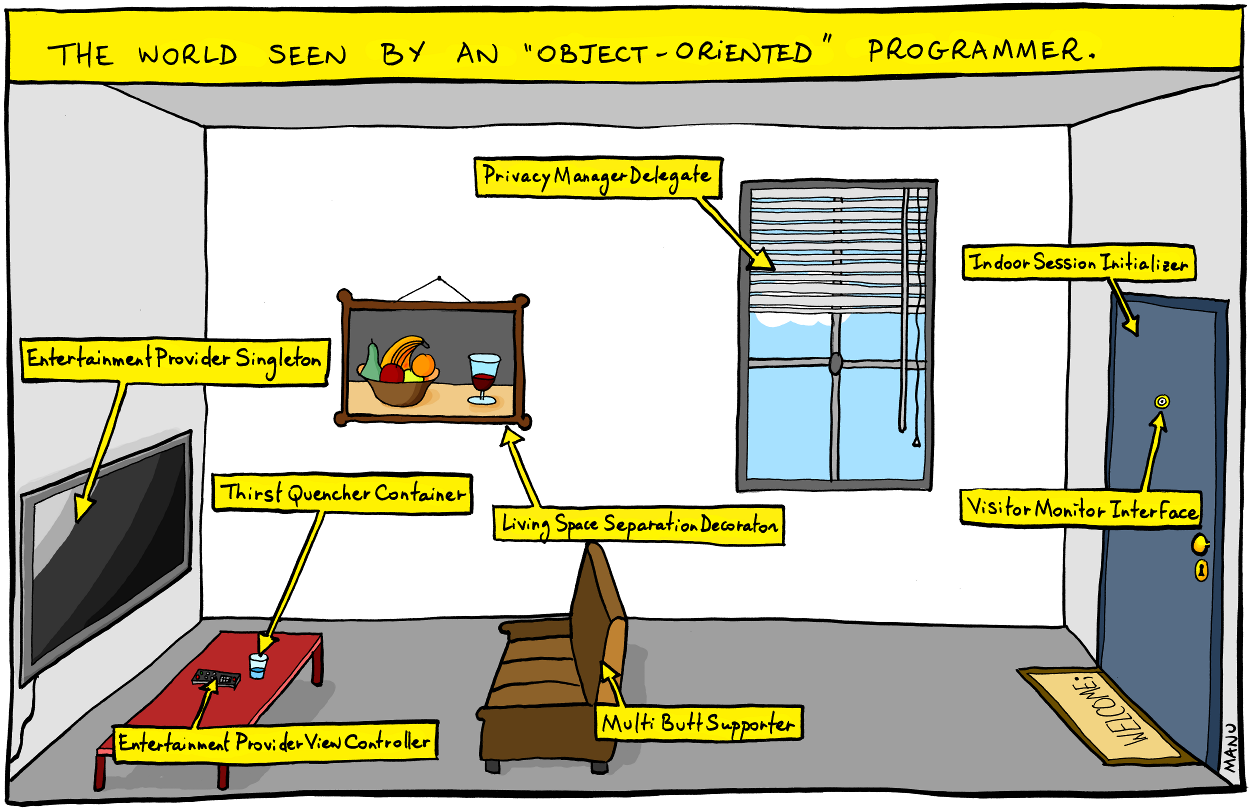
**bool success =**

**VeryImportantMethod(**

**userId,**

**username);**

# **Lego Naming**



One of the bad habits we have as developers, is making the name of variables, methods, classes worse than before. In our intent to describe things with more and more details, actually we do something bad, because the code becomes more messy and complicated because of the long and descriptive names, and it becomes unreadable.

Example 1:  
**public interface IConditionManager**

**{**

**bool CheckCondition();**

**}**

Is this really a manager? What this class manages!? No, this is just a condition, which check that something is true.

**public interface ICondition**

**{**

**bool IsTrue();**

**}**

### Example 2:

**public abstract class AbstractAnimalClass { }**

Is this class name correct? Not at all. At the beginning why this name contains the word Abstract? O yeah, maybe the keyword abstract in the definition of the class is not enough, also the thing that if we want to create a new instanton of that class, it will not compile, and it will throw a compilation error. Second thing: the word Class at the end? The same thing is valid for Object, Obj and etc as well. This is noisy code.

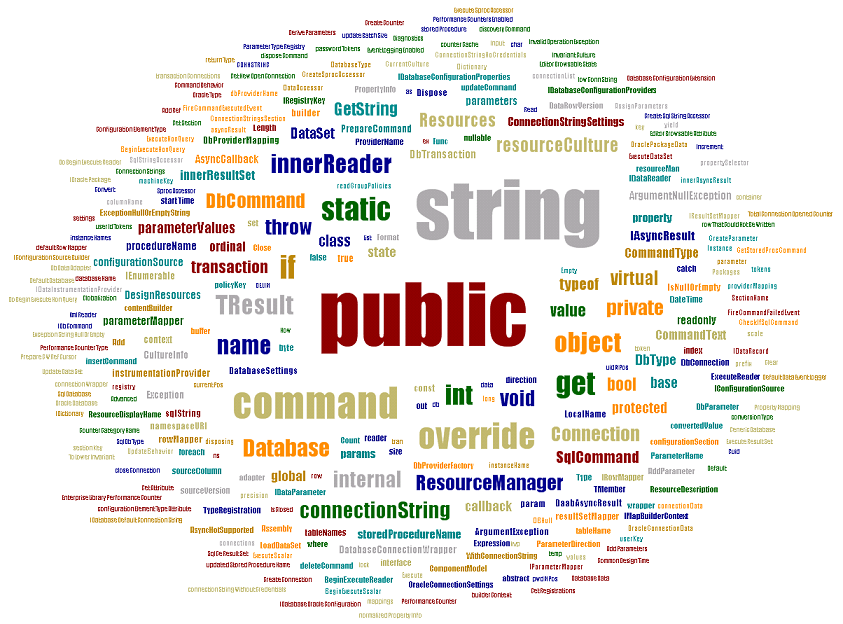
Don’t try to explain what it is, just say it. We don’t want to make everything generic, the projects are not generic, we don’t build a system which just do something generic. Every system does something very specific, and if we describe the domain of the problem in a right way, the code will be cleaner and more readable not only for us, but for all other developers in the project as well.

**Another example**: **GetIDIfUserIdExistsIfNotReturnSomethingElseFilteredByUsernameAndLocation**.

There are a ton of reasons why this name is awful. We want to get the userID right? So, based on what we are going to filter it, does not matter. Its really bad to say these details in the name of the method. First of all, you will pass these things as params and you will be able to see which things are used. Not only that, but what will happen if we decide to remove (and this is happening very often) some params because we don’t need them anymore. I will tell you what happens. We will not change the name. So, the name must say what we want to do, not how we are going to do. So, for this method for example **GetID** is a better name (not the best one). If the code is good, the check will be extracted somewhere else and it will be clear how to use this method and what it really does.

**A good name is something more than just a name, it MUST CHANGE THE THINKING OF THE PERSON WHO READ THE CODE. Good names SHOW INTENTION.**

* **ConnectionManager.CreateConnection(config); //Create a connection? Really? Just like we want to Create a connection with someone.**
* **Connection.ConnectTo(config);**



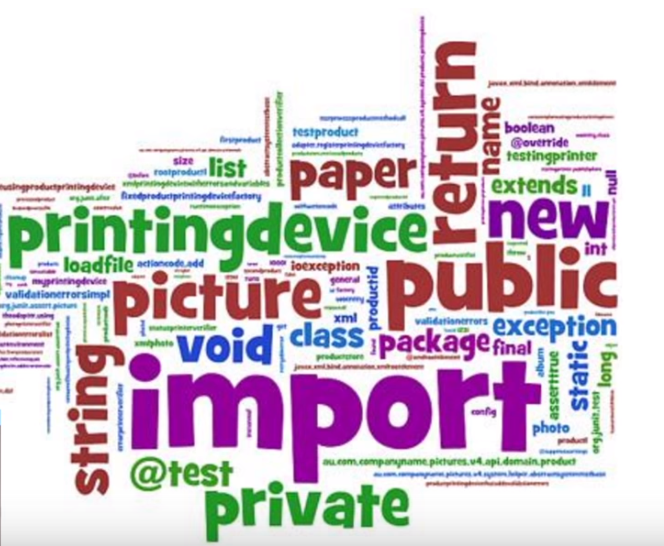
Let’s imagine that the previous image are the most common words in a project. And the more some word is used, the bigger is this word in the image.

What we can understand from it?

* Maybe the language.
* Most of the things are public.
* The dominate type of the variables is string.
* There are a lot of static methods and static classes.
* Etc...

Except that, what other we can understand? Nothing. We can’t instantly say what is the project. Is it banking software, or e-commers or something else. Everyone in the project know the project, but no one outside the project does.

Let’s see something else:



The context is clearer. We can’t say exactly what is the project, but the domain is clear: **PrintingDevice, Paper**.

# **Missing abstraction**

This is a very common problem and you can see it in places where there are many layers of abstraction in one place. Imagine the following thing:

Business Rule

Business Rule

Socket Connection

Business Rule

Database Connection

In this case we have 3 layers of abstraction in the same place. Is there a way to somehow extract the database connection? Of course. It calls FUNCTION/METHOD. We have the same problem with the comments. When you want to describe the next 50 lines of code, this mean that these lines belong to a separate method. Observe the layers of abstraction, to write clear and understandable code.

**Is this mean that we will have a hundred of classes with a single method in them?** Probably not with one method, maybe a couple, but this is the idea. I think the moment is good to mention something. You have heard for DRY (don’t repeat yourself). Let’s think about it. If two or more methods look the same, work with same type of objects and return the same result, NOT ALWAYS mean that they are the same and this code is duplicated. **ITS ALL ABOUT THE CONTEXT**. Imagine the next situation:

Two identical methods, which works with some data, in one of the cases prepare this data for some report, and in the other case return the data for the frontend. You see them, and you think “Fu\*king business, this is a bulshit” and you keep only one of the methods and you delete the other one. You are happy -> you don’t repeat yourself anymore. 😊

After a few month, someone wants to modify reports because the client doesn’t want to group by user but by location instead. Perfect! You know what to do. You deploy it, but after a while someone will come to you because the frontend is broken.

**So:** Are these methods were repeat code? **Absolutely NO!** This was a happy coincidence. These the methods were never meant to be joined because they were not the same thing. So please keep that in mind before trying to follow DRY.

**If (portfolioIdsByTraderId.get(trader.getId())**

**.containsKey(portfolio.getId()))**

**{**

**…**

**}**

Maybe we should comment this code? I don’t think this is the missing thing here. The missing thing is the abstraction. Missing communication. Let consider a different approach.

**If (trader.canView(portfolio))  
{**

**…**

**}**

Even a person who see this code for the first time, will understand what is happening -> we check that the trader have permission to view the portfolio. Details should stay hidden because they are DETAILS. Objects must not give access to their private data, instead they must provide a good API (methods) which other objects should use to communicate with this object.

# **The big three**

There are three things you should be aware of, when you think about a class:

**Ignorance**, **Apathy** и **Selfishness**.

**Ignorance:**

**- How it is implemented?**

**- I don't know...**

**Apathy -**

**- Yeah but how it is working?**

**- I don't care…**

**Selfishness -**

**- Why we are talking what this THING is offering? I need THIS, and i don't care how it gives it to me.**

If we look most egocentric about what one object wants from another, this leads to a good separation because each class will want from the code within it to be smaller as possible and to be clear and readable. If for example you want to check that a tank can shoot, first approach is to “ask” it:

tank.IsRunning == true && tank.HaveMunitions == true && tank.HaveDriver == true && tank.IsNotBroken == true && ……,

but these are really “personal questions”. That you really want to ask it, the most egocentric questions will be: tank.CanFire();