CLEAN CODE

CODE SMELLS

INTRODUCTION

DEFINITION

Clean

- free from dirt; unsoiled; unstained

Code

- a system of words, letters, figures, or symbols used to represent others, especially for the purposes of secrecy

- program instructions

CLEAN CODE

- Easy to understand
- Easy to change
- Easy to maintain

BASIC CONCEPTS

DEFINITIONS THAT ANY DEVELOPER SHOULD NOW

- OOP

  is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as *attributes;* and code, in the form of procedures, often known as *methods*

- Method

  in object-oriented programming (OOP) is a procedure associated with a message and an object. An object is mostly made up of data and behavior, which form the interface that an object presents to the outside world. Data is represented as properties of the object and behavior as methods.
  in class-based programming, methods are defined in a class, and objects are instances of a given class. One of the most important capabilities that a method provides is method overriding. The same name (e.g., area) can be used for multiple different kinds of classes. This allows the sending objects to invoke behaviors and to delegate the implementation of those behaviors to the receiving object.

- Function or procedure calls

  is a sequence of program instructions that perform a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed.
  Subprograms may be defined within programs, or separately in libraries that can be used by multiple programs. In different programming languages, a subroutine may be called a **procedure**, a **function**, a **routine**, a method, or a **subprogram**. The generic term **callable unit** is sometimes used

- Encapsulation
  methods also provide the interface that other classes use to access and modify the data properties of an object. This is known as encapsulation. Encapsulation and overriding are the two primary distinguishing features between methods and procedure calls

- Class
  is an extensible program-code-template for creating objects, providing initial values for state (member variables) and implementations of behavior (member functions or methods). In many languages, the class name is used as the name for the class (the template itself), the name for the default constructor of the class (a subroutine that creates objects), and as the type of objects generated by instantiating the class; these distinct concepts are easily conflated.

- Interface
  is often used to define an abstract type that contains no data or code but defines behaviours as method signatures

- Variable
  value that can change, depending on conditions or on information passed to the program

INHERITANCE VS POLYMORPHISM

Inheritance

- Inheritance is creating a new class using the properties of the already existing class
- Inheritance is basically implemented on clases
- To support the concept of reusability in OOP and reduces the length of code
- Inheritance may be a single inheritance, multiple inheritance, multilevel inheritance, hierarchical inheritance and hybrid inheritance

Polymorphism

- Polymorphism is basically a common interface for multiple form
- Polymorphism is basically implemented on function/methods
- Allows object to decide which form of the function to be invoked when, at compile time(overloading) as well as run time(overriding)
- Polymorphism may be a compile time polymorphism (overloading) or run-time polymorphism (overriding)

COMMENTS

Code smells: comments

Comments should be reserved for technical notes about the code and design

- Inappropriate Information
  - Don't put sensitive information on your comments
- Obsolete Comment
- Redundant Comment
- Commented-Out Code
  - Just remove it. Git keeps the history.
- Never use //TODO
  - Create your own US
- Comments can sometimes be useful:
  - When explaining **why** something is being implemented in a particular way.
  - When explaining complex algorithms (when all other methods for simplifying the algorithm have been tried and come up short).

FUNCTIONS

Code smells: functions

- Too Many Arguments
  - 3 maximun
  - No arguments is best
- Output Arguments
  - It is ok only if you don't update things implicitly.
  - If your function must change the state of something have it change the state of the object it is called on
- Flag Arguments
  - Are confunsing and declare that the function does more than one thing

```
# No
def update(data, replace=False)

# Yes
def replace(data)
def update(data)
```

- Dead Function
  - Remove all unused functions. Git keeps the history.

NAMING

Code smells: naming

- Choose Descriptive Names
  - Make sure the name is descriptive. Names in software are 90 percent of what make software readable
- Use Standard Nomenclature Where Possible
  - Ex: If you are using the Decorator Pattern, you should use the word Decorator in the names of the decoring classes
- Unambiguous Names
  - Choose names that make the working of a function or variable unambiguous
- Use Long Names for Long Scopes
- Names Should Describe Side-Effects
  - Name should describe everything that a function, variable, or class is or does

What is the worst ever variable name?
data

What is the second-worst name?
data2

What is the third-worst name ever?
data_2

CLASSES

Code smells: classes

- Encapsulation
  - Sometimes we need to make a variable or utility function protected so that it can be accessed by a test
- Classes Should Be Small
  - Usually if your classes have more than 100 lines of code you could extract part of the code
- Single Responsability Principle (SPR)
  - every module or class should have responsibility over a single part of the functionality provided by the software, and that responsibility should be entirely encapsulated by the class. All its services should be narrowly aligned with that responsibility
- Organizing for Change
  - Needs will change, therefore code will change

ERROR HANDLING

Code smells: errors

Software error - error resulting from bad code in some program involved in producing the erroneous result

- Use Exceptions Rather Than Return Codes
- Provide Context with Exceptions
- Define the normal Flow
  - Don’t return Null
  - Don’t pass Null

TESTS

Code smells: FIRST

- Fast
- Independent
  - Tests should not depend on each other
- Repeatable
  - In any environment
- Self Validating
  - Should have a boolean output
  - One assert per test
- Timely
  - Unit tests should be written just before the production code that makes them pass

Code smells: test

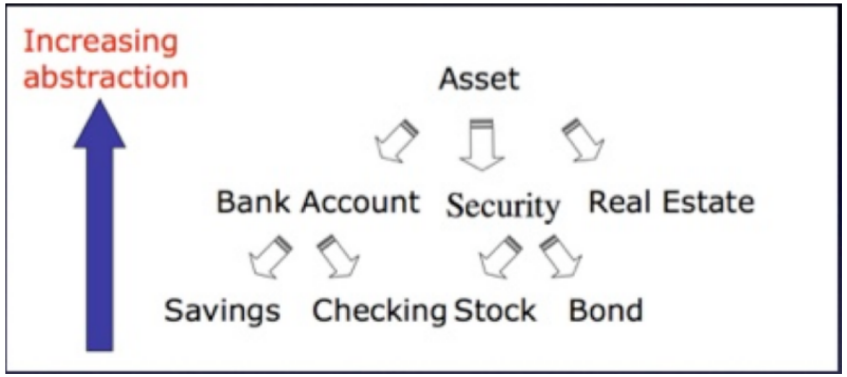
- Insufficient Tests
  - Test everything that could break
- Use a Coverage Tool
- Don't Skip Trivial Tests
- An Ignored Tests is a Question about an Ambiguity
- Test Boundary Conditions
- Exhaustively Test Near Bugs
- Patterns of Failure Are Revealing

GENERAL

Code smells: general

- Multiple Languages in One Source File
  - Don't mix different languages in one only file (depends on the language…)
- Obvious Behavior Is Unimplemented
  - Names matter… implement what you would expect if you were not the main developer
- Incorrect Behavior at the Boundaries
  - Happy path is not the only thing that matters
  - Part of looking for boundaries is looking for special conditions that can cause the test to fail
- Overridden Safeties
  - Your safeties are teh warnings, errors, exception handling – they could catch you in the future
  - Sometimes we will have to override safeties but it could be risky, try to send at least a warning

Code smells: general



- Duplication
  - Follow the Dry principle (Don't Repeat Yourself).
- Code at Wrong Level of Abstraction
  - Heritage, polimorphy and encapsulament
- Base Classes Depending on Their Derivatives
  - When we see base classses mentioning the names of their derivatives, we suspect a problem
  - In general, base clases should know nothing about their derivatives
- Too Much Information
  - Limit what you expose. Well defined modules have very small interfaces that allow you to do a lot with a Little.
  - Concentrate on keeping interfaces very tight and very small. Help keep coupling low by limiting information
- Dead Code
  - Don't catch exceptions that will never be thrown or else blocks that will never be reached.
- Vertical Separation
  - For example: All functions declaration before their usage. Or, declare the function right after its first usage.
- Inconsistency
  - Things that have the same meaning should have consistent conventions.

Code smells: general

- Clutter
  - Variables that aren't used, functions that are never called, comments that add no information… should be removed
- Artificial coupling
  - Things that don't depend upon each other should not be artificially couple
- Feature Envy
  - The scope of a class should not include variables or functions from others
- Selector Arguments
  - Booleans bad as parameters
- Misplaced Responsibility
  - Code should be places where a reader would naturally expect it to be
- Encapsulate conditionals
- Avoid negative Conditionals
- Functions should do One thing

Code smells: general

- Use Explanatory Variables
- Functions Names should Say What They do
- Understand the Algorithm
  - Take time to understand the code
- Make Logical Dependencies Physical
  - If one module depends upon another… that dependency should be physical, not just local. Ask explicity for the module
- Prefer polymorphism* to if/Else or Switch/Case
  - *Only if you are using OO
- Follow Standard Conventions
  - Where to declare instance variables; how to name classes, methods, and variables; where to put braces… our code should provides examples

Code smells: general

- Replace Magic Numbers with Named Constants
- Be Precise
- Structure over Convention
- Don’t be arbitrary
  - Have a reason for the way you structure your code, and make sure that reason is communicated by the structure of the code
- Keep configurable data at High Levels
- Avoid transitive nativation
- A.getB().getC().doSomething()

LINKS OF INTEREST

Cheat Sheet
https://www.planetgeek.ch/wp-content/uploads/2013/06/Clean-Code-V2.2.pdf

Code Smells
https://sourcemaking.com/refactoring/smells

https://es.slideshare.net/KuySengChhoeun/chapter17-of-clean-code

Examples:
https://es.slideshare.net/annuvinayak/code-smells-and-its-type-with-example

https://github.com/ryanmcdermott/clean-code-javascript/blob/master/README.md