ICTA
ideas actioned

Quality Coder
2017v1.0

QUALITY
CODER
2017v1.0

TIQRI iTelaSoft™ WSO2 virtusa® Accelerating Business Outcomes

99X Technology Sysco LABS ZONE24x7

Quality Coder 2017v1.0

A curriculum put together with the expertise of

To walk you through the journey towards being a Quality Coder

Acknowledgements

SPECIAL THANKS

The speakers, Amila De Silva from **99X Technology**, Chamil Jeewantha from **Zone24x7**, Chathura Dilan from **WSO2**, Lasitha Dahanayake from **iTelaSoft**, Niruka Ruhunage from **Tiqri**, Ruwan Wijesinghe from Virtusa and Subhash Vithanapathirana from **Sysco LABS**, for their continued support and utmost effort for developing the manual and our heartiest gratitude to the mentors of Quality Coder workshop, Asiri Warnakulasuriya from **SyscoLabs**, Charith Sooriyaarachchi from **99x Technology**, Damitha Wimalasooriya from **Tiqri**, Isuru Senadheera from **99X Technology**, Maduranga Siriwardena, Pulasthi Mahawithana, Chanuka Dissanayake from **WSO2** for their support in making this workshop a success.

OUR SINCERE GRATITUDE

Chathuranga Wijeratna, Nuwan Weerasinghe, Sankalpa Gamwarige from **Zone24x7**, Chinthaka Dissanayake from **Tiqri**, Hasith Yaggahavita from **99X Technology**, Indaka Raigama from **iTelaSoft**, Shevan Goonetilleke and Yasith Tharindu from **WSO2** for their continued support.

Contents

Quality Code

The ICT industry in Sri Lanka is facing a growing issue where developers produce software code that is of low quality and does not adhere to industry standards. This directly affects the efficiency and standard of the software products built in the country. As a software exporting nation, it is crucial that Sri Lanka upholds the current reputation of High Quality Code. Industry claims that the software quality is upheld by the direct efforts of the Senior Level Developers who must fix the mundane errors of the developers below them.

The usual solution taken up is to let the developer learn while on the job and gradually get them to code with higher quality; this is a time-consuming process. To overcome this issue, 'Quality Code' was drafted as a hand book with best practices. Quality Code guides the developers with the best practices and knowledge extracted from a blend of the industry experts in the IT industry to produce high quality software products.

1.0 Why Quality Code…

What is Quality Code?

Quality is something which you can't define exactly as it depends on the background you're coming from. It is "the standard of something as measured against other things of a similar kind; the degree of excellence of something". When it comes to software, quality of the code highly matters.

But what is Quality Code? "Quality code is something subjective. It varies from a person to another depending on their knowledge, expertise level, taste on quality and so on. Because, the code which we think as the best code might not be the best for someone else", says Subash.

"Clean code is simple and direct.
Clean code reads like well-written prose." [1]

Quality Code/ Clean code is like a good book that you have read. When you read word by word through sentences and paragraphs of that book, the story gets replaced by images in your mind. The story becomes that much clean & clear that you're likely watching a movie of that book. This is what happens when you read something well written. With what Grady Booch says, your code also should be the same. The person who reads your code should not have to put an additional effort to understand the purpose of your code. The reader should be able to understand your code by simply looking at it.

"Clean code always looks like it was written by someone who cares." [2]

When you read a clean code, you'll understand that the original author of that code has genuinely cared for what he's been crafting. It will be evident that the developer has focused on details and taken the effort to keep it simple & orderly.

"If you want your code to be easy to write, make it easy to read." [3]

Think of a code that you've written, maybe a simple class. At the very beginning of that code, you comment on top of it as author with your name. This means the programmer is the author of that code. Authors have a special responsibility to communicate with their readers. The most important thing as a programmer for you to understand is that when you write a line of code, remembering that you're an author, and that you are going to have many readers to it.

Features of Quality Code

In writing quality code there are some features identified by experts that guide authors to write clean code.

Readability

Readability is one of the main features in quality code. It means how easy it is for someone to understand the purpose of your code.

What makes code readable?

- Good naming of variables, functions and classes
- Good comments
- Consistent coding standard
- KISS (Keep It Simple, Stupid) principle.

Extensibility

Extensibility implies how easy it is for you to add a new functionality or new feature to your existing code base. This is another key component in writing quality code.

How to write extensible code,

- Conscious usage of OOP principles, Design patterns.
- SOLID principles.
- DRY (Don't Repeat Yourself) principle.

Testability

Testability is another key component that affects the quality of the code and inspects how much of your code can be automatically tested.

What are the effective approaches for testability?

- Unit Testing with high code coverage.
- Test Driven Development (TDD).

Golden rule of Quality Code is, it is easy to understand & easy to change.

Why Code Quality is Important?

What's the problem with a software that works? A working software might imply that it has met the customer requirements which shows the external quality of the software. But Code Quality consists of both external & the internal quality of the software.

Code quality is the most important fact of software as low code quality affects the maintenance, modifications or adjustments of the software, which is time consuming and leads to immense financial losses.

How to identify Good Code from Bad Code?

Ex: -

Given below is a **Spaghetti** code. If the orange areas are the pieces of code that needs to be changed or modified, see the number of places that you have got to change. It makes it harder to traverse through the code and may lead to you to step on irrelevant segments of code and do unwanted changes that might introduce bugs.

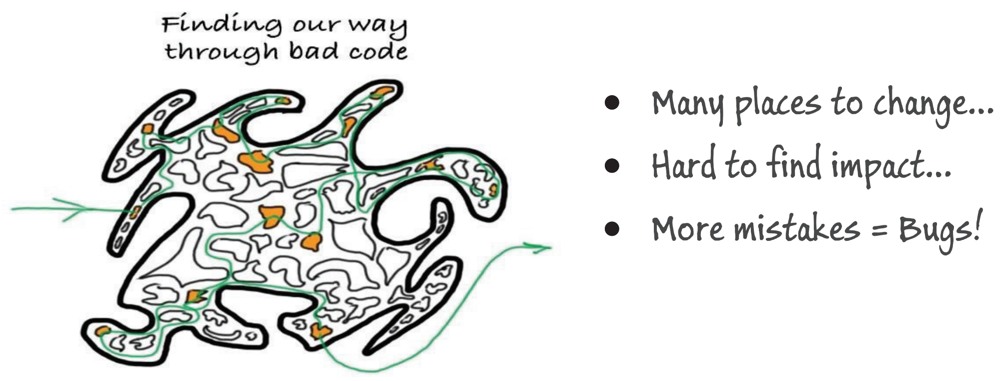


Image Source -https://www.slideshare.net/MarcoBeelen/designing-testable-clean-code-part-of-tdd-serie-of-the-haarlem-software-developer-meetup

Given below is a more **organized Quality Code**. When compared to a spaghetti code only few places are there to be changed. It is easy to find the impacted areas and to navigate from one to another which creates less bugs than before.

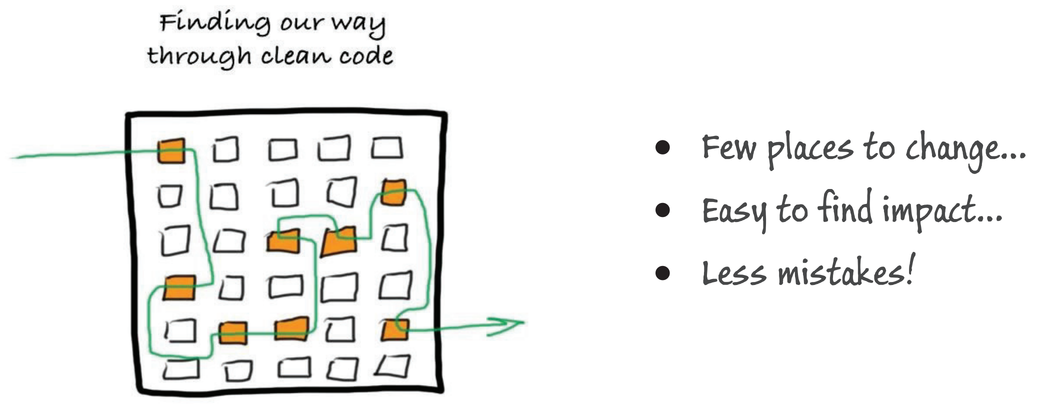


Image Source -https://www.slideshare.net/MarcoBeelen/designing-testable-clean-code-part-of-tdd-serie-of-the-haarlem-software-developer-meetup

Importance of Quality Code

- Code is never written just once and then forgotten
- We read code more than we write!
- Cost of bad code is high
- Quality is a must to be agile

"Programs must be written *for people* to read,
and only incidentally for machines to execute."

Journey towards Quality…

- Quality should start early!
- Learn by reading good code (…and bad code!)
- Code reviews — give feedback, get feedback
- Know your tools — IDE, Source Control, Frameworks etc.
- Passion — Love, care and be proud of your code.
- Practice, practice and practice!!!

Rule of thumb:

"Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live!"

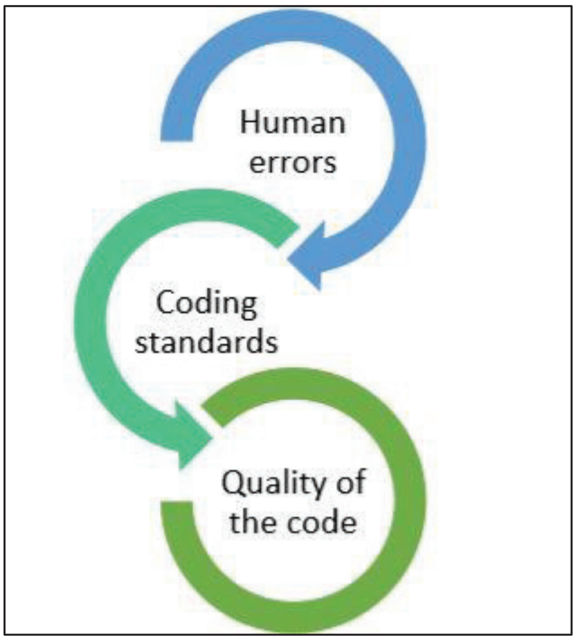
Image Source - https://tipyang.wordpress.com/2013/08/08/ra-7-computer-ethicsnetiquette/

https://www.youtube.com/watch?v=645ltyDm6LU

2.0 Source Control Tools & Peer Review

The main intention of this chapter is to go through source control mechanisms and to learn how we can conduct peer reviews. “Code review is the systematic examination of computer source code. It is intended to find mistakes overlooked in the initial development phase, improving the overall quality of software”

Why Review?



Someone might not see a way of writing a piece of code in the best possible way and another person might see a better way of writing the exact same code. An error free code of one person might be full of errors to another. It is a must to identify these human errors to develop a quality code.

The way you write the code should be readable to everyone. If the code is written without a standard, there’s a high possibility of getting complaints from the readers. Therefore, coding standards is another key component to be considered.

It is possible to produce code that works, even without coding standards and human errors. But we should also look at the code quality. That’s why code reviews are needed.

Tools for Reviewing

To improve the overall quality of the software, code reviews are a must. There are various methods to review code. In peer reviews one or more colleagues examine the code produced by an individual to evaluate its technical content and quality. Given below are some of the commonly used tools used in code review.

CODACY

kallithea

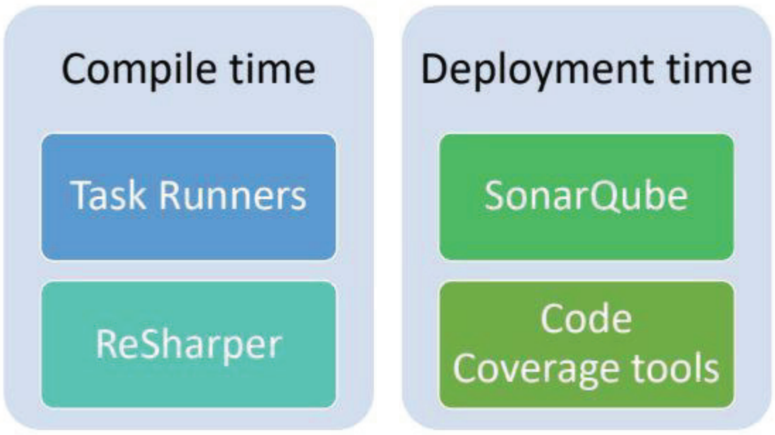
GitLab

Crucible

RhodeCode

Automated Reviews

Certain reviews can be done by a machine itself. There are two types of reviews that act on compile time & deployment time



Ex: - ReSharper (Commonly used in .Net), enforces the author to stick to a specific coding convention and also helps get an idea about code quality. Task Runners analyze the code etc. at compile time.

SonarQube is a review tool used in the deployment phase and runs unit tests, checks it against conventions etc. at the deployment time. Code coverage tools test whether you have enough test coverage for your code at the deployment phase. Authors can also run the code coverage tools in the IDE itself.

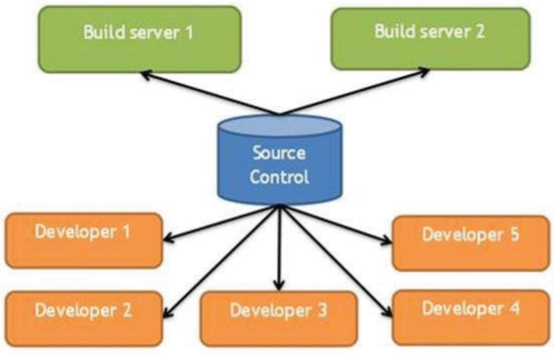
Source Control Tools

Source control is management of software code across many developers at the same time.

If there's a development team to develop a specific product, there should be a system to manage it. That's why the source control systems are being used. Given below is an over view of source control.

https://www.youtube.com/watch?v=T10xPJIkBJk

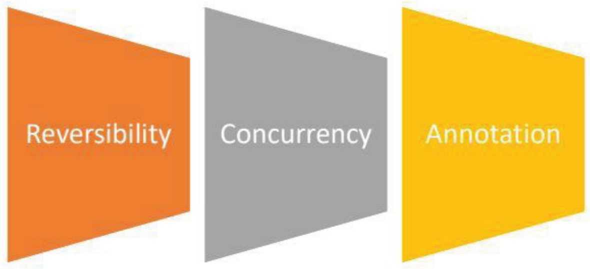
Assume that there's a source control for a product. Each developer uses the source control to work on their own machine. There are some build servers configured to pull the data from the source control. All the developers connect to the server where source control is hosted, fetches the code into their own machine and works on it.



Once the developer is satisfied, he commits the changes to the source control system. Once a developer commits (makes changes to the code and uploads it), if another developer pulls the code from the source control system, the second developer gets updated with all the changes that were committed by the first developer.

The idea is that all the developers should try to maintain the source control server at an 'ok' level. Developers should not commit if there's a break of a functionality. If the developers are using continuous deployment it is possible to get the continuous deployment servers to monitor the source control systems so that it will just take the data and publish it.

Why Source Control?



Source control systems also support maintaining history of code that allows **reversibility**. If a developer wants to reverse the code that he developed, that can be always reversed with the use of source control. It maintains the history of development.

Concurrency means that several developers can work at the same time. Source control helps multiple developers collaborate on the same code.

Annotation helps colleagues in the same team annotate each other's code. If the team has the required tools they can comment and mark on the code for improvement.

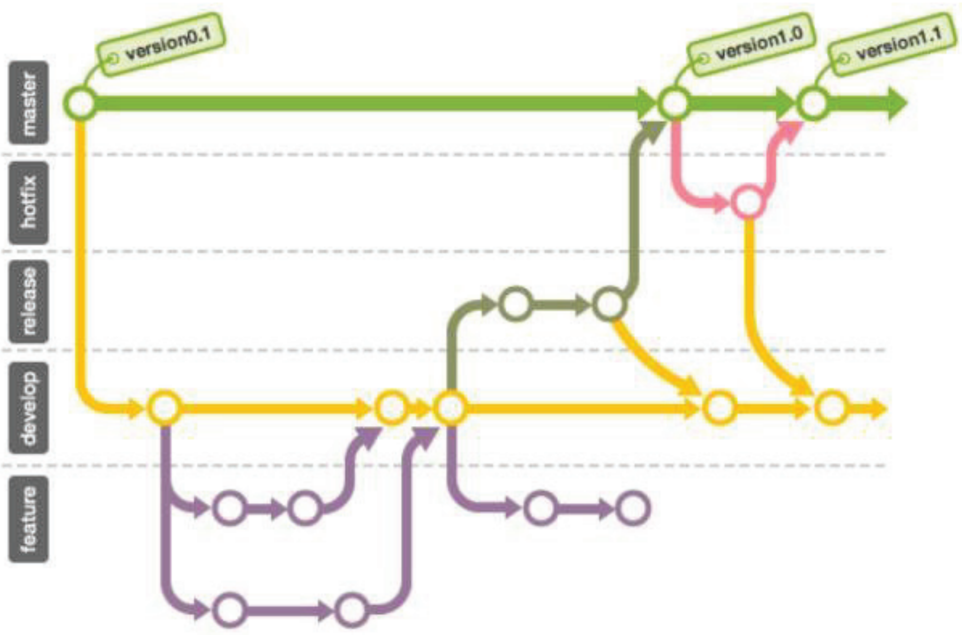
Source Control Tools.

Given below are some of the mostly used source control tools.

Bitbucket GitHub SUBVERSION Microsoft Visual Studio Team Services

Branching

With source control, you also get this concept called **branching**. Assume that there is a sprint and that there are three long running tasks to complete in the sprint that hardly overlap with each other. There's a panel called branching where the developer can take a copy of the source control & create a different branch. Since it's a copy of the branch, whatever changes that are made in the branch won't affect the source control until the developer merges the branch with the **master**.



Master is like the production. It's clean with not many changes on it. **Develop** branch is the branch that developers work on. When two people are working on two different **features** they create their own branches and work on them. Then you do interrelated commits, and once you develop, you merge it with the development branch. Once someone decides that it is time to do a release, you create a **release** branch (Release Candidate). That's what the QA tests. Because development branch is a branch where to do experiments as well. But once you finish it, you create a stable branch for the release.

Once the QA decides that the release is bug free and in good quality, then it is merged into the master branch. But you also have to merge it with the development branch because if you have done any bug fixes those bugs are there in the development branch as well. When you encounter a production bug you create a branch in the **hotfix**. Once the product is delivered and a bug is discovered afterwards, the main intention is to do a minimum change and fix the bug.

Continuous Integration

Continuous Integration (CI) is a development practice that requires developers to **integrate** code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.

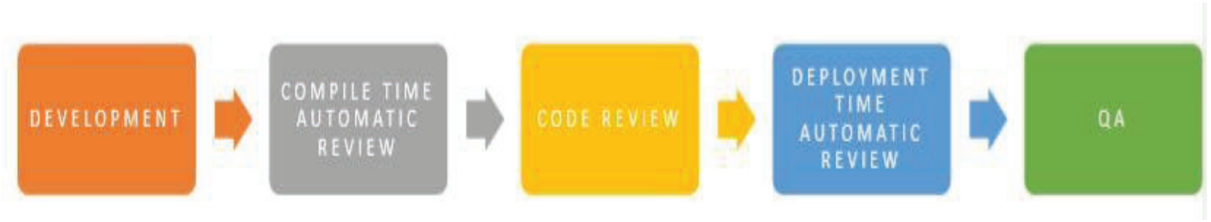
Once you complete branching, you hook your master branch into the built servers in here. Whenever you do a change on your master branch, the continuous integration servers will take that and automatically deploy it on the production environment.

https://www.youtube.com/watch?v=xSv_m3KhUO8

Review as Part of the Process

Review should be a part of the product. Given below shows how a typical review should look like.

Once you do the development you review the code at compile time using tools. Once you complete your compile time review you do the peer review. Next you do the deployment time review with the tools (ex: - SonarQube). Only after completing these will you call QA. Developers should take a lot of responsibility on testing as it directly affects the quality of the software product.


DEVELOPMENT
COMPILE TIME AUTOMATIC REVIEW
CODE REVIEW
DEPLOYMENT TIME AUTOMATIC REVIEW
QA

Lab Exercise

Source control exercise

1. Sign in to GITHUB
2. Create a new repository
3. Download GIT
4. echo "# test" >> README.md
5. git init
6. git add README.md
7. git commit -m "first commit"
8. git remote add origin https://github.com/amiladesilva/test.git
9. git push -u origin master
10. Download GIT Extensions
11. Open the same repository
12. Push it to a different branch

GITHUB for beginners: https://www.youtube.com/watch?v=0fKg7e37bQE

3.0 Static Code Analysis

In the early stages of programming there were no proper rules or specific formats to follow. Most projects had small functions to code and debug with few team members. When programmes become larger as to the enterprise level, the number of development teams grew and the complexity of the programmes grew as well. With the high complexity, people started identifying more errors and patterns that have a high probability of leading to bugs in the programmes.

What is Static Analysis

With a complete analysis of the code, developers realized that there are certain patterns in some codes that have a high probability of leading to errors. These patterns were used to make a set of rules to analyze the code before going in to the compiler.

Static analysis, also called static code analysis is a method of computer program debugging that is done by examining the code without executing the program. Static analysis does not validate the logic of the code but validates whether there's a certain pattern in the code that can be a potential issue leading to a bug.

Image Source - https://libidothanato.wordpress.com/2017/11/14/libido-thanato-kapitel4seite6-libido/

How Static Analysis can help Software Quality

- There are two ways of inspecting software quality
  - Examine the behavior during the run-time (Dynamic analysis), which is done by the computer using the compiler.
  - Inspect source code / Code reviews (Static analysis), which is done by looking at the code and going through it before sending it to the compiler.

- Inspecting and analyzing the source code of the program before it is tested lowers the cost of finding and fixing bugs in software in the early stage of the development cycle.

Static analysis helps to read, review and fix the potential bugs that saves time and effort of developers.

Static Analysis Tools

- Developers are human beings, and everyone make mistakes. So, it is extremely hard to guarantee that things would be done correctly in the first time itself.
- Employing static code analysis tools is one of the best practices in software development.

- Some of the Static analysis tools available are,
  - .NET
    - CodeIt.Right, FxCop, StyleCop etc.
  - Java
    - PMD, CheckStyle, FindBug etc.
  - JavaScript
    - JSHint, JSLint etc.

Static Analysis Platforms

- Static analysis platforms come with server components
- Static analysis platforms support multiple programming languages and produce various matrices for analysis and even maintains historical data.
  - SonarQube, Moose, Kiuwan are some of the examples
- Selecting the right tool
  - There are language specific static analysis tools that come as IDE plug-ins; They are helpful for the purpose of catching for issues while coding.
  - Static analysis platforms support multiple languages, handle multiple projects and run independently without a development environment. They are even suitable for organization level static code analysis and for providing various views and dashboards.

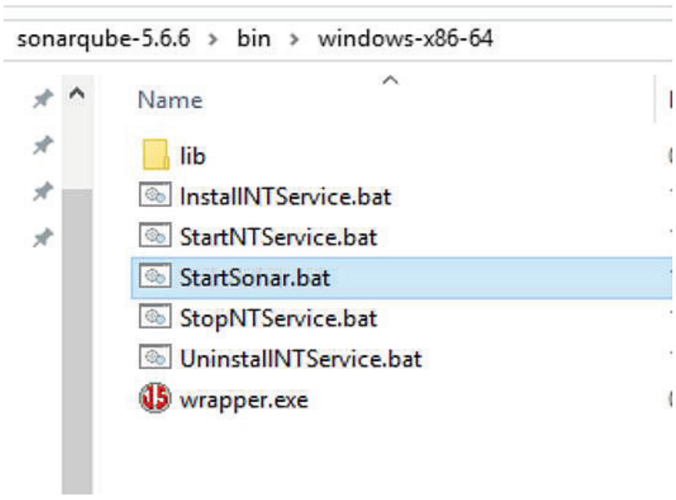
SonarQube

- Supports 20+ programming languages.
- More than 40 open-source and commercial plugins.
- Supports integration with famous build tools such as Maven, Ant, MSBuild, Jenkins and Gradle
- Covers the 7 axes of code quality
  - Architecture & Design
  - Comments
  - Coding rules
  - Potential Bugs
  - Duplication
  - Unit Test
  - Complexity

SonarQube Server and Runner setup

- Download and install Java JDK if it's not available (Java 8).
- Download SonarQube from https://www.sonarqube.org/downloads/ and unzip to a desire location.
    - Make sure port 9000 is available for listening
- Download SonarQube runner from,
    - https://docs.sonarqube.org/display/SCAN/Analyzing+with+SonarQube +Scanner and unzip to a desire location.

Running SonarQube Server

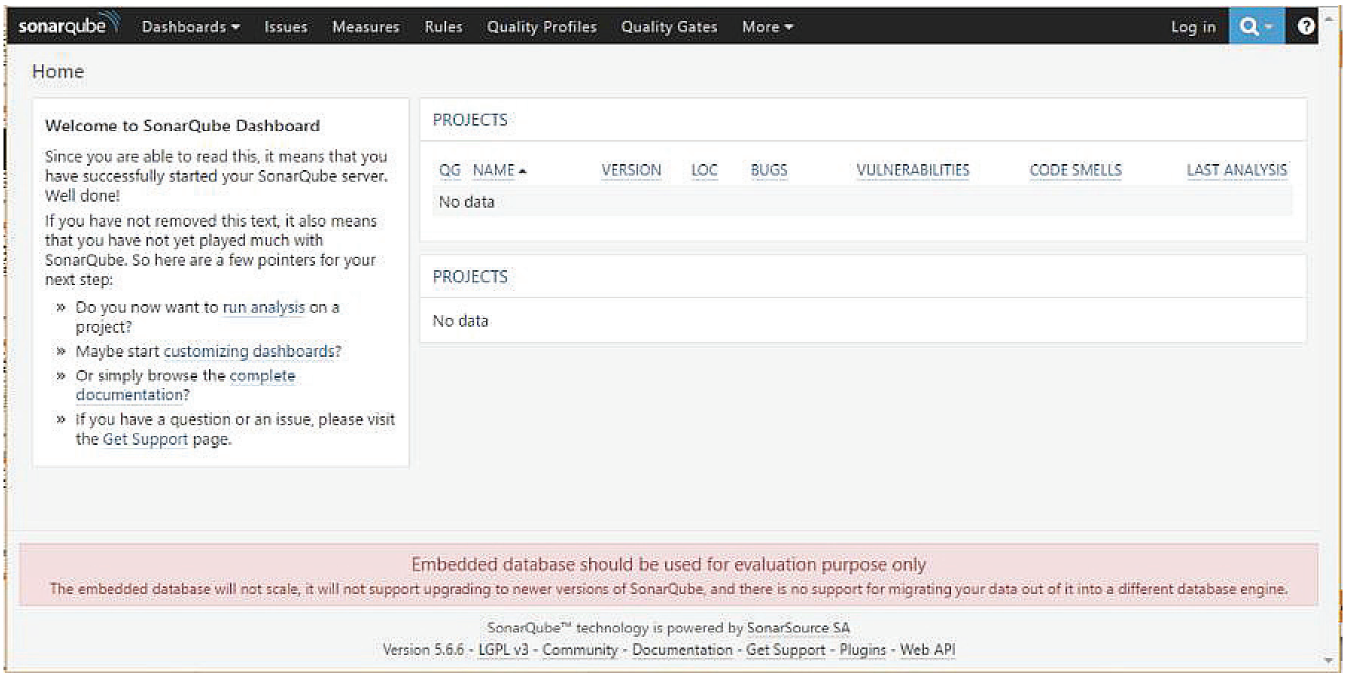


Start the SonarQube server using the startup script available in;

SONAR_HOME/bin/<Your Platform Folder>/<sonar.sh or StartSonar.bat>

Access SonarQube Server

After SonarQube starts, access the web interface of the SonarQube at http://localhost:9000.



Welcome to SonarQube Dashboard

Since you are able to read this, it means that you have successfully started your SonarQube server. Well done!

If you have not removed this text, it also means that you have not yet played much with SonarQube. So here are a few pointers for your next step:

- » Do you now want to run analysis on a project?
- » Maybe start customizing dashboards?
- » Or simply browse the complete documentation?
- » If you have a question or an issue, please visit the Get Support page.

PROJECTS

QG	NAME ▴	VERSION	LOC	BUGS	VULNERABILITIES	CODE SMELLS	LAST ANALYSIS
No data							

PROJECTS

No data

Embedded database should be used for evaluation purpose only

The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.

SonarQube™ technology is powered by SonarSource SA

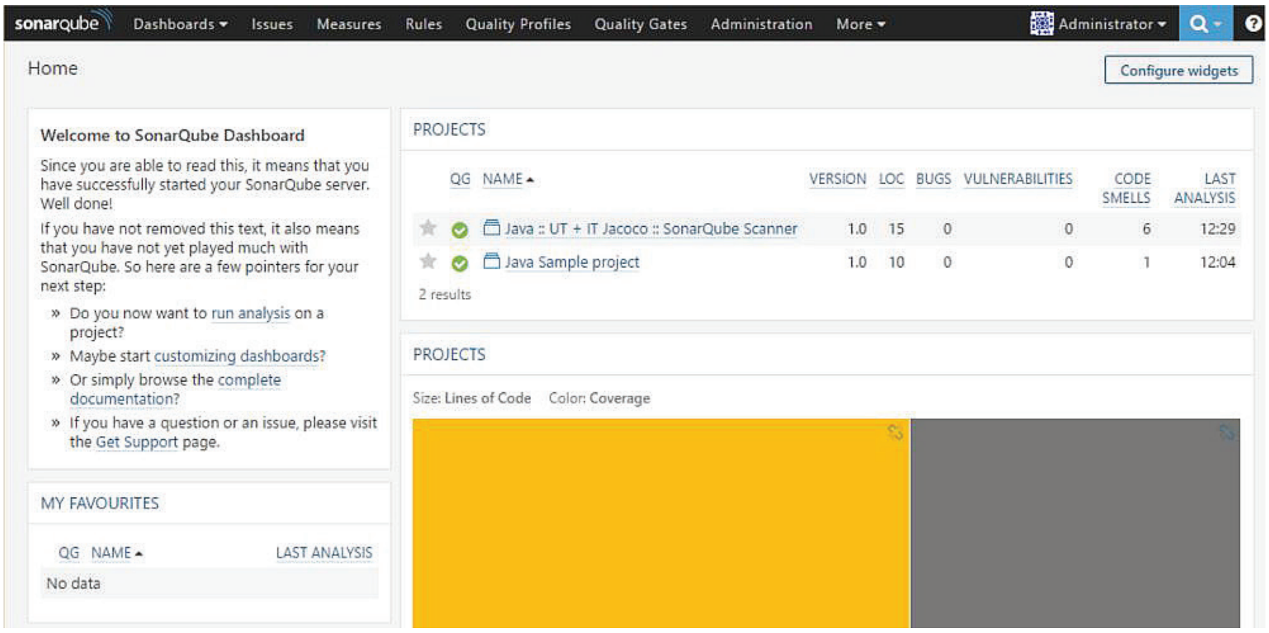
Version 5.6.6 - LGPL v3 - Community - Documentation - Get Support - Plugins - Web API

SonarQube Runner (Standalone)

- Set the environment variable SONAR_RUNNER_HOME and set the value to the path you extracted SonarQube runner zip file (e.g C:\sonar-scanner-3.0.1.733-windows).
- Append Sonar runner bin folder to the path environment variable.
- Update Sonar runner setting in SONAR_RUNNER_HOME/conf/sonar-scanner.properties file (specify correct URL, e.g http://localhost:9000).
- In your project home folder, create a file called 'sonar-project.properties' and enter the following lines as content (change values as you needed).
  - sonar.projectKey=mysample:project
  - sonar.projectName=Java Sample project
  - sonar.projectVersion=1.0
  - sonar.sources=src\\main\\java
- Run 'sonar-scanner' command from your project home (make sure the project has no compile errors)

View SonarQube Analysis Results

- Access SonarQube server at http://localhost:9000



Welcome to SonarQube Dashboard

Since you are able to read this, it means that you have successfully started your SonarQube server. Well done!

If you have not removed this text, it also means that you have not yet played much with SonarQube. So here are a few pointers for your next step:

» Do you now want to run analysis on a project?
» Maybe start customizing dashboards?
» Or simply browse the complete documentation?
» If you have a question or an issue, please visit the Get Support page.

PROJECTS

QG	NAME	VERSION	LOC	BUGS	VULNERABILITIES	CODE SMELLS	LAST ANALYSIS
✓	Java :: UT + IT Jacoco :: SonarQube Scanner	1.0	15	0	0	6	12:29
✓	Java Sample project	1.0	10	0	0	1	12:04

2 results

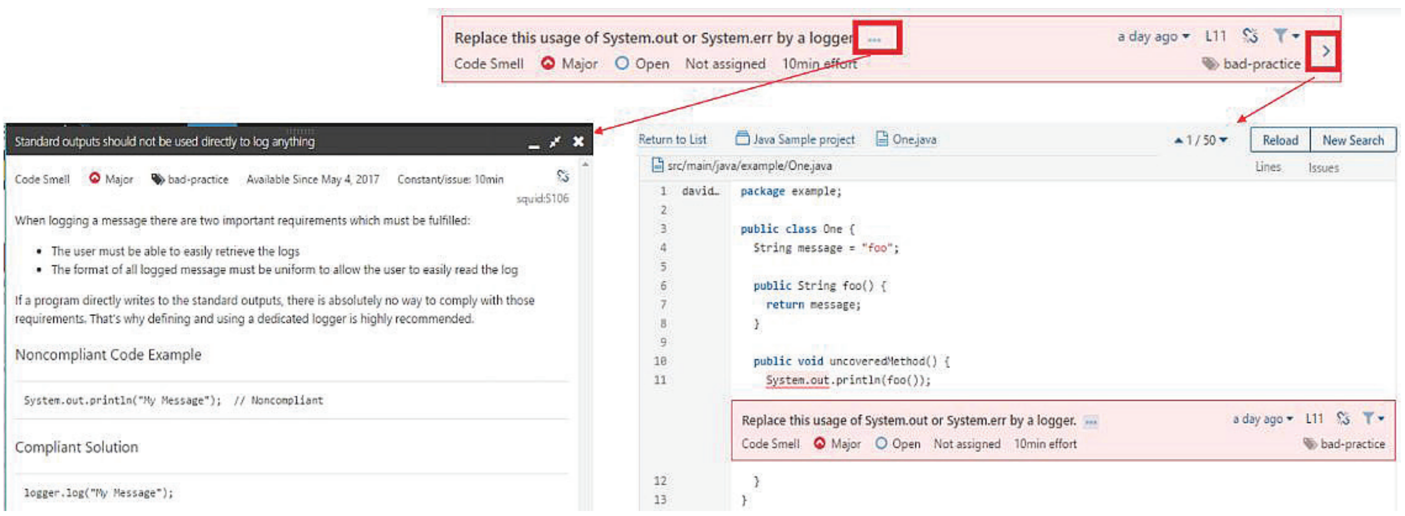
PROJECTS

Size: Lines of Code Color: Coverage

MY FAVOURITES

QG	NAME	LAST ANALYSIS
No data		

- Issue details



Replace this usage of System.out or System.err by a logger ...
Code Smell Major Open Not assigned 10min effort
a day ago L11 bad-practice

Standard outputs should not be used directly to log anything

Code Smell Major bad-practice Available Since May 4, 2017 Constant/issue: 10min
squid:S106

When logging a message there are two important requirements which must be fulfilled:

- The user must be able to easily retrieve the logs
- The format of all logged message must be uniform to allow the user to easily read the log

If a program directly writes to the standard outputs, there is absolutely no way to comply with those requirements. That's why defining and using a dedicated logger is highly recommended.

Noncompliant Code Example

```
System.out.println("My Message");  // Noncompliant
```

Compliant Solution

```
logger.log("My Message");
```

Return to List | Java Sample project | One.java | 1 / 50 | Reload | New Search

src/main/java/example/One.java

```
david…  package example;

        public class One {
          String message = "foo";

          public String foo() {
            return message;
          }

          public void uncoveredMethod() {
            System.out.println(foo());
```

Replace this usage of System.out or System.err by a logger. ...
Code Smell Major Open Not assigned 10min effort
a day ago L11 bad-practice

```
          }
        }
```

Keep in mind…

- Clearing all static analysis issues doesn't mean your code is in good quality.
- Static analysis tools only catch mistake that have a common pattern.
- Therefore, use wisely for your benefit.

SonarQube for beginners: https://www.youtube.com/watch?v=xLO8Q_F3jIg

4.0 Object Oriented Programming

Objects and Classes

If you are asked to identify the objects in a picture, what things would you identify?

Chairs, Table, TV…

Why did you select the TV as an object? Because we humans are naturally tempted to think of those things as objects.

Objects in the real world are much similar to the objects that are built in object-oriented programming. But it isn't the exact same as it is said.

Image Source - http://tjoneswrites.com/wp-content/uploads/2017/10/tv-stand-for-sale-ghana-wall-hung-tv-stand-costco-tv-stand-black-friday-wall-mounted-tv-vs-stand-wall-mounted-tv-floor-stand-attaching-plasma-tv-to-wall.jpg

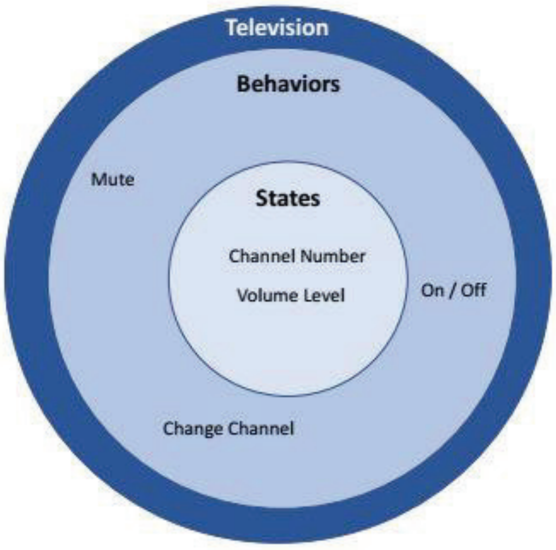
Object

Once you identify the object, then you should understand the behaviors and the states of the object. States are the attributes of an object while the behavior is what you can function.

Ex: - Television

Behaviors are the functions that you can perform in the object 'Television'. Such as change the channel or mute the volume…etc.

States are the attributes of the Television like channel number, volume level…etc.

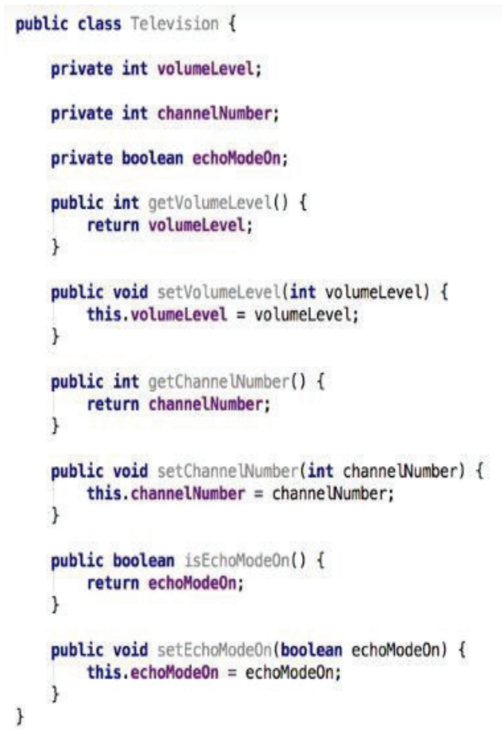


Classes and Objects

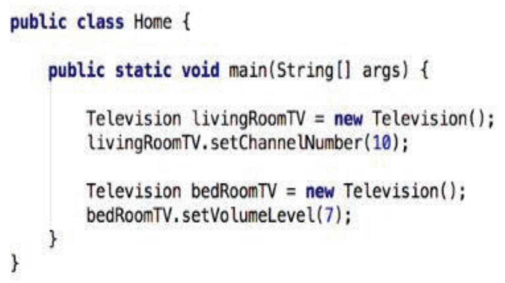
Classes are some kind of a mold where the objects are created out of it. You are creating objects out of classes.

Class

Classes have all the attributes and behaviors. When you’re creating an object you are actually assigning values to those attributes.



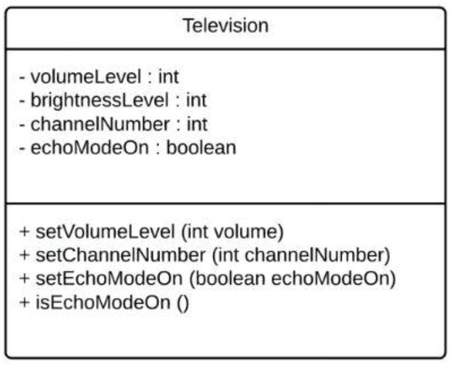
```java
public class Television {

    private int volumeLevel;

    private int channelNumber;

    private boolean echoModeOn;

    public int getVolumeLevel() {
        return volumeLevel;
    }

    public void setVolumeLevel(int volumeLevel) {
        this.volumeLevel = volumeLevel;
    }

    public int getChannelNumber() {
        return channelNumber;
    }

    public void setChannelNumber(int channelNumber) {
        this.channelNumber = channelNumber;
    }

    public boolean isEchoModeOn() {
        return echoModeOn;
    }

    public void setEchoModeOn(boolean echoModeOn) {
        this.echoModeOn = echoModeOn;
    }
}
```



```java
public class Home {

    public static void main(String[] args) {

        Television livingRoomTV = new Television();
        livingRoomTV.setChannelNumber(10);

        Television bedRoomTV = new Television();
        bedRoomTV.setVolumeLevel(7);
    }
}
```

Class

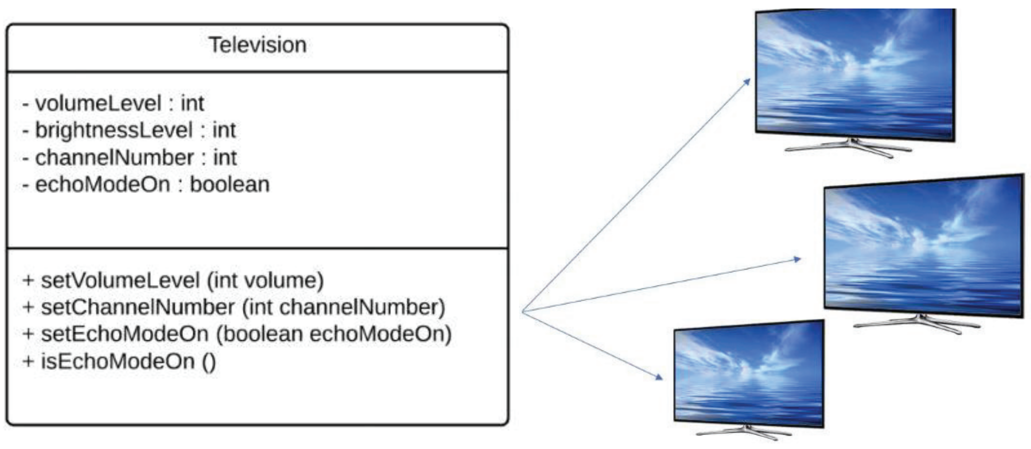
- Extensible program-code-template for creating objects
- The blueprint from which individual objects are created



Object

Given below is a template. From that template you can create objects. It has attributes as well as behaviors. At the end, you’re creating objects out of classes.

- Instance of a class



Television

- Can use a remote control or button to control it.
- You do not open its case to use it.
- If you want to extend its functionalities like to watch a movie, you can simply connect it with a DVD player
- It is a complete product when you buy it. How to use it and external requirements are very well documented.
- It will not crash while you use it.

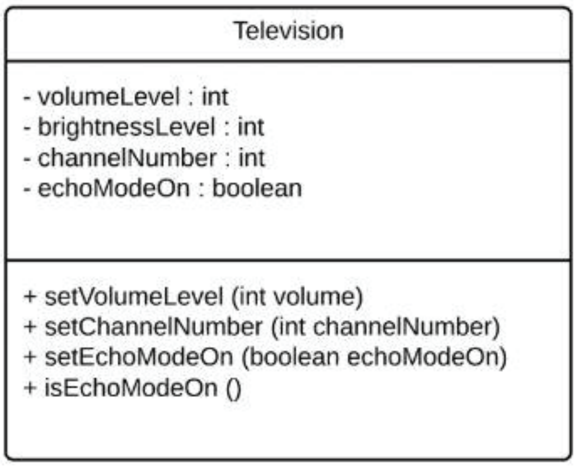
Image Source - https://oimparcial.com.br/noticias/2017/12/vai-precisar-comprar-conversor-digital-saiba-quanto-custa-e-como-funciona/

You have to think of your objects and classes with similar analogy. When you're developing a product, first you are creating the class and then the objects. In doing that, first you need to identify the interfaces - how other things connect with your object. You provide what the customer requires. Therefore, you don't want others to modify it. It is like a solid thing.

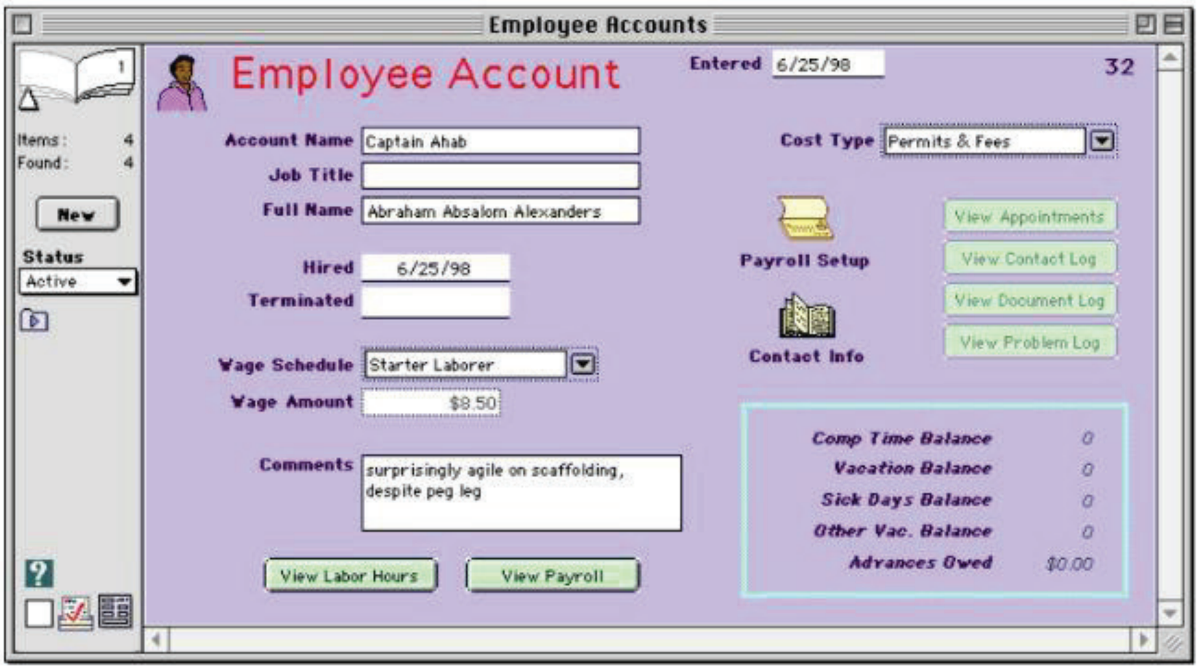
Ex: - In a Television, manufacturers don't expect their customers to remove the casings and do modifications. If someone wants to extend its functionalities they can do it through the interfaces that are provided. Like extending the functionality of a Television via its USB port or HDMI port available...

Class

- Represents a clear concept, regardless of wherever it is uses.
- Has a well-defined interface
- Complete and well documented
- Should be robust



Software objects are conceptually similar to real-world objects. Sometimes they could be different too. Ex: -



Employee Accounts

Items: 4
Found: 4
New
Status: Active

Employee Account

Entered 6/25/98 32

Account Name	Captain Ahab
Job Title	
Full Name	Abraham Absalom Alexanders
Hired	6/25/98
Terminated	
Wage Schedule	Starter Laborer
Wage Amount	$8.50
Comments	surprisingly agile on scaffolding, despite peg leg
Cost Type	Permits & Fees

Payroll Setup
Contact Info

View Appointments
View Contact Log
View Document Log
View Problem Log

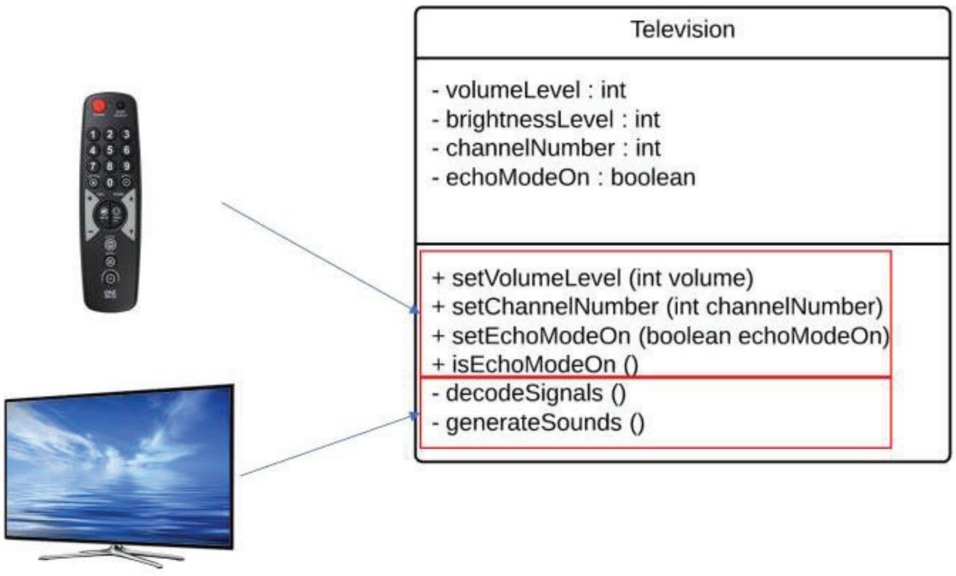
Comp Time Balance	0
Vacation Balance	0
Sick Days Balance	0
Other Vac. Balance	0
Advances Owed	$0.00

View Labor Hours
View Payroll

Concepts of Object Oriented Programming

Encapsulation

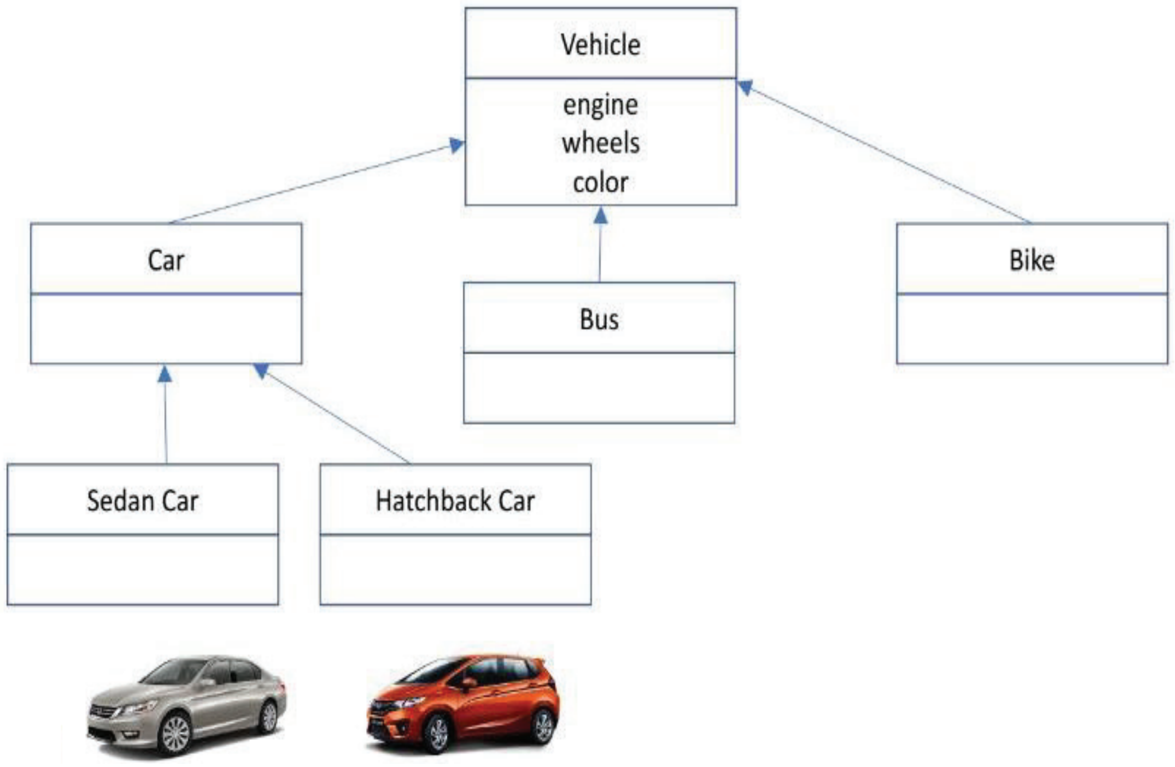
- Hide the mechanics of the object
- You do not want to understand how every bit and piece of the television works in order to operate it.
- The user needs only to understand the interface.
- The programmer can change the implementation but need not notify the user.
- Public methods to describe the interface
- Private method to describe the implementation



Inheritance

If you think of early 90's the mobile had only one facility, take calls. But with the time, it modified with facilities such as sending SMS, browsing Internet and many more. In real world terms this is inheritance. New functionalities are inherited from the old ones.

In an object-oriented scenario, inheritance is a little different from the real-world. Inheritance enables new objects to take on the properties of existing objects. In the below example the Vehicle class is used as the base and the sub classes such as Car, Bus and Bike inherits from the base class.



Polymorphism

Animals can make sounds. For a single method – 'to make sound', different animals will make different sounds.

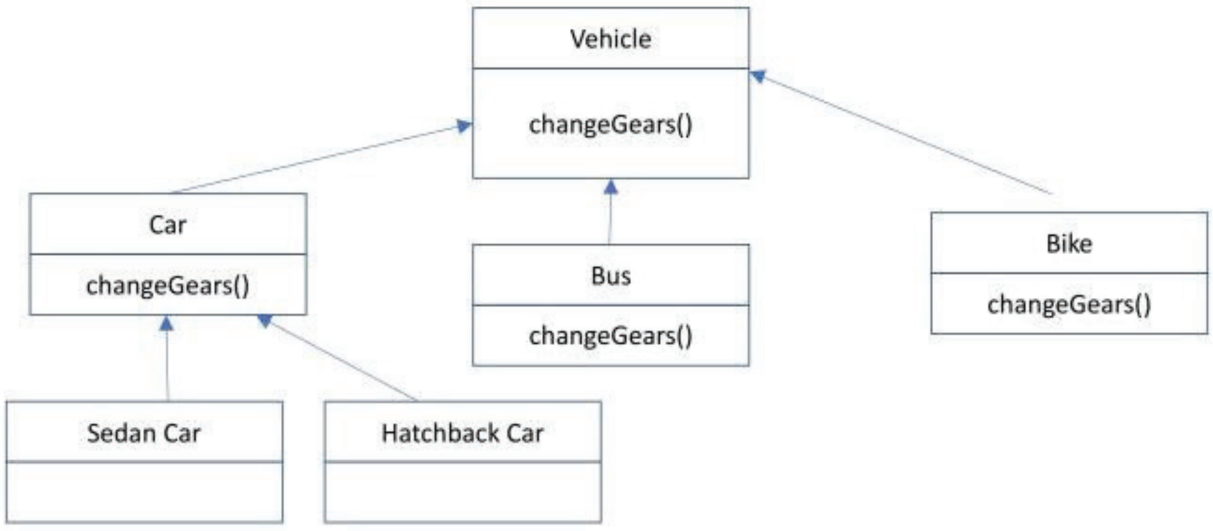
In programming also, a method can be implemented in different forms. This is called polymorphism.

- Polymorphism means "multiple forms".
- Multiple forms of the same method

Polymorphism refers to the ability of an object to provide different behaviors depending on its own nature. There are two ways of it: Overriding & Overloading.

Overriding Methods

Method Overriding is when a method defined in a superclass or interface is re-defined by one of its subclasses, thus modifying/replacing the behavior the superclass provides.



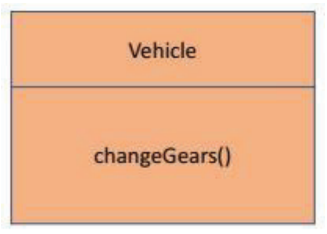
Overloading Methods

It refers to defining different forms of a method. You have one method where it takes different parameters.

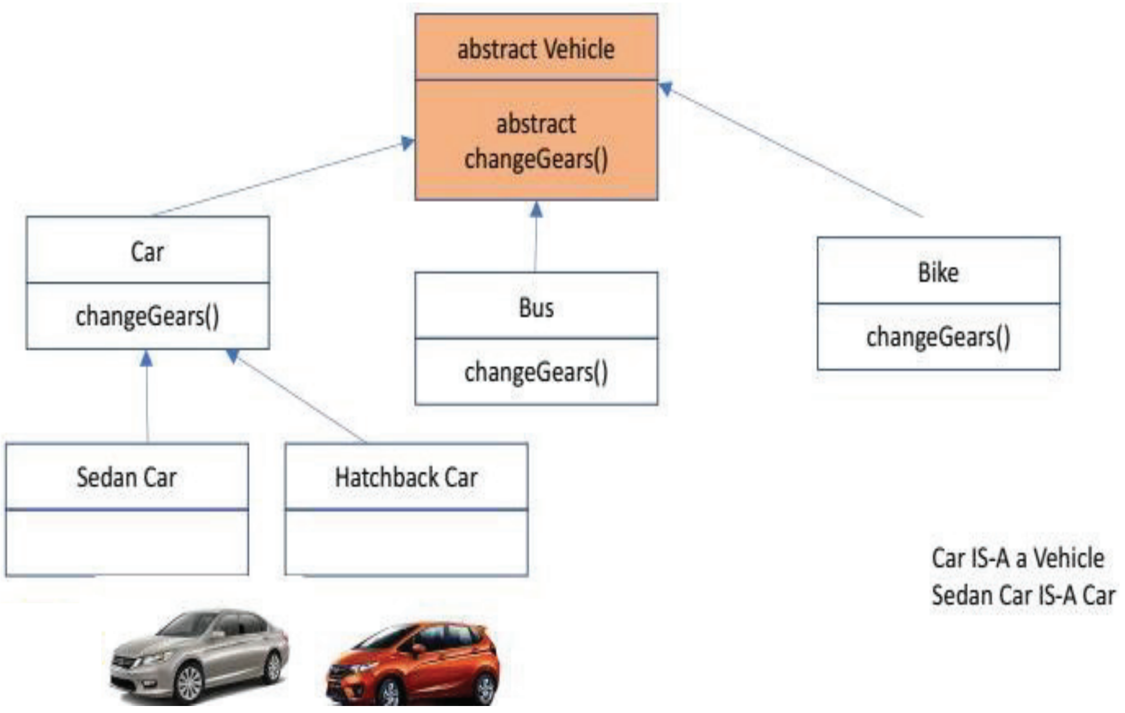
- The same method name can be used, but the number of parameters could be different.
  - add (intx, inty)
  - add (intx, inty, intz)

Abstraction

Through the process of abstraction, a programmer hides all but the relevant data about an object in order to reduce complexity and increase efficiency.



- Can you think about the instance of a vehicle?
  - Cannot be instantiated
  - Can only be used through inheritance

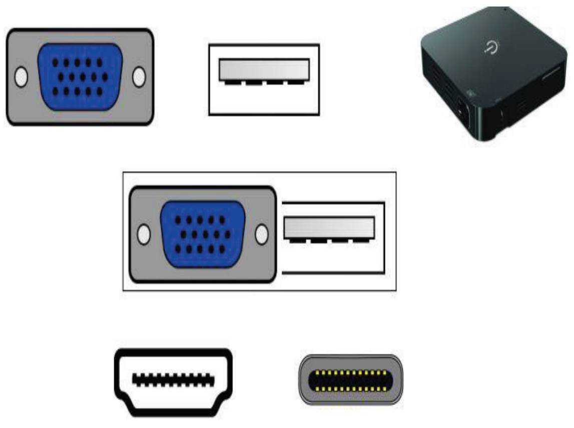


SOLID Principles

It is the acronym for five design principles intended to make software designs more understandable, flexible and maintainable.

Single responsibility principle

Assume there is a VGA port and a USB port. Since they are independent from each other it's easy to do modifications. If they were in the same port when a modification occurs such as the replacement of VGA with a HDMI port, the complete port should be changed.



In the same way if one class handles multiple responsibilities the whole class has to be changed in case of a modification. A class should have one and only one reason to change, meaning that a class should have only one job.

Open for extension, but closed for modification (Open-Closed Principal)

If you need to watch a movie, no one expects to remove the casing and do a modification to the television. Simply the given extension will be used to plug in a device and to watch a movie.

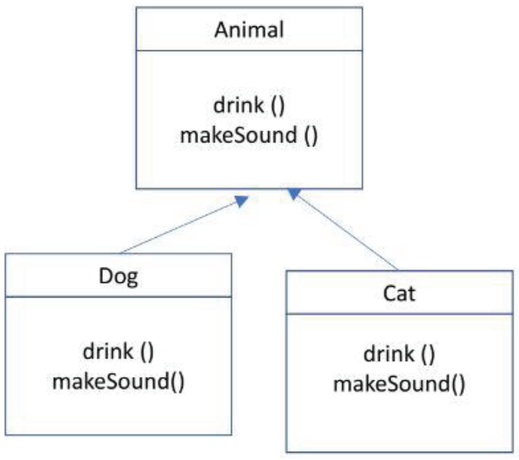
The same way, a class should be open for extension but closed for modification by the client.

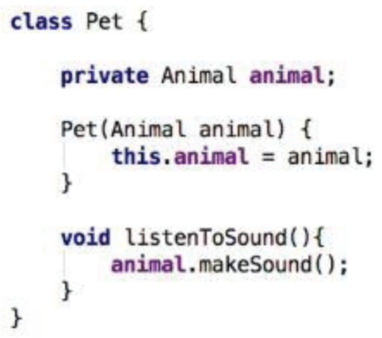
Liskov Substitution Principle

Every subclass/derived class should be substitutable for their base/parent class.

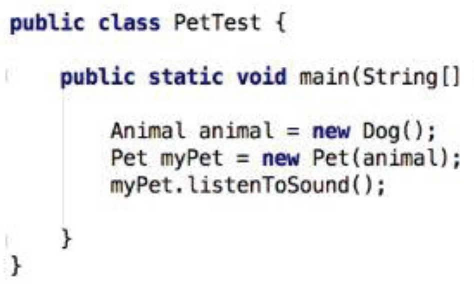
According to Liskov's principle if you have a parent class with its child class, the child class should inherit the functions none other than the exact same of its parent class.

Assume there's a toy cat class that inherits the same animal class given below. The toy cat will do nothing other than making sound. But if you're following the Liskov Substitution Principle, since child class should be substitutable for their parent class the toy cat class should be in a different hierarchy, and not in the Animal class as the toy cat will get the drink functionality if it is in the Animal class, which is a function that a toy cannot perform in the real world.





```
class Pet {

    private Animal animal;

    Pet(Animal animal) {
        this.animal = animal;
    }

    void listenToSound(){
        animal.makeSound();
    }
}
```

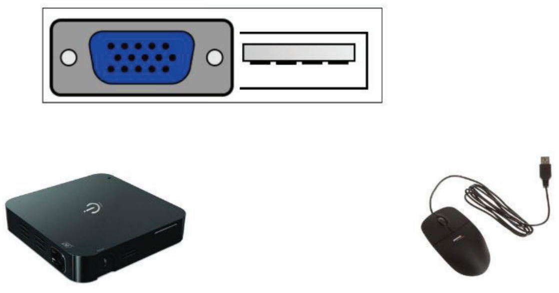


```
public class PetTest {

    public static void main(String[] args) {

        Animal animal = new Dog();
        Pet myPet = new Pet(animal);
        myPet.listenToSound();

    }
}
```

Interface segregation principle

A client should never be forced to implement an interface that it doesn't use, or clients shouldn't be forced to depend on methods they do not use.

Given below is a VGA port integrated with a USB port. For the use of a USB port if someone implements it together with a VGA port also in it, while using the USB port, the VGA port will do nothing. No use of implementing it integrated together.



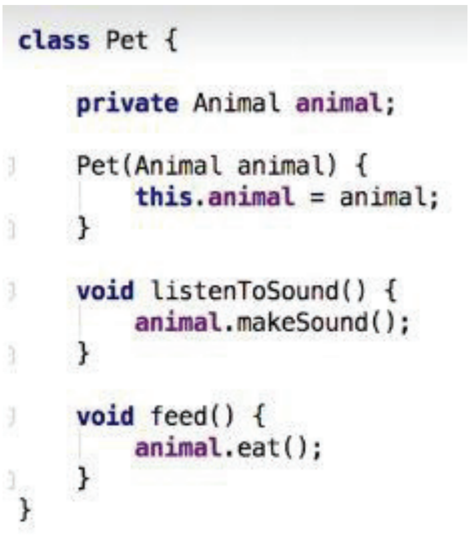
Just the same way, assume in an Animal class you have eat, make sound and fly as attributes. For a sub class of it - ‘bird class’, it will be fine. But for a sub class - ‘dog class’ the fly attribute is not suitable.

In this kind of a scenario instead of a single interface you can have two separate interfaces named eatable and flyable. If the Animal is a bird you can implement eatable and flyable. And if it is a dog you can implement only the eatable interface.

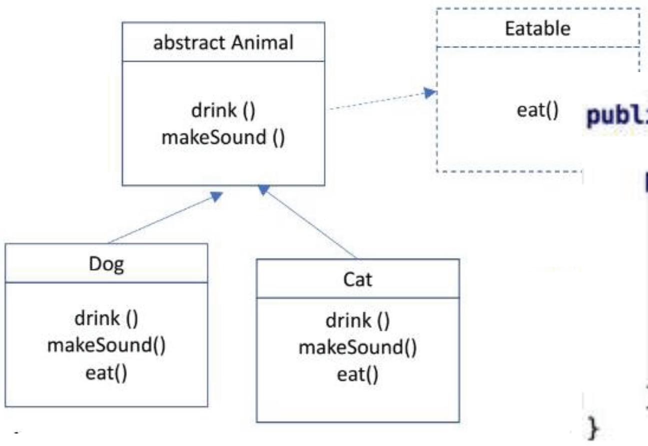
Dependency inversion principle

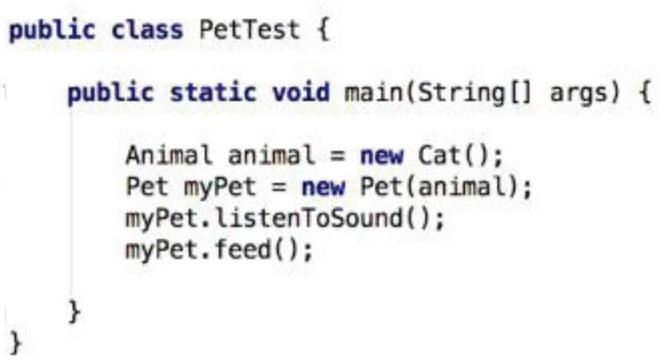
Entities must depend on abstractions not on concretions. It states that the high-level module must not depend on the low-level module, but they should depend on abstractions.

Assume that there’s a message sender class which sends messages in multiple forms (SMS, e-Mail etc.) If you hardcode the values of SMS or e-Mails to the message sender class, in occurrence of a new message sending form you should be able to add that as well. But that makes the high-level module dependent on the low-level one.



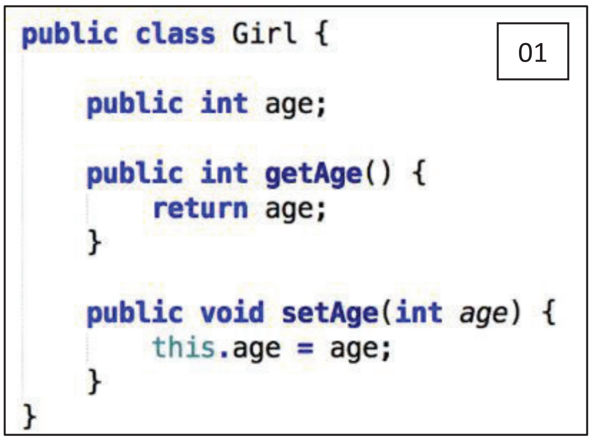
```java
class Pet {

    private Animal animal;

    Pet(Animal animal) {
        this.animal = animal;
    }

    void listenToSound() {
        animal.makeSound();
    }

    void feed() {
        animal.eat();
    }
}
```



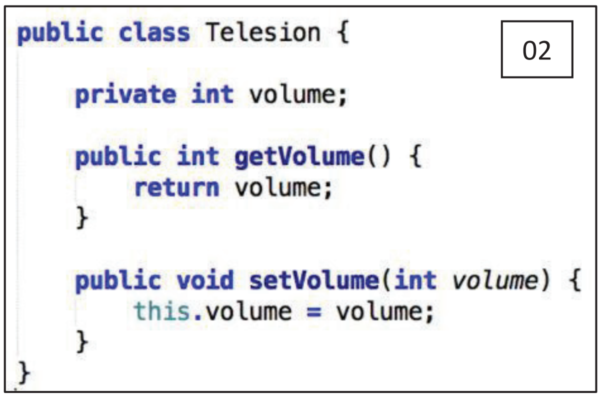


```java
public class PetTest {

    public static void main(String[] args) {

        Animal animal = new Cat();
        Pet myPet = new Pet(animal);
        myPet.listenToSound();
        myPet.feed();

    }
}
```

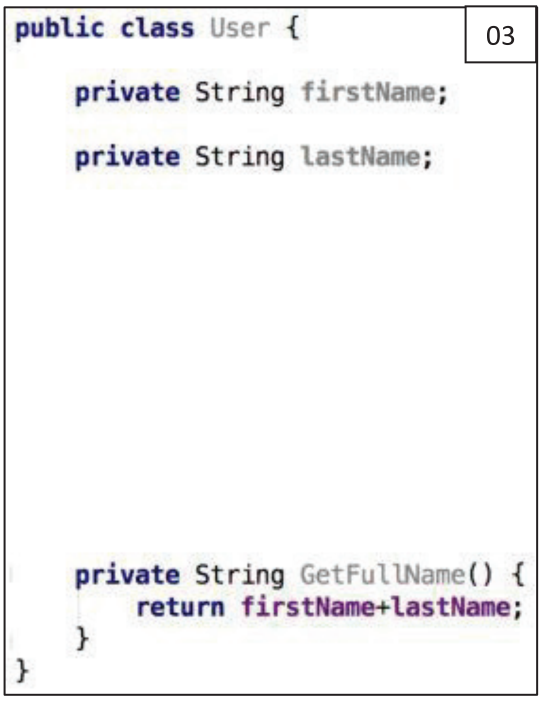
Can you identify the mistakes in each code segment?


01

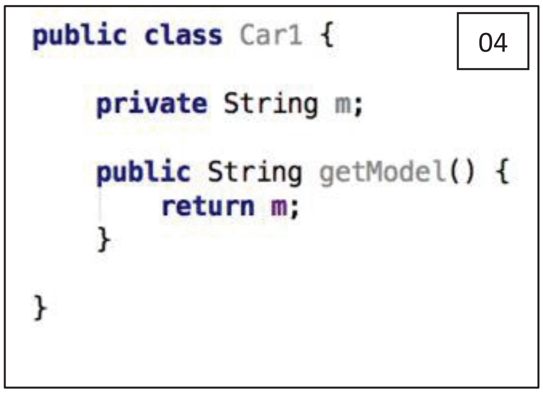
```java
public class Girl {

    public int age;

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```


02

```java
public class Telesion {

    private int volume;

    public int getVolume() {
        return volume;
    }

    public void setVolume(int volume) {
        this.volume = volume;
    }
}
```


03

```java
public class User {

    private String firstName;

    private String lastName;









    private String GetFullName() {
        return firstName+lastName;
    }
}
```


04

```java
public class Car1 {

    private String m;

    public String getModel() {
        return m;
    }

}
```

Case Study

There is a school which is managed by a principal. In that school there are students who are in different grades (Grade 1 to 13). Each grade has multiple class rooms. In each class room students learn subjects, and subjects are taught by teachers. A teacher can be a head of a particular class.

You have to write a software for the school. From your software a parent can view the information about the teacher who teaches a particular subject to his/her child. Your task is to identify the classes (and attributes) in terms of OOP.

5.0 What is a Unit?

What is the smallest unit in the society?

Family is the smallest unit of the society. You cannot break it down in it to any other smaller unit. It cannot be an individual person. But why? To make a unit, there should be a team.

Unit is the smallest entity in programming which you cannot break any further. If you break a unit, it will not be a unit there after.

Image Source -http://www.guibingzhuche.com/group/cartoon-pictures-of-families/

Is coupling good for the society?

For an individual person coupling might be good. But in society if one family depends on some other family, that kind of couplings are unacceptable.

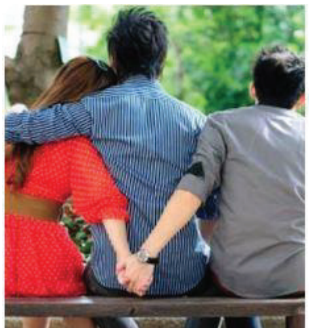


Image Source - https://www.pinterest.com/pin/453808099935067891/

In software, coupling is a measure of how closely connected two modules are. High coupling is bad in software as well. But how?

Assume your family has a family doctor. Whenever any of your family members gets sick you will visit none other than your family doctor. Think of an emergency where your family doctor is unavailable… since your family is tightly coupled to your family doctor the patient will be at risk due to the doctor's unavailability.

This would have not happened if the family used to take medicine from a general doctor than being tightly bound to one specific doctor.

The same way if one unit is tightly coupled with another, and if one unit is affected, the coupled unit will also be affected heavily.

How to reduce coupling?

In order to reduce the inappropriate coupling, you have to increase **Cohesion.**

If your current class is coupled to another class, in case of a modification or adding a new feature to the current class, the other coupled class also gets affected.

Cohesion is about making sure each component does one thing and does it well. By increasing the cohesion, you make one component have little or no interaction with other components in the system. This leads to minimal or zero impact between classes in case of a modification.

Importance of a unit

"If we have stronger families we will have stronger schools and stronger communities with less poverty and less crime." (William Bennett, a former United States secretary of education)

"If we have stronger Units we will have stronger Modules and stronger Software with Rich Features and less Bugs."

Chamil Jeewantha, an Architect @ Zone24x7

What is Unit?

- A **single, indivisible entity**
- The smallest testable part of an application. The smallest unit will depend on the user. Ex: -
    - For an End User -> Entire software might be the smallest unit.
    - QA Engineer -> Single Functionality
    - Architect -> A Module, Component
    - Developer -> A Class (on OOP)

Class? Why not Method (OOP)?

If a class follows the "**Single Responsibility Principle**" (one class should do only one thing [doesn't mean one function]) & the **methods** are **cohesive** (within one single class all the methods should be tightly interrelated) then it's **really hard to use a method independently.**

Example of a Unit

```
public class AndCriteria implements Criteria {
        private final Criteria criteria1;
        private final Criteria criteria2;
 public AndCriteria(Criteria criteria1, Criteria criteria2) {
        this.criteria1 = criteria1;
        this.criteria2 = criteria2;
 }
 public List filter (List values) {
                List filteredList = criteria1.filter(values);
                return criteria2.filter(filteredList);
        }
}
```

In the above example, public class AndCriteria implements Criteria class. The AndCriteria has two criteria's underneath. Any given list will be filtered through criteria1 and the same list's output will be filtered through criteria2, and the filteredList will be returned.

For this same example we can set another AndCritera as critera1 or new criteria named OrCriteria, or a new class called male_gender_filtered_criteria. With its high cohesion this example becomes a good unit which allows to do any amendments without affecting other classes.

Characteristics of a Good Unit

- Not so lengthy – if you have lengthy codes, there's a high probability that you're doing multiple things/ more things than what is expected.
- Self-explained – no need of comments to explain what the code does. Just by looking at the code, it explains its self to the reader.
- Used by its clients via an interface – loose coupling. No one directly uses its class, therefore anytime the class can be removed, and another class can be plugged in.
- SOLID
- Well Tested – whole class must be unit tested for all possible scenarios.
- Compliant with Best Practices (All above + many more)

Common Problems related to Unit

- Too generic names – self unexplained names like…

  Manager
  Service
  Utility
- Unwanted explanations; a generalized class which doesn't have a special name... Ex: -

```
Bottle bottle = new WaterBottle();
bottle.fill();
Holder holder = new WaterHolder()
holder.fill()
```

- Law of Demeter

  Principle of Least Knowledge – you shouldn't get tightly coupled with the implementation of a particular class's inner classes. You should know about the least possible things of the class that you're working on.

  Ex: - man.getBody().getHand().up()

- Many Classes - the classes are transparent, well categorized.

- All in One – one single class for all the work, but if you want to find something this module will put you in trouble for sure.

- Not Tested Separately – since people are not creating proper units, they don't do proper unit tests. Once you start implementing you should start unit testing as well. In that way at the end all your unit tests will be completed.

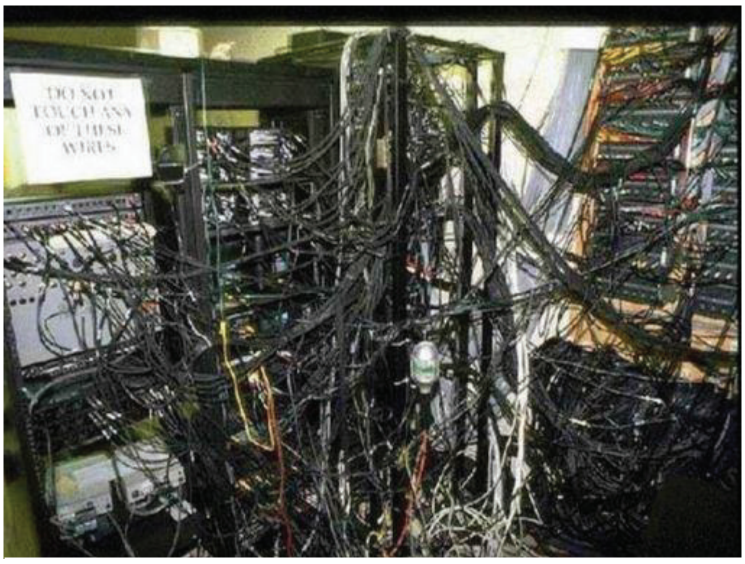


Image Source - https://www.dcig.com/2008/06/datacenter-management-101-part-i-cable-manage.html

Given below is a list of students in a tuition class. Using the given records create a student class list. All the records must be included in the student class.

Exercise:-

Number	Name	Age	Gender	Class
1	Nikini	19	Female	13
2	Piyumal	15	Male	9
3	Sanduni	17	Female	11
4	Chathurika	18	Female	12
5	Nirmal	19	Male	13
6	Sahan	16	Male	10
7	Madhawa	14	Male	8
8	Tharindu	18	Male	12
9	Sachini	16	Female	10
10	Shehani	15	Female	9

1) Write the code to find all male students.
2) Find all the female students older than 18 years.
3) Find male students who are older than 18 or female students who are younger than 18 years.

6.0 What is a Unit Test?

- Select the **smallest piece** of testable software in the application
- **Isolate it** from the rest of the code
- **Determine** whether it **behaves** exactly as **you expect**
- Each unit is **tested separately** before integrating them into modules to test the interfaces between modules

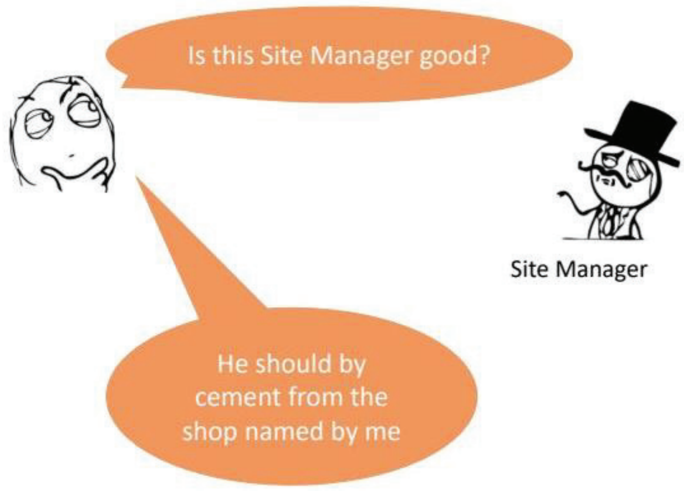
Characteristics of a Good Unit Test

- Automated
- Thorough
- Repeatable
- Independent
    - Test only one thing
    - Should not rely on each other
- Fast
- Professional (Readable, Maintainable, Trustworthy)

Isolating A Unit for Testing

The main aim of isolating each unit of the system is to identify, analyze and fix the defects.

Given below is a real-world scenario of how to isolate a unit for testing.


Is this Site Manager good?
Site Manager
He should by cement from the shop named by me

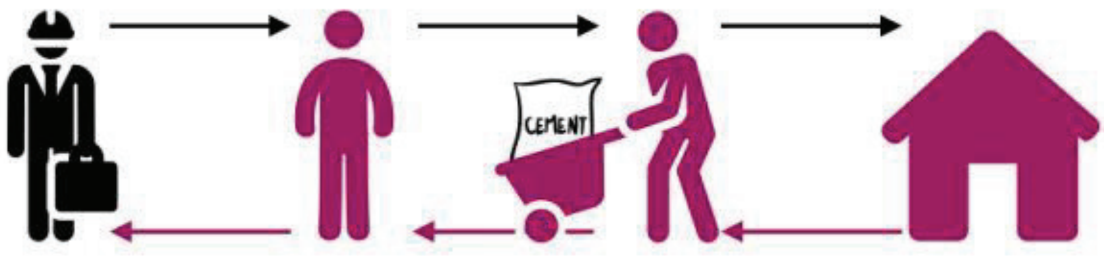
Assume there's a building to be constructed. The building owner needs to assign a trustworthy site manager to construct the building.

To verify something, the 1st thing is to have a controlled environment. In a controlled environment, nothing will change except the facts that are changed by the controller itself

Only if the environment is under control can we test for a particular thing in that environment. Therefore, to make a controlled environment the best option is to create dummies.

To verify whether the site manager works as expected the owner names a specific hardware and instructs the site manager to buy cement from that shop for any cement needs of the building.

Test Flow



How to Test 1.0

- Create a dummy bag of cement
- Create a dummy hardware shop
- Instruct the dummy hardware to send the dummy bag of cement if somebody asks for a bag of cement.
- Give the address of the dummy hardware to the site manager.
- Ask for a bag of cement from the site manager
- Verify whether he has called the given hardware
- Verify whether we receive the dummy cement bag back.

How to Test 1.1

Creating dummy thing means creating a mock. If you want bag of cement mock what you should do is simply create an instance of dummy cement.

```
BagOfCement dummyCement = mock(BagOfCement.class);
//mock of bag of cement is provided
HwShop dummyHw = mock(HwShop.class);
// mock of hardware shop is created
When(dummyHw.buyCement()).thenReturn(dummyCement);
 //When dummyHardware.buyCement is called, then return dummy cement
siteManager.setHwShop(dummyHw);
//Finally, the siteManager is set for the dummyHardware

BagOfCement output = siteManager.askCement();
//Site manager buy a cement bag

assertThat(output, is(dummyCement));
verify(dummyHw).buyCement();
//verification of dummyHardware buyCement is called
```

By the above example it is visible that there is no dependency of other classes. Developer depends only on the interfaces. Developer is simply mocking them and getting the job done.

To test this scenario, two mocks can be created. One mock for criteria 1 and another one for criteria 2. For criteria 1, when the mock list one is given it should return mock list 2 if somebody calls the filter method in mock list 1. If developer provides mock list 2, then it should return mock list 3. That is the expectation.

The expectation of AndCriteria is, whatever the value given should be go through criteria 1. Whatever the output of criteria 1 should go through criteria 2, and whatever the output of criteria two should be returned to the developer. If it happens the AndCriteria works.

How to Test 1.2

```
public class AndCriteria implements Criteria{

        private final Criteria criteria1;
        private final Criteria criteria2;

        public AndCriteria(Criteria criteria1, Criteria criteria2) {
                this.criteria1 = criteria1;
                this.criteria2 = criteria2;
        }
        public List filter(List values) {
                List filteredList = criteria1.filter(values);
                return criteria2.filter(filteredList);
                }
}
```

Example: Requirement of AndCriteria

- The AndCriteria should be an implementation of Criteria
- Should filter a given list by first expression and return a filteredList
- Should filter the filteredList by the second expression and return the finalist

Example: Tests for AndCriteria

1.Should_**FilterThrough1StAnd2ndExpressions**_When_**AListIs Given**

2.Should_**ThrowException**_When_**NullIsProvided**

Example: AndCriteria : Test 1

```
@Test
public void
Should_FilterThrough1StAnd2ndExpressions_When_AListIsGiven(){
        List input = // dummy list
        List lst1 = // dummy list
        List lst2 = // dummy list

        Criteria expr1 = // dummy criteria -> filter(input) returns dummy list (lst1)
        Criteria expr2 = // dummy criteria -> filter(lst1) returns dummy list (lst2)

        AndCriteria criteria = new AndCriteria(expr1, expr2);
        List output = criteria.filter(input);
        assertThat(output, is(lst2));
}
```

Unit testing gets harder only when you have dependencies on other classes. For example, within your class if you are referring directly to another method in a different class, the ease of unit testing won't be there. If you're referring to another static method in another utility class, you'll never get through unit tests because you cannot mock them as they are hard bounded. You cannot easily replace them with mocks…

A class that works with dummies will work with anything. It is that much reusable, collaborative…high cohesive with the other classes. All this luxury is there because it is testable. In Quality Code, if you write a class that is testable independently, that means that class will automatically achieve all the good quality facts of Quality Code.

Example: AndCriteria : Test 2

```
@Test (expected = IllegalArgumentException.class)
public void Should_ThrowException_When_NullIsProvided() {
        List input = null
        AndCriteria criteria = new AndCriteria(expr1, expr2);
        criteria.filter(input);
}
```

Example 2: Evaluate Expression

- The user should provide an expression with relevant value mappings to its variables.
    - Java Evaluate ((a+b)*3)-2 a=5 b=8
- The program should assign a & b values to this expression and evaluate it.

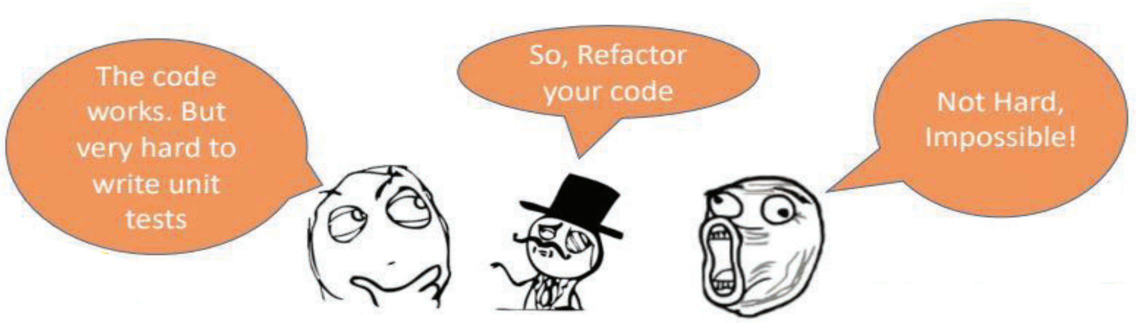
Code for Evaluating an Expression

```
class Evaluate{
        public static void main(String[] args){
                String expr = ... // assign variables with values
                // evaluate the expr
        double value = // the output value of the evaluation
        System.out.println(value);
        }
}
```

Can you write a "Good" automated test?

Common Complaints About Unit Testing

- Deadline is near, **no time** to write tests
- Writing tests **takes longer than** the **production code**
- **Hard to keep** the test suite **up to date**
- **One line of code change breaks 100s of tests**



Solution

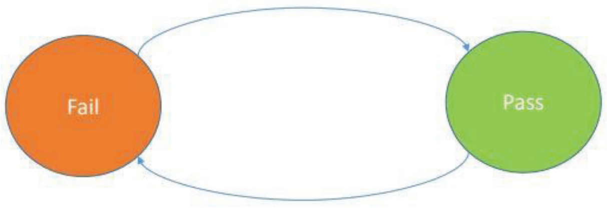
Test First Development (TFD)?

Rules: TFD

1.**Write** a (another) unit **test that fails**

2.**Write** the **minimum production code** until all the tests pass

3.**Repeat** until all your work is done.



Test First Benefits (Vs Test Late)

All the benefits of Unit testing

+

- Write **non-testable codes are impossible**
- Test-first **forces** you to **plan before you code**
- It's **faster than writing** code **without tests**
- It **saves you from lengthy code**
- It **guides you** to **build** good, **SOLID units**
- It **increases your confidence (refactor without fear)**
- Acts as a **real-time progress bar**

What is TDD?

TDD = TFD + Refactoring

- TDD = Test Driven Development
- TFD = Test First Development

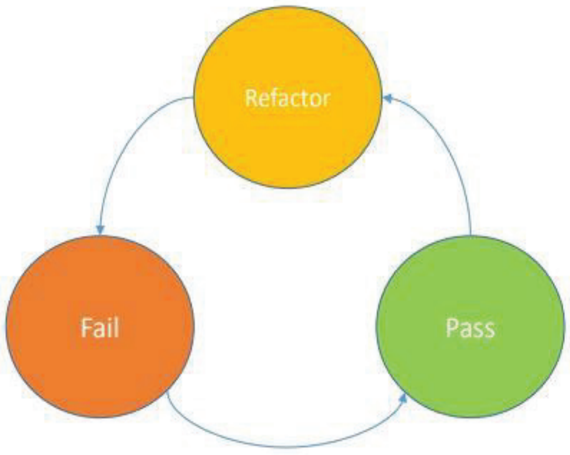
Rules: TDD

1.Write a(nother) unit test that fails

2.Write the minimum production code until all the tests pass

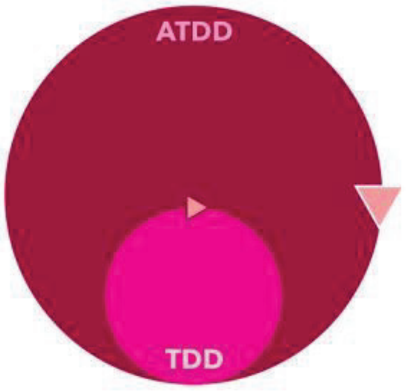
3.**Refactor your code**

4.Repeat until all your work is done



Tools with TDD

- IDE (IntelliJ)
- Test Runner (Junit)
- Mock libraries (Mockito)
- Verification (Hamcrest)



ATDD vs TDD

- TDD to drive the design
- ATDD to make sure all the requirements are implemented

Additional time due to TDD?

- Beginner
    - Lots of time thinking where to start
- Experienced Developer
    - Initially 15 - 17% more
- Big time saving later for both

7.0 Writing Quality Code

Anybody can write working code…what is hard is to write a maintainable code! But why maintainable code is important?

Why Maintainability

- If code is not written well
    - It might work well
    - However, it is not maintainable
        - Difficult to fix defects
        - Difficult to change
        - Difficult to add new features

When the code is difficult to understand, it is hard to change…

In a software product, the initial development cost is a little high. But when compared to long term use of it the cost is negligible. Since these software products are being used for a while (few years or more), if the initial development hasn't been done properly;

- From a client's perspective, bad code would result in
    - Higher maintenance cost
    - Cannot be changed with the changing business needs
    - Most applications have to be re-written after some time, due to bad code

- From a developer's perspective, bad code would make them
    - Less productive
    - Frustrated, by having to do deal with a complex mess everyday

Therefore, it is important to write quality code from the very beginning. So that both parties would benefit without hassle.

What makes the code maintainable

- Simple - If the code is not simple, it's not maintainable. Skilled coders write simple code, not complex code.
- Readable
- Easy to Change
- Has good diagnostic infrastructure
- Has good Unit Test coverage

What makes the code simple

People tempt to write over complex code including all the different technologies that they've learned assuming that it's the best. Then what makes it simple…?

- Keep it simple and stupid (KISS)
    - Best code is not the shortest code
    - Best code is not the highest performant code

  (pre-mature performance optimizations may have negative impact on both performance and maintainability)

    - Best code is the code that anybody can understand easily.

Many people try to implement all possible future requirements and write code or do a design to cater to all of them. But most of the time the future expected requirements may never arise. So, the best thing is not to design something that is not known, which you think it will come as future requirements. You will add unwanted complexity to your code by that.

- You Aren't Going to Need It (YANGI)
    - Don't add complexity to cater for future requirements
    - Don't use design patterns just to cater for unseen future requirements or for the sake of using them

Even though you write simple code if it is not in a readable format people still find it hard to maintain. Even if it is a simple few lines of code that is not readable, it will make issues in maintainability.

What makes the code Readable

- Use meaningful names
- Follow consistent naming convention
- Use small and simple methods
- Properly format the code
- Add proper code comments
- Separate different concerns into deferent classes

Use meaningful names

Code should speak for its self. Without looking at the comments, simply going through the code, people should be able to understand the code. Therefore, always use meaningful variables.

- Class names, method names and variable names should be meaningful, so that the code can describe itself without the need of comments
- Method names should correctly represent exactly what the method does (no more, no less) Ex: -
    - Never call a method 'getOrder', if it creates a new order if one does not exist. Because it should only say what it does.
- Class and variable names should correctly represent exactly what the class, variable represents
    - Never use variable names such as x, y, i, j, obj, emp1, emp2, etc.
- Use nouns to name classes and variables (e.g. class – Order, variable – order)
- Use verbs to name methods (e.g. getName, setAddress, save, createEmployee, etc.)

Follow a consistent naming convention

Within the application also, make sure to follow a consistent naming convention.

- Use consistent verbs to create method names
    - A method with 'get' prefix should never alter the state and should always return a value
    - A method with 'set' prefix should always modify the state and should never return a value
    - Never use different prefixes to represent the same action

    (e.g. never use the below methods to indicate that some entity is inserted to the database within the same application)
- insertOrder, createInvoice, saveEmployee, addCatalogItem
- Never use two words to mean the same thing (same concept) within the same application
- Never use the same word to mean multiple things within the same application

How to make methods looks simple

Method should look simple for it to be readable. Though the content is simple if you write it in a bad way it will look complex.

- Remove nested conditions
- Remove control flags
- Introduce descripting local variables
- Split temporary variables
- Break long methods into smaller methods

Simple Methods- Remove Nested Conditions

- A method with nested conditions

The objective of the below method is that somebody passes the call information. And you are supposed to calculate the call charges.

```
public decimal GetCallCharges(CallInfo callInfo)
{
    decimal charge;

    if (callInfo != null)
    {
        if (callInfo.IsIDDCall)
        {
            charge = GetIDDCallCharges(callInfo);
        }
        else
        {
            if (callInfo.IsPeakTime())
               charge = GetPeakTimeCallCharges(callInfo);
            else
               charge = GetNormalCallCharges(callInfo);
        }
    }
    else
    {
        throw new ArgumentNullException(nameof(callInfo));
    }

    return charge;
}
```

What the code does is simple. But the look of it...?! It is better if you can understand the code just by a looking at it. You’ll have to spend a considerable amount of cognitive cover just to understand this very simple code. Whereas in the second example it is just a matter of a simple glance that is needed to understand the code.

Simple Methods- Remove Nested Conditions

- A method with guard clause

Given below is the same previous concept that calculates the call charge. The main difference in this code is that with a single glance at it you will be able to understand the complete requirement.

```
public decimal GetCallCharges(CallInfo callInfo)
{
    if (callInfo == null)
        throw new ArgumentNullException(nameof(callInfo));

    if (callInfo.IsIDDCall)
        return GetIDDCallCharges(callInfo);

    if (callInfo.IsPeakTime())
        return GetPeakTimeCallCharges(callInfo);

    return GetNormalCallCharges(callInfo);
}
```

The best practice is to do your validations first, then handle the special cases and handle the normal/ general cases at the end. Having multiple return values isn't that bad in programming. Because it can help you to reduce the complexity of your code.

Another programming tactic is to avoid control flags. With lots of control flags you are adding additional complexity to the code.

Simple Methods - Remove Control Flags

- A method with control flags

```
public string GetUsername(int userId)
{
    User requiredUser = null;
    bool userFound = false;
    for (int i = 0; i < users.Count && !userFound; i++)
    {
        if (users[i].UserId == userId)
        {
            requiredUser = users[i];
            userFound = true;
        }
    }

    if (userFound)
        return requiredUser.Name;
    else
        return null;
}
```

- A method with no control flags

```
public string GetUsername(int userId)
{
    foreach (var user in users)
    {
        if (user.UserId == userId)
        {
            return user.Name;
        }
    }

    return null;
}
```

Simple Methods - Descripting local variables

Given below is a code that calculates the area of a triangle. First example uses one single line to calculate the total area where the second example calculates the area in a descriptive manner.

In coding, simple doesn't mean the shortest code. By trying to make the shortest code there's a high possibility that you will end up with a complex code.

- A lengthy expression

```
double totalArea = triangle.width * triangle.height / 2 + rectangle.width * ractangle.height;
```

- After descriptive variables are introduced

```
double areaOfTriangle = triangle.width * triangle.height / 2;
double areaOfRectangle = rectangle.width * ractangle.height;
double totalArea = areaOfRectangle + areaOfRectangle;
```

Simple Methods - Split Temp Variables

Another best practice is never trying to reuse the variables to do multiple things.

- Code that reuses the same variable to represent two logical concepts

E.g. Area of triangle & area of rectangle

```
double totalArea = 0;
double area = triangle.width * triangle.height / 2;
double totalArea += area;
double area = rectangle.width * ractangle.height;
double totalArea += area;
```

- Code after the temp variable is split

```
double areaOfTriangle = triangle.width * triangle.height / 2;
double areaOfRectangle = rectangle.width * ractangle.height;
double totalArea = areaOfRectangle + areaOfRectangle;
```

Break long methods into small methods

Last but not least, if there are very long methods, then also it is very difficult to understand.

- Number of lines in a method should not exceed the number of lines that can be viewed using your IDE
- If a method is too long or too complex, break it into smaller methods by extracting a set of small private methods with the same abstraction level
- The initial method should represent the high-level algorithm as a story
- Subsequent methods should represent the detailed implementation

- If it is hard to break a method, since its internal logic is tightly coupled through lots of local variables,
    - Extract the method into a separate class
    - Convert local variables that cause coupling into member variables of that class
    - Break the method (extract private methods that are at the same level of abstraction)

Given below is another example, which is implemented to do a withdrawal from a bank account. Though it's a very simple thing, the code looks very complex.

Ex 1: -

```
public TransactionReponse Withdraw( string accountNo, decimal amount, int userId)
{
    var account = accountRepository.GetAccount(accountNo)
    if (account.Balance >= amount)
    {
        transactionRepository.AddTransaction(
            new Transaction(accountNo, -amount, userId, DateTime.UtcNow);
        account.Balance -= amount;
        accountRepository.Save(account);
        return new TransactionReponse(TransactionStatus.Completed);
     }
     else
     {
         var odStratergy = odStratergyFactory.GetStratergy(account.AccountType);
         if (odStratergy.CanIssueOD(account))
         {
             decimal preApprovedCreditLimit = odStratergy.GetPreApprovedCreditLimit(account);
             if (preApprovedCreditLimit < amount)
             {
                 odRepository.AddOD(
                     new OverDraft(accountNo, amount, userId, DateTime.UtcNow));
                 transactionRepository.AddTransaction(
                     new Transaction(accountNo, -amount, userId, DateTime.UtcNow);
                 account.Balance -= amount;
                 accountRepository.Save(account);
                 return new TransactionReponse(TransactionStatus.Completed);
              }
         }
         return new TransactionReponse(TransactionStatus.Rejected, StringResources.InsufficientFunds);
     }
}
```

Given below is the same example in a different approach. Methods are simple, and by one look you can simply understand what it does. That is because the complexities of the multiple steps are now hidden inside implementation details, hidden inside private methods. Therefore, the public method can be read as simple as reading English…

Ex 2: -

```
public TransactionReponse Withdraw( string accountNo, decimal amount, int userId)
{
        var account = GetAccount(accountNo);
        if (account.Balance >= amount)
        {
                AddTransaction(account, -amount, userId);
                return new TransactionReponse(TransactionStatus.Completed);
        }
        else
        {
                ProcessAsOD(accountNo, amount, userId);
        }
}
```

Here shows the description of each method. This is how you process the order. Here again you check the strategy

Ex 3: -

```
private TransactionReponse ProcessAsOD( string accountNo, decimal amount, int userId)
{
       var odStratergy = GetODStratergy(account.AccountType);
        if (odStratergy.CanIssueOD(account))
        {
                decimal preApprovedCreditLimit =
                odStratergy.GetPreApprovedCreditLimit(account);
                if (preApprovedCreditLimit < amount)
                {
                        RecodeOD(account, amount, userId);
                        AddTransaction(account, -amount, userId);
                        return new TransactionReponse(TransactionStatus.Completed);
                }
        }
        return new TransactionReponse(TransactionStatus.Rejected,
        StringResources.InsufficientFunds);
}
```

The guideline is all the public methods should be easy to read and it should reveal the high-level algorithm of the method. High level algorithm should never be hidden. It should be public, and you should be able to read it like an English story. That much of clarity should be there followed by detailed explanations of each high-level step that describes the top level public method.

Sometimes this can also be a bit challenging if you have lots of local variables that are being reused. That's where the solution 'break methods' comes in. In reality that is also something doubtful if you're unaware of the tactics of properly breaking the method.

Break Methods - Use methods objects

Use method objects to break complex methods that are tightly coupled through local variables

See the below example

```
public void ComplexMethod()
{
   //Load Data
   var initialData = LoadInitialData();
   var secondaryData = LoadSecondaryData(initialData);
   //Validate secondry data
   //...
   int totalProp1 = 0;
   int averageProp2 = 0;
   int minProp3 = 0;
   foreach (var item in secondaryData)
   {
      //calculate summary
      //...
   }
   //calculate return value using summary result
   //... res = some expression of totalProp1, averageProp2 and minProp3
   return res;
}
```

In the above example values are assigned to each variable. Without these variables, you can't do the calculations. If you try to break it, you may have to pass about ten variables to every method. And then again if you have to change the values of the variable it is even harder. How to solve this…? Use methods objects

Break Methods - Use methods objects

Create a separate method that represents the complex section, or in other words a class that represents that complex method. When the object represents your method all your local variables of that method becomes member variable of your class.

Best practice in the normal situations is never to make local variables as member variables. But when your method becomes a class it is legal to write local variables as member variables. You don't have to pass lots of variables and you can always change that. So, the lifetime of that object will be only limited to execution of that single operation.

Ex: -

```
private class ComplexMethod
{
   public IList<int> InitialData { get; set; }
   public IList<MyEntity> SecondaryData { get; set; }
   public decimal TotalProp1 { get; set; }
   public decimal AverageProp2 { get; set; }
   public decimal MinProp3 { get; set; }
   public decimal ComplexMethod3()
   {
      LoadData();
      Validate();
      CaclulateSummeries();
      return CalculateResult(parameters);
   }
   //Private methods comes here
```

Properly Format the Code

- Vertical Formatting
    - Let the users to read a class in summary and details format
        - Keep constants and member variables of the class at the top to make them easy to find
        - Public interface methods that describe the high-level flows (algorithms) of the actions that need to come next
        - Private methods with low level implementation details should follow
- Higher the density of information in the screen, higher the readability
    - Reduce the number of empty lines
    - Never add unnecessary comments
- Horizontal Formatting
    - Break the statements into multiple lines if they are too long to be seen in a single screen in your IDE
    - Break the statements in logical places so that the readability of the code is not impacted

Add Proper Comments

- Say WHY, Not WHAT
    - What is being done is visible from the code (proper code should be self-explanatory without comments)
    - Use comments to say why you did it in that way, when it is not very obvious
- Below are the valid exceptions to the above rule
    - Add comments as warnings to other programmers on the consequences of possible changes that they might do in the future
    - Add comments to emphasize the importance of some actions in the code, which would otherwise look not so important
    - Add comments to indicate further actions that are needed (TODO:), etc.

- Remove bad comments
    - Redundant comments increase the code length and hence the readability
    - Inaccurate (not properly updated) comments can be very harmful
    - Commented code reduces the readability (act as Zombie code that nobody dares to remove later)

What makes the code easy to change

- One should be able to read, understand and change each code unit, module, etc. without having to analyze the rest of the code base.
    - Use small cohesive classes
    - Encapsulate the implementation logic and expose only a stable interface
- There should be only one unit that needs to be changed to accommodate a single change to the requirements
    - Ensure that the application is broken into a cohesive set of classes
    - Avoid duplicate logic (DRY – Don't Repeat Yourself principle)
    - Avoid hard-coded constants

To make the code easy to change - Use small and cohesive classes

- Create classes that does only one thing and does it completely
- Break application functionality into classes so that any functional change can be achieved by changing a minimum number of classes
- Create classes that represent domain concepts
- Create classes to represent domain commands when there is complex logic associated with processing them
- Always wrap all calls to external services and components in separate classes, so that the rest of the code remains stable from the changes in external dependencies

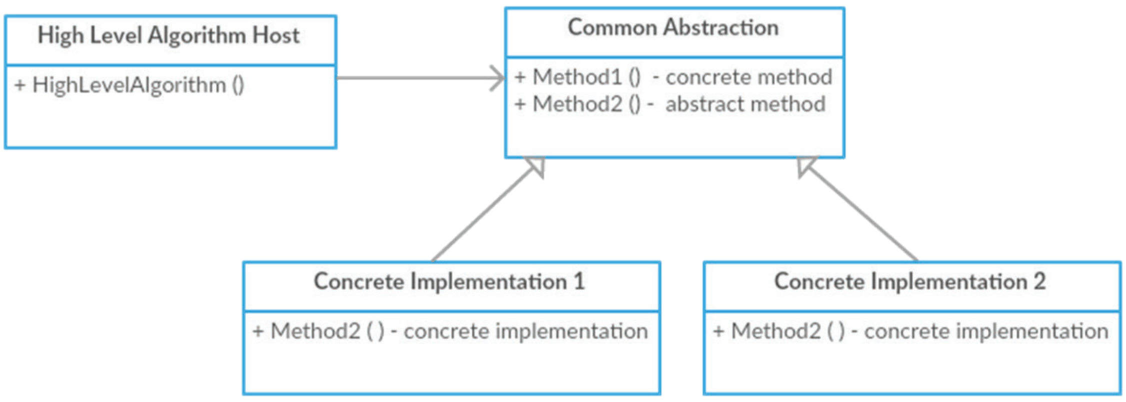
How to encapsulate the implementation

Encapsulation has nothing to do with security. It's all about maintainability.

- Encapsulate an implementation of an algorithm within a method
    - When possible, make output of a method to only depend on its inputs
    - When possible, make the output of a method to only depend on its input and immutable state of the containing object
- Encapsulate the internal data structures and implementation logic of classes using basic OOP features of your programming language (e.g. private and protected keywords)
- Encapsulate the implementation details of software components using façade pattern

Avoid duplicate logic

- Shared Algorithms can be classified into two types based on how they are handled
  - Technical algorithms such as encryption, encoding, searching, sorting, formatting, mathematical calculations, etc.
  - High level algorithms that operate on common abstractions
  - Concrete implementations shared between instances of a common abstractions


High Level Algorithm Host
+ HighLevelAlgorithm ()
Common Abstraction
+ Method1 () - concrete method
+ Method2 () - abstract method
Concrete Implementation 1
+ Method2 ( ) - concrete implementation
Concrete Implementation 2
+ Method2 ( ) - concrete implementation

- Create static helper methods to implement technical algorithms
- Share logic between entities that can be derived from a common abstraction
  - Use inheritance when there is a single common shared abstraction
  - Use strategy design pattern when there are multiple such common abstractions (use composition over inheritance)
- Share high level algorithms that operate on shared abstraction
  - Use polymorphism to create shared logic that can operate on common abstraction
  - Use template design pattern to share high level algorithms within an inheritance hierarchy

Have a good Diagnostic Infrastructure

What has been learned so far is important to understand what we're writing by looking at the code to make it simple and maintainable. But when you deploy the code to the production environment or the QA environment, and there are issues in those environments, a good diagnostic infrastructure is essential. Below points are critical in such situations to figure out what's going on.

- Input Validation
- Exception Handling
- Logging
- Monitoring

Input Validation

- Input validations makes the integration of libraries simple
- Proper validations help us to detect the true cause of the errors without debugging
- Ensure all input parameters are validated to be not null if null is not an acceptable input value
- Ensure all values returned by all internal method calls are validated to be not null if null values are not acceptable
- Ensure all exceptions that can arise due to invalid inputs (e.g. FileNotFound exception) are handled and a friendly custom exception that explains the cause is re-thrown

Exception Handling

- Prefer Exceptions over Error Codes
- At application or thread boundaries
    - Catch all exceptions
        - Log all exceptions
        - Generate a meaningful message for a known exception
        - Generate a generic message for unknown exceptions
        - Display message to user if this is a UI application
        - Return the error message to the client if this is a service application
- At all other places
    - Catch known exceptions and re-throw custom exceptions with meaningful messages with sufficient diagnostic details

Logging

- Look from the point of view of a person who diagnoses a problem when deciding what to log
- Log sufficient details to diagnose problems
- Log messages do not require localization (unless requested by the client)
- Log context details (request parameters, etc.) together with exceptions whenever possible
- Log information such as thread id, request id, session id, etc. to correlate the log entries
- Make the log level configurable
- Ensure that no unhandled exception can go without being logged
- Always log errors even in the production environment

Exercise

Introduction:

Develop a mobile billing engine (a simple class with a method to do the bill generation E.g GenerateBills) that can generate the monthly bill using call detail records (CDR). Assume that the engine is provided with a list of mobile customers and a list of CDRs that contain all the details of the calls made by these customers in the given billing month. The engine should generate bills for all the customers. This bill should include the call charges for each of the calls made by the customer, total call charges for the month, monthly rental, government tax, discounts (when applicable) and the final bill amount.

A customer object includes the below details,

- Full name
- Billing address
- Phone number
- Package Code
- Registered date

A CDR includes below details,

- Phone number of the subscriber originating the call (calling party)
- Phone number receiving the call (called party)
- Starting time of the call (date and time)
- Call duration in seconds

Phone numbers are given in the below format: xxx-xxxxxxx where first 3 digits represent the extension and the last 7 digits represent the unique phone number within that extension.

The billing is based on the package purchased by the customer, call destination (local / long distance) and call time (peak/ off peak). The rates can be calculated per second or per minute usage.

- Local calls – calls where the originating number and the receiving number both have the same extension
- Long distance calls – calls where the originating number and the receiving number have different extensions
- Peak hours – 8:00 am to 8:00 pm (excluding 8:00 pm)
- Off peak hours – 8:00 pm to 8:00 am (excluding 8:00 am)

Note: International Direct Dialing (IDD) calls are out of scope for this version of the invoicing engine.

The government tax will be 20% of the total bill excluding any discounts and taxes (total call charges + monthly rental).

Step 1:

Develop a billing engine that generates the monthly bill using the rules given below.

Note: All figures are given in LKR

- Monthly Rental – 100
- Billing Type – Per minute

Call type	**Per minute charge**	
	Peak Hours	**Off-peak hours**
Local calls	3	2
Long distance calls	5	4

Step 2:

Let the billing charges and the billing type depend on the package. Below are the two packages that need to be supported.

Package	**Monthly Rental**	**Billing Type**	**Call type**	**Per minute charge**	
				Peak Hours	**Off-peak hours**
Package A	100	Per minute	Local calls	3	2
			Long distance calls	5	4
Package B	100	Per second	Local calls	4	3
			Long distance calls	6	5

Step 3:

Add the following two packages to the system.

Package	**Monthly Rental**	**Billing Type**	**Call type**	**Per minute charge**	
				Peak Hours	**Off-peak hours**
Package C	300	Per minute	Local calls	2	1
			Long distance calls	3	2
Package D	300	Per second	Local calls	3	2
			Long distance calls	5	4

Step 4:

- Change the peak and off-peak hours of Package A as
  - Peak hours – 10:00 am to 6:00 pm (excluding 6:00 pm)
  - Off peak hours – 6:00 pm to 10:00 am (excluding 10:00 am)
- Change the peak and off-peak hours of Package C as
  - Peak hours – 9:00 am to 6:00 pm (excluding 6:00 pm)
  - Off peak hours – 6:00 pm to 9:00 am (excluding 9:00 am)
- Make the first minute of all local off-peak calls free of charge for Package B
- Make the first minute of all local calls free of charge for Package C
- Give 40% discount for Package A and Package B if the total call charges for the month exceeds 1000 LKR

References

[1] ‘Clean Code: A Handbook of Agile Software Craftsmanship’ – By Robert C. Martin

[2] ‘Refactoring: Improving the Design of Existing Code’ – By Martin Fowler

[3] ‘Design Patterns: Elements of Reusable Object-Oriented Software’ – By Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

[4] ‘Object-Oriented Analysis and Design with Applications’ – By Grady Booch

[5] ‘Working Effectively with Legacy Code’ - By Michael C. Feathers