Naming conventions ( clean code )

Proper Folder Structure for solid.

1st june clean code :

Things to cover : Clean code , Basic Folder Structure for SOLID

**Clean code**

1. Clean code Formatting : Separate concepts vertically , related code vertically dense,

Related , dependent functions closed , white spaces ,

Indentation

1. Clean code cheat codes
2. Naming Conventions

Class - Should start with Uppercase letter , noun

Ex : Button , System etc..

Interface - Should start with Uppercase letter , adjective

Ex : Runnable , Printable etc..

Method - Should start with lowercase letter , verb

If multiple names are there it should start with small next Letter

Upper case

Ex : draw() , actionPerformed()

Variable - Should start with lowercase letter , no - &,$,\_

avoid single letter variables like x , y.

Ex : id , name

Package - Should start with lowercase , separated by dots.

Ex: com.javatpoint

Constant - Uppercase letter such as RED,YELLOW

Separated by \_

Ex: MIN\_AGE

1. Best Practices

General Rules - Keep it simple always leave code cleaner than you started with

Easier to understand - Be consistent , Do all things in a similar way

Use variables which are self explanatory

Don’t write methods which depend on other methods to

work properly

Use whitespaces and Indentations

Names - Choose meaningful , pronounceable and searchable names

Replace magic numbers with Constants which describe

Avoid Intprogram etc..

Comments - Explain why, not how , in your code

Don’t comment out code remove it

Clarification , warning

Objects and dataStructures - Do one thing , Hide internal Structure

Code smells :

Rigidity - should not be there , no domino effect ( should not be rigid

easier to change )

Fragility - software should not break due to single change

Immobility - code should be replaceable elsewhere

Opacity - hard to understand.

**SOLID design principles**

What ?

Subset of principles promoted by **robert c martin** . The solid acronym was first i

introduced by **michael feathers.**

Why ?

Design principles that enables to manage most of software design problems

**Understandable , flexible , maintainable.**

S - Single responsibility principle

0 - open / close principle

L - liskov substitution principle

I - Interface segregation principle

D - Dependency inversion principle

**S - Single responsibility principle**

**“ A class should have only one reason to change “ - Robert c martin**

Every module or class should have responsibility over a single part of functionality.

**O - open/close principle**

**“ Software entities should be open for extension , but closed for modification“**

The design and writing of the code should be in such a way that new functionality should be added with minimum number of changes

**L - liskov substitution principles**

Barbara Liskov - **“ Objects in a program should be replaceable with instances of their subtypes without altering the correctness of the program”**

If there is a Base class , then reference to the Base class can be replaced with derived class without affecting the functionality of the program module.

**I - Interface segregation principle**

**“Many client specific interfaces are better than one general-purpose interface “**

Don’t enforce clients to implement interfaces that they do not use , one big interface can be broken into smaller interfaces

**D - Dependency inversion principle**

**“ depend on abstractions , not concretions “**

Abstractions should not depend on the details where as the details should depend on

Abstractions

High level modules should not depend on low level modules.

If we Don’t use :

* End up with tight coupling
* Unknown issues
* Code which is not testable
* Duplication of code
* New bug fixes raise while fixing another bug.

If we use :

* Reduction in complexity of the code
* Increase readability , extensibility and maintenance
* Reduce error and implement reusability
* Better testability
* Reduce tight coupling.