

بسم الله الرحمن الرحيم
السلام عليكم ورحمة الله وبركاته

عن أمير المؤمنين أبي حفص عمر بن الخطاب رضي الله عنه ، قال : سمعت رسول الله صلى الله عليه وسلم يقول :
(إنما الأعمال بالنيات ، وإنما لكل امرئ ما نوى ، فمن كانت هجرته إلى الله ورسوله فهجرته إلى الله ورسوله ،
ومن كانت هجرته لدنيا يصيبها أو امرأة ينكحها فهجرته إلى ما هاجر إليه).

رواه إماما المحدثين أبو عبد الله محمد بن إسماعيل بن إبراهيم بن المغيرة بن بردزبه البخاري
الجعفي ، وأبو الحسين مسلم بن الحجاج بن مسلم القشيري النيسابوري رضي الله عنهما في
صحيحيهما اللذين هما أصح الكتب المصنفة.

Today - in shaa ALLAH- I'd like to talk with you about a very important topic , it is even
most important than Desgin Patterns itself. (at least from my point of view)

“ Desgin Principles “

Yes, Desgin Principles.

Definition: “Desgin principles are guidelines that you can follow to make your desgin
better.”

as the definition illustrate, these are guidelines that you follow to make you desgin more
flexiable and more better, this is what it all about.

It is easy to study desgin patterns because these patterns some what are templates that
you know and apply, but it is very important to understant Desgin principles because
these principles are not template applied rather by understanding them well you can
understand desgin patters well and know from where these pattern born to life.

Patterns themselves didn't being patterns unless they follow theses desgin principles.
==> so, Desgin principles lead you to **discover** patterns.

lets go through them :

1- Determine parties that change in you system and encapsulte them :

-Change

Question :What is the thing that all systems in real life should encounter ????

try to answer this question yourself

.....

.....

Answer : CHANGE, yes change.

any System on earth should change by time, either when the user requirements change or when the DBMS change or .. or

so any system should change by time.

So, try to detect parties that will change in the future and encapsulate them.

-Encapsulation and Simplification.

Note : you know what ENCAPSULATE means ????????

you should know what encapsulate means as it is one of the basics of Object-Oriented (Abstraction, encapsulation, polymorphism and inheritance).
(tip : encapsulate related to Information Hiding)

Note :

Do you know the difference between Encapsulation and Simplifying ???

Encapsulation : hide the details of the system (or class) and preventing others from access.

Simplifying : hide details of the system (or class) while allowing others (advanced users) to access these details.

ex :

MS Windows OS (Operating System نظام تشغيل) encapsulate the OS tasks (that mean, it hide the details of the OS and preventing me from knowing it - so i hate it hahaha...)

GNU/Linux OS : simplifying the OS tasks (that means it hide the details of the OS but at the same time show these details to whom are concerned)

Now the difference become apparent (especially for open-source community users -like me)

- Encapsulate what varies

Back to the principle, You have to encapsulate what change, but how you can determine what part is changing?? from experience in designing systems.

- Example

Lets take an example :

```
class Company {
    String EmployeeName;
    Date EmployeeBDate;
    Date EmployeeHireDate;

    String CompanyName;
    Icon logo;
    // other declarations and methods ...
    // ...
    // ...
    //
}
```

this is a class for the company, after a while, the company needed to save a new information about its employees (say ComingDate).

and after a while again it needed to store a new thing about its employees (say Salary)

mmm, what varies in this class ??????????

yes, that is THE employee data.

Go ahead and separate it and encapsulate it.

====>>

<pre>class Company { Employee e; String CompanyName; Icon logo; // other declarations and methods ... // ... // ... // }</pre>	<pre>class Employee { String name; Date bDate; Date hDate; String getName(){ return name; } // other declarations .. // //..... // ... }</pre>
--	--

2- Program to interface not an implementation :

This is second principle, mm, what about it??

This principle advises you to keep your programming to interfaces (i.e. java interfaces and abstract classes) not to implementations (i.e. concrete classes)

- Introduction

ex. if you have two classes Cat and Dog :

```
class Cat {  
    //  
    // declarations  
    //  
    public void makeSound () {  
        naw();  
    }  
  
    public void naw(){  
        System.out.println ("naw..naw.. naw ");  
    }  
}
```

```
class Dog {  
    // declarations  
    //
```

```

    //
    public void makeSound (){
        haw();
    }
    public void haw(){
        System.out.println ("haw waw waw ");
    }
}

```

and here is the class that uses them :

```

class Driver {
    public static void main (){
        Cat c = new Cat();
        Dog d = new Dog();

        c.makeSound();
        d.makeSound();
    }
}

```

let us follow this principle :

we will add a new class “ Animal “ that will be super type for both Cat and Dog with this definition :

```

abstract class Animal {
    abstract void makeSound();
    // other shared methods..
    // ....
}

```

and change both of Cat and Dog to make them extends it.

so Driver will be :

```

class Driver {
    public static void main (){
        Animal a1 = new Cat();
        Animal a2 = new Dog();

        a1.makeSound();
        a2.makeSound();
    }
}

```

- Usage

you may see this is great, but you may say “ mm, great, but tell me just one reason to make me change my design to this new design ??”

This class can answer you :

```

class Zoo {
    public void acceptAnimal ( Animal a ){

```

```

        a.makeSound();
    }
}

```

could you tell me what is the runtime type of a ??
the question in other words: what will a.makeSound() print??

of course you can not tell me exactly what it will print because it will print upon the runtime type of the object passes as a parameter, for ex. if we pass a Dog, it will print “haw waw waw “ and if you pass Cat it will print “naw naw naw “ and this is the polymorphism.

-polymorphism

polymorphism is an object-oriented concept that means that the same function (method) can perform in many shapes (poly means many and morphism means shapes), and this what we have seen; the function makeSound() performed many actions depending on the runtime type of the object passed.

3- favor composition over inheritance :

this is the third and last principle that we will talk about today.

I'll not talk more about this principle as it is related to the Decorator pattern, so we only will take an overview about it.

all of us knows what inheritance is and most of us knows what composition is and know that this do a function and the other do another function.

yes, but via the decorator pattern the composition can act exactly like inheritance; extending types.

why favor composition over inheritance??

because : inheritance as we know it allow extending types at compile-time but composition allow it at runtime, so it allows more flexibility.

-Compile-time extension Vs Runtime extension

compile-time extension as we all know like :

```

class a {
}

```

```

class b extends a {
}

```

when we compile them we know that b extends a.

but runtime extension allow us to make this process at runtime (and this is the decorator pattern that we will illustrate later, just keep in mind that run-time extension through composition is better than compile-time extension through inheritance.