**Some Philosophy of Object-Oriented Design**

**OOP’s Philosophy:** Before the object-oriented programming got introduced there was procedure-oriented programming style then the question arises that **why we need** **OOP’s**?

Why OOP’s is better than Procedural ?

We will discuss the important pillar of OOPS(AEIP), interfaces, cohesion and coupling.

**Encapsulation**: Facilitates us with two things.

1. This is the mechanism for restricting direct access of the object component like variables.

Variables

Code

Direct Access -------------------------------------------> -

**Restrict Direct Access**

1. It’s language construct that facilitates the bundling of data with methods / other functions operating on that data.

Data

Methods

Why it’s so important?

Like every object in real world comes with encapsulation every object behavior changes with the data. So, it’s important to keep the data and behavior as close as possible.

In programming we achieve using objects and classes. This is important because when we create object from the class with different attributes the behavior changes.

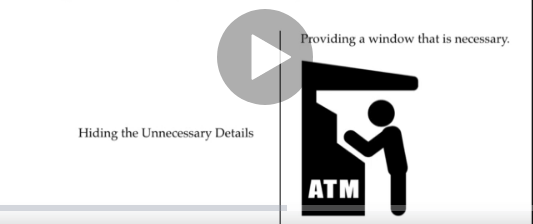
But this change is limited to that object and so all object has different behavior within same class based on the supplied data.

Example: Suppose there is a doctor who looks after many patients. Every patient can have different dieses and based on the symptom doctor prescribe the medicine so, the patients become the class and this class can have different dieses. Based on the dieses the patient behaves and exposes the symptom but what if a doctor messes up with different patient and if the data of patient get mixed. You can imagine what would happen.

Here are some of the advantages of binding a methods and variables together means using encapsulation:

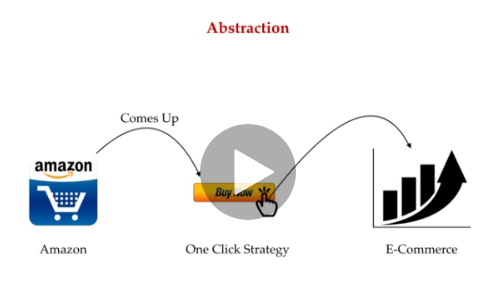
1. Encapsulated code is more flexible and easier to change with the new requirements.
2. Encapsulation in java makes unit testing very easy.
3. Encapsulation in java control who can access what
4. Encapsulation reduces coupling of modules and increases the cohesion inside a module because all pieces of one thing is encapsulated in one place.
5. Encapsulation allows to change one part of the code without affecting other parts of code.

**Abstraction:** Abstraction is most important construct that object-oriented programming provides. **Abstraction** is about hiding unnecessary details and providing a window OR expose that is necessary to the consumers of your classes or methods.

****

**There are two theory for abstraction**

1. **Higher** the abstraction you achieve in your code , **Higher** the excellence and simplicity you achieve.
2. Abstraction works well with trust.



In above example: In Amazon – came with one click strategy and that was prove as game changer for e-commerce business what amazon is trying ? 🡪 To answer this – Amazon is trying to hide all the unnecessary details and pan form the buyer and facilitates the buying in just one click.

All of these stuffs were hidden and done on backend. And hence higher abstraction , higher excellence and simplicity.

Same way in java we hide all the unnecessary functions by keeping them **private** and expose the required function only by making them public.

Now second theory is : **Abstraction works well with trust:** In JDK you call so many methods without knowing much details about them because you trust that whatever is implemented by java is given to you will work well. So, we trust JDK that is why we won’t bother about the details that you just go and use same with everything around you.

Like : Buying a brand Microsoft / Apple PC without worrying about the quality and other things.

**Inheritance:** Java provide the support for inheritance by extending a class and also @override the definition of methods.

Example: From basic mobile phone to smart mobile phone everything is inherited like basic phone has features of **calling**, **messaging** and same features are available in smart phone with additional features like music, camera etc.

**Polymorphism:** Polymorphism in java comes along with other OOP’s concept like Abstraction, Encapsulation and inheritance. It inherited / came from the Greek word **Poly which means many and morphism means forms.**

**Polymorphic 🡪 is something which can take many forms.**

**In polymorphism 🡪 Method Overloading** is a deciding factor which method to call at runtime.

**In polymorphism 🡪 Method Overriding** is a deciding factor which method call at runtime.

Polymorphism concept is so powerful that you can imagine a situation when based on certain condition methods calls changes at runtime.

Think in a reverse for a while when polymorphism was not there ! was it possible to write so concise JDK library? We had to give different name to different method. How it could have look like. **Object class method like hashCode(), equals() are overridden by the classes to provide their own functionality and that is very useful and necessary.**

**Marker interfaces like runnable cloneable gives us chance to implementation class to provide the functionality and jdk calls at runtime.**

Functional programming or functional interfaces in java , function, consumer and supplier all uses polymorphism.

Many of specification written in these states like JPA, FLOW API and oracle only provide interfaces at runtime, the other vendor supplies the implementation, and this is only possible only through the polymorphism.

**Interfaces:** In early time programmer kept thinking that why even interfaces are required when there is no implementation provided? BUT interfaces are the one which provide **highest level of abstraction**

Java achieve two thing using interfaces and combining polymorphism. First one is **interfaces just** can provide the higher level of abstraction without providing implementation and anyone can provide the implementation later on. So, what is the benefit of doing that ? imagine that oracle publisher publishes a set of interfaces to communicate to the database like JPA (Java Persistent API ) and vendor like JBOSS (hibernate , ORM ) or any other vendor supplier implementation. So, what is the benefit of doing that ? think in this way that everyone has to use the same API names , method required to remember, and same set of parameter and method will be implemented by different vendor that is why a good designer program for interfaces first and then the implementation.

And second one is that the interface provides the contracts. Like if you have used Runnable, Cloneable and Marker interfaces or any other interfaces the developer ask for you to implement the interface so that there has to be implementation and the caller function or jdk calls the definition t runtime.

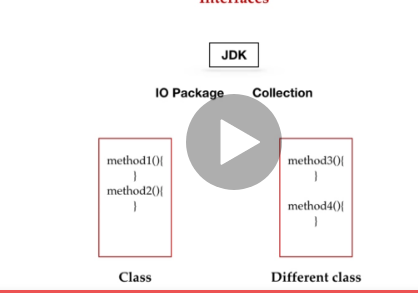
So, interfaces are heavily used to provide the contract. You supply me this definition and I will cover up the rest implementation for you.

**Cohesion:** Cohesion says keep the same things at same place. Like imagine your home you always keep the similar things at the same place so that later when you tried to find them it’s easy for you to find them.

For example: You keeps the books in the library or study room and the kitchen related stuff in the kitchen so **Similarly JDK also keep the classes of the same type at the same place. Like IO related classes in IO package, and Collection related classes in collection all together.**

**So, Highly cohesive classes and function always a better programming design. At class level you keep all the related function in one class and if there are different function between another class.**

**High cohesive is always is better design**



Coupling: Coupling means connecting the parts of the machinery. The best java programmer always thrives to achieve the loose coupling. Why ? Let’s take a real-world example : A car manufacturer building a car and keep the parts highly coupled you can imagine at later point of time the cars break and some parts of car needs to be replaced. The replacement or repair depends on how other parts are coupled with parts that needs to be replaced.

And same with the code. If the code is loosely coupled the bug fixes code improvements optimizations will be easy. You just need to take care of that particular parts or other parts of the codebase remains unaffected.

In big java application codebases is divided in layers to make it loosely coupled. And just to support this principle there are many architecture tiles and principles and paradigm getting popular. Like Microservices.