**OOPs in Java- Encapsulation, Inheritance, Polymorphism, Abstraction**

**Object Oriented Approach : An Introduction**

One of the most fundamental concept of [**OOPs**](http://beginnersbook.com/2013/04/oops-concepts/) is **Abstraction**. Abstraction is a powerful methodology to manage complex systems. Abstraction is managed by well-defined objects and their hierarchical classification. For example a car in itself is a well-defined object, which is composed of several other smaller objects like a gearing system, steering mechanism, engine, which are again have their own subsystems. But for humans car is a one single object, which can be managed by the help of its subsystems, even if their inner details are unknown.

Java is an object oriented language because it provides the features to implement an object oriented model. These features includes **encapsulation**, **inheritance** and**polymorphism**.

**OOPS** is about developing an application around its data, i.e. objects which provides the access to their properties and the possible operations in their own way.

**Principles of OOPs**

**1) Encapsulation**

Below is a real-life example of encapsulation. For the example program and more details on this concept refer **[encapsuation in java with example](http://beginnersbook.com/2013/05/encapsulation-in-java/" \o "encapsulation" \t "_blank)**.

Encapsulation is:

* Binding the data with the code that manipulates it.
* It keeps the data and the code safe from external interference

Looking at the example of a power steering mechanism of a car. Power steering of a car is a complex system, which internally have lots of components tightly coupled together, they work synchronously to turn the car in the desired direction. It even controls the power delivered by the engine to the steering wheel. But to the external world there is only one interface is available and rest of the complexity is hidden. Moreover, the steering unit in itself is complete and independent. It does not affect the functioning of any other mechanism.

Similarly, same concept of encapsulation can be applied to code. Encapsulated code should have following characteristics:

* Everyone knows how to access it.
* Can be easily used regardless of implementation details.
* There shouldn’t any side effects of the code, to the rest of the application.

The idea of encapsulation is to keep classes separated and prevent them from having tightly coupled with each other.

A live example of encapsulation is the class of java.util.Hashtable. User only knows that he can store data in the form of key/value pair in a Hashtable and that he can retrieve that data in the various ways. But the actual implementation like, how and where this data is actually stored, is hidden from the user. User can simply use Hashtable wherever he wants to store Key/Value pairs without bothering about its implementation.

**2) Inheritance**

Below is a theoretical explanation of inheritance with real-life examples. For detailed explanation on this topic refer [**inheritance with examples**](http://beginnersbook.com/2013/03/inheritance-in-java/) and[**types of inheritance in java**](http://beginnersbook.com/2013/05/java-inheritance-types/).

* Inheritance is the mechanism by which an object acquires the some/all properties of another object.
* It supports the concept of hierarchical classification.

For example: Car is a classification of Four Wheeler. Here Car acquires the properties of a four-wheeler. Other classifications could be a jeep, tempo, van etc. Four Wheeler defines a class of vehicles that have four wheels, and specific range of engine power, load carrying capacity etc. Car (termed as a sub-class) acquires these properties from Four Wheeler (termed as a super-class), and has some specific properties, which are different from other classifications of Four Wheeler, such as luxury, comfort, shape, size, usage etc.

A car can have further classification such as an open car, small car, big car etc, which will acquire the properties from both Four Wheeler and Car, but will still have some specific properties. This way the level of hierarchy can be extended to any level.

Java Swing and Awt classes represent best examples for inheritance.

**3) Polymorphism**

Below is a real-life example of polymorphism. For the example program and more details on this OOP concept refer [**polymorphism in java**](http://beginnersbook.com/2013/03/polymorphism-in-java/) and [**runtime & compile time polymorphism**](http://beginnersbook.com/2013/04/runtime-compile-time-polymorphism/).

* Polymorphism means to process objects differently based on their data type.
* In other words it means, one method with multiple implementation, for a certain class of action. And which implementation to be used is decided at runtime depending upon the situation (i.e., data type of the object)
* This can be implemented by designing a generic interface, which provides generic methods for a certain class of action and there can be multiple classes, which provides the implementation of these generic methods.

Lets us look at same example of a car. A car have a gear transmission system. It has four front gears and one backward gear. When the engine is accelerated then depending upon which gear is engaged different amount power and movement is delivered to the car.

Polymorphism could be static and dynamic both. Overloading is static polymorphism while, overriding is dynamic polymorphism.

* **Overloading** in simple words means two methods having same method name but takes different input parameters. This called static because, which method to be invoked will be decided at the time of compilation
* **Overriding** means a derived class is implementing a method of its super class.