**Oops real life Examples**

**Abstraction**

One of the most fundamental concepts of OOPs is **Abstraction**. Abstraction is a process where you show only “relevant” data and “hide” unnecessary details of an object from the user. For example, when you login to your Amazon account online, you enter your user\_id and password and press login, what happens when you press login, how the input data sent to amazon server, how it gets verified is all abstracted away from the you.

Another example of abstraction: 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 human’s car is a one single object, which can be managed by the help of its subsystems, even if their inner details are unknown.

### Encapsulation

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.

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

An 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 Hash table 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 Hash table wherever he wants to store Key/Value pairs without bothering about its implementation.

### Inheritance

* 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 four-wheeler vehicle so assume that we have a class Four-wheeler and a sub class of it named Car. Her Car acquires the properties of a class 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, 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.

### Polymorphism

* Polymorphism means to process objects differently based on their data type.
* In other words, it means, one method with multiple implementations, 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.

Let’s us look at same example of a car. A car has 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. The action is same applying gear but based on the type of gear the action behaves differently or you can say that it shows many forms (polymorphism means many forms)

Polymorphism could be static and dynamic both. [**Method Overloading**](https://beginnersbook.com/2013/05/method-overloading/) is static polymorphism while, [**Method overriding**](https://beginnersbook.com/2014/01/method-overriding-in-java-with-example/) is dynamic polymorphism.

* **Overloading** in simple words means more than one method having the same method name that behaves differently based on the arguments passed while calling the method. This called static because, which method to be invoked is decided at the time of compilation
* **Overriding** means a derived class is implementing a method of its super class. The call to overridden method is resolved at runtime, thus called runtime polymorphism