# CONCEPT OF POLYMORPHISM IN OBJECT ORIENTED PROGRAMMING

Anoop & Sunil Rai
*Department Of Electronics And Computer*
*Dronacharya College Of Engineering, Gurgaon, Haryana*

***Abstract-*** **The objective of this paper is to comprehensive study related to concept of Polymorphism in Object Oriented programming i.e, OOPS. Polymorphism is the ability to exist in different forms. OOP allows objects belonging to different data types to respond to calls of methods of the same name, each one according to an appropriate type-specific behavior. In object-oriented programming, polymorphism refers to a programming language's ability to process objects differently depending on their data type or class. We will now attempt to describe more precisely the concept of polymorphism.**

***Index Terms-*** **OOPS; class; polymorphism.**

## I. INTRODUCTION

The word Polymorphism means of many forms. In programming this word is meant to reuse the single code multiple times. In object oriented programming it's a big question that why the Polymorphism is done, what is the purpose of it in our code?

There are lots of people who don't even know the purpose and usage of Polymorphism. Polymorphism is the 3rd main pillar of OOP without it the object oriented programming is incomplete. Lets go in the depth of Polymorphism.

## II. WHY POLYMORPHISM IS DONE IN OOP?

It's obvious that when we do inheritance between two classes, all the methods and properties of the first class are derived to the other class so that this becomes the child class which adobes all the functionality of base class. It can also possesses its own separate methods.

But there is a big problem in inheriting the second class to the first class as it adobes all the methods same as the base class has, which means that after inheritance both (base class& child class) have the methods of same name and same body as shown in this example:

**____Base Class___**

```
public class fish
{
 public void eat()
  {
      console.writeline("fish eat");
  }
 public void swim()
  {
       console.writeline("fish swim");
  }
 }
```

**_____Derived Class_____**

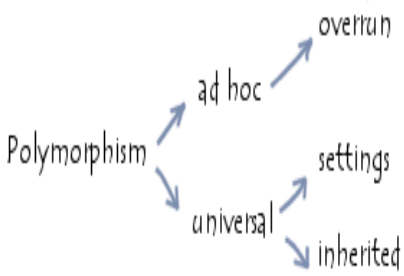
```
Class Dolphen:fish
{

 public void eat()
  {
      console.writeline("fish eat");
  }
 public void swim()
  {
       console.writeline("fish swim");
  }
}
```

## III. TYPES OF POLYMORPHISM

In general there are three types of polymorphism:

- Overloading polymorphism
- Parametric polymorphism
- Inclusion polymorphism




A. Ad hoc polymorphism

Chris Strachey chose the term ad hoc polymorphism to refer to polymorphic functions which can be applied to arguments of different types, but which behave differently depending on the type of the argument to which they are applied (also known as function overloading or operator overloading). The term "ad hoc" in this context is not intended to be pejorative; it refers simply to the fact that this type of polymorphism is not a fundamental feature of the type system. In the example below, the Add functions seem to work generically over various types when looking at the invocations, but are considered to be two entirely distinct functions by the compiler for all intents and purposes:

```
program Adhoc;

function Add( x, y : Integer ) : Integer;
begin
    Add := x + y
end;

function Add( s, t : String ) : String;
begin
    Add := Concat( s, t )
end;

begin
    Writeln(Add(1, 2));
    Writeln(Add('Hello, ', 'World!'));
end.
```

B. Parametric polymorphism

Parametric polymorphism allows a function or a data type to be written generically, so that it can handle values identically without depending on their type.[6] Parametric polymorphism is a way to make a language more expressive, while still maintaining full static type-safety.

The concept of parametric polymorphism applies to both data types and functions. A function that can evaluate to or be applied to values of different types is known as a polymorphic function. A data type that can appear to be of a generalized type (e.g., a list with elements of arbitrary type) is designated polymorphic data type like the generalized type from which such specializations are made.

Parametric polymorphism is ubiquitous in functional programming, where it is often simply referred to as "polymorphism". following example shows a parameterized list data type and two parametrically polymorphic functions on them:

```
data List a = Nil | Cons a (List a)

length :: List a -> Integer
length Nil         = 0
length (Cons x xs) = 1 + length xs

map :: (a -> b) -> List a -> List b
map f Nil         = Nil
map f (Cons x xs) = Cons (f x) (map f xs)
```

C. Overloading Polymorphism

Overloading polymorphism is where functions of the same name exist, with similar functionality, in classes which are completely independent of each other (these do not have to be children of the object class). For example, the complex class, the image class and the link class may each have the function "display". This means that we do not need to worry about what type of object we are dealing with if all we want to do is display it on the screen.

Overloading polymorphism therefore allows us to define operators whose behavior will vary depending on the parameters that are applied to them. Therefore it is possible, for example, to add the + operator and make it behave differently according to whether it refers to an operation between two integers (addition) or between two character strings (concatenation).

## IV. APPLICATION

Polymorphism is the foundation of Object Oriented Programming. It means that one object can behave as

another project. So how does on object can become other, it's possible through following

1. Inheritance
2. Overriding/Implementing parent Class behavior
3. Runtime Object binding

One of the main advantages of it is switch implementations. Let's say you are coding an application which needs to talk to a database. And you happen to define a class which does this database operation for you and its expected to do certain operations such as Add, Delete, Modify. You know that database can be implemented in many ways; it could be talking to file system or a RDBM server such as MySQL etc. So you as programmer would define an interface that you could use, such as...

```
public interface DBOperation {
   public void addEmployee(Employee
newEmployee);
   public void modifyEmployee(int id, Employee
newInfo);
   public void deleteEmployee(int id);
}
```

You can use polymorphism concept in many places, one particle example would be: lets you are writing image decoder, and you need to support the whole bunch of images such as jpg, tif, png etc. So your application will define an interface and work on it directly. And you would have some runtime binding of various implementations for each of jpg, tif, pgn etc.

One other important use is, if you are using java, most of the time you would work on List interface, so that you can use Array List today or some other interface as your application grows or its needs change.

## V. CONCLUSION

As we discussed above that polymorphism has its importance in object oriented programming, we can say that as time passes its importance will also increase rapidly.

## REFERENCES


1. Faizan Ahmed, www.DotNetFunda.com
2. www.CareerRide.com
3. Gurunatha Dogi, www.onlinebuff.com
4. www.wikipedia.com
5. www.health.kioskea.net
6. C. Strachey - *Fundamental Concepts in Programming Languages*
7. Cory Janssen, www.techopedia.com
8. Pierce, B. C. 2002 *Types and Programming Languages.* MIT Press.
9. www.techtarget.com
10. www.stackoverflow.com