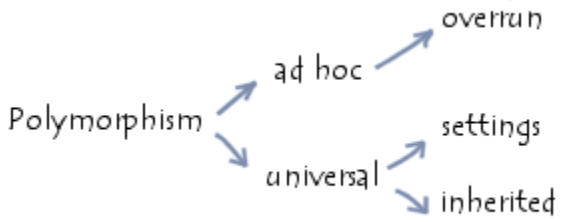
OOP - Polymorphism

Definition of polymorphism

The word *polymorphism* comes from Greek and means *having several different forms*. This is one of the essential concepts of object-oriented programming. Where inheritance is related to classes and (their hierarchy), polymorphism is related to object methods.

In general there are three types of polymorphism:

- Overloading polymorphism
- Parametric polymorphism (also called *template polymorphism*)
- Inclusion polymorphism (also called *redefinition* or *overriding*)



We will now attempt to describe more precisely the above types, but be warned, many people get confused when trying to understand the differences between the three types.

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).

Parametric polymorphism

Parametric polymorphism is the ability to define several functions using the same name, but using different parameters (name and/or type). *Parametric* polymorphism automatically selects the correct method to be adopted according to the type of data passed in the parameter.

We can therefore define several *addition()* methods using the same name which calculates a sum of values.

- The *int addition(int,int)* method would return the sum of two integers.
- The *float addition(float, float)* would return the sum of two floats.
- The *char addition(char, char)* would result as the sum of two characters as defined by the author.
- etc.

A *signature* is the name and type (static) given to the arguments of a function. Therefore it is a method's signature which determines what is called on.

Inclusion polymorphism

The ability to redefine a method in classes that are inherited from a base class is called **specialization**. One can therefore call on an object's method without having to know its intrinsic type: this is **inclusion polymorphism**. This makes it possible to disregard the specialized class details of an object family, by masking them with a common interface (this being the basic class).

Imagine a game of chess, with the objects *king*, *queen*, *bishop*, *knight*, *rook* and *pawn*, each inheriting the *piece* object.

The *movement* method could, using inclusion polymorphism, make the corresponding move according to the object class that is called on. This therefore allows the program to perform *piece.movement* without having to be concerned with each piece's class.

Source: http://en.kioskea.net/contents/424-oop-polymorphism