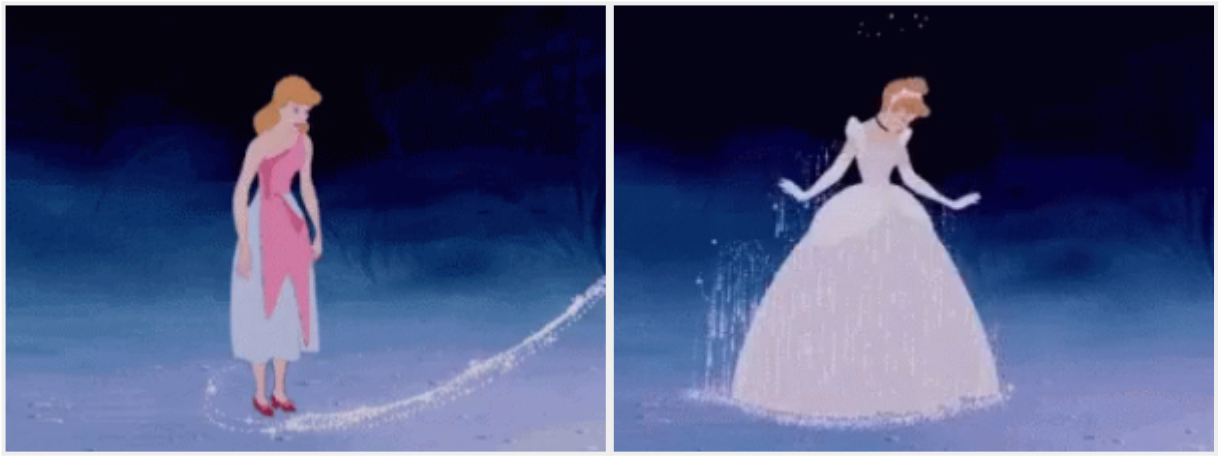
POLYMORPHISM IN PROGRAMMING

Polymorphism is defined as an object that can take on various forms. This article will look at polymorphisms and how they're used in programming.

What is a polymorphism?

At its base level, a polymorphism is part of mathematic type theory. In computer science, a polymorphic object is an object that is capable of taking on multiple forms. The kind of polymorphism the object undergoes depends on *when* the object takes its form and *what part* of the object is transforming.



When the object transforms:

1. Compile-time
2. Dynamic

What does the transforming:

1. Method
2. Object

Polymorphism in programming

> *"In programming languages and type theory, polymorphism is the provision of a single interface to entities of different types, or the use of a single symbol to represent multiple different types."*

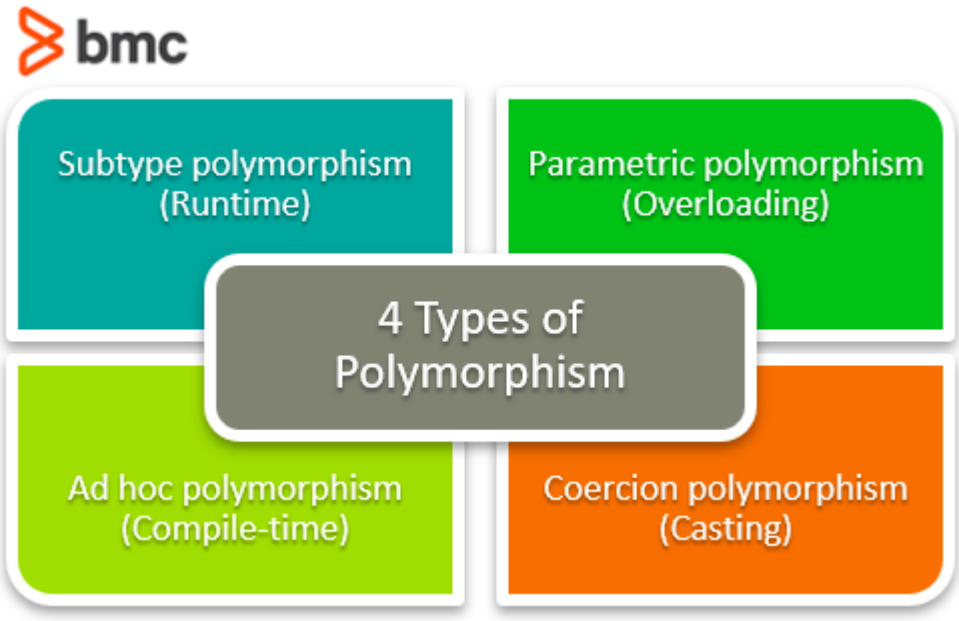
Polymorphism is essential to object-oriented programming (OOP). Objects are defined as classes. They can have properties and methods. For example, we could create an object defined as class Car. This car could have:

- **Properties**, such as: color, make, model, current speed
- **Methods**, which are functions invoked by the class, such as: go, stop, park, turn left, turn right

For the above Car object to be polymorphic, its properties and methods can be called using the same class name "Car" but invoke different types depending on how it is used in the code.

Types of Polymorphism

There are four main types of polymorphism:



Let's look at each.

Subtype polymorphism (Runtime)

Subtype polymorphism is the most common kind of polymorphism. It is usually the one referenced when someone says, "The object is polymorphic."

A subtype polymorphism uses one class name to reference multiple kinds of subtypes at once. In the car example above, we create the object class, "Car", and define multiple subtypes of it to be:

Ford, Chevrolet, Kia, Nissan, Volkswagen, BMW, Audi, and Tesla. Each car has its property color.

All subtypes can be referenced interchangeably by using class Car.

Every car has a property, color, and now we can get the color from each subtype of the Car class.

The polymorphism takes place at runtime. We can write a function to fetch the color of any one of the car subtypes. The function may look like this (not written in any specific programming language):

```
getColor(BMW)
        → returns red
getColor(Audi)
        → returns blue
getColor(Kia)
        → returns yellow
```

Parametric polymorphism (Overloading)

A parametric polymorphism specifically provides a way to use one function (the same code) to interact with multiple types.

An example would be a list of types. A parametric polymorphism could remove, add, or replace elements of this list regardless of the element's type.

The code, which could be a parametric polymorphism function written in Python:

```
for (element in list):
        list.remove(element)
```

Ad hoc polymorphism (Compile-time)

For ad hoc polymorphism, functions with the same name act differently for different types.

Python is dynamically typed (it does not require the type to be specified). The Python addition function is an ad hoc polymorphism because it is a single function of the same name, "+", that works on multiple types.

Both (Python):

```
3 + 4
"Foo" + "bar"
```

Return a value of their corresponding types, int and String, respectively.

```
→ 7
→ "Foobar"
```

For values of type int, the addition function adds the two values together, so 3+4 returns 7. For values of type String, the addition function concatenates the two strings, so "Foo" + "bar" returns, "Foobar".

Coercion polymorphism (Casting)

Coercion polymorphism is the direct transformation of one type into another. It happens when one

type gets cast into another type. Before, polymorphism occurred through interacting with different types through the object class or the functions. An object's type could be selected when the program was run. A single function could work with different types.

A simple example is transforming number values from ints, to doubles, to floats, and vice versa. Depending on the programming language, either the type can be defined on the variable, or sometimes there exists a method on the types to convert their value to another type.

Python will convert types like this:

```
int(43.2) #Converts a double to an int
float(45) #Converts an int to a float
```

Java might convert the type simply by defining the type:

```
int num = 23;
double dnum = num;
```

or

```
double dnum = Double.valueOf(inum);
```

Additional resources

For related reading, explore these resources:

- BMC DevOps Blog
- Asynchronous Programming: A Beginner's Guide
- Abstraction Layers in Programming: An Overview
- What Is Systems Programming?
- Python vs Java: Why Python is Becoming More Popular than Java