# THE CHARACTERISTICS OF AN OBJECT ORIENTATED PARADIGM

BY LUKE BRUNI

## POLYMORPHISM

In Object-Oriented Programming, polymorphism represents the same method but with many different implementations. It allows for there to be different forms of a variable, function or object.
In the case of a variable, it can possess multiple forms to which can be chosen and executed by the program. For instance, this function could be able to store integers or strings and can be called accordingly by the program if given a way to differentiate the form being handled.

## ENCAPSULATION

In object-oriented programming, encapsulation is used for the process of grouping or wrapping either data or functions to perform actions in one single unit, known as a class. This class basically encloses the operations and states inside to make up a particular object, keeping the code safe from potential intrusion.

Modifiers are used within the class make sure that there is a level of restriction to accessing the encapsulated data and not cause much interference.
These modifiers are:

- Private – Where the data stored here can only be accessed to the current class, meaning that no other class can access it.
- Protected – The data here can be accessible to the class and subclasses.
- Public – The data can be accessed from any part of the code

## OBJECTS, SUB OBJECTS AND CONTAINERS

### Objects

Objects are instances of a class, created by instantiation and assigning to a variable. Not only can an instance come from a class, but it may also come from another object as part of prototypal inheritance.

### Subobjects

Should an object come from an object, then it is known as a subobject. These are prominent in arrays and base class objects and works near enough like a subclass, in which the subobject derives from the parent object.

### Containers

A container is an object that's purpose is to hold the elements given and managing the space for said elements. It can allow for easy implementation of data structures like lists and queues, as well as arrays. The containers are basically objects, containing many of the features used within the containers

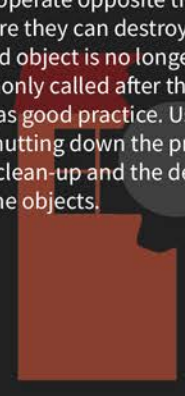
## CONSTRUCTORS & DESTRUCTORS

### Constructors

A constructor is a method in Object-Oriented Programming used for initializing an object, setting it up for use and creation of a new object. These are used to initialize the objects and put them in force. The constructor allows the user to input the values in the predefined class fields, where for instance, you can have a set of values and can easily write out an instance by calling the constructor.
The constructor involves memory allocation and initialization as part of effective start-ups of the program.


### Destructors

These methods operate opposite the constructor where they can destroy the object created once said object is no longer required. These are commonly called after the end of the constructor as good practice. Using these are vital when shutting down the program as it can help with clean-up and the deallocation of memory for the objects.



## ABSTRACT & CONCRETE CLASSES

### Abstract

An Abstract Class can be a method or variable of a class that has one or more abstracted methods. These classes are summarized into characteristics that reflect on the program's operation. These classes cannot have numerous instances from it and this helps the code from being used incorrectly. However, these classes require subclasses in order to define the attributes needed for creating instances.

### Concrete

These classes are derived from the abstract class. These classes, unlike the abstract class, we can create instances from the abstract class before it, implementing the abstract features into that instance.

## OBJECT ORIENTED INTERFACES

These interfaces essentially are a specific programming syntax where a class has certain properties implemented onto that object, essentially declaring the functions inside that given class. An example would be a car, a function is given to it for it to accelerate. What these interfaces do is give the object or class actions for it to function and these options.

```
package
{
    public interface Vehicle
    {
        // NO data VARIABLES are allowed in an interface
        // only function PROTOTYPES


        /**
         * Comments...
         * Anything that wants to be a "Vehicle" must, implement this function
         */
        function start_engine() : void;


    }
}
```

## BASE & DERIVED CLASSES

### Base

This class is the parent class and serves as the foundation for the other classes. The derived classes can reuse the code and functionality in the base class. For instance, you can have a class called 'Car' containing the functions that make up its operation and classes can be derived from this, sharing the operation for its own.

### Derived

These classes, as said earlier, are derived from the base class, taking the code and functionality of that class and using it to create its own unique variants. This is done by allowing to add or modify the existing properties from th base. For instance, from the car class, a sub class can be a 'make and 'model' of 'Car' with its own features and modified properties

## GENERICS AND TEMPLATES

### Generics

Generics are used as a way of using any sort of data types with a class or method and has the compiler generate code to help with that data type. It allows for maximizing resue, type safety and code performance as well as mixing it up by creating your own generics

### Templates

This is mainly a generic class used as a base for the rest of the code and are a staple in generic programming. Within object-oriented programming, some languages that use this have libraries comprosing of templates for programmers to modify, made up of pre-built classes and containers and can help with understanding code

## METHOD REDEFINITION, OVERRIDING & OVERLOADING

### Redefinition

This is mainly used within a descendant class and the compiler chooses the function not being the actual type of object. For example, two classes: 'Car' and 'Engine, where the 'Engine' inherits from 'Car' and you could assign a variable to that engine, which could be a 'V8'. If the variable is given the function to accelerate, it is called in the car class, meaning 'Car.accelerate()' is called when the 'Engine' should be.

### Overriding

Method Overriding is hwere a subclass has a method called and declared there when also it is declared in the main class, this method is overridden from the parent class and the subclass is called instead. For instance, going back to 'Car' and 'Engine', both could have the same parameters (Accelerate, brake, etc.) but calling them would only call the car class.

### Overloading

Method Overloading allows a class to have more than one method to be assigned the same name, granted if their functions are different. It means that you don't need to create different names for functions so long as they are unique to themselves. E.g. a car where you could have a function to return the mileage and the speed but both using the same name.

```
if (selection == "Insert" || selection
    gumball.insertQuarter(); //Function
}
```

## REFERENCES


*Topper Skills [online]. (2018). Available from: <http://www.topperskills.com/tutorials/oop/object-oriented-programming-polymorphism-concepts.html>. [Accessed 26 November 2018].*

*Margaret Rouse (2016). Abstract Class. [online]. TheServerSide. Available from: <https://www.theserverside.com/definition/abstract-class>. [Accessed 26 November 2018].*

*Bruno R. Preiss (1997). Data Structures and Algorithms with Object-Oriented Design Patterns in C [online]. Available from: <https://book.huihoo.com/data-structures-and-algorithms-with-object-oriented-design-patterns-in-c /html/page611.html>. [Accessed 26 November 2018].*

***baeldung (2018). Concrete Class in Java. [online]. Baeldung. Available from: <https://www.baeldung.com/java-concrete-class>. [Accessed 26 November 2018].***

*Techopedia [online]. (2018). Available from: <https://www.techopedia.com/definition/3780/derived-class>. [Accessed 26 November 2018].*

*Xiaoyun Yang (2018). How To Do Object Oriented Programming The Right Way. [online]. Codeburst.io. Available from: <https://codeburst.io/how-to-do-object-oriented-programming-the-right-way-1339c1a25286>. [Accessed 26 November 2018].*

*cppreference.com [online]. (2018). Available from: <https://en.cppreference.com/w/cpp/language/object>. [Accessed 26 November 2018].*

***J Steven Perry (2016). Unit 3: Object-oriented programming concepts and principles. [online]. IBM. Available from: <https://www.ibm.com/developerworks/library/j-perry-object-oriented-programming-concepts-and-principles/index.html>. [Accessed 26 November 2018].***

*cplusplus.com [online]. (2017). Available from: <http://www.cplusplus.com/reference/stl/>. [Accessed 26 November 2018].*

*Manoj Debnath (2016). Understanding C Containers in the C Standard Library. [online]. Codeguru. Available from: <https://www.codeguru.com/cpp/cpp/understanding-c-containers-in-the-c-standard-library.html>. [Accessed 26 November 2018].*

***Interfaces [online]. (n.d). Available from: <https://www.cs.utah.edu/~germain/PPS/Topics/interfaces.html>. [Accessed 28 November 2018].***

*Margaret Rouse (2005). Method. [online]. Whatis.com. Available from: <https://whatis.techtarget.com/definition/method>. [Accessed 29 November 2018].*

*Method Overriding in Java.(n.d) [online]. JavaTPoint. Available from: <https://www.javatpoint.com/method-overriding-in-java>. [Accessed 29 November 2018].*

*Overriding. (n.d) [online]. Techopedia. Available from: <https://www.techopedia.com/definition/24010/overriding>. [Accessed 29 November 2018].*