Definiciones básicas:

Técnicamente, el **objeto** es una unidad que realiza las tareas de un programa en tiempo de ejecución (es decir, se mantiene en la memoria del computador en un tiempo determinado), el cual comprende una colección de elementos de datos (llamados **atributos**), junto con las funciones asociadas utilizadas para operar sobre esos datos (llamados **métodos**). La potencia real de los objetos reside en varias características: los objetos pueden crear otros objetos y los objetos pueden comunicarse (interactuar) con otros objetos. Adicionalmente, el objeto puede considerarse como la representación o creación de algo que existe en la vida real, y que será usado para su implementación en la solución de un problema. Todos los objetos se consideran únicos, por lo que poseen una **identidad**. El conjunto de todos los atributos de un objeto se le conoce como **estado**, y el conjunto de todos los métodos de un objeto se le llama **comportamiento**.

Ejemplo de objeto: El carro de mi papá es una camioneta marca Kia modelo 1979 de placas ABC-123, y puede acelerar, frenar y girar.

carro_de_mi_papa: [tipo: camioneta, marca: Kia, modelo: 1979, placa: ABC-123, acelerar(), frenar(), girar()]

La **clase** es la construcción que representa la *generalización* de las características (definidas por el usuario) que determinan la creación de objetos: sus propiedades elaboradas como estructuras de datos (atributos) y sus operaciones asociadas con sus datos (métodos). Se puede considerar una clase como un modelo, molde o plantilla con el que se pueden crear los objetos (instancias de una clase). Cuando se define la clase, se puede considerar ésta como un tipo especial de datos creado por el programador (de manera parecida a un tipo int o un tipo String), y este tipo especial de datos permite crear objetos. Un objeto hace que la clase sea directamente utilizable. Un objeto creado se considera un ejemplo concreto (particular) de una clase, por ello al objeto creado se le llama instancia.

Ejemplo de clase: Cualquier carro cuenta con un tipo, una marca, un modelo, unas placas, una velocidad, y puede acelerar, frenar y girar, entre otros atributos y métodos.

Carro
tipo marca modelo placas velocidad
acelerar( ) frenar( ) girar( )

La **Programación Orientada a Objetos** (POO) es un método de implementación en el que los programas o aplicaciones se pueden crear y organizar como colecciones o conjuntos cooperativos de objetos que realizan pequeñas tareas, cada uno de los cuales representan una instancia de alguna clase. Esto permite que se puedan generar pequeñas secciones de programas que permitan una facilidad en su creación, mantenimiento y reutilización.

Características de las clases:

Abstracción: La abstracción en sí, se refiere al proceso mental por el cual, a partir de una situación o problema planteado, se captan elementos con características y comportamientos esenciales y particulares, permitiendo que estos elementos puedan ser *descritos* en forma general (qué tienen y qué hacen), y que también puedan ser *aislados* del resto de los otros elementos. Esta descripción y aislamiento permite que algunos elementos puedan ser agrupados para llegar a un nivel de generalización óptimo. Cuando se identifican estos elementos (u objetos), se dice que se está generando una abstracción de las clases que las comprenden.

Encapsulamiento. La encapsulación se refiere a dos características muy interesantes: la primera se le puede llamar con el nombre de **apropiación** o **encapsulamiento** propiamente dicho, y se refiere a que, debido a que se puede crear una clase con sus atributos y métodos, y se puede crear un objeto a partir de la instancia de una clase, el propio objeto tiene la capacidad de utilizar todas sus características internas para él mismo. Es decir, atributos y métodos se encapsulan en una sola entidad, y ella accede a sus propios datos y funciones. Suena obvio, sin embargo, esto lleva a la segunda característica: el objeto, además, tiene la capacidad de permitir o no que otros objetos externamente cuenten con sus características internas. Esta segunda característica se le conoce con el nombre de **ocultación**, y permite mantener un nivel de seguridad y protección de sus datos. Esta característica de seguridad implica que, si se requiere proteger la información de los atributos de una clase, es necesario pensar en crear métodos que permitan modificar o extraer la información que se encuentre en los atributos, y solamente se pueden acceder a los datos de los atributos o modificarlos a partir de los métodos creados para dicha clase. Es posible también hacer una ocultación de algunos de sus métodos, cuando se desee que estos métodos únicamente puedan ser accedidos a través de su propia clase, por lo que se puede hablar de ocultación de estado y de comportamiento.

Sobrecarga. La sobrecarga se refiere a la posibilidad de crear, en el mismo objeto, varios métodos con exactamente el mismo nombre, pero que tengan internamente una implementación *diferente* ya que en algunos casos es necesario desarrollar esta capacidad. Esto significa que un mismo método podría comportarse de manera diferente cuando sea requerido. El programa tiene la habilidad de seleccionar el método que necesita, *diferenciándolo de la cantidad y tipo de datos que se envíen como parámetros al método requerido*. Los parámetros son los valores que se envían a los métodos dentro del paréntesis final de su nombre.

Herencia. Se refiere a la capacidad de construir nuevas clases a partir de clases creadas previamente. Este concepto hace que las clases nuevas (que se pueden llamar “clases hijas” o “subclases” o “derivadas”) tengan los mismos atributos y métodos que ya se han declarado en las clases previas (que se pueden llamar “clases padre” o “superclases”) sin necesidad de declararlas nuevamente. Por ello se dice que la subclase *hereda* de la superclase. Esta característica hace que se reutilice código existente puesto que no se necesita declararla nuevamente. Ya que se habla de clases hijas y padres, se genera lo que se conoce como una jerarquía de clases, en la que cada vez que se modifica un método o un atributo de una clase padre, automáticamente la clase hija contará con dicha modificación sin tener que crearle ninguna línea de código adicional.

Sobreescritura. En una jerarquía de clases, cuando un método de una subclase tiene el mismo nombre y tipo que un método de su superclase, se dice que el método de la subclase reemplaza o *sobrescribe* el método de la superclase. Cuando se llama a un método sobrescrito desde una subclase, ésta siempre se refiere a la versión del método definida en la subclase. La versión del

método definido por la superclase quedará oculta. Si se quiere acceder a la función sobrescrita desde la función de la subclase, se utilizará el método “super”.

Polimorfismo. El polimorfismo es una característica que se genera a partir de la herencia. En muchos casos, es posible que las diferentes clases hijas deban comportarse en forma diferente a la clase padre. Un método heredado podría modificarse completamente para cualquiera de las clases hijas, ya sea complementando su comportamiento (llamando al método de su clase superior) o anulándola completamente para crearle un nuevo comportamiento. Esto implica que una clase padre podría “especializarse” en sus respectivas clases hijas, y cada una de ellas podría comportarse de manera diferente, a pesar que los nombres de sus métodos, e incluso su cantidad y tipo de parámetros *sean exactamente iguales*. Se llama polimorfismo por las diferentes formas (diferentes comportamientos) que pueden tomar las clases hijas.

Implementación a partir de las clases:

Cuando se tiene una abstracción inicial de las clases que deben estar en el sistema, se plasma en un diagrama y se le asignan características adicionales a los atributos y métodos de acuerdo con las características de abstracción, encapsulamiento, sobrecarga, herencia, polimorfismo y sobreescritura, y también de acuerdo al contexto en el que se esté trabajando. Es muy posible que aparezcan más atributos y más métodos después, sin embargo, la ventaja radica en que la programación orientada a objetos hace que se deba preocupar solo de la parte que debe modificarse o crearse sin dañar las demás partes del programa.

Carro
Tipo marca modelo placa
acelerar() frenar() girar()

Carro
- tipo - marca - modelo - placa - velocidad - dirección
+ Carro() + acelerar() + frenar() + frenar(emergencia) + girar() + getTipo() + getMarca() + getModelo() + getPlacas() + getVelocidad() + getDireccion() + setTipo() + setMarca() + setModelo() + setPlaca()

```
1   public class Carro
2   {
3       private String tipo;
4       private String marca;
5       private short modelo;
6       private String placas;
7       private byte velocidad;
8       private byte direccion;
9
10      public Carro( )
11      {
12          velocidad = 0;
13          dirección = 0;
14      }
15
16      public void acelerar( )
17
…       ...
```