**Reporte**

**Variable Estática**

En la programación una variable estática es una variable que ha sido definida al principio del programa estáticamente y cuya existencia será útil durante toda la ejecución del programa. Este tipo de variables se pueden establecer durante el tiempo de ejecución así como pueden ser manipuladas en distintas ocasiones durante la ejecución del programa. Este concepto también puede ser denominado como una variable global.

**Ciclo de vida de las variables**

* La ejecución de un programa ocupa un intervalo de tiempo:
* El tiempo de vida de una variable es el intervalo o intervalos de tiempo durante los cuales la variable existe
* Los intervalos están normalmente incluidos en el intervalo de tiempo de ejecución del programa. A cada uno de ellos se les denomina intervalo de existencia
* Una variable existe durante la ejecución cuando se puede acceder a su valor, en el sentido de que se puede garantizar que dicho valor está almacenado en su localización usando la representación asociada a su tipo.

**Memoria Dinámica**

La memoria dinámica es la memoria que no se define dentro de un programa, se desconoce la cantidad a considerarse. Se permite solicitar memoria al mismo tiempo de ejecución. Conforme avance el programa y se vaya requiriendo mas memoria se le irá solicitando al sistema operativo de acorde a lo necesario. Este tipo de memoria se maneja mediante el uso de punteros por lo tanto resulta imposible conocer el tamaño de la memoria al momento de compilar. Cuando se desarrolla un programa el cual hace uso de la memoria dinámica, es necesario dividir el programa en cuatro partes: datos, texto, pila y la zona libre o heap. En el heap es donde queda la memoria libre para que esta sea utilizada como memoria dinámica.

**Clase**

La clase es una plantilla la cual permite la creación de objetos. Estas son utilizadas como medio de representación de conceptos o entidades. Cada clase busca definir un conjunto de variables. Los objetos creados partir de una clase son denominados instancias de la clase. Las clases son fundamentales para la funcionalidad de la programación orientada a objetos. Las clases están conformadas por campos de datos, métodos o subrutinas, en algunos lenguajes también se manejan las propiedades entre los campos y métodos.

**Objeto**

Un objeto se define como una unidad dentro de un programa de la computadora que consiste en un estado y un comportamiento, suelen tener datos almacenados y de tareas realizables durante el tiempo de ejecución. El objeto se puede generar instanciando una clase, esto ocurre en la programación orientada objetos. Los objetos interactúan unos con otros. Los objetos son capaces de procesar datos recibir mensajes y generar los mismos para ser enviados a otros objetos. Gracias a las clases se crean los objetos que pertenecen a esa clase. Ejemplo la clase\_animal vendría siendo la clase entonces un rinoceronte sería un objeto de la clase\_animal.

**Instanciación**

Es la realización de una clase o prototipo determinados. En el momento que un programa es ejecutado, esto significa que el programa se instancia. Un objeto también se puede definir como una instancia dentro de la clase. En otras palabras se refiere que un miembro de la clase cuenta con atributos en vez de variables.

**Herencia**

La herencia es la transmisión del código entre unas clases y otras. Para soportar un mecanismo de herencia tenemos dos clases: la clase padre y la(s) clase(s) hija(s). La clase padre es la que transmite su código a las clases hijas. En muchos lenguajes de programación se declara la herencia con la palabra "extends".

class Hija extends Padre{ }

Eso quiere decir que todo el código de la clase padre se transmite, tal cual, a la clase hija. Si lo quieres ver así, es como si tuvieras escrito, línea a línea, todo el código de la class "Padre" dentro de las llaves de la class "Hija". Por eso, la herencia es fundamental para reutilizar código, porque no necesitas volver a incorporar el código de Padre e Hija, sino que realmente al hacer el "extends" es como si ya estuviera ahí.

**Sobrecarga**

Es un mecanismo que permite asignar el mismo nombre a [funciones](http://www.ecured.cu/index.php/Funci%C3%B3n_%28programaci%C3%B3n%29) distintas. Para el compilador estas funciones no tienen nada en común a excepción del [identificador](http://www.ecured.cu/index.php/Identificadores_%28Inform%C3%A1tica%29), por lo que se trata en realidad de un recurso semántico del lenguaje que sólo tiene sentido cuando se asigna el mismo nombre a funciones que realizan tareas similares en objetos diferentes.

Cuando se realiza la invocación de una función sobrecargada, es decir que existen otras del mismo nombre en el mismo ámbito, el compilador decide cuál de ellas se utilizará mediante un proceso denominado resolución de sobrecarga ("Overload resolution") aplicando ciertas reglas para verificar cuál de las declaraciones se ajusta mejor al número y tipo de los argumentos utilizados. Es decir, donde existe máxima concordancia entre los argumentos actuales y formales.

**Ciclo de vida de variables**

**Instancia (objeto)**

* Existen cuando se crea un objeto que las contiene
* Se inicializan por defecto si no se hace de modo explícito
* Se destruyen cuando el recolector de basura de Java no encuentra referencias activas para el objeto.

**Estáticas (clase)**

* Se crean cuando la clase se usa por primera vez
* Se inicializan por defecto si no se hace de modo explícito
* Suelen existir por el resto del programa (salvo que este no se haya cargado)

**Locales (bloque)**

* Creadas en la sentencia en la que están definidas
* No se inicializan por defecto, contienen datos imprevisibles
* Se destruyen al salir del bloque (llave final)

Shadowing

Ocurre cuando una variable declarada con un cierto nombre (bloque, método, clase) tiene el mismo nombre declarado que otra, esto es conocido como “disfrazar el nombre”. Se dice que la primera variable es “sombreada” ( de ahí shadow) por la variable interna, mientras que la interna disfraza a la primera, esto puede llegar a una confusión, pero esto depende de la resolución de los nombres en los lenguajes.

**Bibliografía**

Alvarez. M. (2014). *Herencia en Programación Orientada a Objetos*. Octubre 1 2015, de desarrolloweb.com Sitio web:<http://www.desarrolloweb.com/articulos/herencia-en-programacion-orientada-objetos.html>

(2015). *Sobrecarga de funciones (programación)*. Octubre 1 2015, de EcuRed Sitio web:<http://www.ecured.cu/index.php/Sobrecarga_de_funciones_%28programaci%C3%B3n%29>

Castro, C. C., de Leon Amador, P. J. P., & Boró, E. S. (2006). *Introducción a la programación orientada a objetos*. Publicaciones de la Universidad de Alicante

(2015). *Variable Shadowing*. Octubre 1 2015, de Wikipedia Sitio web:<https://en.wikipedia.org/wiki/Variable_shadowing>