**UNIVERSIDAD DE SAN CARLOS DE GUATEMALA**
**CURSO IPC**
**INGENIERÍA EN SISTEMAS**
**GRUPO #1**

**Catedrático: Bryan Prado Marroquín**

# INVESTIGACION PARTE 1


**Nombre: PABLO ANDRES RODRIGUEZ LIMA**

**Carnet:**

| 2 | 0 | 2 | 2 | – | 0 | 1 | 9 | 4 | 7 |
|---|---|---|---|---|---|---|---|---|---|

1. Recursividad

La recursividad en programación Java es un concepto que, normalmente, cuesta entender a los principiantes en este lenguaje. Se trata de algo que puede parecer bastante complejo a primera vista, pero que, en realidad, es relativamente sencillo.

Resumiéndolo mucho, podemos decir que la recursividad en programación Java es la capacidad que existe en este lenguaje de crear funciones que llamen a otras funciones. De esta manera se evita el uso de bucles u otros iteradores. Se trata de una técnica de programación que permite crear instrucciones que se repitan un número n de veces, por eso se trata de una forma de programación que permite evitar el uso de estructuras de datos repetitivas.

2. Que es Programación Orientada a objetos

La programación orientada a objetos establece un equilibrio entre la importancia de los procesos y los datos, mostrando un enfoque más cercano al pensamiento del ser humano. Se introduce un aspecto novedoso respecto al anterior paradigma: la herencia, facilitando el crecimiento y la mantenibilidad. Las bases de la programación orientada a objetos son: abstracción, encapsulación, modularidad y jerarquización.

La abstracción es un proceso mental de extracción de las características esenciales, ignorando los detalles superfluos. Resulta ser muy subjetiva dependiendo del interés del observador, permitiendo abstracciones muy diferentes de la misma realidad La encapsulación es ocultar los detalles que dan soporte a un conjunto de características esenciales de una abstracción. Existirán dos partes, una visible que todos tienen acceso y se aporta la funcionalidad, y una oculta que implementa los detalles internos. El modularidad es descomponer un sistema en un conjunto de partes. Aparecen dos conceptos muy importantes: acoplamiento y cohesión. El acoplamiento entre dos módulos mide el nivel de asociación entre ellos; nos interesa buscar módulos poco acoplados La cohesión de un módulo mide el grado de conectividad entre los elementos que los forman; nos interesa buscar una cohesión alta La jerarquía es un proceso de estructuración de varios elementos por niveles. La programación orientada a objetos implementa estos cuatro conceptos con los siguientes elementos: clases y objetos, atributos y estado, métodos y mensajes, herencia y polimorfismo.

3. Pilares de POO

Clasificar los temas considerados «básicos» puede ser difícil. Por ello, hemos diseñado una hoja de trucos con los principales conceptos necesarios para aprender programación orientada a objetos en Python.

Variable: Nombre simbólico que apunta a un objeto específico (veremos qué significan los objetos a lo largo del artículo).
Operadores aritméticos: Suma (+), resta (-), multiplicación (*), división (/), división entera (//), módulo (%).
Tipos de datos incorporados: Numéricos (enteros, flotantes, complejos), Secuencias (cadenas, listas, tuplas), Booleanos (Verdadero, Falso), Diccionarios y Conjuntos.
Expresiones booleanas: Expresiones en las que el resultado es True o False.
Condicional: Evalúa una expresión booleana y realiza algún proceso dependiendo del resultado. Se maneja mediante sentencias if/else.
Bucle: Ejecución repetida de bloques de código. Pueden ser bucles for o while.
Funciones: Bloque de código organizado y reutilizable. Se crean con la palabra clave def.
Argumentos: Objetos que se pasan a una función.

4. Que es un Diagrama de Clases

   Los diagramas de clases son diagramas estructurales del lenguaje unificado de modelado o UML (unified modeling language). Este lenguaje de modelado visual es un estándar ISO para visualizar los sistemas de la programación orientada a objetos, aunque también permite visualizar procesos de negocio. Con ayuda de elementos gráficos, UML muestra los estados de los sistemas y describe las interacciones entre los elementos que lo componen. Para poder hacerlo, la notación UML define las formas y las líneas para 14 tipos de diagrama.
5. Relaciones entre clases (asociación, composición, herencia)

Asociación

Es una relación entre dos clases que requieren relacionarse (como una orden y un cliente), mediante UML y el diagrama de clases de representa mediante una línea y en algunos casos incluye una flecha siguiendo la estructura de una oración. Esta flecha apunta al objeto y el otro extremo representa al sujeto.

Composición: Es un tipo especial de asociación entre clases conocido como: “contiene a” o “es contenido en”. Ambas clases tiene una vida independiente, pero una de estas (llamada invitado) trabaja en orden a otra (huésped). Esto le permite a huésped usar características de invitado en la ejecución de algunas tareas.  Se representa con una línea con rombo blanco, donde el rombo está en el huésped y el otro lado en el invitado.

Herencia: Tipo especial de agregación. Conocida como "es parte de" o "es un todo de". En este caso las dos partes necesitan de ellas para existir (una no existe sin la otra), de manera que existe una clase (todo) que utiliza características de otra (parte) para la ejecución de alguna tarea.

Se representa con un rombo relleno, que está junto al "todo" y al otro extremo la "parte".

6. Que es una Clase Padre y una clase Hijo

El concepto de herencia conduce a una estructura jerárquica de clases o estructura de árbol, lo cual significa que en la OOP todas las relaciones entre clases deben ajustarse a dicha estructura.

En esta estructura jerárquica, cada clase tiene sólo una clase padre. La clase padre de cualquier clase es conocida como su superclase. La clase hija de una superclase es llamada una subclase.

De manera automática, una subclase hereda las variables y métodos de su superclase (más adelante se explica que pueden existir variables y métodos de la superclase que la subclase no puede heredar. Véase Modificadores de Acceso). Además, una subclase puede agregar nueva funcionalidad (variables y métodos) que la superclase no tenía.

# CONCLUSION

- la recursividad es un recurso útil para evitar usar y optimizar el código a la hora de trabajar ya que permite que se pueda usar definidamente o indefinida un programa lo que hace que sea vital.

- La computadora y el lenguaje de programación es capaz de orientar figuras para que realicen un objetivo de modo que puedan crear una interfaz grafica para que los programas puedan funcionar visualmente mejor, este tipo de lenguaje depende de una estructura y de una nueva interfaz para programar mencionando sus complementos gráficos.

- Las clases en programación nos pueden a optimizar el código ya que podemos crear que por grupos hagan cierta parte de un programa esto conlleva a que sus partes se puedan mezclar, asociar y derivar, además entre ellas destaca que para una misma pueden existir una clase padre y una clase hijo lo que hace que puedan verse vinculadas estructuralmente.