Kevin Andres Alvarez Herrera

202203038

Recursividad

La recursividad es un concepto difícil de entender en principio, pero luego de analizar diferentes problemas aparecen puntos comunes.
En Java los métodos pueden llamarse a sí mismos. Si dentro de un método existe la llamada a sí mismo decimos que el método es recursivo.
Cuando un método se llama a sí mismo, se asigna espacio en la pila para las nuevas variables locales y parámetros.
Al volver de una llamada recursiva, se recuperan de la pila las variables locales y los parámetros antiguos y la ejecución se reanuda en el punto de la llamada al método
Ejemplo 1:
Implementación de un método recursivo.

```
public class Recursividad {

    void repetir() {
        repetir();
    }

    public static void main(String[] ar) {
        Recursividad re=new Recursividad();
        re.repetir();
    }
}
```

Ejemplo 2:

Implementación de un método recursivo que reciba un parámetro de tipo entero y luego llame en forma recursiva con el valor del parámetro menos 1.

```
public class Recursividad {

    void imprimir(int x) {
        System.out.println(x);
        imprimir(x-1);
    }

    public static void main(String[] ar) {
        Recursividad re=new Recursividad();
        re.imprimir(5);
    }
}
```

Ejemplo 3:

Implementar un método recursivo que imprima en forma descendente de 5 a 1 de uno en uno.

```
public class Recursividad {

    void imprimir(int x) {
        if (x>0) {
            System.out.println(x);
            imprimir(x-1);
        }
    }

    public static void main(String[] ar) {
        Recursividad re=new Recursividad();
        re.imprimir(5);
    }
}
```

Ejemplo 4

Imprimir los números de 1 a 5 en pantalla utilizando recursividad.

```
public class Recursividad {

    void imprimir(int x) {
        if (x>0) {
            imprimir(x-1);
            System.out.println(x);
        }
    }

    public static void main(String[] ar) {
        Recursividad re=new Recursividad();
        re.imprimir(5);
    }
}
```

Ejemplo 5

Implementar un método recursivo para ordenar los elementos de un vector.

```
class Recursivdad {
    static int [] vec = {312, 614, 88, 22, 54};

    void ordenar (int [] v, int cant) {
        if (cant > 1) {
            for (int f = 0 ; f < cant - 1 ; f++)
                if (v [f] > v [f + 1]) {
                    int aux = v [f];
                    v [f] = v [f + 1];
                    v [f + 1] = aux;
                }
            ordenar (v, cant - 1);
        }
    }

    void imprimir () {
        for (int f = 0 ; f < vec.length ; f++)
            System.out.print (vec [f] + "  ");
        System.out.println("\n");
    }


    public static void main (String [] ar)  {
        Recursivdad r = new Recursivdad();
        r.imprimir ();
        r.ordenar (vec, vec.length);
        r.imprimir ();
    }
}
```

Que es Programación Orientada a objetos

La **Programación Orientada a Objetos** (POO) es un paradigma de programación, es decir, un modelo o un estilo de programación que nos da unas guías sobre cómo trabajar con él. Se basa en el **concepto de clases y objetos**. Este tipo de programación se utiliza para estructurar un programa de software en piezas simples y reutilizables de planos de código (clases) para crear instancias individuales de objetos.

Clases, objetos e instancias

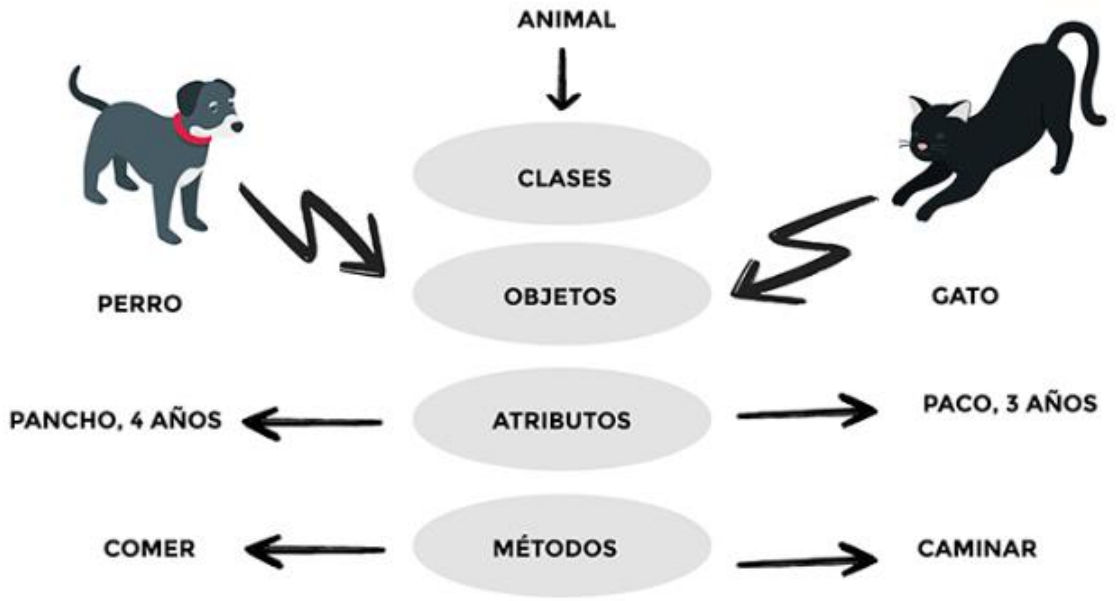
¿Cómo se crean los programas orientados a objetos? Resumiendo mucho, consistiría en hacer clases y crear objetos a partir de estas clases. Las clases forman el modelo a partir del que se estructuran los datos y los comportamientos.

El primer y más importante concepto de la POO es la distinción entre clase y objeto.

Una clase es una plantilla. Define de manera genérica cómo van a ser los objetos de un determinado tipo. Por ejemplo, una clase para representar a animales puede llamarse ‘animal’ y tener una serie de atributos, como ‘nombre’ o ‘edad’ (que normalmente son propiedades), y una serie con los comportamientos que estos pueden tener, como caminar o comer, y que a su

vez se implementan como métodos de la clase (funciones).

Un ejemplo sencillo de un objeto, como decíamos antes, podría ser un animal. Un animal tiene una edad, por lo que creamos un nuevo atributo de ‘edad’ y, además, puede envejecer, por lo que definimos un nuevo método. Datos y lógica. Esto es lo que se define en muchos programas como la definición de una clase, que es la definición global y genérica de muchos objetos.
Ejemplo:



Beneficios de Programación Orientada a Objetos

- Reutilización del código.
- Convierte cosas complejas en estructuras simples reproducibles.
- Evita la duplicación de código.
- Permite trabajar en equipo gracias al encapsulamiento ya que minimiza la posibilidad de duplicar funciones cuando varias personas trabajan sobre un mismo objeto al mismo tiempo.
- Al estar la clase bien estructurada permite la corrección de errores en varios lugares del código.
- Protege la información a través de la encapsulación, ya que solo se puede acceder a los datos del objeto a través de propiedades y métodos privados.
- La abstracción nos permite construir sistemas más complejos y de una forma más sencilla y organizada.

Los pilares de POO

1. Abstracción. La abstracción oculta al usuario la funcionalidad interna de una aplicación: Tomemos por ejemplo la clase carro que hicimos, esta tiene la posibilidad de guardar datos relacionados a la marca y al año de salida al mercado del carro, pero, ¿Por qué solamente estas dos informaciones? Un carro del mundo real tiene más propiedades, como el color y el modelo. Sin embargo, debemos preguntarnos, ¿Son estas informaciones relevantes para nuestro software?
2. Herencia. La herencia nos permite definir múltiples subclases a partir de una clase ya definida, Compartir código es una importante y crucial característica de cualquier proyecto de software. Compartir código permite ahorrar trabajo cuando queremos hacer un cambio en nuestro sistema; permite que un solo algoritmo pueda procesar distintas clases de entidades; entre otras cosas.

   Hay varias maneras de compartir código, una de ellas es utilizando herencia. La herencia es una relación especial entre dos clases, la clase base y la clase derivada, en donde la clase derivada obtiene la habilidad de utilizar ciertas propiedades y funcionalidades de la clase base, incluso pudiendo sustituir funcionalidad de la clase base. La idea es que la clase derivada "hereda" algunas de las características de la clase base.

3. Polimorfismo: Cuando empezamos a hablar de herencia, dijimos que la herencia "permite que un solo algoritmo pueda procesar distintas clases de entidades". La idea es que podemos tener una función la cual reciba un parámetro, como una clase base, y podemos pasarle a ese método objetos que sean instancias de las clases derivadas de dicha clase base. Lo mismo ocurre si el método recibe como parámetro una interfaz. Podemos pasarle a dicho método cualquier clase que implemente dicha interfaz.
4. Encapsulación: El encapsulamiento nos permite controlar quien puede ver y utilizar los distintos módulos internos de nuestro sistema. En términos de clases, con el encapsulamiento definimos el acceso a los miembros de la clase.

Que es un Diagrama de Clases

Los diagramas de clases son uno de los tipos de diagramas más útiles en UML, ya que trazan claramente la estructura de un sistema concreto al modelar sus clases, atributos, operaciones y relaciones entre objetos. Con nuestro software de generación de diagramas UML, la creación de estos diagramas no es tan abrumadora como podría parecer.

Beneficios de los diagramas de clases

Los diagramas de clases ofrecen una serie de beneficios para toda organización. Usa los diagramas de clases UML para:

- Ilustrar modelos de datos para sistemas de información, sin importar qué tan simples o complejos sean.
- Comprender mejor la visión general de los esquemas de una aplicación.
- Expresar visualmente cualesquier necesidades específicas de un sistema y divulgar esa información en toda la empresa.
- Crear diagramas detallados que resalten cualquier código específico que será necesario programar e implementar en la estructura descrita.
- Ofrecer una descripción independiente de la implementación sobre los tipos empleados en un sistema que son posteriormente transferidos entre sus componentes.

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.

***** Los constructores no son heredados por las subclases.

Para crear una subclase, se incluye la palabra clave **extends** en la declaración de la clase.

class nombreSubclase **extends** nombreSuperclase{

}

En Java, la clase padre de todas las clases es la clase **Object** y cuando una clase no tiene una superclase explícita, su superclase es Object.