* En definitiva: La sobrecarga se resuelve en tiempo de compilación utilizando los nombres de los métodos y los tipos de sus parámetros; el polimorfismo se resuelve en tiempo de ejecución del programa, esto es, mientras se ejecuta, en función de la clase a la que pertenece el objeto.

**Ahora veremos un ejemplo aplicado a la programación.**

En la POO la definición de polimorfismo es **la característica que un objeto tiene para poder tomar la forma o el lugar de un tipo diferente**, pero esta definición nos puede perturbar, así que en términos más comunes es la**cualidad que tienen los objetos de compartir características comunes entre ellos utilizando una superclase que contenga esas características comunes y que podamos usar los objetos como si fueran del mismo tipo de la superclase**.  
  
Aquí les va el ejemplo, ahora si en código:  
  
Tenemos la clase animal definida de la siguiente forma:  
  
En esta clase definimos una característica común a todos los animales, todos los animales comen.  
  
  
public class Animal {  
    
    public void comer(){  
        System.out.println("El animal come");  
    }  
    
}  
  
  
Ahora definimos la clase de un animal especifico, por ejemplo un Leon, ojo, aquí también la clase Leon tiene definido el método comer con la misma firma que en la clase Animal, pero notemos que el bloque del método no es el mismo que en la clase Animal, a esto se le llama **sobre-escritura de método** y recuerdan en el ejemplo el apartado donde digo que la forma en que cada animal hace uso de sus características es la clave del polimorfismo, pues aqui en la POO es precisamente la sobre-escritura de métodos.; Notemos que el método atacar no esta definido en la clase Animal y por lo tanto no esta sobre-escrito, este método es propio del Leon.  
  
  
public class Leon extends Animal{  
    
    @Override  
    public void comer(){  
        System.out.println("el leon come carne");  
    }  
    
    public void atacar(){  
        
    }  
    
}  
  
  
Ahora probamos estas dos claces y veremos el resultado:  
  
  
public class Polmorfismo {  
  
    public static void main(String[] args) {  
        // TODO code application logic here  
        Animal a = new Animal();  
        Animal l = new Leon();  
        a.comer();  
        l.comer();  
        
    }  
}  
  
  
El resultado de la ejecución de esta clase veremos que cada objeto hace uso del método comer de una forma muy particular.  
  
  
run:  
El animal come  
el leon come carne  
BUILD SUCCESSFUL (total time: 0 seconds)  
  
Aquí nos podemos dar cuenta que cada objeto implementa de manera distinta la acción de comer, sin embargo en este ejemplo podríamos apreciar claramente esta deducción, ahora vemos el siguiente código.  
  
Podemos apreciar que, tenemos una lista de animales y, esa lista contiene objetos de tipo Animal, uno es un objeto de tipo Animal y otro es de tipo Leon, aparentemente son objetos de distintos tipo lo cual es correcto, pero, si recordamos, la clase Leon hereda de la clase animal, por lo tanto también es un animal y ambos tiene la característica de comer aunque de distinta forma, ahora al recorrernos sobre la lista notemos que solo obtenemos el objeto en la posición actual sin importar de que tipo y simplemente invocamos al método comer, la pregunta es, como sabemos a que método se invocará de que clase, en cada posición de la lista?, bueno, esto es gracias al polimorfismo ya que ambos son animales pero cada uno implementa de manera distinta el método, lo explicaré mejor.  
  
Cuando se inicio el recorrido en la lista, el compilador obtiene el objeto en la posición actual y evalúa de que tipo es el objeto que se recupera en esa posición y es así como sabe a que método se invocará, si al de la clase Animal o al de la clase Leon, recordemos que cada instancia aunque sea del mismo tipo, ocupa direcciones de memoria distintas, por lo tanto, la ejecución de los métodos de la clase solo afectan a esa instancia y no a las demás, esto lo explicare detalladamente en otro post.  
  
  
public class Polimorfismo {  
  
    public static void main(String[] args) {  
        List<Animal> animales = new ArrayList<Animal>();  
        Leon l = new Leon();  
        Animal a = new Animal();  
        
        animales.add(a);  
        animales.add(l);  
        
        for( int x = 0; x < animales.size(); x++ ){  
            animales.get(x).comer();  
        }  
    }  
}  
  
  
  
El resultado será el siguiente:  
  
  
run:  
El animal come  
el leon come carne  
BUILD SUCCESSFUL (total time: 1 second)  
  
Espero que esta explicación y el ejemplo haya sido suficientemente bueno para entender el concepto de polimorfismo, sin temor a equivocarme esto puede ayudarles a comprender el tema pero cada uno le dará un comprensión distinta, en un siguiente post daré algunos tips sobre el tema, en función de las reglas del compilador para este concepto, que considero son necesario para tener un conocimiento más amplio sobre ello.  
  
  
**Reglas del compilador:**

1. El polimorfismo se aplica a variables de instancia cuando una clase hereda de otra o cuando una clase implemente una interfaz.
2. La invocación a un método de una subclase debe realizarse a través de una referencia de la superclase.
3. El método invocado debe ser también un miembro de la superclase.
4. La firma del método en las clases derivadas debe ser la misma en la superclase.
5. El tipo de acceso del método en las clases derivadas no debe ser más restringido que en la superclase.
6. El tipo de retorno del método debe ser igual en las clases derivadas que en la superclase.