**Herencia**

La herencia es otra de las características de la Programación Orientada a Objetos, nos da la facilidad de crear funcionalidades en clases derivados de otras. En los módulos anteriores hemos hecho una pequeña introducción para el concepto, ahora, conoceremos más a fondo sus posibilidades.

Java, como lenguaje de programación orientado a objetos nos da esa facilidad, nos permite crear estructuras de clase utilizando todas las cualidades de la herencia.

**Definición:**

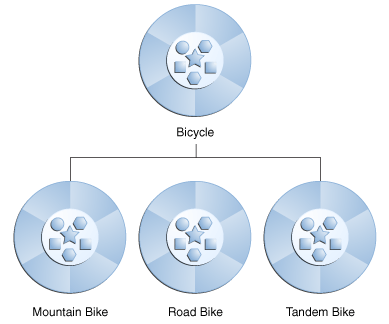
*Es la capacidad de pasar las características de campos y métodos de una clase a otra, heredarlas para generar una jerarquía que permita la generación de código más eficiente. Las clases que derivan de otras son llamadas subclases, y la clase de la cual derivan otras es llamada superclase (Clase base / Clase padre).*

**Algunas generalidades en herencia**

* Las clases ubicadas arriba son generales o abstractas.
* Las clases ubicadas abajo son más específicas o concretas.
* La herencia nos da la posibilidad de tener un código más organizado y fácil de entender.
* La evolución y mantenimiento del software se simplifican y reducen los costos de operación.
* Las clases comparten características comunes, lo que elimina la repetición de código.
* En java la herencia permite a una clase tener un número ilimitado de subclases y una superclase.
* Los constructores no son miembros, por lo que no son heredados, pero si puede ser invocado por una subclase.

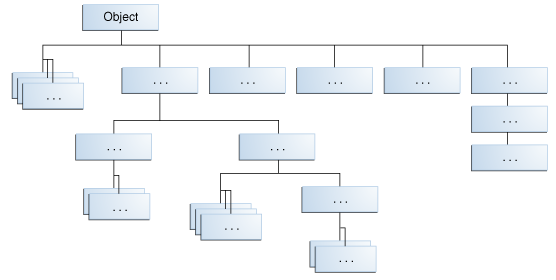
La idea de la herencia es simple pero poderosa, cuando queremos definir una nueva clase en nuestro proyecto y ya existen algunas características que necesitamos en otra clase, podemos derivarla de esa existente sin necesidad de escribir ni depurarla nosotros mismos.

**Representación básica:**



En este caso el objeto Bicycle representa la superclase, de la cual surgen tres clases diferentes, llamadas subclases, en la representación *Mountain Bike, Road Bike y Tandem Bike* (Tres diferentes tipos de bicicletas), comparten las características y comportamientos de la superclase Bicycle.

En Java, todas las clases descienden de la clase *java.lang.object.* A excepción de Object, todas las clases tienen una superclase.



La clase **Object**, está arriba de toda la jerarquía de clases en Java, definida en el paquete java.lang, se encarga de definir el comportamiento común para todas las clases incluyendo las que escribimos.

**Ejemplo básico de aplicación de herencia en Java**

public class Bicicleta

{

   // Campos de la clase Bicicleta

   public int cadencia;     //Revolciones por mínuto o RPM

   public int engranaje;

   public int velocidad;

   // Un Constructor de la clase bicicleta

   public Bicicleta(int cadenciaInicial, int velocidadInicial, int engranajeInicial) {

       engranaje = engranajeInicial;

       cadencia = engranajeInicial;

       velocidad = velocidadInicial;

  }

   // 4 métodos de la clase bicicleta

   public void establecerCadencia(int nuevoValor) { //Set

       cadencia = nuevoValor;

  }

   public void establecerEngranaje(int nuevoValor) {

       engranje = nuevoValor;

  }

   public void Frenar(int decrementar) {

       velocidad -= decrementar;

  }

   public void Acelerar(int incrementar) {

       velocidad += incrementar;

  }

}

Ahora, definiremos una subclase que extienda de la clase Bicicleta:

Usamos **extends** para heredar las características y comportamiento de una clase padre.

public class bicicletaMontanesa extends Bicicleta

{

   // Nuevo campo para la clase bicicletaMontanesa

   public int alturaAsiento;

​

   // Constructor para la clase bicicletaMontanesa

   public bicicletaMontanesa(int alturaInicial,

                       int cadenciaInicial,

                       int velocidadInicial,

                       int engranajeInicial)

{

       super(cadenciaInicial, velocidadInicial, engranajeInicial);

       alturaAsiento = alturaInicial;

  }

   // Agregamos un nuevo método a la subclase

   public void establecerAltura(int nuevoValor)

{

       alturaAsiento = nuevoValor;

  }

}

* Los atributos definidos en el primer bloque de código corresponden a la clase padre o superclase **Bicicleta**
* Los atributos definidos en el segundo bloque de código corresponden a la subclase bicicletaMontanesa, en donde se han definido algunas propiedades que son agregadas con las que ya se han heredado de la clase padre.
* Sin hacer más tenemos una clase **bicicletaMontnesa** que tiene atributos y métodos heredados lo que simplifica la codificación en aplicaciones complejas y que mejora la eficiencia en el desarrollo.

**Subclases**

Heredan todos los miembros públic y protected de la superclase, sin importar en que paquete está.

**Consideraciones:**

Si la subclase está en el mismo paquete que su clase padre, también puede heredar los miembros privados de esa clase.

**¿Qué se hace con los miembros que se heredan?**

Los campos heredados pueden usarse como cualquier otro.

Se puede definir un campo en la subclase con el mismo nombre, lo que permitirá su ocultación.

Declarar nuevos campos en la subclase que no estén definidos en la clase padre.

Los métodos heredados pueden ser usados tal y como son.

Podemos definir una nueva instancia en la subclase con el mismo nombre de la clase padre, lo que permitirá la sobreescritura.

Podemos declarar nuevos métodos que no estén definidos en la superclase

**Miembros privados en Superclase**

Para acceder y usar los campos privados de una superclase, se hace por medio de sus métodos *públic* o *protected*.

Las clases anidadas en el mismo paquete tienen acceso a los miembros privados. Por lo tanto, una clase anidada *protected* o *public* heredada por una subclase tiene acceso indirecto a todos los miembros privados de la superclase.

**Clases anidadas**

Las clases anidadas se usan cuando queremos definir una lógica de aplicación agrupada en un solo lugar, es cuando esa clase solo será usada sólo por otra clase, ayuda a incrementar la encapsulación y promueve una modificación y mantenimiento de código rápida. La estructura de las clases anidadas es la siguiente:

class OuterClass

{

  ...

   class NestedClass

{

      ...

}

}

Las clases anidadas pueden ser: Estáticas, declaradas como *static*. Las que no son estáticas están definidas como clases internas o *Inner Classes*

class OuterClass

{

  ...

   static class StaticNestedClass

{

      ...

  }

   class InnerClass

{

      ...

  }

}

**Polimorfismo**

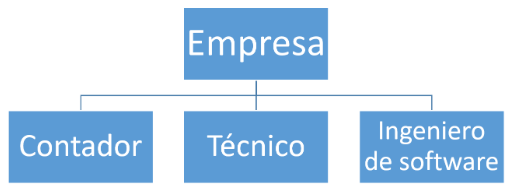
Si buscamos el término polimorfismo en nuestro diccionario, encontramos la siguiente definición:

*"Calidad de Poliformo. Propiedad de las especies de seres vivos cuyos individuos pueden presentar diferentes formas o aspectos, bien por diferenciarse en castas, como las termitas, bien por tratarse de distintas etapas del ciclo vital, como la oruga y la mariposa."*

La idea se aplica también en la programación Orientada a objetos, la función que pueden cumplir las subclases con diferentes comportamientos y al mismo tiempo compartir alguna funcionalidad de las clases padre.

**Algunas consideraciones importantes en cuanto al polimorfismo**

* El polimorfismo está directamente relacionado con la herencia.
* Cuando implementamos un método abstracto en la clase padre, las subclases deben implementarlo de forma obligatoria.
* Las subclases pueden tomar comportamientos diferentes, y ese es el potencial del polimorfismo. El funcionamiento básico es el siguiente:



Consideremos a **Empresa** como Superclase que tiene los siguientes atributos y métodos que representan a los integrantes de una empresa común:

* numeroCedula
* nombre
* edad
* recibirCapacitaciones()
* diaTrabajo()

Hemos declarado algunas características que son comunes para las subclases, todos tienen un nombre, una edad y un número de cédula. También declaramos los métodos recibirCapacitaciones() y diaTrabajo, son acciones que un contador, técnico o ingeniero de software puede ejecutar.

La definición de esos métodos en la clase Padre sería:

public void recibirCapacitaciones()

{

    System.out.println("Recibiendo capacitacion de ética");

}

public abstract void diaTrabajo();

La declaración del método abstracto **diaTrabajo** obliga a las subclases a implementarse en cada una de ellas.

Ahora, consideremos a **Contador, Técnico e Ingeniero de Software** como subclases que serán heredados de la clase padre Empresa, que pueden tener los siguientes atributos:

**Contador**

nombreColegiatura

registrarContabilidad()

**Técnico**

especializacion

repararComputadoras()

**Ingeniero de Software**

tecnologia

escribirProgramas()

Se han declarado entonces nuevos atributos y métodos que se sumarán a los que ya hemos heredado de la clase empresa, generando un concepto de especialización y diferenciación de cada una de sus clases.

Todas las subclases reciben capacitaciones, nada cambia, entonces no es necesario sobrescribir métodos, por defecto ya lo hemos heredado de la clase Empresa.

Con respecto al método **diaTrabajo()** las tres subclases comparten la acción pero cada uno hace diferentes cosas.

Para un contador agregaríamos lo siguiente en la clase **Contador**

@Override

  public void diaTrabajo()

{

     System.out.println("Registra los asientos contables y estados financieros");

  }

Para la clase Técnico, el código para **diaTrabajo** sería el siguiente:

@Override

  public void diaTrabajo()

{

     System.out.println("Dar soporte a los equipos de computo");

  }

Para la clase Ingeniero de Software, el código para el método **diaTrabajo()** es el siguiente:

 @Override

  public void diaTrabajo()

{

     System.out.println("Escribir código de programas para la empresa");

  }

Todas las subclases implementan el método **diaTrabajo()** con la etiqueta **@Override** lo que nos permite sobrescribir los métodos de la clase padre, lo que redefine el comportamiento de la misma. Si vemos cada empleado ejecuta una acción diferente en el método.

Al tener una sobre escritura del método **diaTrabajo()**, estamos redefiniendo a cada uno de los objetos de la empresa a tener comportamientos totalmente diferentes, compartiendo siempre las características de la clase Padre.