El siguiente ejemplo ilustra la implementación de Herencia en Java

La clase base será Bicicleta y la clase derivada BicicletaMontanera

La clase Bicicleta tiene atributos marcha (para indicar cantidad de marchas o cambios que tiene) y velocidad para registrar la velocidad. Crearemos constructor, setters & getters y método toString. Además de método acelerar y frenar que cambian el atributo velocidad

La clase BicicletaMontanera extenderá de Bicicleta y tendrá otro atributo alturaAsiento. Crearemos el constructor para inicializar todos los atributos tanto los nuevos y los heredados de su clase padre. Revisaremos el uso de super para hacer referencia a métodos de la clase padre

La clase TestBicicleta la usaremos para crear objetos de las nuevas clases creadas.

**Desarrollo**

Una clase en Java puede ser la clase padre de cualquier otra clase, no necesita ningún modificador especial. Lo único que puede prohibir que una clase sea la clase padre o base de otra es que tenga el modificador final. La clase Bicicleta tiene la siguiente implementación:

public class Bicicleta {

private int marcha; //atributo para indicar la cantidad de cambios que posee

private int velocidad;

public Bicicleta(int marcha, int velocidad) {

this.marcha = marcha;

this.velocidad = velocidad;

}

public int getMarcha() {

return marcha;

}

public void setMarcha(int marcha) {

this.marcha = marcha;

}

public int getVelocidad() {

return velocidad;

}

public void setVelocidad(int velocidad) {

this.velocidad = velocidad;

}

public void acelerar(int incremento){

velocidad+=incremento;

}

public void frenar(int incremento){

velocidad-=incremento;

}

@Override

public String toString() {

return "Bicicleta:" + "Cantidad de marchas=" + marcha + ", velocidad=" + velocidad ;

}

}

Ahora vamos a implementar la clase hija BicicletaMontanera, esta clase para indicar que hereda de Blicicleta debe tener en su linea de declaracion la palabra extends. Esta clase va a heredar todos los atributos y métodos visibles en la clase padre(todos menos los privados)

La clase BicicletaMontanera tendrá un nuevo atributo que se inicializa en el contructor junto con los otros atributos heredados de Bicicleta. Los atributos de Bicicleta se inicializan llamando al constructor de Bicicleta con la llamada en el constructor con super. La llamada con super debe ser la primera línea en el constructor

Para el método toString también hay una llamada al método toString de la clase padre para recuperar los valores de los atributos. Una llamada a un método de la clase padre también se lo hace con la notación super.nombreMetodo

public class BicicletaMontanera extends Bicicleta {

private int alturaAsiento;

public BicicletaMontanera(int marcha, int velocidad, int alturaAsiento) {

super(marcha, velocidad);

this.alturaAsiento = alturaAsiento;

}

public int getAlturaAsiento() {

return alturaAsiento;

}

public void setAlturaAsiento(int alturaAsiento) {

this.alturaAsiento = alturaAsiento;

}

@Override

public String toString() {

return super.toString() + " alturaAsiento=" + alturaAsiento ;

}

}

En la clase TestBicicleta crearemos objetos de Bicicleta y BicicletaMontanera

public class TestBicicleta {

public static void main(String[] args) {

Bicicleta bici1 = new Bicicleta(3,50);

System.out.println(bici1);

BicicletaMontanera bici2 = new BicicletaMontanera(3,100,25);

System.out.println(bici2);

}

}

Para ilustrar el ejemplo de sobrecarga en la clase Bicicleta agregamos dos nuevos métodos

para cambiar el atributo velocidad

public void cambiarVelocidad(int valor, String accion) {

if (accion.equals("aumentar")){

this.velocidad+=valor;

}else {

this.velocidad-=valor;

}

}

public void cambiarVelocidad(int valor) {

this.velocidad+=valor;

}

Para el ejemplo de sobreescritura la clase hija BicicletaMontanera va a cambiar la funcionalidad del método cambiarVelocidad(int valor) para disminuir la velocidad

@Override

public void cambiarVelocidad(int valor) {

this.velocidad-=valor;

}

**Clases Abstractas**

Las clases abstractas son clases que por lo general tienen al menos un método abstracto, es decir un método que no tendrá implementación y que serán las clases hijas quienes lo implementen

Cambiaremos a la clase Bicicleta para que sea abstracta

public abstract class Bicicleta

Pdemos crear un método asbracto

public abstract void cambiarPiezas();

Las clases hijas deben escribir el cuerpo de este método. Por ejemplo la clase BicicletaMontanera puede tener la siguiente implementación

public void cambiarPiezas() {

System.out.println("Cambiando piezas");

}

Al cambiar la clase abstracta ya no podremos crear un objeto de esta clase. Comentaremos en el main la creación del objeto Bicicleta

Creamos una nueva clase hija de Bicicleta llamada BicicletaCarrera

public class BicicletaCarrera extends Bicicleta {

private float peso;

public BicicletaCarrera(int marcha, int velocidad,float peso) {

super(marcha,velocidad);

this.peso = peso;

}

@Override

public String toString() {

return super.toString() + " peso=" + peso ;

}

//metodo propio de la clase

public void registrar() {

System.out.println("Metodo registrar");

}

}

Esta clase implementa un nuevo método registrar()

En la clase TestBicicleta creamos nuevos objetos para este tipo y utilizamos un arraylist para guardar todos los objetos creados

public static void main(String[] args) {

//arraylist de Bicicleta me permite tener varios tipos de bicicletas

ArrayList<Bicicleta> listaBici = new ArrayList<>();

BicicletaMontanera bici2 = new BicicletaMontanera(3,100,25);

bici2.acelerar(20);

//System.out.println(bici2);

bici2.cambiarVelocidad(50);

//System.out.println(bici2);

listaBici.add(bici2);

BicicletaCarrera bici3 = new BicicletaCarrera(3,100,50);

listaBici.add(bici3);

bici3.registrar();

//el siguiente tipo bicicleta se declara con el tipo de la clase base

Bicicleta bici4 = new BicicletaCarrera(3,120,60);

listaBici.add(bici4);

//para tener acceso al metodo registrar hay que hacer downcasting

BicicletaCarrera bici5 =((BicicletaCarrera)bici4);

bici5.registrar();

//iterar en el arraylist para mostrar los objetos

for(Bicicleta b:listaBici) {

System.out.println(b);

}

//iteracion usando for

for(int i=0;i<listaBici.size();i++) {

Bicicleta b = listaBici.get(i);

System.out.println(b);

}

}