

2015

QUINTO ENSAYO

Universidad Nacional de Ingeniería

Chunqui Vela José Lendel 20131032H



1. Sobrecarga de métodos

Es posible declarar dos métodos con el mismo nombre, pero diferenciados con el número o por el tipo de parámetros que reciben. A esta cualidad se le llama sobrecarga de métodos.

2. Herencia

Es una característica propia de los lenguajes orientados a objetos. Se define como la capacidad de reutilizar las propiedades de una clase para crear una nueva clase, “extendiendo” la primera.

La clase original, la cual vamos a “extender”, se denomina “clase original” o “superclase”. La nueva clase que será la extensión de la primera, se denomina “clase derivada” o “subclase”.

La forma de codificar la herencia es mediante la palabra reservada “extends”, y su uso se muestra a continuación

```
Class NombreClaseDerivada extends NombreSuperClase{  
    //  
}
```

La clase derivada reutiliza los métodos y atributos de la clase de la cual hereda, y puede añadir sus propios atributos y métodos (también puede sobrecargar los métodos que hereda).

En cuanto a la accesibilidad, el encapsulamiento sigue rigiendo en las clases heredadas. Así, los métodos y variables privados de una superclase, no son accesibles a los métodos y constructores de la subclase.

Número de subclases, superclases

Una clase puede ser “padre” de muchas clases, mientras que una subclase solo puede tener “un solo padre”. Este hecho hace que una jerarquía de clases (colección de clases que se relacionan mediante herencia) forme una estructura de árbol.

La clase Object

Todas las clases que creamos heredan implícitamente de la clase Object. Esta herencia puede ser directa, es decir, una clase que creamos puede ser hija directa de Object; o indirectamente, o sea, que nuestra clase creada hereda de uno de los hijos de Object.

Forma de invocar los métodos y constructores heredados

Para llamar un método, o bien el constructor, de la superclase en una de sus subclases, se usa la palabra reservada “super”, de la siguiente forma:

super() //Invoca al constructo de la superclase

super.metodoSuperClase //Invoca a un método de la superclase

3. Polimorfismo

Es otra característica propia de los lenguajes de programación orientada a objetos. Se define como la posibilidad de sobre escribir un método de la clase “padre” en una clase derivada, pudiendo “actualizar” este método o “especializarlo”.

Observación Importante: Una clase hija puede invocar los métodos de la clase padre, pero la clase padre no puede invocar los métodos de sus clases hijas.

4. Clases y métodos abstractos

Un método abstracto es un método que se declara en una clase sin código de implementación (solo está el nombre del método, pero “adentro” no hay nada de código). Una clase abstracta es aquella clase que posee al menos un método abstracto.

Las clases abstractas no se pueden instanciar (no se pueden crear objetos de estas clases), pero si pueden heredar. Las clases hijas de una clase abstracta son las que se encargan de darle funcionalidad a los métodos abstractos de su clase “padre”.

Se utilizan las clases abstractas cuando queremos crear una clase que tenga lineamientos generales para algunos objetos que utilizaremos, y que nunca necesitemos instanciarla.

Ejemplo de aplicación de Herencia, Polimorfismo y Clases abstractas

Se creará una clase Vehículo que tendrá los lineamientos generales de varias clases de objetos como por ejemplo un carro o una moto. Se aplicará herencia para esto.

```
Vehiculo.java x Main.java x Moto.java x Carro.java x
package pe.uni.fiis.misiones;

public abstract class Vehiculo {

    private Integer caballosFuerza;
    private Integer ruedas;
    private Integer velocidad;

    public Vehiculo(Integer caballosFuerza, Integer ruedas, Integer velocidad) {
        this.caballosFuerza = caballosFuerza;
        this.ruedas = ruedas;
        this.velocidad = velocidad;
    }

    //Metodos Getters
    public Integer getCaballosFuerza() {return caballosFuerza;}
    public Integer getRuedas() {return ruedas;}
    public Integer getVelocidad() {return velocidad;}

    //Setters
    public void setCaballosFuerza(Integer caballosFuerza) {this.caballosFuerza = caballosFuerza;}
    public void setRuedas(Integer ruedas) {this.ruedas = ruedas;}
    public void setVelocidad(Integer velocidad) {this.velocidad = velocidad;}

    //Métodos
    public abstract void acelera();
    public void frenar(){
        velocidad=0;
    }
}
```

La clase Vehículo tiene el método abstracto “acelera”. Como se ve no tiene ninguna línea de código adentro, solo está declarado.

Ahora creamos las clases Moto y Carro:

```
Vehiculo.java x Main.java x Moto.java x Carro.java x
package pe.uni.fiis.misiones;

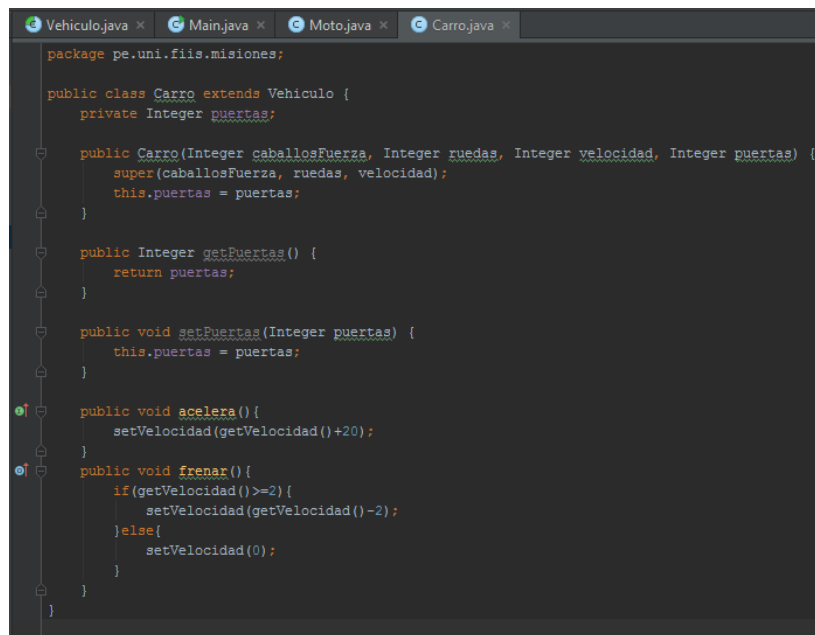
public class Moto extends Vehiculo{
    private Boolean reparar_cadena;

    //Constructor
    public Moto(Integer caballosFuerza, Integer ruedas, Integer velocidad, Boolean reparar_cadena) {
        super(caballosFuerza, ruedas, velocidad);
        this.reparar_cadena = reparar_cadena;
    }

    public Boolean getReparar_cadena() {
        return reparar_cadena;
    }

    public void acelera(){
        setVelocidad(getVelocidad()+10);
    }

    public void frenar(){
        if(getVelocidad()>=5){
            setVelocidad(getVelocidad()-5);
        }else{
            setVelocidad(0);
        }
    }
}
```



```

package pe.uni.fiis.misiones;

public class Carro extends Vehiculo {
    private Integer puertas;

    public Carro(Integer caballosFuerza, Integer ruedas, Integer velocidad, Integer puertas) {
        super(caballosFuerza, ruedas, velocidad);
        this.puertas = puertas;
    }

    public Integer getPuertas() {
        return puertas;
    }

    public void setPuertas(Integer puertas) {
        this.puertas = puertas;
    }

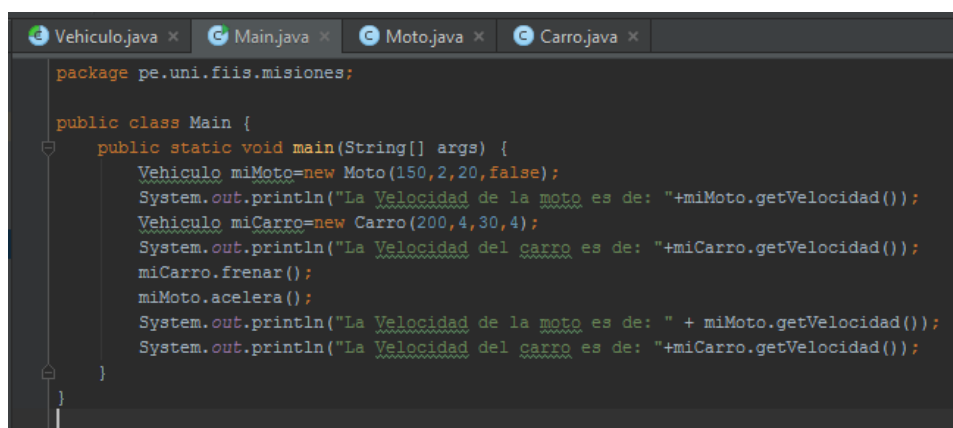
    public void acelera() {
        setVelocidad(getVelocidad()+20);
    }

    public void frenar() {
        if(getVelocidad()>=2){
            setVelocidad(getVelocidad()-2);
        }else{
            setVelocidad(0);
        }
    }
}

```

Las subclases están obligadas a implementar el método acelera, pues éste es un método abstracto en la superclase. Si no lo hicieran, están obligadas a ser también clases abstractas. Ambas subclases definen distinto el método acelera (el carro acelera más que la moto).

En el caso del método frenar, vemos que este está implementado de distintas formas en las subclases. En este caso, cada subclase sobrescriben el método frenar (un carro frena más rápido, y ninguno de estos tipos de vehículo frenan de golpe). Pueden especializar el método, agregando (sobrecargando), y no sobrescribiendo, algunos comportamientos particulares. Al implementar el método Main se tiene:



```

package pe.uni.fiis.misiones;

public class Main {
    public static void main(String[] args) {
        Vehiculo miMoto=new Moto(150,2,20,false);
        System.out.println("La Velocidad de la moto es de: "+miMoto.getVelocidad());
        Vehiculo miCarro=new Carro(200,4,30,4);
        System.out.println("La Velocidad del carro es de: "+miCarro.getVelocidad());
        miCarro.frenar();
        miMoto.acelera();
        System.out.println("La Velocidad de la moto es de: " + miMoto.getVelocidad());
        System.out.println("La Velocidad del carro es de: "+miCarro.getVelocidad());
    }
}

```

Notar que los objetos miCarro y miMoto llaman el método getVelocidad de la clase padre Vehiculo.

Bibliografía:

<https://www.youtube.com/watch?v=1d35MF30xDo>

Introducción al lenguaje de programación Java UNAM

Fundamentos de programación en Java (Jorge Martínez Ladrón de Guevara)

http://profesores.fi-b.unam.mx/carlos/java/java_basico4_8.html

<http://jarroba.com/polimorfismo-en-java-parte-i-con-ejemplos/>