FACULTAD DE INGENIERÍA INDUSTRIAL Y SISTEMAS

Ensayo:

HERENCIA Y POLIMORFISMO

Alumno:

- Pariona Chuquiure Juan Martin (20134060B)

Profesor:

- Rony Hancco Carpio

Curso:

- LPOO

Ciclo:

- 2015-I

HERENCIA:

Una subclase hereda de una superclase mediante extenderla (extends). La subclase aparte de tener sus atributos, toma las declaraciones de variables y métodos de la superclase. No olvidar que puede añadir nuevas variables y métodos.

En este caso Moto es la subclase y Vehiculo es la superclase.

```
public class Moto extends Vehiculo {
}
```

La cadena de herencia se refiere a que se pueden crear muchas subclases, es más, una subclase puede tener subclases. Lo recomendable es limitarlo.

A diferencia de la subclase, la superclase es única.

Ejemplo:

```
public class Moto extends Vehiculo  {                    // La clase Moto hereda de la clase Vehiculo
        public void acelera(){
                System.out.println("Moto");
                }
        }

public class Atributos {
        public static void main(String[] args) {
// Creamos un objeto vehic y llama al método acelera()
Vehiculo vehic  = new Vehiculo();
vehic.acelera();

// Creamos un objeto motito y llamamos a ambos métodos acelera() y frena()
Moto motito = new Moto();
motito.acelera();
motito.frena();
```

Palabras clave:

Private: nos ofrece garantía, ya que ninguna otra clase puede acceder al método o a la variable.

Protected: permite que una subclase independiente del paquete en el cual se encuentran, acceda directamente a la superclase del cual hereda. Es controlable.

Public: permite acceso a todas las clases (subclases y superclases). Puede convertirse en peligroso.

Ejemplo:

public class Numeros{

    public Integer a = 1;

    protected Integer b = 2;

    private Integer c = 3;

}

public class Enteros extends Numeros {

    public void ensayo() {

        a = 4; // Variable publica heredada

        b = 5; // Variable protegida heredada

        k = 6; // ERROR. No se puede acceder a la variable

Constructores:

No son heredados ni pueden ser redefinidos. La palabra clave super puede ser usada para explícitamente llamar al constructor de una superclase.

Ejemplo de la clase Moto que hereda de la clase Vehiculo.

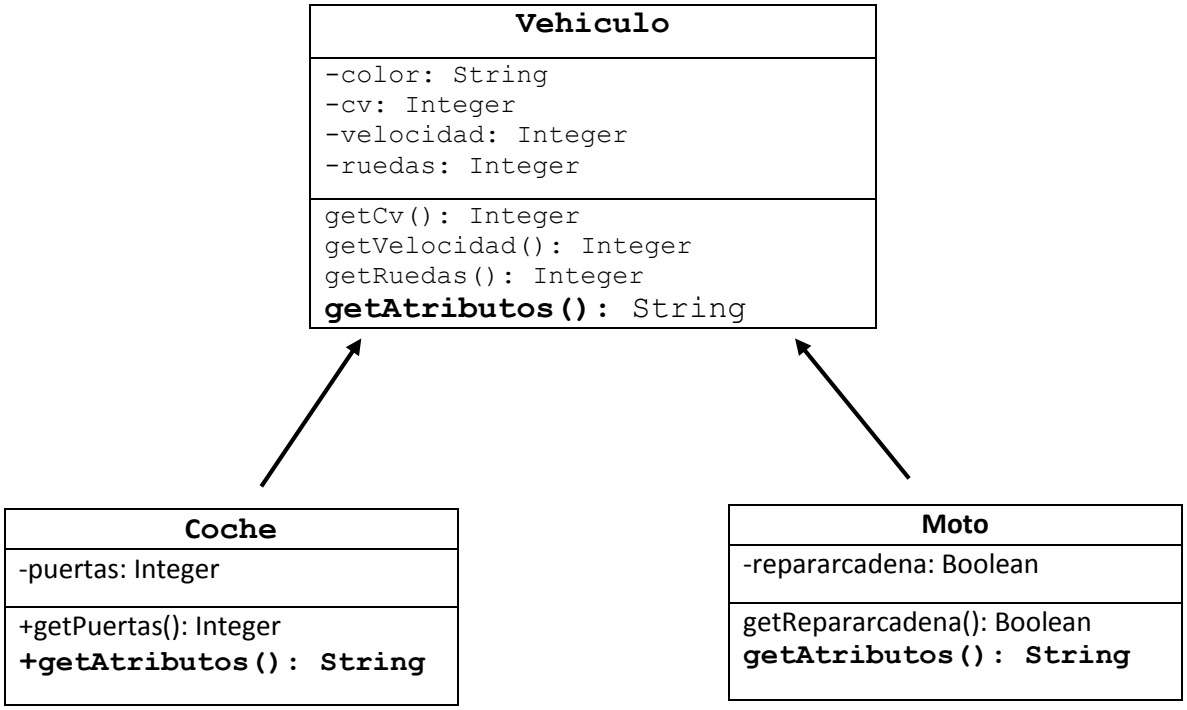
```
//Constructor
public Moto(String color, Integer cv, Boolean reparar_cadena){

    super(color, cv, 2);
    this.reparar_cadena = reparar_cadena;

}
```

POLIMORFISMO:

Es la capacidad del lenguaje de programación (Java) de elegir 1 o más métodos en función del contexto.

Por ejemplo:

Supongamos que la superclase es Vehiculo y hay 2 subclases que son Coche y Moto.



En cada método se invoca a super.getAtributos() para que se muestren los atributos del vehículo y luego los atributos de cada subclase. Estos métodos se sobrescriben el método mencionado de la superclase (Vehiculo). A esto se le conoce como POLIMORFISMO.

Por lo tanto, así queda el comportamiento de los métodos sobrescritos en la superclase y las subclases.

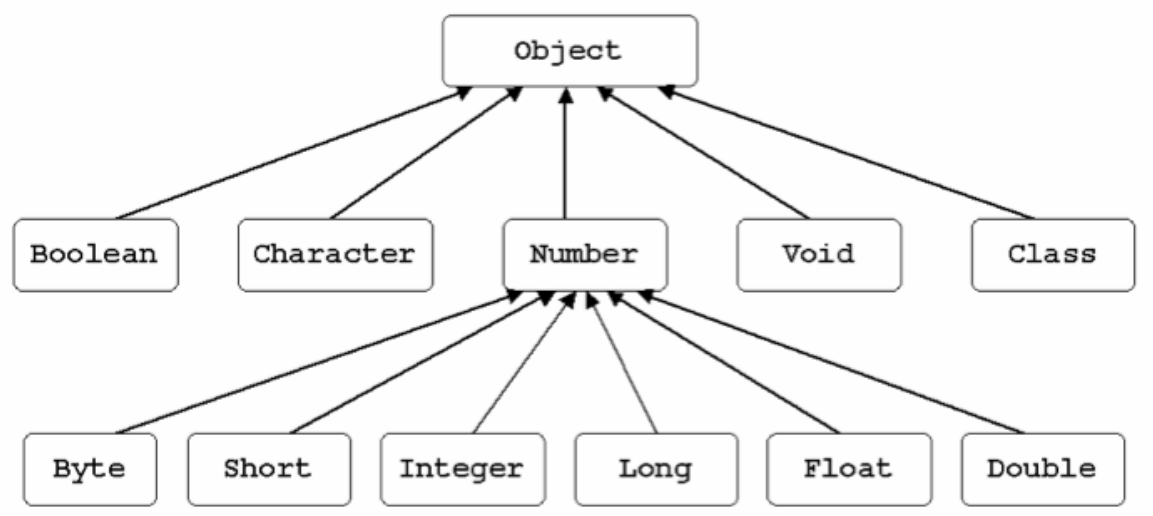
Vehiculo vehic = new Vehiculo (“azul”, 300, 20, 4);

Coche cochecito = new Coche (“verde”, 200, 15, 4, 4); // este último 4 se refiere a las puertas.

Moto motito = new Moto (“rojo”, 100, 10, 2, false); //”false” se refiere a que no necesita reparar.

Jerarquía de herencia:

Toda clase Java puede ser usada como una clase base para extender sus atributos. En este lenguaje de programación todas las clases están relacionadas en una única jerarquía, denominada jerarquía de herencia. Cualquier objeto de un programa desarrollado en Java se ve como una instancia de la clase Object.



Sobrecarga:

Consiste en que puedes tener 2 o más métodos con el mismo nombre, por ejemplo el método "set".

```
public void set (Integer num){
    num1=num;
}

public void set (Double num){
    num2=num;
}

public void set (Integer num1, Double num2){
    this.num1=num1;
    this.num2=num2;
}
```

Java elegirá el método adecuado de acuerdo al contexto.

Primer caso:

```
public class Principal {
    public static void main(String[] args) {
        Numero num = new Numero();
        num.set(10);
        System.out.println("el numero 1 es " + num.getNum1());
    }
}
```

Elegirá el primer método, ya que el número 10 es entero (Integer).

Segundo caso:

```
public class Principal {
    public static void main(String[] args) {
        Numero num = new Numero();

        num.set(5.5);
        System.out.println("el numero 2 es " + num.getNum2());
    }
}
```

Elegirá el Segundo método, ya que el número 5.5 es real (Double)

Tercer caso:

```
public class Principal {
          public static void main(String[] args) {
              Numero num = new Numero();

              num.set(10, 5.5);
              System.out.println("el numero 1 es " + num.getNum1()+ " y
      el numero 2 es " + num.getNum2())
      }
}
```

Elegirá el Tercer método, ya que el método depende de 2 números, uno entero y otro real.