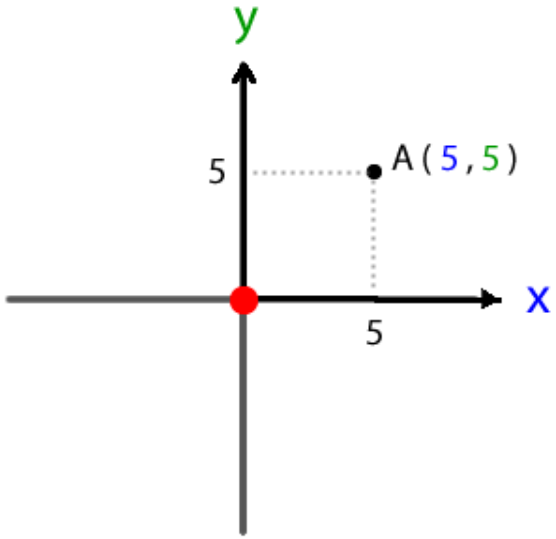
Paradigmas de Programación
IECI

Programación Orientada a Objetos (POO) y Java

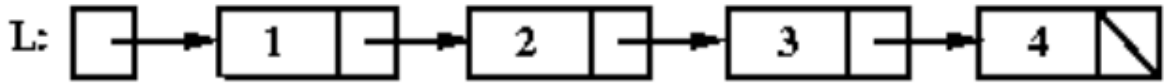
Profesor: Oscar San Martín
osanmartin@ubiobio.cl

Ejercicio

- Cree la clase Coordenada en 2D.



- Cree la clase Lista enlazada simple



Imprimir el estado actual de un objeto

toString: información de un objeto

```
class Coordenada {
  //todo lo demás
  public String toString() {
    return
"("+coordenadaX+","+coordenadaY+")";
  }
}
```

```
//en main
System.out.println(punto1.toString());
```

Pantalla

```
(2.0,3.0)
```

Comparar si estado de dos objetos es el mismo

```
    @Override
    public boolean equals(Object x) {
    if (!( x instanceof Coordenada )) return false;
    else {
        return this.obtenerCoordenadaX( ) == ((Coordenada)x).obtenerCoordenadaX( )
                &&  this.obtenerCoordenadaY( ) == ((Coordenada)x).obtenerCoordenadaY( );
    }
}
```

Crear librerías en Java: paquetes

- *Un paquete se usa para organizar una colección de clases.*
- *La instrucción package indica que una clase será parte de un paquete y debe preceder a la definición de una clase.*
- *La directiva import se usa para poder utilizar un paquete en la implementación de una clase.*
- *El paquete java.lang.* se importa automáticamente.*

Herencia:

- *Un objetivo de la POO es la reutilización de código.*
- *En un lenguaje de POO el mecanismo básico de reutilización de código es la herencia.*
- *La herencia es el principio básico de la programación orientada a objetos empleado para reutilizar código entre clases relacionadas.*
- *La herencia modela las relaciones ES-UN.*
- *Por ejemplo, un Círculo ES-UNA Figura y un Coche ES-UN Vehículo.*
- *Las relaciones de herencia forman jerarquías.*

Herencia:

- *La herencia permite derivar clases a partir de una clase base sin modificar la implementación de esta última.*
- *Cada clase derivada es una clase completamente nueva, compatible con la clase base de la que deriva.*
- *Si X ES-UN Y entonces X es una subclase de Y e Y es una superclase de X. Ambas relaciones son transitivas.*

Herencia:

```
public class Hija extends Padre {

}
```

Herencia

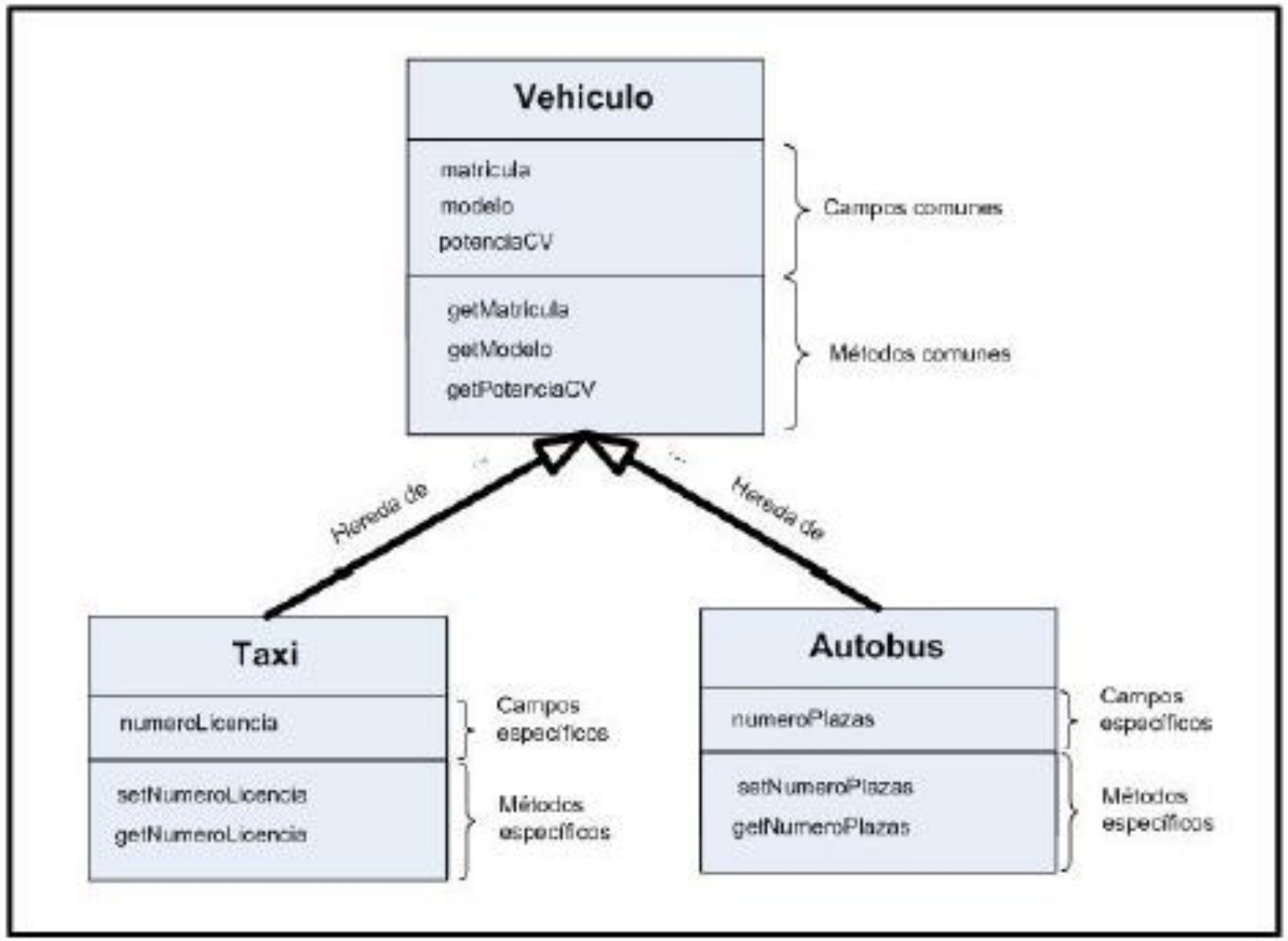
```
public class Persona {
    public String nombre;
    public String rut;
    public int edad;

    public Persona(String n, String r, int e) {
        nombre=n;
        rut=r;
        edad=e;
    }

    @Override  public boolean equals(Object x) {
        Persona aux;
        if ( x instanceof Persona) {
            aux=(Persona) x;
            return this.toString().equals(aux.toString());
        }return false;
    }
    @Override
    public String toString() {
        return " Rut "+rut+" , Nombre "+nombre+" , Edad "+edad;
    }

}
```

```
public class Alumno extends Persona {

    public double nota_media;

    public Alumno(String n, String r, int e,double nota_media) {
        super(n,r,e);
        this.nota_media=nota_media;
    }

    //NECESITO SOBRE ESCRIBIR EL METODO TO STRING  PARA MOSTRAR LA NOTA MEDIA

    @Override
    public String toString() {
        return super.toString()+", Nota "+nota_media+" ; ";
    }

}
```

Método super

- *Si no se implementa ningún constructor se genera por defecto un constructor sin parámetros.*
- *Dicho constructor invoca al constructor sin parámetros de la clase base para inicializar la porción heredada, mientras que para los atributos adicionales se emplea la inicialización por defecto.*
- *super se emplea para llamar al constructor de la clase base y hace referencia a la clase padre.*

Ejercicio


Vehiculo
matricula
modelo
potenciaCV
getMatricula
getModelo
getPotenciaCV
Campos comunes
Métodos comunes
Hereda de
Hereda de
Taxi
numeroLicencia
setNumeroLicencia
getNumeroLicencia
Campos específicos
Métodos específicos
Autobus
numeroPlazas
setNumeroPlazas
getNumeroPlazas
Campos específicos
Métodos específicos