Programación Orientada a Objetos

Dra. Maricela Claudia Bravo Contreras

mari_clau_18@hotmail.com

Principios de la Programación Orientada a Objetos

Principios de La POO

- Son las propiedades fundamentales que orientan el estilo de programación que permite crear código reutilizable, usable y de fácil mantenimiento.

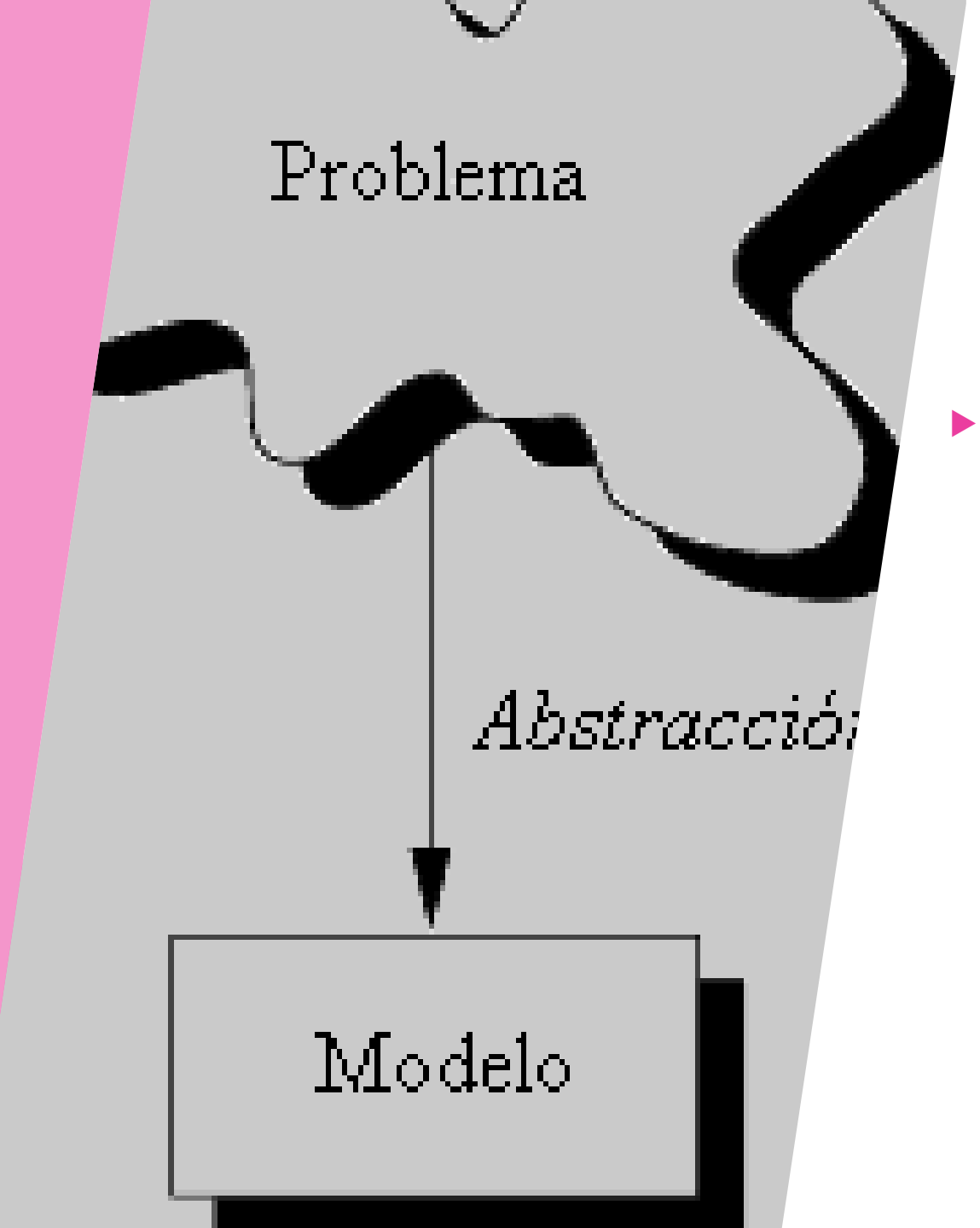
1. Principio de abstracción
2. Principio de encapsulamiento
3. Principio de modularidad
4. Principio de paso de mensajes
5. Principio de herencia
6. Principio de polimorfismo

Principio de Abstracción

Abstraer es suprimir u ocultar algunos detalles de un proceso o de un elemento, para resaltar algunos aspectos, detalles o estructuras.

La abstracción de objetos se refiere a la vista externa del objeto (interfaces).

La abstracción es la forma en que nuestra mente modela la realidad, formando los objetos.



Abstracción

- El modelo define una ***perspectiva abstracta*** del problema
  - Los ***datos*** que son afectados
  - Las ***operaciones*** que se aplican sobre los datos

Encapsulamiento

- Proceso por el que se ocultan:
  - Las estructuras de datos
  - Los detalles de la implementación
  - Permite considerar a los objetos como "cajas negras", evitando que otros objetos accedan a detalles que NO LES INTERESA
  - Una vez creada la clase, las funciones usuarias no requieren conocer los detalles de su implementación

Encapsulamiento

Toda clase tiene un conjunto de ***atributos*** y ***métodos*** asociados.

Todos ellos están ***encapsulados*** o contenidos dentro de la misma clase, de manera que son ***miembros*** de dicha clase.

Esos métodos y atributos pueden ser utilizados por ***otras*** clases ***sólo si*** la clase que los encapsula les brinda los ***permisos*** necesarios para ello.

Encapsulamiento

A los métodos también se les puede aplicar distintos modificadores de acceso, por lo que un método puede ser también marcado como **private**.

La recomendación general es inicialmente hacer todo privado e irlo haciendo público conforme se va necesitando.

Entre menos métodos públicos tenga una clase es más fácil de entender.

No se recomienda tener atributos públicos en lo absoluto.

Ejemplo Encapsulamiento

```
public class Animal
{
  // Definición de atributos privados
   private int numOjos;
   private String formaMoverse;
   private String tipoRespiracion;
   private String tipoReproduccion;
   private String haceRuido;
```

Ejemplo Encapsulamiento

```
//Método constructor
public Animal(int numOjos, String formaMoverse,
       String tipoRespiracion,
       String tipoReproduccion,
       String haceRuido)
  {
   super();
   this.numOjos = numOjos;
   this.formaMoverse = formaMoverse;
   this.tipoRespiracion = tipoRespiracion;
   this.tipoReproduccion = tipoReproduccion;
   this.haceRuido = haceRuido;
  }
```

Ejemplo Encapsulamiento

```
public int getNumOjos() {
  return numOjos;}

public String getFormaMoverse() {
  return formaMoverse;}

public String getTipoRespiracion() {
  return tipoRespiracion;}

public String getTipoReproduccion() {
  return tipoReproduccion;}

public String getHaceRuido() {
  return haceRuido;}
```

Métodos de acceso a los atributos

Principio de Herencia

Organización Jerárquica

Herencia

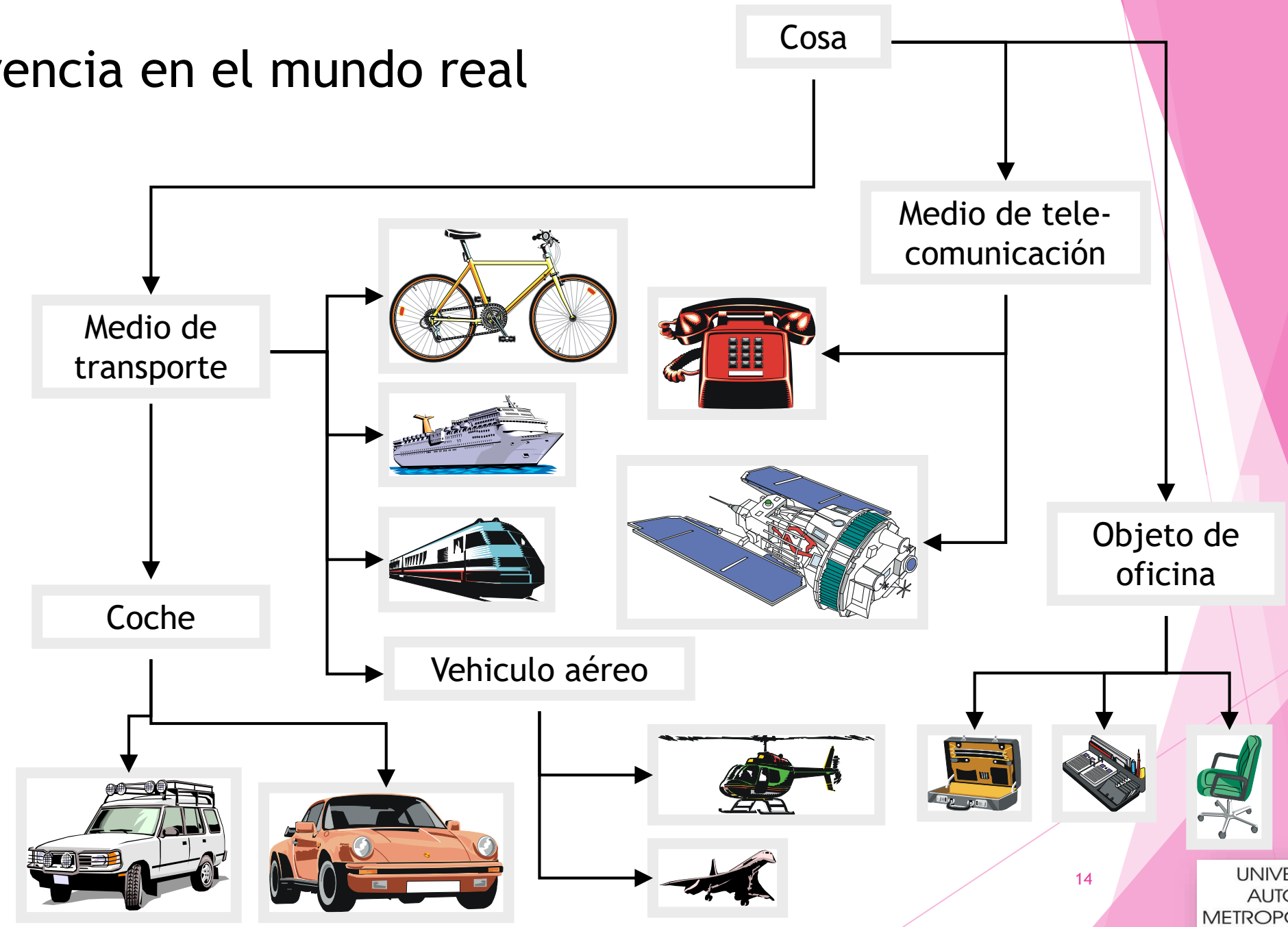
La herencia es un mecanismo que permite la definición de una clase a partir de la definición de otra ya existente.

La herencia permite compartir automáticamente métodos y atributos entre clases, subclases y objetos.

Permite reutilizar código creando nuevas clases a partir de las existentes (construidas y depuradas).

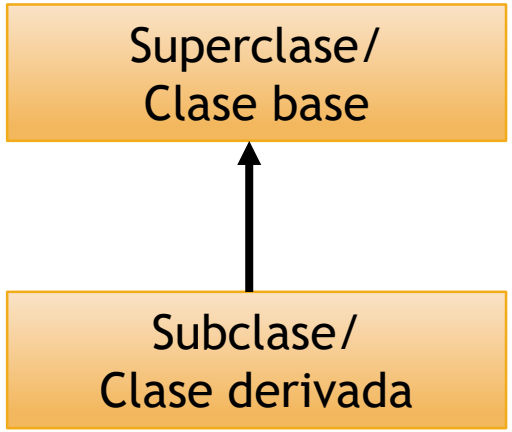
Compromete una relación de jerarquía (es-un).

Herencia en el mundo real



Herencia

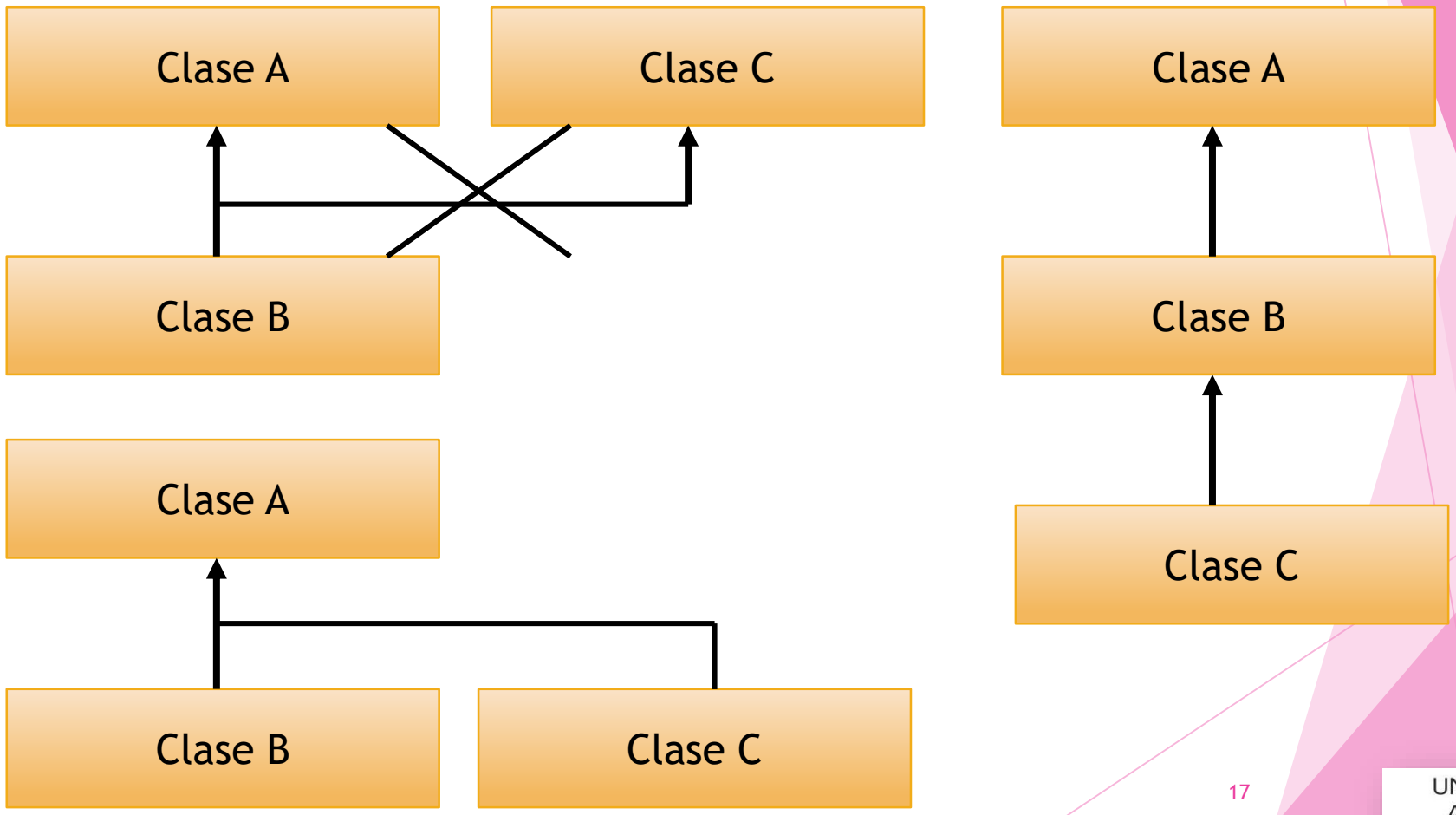
- En POO, la clase de la que se va a heredar se llama superclase o clase base. Mientras que la que hereda se le conoce como subclase o clase derivada.



Herencia

- Reglas básicas para la herencia en Java
  - No está permitida la herencia múltiple
  - Si es posible la herencia multinivel, A puede ser heredada por B y C puede heredar de B
  - Una clase puede ser heredada por varias clases.

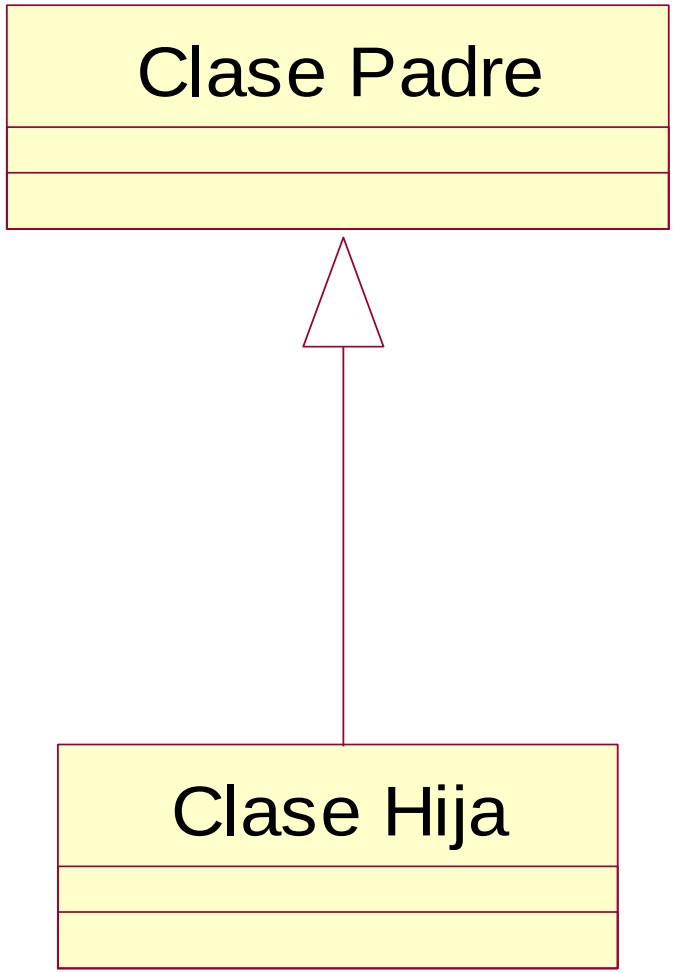
Herencia



Herencia en Java

- Java permite definir una clase como subclase de una clase padre.

```
class clase_hija extends
    clase_padre
{
.....
}
```

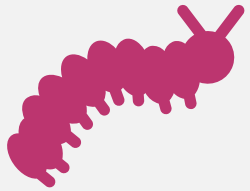


Constructores y Herencia

- Cuando se declara un objeto de una clase derivada, se ejecutan los constructores siguiendo el orden de derivación, es decir, primero el de la clase base, y después los constructores de las clases derivadas de arriba a abajo.
- Para pasar parámetros al constructor de la clase padre:

super *(para1, para2, ..., paraN)*

Ejemplo 1 de Herencia: el Zoológico

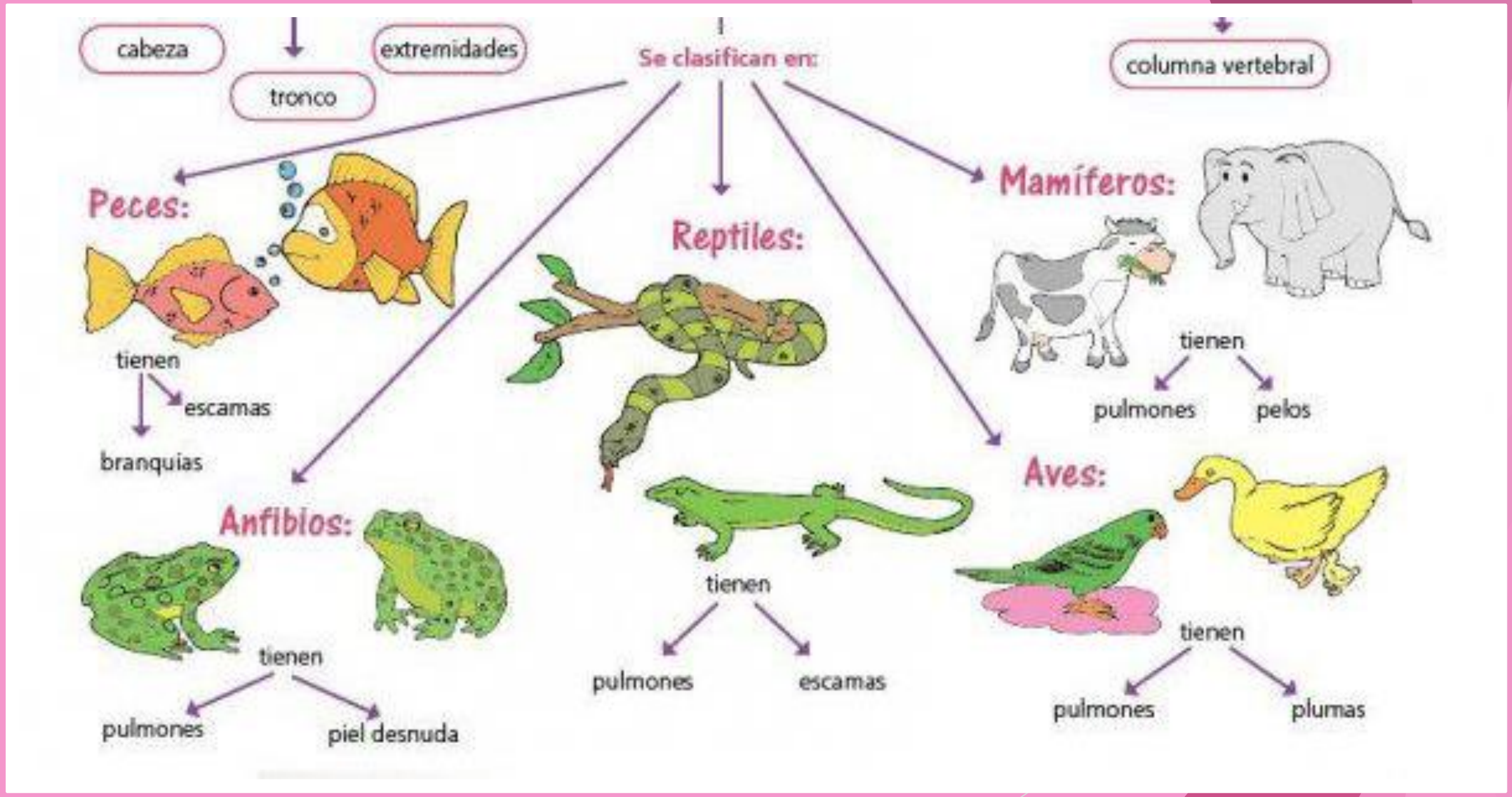


Se requiere el desarrollo de un programa que proporcione información básica sobre todos los animales:

- Identificación, nombre y edad
- Región donde habita
- Tipo de alimentación
- Forma de reproducción
- Forma de moverse
- Forma de comunicarse

Además se deberá proporcionar información específica de cada animal de acuerdo a la clasificación de animales vertebrados a la que pertenece: medio de respiración (branquias o pulmones) y tipo de piel (escamas, plumas o pelos).

- Peces
- Anfibios
- Reptiles
- Aves
- Mamíferos


cabeza
tronco
extremidades
Se clasifican en:
columna vertebral
Peces:
tienen
escamas
branquias
Anfibios:
tienen
pulmones
piel desnuda
Reptiles:
tienen
pulmones
escamas
Mamíferos:
tienen
pulmones
pelos
Aves:
tienen
pulmones
plumas

Sobreescritura de métodos en la Herencia

- Una subclase hereda todos los métodos de su superclase que son accesibles a dicha subclase a menos que la subclase sobreescriba los métodos.

- Una subclase sobreescribe un método de su superclase cuando define un método con las mismas características (nombre, número y tipo de argumentos) que el método de la superclase.

- Las subclases emplean la sobreescritura de métodos la mayoría de las veces para agregar o modificar la funcionalidad del método heredado de la clase padre.

Casting en Java

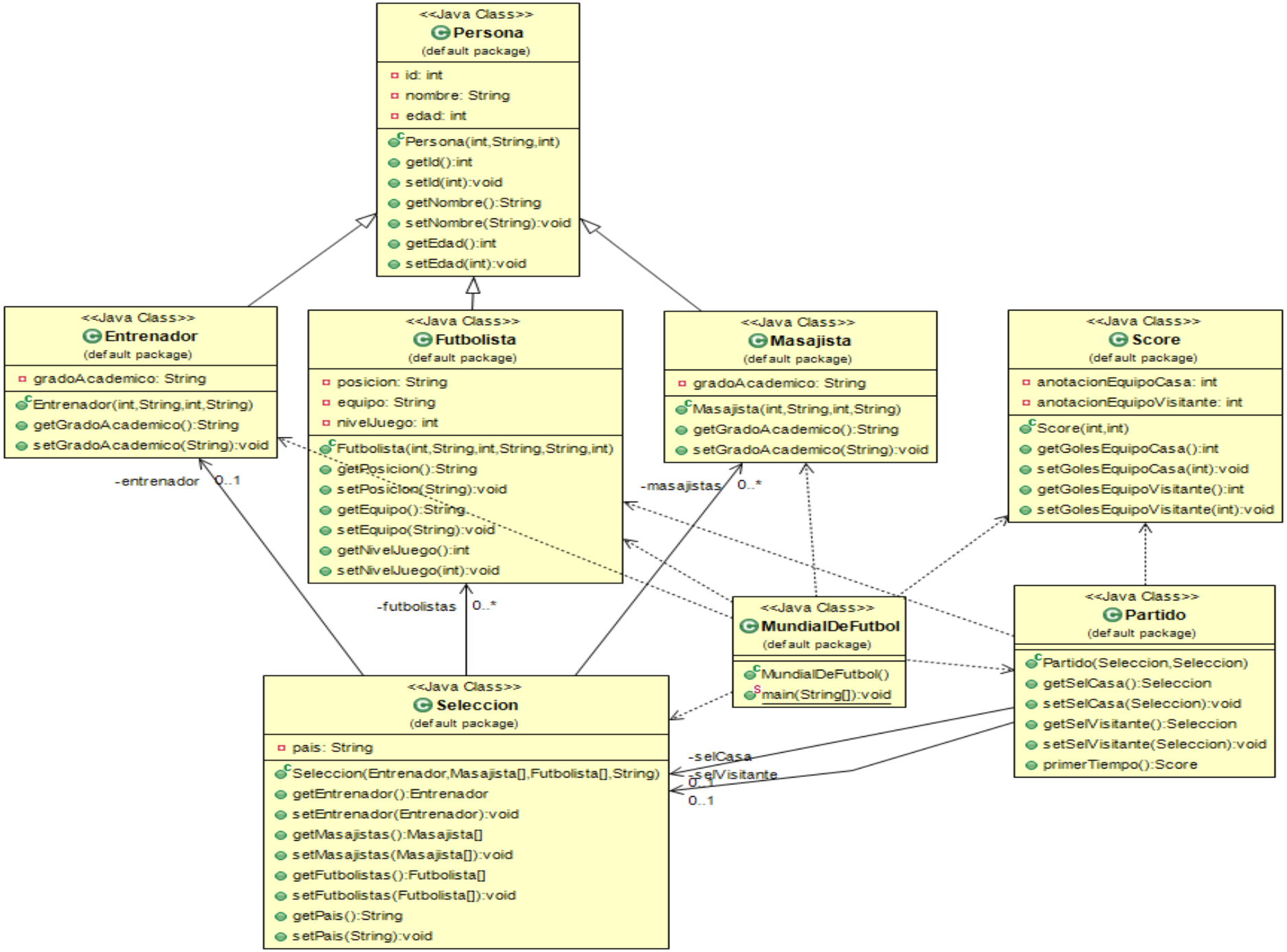
- El casting es un procedimiento para transformar una variable primitiva de un tipo a otro
- O para transformar un objeto de una clase a otra clase siempre y cuando haya una relación de herencia entre ambas

Ejemplo 2 de Herencia: la Selección Mexicana de Futbol

Simular el comportamiento que tendrían los diferentes integrantes de la selección mexicana de futbol; tanto los Futbolistas como el cuerpo técnico (Entrenadores, Masajistas, etc…).

Para simular este comportamiento se requieren tres clases que van a representaran objetos tipo Futbolista, Entrenador y Masajista.

De cada unos de estos objetos se requiere definir los atributos y una serie de acciones que reflejaremos en sus métodos.



UNIVERSIDAD AUTONOMA METROPOLITANA
Casa abierta al tiempo
Azcapotzalco

Ejemplo 3 de Herencia: alquiler de vehículos

- Se necesita desarrollar un programa en Java que permita calcular los precios de renta de una empresa de alquiler de vehículos.
- Cada vehículo se identifica de manera única por su número de placa. La empresa renta diferentes tipos de vehículos, tanto para transporte de personas como de carga.
- Los vehículos alquilados por la empresa son coches, microbuses, camionetas de carga y camiones. El precio base de alquiler de cualquier vehículo se calcula a razón de $500.00 pesos diarios.
- En el caso de alquiler de un coche, al precio base se le suma la cantidad de $20.00 pesos por número de asientos.
- El alquiler de microbuses es igual al de los coches, solo que se añade una cantidad de $30.00 pesos por número de asientos independientemente de los días de alquiler.
- El precio de los camiones de carga es el precio base más $30.00 X PMA (PMA es el peso máximo autorizado en toneladas).
- En el caso de las camionetas, al precio base se le suma un precio fijo de $550.00 pesos independientemente de los días de alquiler.

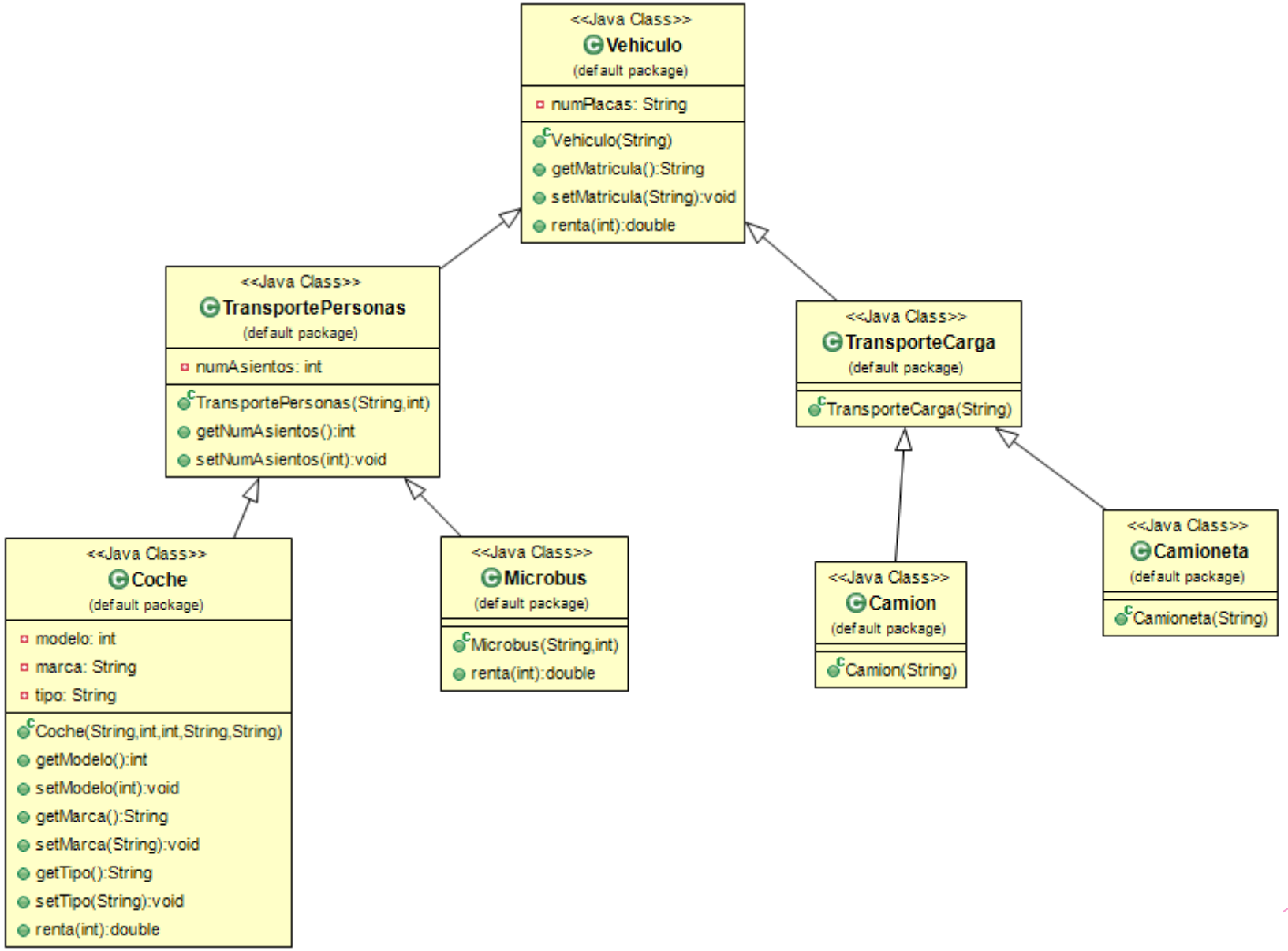


Diagrama de clases

Principio de polimorfismo

Polimorfismo

- Capacidad que permite a dos clases diferentes responder de forma distinta a un mismo mensaje
- Esto significa que dos clases que tengan un método con el mismo nombre y que respondan al mismo tipo de mensaje (es decir, que reciban los mismo parámetros), ejecutarán acciones distintas

Polimorfismo

```
public class Vehiculo
{
        private String noPlaca;

        public Vehiculo(String noPlaca)
        {
                this.noPlaca = noPlaca;
        }

        public float rentaBase(int dias)
        {
                float renta = 500;
                return renta * dias;
        }

}
```

```
public class TransportePersona extends Vehiculo
{
        private int noPlazas;

        public TransportePersona(String noPlaca, int noPlazas)
        {
                super(noPlaca);
                this.noPlazas = noPlazas;
        }

}
```

```
public class Coche extends TransportePersona
{
        private int modelo;
        private String marca;
        private String gama;

        public Coche(String noPlaca, int noPlazas, int modelo, String marca, String gama)
        {
                super(noPlaca, noPlazas);
                this.modelo = modelo;
                this.marca = marca;
                this.gama = gama;
        }

        @Override
        public float rentaBase(int dias)
        {
                float adicional = this.getNoPlazas() * 20 * dias;
                return super.rentaBase(dias) + adicional;
        }

}
```

UNIVERSIDAD AUTONOMA METROPOLITANA
Casa abierta al tiempo Azcapotzalco

Principio de Modularidad

Principio de Modularidad

- La modularidad permite dividir un problema complejo en varios módulos o partes diferentes, donde cada módulo resolverá una parte de un problema grande, y después interactuarán todos los módulos.
- Cada módulo debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes.
- Se debe tener en cuenta los conceptos asociados de dependencia, acoplamiento, cohesión, interfaz, encapsulación y abstracción

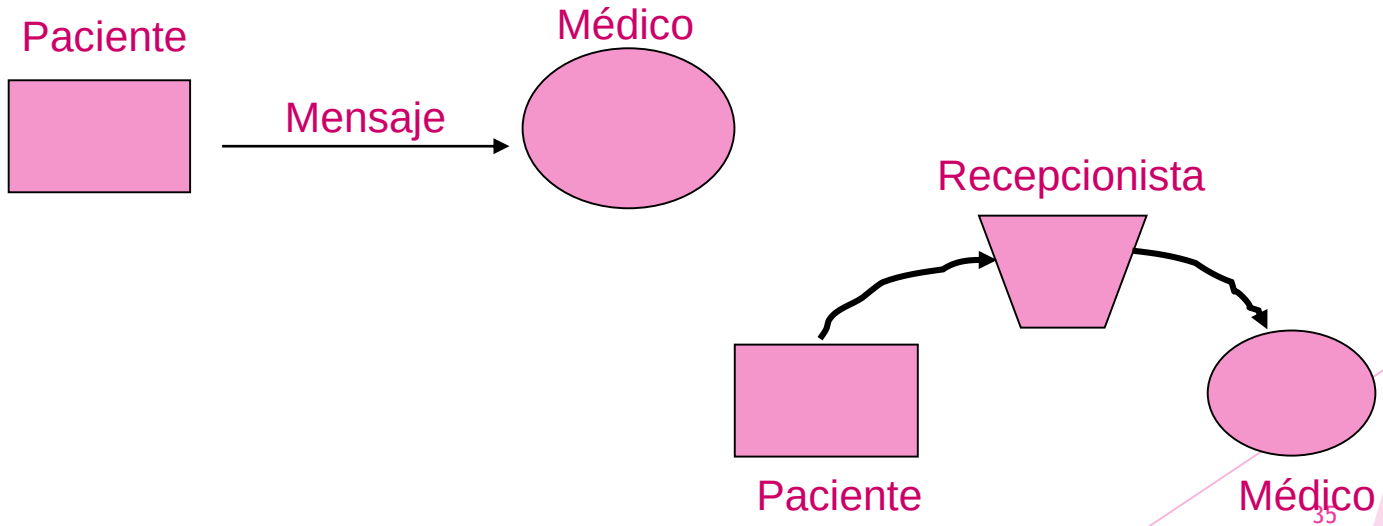
Principio de paso de mensajes

Mensaje

- Mecanismo por el cual ***se solicita*** una acción sobre el objeto
- Un programa en ejecución es una colección de objetos que se crean, ***interactúan*** y se destruyen
- La ***interacción*** se basa en ***mensajes*** que son enviados de un objeto a otro, de modo que el emisor le pide al receptor la ejecución de un método

Mensajes

- Un objeto invoca un método como una reacción al recibir un mensaje
- La interpretación del mensaje dependerá del receptor



Mensajes

- Con el paso de mensajes los objetos pueden solicitar a otros objetos que realicen alguna acción o que modifiquen sus atributos.
- Junto con el paso de mensajes se implementan llamadas a los métodos de un objeto, lo que nos permite hacer que ese objeto realice determinada acción que está dentro de su comportamiento.

Dudas O comentarios???

UNIVERSIDAD AUTONOMA METROPOLITANA
Casa abierta al tiempo Azcapotzalco