El futuro digital es de todos

MinTIC

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Mision TIC 2022

SEMANA 2

INICIAMOS 8:05PM

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Luisa Fernanda Restrepo.

Agenda

- Introducción Clases y Objetos
- UML
- Clases y objetos en Java
- Constructores
- Sobrecarga
- This
- Contantes y static

Introducción Clases y Objetos

Luisa Fernanda Restrepo.

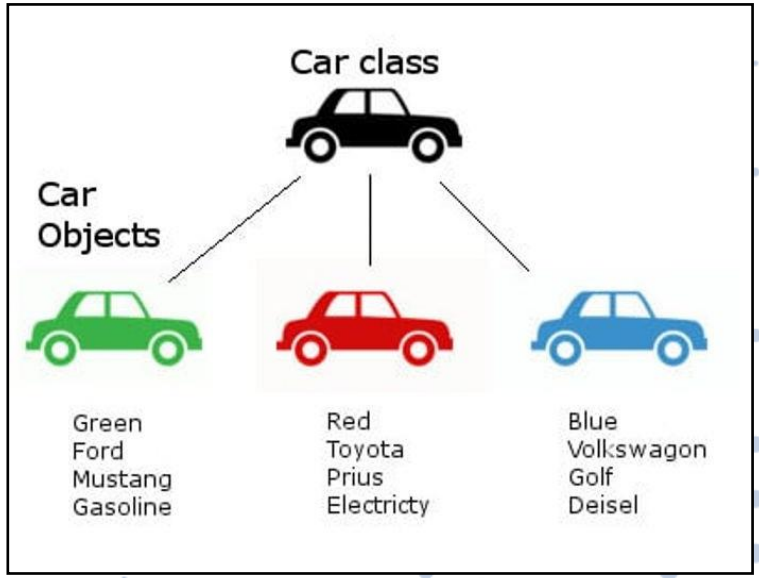
POO

¿Sabes que es la programación orientada a objetos?

- La programación orientada a objetos (POO) es esencialmente una tecnología para desarrollar software reutilizable.

POO – Características

- La POO organiza los programas de manera que representan la interacción de las cosas en el **mundo real**.
- En la POO un programa consta de un conjunto de **objetos**.
- En la POO los objetos son **abstracciones** de cosas del mundo real.
- En la POO cada objeto es responsable de unas **tareas**.
- En la POO cada objeto es un ejemplar (instancia) de una **clase**.
- En la POO las clases se pueden organizar en una jerarquía de **herencia** (se ve mas adelante).
- La programación OO es una simulación de un modelo del universo.



POO – Objetos

- La POO implica programar utilizando **objetos**. Un objeto representa una entidad del mundo real que puede **identificarse** claramente.

- **Por ejemplo:** un estudiante, un escritorio, un círculo, una casa, un carro, cualquier objeto en especifico.

- Cada objeto tiene una identidad única, unas propiedades (**atributos**), y unos comportamientos (**métodos**).

POO – Objetos – Propiedades

Propiedades (atributos):

- Representan datos con valores pertenecientes al objeto.
- **Por ejemplo:** un objeto estudiante podría tener: nombre, edad, sexo, correo, entre otros.

Mencione algunas propiedades que podría tener un objeto "Carro"

Mencione algunas propiedades que podría tener un objeto "Circulo" – "Rectangulo"

POO – Objetos – Comportamientos

Comportamientos (métodos):

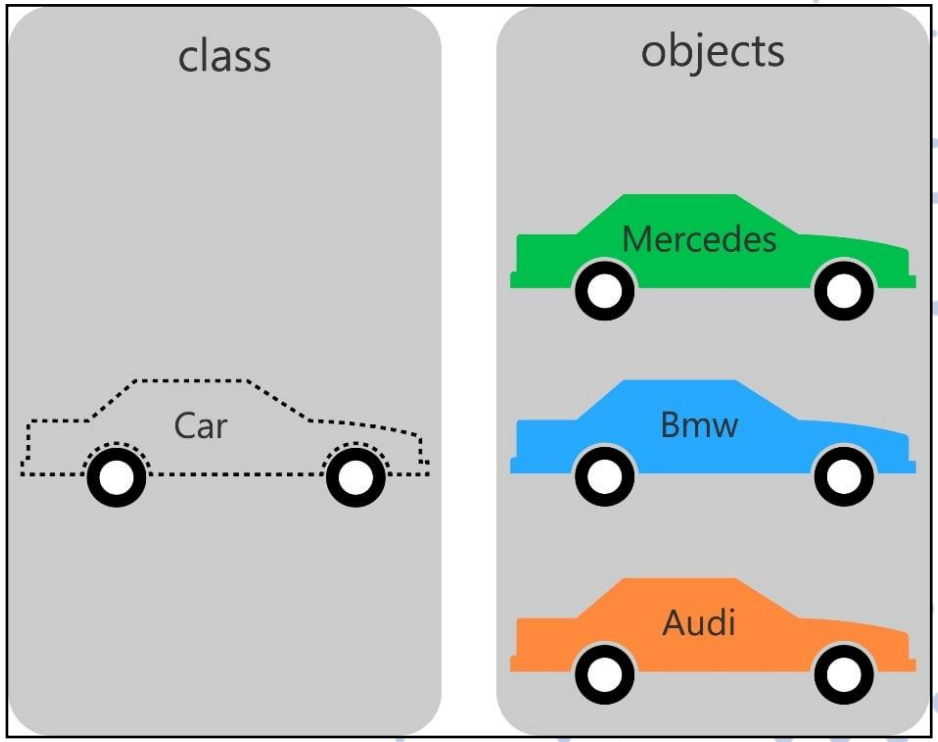
- Representan acciones o métodos. Cada objeto puede invocar o ejecutar diferentes tipos de acciones.
- **Por ejemplo:** un objeto circulo podría invocar métodos como getArea() – para obtener el área del circulo. O getPerimeter() para obtener el perímetro.

Mencione algunos comportamientos que podría tener un objeto "Carro"

encender()
apagar()
tanquear()
obtenerCombustibleActual()

POO – Clases

- Los objetos del mismo tipo se definen usando una clase común.
- Una **clase** es una plantilla, modelo o contrato que define cuáles serán los atributos y métodos de un objeto.
- Un objeto es una **instancia** de una clase.
- Un programador puede crear muchas instancias de una clase.
- Crear una instancia se conoce como instanciación.



UML

Luisa Fernanda Restrepo.

UML

- UML (lenguaje unificado de modelado) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad.
- La ventaja de utilizar modelos UML, es que permiten que programadores de diferentes regiones o incluso continentes, “hablen” en un mismo idioma.
- Existen muchos tipos de modelos o diagramas UML. En este curso nos enfocaremos en el “diagrama de clases UML”.

Para qué usarlo…

- Como medio para facilitar la discusión sobre el sistema existente o propuesto;
- Como una forma de documentar un sistema existente;
- Como una descripción detallada del sistema que permita generar una implementación del sistema;
- Para aclarar lo que el sistema hace;
- Permite conducir los **requisitos** para etapas posteriores del proyecto;
- Explicar los **requisitos** propuestos a otros participantes;
- Discutir el sistema y documentarlo para después ser implementado.

Modelos en UML

Modelos estructurales

- Diagrama de clases
- Diagrama de componentes
- Diagrama de despliegue

Modelos de comportamiento

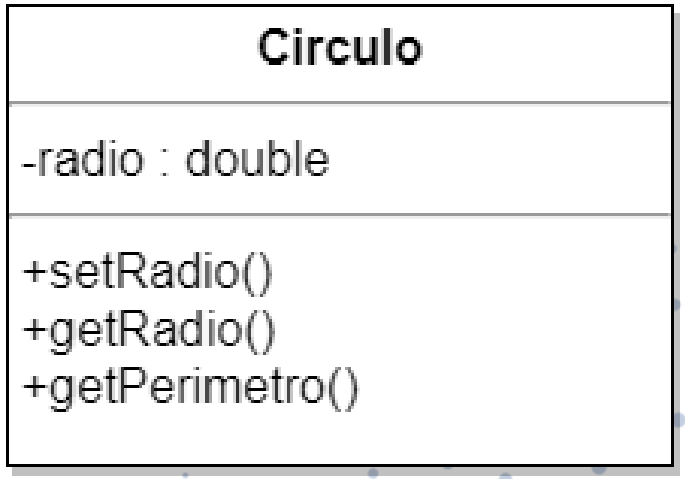
- Diagrama de casos de uso
- Diagrama de actividades
- Diagrama de máquinas de estado

Modelos de interacción

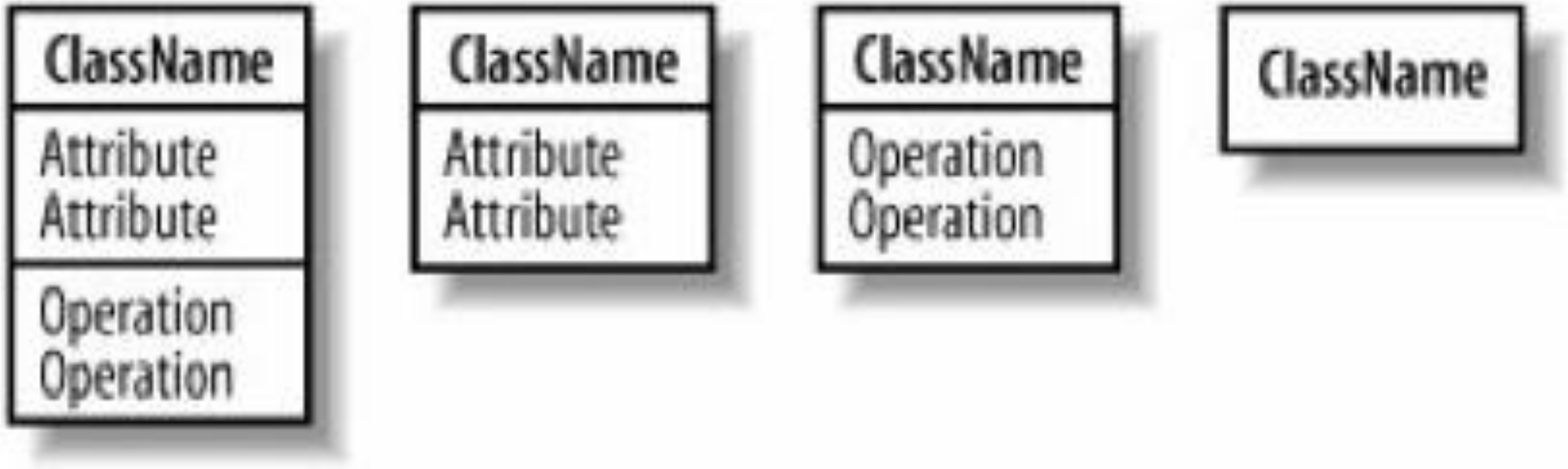
- Diagrama de secuencia
- Diagrama de comunicación

UML – Diagrama de clases

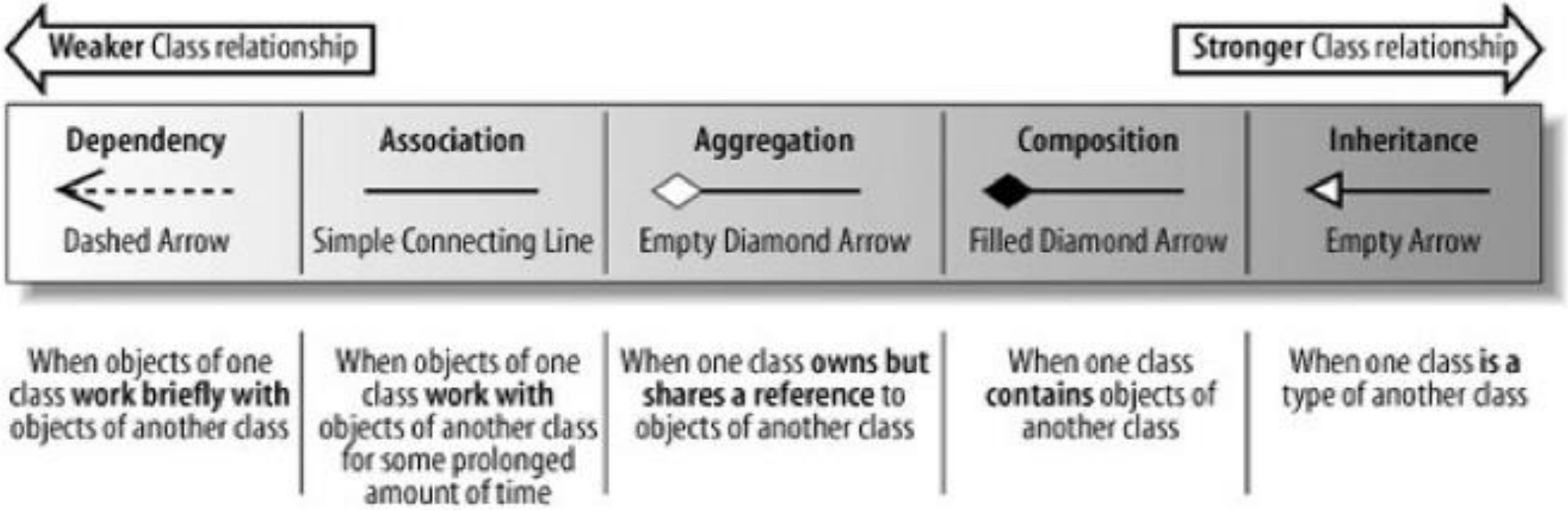
- Se utiliza para representar la clases del sistema.
- Permite obtener un panorama general de mi aplicación sin necesidad de ir a mirar 200 archivos o más.
- Permite observar de manera grafica como se relacionan las diferentes clases de mi sistema (se ve mas adelante).
- Permite establecer un diseño de mi aplicación y presentársela a otros colegas o interesados.

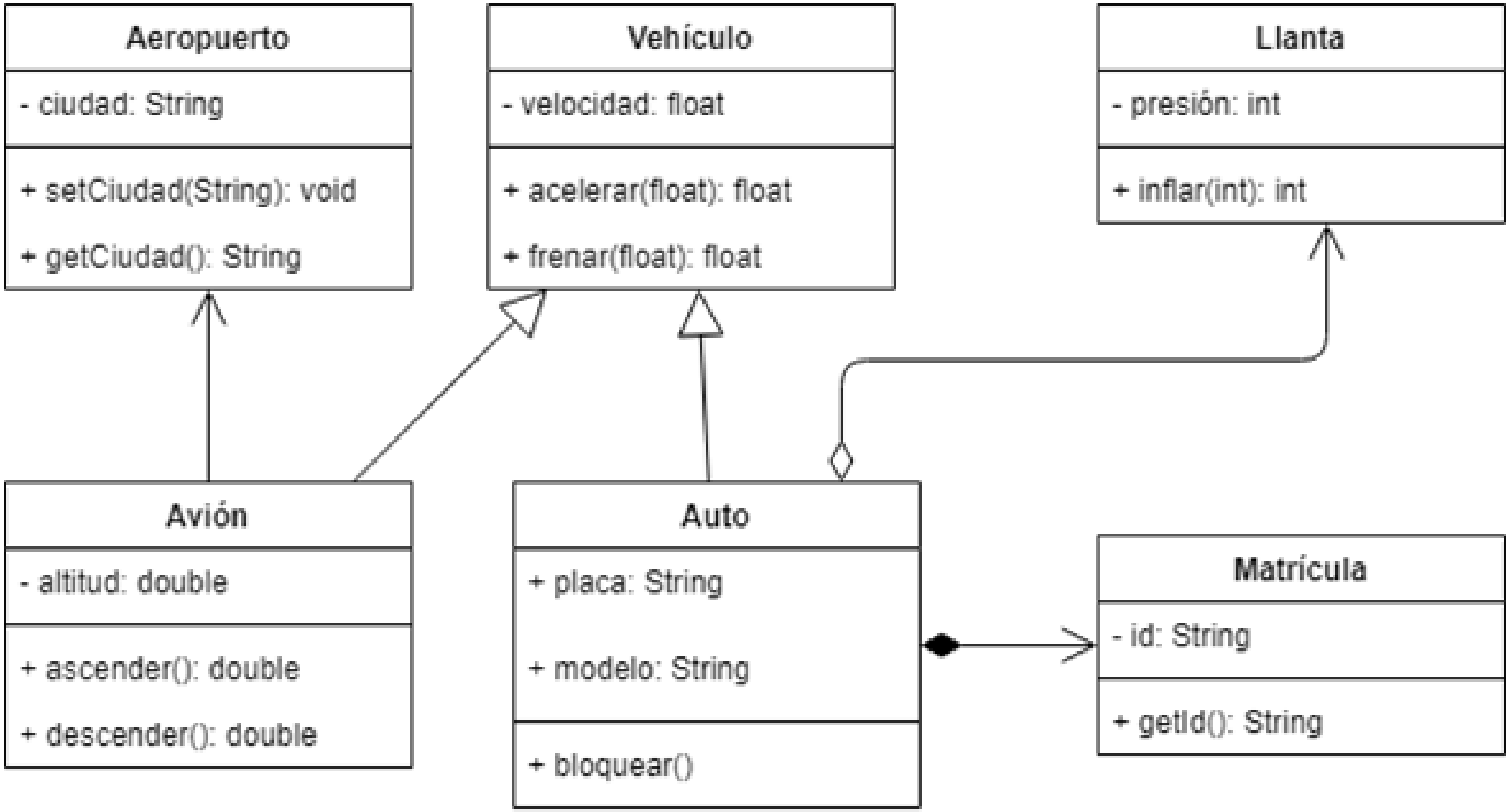


4 Diferentes formas de mostrar una clase



Tipos de relaciones




Aeropuerto
- ciudad: String
+ setCiudad(String): void
+ getCiudad(): String
Vehículo
- velocidad: float
+ acelerar(float): float
+ frenar(float): float
Llanta
- presión: int
+ inflar(int): int
Avión
- altitud: double
+ ascender(): double
+ descender(): double
Auto
+ placa: String
+ modelo: String
+ bloquear()
Matrícula
- id: String
+ getId(): String

Clases en UML

- En su forma más simple, una clase en UML se dibuja como un rectángulo dividido en hasta tres secciones.
  - La sección superior contiene el nombre de la clase (centrado y singular).
  - La sección central contiene los atributos o propiedades de la clase.
    - ***visibilidad nombre_Atributo : tipo_variable***
  - La sección final contiene los métodos que representan el comportamiento que exhibe la clase.
    - ***visibilidad nombreMétodo(parámetros) : tipoRetorno***

Nota: Las secciones de atributos y operaciones son opcionales.

UML – Visibilidad

¿Cómo revela una clase sus métodos y atributos a otras clases?

- **R://** Usando **visibilidad**
- Hay cuatro tipos diferentes de visibilidad que se pueden aplicar a los atributos y métodos de una clase (Publica "+", Protegida, de Paquete y Privada "-").

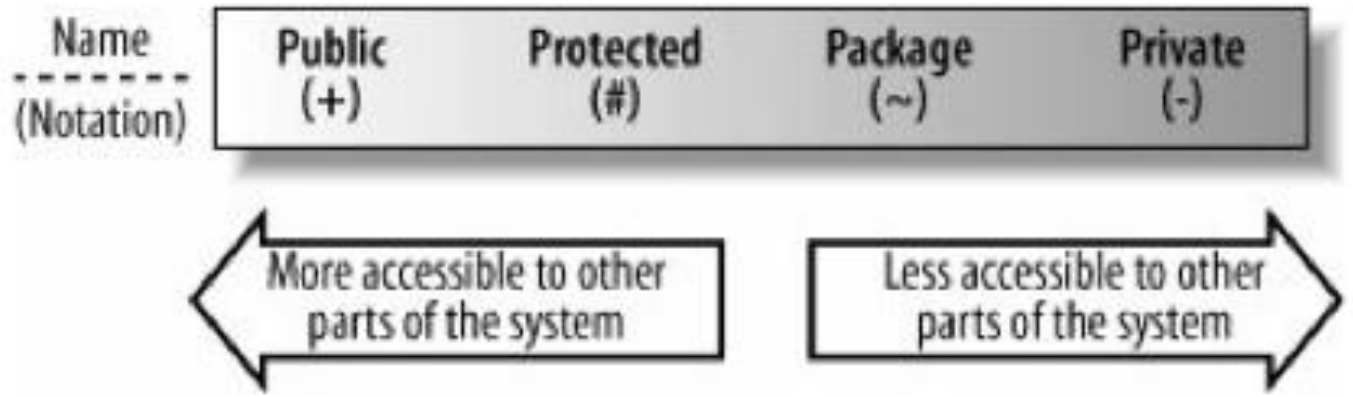
Por el momento solo trabajaremos con visibilidad publica (+) y privada (-).

Por el momento le sugerimos que cuando haga un diagrama de clases, ponga todos los atributos de todas las clases con "-" y todos los métodos con "+".

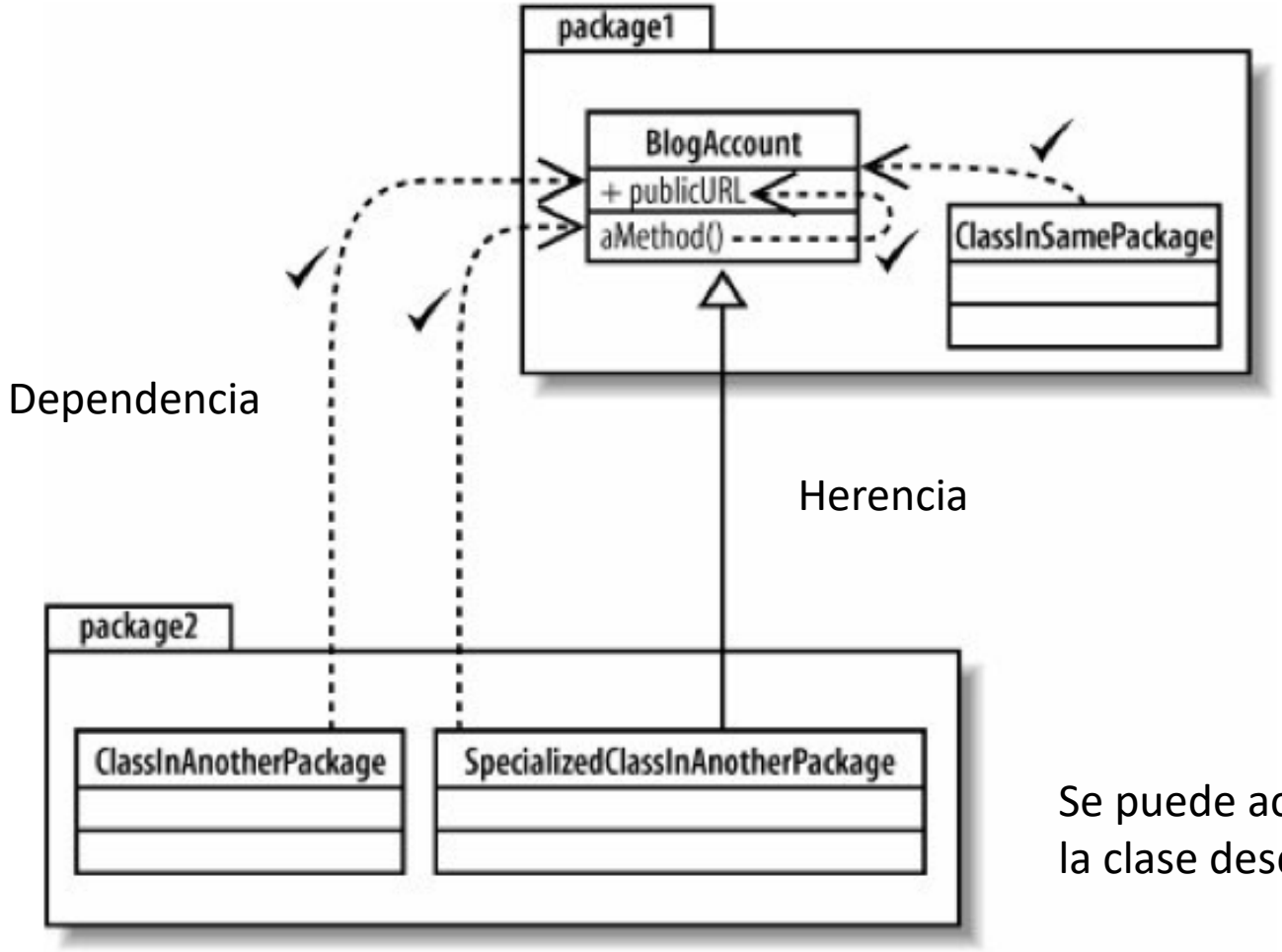
Table 3. Visibility Options on UML Class Diagrams

Visibility	Symbol	Accessible to
Public	+	All objects within your system
Protected	#	Instances of the implementing class and its subclasses
Private	-	Instances of the implementing class
Package	~	Instances of classes within the same package

4 diferentes tipos de visibilidad

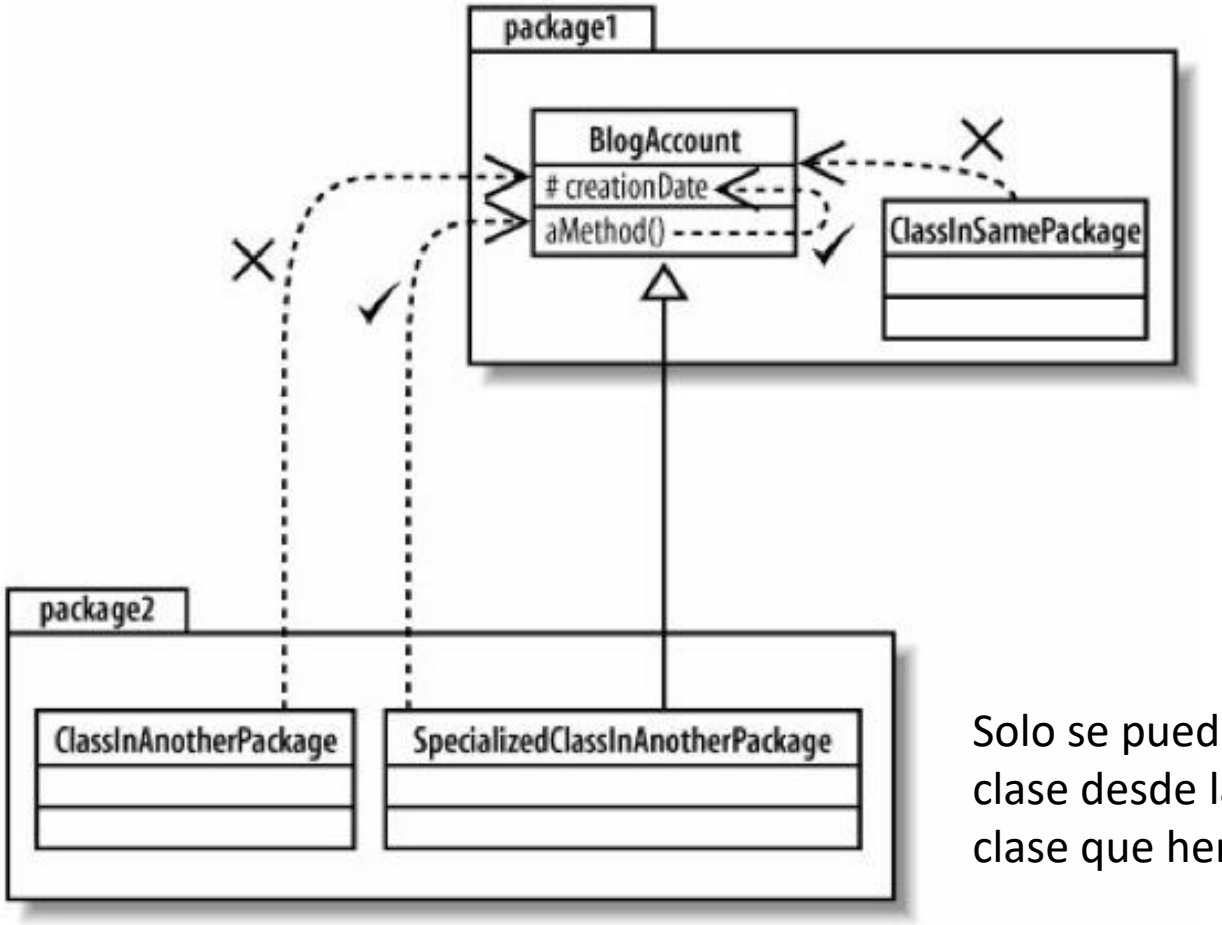


Visibilidad Pública



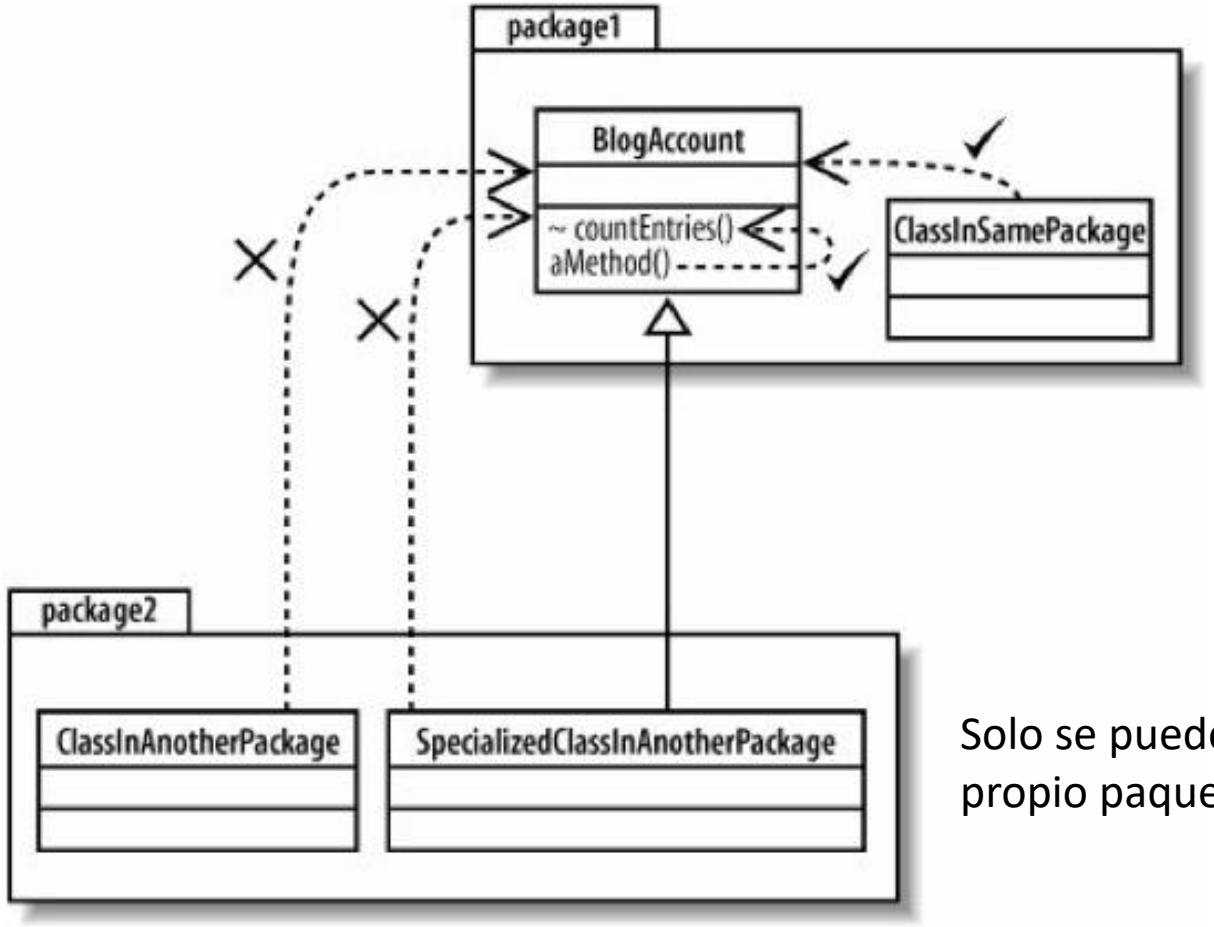
Se puede acceder al miembro de la clase desde cualquier lugar

Visibilidad Protegida



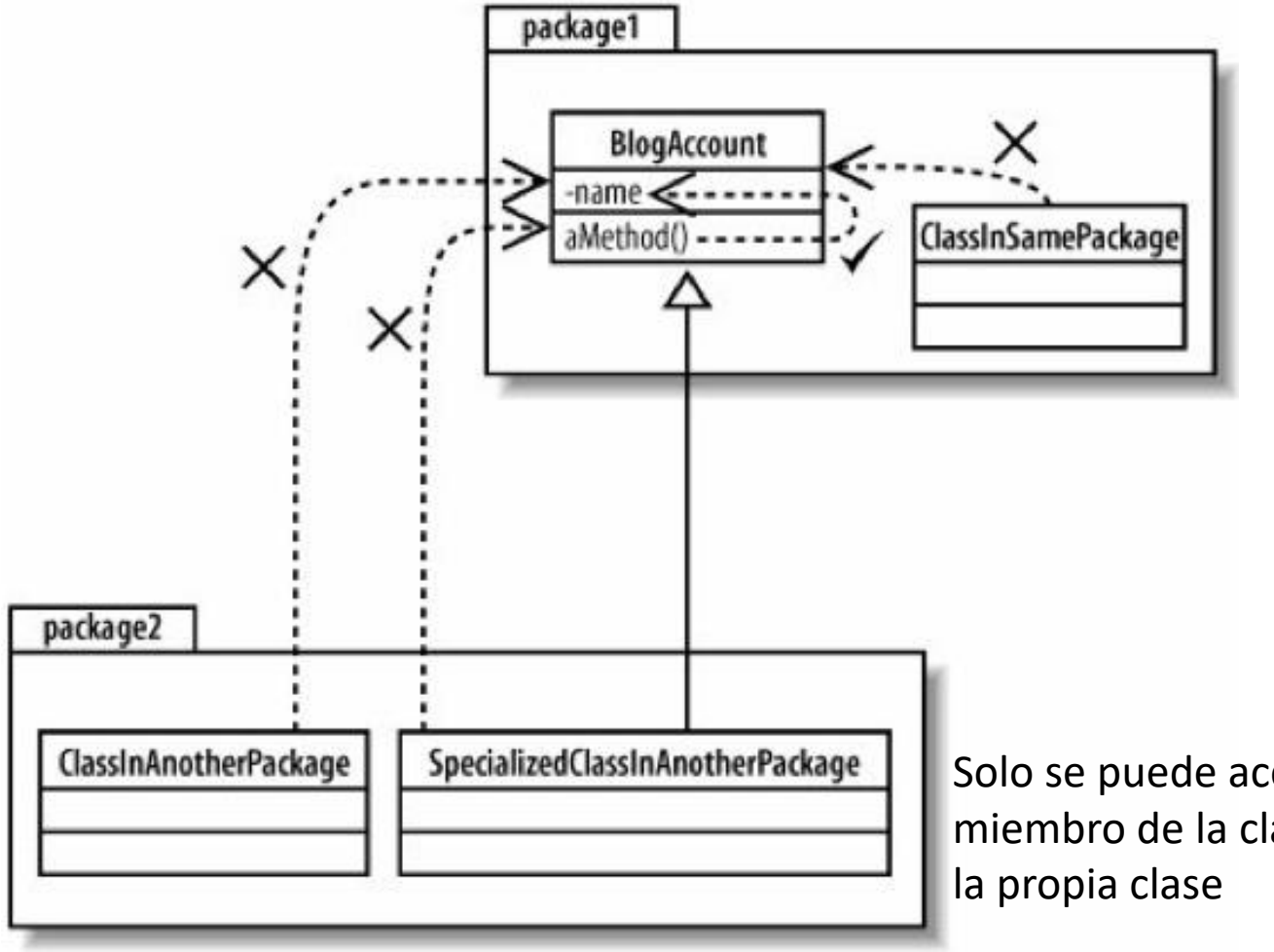
Solo se puede acceder al miembro de la clase desde la propia clase o desde la clase que hereda de ella

Visibilidad de Paquete



Solo se puede acceder desde el propio paquete

Visibilidad privada



Solo se puede acceder al miembro de la clase desde la propia clase

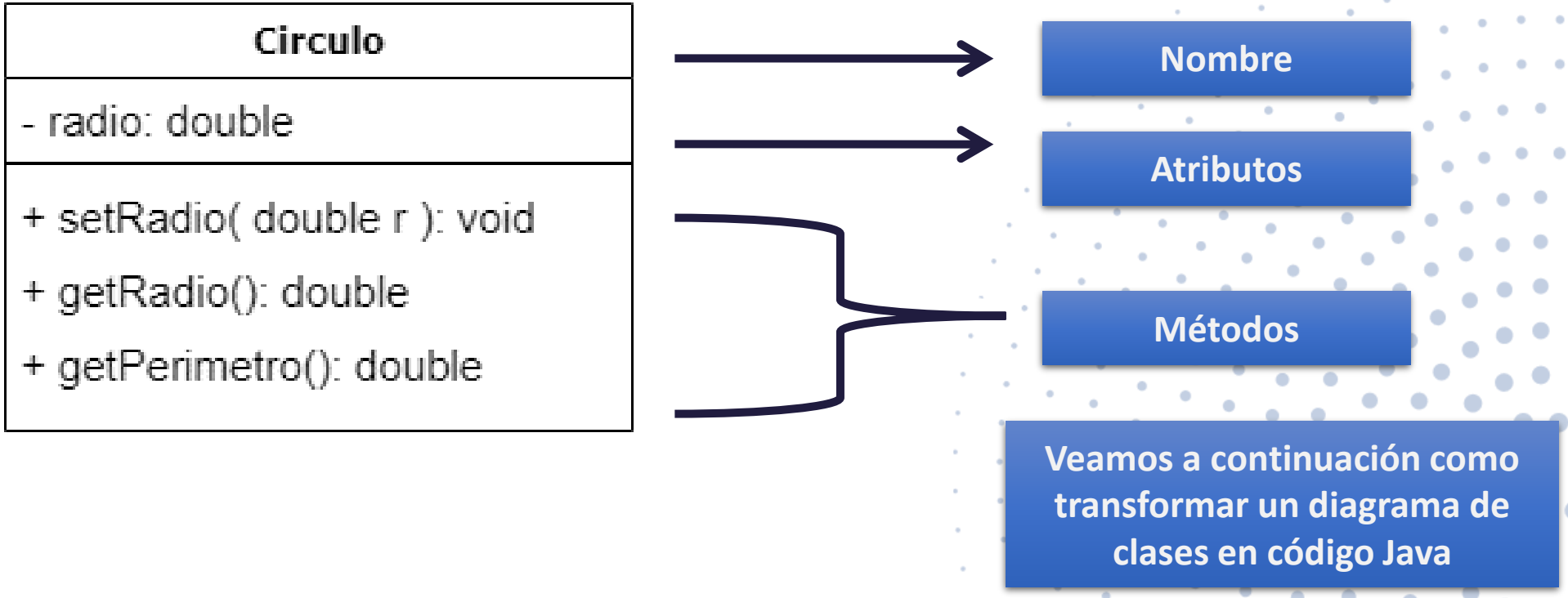
Clases y objetos en Java

FACULTAD DE INGENIERIA UdeA
Una facultad para la sociedad del aprendizaje

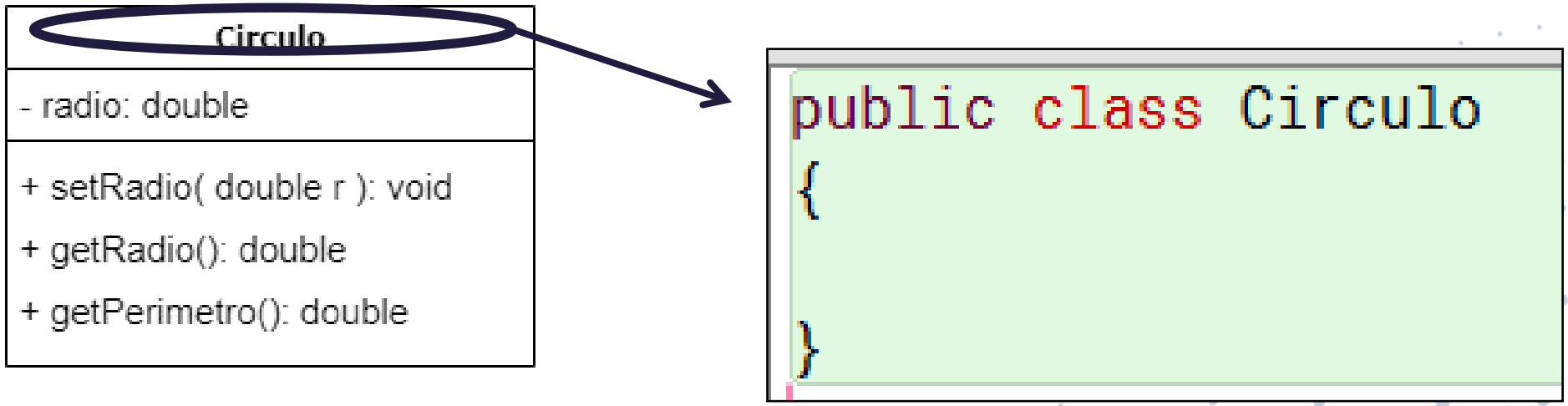
Luisa Fernanda Restrepo.

Clases en Java

- Una clase Java usa variables para definir campos de datos (atributos) y métodos para definir los comportamientos.

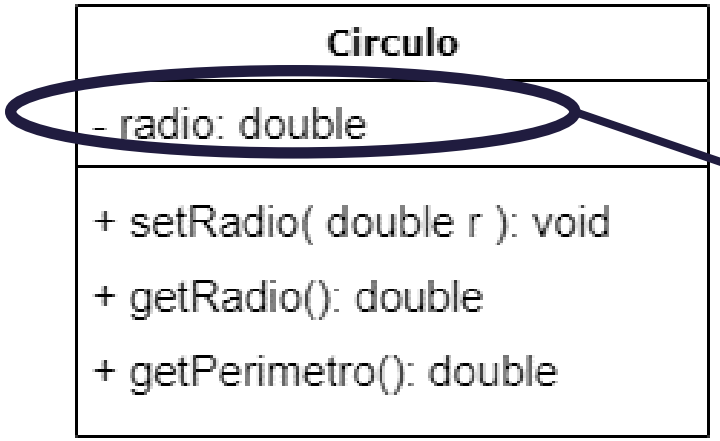


Clases en Java – II



Creemos una nueva clase Circulo

Clases en Java – II




```
public class Circulo
{
    private double radio;
}
```

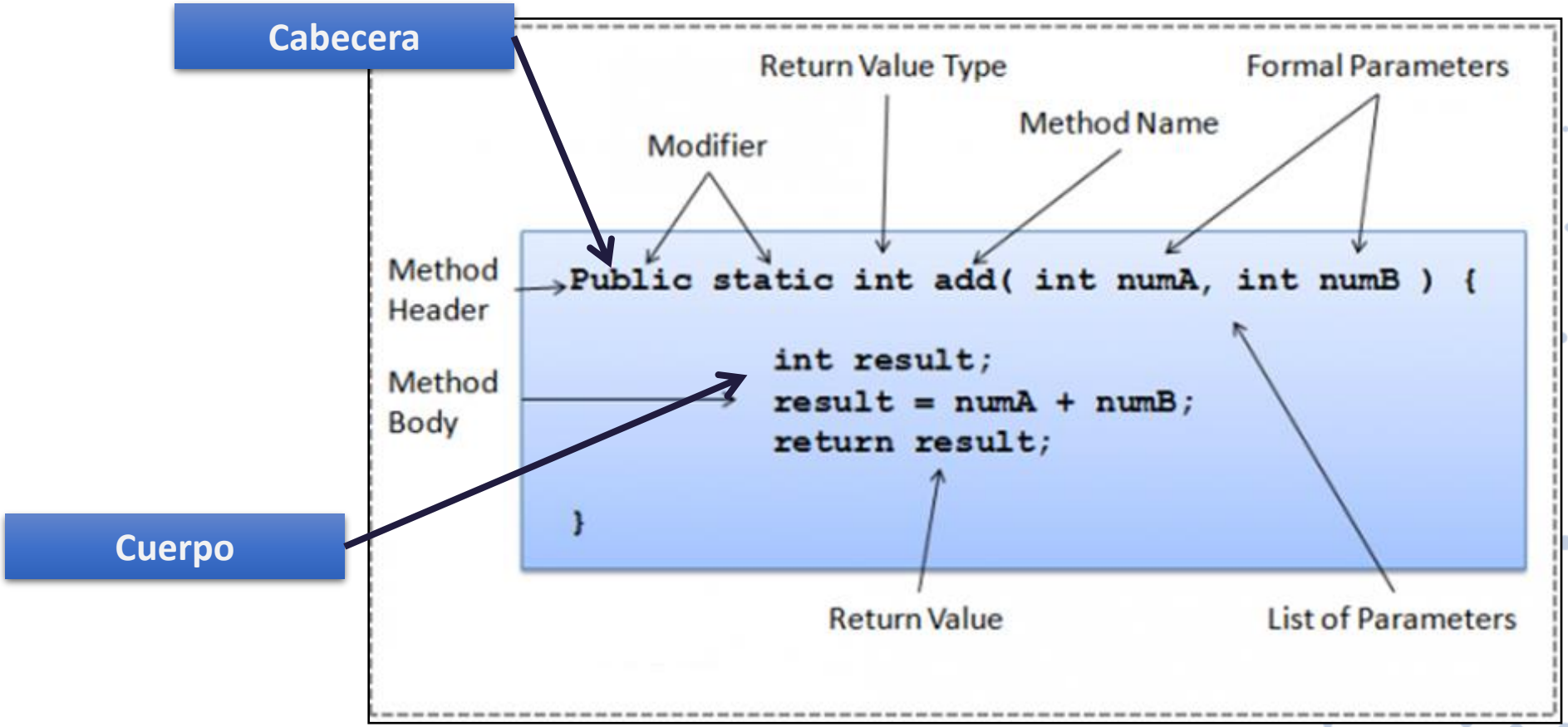
¿Por qué se le coloca visibilidad privada?

El "-" se transforma en "private"

Sobre los métodos

- Java define la siguiente estructura para crear métodos dentro de una clase

visibilidad tipoRetorno nombreMetodo(parámetros) {}



static keyword: https://www.javatpoint.com/static-keyword-in-java

Clases en Java – III

¿Por qué se le coloca visibilidad "public" a los métodos?

Circulo
- radio: double
+ setRadio(double r): void + getRadio(): double + getPerimetro(): double

```java
public class Circulo
{
    //Atributos
    private double radio;

    //Métodos
    public void setRadio(double r){
        this.radio = r;
    }

    public double getRadio(){
        return this.radio;
    }

    public double getPerimetro(){
        return 2 * this.radio * Math.PI;
    }
}
```

"this" es una palabra reserva de Java que permite entre otras cosas, acceder a atributos y métodos dentro de una clase.

Sugerencia: utilice siempre "this." para referirse a atributos o métodos de la clase (dentro de la clase).

Accediendo a los objetos via variables de referencia

- Se accede a los objetos a través de las variables de referencia del objeto. Esas variables se declaran utilizando la siguiente sintaxis:
  - ***NombreClase nombreVariableReferencia;***
- Una clase es un tipo de referencia, lo que significa que una variable del tipo de clase puede hacer referencia a una instancia de la clase. La siguiente declaración declara que la variable circulo1 es del tipo Circulo:

```
Circulo circulo1;
```

- La variable circulo1 puede hacer referencia a un objeto del Circulo. La siguiente declaración crea un objeto y asigna su referencia a circulo1 :
- Se puede escribir en una sola declaración:

```
Circulo circulo1;
circulo1 = new Circulo();
```

```
Circulo circulo1 = new Circulo();
circulo1.setRadio(5);
```

Creación de objetos

- Creamos una clase “Principal” que tenga un método “main”, y dentro de ese método “main”, creamos los objetos y llamamos los métodos necesarios:

```
public class Principal {
    public static void main(String[] args) {
        Circulo circulo1 = new Circulo();
        circulo1.setRadio(5);
        double peric1 = circulo1.getPerimetro();
        System.out.println("Perimetro de c1: " + peric1);

        Circulo circulo2 = new Circulo();
        circulo2.setRadio(10);
        System.out.println("Perimetro de c2: " + circulo2.getPerimetro());
    }
}
```

Opciones

```
Perimetro de c1: 31.41592653589793
Perimetro de c2: 62.83185307179586
```

Sobre los atributos

- Formas de acceder a un atributo (desde un método externo):

```
 Producto p1 = new Producto();
 p1.precio /* si el atributo es publico */
 p1.getPrecio() /* si el atributo es privado y existe el
método publico */
```

Visibilidad publica y privada

```java
public class Principal {
    public static void main(String[] args) {
        Vehiculo vehiculo1 = new Vehiculo();
        vehiculo1.nombre = "Tesla Model X";
        // vehiculo1.precio = 40000; // invalido por ser p
        vehiculo1.setPrecio(40000);
        System.out.println("Nombre: " + vehiculo1.nombre);
        System.out.println("Precio: " + vehiculo1.getPrecio());
    }
}
```

Solo se puede acceder a los atributos privados desde el interior de la clase, y se puede acceder a los atributos públicos desde cualquier otra clase.

```java
public class Vehiculo
{
    public String nombre;
    private int precio;

    public void setPrecio(int p){
        this.precio = p;
    }

    public int getPrecio(){
        return this.precio;
    }
}
```

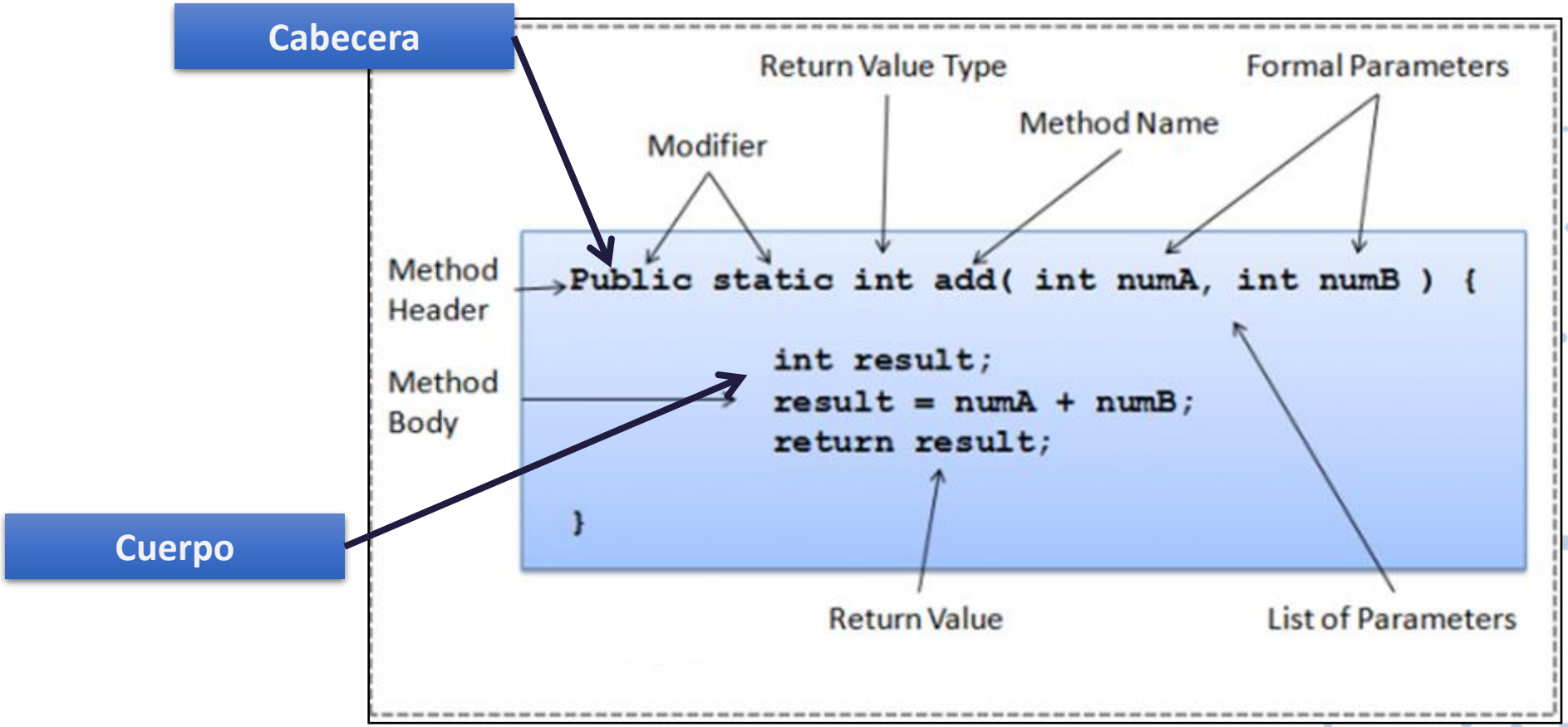
Opciones

```
Nombre: Tesla Model X
Precio: 40000
```

Sobre los métodos

- Java define la siguiente estructura para crear métodos dentro de una clase

visibilidad tipoRetorno nombreMetodo(parámetros) {}



static keyword: https://www.javatpoint.com/static-keyword-in-java

Métodos con y sin retorno

```java
public class Principal {
    public static void main(String[] args) {
        Vehiculo vehiculo1 = new Vehiculo();
        vehiculo1.nombre = "Tesla Model X";
        // vehiculo1.precio = 40000; // invalido por ser private
        vehiculo1.setPrecio(40000);
        System.out.println("Nombre: " + vehiculo1.nombre);
        System.out.println("Precio: " + vehiculo1.getPrecio());
    }
}
```

```java
public class Vehiculo
{
    public String nombre;
    private int precio;

    public void setPrecio(int p){
        this.precio = p;
    }

    public int getPrecio(){
        return this.precio;
    }
}
```

Los métodos sin retorno “void” se utilizan por lo general para establecer valores a atributos, imprimir mensajes, entre otros

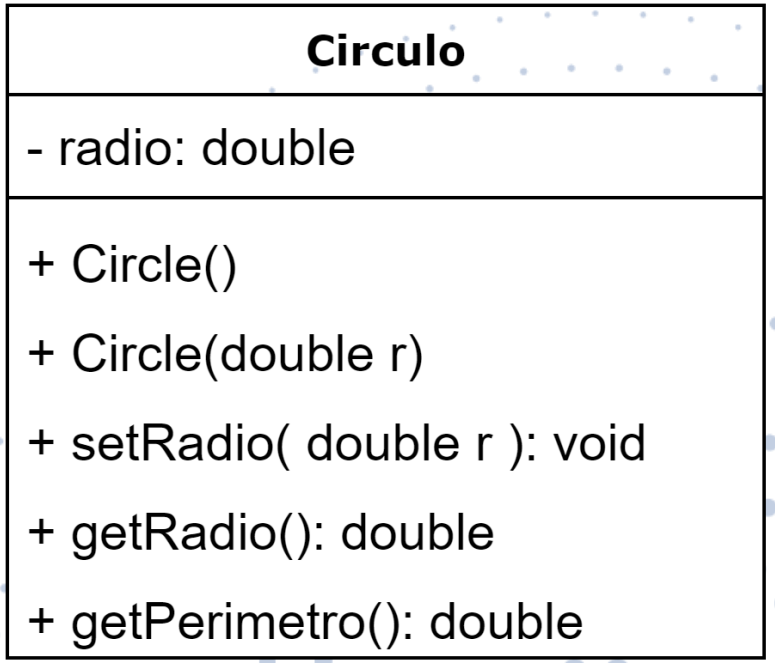
Los métodos con retorno, deben utilizar la palabra reserva “return” para devolver un valor del tipo declarado en la cabecera del método

Constructores

Luisa Fernanda Restrepo.

Constructores

- Los constructores son un tipo especial de método. Tienen tres características:
  - Un constructor debe tener el **mismo nombre** que la clase misma.
  - Los constructores **no tienen un tipo de retorno**.
  - Los constructores se invocan utilizando la palabra reservada "**new**" cuando se crea un objeto.
- Los constructores sirven para inicializar los objetos (generalmente para inicializar los valores de los atributos de los objetos).



Constructor de Circulo

- Modifique la clase Circulo y créele el siguiente constructor:

¿Qué pasa cuando se crea un nuevo objeto de la clase Circulo?

¿Qué imprime?

```
public class Circulo
{
    private double radio;

    public Circulo(){
        this.radio = 1;
    }
```

Opciones

```
Radio de c1: 1.0
```

```
public class Principal {
    public static void main(String[] args) {
        Circulo circulo1 = new Circulo();
        System.out.println("Radio de c1: " + circulo1.getRadio());
    }
}
```

Constructor de Circulo - 2

- Cree el siguiente constructor "adicional" a la clase Circulo

```
public class Circulo
{
    private double radio;

    public Circulo(){
        this.radio = 1;
    }

    public Circulo(double r){
        this.radio = r;
    }
```

¿Cómo se debería invocar ese nuevo constructor?

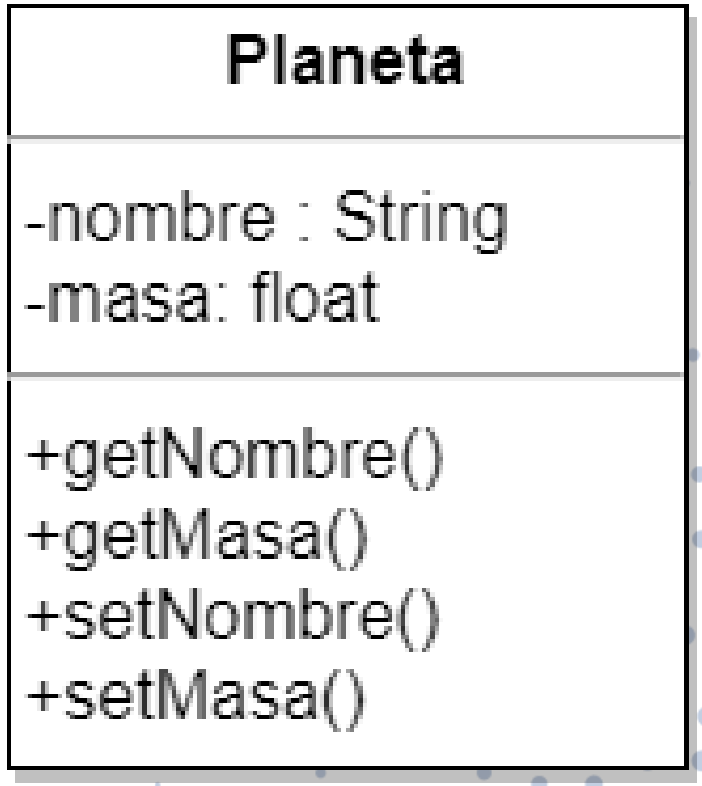
```
import java.util.Scanner;
public class Principal {
    public static void main(String[] args) {
        Circulo circulo1 = new Circulo(56.7);
        System.out.println("Radio de c1: " + circulo1.getRadio());
    }
}
```

Opciones

```
Radio de c1: 56.7
```

Ejercicio

- Cree la siguiente clase en Java.
- Cree el constructor vacío de planeta, y asigne una masa por defecto igual a 4000.
- Cree un constructor que reciba el nombre y la masa
- Cree la clase Principal, cree objetos de la clase Planeta y pruebe los métodos que crearon.



Enviar un objeto a un método

- Hasta ahora, todo lo que se ha discutido es cómo se pueden enviar tipos de **datos primitivos** a un método (byte, short, int, long, float, double, boolean, char).

- Sin embargo, los datos a menudo son más complejos que un dato primitivo.

- Por lo que sería útil tener una forma de enviar no solo un elemento o dos, sino un objeto completo a un método.

Clase Point

- Codifique la siguiente clase:
- Clase que define puntos en coordenadas x y y.

```
public class Point
{
    private double x;
    private double y;

    public Point(){
        this.x = 0;
        this.y = 0;
    }
```

Recepción de datos primitivos

```
public void setX(double x){
    this.x = x;
}

public double getX(){
    return this.x;
}

public void setY(double y){
    this.y = y;
}

public double getY(){
    return this.y;
}
```

Distancia entre dos puntos

- Suponga que se le pide calcular la distancia entre dos puntos.
- La formula es la siguiente:

$$\text{dist} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

¿Donde deberíamos implementar ese calculo?

¿En que clase?

Opción 1: en la clase Principal

Codifique lo siguiente

```
public class PrincipalPoint
{
    public static void main(String[] args) {
        Point p1 = new Point();
        p1.setX(4);
        p1.setY(4);
        Point p2 = new Point();
        p2.setX(4);
        p2.setY(5);
        double dist = Math.sqrt(Math.pow(p1.getX()-p2.getX(),2)
            + Math.pow(p1.getY()-p2.getY(),2));
        System.out.println(dist);
    }
}
```

¿Problemas con poner este código aquí?

Opción 2: en la clase Point

```
public class Point
{
    private double x;
    private double y;

    public Point(){
        this.x = 0;
        this.y = 0;
    }

    public double distance(Point p2){
        double dist = Math.sqrt(Math.pow(this.x-p2.getX(),2)
            + Math.pow(this.y-p2.getY(),2));
        return dist;
    }
```

Datos del propio objeto (desde el que se invoca este método)

Codifique lo siguiente

Datos del objeto recibido

Opción 2 – modificación clase Principal

```
public class PrincipalPoint
{
    public static void main(String[] args) {
        Point p1 = new Point();
        p1.setX(4);
        p1.setY(4);
        Point p2 = new Point();
        p2.setX(4);
        p2.setY(5);
        double dist = p1.distance(p2);
        System.out.println("Distancia: "+dist);
    }
}
```

Codifique lo siguiente

Ahora se puede reutilizar fácilmente ese método

¿Ventajas de esta propuesta?

Opciones

```
Distancia: 1.0
```

Retornando un objeto

- Suponga que queremos encontrar el punto medio entre dos puntos.

$$midx = \frac{x_1 + x_2}{2} \qquad midy = \frac{y_1 + y_2}{2}$$

- Haga un método llamado midPoint() en la clase Point, que encuentre el punto medio entre dos puntos (debe recibir el objeto p2 como parámetro). Y la función deberá retornar un nuevo objeto (con las coordenadas x y y del punto medio) de la clase Point.

Retornando un objeto – II

```
public class Point
{
    private double x;
    private double y;

    public Point(){
        this.x = 0;
        this.y = 0;
    }

    public Point midPoint(Point p2){
        Point midP = new Point();
        midP.setX((this.x+p2.getX())/2);
        midP.setY((this.y+p2.getY())/2);
        return midP;
    }
```

Retorno de un dato “no primitivo”

Retornando un objeto – III

```
public class PrincipalPoint
{
    public static void main(String[] args)
        Point p1 = new Point();
        p1.setX(4);
        p1.setY(4);
        Point p2 = new Point();
        p2.setX(4);
        p2.setY(5);
        double dist = p1.distance(p2);
        midp = p1.midPoint(p2);
        System.out.println("Distancia: "+dist);
        System.out.println("Punto medio: x "+
           midp.getX()+" y "+midp.getY());
}
```

Retorno de un dato "no primitivo"

Opciones

```
Distancia: 1.0
Punto medio: x 4.0 y 4.5
```

¿Qué tipo de variable debería colocar?

Sobrecarga

Luisa Fernanda Restrepo.

Sobrecarga

- El constructor en el ejemplo anterior inicializa las variables x y y en 0 como valor predeterminado.

- Pero un programador podría desear inicializar esas variables con otro valor (evitando tener que usar los get y los set).

- Para eso, lo que se puede hacer es sobrecargar el constructor.

Constructor sobrecargado

```
public class Point
{
    private double x;
    private double y;

    public Point(){
        this.x = 0;
        this.y = 0;
    }

    public Point(double x, double y){
        this.x = x;
        this.y = y;
    }
```

Sobrecarga

Sobrecarga adicional

```java
public Point(){
    this.x = 0;
    this.y = 0;
}

public Point(double x, double y){
    this.x = x;
    this.y = y;
}

public Point(Point p2){
    this.x = p2.getX();
    this.y = p2.getY();
}
```

¿Para que serviría el tercer constructor?

Si ya cree un objeto (suponga que con el primer constructor), ¿puedo luego invocar el tercer constructor sobre el mismo objeto?

Sobrecarga de métodos

- También se pueden sobrecargar métodos:

```
public double distance(double x, double y){
    double dist = Math.sqrt(Math.pow(this.x-x,2)
        + Math.pow(this.y-y,2));
    return dist;
}

public double distance(Point p2){
    double dist = Math.sqrt(Math.pow(this.x-p2.getX(),2)
        + Math.pow(this.y-p2.getY(),2));
    return dist;
}
```

Ejemplo 1

- ¿Algún comentario sobre los siguientes códigos?

Sobrecarga invalida

```
public double distance(double z, double h ){
    double dist = Math.sqrt(Math.pow(this.x-x,2)
        + Math.pow(this.y-y,2));
    return dist;
}

public double distance(double x, double y){
    double dist = Math.sqrt(Math.pow(this.x-x,2)
        + Math.pow(this.y-y,2));
    return dist;
}
```

No se puede hacer sobrecarga si se recibe la misma cantidad de parámetros con el mismo tipo

Ejemplo 2

Sobrecarga invalida

- ¿Algún comentario sobre los siguientes códigos?

```
public int distance(double z, double h ){
    double dist = Math.sqrt(Math.pow(this.x-x,2)
        + Math.pow(this.y-y,2));
    return dist;
}

public double distance(double x, double y){
    double dist = Math.sqrt(Math.pow(this.x-x,2)
        + Math.pow(this.y-y,2));
    return dist;
}
```

No se puede hacer sobrecarga únicamente cambiando el tipo de retorno

Importante

- La sobrecarga de métodos es valida siempre y cuando: no exista otro método con el mismo nombre que tenga la misma cantidad de parámetros y los mismos tipos de cada uno de los parámetros recibidos en el mismo orden.

- Finalmente, no se puede hacer sobrecarga de un método que ya existe, si solo se cambia el tipo de visibilidad, o el tipo de retorno.

This

Luisa Fernanda Restrepo.

This

- ¿Qué pasa con el siguiente código?

```
public Point(double x, double y){
    x = x;
    y = y;
}
```

Solución

```
public Point(double x, double y){
    this.x = x;
    this.y = y;
}
```

- La respuesta a la pregunta anterior, es que los parámetros y las variables locales declaradas en un método tienen prioridad sobre cualquier variable declarada globalmente en el objeto (atributos).
- Por lo tanto, toca utilizar **this** para poder diferenciar, al atributo x del objeto, del parámetro x recibido en el método (constructor).
- Recuerde la recomendación del curso, siempre que pueda, utilice **this.**

This – invocando otro constructor

- La palabra reservada this, también se puede utilizar para invocar otro constructor desde un constructor.
- Veamos el siguiente ejemplo:

```
public class Point
{
    private double x;
    private double y;

    public Point(){
        this(0,0);
    }

    public Point(double x, double y){
        this.x = x;
        this.y = y;
    }
```

Método main

```
public class PrincipalPoint
{
    public static void main(String[]
        Point p1 = new Point();
    }
}
```

```
publi lass Point
{
    priva  double x;
    privat  uble y;

    public Point(){
        this(0,0);
    }

    public Point(double x, double y){
        this.x = x;
        this.y = y;
    }
```

¿Cuáles serian los valores x y y del objeto p1?

This – invocando otro constructor – II

- La ventaja de invocar un constructor desde otro constructor, es que generalmente la lógica de asignación quedará en unos pocos constructores. Lo cual, facilita el mantenimiento y evolución de los constructores.

```
public Point(){
    this(0,0);
}
```

Si usted utiliza this() para invocar otro constructor, esa invocación se deberá realizar siempre en la primera línea del cuerpo del constructor que planea realizar la invocación.

```
public Point(){
    this.x = 0.0;
    this.y = 0.0;
}

public Point(double x, double y){
    this.x = x;
    this.y = y;
}

public Point(Point p2){
    this.x = p2.getX();
    this.y = p2.getY();
}
```

Modifique estos 2 constructores para que ahora utilicen this()

Constantes y static

Luisa Fernanda Restrepo.

Constantes

- Codifique lo siguiente:

```
public class Gafa
{
    private String nombre;
    private double precio;

    public Gafa(String n, double p){
        this.nombre = n;
        this.precio = p;
    }
```

```
public double precioConIva(){
    int iva = 19;
    double total = this.precio+(this.precio*iva/100);
    return total;
}

public double precioConIvaYEnvio(){
    int iva = 19;
    int envio = 2000;
    double total = this.precio+(this.precio*iva/100)+envio;
    return total;
}
```

Constantes – opción 2

¿Qué le parece esta opción?

```
public class Gafa
{
    private String nombre;
    private double precio;
    private int iva = 19;
    private int envio = 2000;
```

```
public double precioConIva(){
    double total = this.precio+(this.precio*this.iva/100);
    return total;
}

public double precioConIvaYEnvio(){
    double total = this.precio+(this.precio*this.iva/100)+this.envio;
    return total;
}
```

Suponga que en el proyecto se definió que el iva y el envío son constantes

¿Algún problema?

Constante Nombrada (final variable)

- El valor de una variable puede cambiar durante la ejecución de un programa, pero una constante con nombre o simplemente constante, representa datos permanentes que nunca cambian.

```
final tipo_variable NOMBRECONSTANTE = valor;
```

- Ejemplo:

```
final double PI = 3.14159;
```

Constantes – opción 3

¿Qué le parece esta opción?

```
public class Gafa
{
    private String nombre;
    private double precio;
    private final int IVA = 19;
    private final int ENVIO = 2000;
```

```
public double precioConIva(){
    double total = this.precio+(this.precio*this.IVA/100);
    return total;
}

public double precioConIvaYEnvio(){
    double total = this.precio+(this.precio*this.IVA/100)+this.ENVIO;
    return total;
}
```

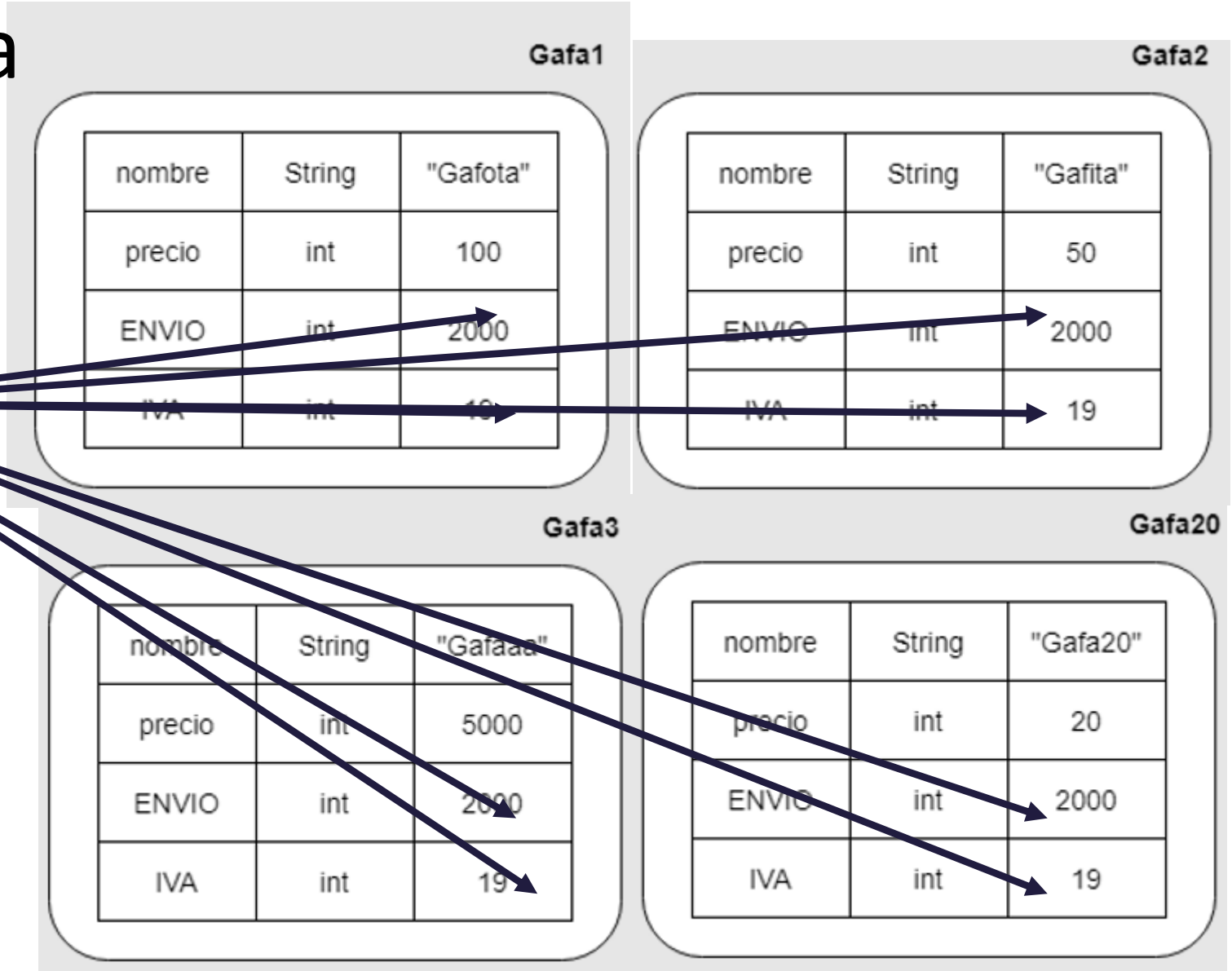
¿Qué pasaría si se crean 20 objetos tipo Gafa?

Palabra reservada para declarar variable constante. Se recomienda definir las constantes en mayúscula.

Memoria en Java caso anterior

Se gasta espacio en memoria repitiendo los mismos valores 20 o más veces

¿Algún comentario?



Constantes – opción 4

```java
public class Gafa
{
    private String nombre;
    private double precio;
    private final static int IVA = 19;
    private final static int ENVIO = 2000;
```

```java
public double precioConIva(){
    double total = this.precio+(this.precio*this.IVA/100);
    return total;
}

public double precioConIvaYEnvio(){
    double total = this.precio+(this.precio*this.IVA/100)+this.ENVIO;
    return total;
}
```

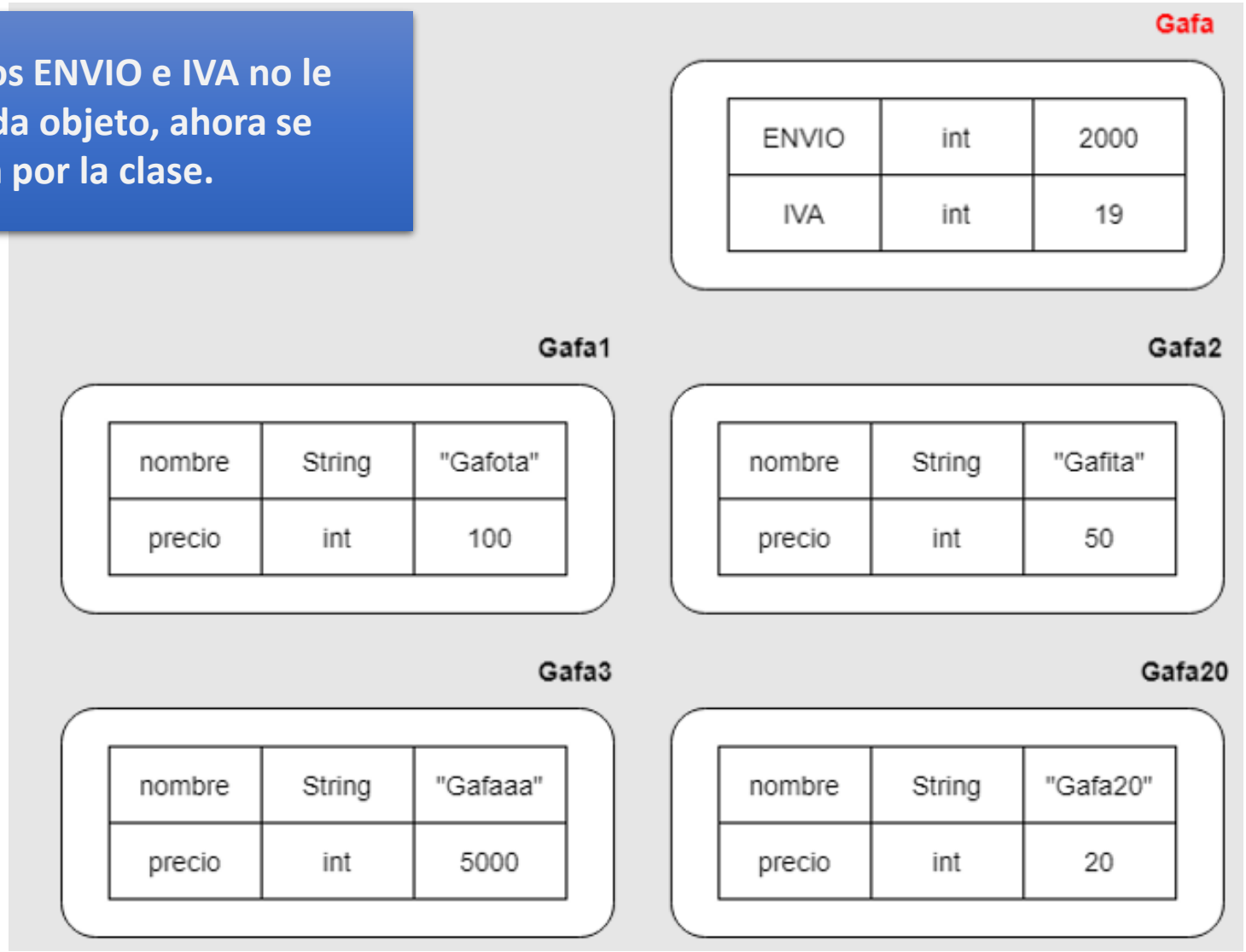
Palabra reservada para declarar un atributo de clase (estático). Este atributo se compartirá entre todos los objetos de esa clase.

Veamos la memoria en el caso anterior

Ahora los atributos ENVIO e IVA no le pertenecen a cada objeto, ahora se comparten por la clase.

¿Debería definir nombre y precio como "static"?

R/= NO. Ya que el nombre y precio es propio de cada objeto. No se deberían compartir.



Métodos de clase (static)

- Los **métodos de clase** (o métodos estáticos), son método que:
  - Se definen dentro de una clase
  - Se definen mediante el uso de la palabra reservada static
  - Se pueden invocar sin necesidad de usar objetos (sin necesidad de crear instancias).

Ejemplo método de clase

Codifique lo siguiente

Nota: no se puede utilizar la palabra reservada "this" dentro de un método estático

```
public class Gafa
{
    private String nombre;
    private double precio;
    private final static int IVA = 19;
    private final static int ENVIO = 2000;

    public Gafa(String n, double p){
        this.nombre = n;
        this.precio = p;
    }

    public static void imprimirDatosGenerales(){
        System.out.println("Todos los productos vendidos tiene un iva de : ");
        System.out.println(Gafa.IVA);
        System.out.println("Todos los productos vendidos tiene un costo de envio de : ");
        System.out.println(Gafa.ENVIO);
    }
```

Ejemplo método de clase – parte 2

```
public class PrincipalGafa
{
    public static void main(String[] args) {
        Gafa.imprimirDatosGenerales();
    }
}
```

Forma de invocar un método estático

```
Todos los productos vendidos tiene un iva de :
19
Todos los productos vendidos tiene un costo de envio de :
2000
```

Math.pow()

¿De que tipo es el método Math.pow()?

```
public class PrincipalGafa
{
    public static void main(String[] args) {
        System.out.println(Math.pow(2,2));

    }
}
```

Math.pow() es un método "static", ya que como vemos, no necesitamos instanciar un objeto de la clase Math, para poder utilizar el método "pow"

```
     * @param a the number
     * @param b the power t
     * @return a<sup>b</sup>
     */
    public static double pow(double a, double b)
    {
        return VMMath.pow(a,b);
    }
```

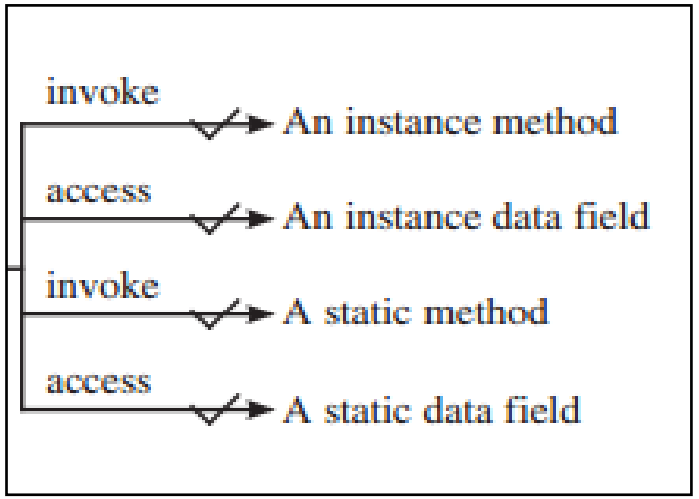
- ***Sugerencia:*** *siempre que vaya a invocar un método de clase, o una variable de clase, invóquela siguiente el patrón:*
  - *NombreClase.variableDeClase*
  - *NombreClase.metodoDeClase()*

- ***Sugerencia:*** *si dentro de un método de una clase, usted observa que no necesita utilizar la palabra* ***this*** *probablemente ese método sea mejor definirlo como método de clase.*

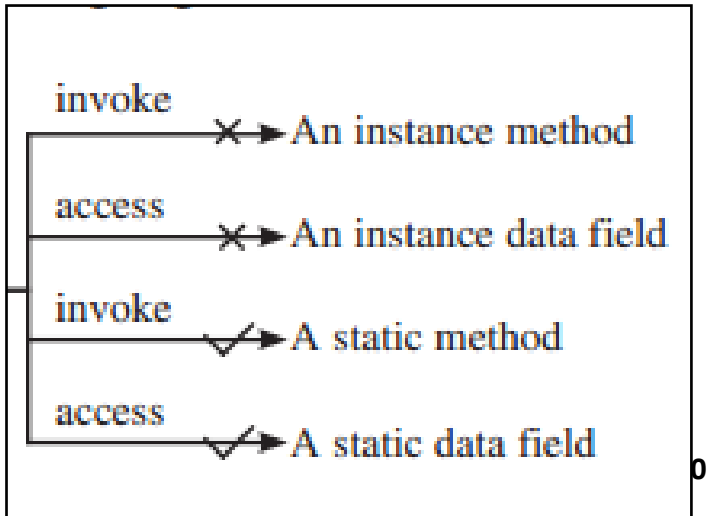
Métodos estáticos restricciones

```java
public class A {
int i = 5;
static int k = 2;

public void m1() {
i = i + k + m2(i, k); //OK, por que un método de instancia
//puede usar variables estaticas y metodos estaticos
}

public static int m2(int i, int j) {
return (int)(Math.pow(i, j));
}

public static void main(String[] args) {
//int j = i; // Incorrecto por que es la instancia de una variable
//m1(); // Incorrecto por que m1() es la instancia de un método
m2(3,4); //OK, invoca un método estático

A a = new A();
int j = a.i; // OK, por que a.i accede a la variable de instancia
             //desde el objeto
a.m1(); // OK, a.m1() invoca la instancia del método desde el objeto
a.m2(3,4); //OK
}
}
```

MÉTODO DE INSTANCIA



MÉTODO ESTÁTICO



Variables de clase publicas

- Por lo general, se suelen colocar las **variables de clase** como publicas, ya que esto permite un fácil acceso a estos datos desde otras clases (sin necesidad de instanciar objetos, o acceder a través de métodos de clase).

Acceso a variable static mediante objeto

```
public class Gafa
{
    private String nombre;
    private double precio;
    public final static int IVA = 19;
    public final static int ENVIO = 2000;
```

```
public static void main(String[] args) {
    Gafa g1 = new Gafa("Super gafa", 200);
    System.out.println("Accediendo a variable static desde objeto: " +g1.IVA);
    System.out.println("Accediendo a variable static sin objeto: " +Gafa.IVA);
}
```

Acceso a variable static mediante clase (opción recomendada)

```
Opciones
Accediendo a variable static desde objeto: 19
Accediendo a variable static sin objeto: 19
```

Math.PI

¿De que tipo es el atributo Math.PI?

```
public class PrincipalGafa
{
    public static void main(String[] args) {
        System.out.println(Math.PI);

    }
}
```

Math.PI es un atributo de clase (de la clase Math), el cual es publico, y que además representa una constante.

```
/**
 * The most accurate approximatio
 * <code>3.1415926535897932</code>
 * to its circumference.
 */
public static final double PI = 3.141592653589793;
```

Ejercicio contador

- Modifique la clase Gafa.
- Cree un atributo de clase(static) llamado contador (int).
- Cada vez que se cree una Gafa (se invoque un constructor) aumente en 1 el contador.
- Cree una clase principal.
- Cree 5 objetos gafas.
- Luego imprima el valor de Gafa.contador (deberá aparecer 5).

Solución Gafa

```java
public class Gafa
{
    private String nombre;
    private double precio;
    private final static int IVA = 19;
    private final static int ENVIO = 2000;
    public static int contador = 0;

    public Gafa(){
        contador++;
    }

    public double precioConIva()
    {
        double total = this.precio + (this.precio*this.IVA/100);
        return total;
    }

    public double precioConIvaYEnvio()
    {
        double total = this.precio + (this.precio*this.IVA/100)+ this.ENVIO;
        return total;
    }

    public static void imprimirDatosGenerales()
    {
        System.out.println(Gafa.IVA);
    }
}
```

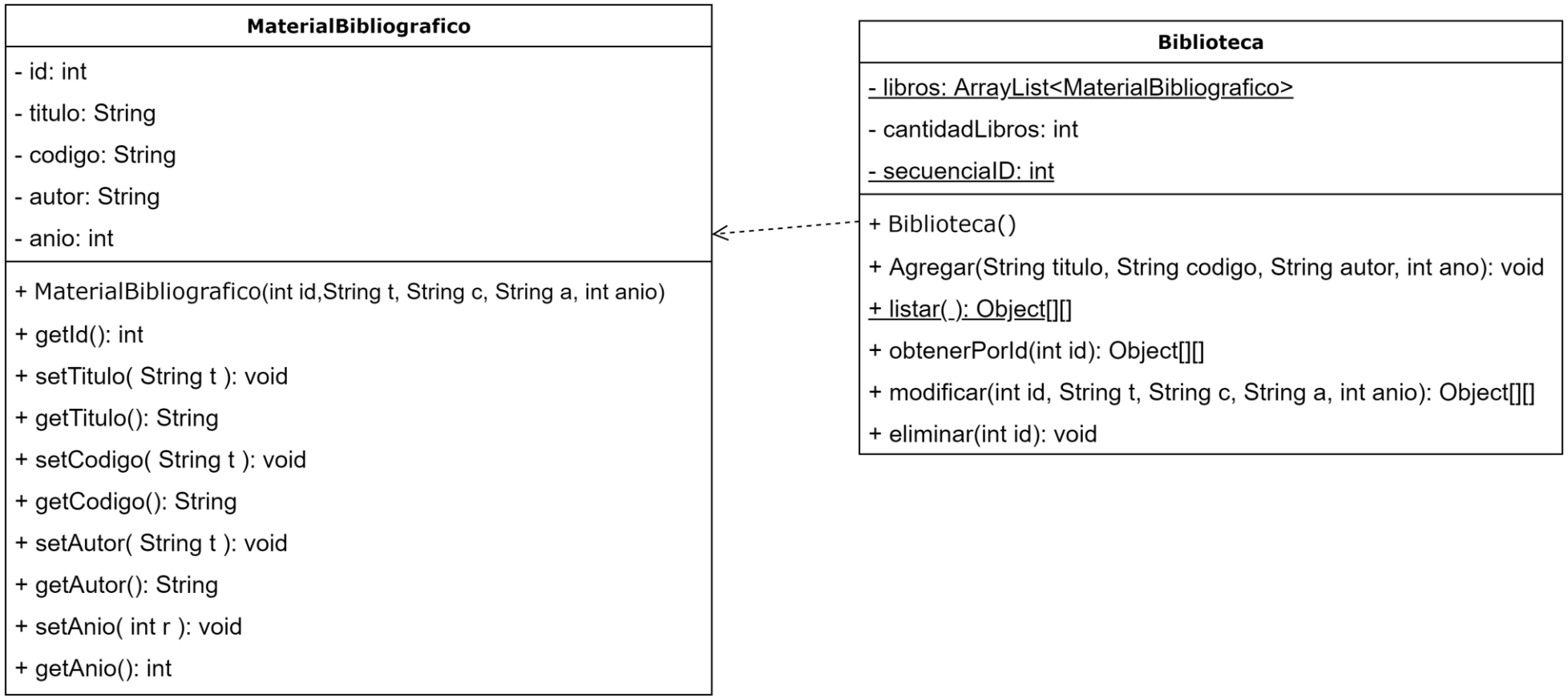
```java
public class PrincipalGafa
{
    public static void main(String[] args)
    Gafa g1 = new Gafa();
    Gafa g2 = new Gafa();
    Gafa g3 = new Gafa();
    Gafa g4 = new Gafa();
    Gafa g5 = new Gafa();

    System.out.println(Gafa.contador);
    }
}
```

Ejercicio midPoint

- Modifique el método midPoint() en la clase Point, ahora deberá ser un método de clase.

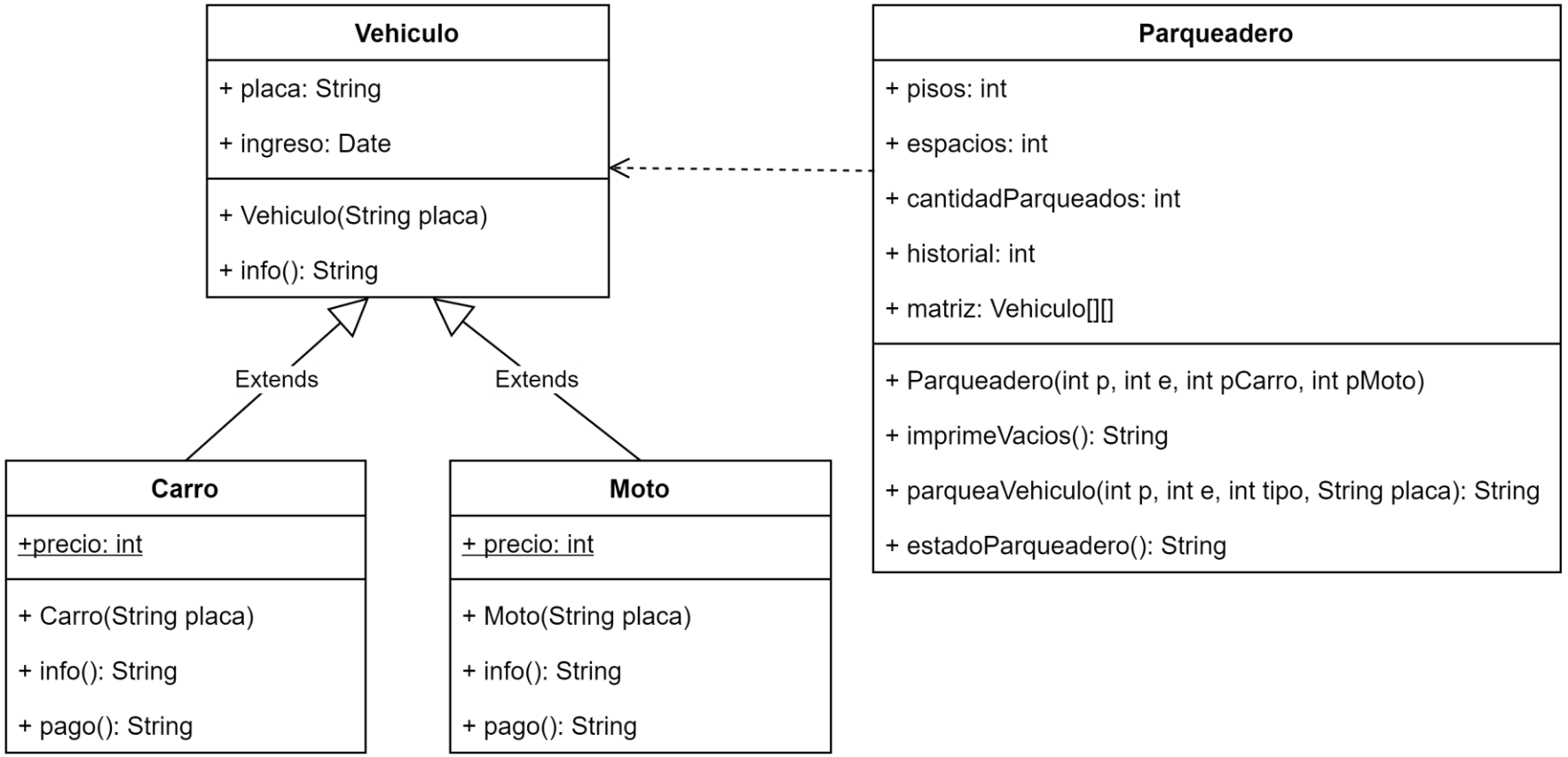
Ejercicio MaterialBibliográfico-PROYECTO 1

- Codificar la siguiente estructura en Java:



Ejercicio Parqueadero - PROYECTO 2

- Codificar la siguiente estructura en Java:



Referencias

Basado en el material elaborado por: Daniel Correa (docente EAFIT).

Liang, Y. D. (2017). Introduction to Java programming: comprehensive version. Eleventh edition. Pearson Education.

Streib, J. T., & Soma, T. (2014). *Guide to Java*. Springer Verlag.