

Clases / Métodos

(Java)

Clases con Métodos

(Java)

Una clase es una *plantilla* que define un objeto. La clase agrupa atributos y métodos que operarán sobre los datos.

Los atributos son *variables* que definen el *tipo de acceso* y el alcance:

Modificador de Acceso
+public
#protected
<u>+static</u>
-private

Una variable declarada dentro de corchetes “{” y “}” en un método, tiene alcance solamente dentro de los corchetes.

Clases con Métodos

(Java)

El esquema general de creación de elementos en programación orientada a objetos

```
class [nombre de la clase] {  
    [atributos o variables de la clase]  
    [métodos o funciones de la clase]  
    [main]  
}
```

```
class NombreClase {  
    //Declarar variables de instancia  
    tipo variable1;  
    tipo variable2;  
    //Declarar métodos  
    tipo metodo1(parámetros) {  
        //Instrucciones de método  
    }  
    tipo metodo2(parámetros) {  
        //Instrucciones del método  
    }  
}
```

Clases con Métodos

(Java)

//Le damos un nombre "MiClase" a la clase

```
public class MiClase {
```

```
    //Atributos de la clase
```

```
    private String atributo1;
```

```
    private int atributo2;
```

```
    private float atributo3;
```

```
    //Constructor con el mismo nombre de la clase
```

```
    public MiClase() {
```

```
        //el constructor es público no lleva void y no devuelve nada
```

```
        //Métodos de la clase
```

```
        public void metodo1() { //Método vacío que no retorna nada }
```

```
        public String metodo2() {
```

```
            return "Esto en una cadena"
```

```
        }
```

```
    }
```

Clases con Métodos

(Java)

```
public class Persona {  
    public void inicializar() { }  
    public void imprimir() { }  
    public void esMayorEdad() { }  
  
    public static void main(String[] args) {  
        Persona persona1 = new Persona();  
        persona1.inicializar();  
        persona1.imprimir();  
        persona1.esMayorEdad();  
    }  
}
```

Métodos

(Java)

Un método con o sin parámetros tiene la siguiente sintaxis:

```
public void [nombre del método]() {  
    [instrucciones]  
}
```

```
public void [nombre del método]([parámetros]) {  
    [instrucciones]  
}
```

Métodos

(Getter y Setter)

Son métodos que permiten tener **alcance** a los atributos encapsulados

Permiten **inicializar** los atributos o **conseguir** el contenido de un atributo

Setter

/ Definidor / Inicializador

Getter

/ Captador / *Obtenedor

*Palabra inexistente

Ing. Fernando Ospina Marín
fospinam@gmail.com

Clases

(Getter y Setter)

Sintaxis el método **Getter**:

```
public tipo-dato_a_devolver nombre_del_método () {  
    return dato_a_devolver;  
}
```

Sintaxis el método **Setter**:

```
public void nombre_del_método () {  
}
```


Clases abstractas

(Java)

Las **clases abstractas** son las clases que se declaran pero no contienen implementación, **no se pueden instanciar** y requieren subclases para proporcionar implementaciones para métodos abstractos. **Solo pueden ser heredadas.**

Proporciona una interfaz para las subclases derivadas .

Las clases abstractas contienen uno o más **métodos abstractos**. Los métodos abstractos son aquellos que solamente tienen una declaración, pero no una implementación detallada de las funcionalidades

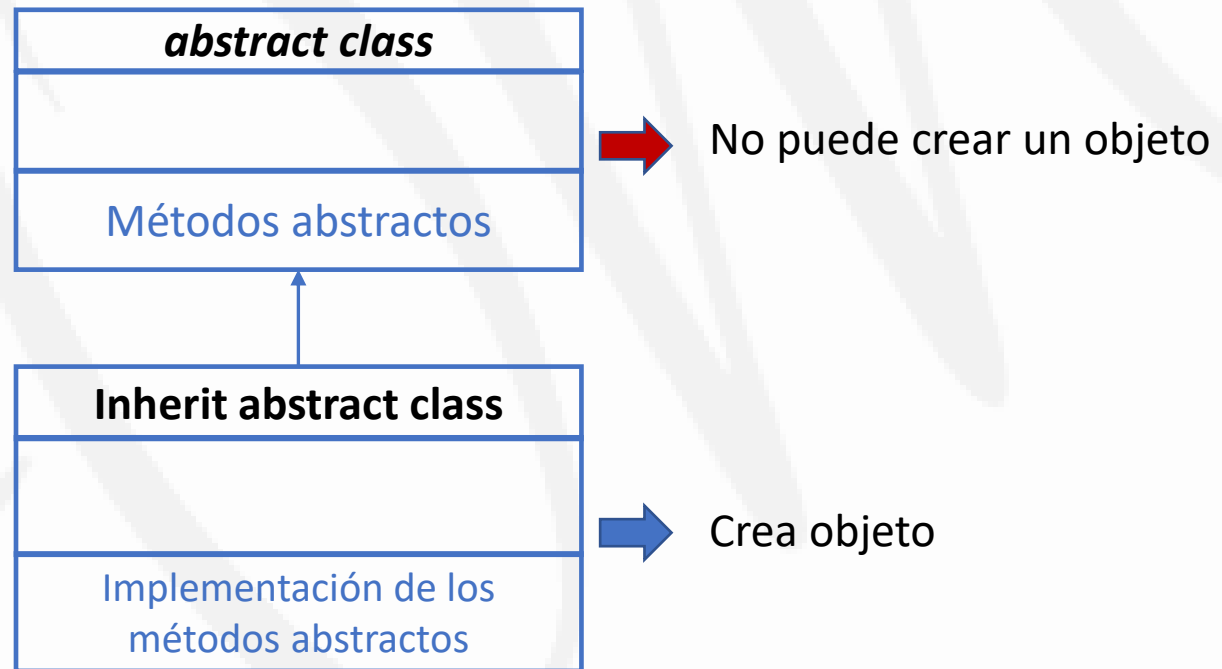
Clases abstractas

(Java)

Las **clases derivadas** de las **clases abstractas** deben implementar necesariamente todos los métodos abstractos para poder crear una clase que se ajuste a la interfaz definida. Si no se define alguno de los métodos no se podrá crear la clase.

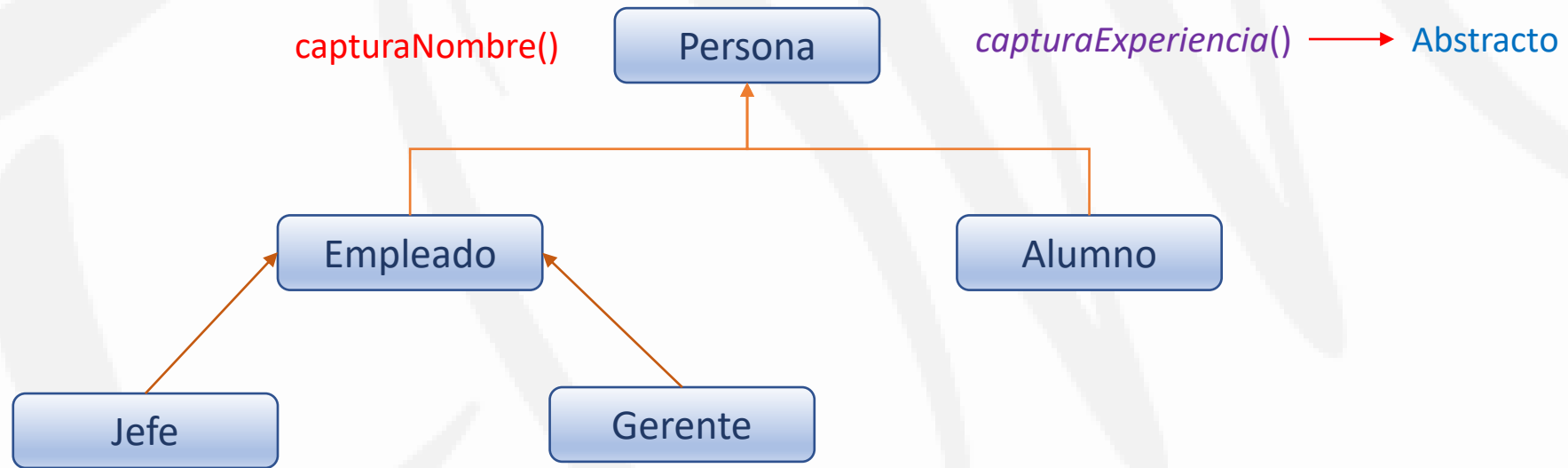
Clases abstractas

(Java)



Clases Abstractas

(Jerarquía de Herencia)



Todas las clases que heredan uno o más métodos abstracto(s) están **obligadas** a sobre escribir el o los método(s) abstracto(s) heredado(s)

Clases con Métodos

(Java)

```
abstract class Persona {  
    public String capturaNombre(){  
        acciones  
    }  
    public abstract String capturaExperiencia();  
}
```

Una clase **abstracta** tiene por lo menos 1 método abstracto

Constructores

(Java)

Son métodos **públicos** que permiten **inicializar o dar valores** a los atributos de la clase.

Una clase puede tener **varios constructores** y se usan para inicializar las variables **dependiendo del contexto** que requiera el programa, es decir, **se inicializan de diferente forma los atributos de la clase**.

Los métodos **constructores siempre** llevan el nombre de la clase y no llevan **void** y **NO devuelven** ningún tipo de dato.

Constructores

(Java)

Se pueden crear clases **sin constructores***. (Solo con **GETTERS** y **SETTERS**)

Java interpreta que debe crear el constructor por **default**, que consiste en un constructor **sin parámetros**.

*No es común esta situación

Sobre carga de Constructores y Métodos

(Java)

Para tener más de un constructor se debe sobrecargar el método

Todo método puede ser **sobrecargado**, lo que significa que dos o más métodos se **identifican con el mismo nombre** y **se diferencian** por **la cantidad** o el **tipo de parámetros** que reciben.

Usando el operador **this** es posible invocar un constructor de la clase

Clases con Métodos

(Python)

Python no tiene clases abstractas por defecto, pero tiene un **módulo** para definir las clases de base abstracta. El nombre de **módulo** es **ABC** (**Abstract Base Classes**).

Un método se convierte en un método abstracto con la ayuda de una palabra clave decoradora llamada **@abstractmethod**.

Método toString

(Java)

Se hereda de `java.lang.Object`, lo que implica sobrescribir el método.

El método `toString` está disponible para todos los objetos de Java.

Se usa para mostrar los atributos de un objeto.

El método `devuelve` una cadena de texto al momento de instanciar el objeto (invocación indirecta).

Puede invocarse directamente (`nombreObjeto.toString()`)

Método toString

(Java)

```
public class Letrero {  
    String aviso;  
  
    public Letrero(String aviso) {  
        this.aviso = aviso;  
    }  
    public String getAviso() {  
        return aviso;  
    }  
    public void setAviso(String aviso) {  
        this.aviso = aviso;  
    }  
    @Override  
    public String toString() {  
        return "clase Letrero {" + "aviso=" + aviso + '}';  
    }  
}
```

Método toString

(Java)

```
public class progAviso {  
  
    public static void main(String args[]) {  
        Letrero letrero = new Letrero("Universidad Nacional de Colombia");  
        System.out.println(letrero+" .. Invocación Indirecta");  
        System.out.println(letrero.toString()+" .. Invocación Directa");  
    }  
}
```

Relaciones entre Clases

(Java)

Relaciones entre clases

(UML)

Las relaciones **establecen**, **indican** o **muestran** la forma en cómo se comunican las clases entre si.

El tipo de relación depende del **propósito** de la misma y de las características que se les atribuyan.





TIENE (Asociación)
Agregación (usa)
Composición (posee)

ES
Herencia

Relaciones entre clases

(UML)

Tipos:

- **Asociación:** 
 - Un objeto de una clase se puede comunicar con los atributos o métodos de otra clase
- **Inclusión:**
 - Un objeto es atributo de otra clase:
 - **Agregación:** un objeto es parte de otro sin que dependa de su existencia 
 - **Composición:** se compone de otros objetos y depende de su existencia 
- **Generalización o Especialización:**
 - Hace referencia a la herencia 

Relaciones entre clases

(UML)

