

Bienvenid@ a la colección de ejercicios de la clase ICB0003-S09-C01 Taking advantage of OOP concepts. Estos ejercicios son una oportunidad para que practiques los conceptos que has aprendido en la clase antes de moverte a la siguiente.

Antes de comenzar con los ejercicios puedes recrear los ejemplos dados en los videos y así coger más confianza.

En los ejercicios en los que tengas que diseñar el diagrama de clases, puedes usar [www.draw.io](http://www.draw.io), una aplicación gratuita que te permite diseñar todo tipo de diagramas online.

---

En los ejercicios en los que tengas que desarrollar código, te recomiendo que lo escribas en el entorno de desarrollo que hayas escogido (IntelliJ es el recomendado en esta asignatura). De algunos ejercicios tienes la solución y también hay mucha información en internet que puede ayudarte, pero no te recomiendo que copies y pegues código, sino que lo escribas y veas qué hace IntelliJ por ti. El objetivo es obtener experiencia escribiendo código desde 0. Es importante que entiendas qué estás haciendo en cada paso.

Estos ejercicios no son obligatorios ni evaluables, solamente son material extra para ayudarte en tu aprendizaje.

---

## Ejercicio 01: Gestión centro educativo

---

El objetivo de este ejercicio es aplicar herencia y comprobar sus beneficios: evitar repeticiones de código, facilidad a la hora de hacer modificaciones, facilitada a la hora de añadir funcionalidades.

Para resolver este ejercicio se recomienda haber consolidado los conocimientos expuestos en ICB0003-S09-C01-V01, ICB0003-S09-C01-V02.

En el proceso de diseño de una aplicación para un centro educativo se han identificado las siguientes clases.

class Estudiante	class Intercambio	class Profesor
String nombre String dirección String nif LocalDate fechaAlta LocalDate fechaNacimiento	String nombre String dirección String nie LocalDate fechaAlta LocalDate fechaNacimiento String paisOrigen	String nombre String dirección String nif LocalDate fechaAlta LocalDate fechaNacimiento String numSS String titulación String especialidad
class Administración		
String nombre String dirección String nif LocalDate fechaAlta LocalDate fechaNacimiento String numSS String posición		

Notas:

- Intercambio representa los estudiantes que provienen de un programa de intercambio con el extranjero

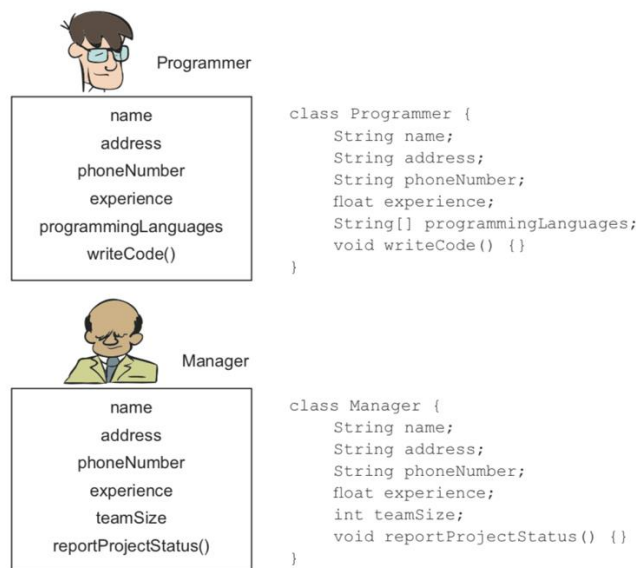
Propón la jerarquía de clases que minimice las duplicidades de código. Puedes añadir nuevas clases si lo consideras necesario.

## Ejercicio 02: Employee (versión 0.0)

El objetivo de este ejercicio es saber aplicar herencia y comprobar sus beneficios: evitar repeticiones de código, facilitar hacer modificaciones, facilitar añadir funcionalidades.

Para resolver este ejercicio se recomienda haber consolidado los conocimientos expuestos en ICB0003-S09-C01-V01, ICB0003-S09-C01-V02.

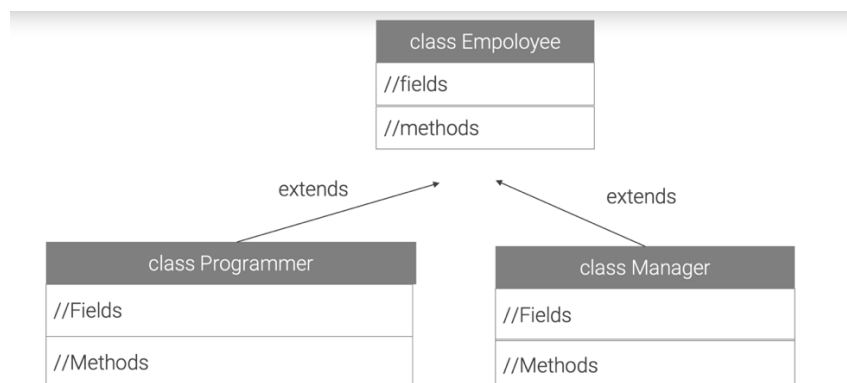
Imagina que tienes que desarrollar una aplicación para la gestión de emplead@s de una organización. Los puestos de trabajo de esa organización son dos: Programadores/Programadoras y Jefes/Jefas de proyecto. De cualquier emplead@ de esta organización se quiere registrar: nombre, dirección, teléfono, experiencia. De los programadores además se quiere registrar los lenguajes de programación que conoce y de los jefes de proyecto el tamaño del equipo de personas que dirige. La siguiente imagen muestra los atributos y métodos de estos puestos de trabajo, así como su representación en clases :



### Ejercicio 02 Tarea #1

¿Qué características son comunes en ambos puestos de trabajo?

Identifícalas y define una clase denominada Employee que las contenga. Completa el siguiente diagrama de clases



## Ejercicio 02 Tarea #2

---

Define un nuevo proyecto en IntelliJ donde:

- Crea la clase Employee con todos sus atributos
- Crea la clase Programmer que extienda de Employee y define sus atributos y métodos propios.
- Crea la clase Manager que extienda de Employee y define sus atributos y métodos propios.
- En la implementación de los métodos writeCode(), reportProjectStatus() muestra por consola "writing code" y "reporting Project status".
- En el método main crea 2 objetos de las 2 subclases. Comprueba que tienes acceso (de lectura y escritura) a los atributos name, address, phoneNumber, experience, así como a sus atributos y métodos propios.

Céntrate en el código más que en la lógica: No te preocupes, por ahora, de aplicar encapsulación, ni de crear métodos constructores propios.

## Ejercicio 02 Tarea #3

---

Ahora te dicen dos cambios:

- Que van a crear un nuevo departamento de recursos humanos en la organización y es necesario registrar este nuevo tipo de empleado en la organización. Este nuevo tipo no tiene características específicas, pero si debe de poder pagar nominas "managePayroll".
- Que de todos los empleados de la organización tienes que registrar su facebookID

¿Qué cambios serían necesarios en tu código?

---

## Ejercicio 03: Employee (versión 1.0)

---

El objetivo de este ejercicio es aplicar encapsulación en una aplicación que tenga relaciones de herencia, así como utilizar la palabra clave del lenguaje "super" y el concepto de clase abstracta.

Para resolver este ejercicio se recomienda haber consolidado los conocimientos expuestos en ICB0003-S09-C01-V01.. ICB0003-S09-C01-V06.

Continúa con la aplicación Employee para crear una nueva versión mejorada 1.0.

### Ejercicio 03 Tarea #1

---

Aplica encapsulación a todas las clases: protegiendo los atributos con los modificadores de acceso private/protected, creando constructores propios.

¿Puedes considerar la clase Employee como abstracta?

### Ejercicio 03 Tarea #2

---

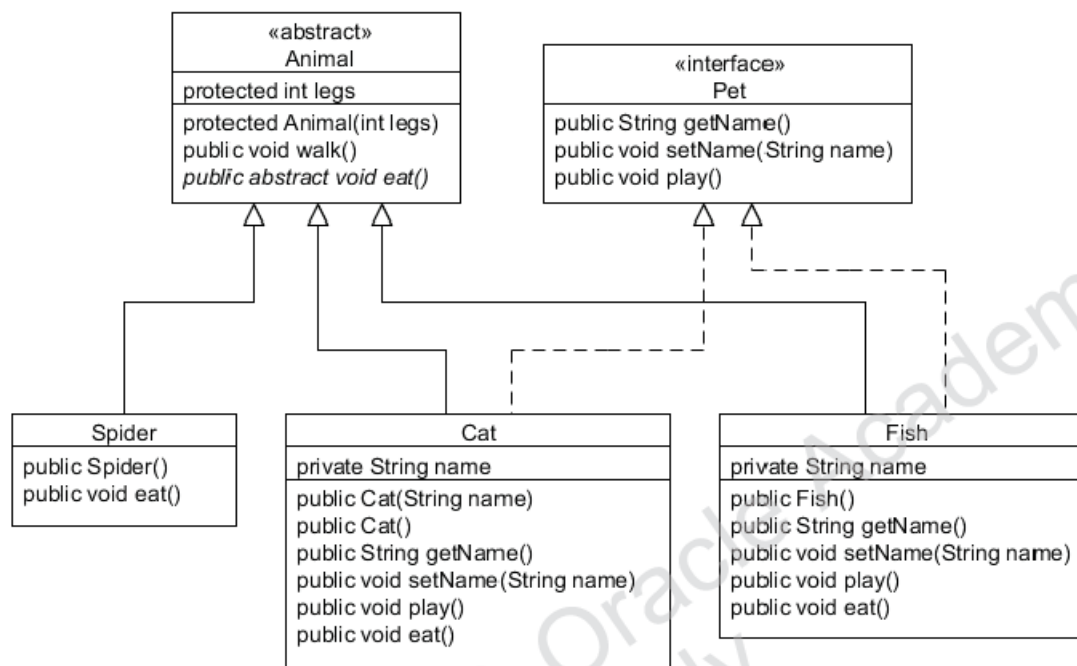
Permite que las clases Programmer, Manager, HR dispongan del método display() que retorne una String con los valores de sus atributos. Evita repeticiones de código y si es necesario crea en la clase base Employee un método base display() que te permita reutilizarlo en las subclases.

## Ejercicio 04: Pet

El objetivo de este ejercicio es aplicar todos los conceptos sobre la POO: herencia, clases abstractas, interfaces, y polimorfismo.

Para resolver este ejercicio se recomienda haber consolidado los conocimientos expuestos en ICB0003-S09-C01-V01.. ICB0003-S09-C01-V08.

En los materiales del curso, dispones del código fuente de la aplicación Java Pet. El objetivo de esta actividad es completar la aplicación de acuerdo al siguiente diagrama de clases.



Sigue los siguientes pasos:

- Abre el proyecto en tu IDE IntelliJ y tomate un tiempo para revisar su contenido. Al ejecutar la aplicación debería mostrarse texto en la consola de salida.
- Define la interface `Pet` en el package `com.company` de acuerdo a los requisitos especificados en el diagrama de clases
- Define la clase `Fish` en el package `com.company`, de acuerdo a los requisitos especificados en el diagrama de clases teniendo en cuenta:
  - Un `Fish` tiene 0 legs, por tanto en su método constructor se pasará un 0 al constructor de su superclase
  - La implementación del método `play()` consistirá en mostrar por consola "Just keep swimming."
  - La implementación del método `eat()` consistirá en mostrar por consola "Fish eat pond scum"
  - Sobre-escribirá el método `walk()` de la superclase `Animal`, llamando primero al método `walk` de la superclase y luego mostrará por consola "Fish, of course, can't walk, they swim"
- Define la clase `Cat` en el package `com.company`, de acuerdo a los requisitos especificados en el diagrama de clases teniendo en cuenta

- Un Cat tiene 4 legs, por tanto en su método constructor que recibe como argumento una string, se pasará un 4 al constructor de su superclase
  - En el método constructor que no tiene argumentos, se llamará al otro método constructor de la clase pasándole como parámetro "Fluffy"
  - La implementación del método play() consistirá en mostrar por consola el valor del atributo name seguido de " likes to play with string."
  - La implementación del método eat() consistirá en mostrar por consola "Cats like to eat spiders and fish"
- e. Añade en Main.java, el código necesario para testear las clases Cat y Fish que has codificado: usa cada método constructor y cada método (play(), eat(), walk()).

¿Qué ocurre si se testea un objeto de la clase Fish con una referencia a Animal? ¿Y si se testea un objeto de la clase Cat con una referencia a Pet?

<pre>Animal a = new Fish(); a.play(); a.eat(); a.walk();</pre>	<pre>Pet p = new Cat(); p.play(); p.eat(); p.walk();</pre>
--	--

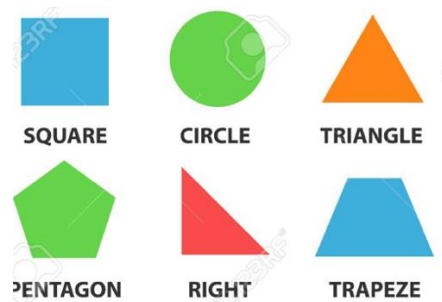
- f. En Main.java, implementa el método public void playWithAnimal(Animal a), de forma que:
- a. Si el objeto recibido como argumento implementa la interface Pet, llama al método play(). Comprueba que como el método play() está declarado en la interface Pet, antes de llamar al método deberás realizar el casting del argumento a Pet.
  - b. Si el objeto recibido como argumento NO implementa la interface Pet, muestra por consola "Danger! Wild animal."
- g. En Main.java, usa el método playWithAnimal(Animal a), utilizando como argumento objetos de las clases Spider, Cat, Fish

## Ejercicio 05: Figuras

El objetivo de este ejercicio es aplicar todos los conceptos sobre la POO: herencia, clases abstractas, interfaces, y polimorfismo.


Para resolver este ejercicio se recomienda haber consolidado los conocimientos expuestos en ICB0003-S09-C01-V01 ... ICB0003-S09-C01-V08.

Se quiere diseñar una aplicación que permita a niños dibujar de manera sencilla. Estos podrán seleccionar figuras geométricas (triángulo, rectángulo, círculo, etc) predefinidas y arrastrarlas a una zona de dibujo. Una vez en la zona de dibujo, a una figura se le puede cambiar de ubicación, de tamaño, de color de fondo, de color de línea y de grosor de línea.



### Ejercicio 05 Tarea #1

Diseña la jerarquía de clases para la aplicación, indicando algunos atributos y métodos dentro de las clases relacionados con el enunciado. Es una actividad abierta, donde el grado de complejidad depende de las capacidades que se quieran añadir a la aplicación. No es necesario que diseñes la aplicación con todas las figuras geométricas, basta con que diseñes al menos 3 figuras geométricas, por ejemplo Rectángulo, Círculo, Triángulo.

	<p><b>Hint:</b></p> <p>Ten en cuenta en tu diseño:</p> <ul style="list-style-type: none"><li>• La correcta encapsulación de la información. Para ello crea los constructores necesarios en cada clase y aplica los modificadores de acceso adecuados en cada caso: private, protected, public.</li><li>• Proteger los métodos que no quieras que sean sobre-escritos por las subclases con la palabra clave final.</li></ul>
---	--

### Ejercicio 05 Tarea #2

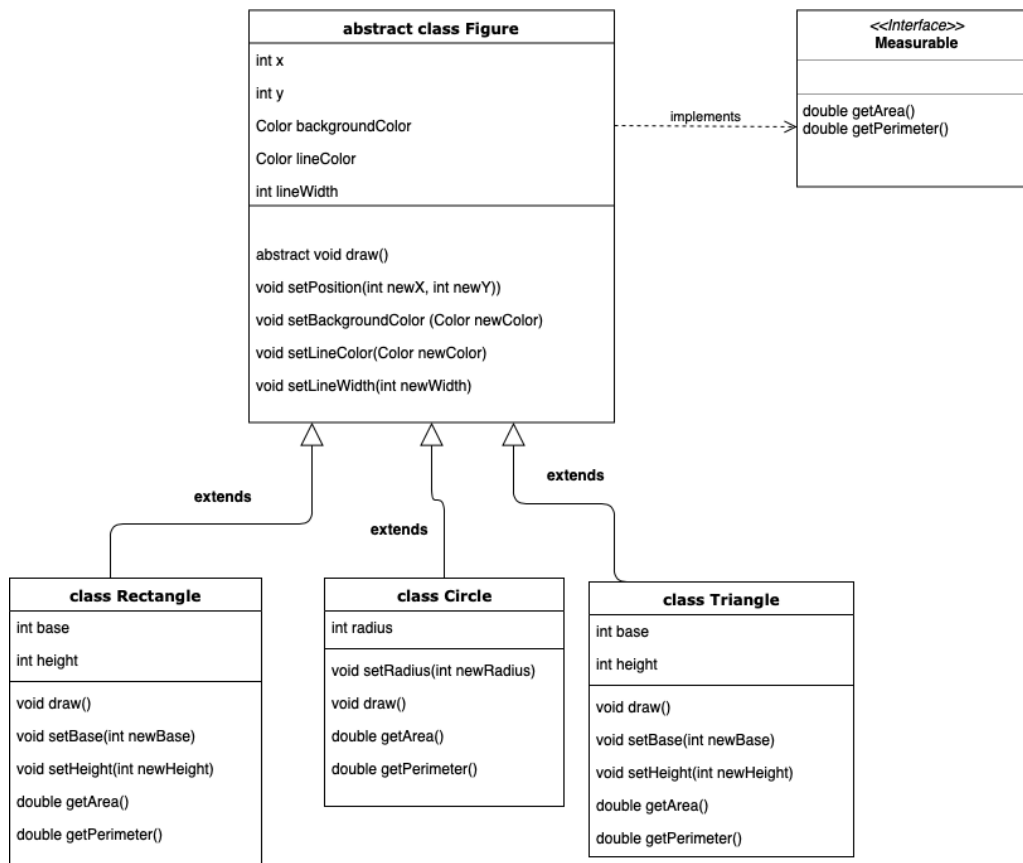
Todas las figuras geométricas tienen la capacidad de ser medidas, para calcular su área y su perímetro. Añade esta capacidad a tu jerarquía de clases.



**Ejercicio 05 Tarea #3** Correo electrónico o ID: Contraseña: He olvidado mi contraseña Iniciar sesión Inicio de sesión en el número de teléfono móvil

### Crear cuenta

Crea un Java application project en IntelliJ, donde desarrolles la jerarquía de clases diseñada en las tareas anteriores. Si no lo has diseñado puedes utilizar la siguiente jerarquía de referencia:



Ten en cuenta que:

- La implementación del método `draw()` en cada una de las clases consistirá en mostrar por consola "Drawing" seguido del nombre de la figura, su posición (x,y), el color de fondo, el color de línea, el grueso de la línea, así como sus dimensiones.
- La implementación del método `getArea()` en cada una de las clases, consistirá en calcular y retornar el valor del área de la figura. Por ejemplo en un rectángulo  $\text{área} = \text{base} \times \text{height}$
- La implementación del método `getPerimeter()` en cada una de las clases, consistirá en calcular y retornar el valor del perímetro de la figura. Por ejemplo en un rectángulo  $\text{perimeter} = 2 \times \text{base} + 2 \times \text{height}$

Añade en `Main.java`, el código necesario para testear las clases `Rectangle`, `Circle`, `Triangle` que has codificado: usa cada método constructor y cada método (`play()`, `eat()`, `walk()`).



## Ejercicio 05 Tarea #4

El objetivo de esta actividad es comprobar la necesidad de usar interfaces del lenguaje Java.

- En Main.java crea un array de objetos Figure.
- Imprime por consola el contenido del array de figuras mostrando el área de cada figura y su tipo (Rectangle, Circle, ...)
- Haz que la clase Figure implemente la interface Java Comparable, y aprovéchalo para ordenar el array de objetos figura. Criterio de ordenación por ejemplo será por área (de menor a mayor)
- Imprime de nuevo por consola el contenido del array de figuras mostrando el área de cada figura y su tipo (Rectangle, Circle, ...), para comprobar que se ha ordenado correctamente



### Hint:

Para ordenar el array de figuras, en lugar de implementar tu propio código, recuerda que Java dispone de la instrucción `Arrays.sort(array)`, que permite ordenar el array introducido como parámetro de acuerdo al criterio establecido para el tipo de objetos que contiene el array

`Figure[] array = ....`

`Arrays.sort(array);` //ordena el array de acuerdo al criterio establecido en la clase Figure al implementar la interface Comparable

A continuación, se muestra un ejemplo de salida por consola:

### FIGURES ARRAY

#### Before sorting:

```
Rectangle, area: 5000.0  
Circle, area: 15707.963267948966  
Triangle, area: 2500.0  
Triangle, area: 150.0  
Circle, area: 15707.963267948966
```

#### After sorting:

```
Triangle, area: 150.0  
Triangle, area: 2500.0  
Rectangle, area: 5000.0  
Circle, area: 15707.963267948966  
Circle, area: 15707.963267948966
```