

Laboratorio 9: Herencia (v2.0)

Programación 2

Ángel Herranz

aherranz@fi.upm.es

Universidad Politécnica de Madrid

2023-2024

Este laboratorio va a girar alrededor de uno de los conceptos que definen la orientación a objetos: la herencia (o subtipado).

Se pide: Tendrás que entregar 13 ficheros:

- `TestFiguras.java`: para poder asegurarte que todo lo que vas haciendo es correcto tendrás que escribir tus propias pruebas, y recuerda que **deberías empezar por aquí** aplicando el *desarrollo dirigido por las pruebas*.
- `IFigura.java`, `Figura.java`, `Circulo.java`, `Poligono.java`, `PoligonoRegular.java`, `Rectangulo.java`, `Isosceles.java`, `Cuadrado.java`, `Equilatero.java`, `Hexagono.java`, `Punto2D.java`: clases para representar figuras geométricas.
- `CentroDeMasas.java`: un programa principal para calcular el centro de masas de varias figuras.

Se espera en todo momento que escribas las pruebas antes de escribir las funcionalidades y que continuamente vayas ejecutando tus propias pruebas:

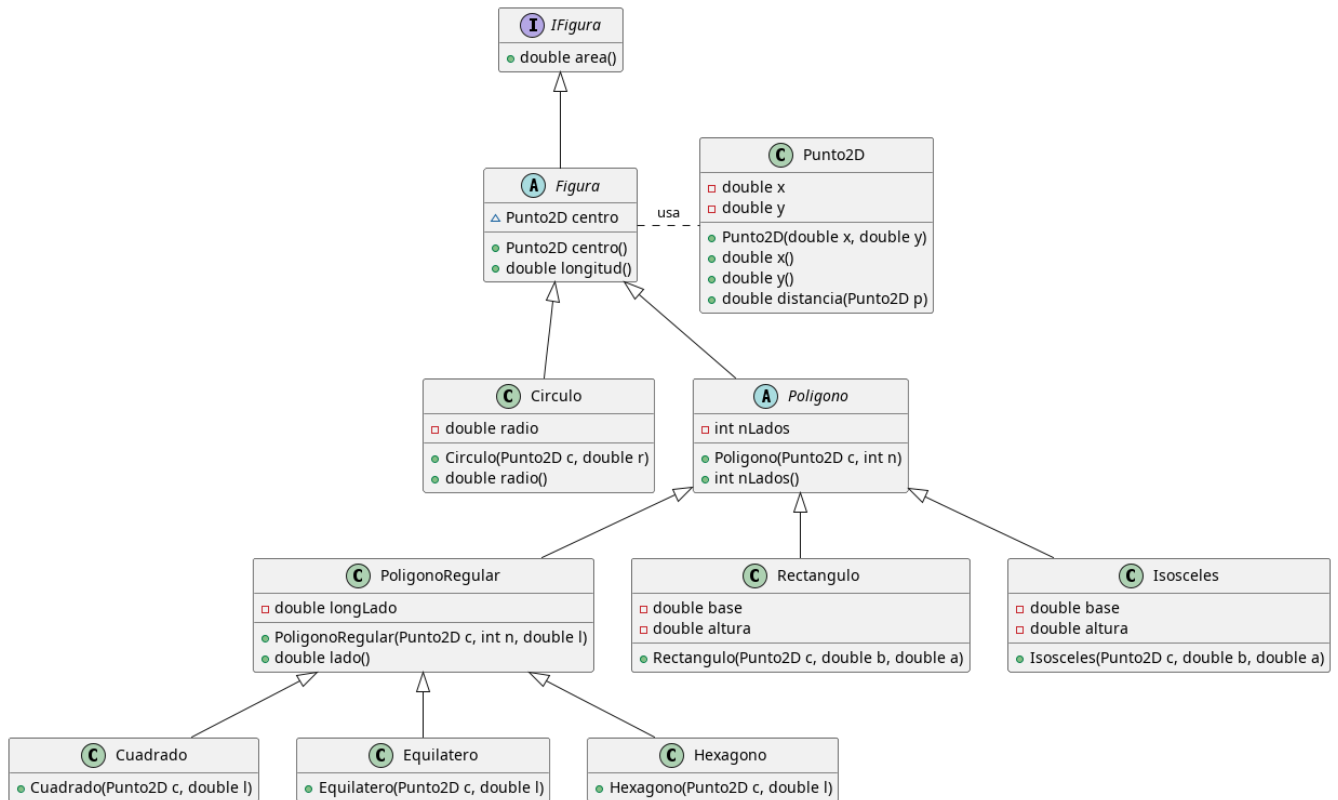
```
C:\Users\UPM\lab02> javac TestFiguras.java
```

```
C:\Users\UPM\lab02> java -ea TestFiguras
```

Nota: No olvides utilizar el *flag* `-ea` al ejecutar los tests.

Ejercicio 1. Ten a mano las transparencias de las sesiones 19 y 20.

Ejercicio 2. Observa el siguiente diagrama de clases¹, los tipos con una **I** son *interfaces*, con una **C** son clases y los que llevan una **A** son clases *abstractas*, ahora veremos lo que significa:




Los símbolos delante de las propiedades (atributos y métodos) indican:

- visibilidad **private**.
- △ visibilidad **protected**.
- visibilidad **public**.

interface Ejercicio 3. Empezaremos con el TAD (*Tipo Abstracto de Datos*) de las figuras. Ya sabemos que una buena forma de describir un TAD en Java es utilizando un *interface*. Lo llamaremos IFigura y tendrá los métodos que queremos que tengan todas las figuras geométricas, a saber:

- Método observador centro: devuelve un punto en dos dimensiones, tendrás que modelizar los puntos.
- Método observador longitud (equivalente al perímetro): devuelve un *double*.
- Método observador area: devuelve un *double*.

¹Puedes consultar el capítulo sobre diagrama de clases del libro *Distilled UML* de Fowler.

 **abstract** **Ejercicio 4.** La primera clase que vamos a implementar es la clase de *todas las figuras*: Figura. Las figuras, como se puede ver en el diagrama del ejercicio 2, pueden ser círculos, polígonos y dentro de los polígonos tenemos diferentes polígonos.

La clase Figura debe *implementar* el API de IFigura y como todas las figuras van a tener un centro podrás colocar un atributo centro en la clase Figura


Ahora toca implementar los métodos de IFigura. El método centro es trivial, pero... ¿cómo se puede implementar la longitud o el área de cualquier figura geométrica? Yo no sabría hacerlo. Pero si no lo hacemos no compila, Pruébalo.

Cuando en una clase no sabemos implementar un método podemos declararlo como *abstracto* y no implementarlo. Prueba:

```
public class Figura implements IFigura {  
    public abstract double area();  
    public abstract double longitud();  
}
```


Ahora te toca compilar y entender y resolver el error de compilación:

```
$ javac Figura.java  
Figura.java:1: error: Figura is not abstract and does not override abstract  
    method longitud() in Figura  
public class Figura implements IFigura {
```

 **Ejercicio 5.** Como has podido experimentar, las *clases abstractas* son aquellas que tienen métodos abstractos y los *métodos abstractos* son métodos que aún no se sabe cómo implementar.

Esta frase abre muchas preguntas. Te animo a que explores algunas:


- ¿Es posible crear instancias de una clase abstracta usando por ejemplo **new** Figura(p) (donde p es un punto)?
- Pero si los métodos area y longitud no tienen implementación... ¿para qué sirven?

 **Herencia** **Ejercicio 6.** Ahora te toca completar la implementación de todas las clases de la jerarquía de herencia del diagrama del ejercicio 2. Como verás todas las decisiones tienen sentido y podemos contar la historia de su modelización usando la herencia:

- Las figuras pueden ser círculos o polígonos.
- Todos los círculos **son** figuras y todos los polígonos **son** figuras.
- La longitud de un círculo es la longitud de su circunferencia.
- Los círculos tienen radio.
- Los polígonos tienen un número de lados.
- La longitud de un polígono es su perímetro pero como no sabemos calcularlo bien lo dejaremos como **abstracto** y por lo tanto la clase Poligono es **abstracta**.
- Tampoco sabemos calcular el área de cualquier polígono así que ese método sigue siendo **abstracto**.
- El resto de clases (PoligonoRegular y sus *hijos* Cuadrado, Equilatero y Hexágono, así como las clases Rectangulo y Isosceles) ya no son abstractas ya que con sus atributos podrás implementar todos los métodos.


- Observa cómo puedes implementar los métodos longitud y area en PoligonoRegular y puedes **especializarlo** en sus subclases.

De nuevo surgen dificultades y, esperamos, muchas preguntas. Quizás los siguientes ejercicios pueden resolverlas.

 **protected** **Ejercicio 7.** ¿Has visto qué pasa cuando desde una **clase hija** quieres acceder a una propiedad *privada* del **padre**?

Desde luego, si una propiedad es privada no queremos hacerla **public**. Sin embargo hay un **modificador intermedio** que se llama **protected** que permite que dicha propiedad sea visible en las subclases.

Hemos aplicado este conocimiento al atributo centro de Figura para que pueda ser accedido desde las subclases.


 **super()** **Ejercicio 8.** ¿Has conseguido establecer el número de lados (atributo nLados) en las clases hijas de Poligono sin modificar su visibilidad?


Es importante que no cambies ese modificador, es decir, el atributo nLados introducido en la clase abstracta Poligono tiene que ser privado. Pero entonces... ¿Cómo se puede establecer dicho atributo en las subclases?

Aquí es donde entra en juego la posibilidad de invocar el constructor del padre de forma explícita. Por ejemplo, el constructor Poligono puede ser invocado desde los constructores de sus subclases. Por ejemplo, en el constructor de Rectangulo se puede hacer esta llamada:

```
super(p, 5);
```

y lo que ocurre es que invoca el constructor de su superclase inmediata (es decir Poligono).

 La *super-clase* Object **Ejercicio 9.** Recuerda que todas las clases hereda de Object y por lo tanto es necesario **especializar** los métodos equals, toString y, quizás si te atreves **clone**.

 Centro de masas **Ejercicio 10.** Tienes que hacer un programa principal CentroDeMasas que lea de la entrada estándar múltiples figuras, una por línea. Por ejemplo²:

```
C 1.0 2.0 3.0
R 10.0 20.0 3.0 4.0
I -10.0 -20.0 3.0 4.0
P 20.0 20.0 5 1.0
Q 5.0 2.0 1.0
E 9.0 -1.0 10.0
H -10.0 10.0 3.0
```

El resultado de tu programa tiene que ser el centro de masas y deberás escribirlo en la salida estándar. En centro de masas lo puedes calcular usando el área para ponderar la masa alrededor del centro de cada figura.

Nota: la forma de realizar este programa es usando un método auxiliar leerFigura:

```
public static IFigura leerFigura() {
    ...
}
```

Además, probablemente necesitarás una colección de figuras, usa el TAD IList del paquete tads.jar.

²Cada letra identifica una figura, sólo el cuadrado es raro porque usa la letra Q, la letra P es para polígonos regulares. Los siguientes dos doubles son las coordenadas del centro. El resto es información específica de la figura.