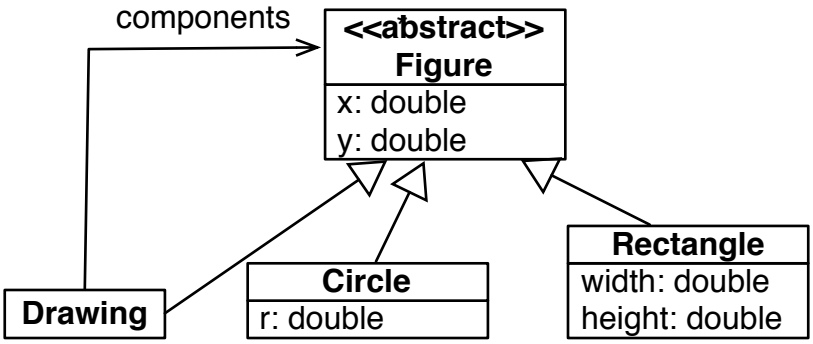
Dada la jerarquía de clases:



En la que todas las clases son **inmutables**:

- todas reciben en su constructor todos los atributos que necesitan (en el caso de `Drawing`, además de las coordenadas, se recibe una `List<Figure>` con las `components`)
- tienen getters de sus propiedades (en el caso de `Drawing`, para evitar que alguien use la lista para modificar las `components`, se devuelve una lista inmodificable de sus componentes, usando `Collections.unmodifiableList`)

Se pide:

1. Para el caso de la clase Drawing, a veces nos resultaría cómodo poder ir añadiendo figuras a un drawing poco a poco, cosa que es imposible ya que se trata de una clase inmutable.
   - ¿Qué patrón aplicaríais para conseguirlo?
   - Mostrad la implementación en Java. En este apartado no podéis modificar ninguna de las clases de la jerarquía de `Figure`.

2. Sobre las figuras querremos definir muchas operaciones (p.e. trasladar, escalar, calcular el área, el perímetro, etc., etc.) pero no querremos tener que modificar las clases cada dos por tres.
   Es decir, necesitaréis crear una infraestructura en la jerarquía de `Figure` para poder añadir operaciones sobre `Figure` sin necesidad de, posteriormente, modificarlas.
   - ¿Qué patrón aplicaríais para conseguirlo?
   - Este patrón puede aplicarse de varias formas en cuanto a dónde se realiza el recorrido de las partes de un compuesto. En este caso implementad la versión en la que el recorrido se realiza fuera de la clase `Drawing`.
   - Las implementaciones en todas las subclases de `Figure` parecen iguales, pero no podemos unificarlas en la clase `Figure`. ¿Por qué?

3. Implementad, usando la infraestructura que habéis creado en el punto anterior, una clase, `AreaDoubler`, que os permita doblar el área de una figura.
   - Obviamente el resultado será una nueva figura, ya que las figuras son inmutables.
   - En el caso de un `Drawing`, se doblarán las áreas de cada una de las figuras que contiene. Recordad, por coherencia con el punto anterior, que es aquí dónde haréis el recorrido de los componentes de un `Drawing`.
   - En el caso de un `Rectangle`, considerad la implementación que escala tanto la anchura como altura con razón `Math.sqrt(2.0)` y, en el caso de un `Circle`, lo que se escala con esa misma razón es el radio.