7.E. Interfaces.

1. Interfaces.

Has visto cómo la **herencia** permite definir **especializaciones** (o **extensiones**) de una **clase base** que ya existe sin tener que volver a repetir de todo el código de ésta. Este mecanismo da la oportunidad de que la nueva **clase especializada** (o **extendida**) disponga de toda la **interfaz** que tiene su clase base.

También has estudiado cómo los **métodos abstractos** permiten establecer una **interfaz** para marcar las **líneas generales de un comportamiento común de superclase** que deberían compartir de todas las **subclases**.

Si llevamos al límite esta idea de **interfaz**, podrías llegar a tener una **clase abstracta** donde todos sus métodos fueran abstractos. De este modo estarías dando únicamente el **marco de comportamiento**, sin ningún método implementado, de las posibles **subclases** que heredarán de esa **clase abstracta**. La idea de una **interfaz** (o **interface**) es precisamente ésa: **disponer de un mecanismo que permita especificar cuál debe ser el comportamiento que deben tener todos los objetos que formen parte de una determinada clasificación** (no necesariamente jerárquica).

Una **interfaz** consiste principalmente en una lista de declaraciones de métodos sin implementar, que caracterizan un determinado comportamiento. Si se desea que una clase tenga ese comportamiento, tendrá que implementar esos métodos establecidos en la **interfaz**. En este caso no se trata de una relación de **herencia** (la clase **A** es una especialización de la clase **B**, o la subclase **A** es del tipo de la superclase **B**), sino más bien una relación "de implementación de comportamientos" (la clase **A** implementa los métodos establecidos en la **interfaz B**, o los comportamientos indicados por **B** son llevados a cabo por **A**; pero no que **A** sea de clase **B**).

Imagina que estás diseñando una aplicación que trabaja con clases que representan distintos tipos de animales. Algunas de las acciones que quieres que lleven a cabo están relacionadas con el hecho de que algunos animales sean **depredadores** (por ejemplo: **observar** una **presa**, **perseguirla**, **comérsela**, etc.) o sean **presas** (**observar**, **huir**, **esconderse**, etc.). Si creas la clase **León**, esta clase podría implementar una interfaz **Depredador**, mientras que otras clases como **Gacela** implementarían las acciones de la interfaz **Presa**. Por otro lado, podrías tener también el caso de la clase **Rana**, que implementaría las acciones de la interfaz **Depredador** (pues es cazador de pequeños insectos), pero también la de **Presa** (pues puede ser cazado y necesita las acciones necesarias para protegerse).