A-
1-

La herencia es un mecanismo en la programación orientada a objetos (POO) que permite a una clase, llamada clase hija o clase derivada, heredar las propiedades y métodos de otra clase, llamada clase padre o clase base.
El principal objetivo de la herencia en POO es facilitar la creación de nuevas clases a partir de clases existentes, promoviendo la reutilización de código y la organización eficiente del software. Al heredar de una clase base, la clase hija puede utilizar y ampliar la funcionalidad de la clase padre, evitando la repetición de código y fomentando la modularidad y la extensibilidad del software.

2-

La encapsulación es otro principio fundamental en POO que consiste en agrupar los datos (propiedades) y los métodos que operan sobre esos datos dentro de una clase, ocultando la implementación interna y exponiendo solo una interfaz pública.
En el contexto de la herencia, la encapsulación juega un papel crucial al controlar la visibilidad de las propiedades y métodos de la clase padre para la clase hija. Esto permite a la clase padre controlar cómo se accede a sus datos y métodos desde la clase hija, asegurando la integridad y la seguridad del código.

3-

-Public: Las propiedades y métodos declarados como public son accesibles desde cualquier parte del código, incluyendo las clases hijas.

-Private: Las propiedades y métodos declarados como private solo son accesibles dentro de la misma clase. Las clases hijas no pueden acceder a ellos directamente.

-Protected: Las propiedades y métodos declarados como protected son accesibles dentro de la misma clase y en las clases hijas, pero no desde otras clases externas.

4-
La especialización es un proceso en la herencia que permite crear nuevas clases que heredan las propiedades y métodos de una clase base y añaden características específicas. Esta especialización se logra mediante la sobreescritura de métodos, donde la clase hija redefine un método heredado de la clase padre para que tenga un comportamiento diferente.

5-

-Mayor flexibilidad: Permite crear clases con funcionalidades combinadas de diferentes clases base.

-Reutilización de código: Permite reutilizar código de múltiples clases base.

Desventajas:

-Complejidad: La herencia múltiple puede dificultar la comprensión y el mantenimiento del código.

-Conflictos de herencia: Si dos clases padre tienen un método con el mismo nombre, puede haber ambigüedad sobre cuál se utiliza en la clase hija.

-Problemas de diamante: La herencia múltiple puede llevar a problemas de diamante, donde una clase hija hereda de dos clases padre que, a su vez, heredan de una misma clase base. Esto puede crear ambigüedades y problemas de orden de resolución de métodos.