**PROGRAMACIÓN 2: TRABAJO PRÁCTICO 2**

1. Es un mecanismo que permite que una clase (clase padre) le pase (herede) sus atributos y métodos a otra clase (clase hija). Esto permite la reutilización del código, generar poliformisfo y facilita la organización del código.
2. Se representa como una línea sólida con una flecha dirigida desde la clase hija a la clase padre.
3. La herencia puede usarse cuando hay una relación de pertenencia entre clases, es decir, que una clase (la clase hija) pertenece al conjunto que representa otra clase (clase padre) y se desea mantener los atributos y métodos de la clase padre pero extender su funcionalidad sin mofidicarla, o también es útil cuando se desea modelar una jerarquía de clases.
4. Es un conjunto de métodos que una clase en específico debe implementar.
5. Se representa con un rectángulo con el nombre de la interface en la parte superior precedido con el símbolo <<interface>>, y los métodos que contiene en la parte inferior.
6. Es recomendable cuando se desea definir un conjunto de métodos que deben implementarse en varias clases de forma independiente.
7. Las interfaces definen contratos para que las clases las implementen y permiten la implementación de múltiples herencias, mientras que las clases abstractas proporcionan una estructura base y funcionalidad compartida para las subclases y no permiten múltiples herencias.

* **Asociación:** Indica que un objeto de una clase está relacionado con uno o más objetos de otra clase.
* **Agregación:** Indica que un objeto de una clase contiene o es parte de un objeto de otra clase.
* **Composición:** Indica que un objeto de una clase es responsable de crear y destruir objetos de otra clase.
* **Herencia:** Indica que una clase hereda los atributos y métodos de otra clase.
* **Implementación:** Indica que una clase implementa una interface.
* **Asociación:** Se representa con una línea sólida.
* **Agregación:** Se representa con una línea sólida con un rombo vacío en un extremo, que va desde la clase que contiene la colección o grupo con la clase que representa los elementos individuales.
* **Composición:** Se representa con una línea sólida con un rombo lleno en un extremo, que va desde el componente secundario al principal.
* **Herencia:** Se representa con una línea sólida con una flecha triangular vacía, que va desde la clase hija hasta la clase padre.
* **Implementación:** Se representa con una línea punteada con una flecha triangular llena, que va desde la interface hasta la clase que la implementa.
* **Asociación:** Se usa cuando dos clases están relacionadas de forma general y sin dependencia fuerte entre ellas.
* **Agregación:** Se usa cuando una clase contiene o es parte de otra clase, pero la relación es menos estricta que la composición, cuando un objeto puede existir independientemente de otro objeto.
* **Composición:** Se usa cuando una clase contiene o es parte de otra clase y tiene una relación de dependencia fuerte, cuando un objeto no puede existir o tiene una relación de vida útil con otro objeto.
* **Herencia:** Se usa cuando una clase es una extensión o especialización de otra clase.
* **Implementación:** Se usa cuando una clase implemente una interface.