|  |  |
| --- | --- |
| Polimorfismo | Un concepto en la programación orientada por objetos que les permite a las clases tener varias formas y comportarse como sus superclases. |
| Sobrescribir métodos | Implementar métodos en una subclase que tiene el mismo prototipo (los mismos parámetros, nombre de método, y tipo de retorno) que otro método en la superclase. |
| final | Una palabra clave en Java usada para limitar las subclases a partir de la extensión de una clase, sobrescribiendo métodos o cambiando datos. |
| inmutable | Una propiedad de una case estática que hace que la clase no pueda ser extendida o que sus datos sean cambiados. |
| Sobrecarga de métodos | Implementar un método con el mismo nombre que otro método en la misma clase que tiene diferentes parámetros o un tipo de retorno diferente. |
| Enlace Dinámico | El proceso por el cual Java es capaz de determinar qué método invocar cuando los métodos han sido sobrescritos |
| Abstract | Una palabra clave en Java que permite que las clases sean extendidas, pero las clases no pueden ser instanciadas (construidas)y cuando se aplican los métodos, se indica que los métodos deberían ser implementados en todas las subclases de la clase. |

1. *(eclipse clase: PuntoUno)* Este código ejecuta lo siguiente:

Inside Class B's Method!

Inside Class C's Method!

Inside Class A's Method!

2. Algunas de las diferencias entre la clase abstracta e interfaz son, una clase abstracta puede tener métodos abstractos y concretos, mientras que una interfaz solo puede tener métodos abstractos. Además, una clase puede heredar de una sola clase abstracta pero implementar múltiples interfaces. Las clases abstractas pueden tener constructores, mientras que las interfaces no. Una clase abstracta puede ser extendida por una clase concreta, no abstracta u otra clase abstracta mientras que una interfaz solo puede ser implementada por una clase.

Las clases abstractas se usan cuando se necesita proporcionar una base común para varias clases relacionadas con algunos métodos abstractos y otros métodos concretos, y las interfaces se usan cuando se necesita definir un conjunto de métodos que debe implementar cualquier clase que se ajuste a la interfaz. En algunos casos, puede ser apropiado o eficiente usar ambas estructuras juntas para proporcionar jerarquías de clases complejas con especificaciones de métodos comunes.

3.

|  |  |
| --- | --- |
| **Código** | **¿Siempre compila, a veces compila, no compila?** |
| a = new B(); | Siempre compilará porque B es un tipo de A y se puede crear una instancia de B y asignarla a una variable de tipo A. |
| d = new C(); | No compilará porque no se puede crear una instancia de C o D, ya que C es abstracta y D hereda de ella. |
| b.methodA(); | Siempre compilará porque B implementa A, lo que garantiza una implementación para methodA(). |
| e.methodA(); | Siempre compilará porque E hereda de B, que implementa A, lo que significa que E también implementa methodA(). |
| c = new C(); | Siempre compilará porque se puede crear una nueva instancia de C sin problemas. |
| (D)c.methodC(); | No compila porque C es abstracta y no se puede crear una instancia directamente. |

4. *eclipse clase: Factorial y Main. Se ejecuta desde Main*